

Bruno Nicenboim, Daniel Schad, and Shravan Vasishth

An Introduction to Bayesian Data Analysis for Cognitive Science

This book is dedicated to the cognitive science community.

Contents

Preface	13
About the Authors	21
I Foundational ideas	1
1 Introduction	3
1.1 Probability	3
1.2 Conditional probability	6
1.3 The law of total probability	7
1.4 Discrete random variables: An example using the Binomial distribution	9
1.4.1 The mean and variance of the Binomial distribution	11
1.4.2 What information does a probability distribution provide?	14
1.5 Continuous random variables: An example using the Normal distribution	20
1.5.1 An important distinction: probability vs. density in a continuous random variable	23
1.5.2 Truncating a normal distribution	24
1.6 Bivariate and multivariate distributions	26
1.6.1 Example 1: Discrete bivariate distributions	26
1.6.2 Example 2: Continuous bivariate distributions	31
1.6.3 Generate simulated bivariate (multivariate) data	33
1.7 An important concept: The marginal likelihood (integrating out a parameter)	37
1.8 Summary of useful R functions relating to distributions	40
1.9 Summary	40
1.10 Further reading	41

1.11	Exercises	41
2	Introduction to Bayesian data analysis	45
2.1	Bayes' rule	45
2.2	Deriving the posterior using Bayes' rule: An analytical example	46
2.2.1	Choosing a likelihood	47
2.2.2	Choosing a prior for θ	48
2.2.3	Using Bayes' rule to compute the posterior $p(\theta n, k)$	51
2.2.4	Summary of the procedure	53
2.2.5	Visualizing the prior, likelihood, and the posterior	54
2.2.6	The posterior distribution is a compromise between the prior and the likelihood	55
2.2.7	Incremental knowledge gain using prior knowledge	57
2.3	Summary	58
2.4	Further reading	59
2.5	Exercises	59
II	Regression models with brms	63
3	Computational Bayesian data analysis	65
3.1	Deriving the posterior through sampling	65
3.1.1	Bayesian Regression Models using 'Stan': brms .	66
3.2	Prior predictive distribution	77
3.3	The influence of priors: sensitivity analysis	84
3.3.1	Flat, uninformative priors	84
3.3.2	Regularizing priors	85
3.3.3	Principled priors	85
3.3.4	Informative priors	85
3.4	Revisiting the button-pressing example with different priors	86
3.5	Posterior predictive distribution	91
3.5.1	Comparing different likelihoods	94
3.5.2	The log-normal likelihood	95

3.5.3	Re-fitting a single subject pressing a button repeatedly with a log-normal likelihood	96
3.6	Summary	106
3.7	Further reading	107
3.8	Exercises	107
4	Bayesian regression models	109
4.1	A first linear regression: Does attentional load affect pupil size?	109
4.1.1	Likelihood and priors	110
4.1.2	The <code>brms</code> model	116
4.1.3	How to communicate the results?	122
4.1.4	Descriptive adequacy	123
4.2	Log-normal model: Does trial affect response times?	124
4.2.1	Likelihood and priors for the log-normal model	127
4.2.2	The <code>brms</code> model	135
4.2.3	How to communicate the results?	136
4.3	Logistic regression: Does set size affect free recall?	139
4.3.1	The likelihood for the logistic regression model	141
4.3.2	Priors for the logistic regression	144
4.3.3	The <code>brms</code> model	149
4.3.4	How to communicate the results?	151
4.3.5	Descriptive adequacy	153
4.4	Summary	155
4.5	Further reading	156
4.6	Exercises	156
5	Bayesian hierarchical models	159
5.1	A hierarchical model with a normal likelihood: The N400 effect	163
5.1.1	Complete pooling model (M_{cp})	166
5.1.2	No pooling model (M_{np})	169
5.1.3	Varying intercepts and varying slopes model (M_v)	174
5.1.4	Correlated varying intercept varying slopes model (M_h)	184
5.1.5	By-subjects and by-items correlated varying intercept varying slopes model (M_{sih})	189

5.1.6	Beyond the maximal model–Distributional regression models	198
5.2	A hierarchical log-normal model: The Stroop effect	207
5.2.1	A correlated varying intercept varying slopes log-normal model	208
5.3	Why fitting a Bayesian hierarchical model is worth the effort	215
5.4	Summary	217
5.5	Further reading	217
5.6	Exercises	217
6	The Art and Science of Prior Elicitation	227
6.1	Eliciting priors from oneself for a self-paced reading study: A simple example	229
6.2	Eliciting priors from experts	246
6.3	Deriving priors from meta-analyses	248
6.4	Using previous experiments' posteriors as priors for a new study	254
6.5	Summary	255
6.6	Further reading	255
7	Workflow	257
7.1	Model building	258
7.2	Principled questions on a model	261
7.2.1	Prior predictive checks: Checking consistency with domain expertise	261
7.2.2	Computational faithfulness: Testing for correct posterior approximations	267
7.2.3	Model sensitivity	268
7.2.4	Posterior predictive checks: Does the model adequately capture the data?	269
7.3	Exemplary data analysis	271
7.3.1	Prior predictive checks	272
7.3.2	Adjusting priors	281
7.3.3	Computational faithfulness and model sensitivity	284
7.3.4	Posterior predictive checks: Model adequacy	286
7.4	Summary	289

7.5	Further reading	290
8	Contrast coding	291
8.1	Basic concepts illustrated using a two-level factor	292
8.1.1	Default contrast coding: Treatment contrasts	295
8.1.2	Defining comparisons	296
8.1.3	Sum contrasts	298
8.1.4	Cell means parameterization and posterior comparisons	301
8.2	The hypothesis matrix illustrated with a three-level factor	303
8.2.1	Sum contrasts	305
8.2.2	The hypothesis matrix	306
8.2.3	Generating contrasts: The <code>hypr</code> package	310
8.3	Other types of contrasts: illustration with a factor with four levels	313
8.3.1	Repeated contrasts	315
8.3.2	Helmert contrasts	318
8.3.3	Contrasts in linear regression analysis: The design or model matrix	323
8.3.4	Polynomial contrasts	326
8.3.5	An alternative to contrasts: monotonic effects	327
8.4	What makes a good set of contrasts?	331
8.4.1	Centered contrasts	332
8.4.2	Orthogonal contrasts	333
8.4.3	The role of the intercept in non-centered contrasts	335
8.5	Computing condition means from estimated contrasts	338
8.6	Summary	347
8.7	Further reading	348
8.8	Exercises	348
9	Contrast coding for designs with two predictor variables	353
9.1	Contrast coding in a factorial 2×2 design	353
9.1.1	Nested effects	360
9.1.2	Interactions between contrasts	366
9.2	One factor and one covariate	372
9.2.1	Estimating a group difference and controlling for a covariate	372

9.2.2	Estimating differences in slopes	378
9.3	Interactions in generalized linear models (with non-linear link functions) and non-linear models	383
9.4	Summary	390
9.5	Further readings	391
9.6	Exercises	391
III	Advanced models with Stan	395
10	Introduction to the probabilistic programming language Stan	397
10.1	Stan syntax	402
10.2	A first simple example with Stan: Normal likelihood	404
10.3	Another simple example: Cloze probability with Stan with the Binomial likelihood	413
10.4	Regression models in Stan	415
10.4.1	A first linear regression in Stan: Does attentional load affect pupil size?	415
10.4.2	Interactions in Stan: Does attentional load interact with trial number affecting pupil size?	425
10.4.3	Logistic regression in Stan: Does set size and trial affect free recall?	429
10.5	Summary	435
10.6	Further reading	435
10.7	Exercises	436
11	Complex models and reparametrization	441
11.1	Hierarchical models with Stan	441
11.1.1	Varying intercept model with Stan	441
11.1.2	Uncorrelated varying intercept and slopes model with Stan	446
11.1.3	Correlated varying intercept varying slopes model	457
11.1.4	By-subject and by-items correlated varying intercept varying slopes model	471
11.2	Summary	475
11.3	Further reading	475
11.4	Exercises	476

o.o	<i>Contents</i>	9
12	Custom distributions in Stan	479
12.1	A change of variables with reciprocal normal distribution	479
12.1.1	Simulation based calibration	489
12.2	A custom distribution: Re-implementing the exponential distribution manually	491
12.3	Further reading	497
IV	Other useful models	499
13	Meta-analysis and measurement error models	501
13.1	Meta-analysis	502
13.1.1	A meta-analysis of similarity-based interference in sentence comprehension	502
13.2	Measurement-error models	516
13.2.1	Accounting for measurement error in a voice onset time model	516
13.3	Summary	525
13.4	Further reading	525
13.5	Exercises	525
14	SAT	529
V	Model comparison and hypothesis testing	531
15	Introduction to model comparison	533
15.1	Further reading	536
16	Bayes factors	537
16.1	Hypothesis testing using the Bayes factor	538
16.1.1	Marginal likelihood	538
16.1.2	Bayes factor	545
16.2	Examining the N400 effect with Bayes factor	550
16.2.1	Sensitivity analysis	558
16.2.2	Non-nested models	560
16.3	The influence of the priors on Bayes factors: beyond the effect of interest	563
16.4	Bayes factor in Stan	567
16.5	Bayes factors in theory and in practice	570

16.5.1	Bayes factors in theory: Stability and accuracy	570
16.5.2	Bayes factors in practice: Variability with the data	576
16.6	Summary	585
16.7	Further reading	586
16.8	Exercises	586
17	Cross-validation	589
17.1	Expected log predictive density of a model	589
17.2	K-fold and leave-one-out cross-validation	593
17.3	Testing the N400 effect using cross-validation	597
17.3.1	Cross-validation with PSIS-LOO	598
17.3.2	Cross-validation with K-fold	601
17.3.3	Leave-one-group-out cross-validation	603
17.4	Comparing different likelihoods with cross-validation	604
17.5	Issues with cross-validation	608
17.6	Cross-validation in Stan	612
17.6.1	PSIS-LOO-CV in Stan	612
17.7	Summary	621
17.8	Further reading	621
17.9	Exercises	622
VI	Computational cognitive modeling with Stan	623
18	Introduction to computational cognitive modeling	625
18.1	Further reading	625
19	Multinomial processing trees	627
19.1	Modeling multiple categorical responses	627
19.1.1	A model for multiple responses using the multinomial likelihood	630
19.1.2	A model for multiple responses using the categorical distribution	635
19.2	Modeling picture naming abilities in aphasia with MPT models	636
19.2.1	Calculation of the probabilities in the MPT branches	639
19.2.2	A simple MPT model	639

0.0 <i>Contents</i>	11
19.2.3 An MPT assuming by-item variability	644
19.2.4 A hierarchical MPT	649
19.3 Further reading	660
19.4 Exercises	660
20 Mixture models	667
20.1 A mixture model of the speed-accuracy trade-off: The fast-guess model account	669
20.1.1 The global motion detection task	670
20.1.2 A very simple implementation of the fast-guess model	672
20.1.3 A multivariate implementation of the fast-guess model	683
20.1.4 An implementation of the fast-guess model that takes instructions into account	689
20.1.5 A hierarchical implementation of the fast-guess model	700
20.2 Summary	718
20.3 Further reading	719
20.4 Exercises	719
21 A simple accumulator model to account for choice response time	721
21.1 Modeling a lexical decision task	722
21.1.1 Modeling the lexical decision task with the log-normal race model	730
21.1.2 A generative model for a race between accumulators	732
21.1.3 Fitting the log-normal race model	736
21.1.4 A hierarchical implementation of the log-normal race model	747
21.1.5 Dealing with contaminant responses	756
21.2 Further reading	769
References	771
Index	803



Preface

This book is intended to be 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 time measurements (e.g., self-paced reading, eye-tracking-while-reading, voice onset time), event-related potentials, pupil sizes, accuracies (e.g., recall tasks, yes-no questions), categorical answers (e.g., picture naming), choice-reaction time (e.g., Stroop task, motion detection task), etc. 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

and interpreting results of data analyses in a more nuanced manner, and lead to more measured claims in the published literature.

This book is still a work in progress, and it's still incomplete. Please report typos, errors, or suggestions at <https://github.com/vasishth/bayescogsci/issues>

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 and 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) to reproduce the examples presented. There are many good online resources on R that the reader can consult. Examples are: R for data science (<https://r4ds.had.co.nz/>), and Efficient R programming (<https://csgillespie.github.io/efficientR/>).

We also assume that the reader is familiar with basic linear modeling, and linear mixed models (Bates, Mächler, et al. 2015a; Baayen, Davidson, and Bates 2008). In order to understand the Cognitive Science literature, it is critical to have a thorough understanding of frequentist statistics. We will cover frequentist statistics in another book (Vasishth, Schad, et al. 2021b); here, we focus only on Bayesian methodology, building on the linear modeling framework.

Developing the right mindset for this book

One very important characteristic that the reader should bring to this book is a can-do spirit. There will be many places where the going will get tough, and the reader will have to play around with the material, or refresh their understanding of arithmetic or middle-school algebra. The basic principles of such a can-do spirit are nicely summarized in the book by Burger and Starbird (2012). Although we cannot summarize the insights in that book in a few words, inspired by these authors' work, we provide a short enumeration of the kind of mindset we want the reader to cultivate:

- Spend time on the basic, apparently easy material; make sure you understand it deeply. Look for gaps in your understanding. Reading different presentations of the same material (in different books or articles) can yield new insights.
- Let mistakes and errors be your teacher. We instinctively recoil from our mistakes, but errors are ultimately our friends; they have the potential to teach us more than our correct answers can. In this sense, a correct solution is less interesting than an incorrect one.
- When you are intimidated by some exercise or problem, give up and admit defeat immediately. This relaxes the mind; you've already given up, there's nothing more to do. Then, after a while, try to solve a simpler version of the problem. Sometimes, it is useful to break the problem down to smaller parts, each of which may be easier to solve.
- Create your own questions. Don't wait to be asked questions; develop your own problems and then try to solve them.
- Don't expect to understand everything in the first pass. Just mentally note the gaps in your understanding, and return to them later and work on these gaps.
- Step back periodically to try to sketch out a broader picture of what you are learning. Writing down what you know, without looking up anything, is one helpful way to achieve this. Don't wait for the teacher to give you bullet-point summaries of what you should have learned; develop such summaries yourself.
- Develop the art of finding information. When confronted with some-

thing you don't know, or with some obscure error message, use google to find some answers.

As instructors, we have noticed over the years that students with such a mindset generally do very well. Some students already have that spirit, but others need to explicitly develop it. We firmly believe that everyone can develop such a mindset; but one may have to work on acquiring it.

In any case, such an attitude is absolutely necessary for a book of this sort.

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 boxes. These boxes provide a more formal development (useful to transition to more advanced textbooks like Gelman et al. 2014), or deal with tangential aspects of the topics presented in the chapter.

Here are some suggested paths through this book, depending on the reader's goals:

- For a short course for complete beginners, read chapters 1 to 5. We usually cover these five chapters in a five-day summer school course that we teach annually.
 - For a course that focuses on regression models with the R package `brms`, read chapters 1 to 9, and optionally 13 and 14.
 - For an advanced course that focuses on complex models with Stan, read chapters 10 to 21.
-

Online materials

The entire book, including all data and source code, is available online for free on <https://vasishth.github.io/bayescogsci/book>. The solutions to exercises are provided upon request.

Software needed

Before you start, please install

- R (<https://cran.r-project.org/>) and RStudio (<https://www.rstudio.com/>), or any other Integrated Development Environment that you prefer, such as Visual Studio Code (<https://code.visualstudio.com/>).
- The R package `rstan` (please pay close attention to the installation instructions!):
 - Instructions for Windows (<https://github.com/stan-dev/rstan/wiki/Installing-RStan-on-Windows>)
 - Instructions for Mac or Linux (<https://github.com/stan-dev/rstan/wiki/Installing-RStan-on-Mac-or-Linux>)
- The R packages MASS, dplyr, tidyr, purrr, extraDistr, ggplot2, loo, bridgesampling, brms, bayesplot, tictoc, hypr, bcogsci, can be installed the usual way: `install.packages(c("MASS", "dplyr", "tidyr", "purrr", "extraDistr", "ggplot2", "loo", "bridgesampling", "brms", "bayesplot", "tictoc", "hypr", "bcogsci"))`
- The data and Stan models used in this book can be installed using `remotes::install_github("bnicenboim/bcogsci")`

In every R session, load these packages. These load commands could be placed in one's `.Rprofile` file.

```
library(MASS)
## be careful to load dplyr after MASS
library(dplyr)
library(tidyr)
library(purrr)
library(extraDistr)
library(ggplot2)
library(loo)
library(bridgesampling)
library(brms)
library(rstan)
library(bayesplot)
```

```
library(tictoc)
library(hypr)
library(bcogsci)
## Save compiled models:
rstan_options(auto_write = FALSE)
## Parallelize the chains using all the cores:
options(mc.cores = parallel::detectCores())
# To solve some conflicts between packages
select <- dplyr::select
extract <- rstan::extract
```

Acknowledgments

We are grateful to the many generations of students at the University of Potsdam, various summer schools at ESSLII, 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) held annually at Potsdam, Germany. 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. We would also like to thank Christian Robert (otherwise known as Xi'an), Robin Ryder, Nicolas Chopin, Michael Betancourt, Andrew Gelman, and the Stan developers (especially Bob Carpenter and Paul-Christian Bürkner) for their advice; and Athanassios Protopapas, and Masataka Ogawa for catching typos and other errors in the book. Thanks also go to Jeremy Oakley and other statisticians at the School of Mathematics and Statistics, University of Sheffield, UK, for helpful discussions.

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 September 2019; this stay at ZiF was part of the ac-

tivities 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 write without the following software: R (Version 4.1.1; R Core Team 2019) and the R-packages *afex* (Singmann et al. 2020), *barsurf* (Version 0.7.0; Spurgle 2020a), *bayesplot* (Version 1.8.1; Gabry and Mahr 2019), *bcogsci* (Version 0.0.0.9000; Nicenboim, Schad, and Vasishth 2020), *bibtex* (Version 0.4.2.3; Francois 2017), *bivariate* (Version 0.7.0; Spurgle 2020b), *bookdown* (Version 0.22; Xie 2019a), *bridgesampling* (Version 1.1.2; Gronau, Singmann, and Wagenmakers 2020), *brms* (Version 2.16.1; Bürkner 2019), *citr* (Aust 2019), *cmdstanr* (Version 0.4.0.9000; Gabry and Češnovar 2021), *cowplot* (Version 1.1.1; Wilke 2020), *dplyr* (Version 1.0.7; Wickham, François, et al. 2019), *DT* (Version 0.19; Xie, Cheng, and Tan 2019), *extraDistr* (Version 1.9.1; Wolodzko 2019), *forcats* (Version 0.5.1; Wickham 2019a), *gdtools* (Version 0.2.3; Gohel et al. 2019), *ggplot2* (Version 3.3.5; Wickham, Chang, et al. 2019), *gridExtra* (Version 2.3; Auguie 2017), *htmlwidgets* (Version 1.5.4; Vaidyanathan et al. 2018), *hypr* (Version 0.2.2; Schad et al. 2019; Maximilian M. Rabe et al. 2020), *intoo* (Version 0.4.0; Spurgle and Bode 2020), *kableExtra* (Version 1.3.4; Zhu 2019), *knitr* (Version 1.36; Xie 2019b), *lingpsych* (Version 0.0.0.9000; Vasishth, Schad, et al. 2021a), *lme4* (Version 1.1.27.1; Bates, Mächler, et al. 2015b), *loo* (Version 2.4.1; Vehtari, Gelman, and Gabry 2017a; Yao et al. 2017), *MASS* (Version 7.3.54; Ripley 2019), *Matrix* (Version 1.3.4; Bates and Maechler 2019), *miniUI* (Version 0.1.1.1; Cheng 2018), *papaja* (Version 0.1.0.9997; Aust and Barth 2020), *pdfTools* (Version 3.0.1; Ooms 2021), *purrr* (Version 0.3.4; Henry and Wickham 2019), *Rcpp* (Version 1.0.7; Eddelbuettel et al. 2019), *readr* (Version 2.0.2; Wickham, Hester, and Francois 2018), *RefManageR* (Version 1.3.0; McLean 2017), *remotes* (Version 2.4.1; Hester et al. 2021), *rmarkdown* (Version 2.11; Allaire et al. 2019), *rstan* (Version 2.21.2; Guo, Gabry, and Goodrich 2019), *servr* (Version 0.23; Xie 2019c), *SIN* (Version 0.6; Drton. 2013), *StanHeaders* (Version 2.21.0.7; Goodrich et al. 2019), *stringr* (Version 1.4.0; Wickham 2019b), *texPreview* (Version 1.5; Sidi and Polhamus 2020), *tibble* (Version 3.1.5; Müller and Wickham 2020), *tictoc* (Version 1.0.1; Izrailev 2014), *tidyR* (Version 1.1.4; Wickham and Henry 2019), *tidyverse* (Version 1.3.1; Wickham, Averick, et al. 2019), and *webshot* (Version 0.5.2; Chang 2018).

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



About the Authors

Bruno Nicenboim (<https://bnicenboim.github.io>) is an assistant professor in the department of Cognitive Science and AI at Tilburg University, the Netherlands. 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 a Masters degree in Linguistics in Tel Aviv University. During this time, he also worked in several IT companies. He then moved to Germany where he completed a PhD in Cognitive Science at the University of Potsdam, and worked two years as a postdoctoral researcher. His research interests are Bayesian methods, computational cognitive modeling, sentence comprehension, memory processes, decision making, and predictive processing. He regularly teaches short courses on Bayesian data analysis.

Daniel J. Schad (<https://danielschad.github.io/>) is professor of Quantitative Methods in the Psychology department at the Health and Medical University (HMU), at 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 and Mathematical 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 - Universitätsmedizin Berlin, Germany (partly also at the University of Potsdam), with research visits at the ETH Zürich, Switzerland, and the University College London, UK, working on model-free and model-based decision-making and Pavlovian-instrumental transfer in alcohol dependence, and on the cognitive and brain mechanisms underlying Pavlovian conditioning. He has worked as a postdoctoral researcher at the University of Potsdam, conducting research on quantitative methods in Cognitive Science, including contrasts, properties of significance tests, Bayesian Workflow,

and Bayes factor analyses, and has been assistant professor at the department of Cognitive Science and AI at Tilburg University.

Shravan Vasishth (<http://vasishth.github.io>) is professor of Psycholinguistics at the University of Potsdam, Germany. He holds the chair for Psycholinguistics and Neurolinguistics (Language Processing). After completing his Bachelors degree in Japanese from Jawaharlal Nehru University, New Delhi, India, he spent five years in Osaka, Japan, where he studied Japanese at the Osaka University of Foreign Studies, conducted research at Osaka University, and worked as an in-house translator in a patent law firm in Osaka. He completed an MS in Computer and Information Science (2000-2002) and a PhD in Linguistics (1997-2002) from the Ohio State University, Columbus, USA, and an MSc in Statistics (2011-2015) from the School of Mathematics and Statistics, University of Sheffield, UK. He is a chartered statistician (the Royal Statistical Society, UK; id 128307), and a member of the International Society for Bayesian Analysis. He is on the editorial board of the open access journal *Glossa: Psycholinguistics*. His research focuses on computational modeling of sentence processing in unimpaired and impaired populations, and the application of mathematical, computational, experimental, and statistical methods (particularly Bayesian methods) in linguistics and psychology. He created an annual summer school, Statistical Methods in Linguistics and Psychology (SMLP), which started in 2017: vasishth.github.io/smlp. He regularly teaches short courses on statistical data analysis (Bayesian and frequentist methods).

Part I

Foundational ideas



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 definitions are based on the axiomatic definition of probability by Kolmogorov (1933).

In the context of data analysis, we will talk about probability in the following way. Consider some data that we 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 *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 either have a correct or incorrect answer. We will say that the number of correct responses from a subject is generated from 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*.

¹Here, 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.

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.

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 outcomes (from 0 correct responses to 10). We are using lower-case $p(\cdot)$ here, and this is distinct from $P(\cdot)$, which we will use to talk about probabilities.

We will write that this PMF $p(y)$ depends on, or is conditional on, a particular fixed but unknown value for θ ; the PMF will be written $p(y|\theta)$.²

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

²A notational aside: In frequentist treatments, the PMF would be written $p(y; \theta)$, i.e., with a semi-colon rather than the conditional distribution marked by the vertical bar. The semi-colon is intended to indicate that in the frequentist paradigm, the parameters are fixed point values; by contrast, in the Bayesian paradigm, parameters are random variables. This has the consequence that for Bayesian, the distribution of y , $p(y)$ is really a conditional distribution, conditional on a random variable, here θ . For the frequentist, $p(y)$ requires some point value for θ , but it cannot be a conditional distribution because θ is not a random variable. We define conditional distributions later in this section.

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 what is called the *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 (but not identical) 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 been raining; 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.

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

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 multiplying 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)\$.

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:

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

Equating the two expansions, we get:

$$P(A|B)P(B) = P(B|A)P(A) \quad (1.3)$$

Dividing both sides by \$P(B)\$:

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

The above statement is Bayes' rule, and is the basis for all the statistical inference we will do in this book.

1.3 The law of total probability

Related to the above discussion of conditional probability is the law of total probability. Suppose that we have \$A_1, \dots, A_n\$ distinct events that are pairwise disjoint which together make up the entire sample space \$S\$; see Figure 1.1. Then, \$P(B)\$, the probability of an event \$B\$, will be the sum of the probabilities \$P(B \cap A_i)\$, i.e., the sum of the joint probabilities of \$B\$ and each \$A\$ occurring. Formally:

$$P(B) = \sum_{i=1}^n P(B \cap A_i) \quad (1.5)$$

Because of the conditional probability rule, we can rewrite this as:

$$P(B) = \sum_{i=1}^n P(B|A_i)P(A_i) \quad (1.6)$$

Thus, the probability of B is the sum of the conditional probabilities $P(B|A_i)$ weighted by the probability $P(A_i)$. We will see the law of total probability in action below when we talk about *marginal likelihood*.

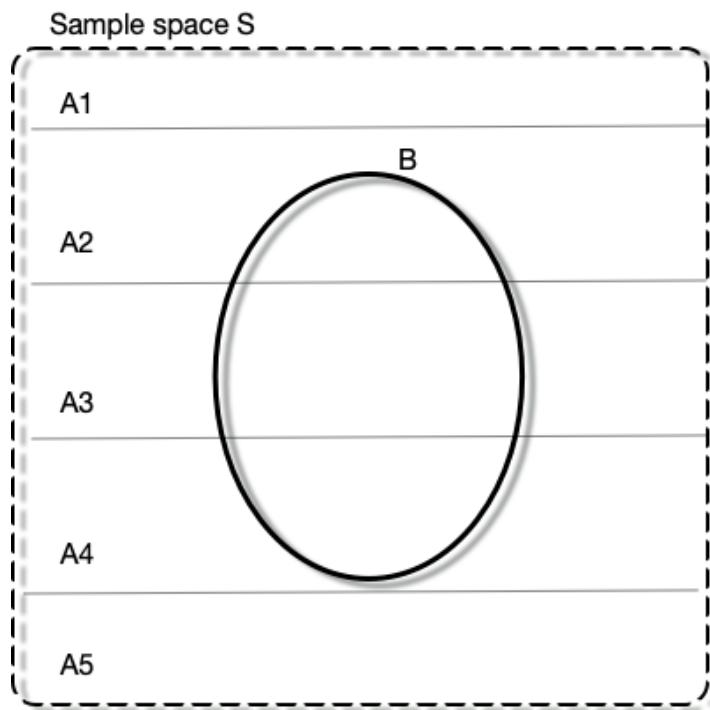


FIGURE 1.1: Illustration of law of total probability.

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. After discussing this discrete random variable, we present another 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.

1.4 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: given n trials, there can be only two possible outcomes in each trial, a success or a failure, and there is some true unknown probability θ of success that we want to estimate. When the number of trials is one, the random variable is said to have a Bernoulli distribution.

One way to empirically estimate this probability of success is to carry out a *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 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 successes 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 5 7 5 6 6 4 4 5 6 4 7 6 7 3 3 5 4 8 4
```

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: $y = 0, 1, 2, \dots, 10$ 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 1.2 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.

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

$$\text{Binomial}(k|n, \theta) = \binom{n}{k} \theta^k (1 - \theta)^{n-k} \quad (1.7)$$

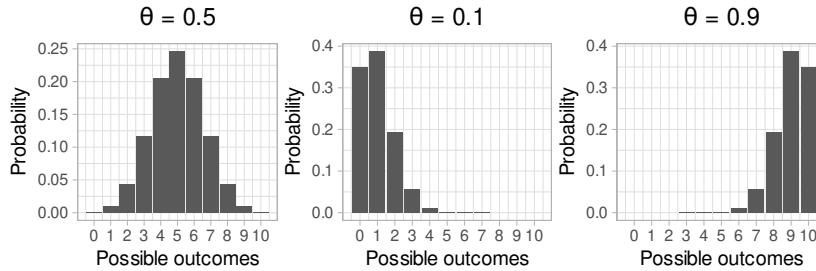


FIGURE 1.2: Probability mass functions of a binomial distribution assuming 10 trials, with 50%, 10%, and 90% probability of success.

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 second 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 $Y \sim \text{Binomial}(n, \theta)$, 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 $Y \sim f(\theta)$. We are using $f(\cdot)$ synonymously with $p(\cdot)$ to represent a probability distribution.

1.4.1 The mean and variance of the Binomial distribution

It is possible to analytically compute the mean (expectation) and variance of the PMF associated with the Binomial random variable Y .

The expectation of a discrete random variable Y with probability mass function $f(y)$, is defined as

$$E[Y] = \sum_y y \cdot f(y) \quad (1.8)$$

As a really simple example, suppose that we toss a fair coin once. The pos-

sible outcomes are Tails (represented as 0) and Heads (represented as 1), each with equal probability, 0.5. The expectation is:

$$E[Y] = \sum_y y \cdot f(y) = 0 \cdot 0.5 + 1 \cdot 0.5 = 0.5 \quad (1.9)$$

The expectation has the interpretation that if we were to do the experiment a large number of times and calculate the sample mean of the observations, in the long run we would approach the value 0.5. Another way to look at the above definition is that the expectation gives us the weighted mean of the possible outcomes, weighted by the respective probabilities of each outcome.

Without getting into the details of how these are derived mathematically (Kerns 2014), we just state here that the mean of Y (the expectation $E[Y]$) and variance of Y (written $Var(Y)$) of a Binomial distribution with parameter θ and n trials are $E[Y] = n\theta$ and $Var(Y) = n\theta(1 - \theta)$.

In the Binomial example above, 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) parameter θ . Once we have estimated θ in this way, we can also obtain an 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 the parameter θ varies (from 0 to 1).

For example, suppose that we 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:

$$\text{Binomial}(k = 7, n = 10|\theta) = \binom{10}{7} \theta^7 (1 - \theta)^{10-7} \quad (1.10)$$

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 1.3.

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. In the binomial, the proportion of successes k/n happens to be the maximum likelihood estimate of the parameter θ .

A crucial point: the “most likely” value of the parameter is with respect to the data at hand. The data are used to choose as an estimate of the unknown parameter a value for which the probability (discrete case) or probability density (continuous case) of getting the sample values is a maximum. The MLE from a particular sample of data need not invariably give 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 just happened to observe only one success out of ten by chance, even if the true

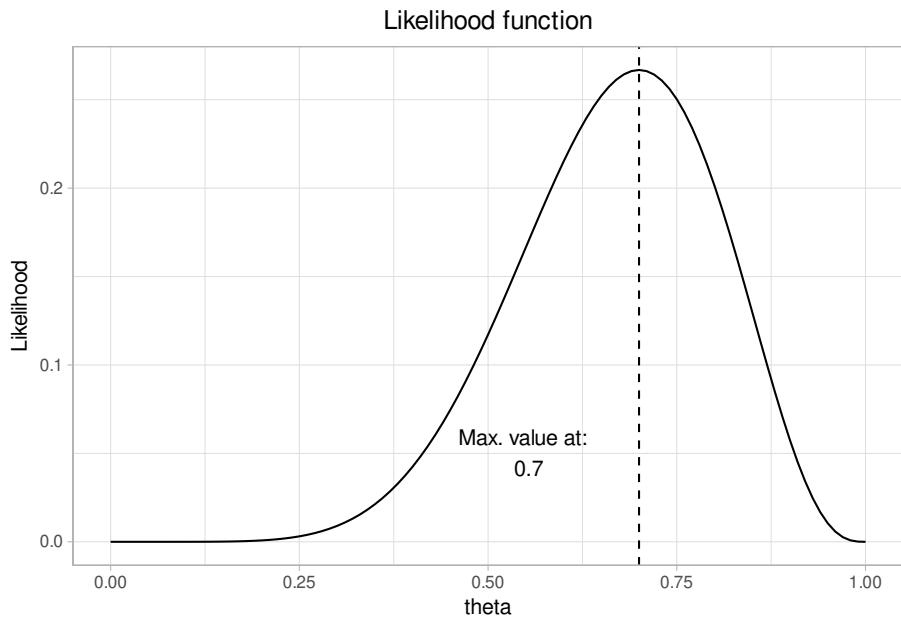


FIGURE 1.3: The likelihood function for 7 successes out of 10.

θ were 0.5. If we were to repeatedly run the experiment, in the long run, the MLE computed each time would converge around the true value of the parameter. Figure 1.4 illustrates this point.

1.4.2 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 ”; and “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 decided on a probability distribution that is assumed to have generated the data, and how to extract this information using R. 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 introduced above.

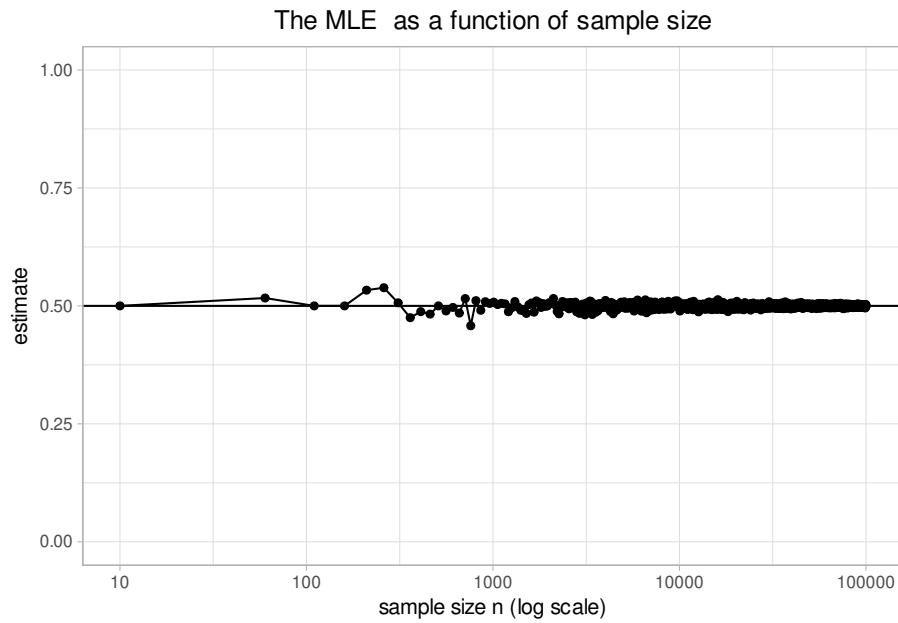


FIGURE 1.4: The plot shows the estimate of the mean proportion of successes sampled from a Binomial distribution with true probability of success 0.5, with increasing sample sizes. As the sample size increases, the estimate converges to the true value of 0.5.

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

The Binomial distribution shown in Figure 1.2 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] 0.246
```

The probabilities of success when θ is 0.1 or 0.9 can be computed by re-

placing 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] 0.00149 0.00149
```

The probability of a particular outcome like $k = 5$ successes is only computable in the discrete case. In the continuous case, the probability of obtaining a particular point value will always be zero (we discuss this when we turn to continuous probability distributions below).

1.4.2.2 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] 0.0547
```

Mathematically, we could write the above summation as:

$$\sum_{k=0}^2 \binom{n}{k} \theta^k (1-\theta)^{n-k} \quad (1.11)$$

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] 0.0547
```

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] 0.0547
```

The specification `lower.tail = TRUE` (the default value) ensures that the summation goes from 2 to numbers smaller than 2 (which lie in the lower tail of the distribution in Figure 1.2). If we wanted to know what the probability is of obtaining 3 or more successes out of 10, we can set `lower.tail` to `FALSE`:

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

```
## [1] 0.945
```

```
## equivalently:  
## sum(dbinom(3:10,size = 10, prob = 0.5))
```

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 1.5.

1.4.2.3 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 1.5, 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:

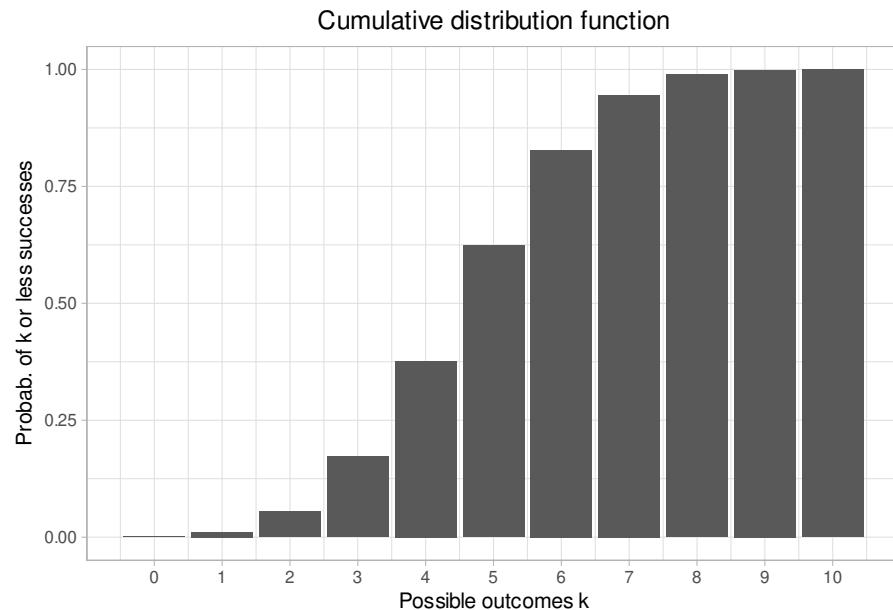


FIGURE 1.5: The cumulative distribution function for a Binomial distribution assuming 10 trials, with 50% probability of success.

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

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

One can visualize the inverse CDF of the Binomial as in Figure 1.6.

1.4.2.4 Generate simulated data from a $\text{Binomial}(n, \theta)$ 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] 4
```

The above code generates the number of successes in an experiment with 10 trials. Repeatedly run the above code; we will get different sequences

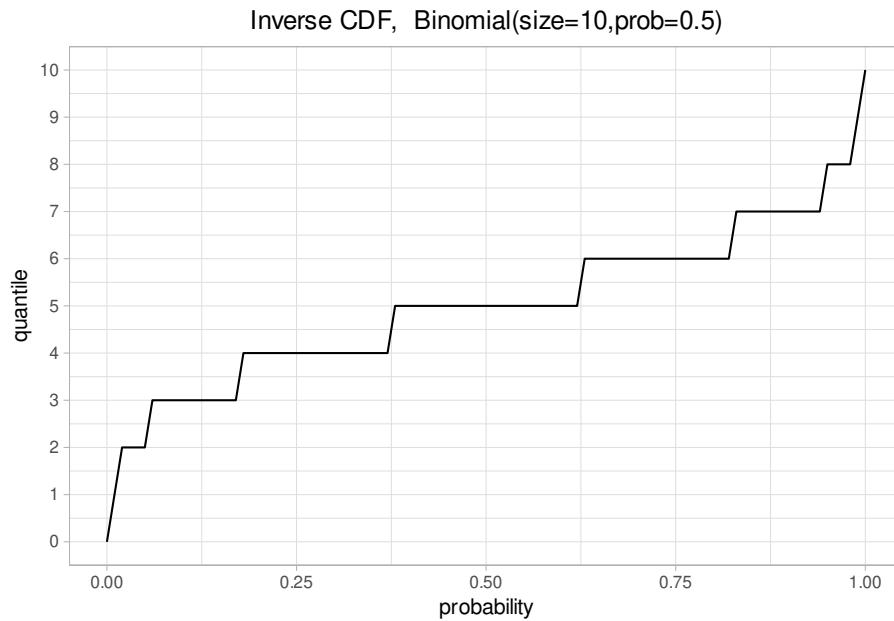


FIGURE 1.6: The inverse CDF for the Binomial(size=10,prob=0.5).

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))
```

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

```
mean(y) * 10
```

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

```
sum(y)
```

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

As mentioned earlier, if there is only one trial, then instead of the Binomial distribution, we have a Bernoulli distribution. For example, if we

have 10 observations from a Bernoulli distribution, where the probability of success is 0.5, we can simulate data as follows using the function `rbern` from the package `extraDistr`.

```
rbern(n=10,prob=0.5)
```

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

Compare this with the code shown above for generating binomial data: `rbinom(10, size = 1, prob = 0.5)`.

1.5 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 we 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:

$$\text{Normal}(y|\mu, \sigma) = f(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) \quad (1.12)$$

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 1.7 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.

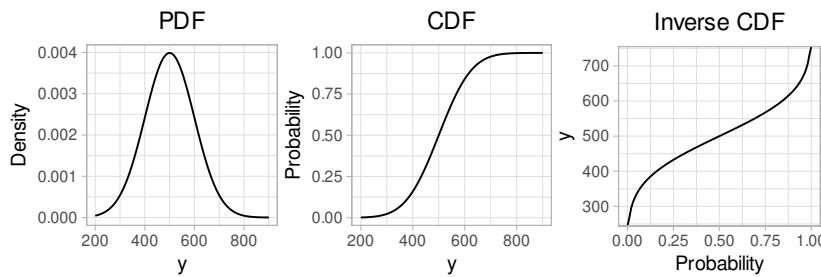


FIGURE 1.7: The PDF, CDF, and inverse CDF for the $\text{Normal}(\mu = 500, \sigma = 100)$.

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] 0.976
```

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 it is only meaningful to ask about probabilities between two different point values; 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 of observing that value q or something less (or more) than it is p ? 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 $\text{Normal}(500, 100)$ distribution. Formally, we would write this as $P(Y < q)$.

```
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%.

- Generate 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 sample mean and standard deviation³ are:

$$\bar{y} = \frac{\sum_{i=1}^n y_i}{n} \quad (1.13)$$

$$sd(y) = \sqrt{\frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n}} \quad (1.14)$$

For example, we can generate 10 data points using the `rnorm` function, and then use the simulated data to compute the mean and standard deviation:

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

```
## [1] 495
```

```
sd(y)
```

```
## [1] 103
```

Again, the sample mean and sample standard deviation computed from a particular (simulated or real) data set need not necessarily be close to the true values of the respective parameters. Especially when sample size is small, one can end up with missestimates of the mean and sd.

³R will compute the standard deviation 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 $n - 1$.

Incidentally, simulated data can be used to generate all kinds of statistics. For example, we can compute the lower and upper bounds of a 95% confidence interval from simulated data as follows:

```
quantile(y, probs = c(0.025, 0.975))
```

```
## 2.5% 97.5%
##   339    617
```

Later on, we will be using samples to produce summary statistics like the ones shown above.

1.5.1 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] 0.242
```

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 cumulative distribution function or CDF: in R, for the Normal distribution, this is 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] 0.954
```

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 $\text{Binomial}(n = 10, \theta = 0.5)$ can be computed using either `dbinom` or `pbinom`:

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

```
## [1] 0.0439
```

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

```
## [1] 0.0439
```

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.

1.5.2 Truncating a normal distribution

In the above discussion, the support for the normal distribution ranges from minus infinity to plus infinity. One can define PDFs with a more limited support; an example would be a normal distribution whose PDF $f(x)$ is such that the lower bound is truncated at 0 to allow only positive values. In such a case, the area under the range minus infinity to zero ($\int_{-\infty}^0 f(x) dx$) will be 0 because the range lies outside the support of the truncated normal distribution. Also, if one truncates a probability density function like the standard Normal ($\text{Normal}(0, 1)$) at 0, in order to make the area between zero and plus infinity sum up to 1, we would have to multi-

ply the truncated distribution $f(x)$ by some factor k such that the following integral sums to 1:

$$k \int_0^\infty f(x) dx = 1 \quad (1.15)$$

Clearly, this factor is $k = \frac{1}{\int_0^\infty f(x) dx}$. For the standard normal, this integral is easy to compute; we just calculate the complement of the cumulative distribution (ccdf):

```
pnorm(0, mean = 0, sd = 1, lower.tail = FALSE)
```

```
## [1] 0.5
```

```
## alternatively:
```

```
1 - pnorm(0, mean = 0, sd = 1, lower.tail = TRUE)
```

```
## [1] 0.5
```

Also, if we had truncated the distribution at 0 to the right instead of the left (allowing only negative values), we would have to find the factor k in the same way as above, except that we would have to find k such that:

$$k \int_{-\infty}^0 f(x) dx = 1 \quad (1.16)$$

For the standard normal case, in R, this factor would require us to use the CDF:

```
pnorm(0, mean = 0, sd = 1, lower.tail = TRUE)
```

```
## [1] 0.5
```

Later in this book, we will be using such truncated distributions when doing Bayesian modeling, and when we use them, we will want to multiply the truncated distribution by the factor k to ensure that it is still a proper PDF that sums to 1.

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

	X=1	X=2	X=3	X=4	X=5	X=6	X=7
y=0	0.018	0.023	0.040	0.043	0.063	0.049	0.055
y=1	0.031	0.053	0.086	0.096	0.147	0.153	0.142

1.6 Bivariate and multivariate distributions

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

1.6.1 Example 1: Discrete bivariate distributions

Starting with the discrete case, consider the discrete bivariate distribution shown below. These are data from an experiment where, inter alia, in each trial a Likert acceptability rating and a question-response accuracy were recorded (the data are from a study by Laurinavichyute (2020), used with permission here). Load the data by loading the R package `bcogsci`.

```
data("df_discreteagrmt")
```

Figure 1.8 shows the *joint probability mass function* of two random variables X and Y. The random variable X consists of 7 possible values (this is the 1–7 Likert response scale), and the random variable Y is question-response accuracy, with 0 representing an incorrect response, and 1 representing a correct response.

One can also display Figure 1.8 as a table; see Table 1.1.

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

Table 1.1 shows the joint probability mass function $p_{X,Y}(x, y)$.

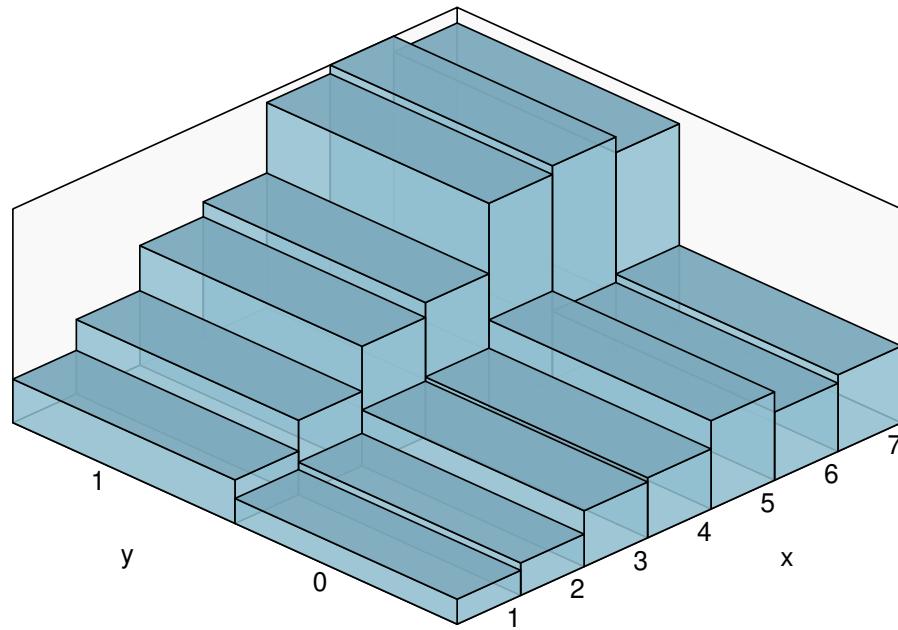


FIGURE 1.8: Example of a discrete bivariate distribution. In these data, in every trial, two pieces of information were collected: Likert responses and yes-no question responses. The random variable X represents Likert scale responses on a scale of 1-7, and the random variable Y represents 0, 1 (incorrect, correct) responses to comprehension questions.

1.6.1.1 Marginal distributions

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

$$p_Y(y) = \sum_{x \in S_X} p_{X,Y}(x, y). \quad (1.17)$$

Similarly, the marginal distribution p_X is defined as:

$$p_X(x) = \sum_{y \in S_Y} p_{X,Y}(x, y). \quad (1.18)$$

p_Y is computed, by summing up the rows; and p_X by summing up the

TABLE 1.2: The joint PMF for two random variables X and Y, along with the marginal distributions of X and Y.

	X=1	X=2	X=3	X=4	X=5	X=6	X=7	P(Y)
y=0	0.018	0.023	0.04	0.043	0.063	0.049	0.055	0.291
y=1	0.031	0.053	0.086	0.096	0.147	0.153	0.142	0.709
P(X)	0.049	0.077	0.126	0.139	0.21	0.202	0.197	

columns. We can see why this is called the marginal distribution; the result appears in the margins of the table.

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

```
##   y=0   y=1
## 0.291 0.709
```

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

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

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

```
##   x=1   x=2   x=3   x=4   x=5   x=6   x=7
## 0.0491 0.0766 0.1257 0.1394 0.2102 0.2020 0.1969
```

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

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

The marginal probabilities sum to 1, as they should. Table 1.2 shows the marginal probabilities.

To compute the marginal distribution of X, one is summing over all the Ys; and to compute the marginal distribution of Y, one sums over all the X's.

We say that we are *marginalizing out* the random variable that we are summing over. One can also visualize the two marginal distributions using barplots (Figure 1.9).

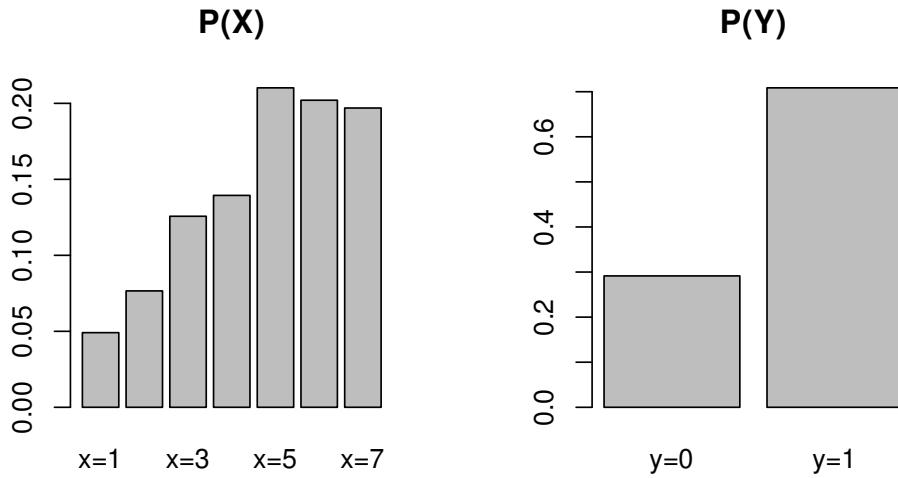


FIGURE 1.9: The marginal distributions of the random variables X and Y, presented as barplots.

1.6.1.2 Conditional distributions

For computing conditional distributions, recall that conditional probability is defined as:

$$p_{X|Y}(x | y) = \frac{p_{X,Y}(x, y)}{p_Y(y)} \quad (1.19)$$

and

$$p_{Y|X}(x | y) = \frac{p_{X,Y}(x, y)}{p_X(x)} \quad (1.20)$$

The conditional distribution of a random variable X given that $Y = y$, where y is some specific (fixed) value, is:

$$p_{X|Y}(x | y) = \frac{p_{X,Y}(x, y)}{p_Y(y)} \quad \text{provided } p_Y(y) = P(Y = y) > 0 \quad (1.21)$$

TABLE 1.3: A table for listing conditional distributions of X given Y.

	X=1	X=2	X=3	X=4	X=5	X=6	X=7
p_X Y(x y=0)	0.062						
p_X Y(x y=1)							

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

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

$$\begin{aligned} p_{X|Y}(1 | 0) &= \frac{p_{X,Y}(1, 0)}{p_Y(0)} \\ &= \frac{0.018}{0.291} \\ &= 0.062 \end{aligned} \tag{1.22}$$

This conditional probability value will occupy the cell X=1, Y=0 in Table 1.3 summarizing the conditional probability distribution $p_{X|Y}$. In this way, one can fill in the entire table, which will then represent the conditional distributions $p_{X|Y=0}$ and $p_{X|Y=1}$. The reader may want to take a few minutes to complete Table 1.3.

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

1.6.1.3 Covariance and correlation

Here, we briefly define the covariance and correlation of two discrete random variables. For detailed examples and discussion, see the references at the end of the chapter.

The covariance of two (discrete) random variables X and Y is defined as follows. $E[\cdot]$ refers to the expectation of a random variable.

$$Cov(X, Y) = E[(X - E[X])(Y - E[Y])] \tag{1.23}$$

It is possible to show that this is equivalent to:

$$\text{Cov}(X, Y) = E[XY] - E[X]E[Y] \quad (1.24)$$

The expectation $E[XY]$ is defined to be:

$$E[XY] = \sum_x \sum_y xyf_{X,Y}(x, y) \quad (1.25)$$

If the standard deviations of the two random variables is σ_X and σ_Y , the correlation between the two random variables, ρ_{XY} , is defined as:

$$\rho_{XY} = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y} \quad (1.26)$$

1.6.2 Example 2: Continuous bivariate distributions

Consider now the continuous bivariate case; this time, we will use simulated data. Consider two normal random variables X and Y , each of which coming from, for example, a $\text{Normal}(0, 1)$ distribution, with some correlation ρ between the two random variables.

A bivariate distribution for two random variables X and Y , each of which comes from a normal distribution, is expressed in terms of the means and standard deviations of each of the two distributions, and the correlation ρ_{XY} between them. We assume here that the mean of both random variables is 0. The standard deviations and correlation are expressed in a special form of a 2×2 matrix called a variance-covariance matrix Σ . If ρ_{XY} is the correlation between the two random variables, and σ_X and σ_Y the respective standard deviations, the variance-covariance matrix is written as:

$$\Sigma = \begin{pmatrix} \sigma_X^2 & \rho_{XY}\sigma_X\sigma_Y \\ \rho_{XY}\sigma_X\sigma_Y & \sigma_Y^2 \end{pmatrix} \quad (1.27)$$

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

One occasionally encounters a “singularity” warning when fitting hierarchical models with the package `lme4`. This warning signals the fact that a variance-covariance matrix involved in the model is not invertible. For more details on what this means, see Fieller (2016).

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

$$\begin{pmatrix} X \\ Y \end{pmatrix} \sim \mathcal{N}_2 \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma \right) \quad (1.28)$$

The joint PDF is written with reference to the two variables $f_{X,Y}(x, y)$. It has the property that the area under the curve sums to 1. Formally, we would write this as a double integral: we are summing up the area under the curve for both X and Y (hence two integrals).

$$\iint_{S_{X,Y}} f_{X,Y}(x, y) dx dy = 1 \quad (1.29)$$

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

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

$$\begin{aligned} F_{X,Y}(u, v) &= P(X < u, Y < v) \\ &= \int_{-\infty}^u \int_{-\infty}^v f_{X,Y}(x, y) dy dx \text{ for } (x, y) \in \mathbb{R}^2 \end{aligned} \quad (1.30)$$

Just as in the discrete case, the marginal distributions can be derived by marginalizing out the other random variable:

$$f_X(x) = \int_{S_Y} f_{X,Y}(x, y) dy \quad f_Y(y) = \int_{S_X} f_{X,Y}(x, y) dx \quad (1.31)$$

Here, S_X and S_Y are the respective supports.

Here, the integral sign \int is the continuous equivalent of the summation sign \sum in the discrete case. Luckily, we will never have to compute such integrals ourselves; but it is important to appreciate how a marginal distribution arises from a bivariate distribution—by integrating out or marginalizing out the other random variable.

A visualization will help. The figures below shows a bivariate distribution

with zero correlation (Figure 1.10), a negative (Figure 1.11) and a positive correlation (Figure 1.12).

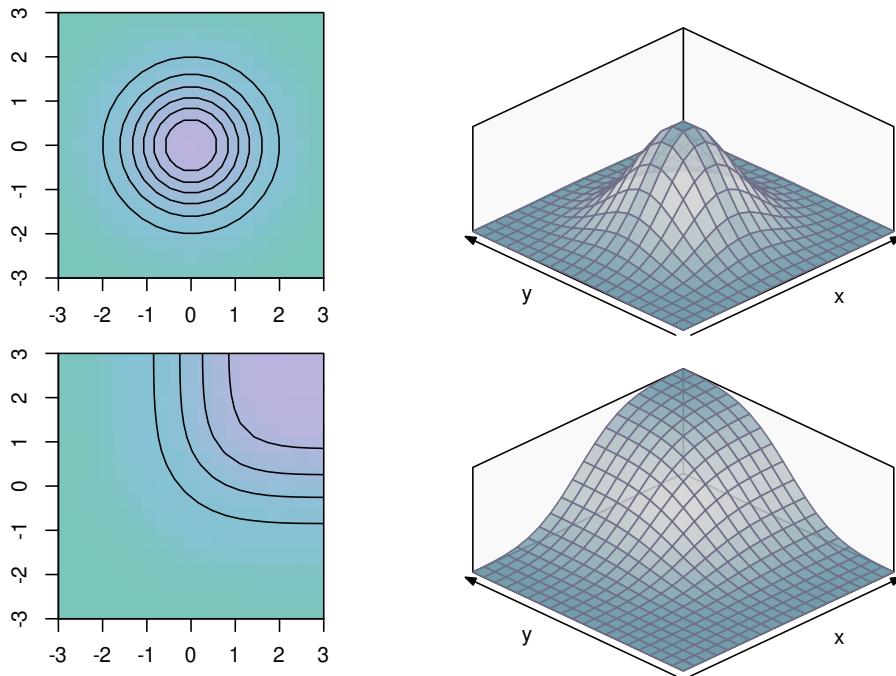


FIGURE 1.10: A bivariate Normal distribution with zero correlation. Shown are four plots: the top-right plot shows the three-dimensional bivariate density, the top-left plot the contour plot of the distribution (seen from above). The lower plots show the cumulative distribution function from two views,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

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

1.6.3 Generate simulated bivariate (multivariate) data

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

Here is how we would generate such data. First, define a variance-

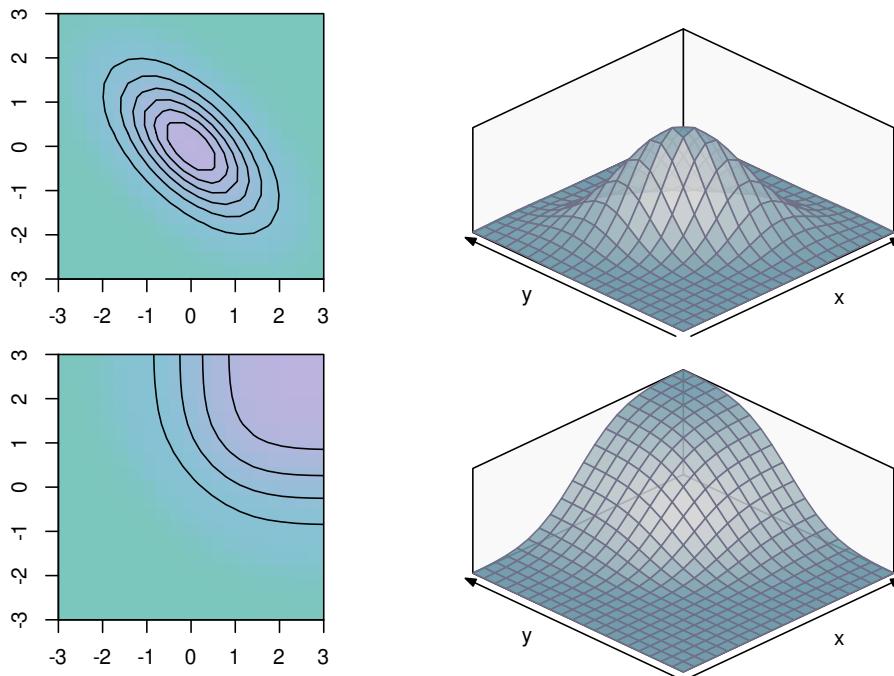


FIGURE 1.11: A bivariate Normal distribution with a negative correlation of -0.6 . Shown are four plots: the top-right plot shows the three-dimensional bivariate density, the top-left plot the contour plot of the distribution (seen from above). The lower plots show the cumulative distribution function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

covariance matrix; then, use the multivariate analog of the `rnorm` function, `mvrnorm`, from the `MASS` package to generate 100 data-points.

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

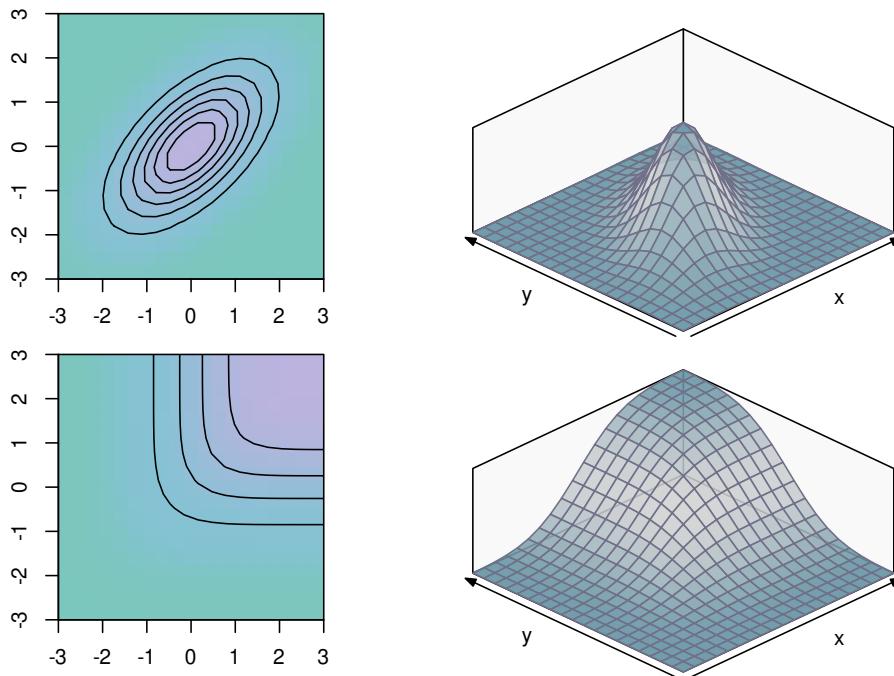


FIGURE 1.12: A bivariate Normal distribution with a positive correlation of 0.6. Shown are four plots: the top-right plot shows the three-dimensional bivariate density, the top-left plot the contour plot of the distribution (seen from above). The lower plots show the cumulative distribution function from two views, as a three-dimensional plot and as a contour plot.

```

Sigma = Sigma
)
head(u, n = 3)

```

```

##      [,1]  [,2]
## [1,] 1.45 10.83
## [2,] -6.19 -3.88
## [3,]  5.06  5.79

```

Figure 1.13 confirms that the simulated data are positively correlated.

One final useful fact about the variance-covariance matrix—one that we will need later—is that it can be decomposed into the component standard

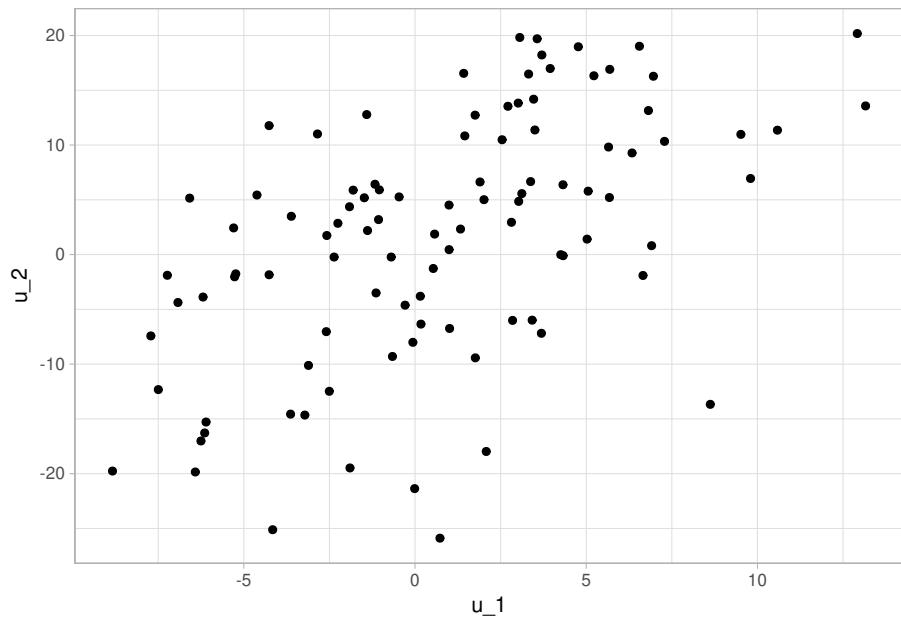


FIGURE 1.13: The relationship between two positively correlated random variables, generated by simulating data using the R function `mvrnorm` from the MASS library.

deviations and an underlying correlation matrix. For example, consider the matrix above:

```
Sigma
```

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

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

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

```

## [1] 5 10

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

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

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

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

```

Given these two matrices, one can reassemble the variance-covariance matrix.⁴

```

sd_diag %*% corrmatrix %*% sd_diag

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

```

1.7 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

⁴There is a built-in convenience function, `sdcor2cov` in the `SIN` package that does this calculation, taking the vector of standard deviations (not the diagonal matrix) and the correlation matrix to yield the variance-covariance matrix: `sdcor2cov(stddev = sds, corr = corrmatrix)`.

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

$$p(k = 8, n = 10 | \theta) = \binom{10}{8} \theta^8 (1 - \theta)^2 \quad (1.32)$$

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/density 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:

$$\begin{aligned} p(k = 8, n = 10) &= \binom{10}{8} \theta_1^8 (1 - \theta_1)^2 \times p(\theta_1) \\ &\quad + \binom{10}{8} \theta_2^8 (1 - \theta_2)^2 \times p(\theta_2) \\ &\quad + \binom{10}{8} \theta_3^8 (1 - \theta_3)^2 \times p(\theta_3) \end{aligned} \quad (1.33)$$

Writing the θ values and their probabilities, we get:

$$\begin{aligned}
 p(k = 8, n = 10) &= \binom{10}{8} 0.1^8 (1 - 0.1)^2 \times \frac{1}{3} \\
 &\quad + \binom{10}{8} 0.5^8 (1 - 0.5)^2 \times \frac{1}{3} \\
 &\quad + \binom{10}{8} 0.9^8 (1 - 0.9)^2 \times \frac{1}{3}
 \end{aligned} \tag{1.34}$$

Thus, a marginal likelihood is a kind of weighted sum of the likelihood, weighted by the possible values of the parameter.⁵

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:

$$p(k = 8, n = 10) = \int_0^1 \binom{10}{8} \theta^8 (1 - \theta)^2 d\theta \tag{1.35}$$

This statement is computing something similar to what we computed above with the three discrete parameter values, 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 compute the integral in R:

```

BinLik <- function(theta) {
  choose(10, 8) * theta^8 * (1 - theta)^2
}
integrate(BinLik, lower = 0, upper = 1)$value
## [1] 0.0909

```

⁵Where does the above formula come from? It falls out from the law of total probability discussed above!

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

1.8 Summary of useful R functions relating to distributions

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

TABLE 1.4: Important R functions relating to random variables.

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

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. One can look up these different distributions in, for example, Blitzstein and Hwang (2014).

1.9 Summary

This chapter briefly reviewed some very basic concepts in probability theory, univariate discrete and continuous random variables, and bivariate distributions. An important set of functions we encountered are the d-p-q-r family of functions for different distributions; these are very useful for understanding the properties of commonly used distributions, visualizing distributions, and for simulating data. Distributions will play a central role in this book; for example, knowing how to visualize distribu-

tions will be important for deciding on prior distributions for parameters. Other important ideas we learned about were marginal and conditional probability, marginal likelihood, and how to define multivariate distributions; these concepts will play an important role in Bayesian statistics.

1.10 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. Ross (2002) is a more advanced treatment which discusses random variable theory and illustrates applications of probability theory. A good formal introduction to mathematical statistics (covering classical frequentist theory) is Miller and Miller (2004). The freely available book by Kerns (2014) introduces frequentist and Bayesian statistics from the ground up in a very comprehensive and systematic manner; the source code for the book is available from <https://github.com/gjkerns/IPSUR>. The open-access book, *Probability and Statistics: a simulation-based introduction*, by Bob Carpenter is also worth studying: <https://github.com/bob-carpenter/prob-stats>. A thorough introduction to the matrix algebra needed for statistics, with examples using R, is provided in Fieller (2016).

1.11 Exercises

Exercise 1.1. Practice using the `pnorm` function - Part 1

Given a normal distribution with mean 500 and standard deviation 100, use the `pnorm` function to calculate the probability of obtaining values between 200 and 800 from this distribution.

Exercise 1.2. Practice using the `pnorm` function - Part 2

Calculate the following probabilities. Given a normal distribution with mean 800 and standard deviation 150, what is the probability of getting

- a score of 700 or less
- a score of 900 or more
- a score of 800 or more

Exercise 1.3. Practice using the `pnorm` function - Part 3

Given a normal distribution with mean 600 and standard deviation 200, what is the probability of getting

- a score of 550 or less.
- a score between 300 and 800.
- a score of 900 or more.

Exercise 1.4. Practice using the `qnorm` function - 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 0.10.
- the area (the probability) to the left of the upper boundary is 0.90.

Exercise 1.5. Practice using the `qnorm` function - Part 2

Given a normal distribution with mean 650 and standard deviation 125. There exist two quantiles, the lower quantile q_1 and the upper quantile q_2 , that are equidistant from the mean 650, such that the area under the curve of the Normal between q_1 and q_2 is 80%. Find q_1 and q_2 .

Exercise 1.6. Practice getting summaries from samples - Part 1

Given data that is generated as follows:

```
data_gen1 <- rnorm(1000, 300, 200)
```

Calculate the mean, variance, and the lower quantile q_1 and the upper quantile q_2 , that are equidistant and such that the range of probability between them is 80%.

Exercise 1.7. Practice getting summaries from samples - Part 2.

This time we generate the data with a truncated normal distribution from the package `extraDistr`. The details of this distribution will be discussed later in 4.1 and in the Box 4.1, but for now we can treat it as an unknown generative process:

```
data_gen1 <- rtnorm(1000, 300, 200, a = 0)
```

Calculate the mean, variance, and the lower quantile q_1 and the upper quantile q_2 , that are equidistant and such that the range of probability between them is 80%.

Exercise 1.8. Practice with a variance-covariance matrix for a bivariate distribution.

Suppose that you have a bivariate distribution where one of the two random variables comes from a Normal distribution with mean $\mu_X = 600$ and standard deviation $\sigma_X = 100$, and the other from a Normal distribution with mean $\mu_Y = 400$ and standard deviation $\sigma_Y = 50$. The correlation ρ_{XY} between the two random variables is 0.4. Write down the variance-covariance matrix of this bivariate distribution as a matrix (with numerical values, not mathematical symbols), and then use it to generate 100 pairs of simulated data points. Plot the simulated data such that the relationship between the random variables X and Y is clear. Generate two sets of new data (100 pairs of data points each) with correlation -0.4 and 0 , and plot these alongside the plot for the data with correlation 0.4 .



2

Introduction to Bayesian data analysis

Before we can start analyzing realistic data sets using Bayes' rule, it is important to understand the application of Bayes' rule in one of the simplest of cases, data involving the Binomial likelihood. This simple case is important to understand because it encapsulates the essence of the Bayesian approach to data analysis, and because it allows us to analytically work out the posterior distribution of the parameter of interest, using just a pen and paper. This simple case also helps us to appreciate a crucial point: the posterior distribution of a parameter is a compromise between the prior and the likelihood. This important insight will play a central role in the realistic data analysis situations we will cover in the remainder of this book.

2.1 Bayes' rule

Recall Bayes' rule: When A and B are observable discrete events (such as “it has been raining” or “the streets are wet”), we can state the rule as follows:

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)} \quad (2.1)$$

Given a vector of data \mathbf{y} , Bayes' rule allows us to work out the posterior distributions of the parameters of interest, which we can represent 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 function (continuous case) or a probability mass function (discrete case).

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

The above statement can be rewritten in words as follows:

$$\text{Posterior} = \frac{\text{Likelihood} \cdot \text{Prior}}{\text{Marginal Likelihood}} \quad (2.3)$$

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 as a function of Θ .
- The *Prior* is the initial probability distribution of the parameter(s), 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.

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

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

“It’s raining. I’m going to take the ...”

Suppose that 100 subjects are asked to complete the sentence. If 80 out of 100 subjects complete the sentence with “umbrella,” the estimated cloze probability or predictability (given the preceding context) would be $\frac{80}{100} = 0.8$. 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: $\hat{\theta} = 0.8$.

An important point: 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 subjects ($\frac{8}{10}$), 100 subjects ($\frac{80}{100}$), or 100000 subjects ($\frac{80000}{100000}$). The uncertainty of the estimate 0.8 is different in each of these cases, and that is very relevant when drawing conclusions from data.

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

2.2.1 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.

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

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 $n = 100$ and $k = 80$, these data are now a fixed quantity. The PMF above now becomes a function of θ , the likelihood function:

$$p(k = 80|n = 100, \theta) = \binom{n}{k} \theta^{80} (1 - \theta)^{20} \quad (2.5)$$

The above function is 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).

Recall 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 .

2.2.2 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 ranges 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.

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

The term $B(a, b)$ expands to $\int_0^1 \theta^{a-1} (1 - \theta)^{b-1} d\theta$, and is a normalizing constant that 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 ?? shows

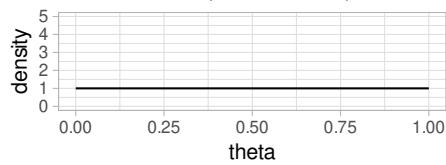
¹In some textbooks, you may see the PDF of the Beta distribution with the normalizing constant $\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}$ (the expression $\Gamma(n)$ is defined as $(n-1)!$):

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

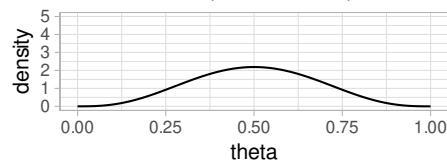
These two statements for the Beta distribution are identical because $B(a, b)$ can be shown to be equal to $\frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$ (Ross 2002).

the different Beta distribution shapes given different values of a and b .

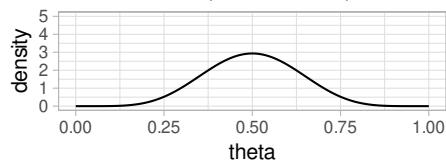
Beta($a = 1, b = 1$)



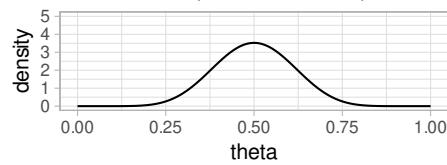
Beta($a = 4, b = 4$)



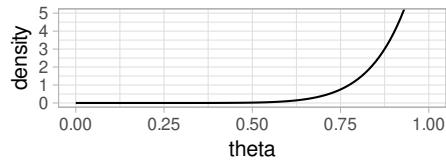
Beta($a = 7, b = 7$)



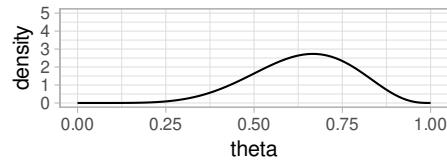
Beta($a = 10, b = 10$)



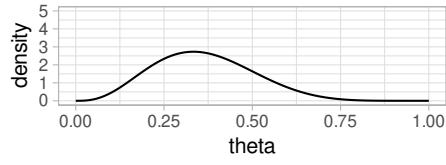
Beta($a = 10, b = 1$)



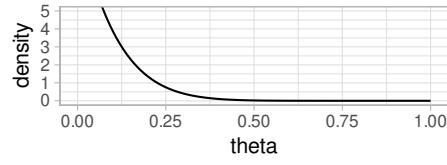
Beta($a = 7, b = 4$)



Beta($a = 4, b = 7$)



Beta($a = 1, b = 10$)



As in the Binomial and Normal distributions that we saw in chapter 1, one can analytically derive the formulas for the expectation and variance of the Beta distribution. These are:

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

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] 0.184 0.816
```

The credible interval chosen above is an equal-tailed interval: the area below the lower bound and above the upper bound is the same (0.025 in the above case). One could define alternative intervals; for example, in a distribution with only one mode (a unimodal distribution), one could choose to use the narrowest interval that contains the mode. This is called the highest posterior density interval (HDI). In skewed posterior distributions, the equal-tailed credible interval and the HDI will not be identical, because the HDI will have unequal tail probabilities. Some authors, such as Kruschke (2014), prefer to report the HDI. We will use the equal-tailed interval in this book, simply because this is the standard output in `stan` and `brms`.

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] 0.289 0.711
```

In Figure ??, 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, such as *flat*, *non-informative prior*, or *uninformative 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. In the above example, the larger our param-

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

We will discuss prior specification in detail later in chapter 6. 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:

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

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 .

2.2.3 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:

$$\text{Posterior} = \frac{\text{Likelihood} \cdot \text{Prior}}{\text{Marginal Likelihood}} \quad (2.9)$$

Replace the terms for likelihood and prior into this equation:

$$p(\theta|n = 100, k = 80) = \frac{\left[\binom{100}{80} \theta^{80} (1 - \theta)^{20} \right] \times \left[\frac{1}{B(4, 4)} \times \theta^3 (1 - \theta)^3 \right]}{p(k = 80)} \quad (2.10)$$

where $p(k = 80)$ is $\int_0^1 p(k = 80|n = 100, \theta) p(\theta) d\theta$. 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:

$$p(\theta|n = 100, k = 80) = \left[\frac{\binom{100}{80}}{B(4, 4) \times p(k = 80)} \right] [\theta^{80} (1 - \theta)^{20} \times \theta^3 (1 - \theta)^3] \quad (2.11)$$

The first term that is in square brackets, $\frac{\binom{100}{80}}{B(4,4) \times p(k=80)}$, is all the constants collected together, and is the normalizing constant we have seen before; it makes the posterior distribution $p(\theta|n = 100, k = 80)$ 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.

$$p(\theta|n = 100, k = 80) \propto [\theta^{80}(1 - \theta)^{20} \times \theta^3(1 - \theta)^3] \quad (2.12)$$

A common way of writing the above equation is:

$$\text{Posterior} \propto \text{Likelihood} \times \text{Prior} \quad (2.13)$$

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.

$$p(\theta|n = 100, k = 80) \propto [\theta^{80+3}(1 - \theta)^{20+3}] = \theta^{83}(1 - \theta)^{23} \quad (2.14)$$

The expression on the right-hand side corresponds to a Beta distribution with parameters $a = 84$, and $b = 24$. This becomes evidence if we rewrite the right-hand side such that it represents the core part of a Beta PDF (see equation (2.6)). All that is missing is a normalizing constant which would make the area under the curve sum to one.

$$\theta^{83}(1 - \theta)^{23} = \theta^{84-1}(1 - \theta)^{24-1} \quad (2.15)$$

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^83 * (1 - theta)^23
}
(AUC <- integrate(PostFun, lower = 0, upper = 1)$value)
```

```
## [1] 8.32e-26
```

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 $B(84, 24)$. This term is in fact the integral we computed above.

$$p(\theta|n = 100, k = 80) = \frac{1}{B(84, 24)} \theta^{84-1} (1-\theta)^{24-1} \quad (2.16)$$

Now, this function will sum to one:

```
PostFun <- function(theta) {
  theta^83 * (1 - theta)^23 / AUC
}
round(integrate(PostFun, lower = 0, upper = 1)$value, 2)
```

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

2.2.4 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 $p(\theta|n, k) \sim Beta(84, 24)$. 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: 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 conjugate 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:

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

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

2.2.5 Visualizing the prior, likelihood, and the posterior

We established in the example above that the posterior is a Beta distribution with parameters $a = 84$, and $b = 24$. We visualize the likelihood, prior, and the posterior alongside each other in Figure 2.1.

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):

$$E[\hat{\theta}] = \frac{84}{84 + 24} = 0.78 \quad (2.18)$$

$$\text{var}[\hat{\theta}] = \frac{84 \cdot 24}{(84 + 24)^2(84 + 24 + 1)} = 0.0016 \quad (2.19)$$

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 = 84, shape2 = 24)
```

```
## [1] 0.695 0.851
```

Typically, we would summarize the results of a Bayesian analysis by displaying the posterior distribution of the parameter (or parameters) graph-

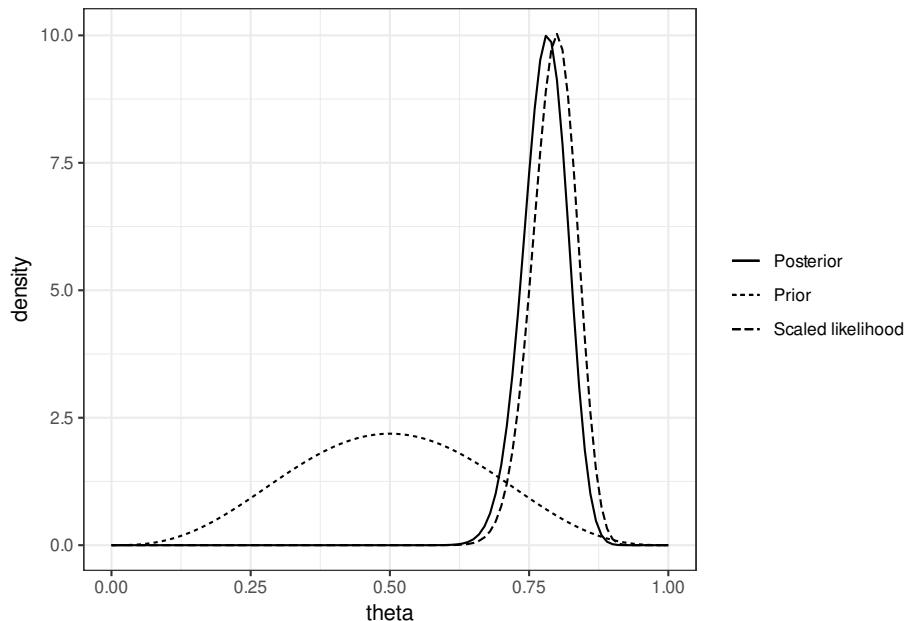


FIGURE 2.1: The (scaled) likelihood, prior, and posterior in the Beta-Binomial conjugate example. The likelihood is scaled to integrate to 1 to make it easier to compare to the prior and posterior distributions.

ically, 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.

2.2.6 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:

$$p(\theta | n, k) \propto p(k | n, \theta) p(\theta) \quad (2.20)$$

The four corresponding posterior distributions would be:

$$p(\theta | k, n) \propto [\theta^{80}(1 - \theta)^{20}][\theta^{2-1}(1 - \theta)^{2-1}] = \theta^{82-1}(1 - \theta)^{22-1} \quad (2.21)$$

$$p(\theta | k, n) \propto [\theta^{80}(1 - \theta)^{20}][\theta^{3-1}(1 - \theta)^{3-1}] = \theta^{83-1}(1 - \theta)^{23-1} \quad (2.22)$$

$$p(\theta | k, n) \propto [\theta^{80}(1 - \theta)^{20}][\theta^{6-1}(1 - \theta)^{6-1}] = \theta^{86-1}(1 - \theta)^{26-1} \quad (2.23)$$

$$p(\theta | k, n) \propto [\theta^{80}(1 - \theta)^{20}][\theta^{21-1}(1 - \theta)^{21-1}] = \theta^{101-1}(1 - \theta)^{41-1} \quad (2.24)$$

We can visualize each of these triplets of priors, likelihoods and posteriors; see Figure 2.2.

If you hold the likelihood function constant (held constant at $n = 100, k = 80$ in the above example), the tighter the prior, the greater the extent to which the posterior orients itself 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 *uncertainty* 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 100 to, say, 1000000. Suppose we still get a sample mean of 0.8 here, so that $k = 800000$. Now, the posterior mean will be influenced almost entirely by the sample mean. This is because, in the general form for the

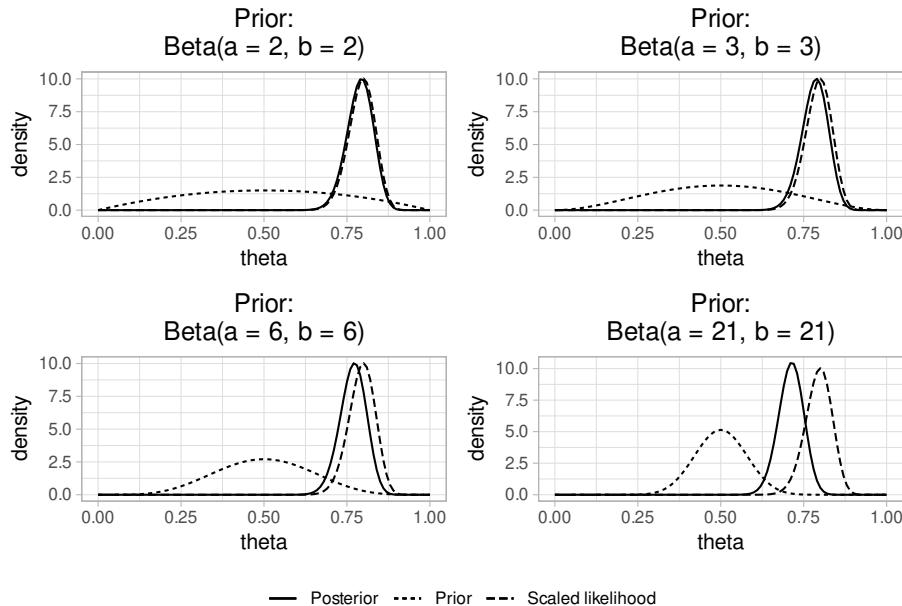


FIGURE 2.2: The (scaled) likelihood, prior, and posterior in the Beta-Binomial conjugate example, for different uncertainties in the prior. The likelihood is scaled to integrate to 1 to make its comparison easier.

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.

2.2.7 Incremental knowledge gain using prior knowledge

In the above example, we used an artificial example where we asked 100 subjects 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 80 instances of “umbrella”, and using a $Beta(4, 4)$ prior, we derived the posterior to be $Beta(84, 24)$.

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 100 subjects, and this time 60 produced “umbrella”. We could now use our new prior ($\text{Beta}(84, 24)$) to obtain an updated posterior. We have $a = 84$, $b = 24$, $n = 100$, $k = 60$. This gives us as posterior: $\text{Beta}(a + k, b + n - k) = \text{Beta}(84 + 60, 24 + 100 - 60) = \text{Beta}(144, 64)$.

Now, if we were to pool all our data that we have from the two experiments, then we would have as data $n = 200$, $k = 140$. Suppose that we keep our initial prior of $a = 4$, $b = 4$. Then, our posterior would be $\text{Beta}(4 + 140, 4 + 200 - 140) = \text{Beta}(144, 64)$. This is exactly the same posterior that we got when first analyzed the first 100 subjects’ data, derived the posterior, and then used that posterior as a prior for the next 100 subjects’ data.

This toy example illustrates an important point that has great practical importance 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, Engelmann, and Vasishth (2017) and Nicenboim, Roettger, and Vasishth (2018). Vasishth and Engelmann (2021) 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.

2.3 Summary

In this chapter, we learned 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.

2.4 Further reading

Accessible introductions to conjugate Bayesian analysis are Lynch (2007), and Lunn et al. (2012). Somewhat more demanding discussions of conjugate analysis are in Lee (2012), Carlin and Louis (2008), Christensen et al. (2011), O'Hagan and Forster (2004) and Bernardo and Smith (2009).

2.5 Exercises

Exercise 2.1. 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:

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

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.

Exercise 2.2. Conjugate forms 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 $\text{Binomial}(n, \theta)$ distribution. Given k successes in n trials coming from a Binomial distribution, we define a $\text{Beta}(a, b)$ prior on the parameter θ .

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

- Practical application

We ask 10 yes/no questions from a subject, and the subject returns 0 correct answers. We assume a Binomial likelihood function for these data. Also assume a $\text{Beta}(1, 1)$ 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.

Exercise 2.3. 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 $\text{Beta}(a, b)$, 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 $\sum_{i=1}^n x_i$.

Exercise 2.4. Conjugate forms 3

The Gamma distribution is defined in terms of the parameters a, b : $\text{Ga}(a, b)$. Given some data x , the probability density function is:

$$\text{Ga}(x|a, b) = \frac{b^a x^{a-1} \exp\{-bx\}}{\Gamma(a)} \quad (2.26)$$

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

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

It turns out that if we assume a $Ga(a,b)$ prior distribution and the above Exponential likelihood, the posterior distribution of λ is a Gamma distribution. In other words, the $Gamma(a,b)$ prior on the λ parameter in the Exponential distribution will be written:

$$Ga(\lambda|a, b) = \frac{b^a \lambda^{a-1} \exp\{-b\lambda\}}{\Gamma(a)} \quad (2.28)$$

Find the parameters a' and b' of the posterior distribution.

Exercise 2.5. Conjugate forms 4

- 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 particular time period (here, one day) can be modeled by a Poisson distribution:

$$f(x | \theta) = \frac{\exp(-\theta)\theta^x}{x!} \quad (2.29)$$

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.

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

[Hint: 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.]

- b) 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.

- 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.

Part II

Regression models with brms



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 section 2.2, 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. For more details about sampling algorithms, see the further readings suggested in section 3.7.

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. For now let’s assume that we used some probabilistic programming language to obtain 20000 samples from the posterior distribution of the cloze probability, θ : 0.759, 0.762, 0.805, 0.736, 0.867, 0.765, 0.8, 0.693, 0.815, 0.811, 0.777, 0.677, 0.799, 0.783, 0.721, 0.808, 0.818, 0.812, 0.689, 0.767, ... Figure 3.1 shows that the approximation of the posterior looks quite similar to the analytically derived posterior. The difference between the analytically computed and approximated mean and variance are 0.0002 and -0.000001 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, Wiecki, and Fonnesbeck 2016), `Turing` (Ge, Xu, and Ghahramani 2018), and `Stan` (Carpenter et al. 2017). These probabilistic programming 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 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 parametrize the models and to avoid convergence issues (these topics will be covered in detail later in this book).

¹The Python package `pymc3` and the Julia library `Turing` are recent exceptions since they are fully integrated into their respective languages.

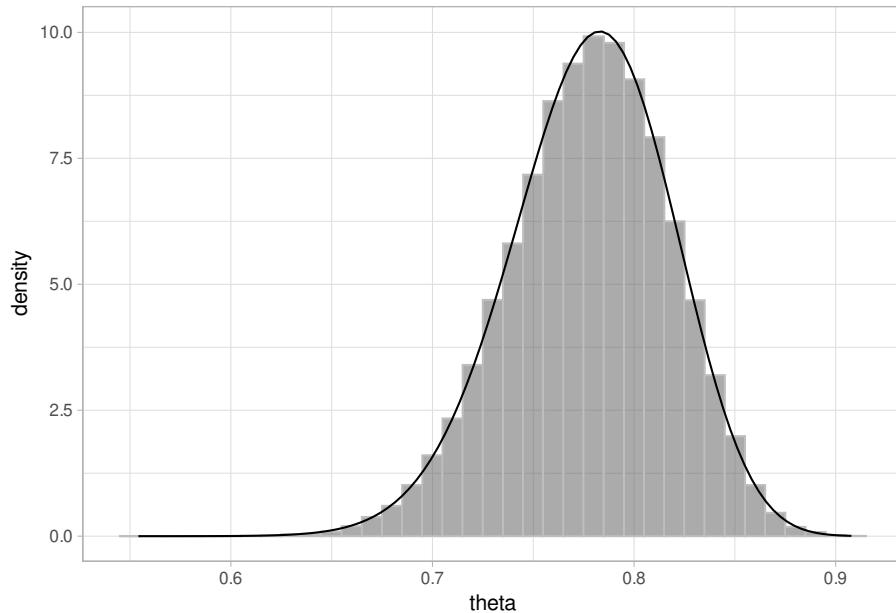


FIGURE 3.1: Histogram of the samples of θ from the posterior distribution generated via sampling. The black line shows the density plot of the analytically derived posterior.

There are some alternatives that allow Bayesian inference in R without having to fully specify the model “by hand”. The packages `rstanarm` (Goodrich et al. 2018) and `brms` (Bürkner 2019) provide Bayesian equivalents of many popular R model-fitting functions, such as `(gl)mer` (Bates, Mächler, et al. 2015a); both these packages use Stan for the back-end estimation and sampling. Another new alternative is JASP (JASP Team 2019), which 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 two parts 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 writ-

ing the models entirely in Stan. We revisit the models of this chapter and the following one in the introduction to Stan in chapter 10.

3.1.1.1 A simple linear model: A single subject 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 subject repeatedly pressing the space bar as fast as possible, without paying attention to any stimuli. The data are response times in milliseconds 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 (unknown) underlying time, μ ms, that the subject 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 response times are generally skewed; we will fix this assumption later).

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

$$rt_n \sim Normal(\mu, \sigma) \quad (3.2)$$

where $n = 1, \dots, N$, and rt is the dependent variable (response 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. This 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, $rt \sim 1$, 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 $\hat{\mu}$ will estimate the intercept of the model, in our case μ . If the reader is completely unfamiliar with linear models, the references in section 4.5 will be helpful.

For a Bayesian linear 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 (or 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: A flat (or very wide) prior will almost never be the best approximation of what we know. We will discuss prior specification in detail in chapter 6.

In this case, even if we know very little about the task, we know that pressing the spacebar will take at most a couple of seconds. We'll use flat priors in this section for pedagogical purposes; the next chapters will show more realistic uses of priors.

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

First, load the data frame `df_spacebar` from `bcogsci` package

```
data("df_spacebar")
df_spacebar
```

```
## # A tibble: 361 x 2
##       rt trial
##   <int> <int>
## 1    141     1
## 2    138     2
## 3    128     3
## # ... with 358 more rows
```

It is a good idea to plot 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.

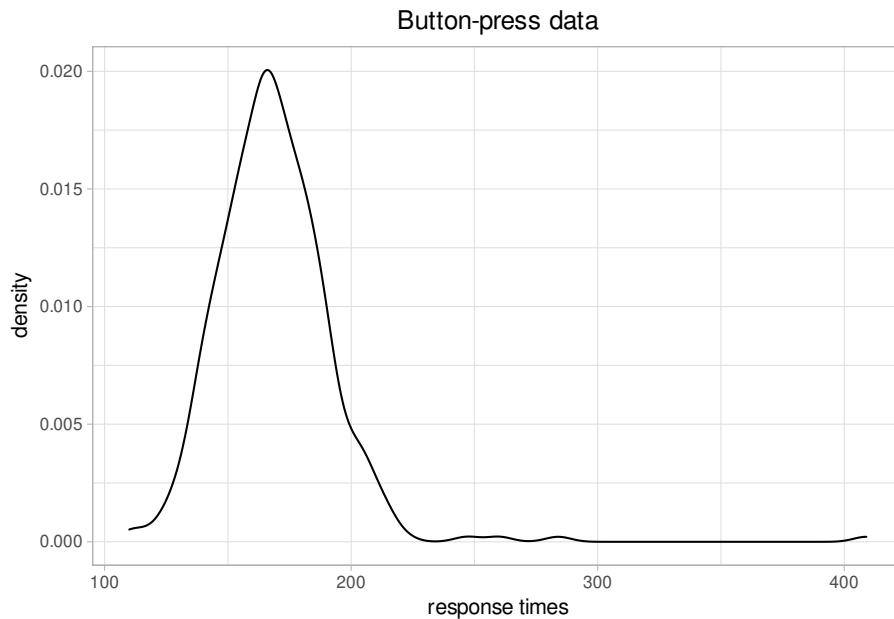


FIGURE 3.2: Visualizing the button-press data.

3.1.1.1.1 Specifying the model in `brms`

We'll fit the model defined by Equations (3.2) and (3.3) with `brms` in the following way; as we mentioned before, the uniform distribution is not appropriate, and we will ignore this warning *for now*.²

```
fit_press <- brm(rt ~ 1,
  data = df_spacebar,
  family = gaussian(),
  prior = c(
    prior(uniform(0, 60000), class = Intercept),
```

²The problem here is that although the parameter for the intercept is assigned a uniform distribution bounded between 0 and 60000 ms, the sampler might start sampling from an initial value outside this range. The sampler can start from an initial value that is outside the 0-60000 range because the initial value is chosen randomly (unless the user specifies an initial value explicitly). For now, it is enough to note that this prior is a bad idea for a model and it's shown here for pedagogical purposes only.

```
prior(uniform(0, 2000), class = sigma)
),
chains = 4,
iter = 2000,
warmup = 1000
)

## Warning: It appears as if you have specified an upper bounded prior on a parameter that has no na
## If this is really what you want, please specify argument 'ub' of 'set_prior' appropriately.
## Warning occurred for prior
## sigma ~ uniform(0, 2000)
```

The `brms` code has some differences from a model fit with `lm`. 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`. Other linking functions are possible, exactly as in the `glm` function. The default for `brms` is `gaussian()`.
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 priors 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 10, but here we explain the basic process.

3.1.1.1.2 Sampling and convergence in a nutshell

We start four chains independently from each other. Each chain “searches” for samples of the posterior distribution in a multidimensional space, where each parameter corresponds to a dimension. The shape of this space is determined by the priors and the likelihood. The chains start at random locations, and in each iteration they take one sample each. When sampling begins, the samples do not belong to the posterior distributions of the parameters. Eventually, the chains end up in the vicinity of the posterior distribution, and from that point onward the samples will belong to the posterior.

Thus, when sampling begins, the samples from the different chains can be far from each other, but *at some point* they will “converge” and start delivering samples from the posterior distributions. Although there are no guarantees that the number of iterations we run the chains for will be sufficient for obtaining samples from the posteriors, the default values of `brms` (and Stan) are in many cases sufficient to achieve convergence. When the default number of iterations do not suffice, `brms` (actually, Stan) will print out warnings, with suggestions for fixing the convergence problems. If all the chains converge to the same distribution, by removing the “warmup” samples, we make sure that we do not get samples from the initial path to the posterior distributions. The default in `brms` is that half of the total number of iterations in each chain (which default to 2000) will count as “warmup”. So, if one runs a model with four chains and the default number of iterations, we will obtain a total of 4000 samples from the four chains, after discarding the warmup iterations.

Figure 3.3 shows the path of the chains from the warmup phase onwards. Such plots are called trace plots or caterpillar plots. We show the warmup only for illustration purposes; generally, one should only inspect the chains after the point where we assume that convergence has been achieved (i.e., after the dashed line). The chains should look like a “fat hairy caterpillar”. Compare the trace plot of our model in Figure 3.3 with the trace plot of a model that did not converge, shown in Figure 3.4.

Trace plots are not always diagnostic as regards convergence. The trace plots might look fine, but the model may not have converged. Fortunately, Stan automatically runs several diagnostics with the information from the chains, and if there are no warnings after fitting the model and the trace plots look fine, we can be reasonably sure that the model converged, and assume that 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.

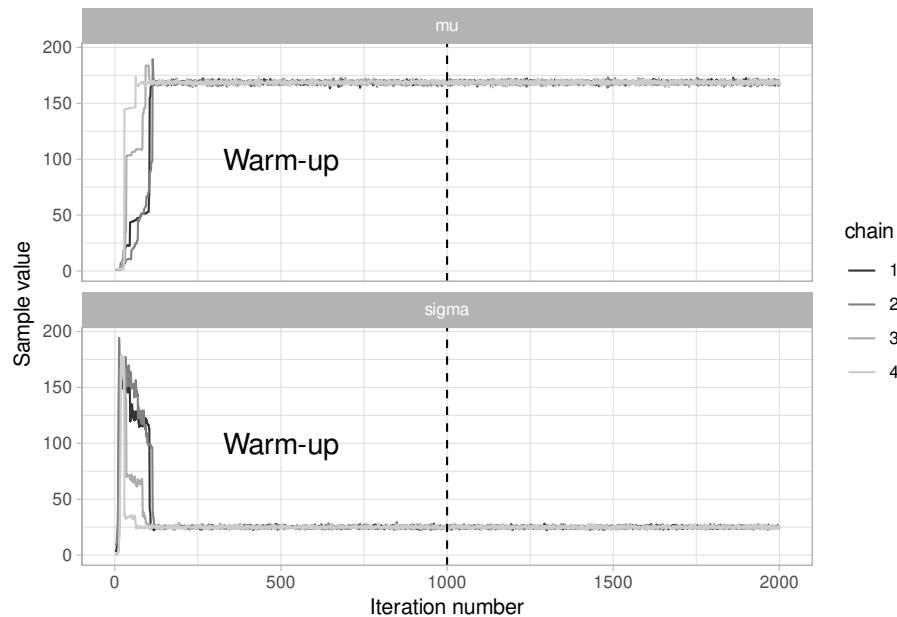


FIGURE 3.3: Trace plot of our `brms` model for the button-pressing data.

3.1.1.3 Output of `brms`

Once the model has been fit (and assuming that we got no warning messages about convergence problems), we can print out the samples of the posterior distributions of each of the parameters:

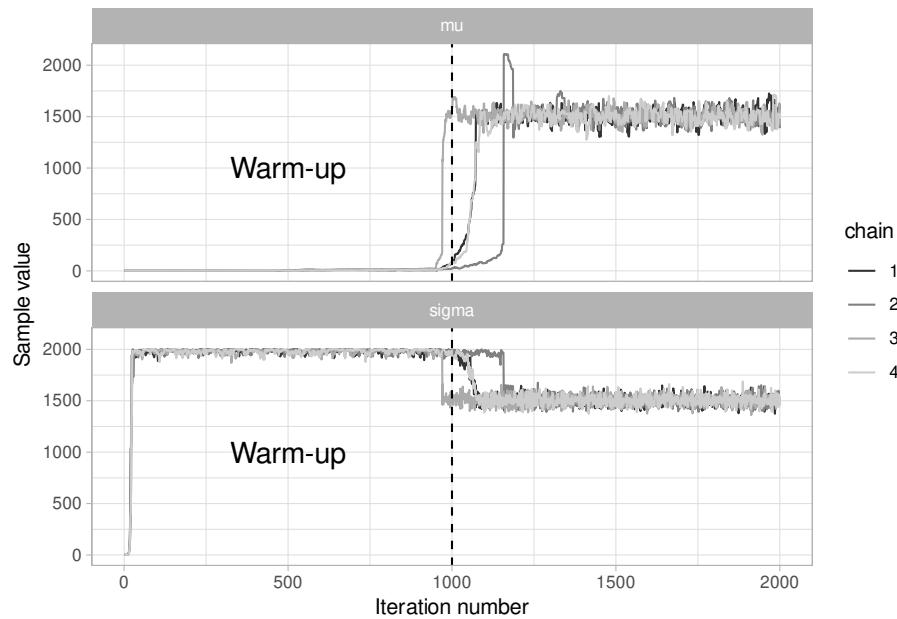


FIGURE 3.4: Trace plot of a model that **did not** converge. We can diagnose the non-convergence by the observing that the chains do not overlap—each chain seems to be sampling from a different distribution.

```
as_draws_df(fit_press)
```

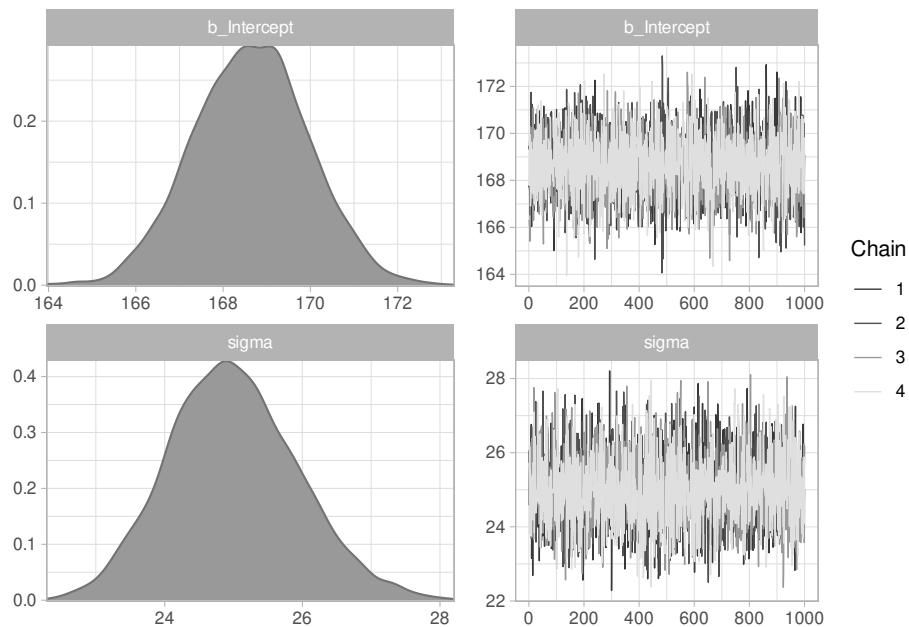
```
## # A draws_df: 1000 iterations, 4 chains, and 3 variables
##   b_Intercept sigma  lp__
## 1       168     25 -1688
## 2       168     25 -1688
## 3       168     26 -1689
## 4       167     25 -1689
## 5       169     26 -1688
## 6       169     24 -1688
## 7       170     25 -1688
## 8       168     26 -1689
## 9       167     27 -1690
## 10      166     23 -1692
## # ... with 3990 more draws
```

```
## # ... hidden reserved variables '.chain', '.iteration', '.draw'}
```

The term `b_Intercept` in the `brms` output corresponds to our μ , and lp is not really part of the posterior, it's the density of the unnormalized log posterior for each iteration. We will discuss lp later (in Box 10.1).

Plot the density and trace plot of each parameter after the warmup:

```
plot(fit_press)
```



`brms` provides a nice, if somewhat verbose, summary:

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

```
## Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: rt ~ 1
##   Data: df_spacebar (Number of observations: 361)
##   Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup draws = 4000
```

```

## 
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   168.64      1.31   166.06   171.17 1.00     2830     2456
## 
## Family Specific Parameters:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma      25.00      0.93   23.29   26.91 1.00     3583     2464
## 
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).

```

The Estimate is just the mean of the posterior sample, and the CIs mark the lower and upper bounds of the 95% credible intervals:

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

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

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

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

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

```
## 2.5% 97.5%
##    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:

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

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 $y_{pred_1}, \dots, y_{pred_N}$ from a data set \mathbf{y}_{pred} of length N , given a vector of priors Θ and our likelihood $p(\cdot|\Theta)$; (in our example, $\Theta = \langle \mu, \sigma \rangle$). The prior predictive density is written as follows:

$$\begin{aligned}p(\mathbf{y}_{pred}) &= p(y_{pred_1}, \dots, y_{pred_n}) \\ &= \int_{\Theta} p(y_{pred_1}|\Theta) \cdot p(y_{pred_2}|\Theta) \cdots p(y_{pred_N}|\Theta) p(\Theta) d\Theta\end{aligned}\tag{3.5}$$

In essence, we integrate out the vector of parameters, and we end up with the probability distribution of possible data sets 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.

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 into the probability density/mass function to generate a data set $y_{pred_1}, \dots, y_{pred_n}$.

Each sample is an imaginary or potential data set.

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 section 3.1.1.1. Although this approach works, it's quite slow (it takes about 4 seconds). See Box 3.1 for a

more efficient version of this function. We will see in section 3.5.3 that it's possible to use `brms` to sample from the priors, ignoring the `rt` in the data by setting `sample_prior = "only"`. However, since `brms` still depends on Stan's sampler, which uses Hamiltonian Monte Carlo, the prior sampling process can also fail to converge, especially when one uses very uninformative priors, as the ones in this example. In contrast, our function using `rnorm` cannot have convergence issues and will always produce independent samples.

```
N_samples <- 1000
N_obs <- nrow(df_spacebar)
mu_samples <- runif(N_samples, 0, 60000)
sigma_samples <- runif(N_samples, 0, 2000)
tic()
prior_pred <- normal_predictive_distribution(
  mu_samples = mu_samples,
  sigma_samples = sigma_samples,
  N_obs = N_obs
)
toc()
```

```
## 2.671 sec elapsed
```

```
prior_pred
```

```
## # A tibble: 361,000 x 3
##   trialn rt_pred  iter
##   <dbl>    <dbl> <dbl>
## 1       1 12044.     1
## 2       2  9652.     1
## 3       3  9234.     1
## # ... with 360,997 more rows
```

Box 3.1. A more efficient prior predictive distribution function

We can create a more efficient function in the following way using a `map_` function from the `purrr` package. With this function, we see an approximately 10-fold increase in speed. 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.

The `purrr` function `map2_dfr` (which works similarly to the base R function `lapply` and `Map`) essentially runs a for-loop, and builds a data frame with the output. It iterates over the values of two vectors (or lists) simultaneously, here, `mu_samples` and `sigma_samples` and, in each iteration, it applies a function to each value of the two vectors, here, `mu` and `sigma`. The output of each function is a data frame (or tibble in this case) with `N_obs` observations which is bound in a larger data frame at the end of the loop. Each of these data frames bound together represents an iteration in the simulation, and we identify the iterations by setting `.id = "iter"`.

Although this method for generating prior predictive distributions is a bit involved, it presents an advantage in comparison to the more straightforward use of `predict()` (or `posterior_predict()`, which can also generate prior predictions) together with setting `sample_prior = "only"` in the `brms` model (as we will do in section 3.5.3). Namely, here we don't depend on Stan's sampler, and that means that no matter the number of iterations in our simulation or how uninformative our priors are, we won't face convergence problems.

```

library(purrr)
# Define the function:
normal_predictive_distribution <- function(mu_samples,
                                             sigma_samples,
                                             N_obs) {
  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))
}
# Test it below:
tic()
prior_pred <- normal_predictive_distribution(
  mu_samples = mu_samples,
  sigma_samples = sigma_samples,
  N_obs = N_obs
)
toc()

## 0.619 sec elapsed

```

Figure 3.5 shows the first 18 samples of the prior predictive distribution (i.e., 18 predicted data sets) with the code below.

```

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

```

The prior predictive distribution in Figure 3.5 shows prior data sets that

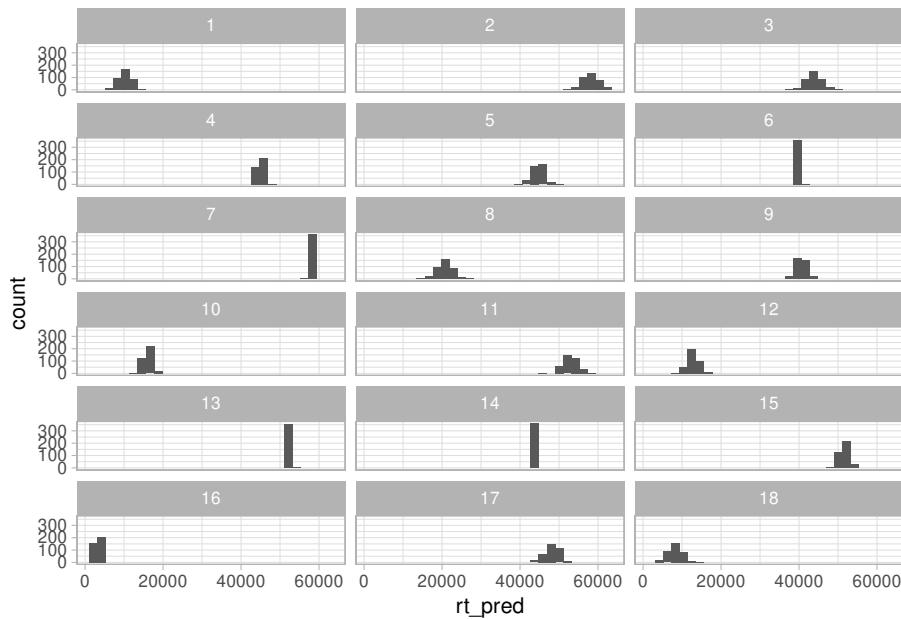


FIGURE 3.5: Eighteen samples from the prior predictive distribution of the model defined in section 3.1.1.1.

are not realistic: Besides the fact that the data sets show that response times distributions are symmetrical—and we know that they are generally right-skewed—some data sets present response times that are unrealistically long. Worse yet, if we inspect enough samples from the prior predicted data, we will find that a few data sets have negative press time values.

We can also look at the distribution of summary statistics in the prior predictive data. 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. For example, in the button-pressing example, it seems reasonable to assume that response time average are between 200-600 ms, response times are very unlikely to be below 50 ms (given the delays in keyboards), and even long lapses of attention won't be over

a couple of seconds.³ This distribution of summary statistics is shown in Figure 3.6.

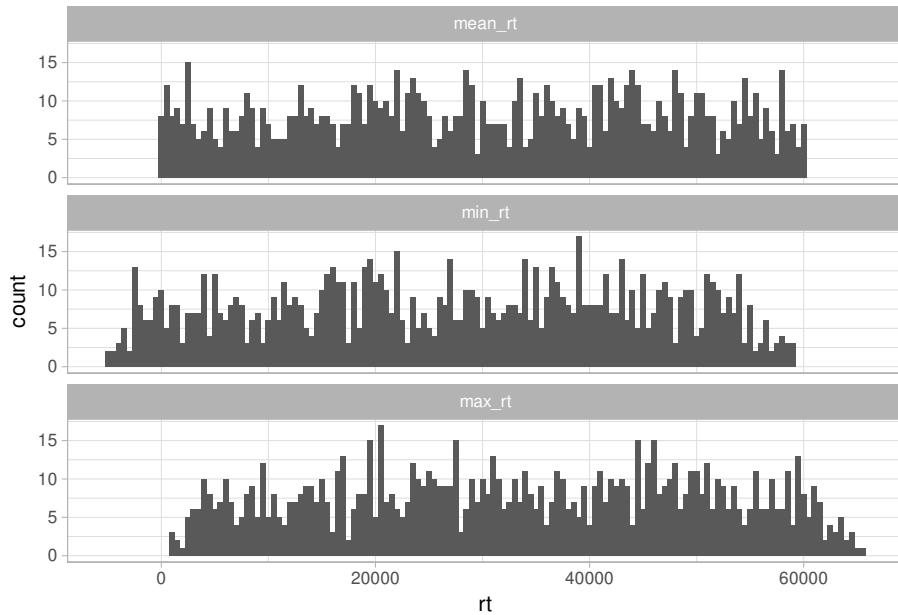


FIGURE 3.6: Prior predictive distribution of mean, minimum, and maximum value of the button-pressing model defined in section 3.1.1.1.

Figure 3.6 shows 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 ms is equally likely, even though we know that a value close to 0 or to 60000 ms would be extremely surprising. It should be clear that this 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, Simpson, and Betancourt 2017), and our likelihood is a normal distribution, which will allow for negative samples no matter the value of the parameter μ .

³We'll see later how to generate prior predictive distributions of statistics such as mean, minimum, or maximum value in section 3.5.3 using `brms` and `pp_check`.

To summarize the above discussion, our priors are clearly not very realistic given what we know about response 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. In the Bayesian community, there is no fixed nomenclature for classifying different kinds of priors. For this book, we have chosen specific names for each type of prior, but this is just a convention that we follow for consistency. There are also other classes of prior that we do not discuss in this book. An example is improper priors such as $\text{Uniform}(-\infty, +\infty)$, which are not proper probability distributions because the area under the curve does not sum to 1.

When thinking about priors, the reader should not get hung up on what precisely the name is for a particular type of prior; they should rather focus on what that prior means in the context of the research problem.

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 usually do have some minimal information about our parameters of interest. At the very least, we know the order of magnitude (response times will be in milliseconds and not days, EEG signals some microvolts and not volts, etc.). Third, uninformative priors make the sampling slower and might lead to convergence problems. Unless we have a large amount of data, it would be wise to avoid them. Fourth, it is not always clear to which parametrization of a given distri-

bution we should assign the flat priors. For example, the Normal distribution is sometimes defined based on its standard deviation (σ), variance (σ^2), or precision ($1/\sigma^2$): a flat prior for the standard deviation is not flat for the precision of the distribution. Although it is sometimes possible to find an uninformative prior that is univariant under a change of parameters (also called Jeffreys priors; Jaynes 2003, sec 6.15; Jeffreys 1939, Chap 3), it's not always the case. Finally, if we want to compute Bayes factors, uninformative priors can lead to very misleading conclusions (chapter 16).

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 *regularizing priors*. These are priors that down-weight extreme values (that is, they provide regularization), they are usually not very informative, and mostly let the likelihood dominate in determining the posteriors. These priors are theory-neutral; that is, they usually do not bias the parameters to values supported by any prior belief or theory. The idea behind this type of prior is to help to stabilize computation. These priors are sometimes called *weakly informative* or *mildly informative* priors. For many applications, they perform well, but as we will see later in chapter 16, 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 what our data do and do not look like, we can build priors that truly reflect the properties of potential data sets, using prior predictive checks. In this book, we will encounter many examples of this class of priors.

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 subjects.

These four options constitute a continuum. The uniform prior from the last model (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. Informative priors, in the sense defined above, will be used only relatively rarely; they become more important to consider when doing Bayes factor analyses (chapter 16).

3.4 Revisiting the button-pressing example with different priors

What would happen if we use even wider priors for the model defined previously (in section 3.1.1.1)? We could assume that every mean between -10^{10} and 10^{10} ms is equally likely. This prior is clearly unrealistic and even illogical, we are not expecting negative response times! 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. As before, we'll get a warning from `brms`. We'll ignore it for now.

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

```
# 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_spacebar,
  family = gaussian(),
  prior = c(
    prior(uniform(-10^10, 10^10), class = Intercept),
    prior(uniform(0, 10^10), class = sigma)
```

```

)
)

## Warning: It appears as if you have specified an upper bounded prior on a parameter that has no na
## If this is really what you want, please specify argument 'ub' of 'set_prior' appropriately.
## Warning occurred for prior
## sigma ~ uniform(0, 10^10)
```

Even with these extremely unrealistic priors the output of the model is virtually identical to the previous one (see Figure 3.7)!

```

fit_press_unif

## ...
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   168.64      1.34   166.02   171.20 1.01     3570     2531
## 
## Family Specific Parameters:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma      25.02      0.94    23.28    26.98 1.00     3522     2750
## 
## ...
```

What would happen if we had used very informative priors? We will assume that mean values very close to 400 ms are the most likely, and that the standard deviation of the response times is very close to 100. We know that this information is clearly wrong, because we have already seen the output of some models. The $Normal_+$ notation indicates a normal distribution truncated in zero that only allows positive values (Box 4.1 discusses this type of distribution in detail):

$$\begin{aligned}\mu &\sim Normal(400, 10) \\ \sigma &\sim Normal_+(100, 10)\end{aligned}\tag{3.7}$$

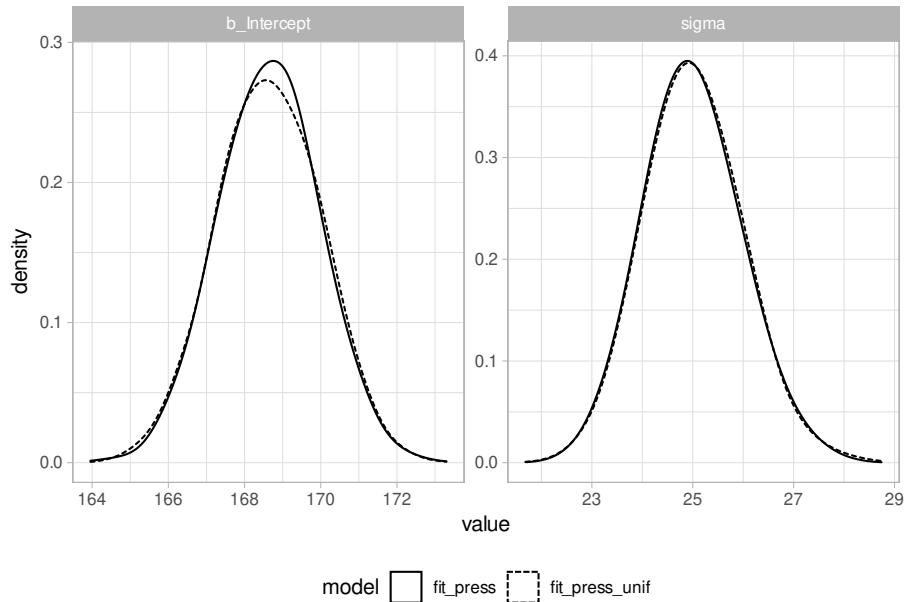


FIGURE 3.7: Comparison of the posterior distributions from the model with extremely unrealistic priors, `fit_press_unif`, against the previous model with more “realistically” bounded uniform distributions (but still not recommended), `fit_press`.

```
fit_press_inf <- brm(rt ~ 1,
  data = df_spacebar,
  family = gaussian(),
  prior = c(
    prior(normal(400, 10), class = Intercept),
    # brms knows that SDs need to be bounded
    # to exclude values below 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_inf

## ...
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept    172.99     1.42   170.31   175.90 1.00     2729     2389
##
## Family Specific Parameters:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma      26.13     1.05   24.15   28.32 1.00     2678     2424
##
## ...
```

Finally, we'll choose (relatively) principled priors. For this we rely on our experience with previous similar experiments. We expect that the mean reaction time would be around 200 ms, but we have a large degree of uncertainty regarding that value. For that reason we decide on a relatively wide prior, $Normal(200, 100)$. Given that we only have one subject and the task is very simple, we don't expect a large standard deviation: We settle on a location of 50 ms for a truncated normal distribution, but we allow for relatively large uncertainty with $Normal_+(50, 50)$:

$$\begin{aligned}\mu &\sim Normal(200, 100) \\ \sigma &\sim Normal_+(50, 50)\end{aligned}\tag{3.8}$$

How do we know that these priors are principled? This largely depends on our domain knowledge; we will return to the use of domain knowledge in prior specification in chapter 6. But we can achieve a better understanding of what our priors imply by visualizing the priors by plotting them, and carrying out prior predictive checks. We skip these steps here, but revisit these issues in chapters 6 and 7. In these chapters, we give more detailed information about choosing priors and a suggested workflow.

```
fit_press_reg <- brm(rt ~ 1,
  data = df_spacebar,
  family = gaussian(),
```

```

prior = c(
  prior(normal(200, 100), class = Intercept),
  prior(normal(50, 50), class = sigma)
)
)

```

Our new estimates are virtually the same as before:

```

fit_press_reg

## ...
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   168.61     1.32   165.96   171.14 1.00    3705    2488
## 
## Family Specific Parameters:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma      25.00     0.94    23.29    26.90 1.00    3644    2988
## 
## ...

```

The above examples of using different priors should not be misunderstood to mean that priors never matter. When there is enough data, the likelihood will dominate in determining the posterior distributions. What constitutes “enough” data is also a function of the complexity of the model; as a general rule, more complex models require more data. For example, parameters that represent group-level (also known as random) effects in hierarchical (or mixed) models, require much more data than population-level (also known as fixed) effects. This means that, in general, priors will have a greater impact on the estimation of group-level parameters (also known as random effects) than of population-level parameters. An important caveat here is that the posterior uncertainty of the population-level parameters will also be affected by the uncertainty in the group-level parameters. In other words, if the posterior estimates for the random effects have wide uncertainty, then the posteriors of the fixed effects will be affected by that uncertainty.

Even in cases where there is enough data and the likelihood dominates in determining the posteriors, regularizing, principled priors (i.e., priors that are more consistent with our a priori beliefs about the data) will in general speed-up model convergence.

In order to determine the extent to which the posterior is influenced by our priors, it is a good practice to carry out 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 examples from psycholinguistics, see Vasishth et al. 2013; Vasishth and Engelmann 2021). We will see later (chapter 16) that sensitivity analysis becomes crucial for reporting Bayes factors; 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 data sets 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 gives us some indication of what future data might look like, given the data and model of course.

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

$$p(y_{pred} | y) = \int_{\Theta} p(y_{pred}, \Theta | y) d\Theta = \int_{\Theta} p(y_{pred} | \Theta, y) p(\Theta | y) d\Theta \quad (3.9)$$

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

$$p(y_{pred} | y) = \int_{\Theta} p(y_{pred} | \Theta) p(\Theta | y) d\Theta \quad (3.10)$$

In Equation (3.10), we are conditioning y_{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 y_{pred} given our maximum likelihood estimate of Θ (a point value). As with the prior predictive distribution, we can avoid performing the integration explicitly by generating samples from the posterior predictive distribution. We can use the same function that we created before, `normal_predictive_distribution`, 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_spacebar)
mu_samples <- as_draws_df(fit_press)$b_Intercept
sigma_samples <- as_draws_df(fit_press)$sigma
normal_predictive_distribution(
  mu_samples = mu_samples,
  sigma_samples = sigma_samples,
  N_obs = N_obs
)

## # A tibble: 1,444,000 x 3
##       iter trialn rt_pred
##   <dbl>   <int>   <dbl>
## 1     1      1    171.
## 2     1      2    188.
## 3     1      3    166.
## # ... with 1,443,997 more rows
```

The `brms` function `posterior_predict()` is a convenient function that gives us samples from the posterior predictive distribution. If we use `posterior_predict(fit_press)`, we obtain the predicted response times in a matrix, with the samples as rows and the observations (data-points) as columns. (Bear in mind that if we fit a model with `sample_prior = "only"`, the dependent variable is ignored and `posterior_predict` will give us samples from the prior predictive distribution).

We can use the posterior predictive distribution to examine the “descrip-

tive 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). For this reason, comparing the descriptive adequacy of different models is not enough to differentiate between their relative performance. When doing model comparison, it is important to consider the criteria that Roberts and Pashler (2000) define. Although Roberts and Pashler (2000) are more interested in process models and not necessarily Bayesian models, their criteria are important for any kind of model comparison. Their main point is that it is not enough to have a good fit to the data for a model to be convincing. One should check that the range of predictions that the model makes is reasonably constrained; if a model can capture any possible outcome, then the model fit to a particular data set is not so informative. Thus, although posterior predictive checking is important, it is only a sanity check to assess whether the model behavior is reasonable.

In many cases, we can simply use the plot functions from `brms` (that act as wrappers for `bayesplot` functions). The plotting function `pp_check`, for example, takes as arguments the model, the number of predicted data sets, and the type of visualization, and it can show us different visualizations of posterior predictive checks. In these type of plots, the observed data are plotted as y and predicted data as y_{rep} . Below, we use `pp_check` to investigate how well the observed distribution of response times fit our model based on some number (11 and 100) of samples of the posterior predictive distributions (that is, simulated data sets); see Figures 3.8 and 3.9.

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

```
pp_check(fit_press, ndraws = 100, type = "dens_overlay")
```

The data is slightly skewed and has no values shorter than 100 ms, while the predictive distributions are centered and symmetrical; see figures 3.8

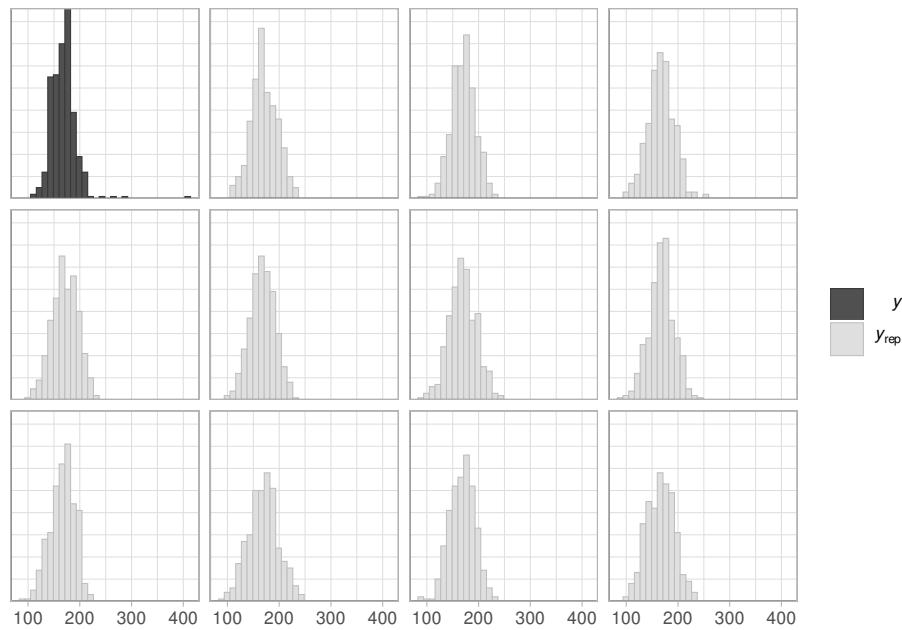


FIGURE 3.8: Histograms of eleven samples from the posterior predictive distribution of the model `fit_press` (y_{rep}).

and 3.9. 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

Response times are not usually normally distributed. A more realistic distribution is the log-normal. A random variable (such as time) that is log-normally distributed takes only positive real values and is right-skewed. Although other distributions can also produce data with such properties, the log-normal will turn out to be a pretty reasonable distribution for response times.

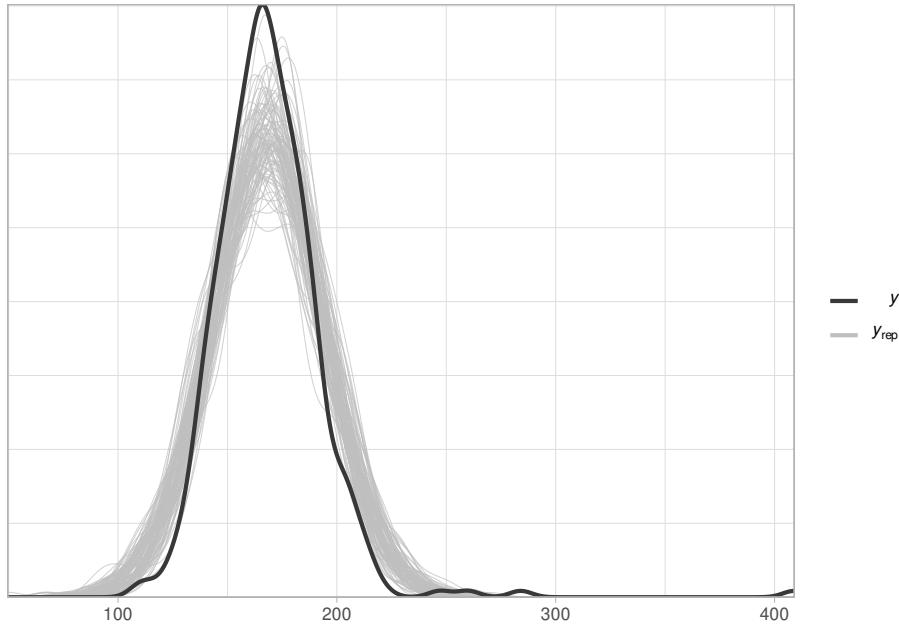


FIGURE 3.9: Posterior predictive check that shows the fit of the model `fit_press` in comparison to data sets from the posterior predictive distribution using an overlay of density plots.

3.5.2 The log-normal likelihood

If y is log-normally distributed, this means that $\log(y)$ is normally distributed.⁴ The log-normal distribution is also defined using μ and σ , but these are on the log ms scale; they correspond to the mean and standard deviation of the logarithm of the data y , $\log(y)$, which will be normally distributed. 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 . Equation (3.11) shows the relationship between the log-normal and the normal.

$$\begin{aligned} \log(y) &\sim \text{Normal}(\mu, \sigma) \\ y &\sim \text{LogNormal}(\mu, \sigma) \end{aligned} \tag{3.11}$$

We can obtain samples from the log-normal distribution, using the nor-

⁴More precisely, $\log_e(y)$ or $\ln(y)$, but we'll write it as just $\log()$.

mal distribution by first setting an auxiliary variable, z , so that $z = \log(y)$. This means that $z \sim \text{Normal}(\mu, \sigma)$. Then we can just use $\exp(z)$ as samples from the $\text{LogNormal}(\mu, \sigma)$, since $\exp(z) = \exp(\log(y)) = y$. The code below produces Figure 3.10.

```
mu <- 6
sigma <- 0.5
N <- 500000
# Generate N random samples from a log-normal distribution
sl <- rlnorm(N, mu, sigma)
ggplot(tibble(samples = sl), aes(samples)) +
  geom_histogram(aes(y = ..density..), binwidth = 50) +
  ggtitle("Log-normal distribution\n") +
  coord_cartesian(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(samples)) +
  geom_histogram(aes(y = ..density..), binwidth = 50) +
  ggtitle("Exponentiated samples from\na normal distribution") +
  coord_cartesian(xlim = c(0, 2000))
```

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

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

$$rt_n \sim \text{LogNormal}(\mu, \sigma) \quad (3.12)$$

But now the scale of our priors needs to change! We'll continue with uniform priors for ease of exposition, even though, as we mentioned earlier, these are not really appropriate here. (Below, we show these uniform priors, as well as examples of more realistic and useful priors.)

$$\begin{aligned} \mu &\sim \text{Uniform}(0, 11) \\ \sigma &\sim \text{Uniform}(0, 1) \end{aligned} \quad (3.13)$$

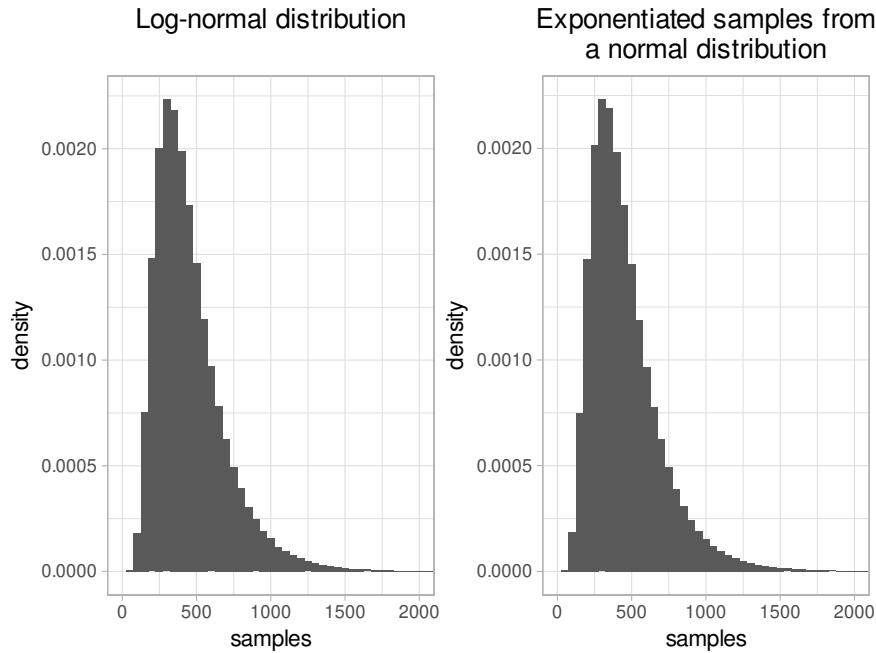


FIGURE 3.10: Two log-normal distributions with the same parameters generated by either generating samples from a log-normal distribution or exponentiating samples from a normal distribution.

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 $\log(\text{milliseconds})$.
- *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 $scale \sigma$ will be $\exp(\mu + \sigma^2/2) \times \sqrt{\exp(\sigma^2) - 1}$. Unlike the normal distribution, the spread of the log-normal distribution depends on both μ and σ .

To understand the meaning of our priors on the millisecond scale, we need to take into account both the priors and the likelihood. We can do this by generating a prior predictive distribution. We can just exponentiate the samples produced by `normal_predictive_distribution()` (or, alternatively, we could have edited the function and replaced `rnorm` with `rlnorm`).

```
N_samples <- 1000
N_obs <- nrow(df_spacebar)
mu_samples <- runif(N_samples, 0, 11)
sigma_samples <- runif(N_samples, 0, 1)
prior_pred_ln <- normal_predictive_distribution(
  mu_samples = mu_samples,
  sigma_samples = sigma_samples,
  N_obs = N_obs
) %>%
  mutate(rt_pred = exp(rt_pred))
```

Next, plot the distribution of some representative statistics; see Figure 3.11.

We cannot generate negative values any more, since $\exp(\text{any finite real number}) > 0$. These priors might work in the sense that the model might converge; but we can choose better, regularizing priors for our model, such as the following:

$$\begin{aligned}\mu &\sim \text{Normal}(6, 1.5) \\ \sigma &\sim \text{Normal}_+(0, 1)\end{aligned}\tag{3.14}$$

The prior for σ here is a truncated distribution, and although its location is zero, this is not its mean. We can calculate its approximate mean from a large number of random samples of the prior distribution using the function `rtnorm` from the package `extraDistr`. In this function, we have to set the parameter `a = 0` to express the fact that the normal distribution is truncated from the left at 0.

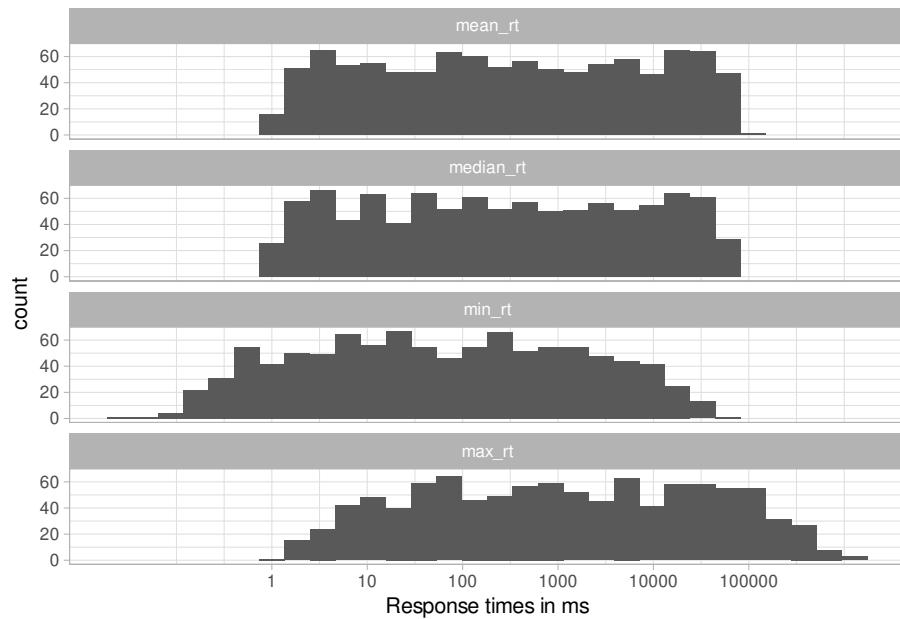


FIGURE 3.11: Prior predictive distribution of mean, median, minimum, and maximum value of the log-normal model with priors defined in Equation (3.13). The x-axis is log-transformed.

```
mean(rtnorm(100000, 0, 1, a = 0))
```

```
## [1] 0.797
```

Although μ can be negative, the dependent variable can't be, since the exponent of a finite, negative real number, $\exp(\text{some finite negative real value})$, 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
## 20.1 8103.1

```

This means that our prior for μ is still not too informative (these are medians; the actual values generated by the log-normal distribution can be much more spread out). We can now plot the distribution of some representative statistics of the prior predictive distributions. We use `brms` to sample from the priors ignoring the `rt` data, by setting `sample_prior = "only"`.

If we want to use `brms` to generate prior predictive data in this manner before collecting the data, we do need to have *some* simulated values that represent the vector of dependent variables, `rt`. Because these values will be plotted alongside with the prior predictive distributions in `pp_check`, they can be used to compare the prior predictive data with the simulated `rt` vector. In this case, as an example, we choose $Uniform(0, 10000)$ to generate the simulated `rt` vector. We need to specify that the family is `lognormal()`. In our first example, we had used the family `gaussian()`.

```

df_spacebar_ref <- df_spacebar %>%
  mutate(rt = runif(n(), 0, 10000))
fit_prior_press_ln <- brm(rt ~ 1,
  data = df_spacebar_ref,
  family = lognormal(),
  prior = c(
    prior(normal(6, 1.5), class = Intercept),
    prior(normal(0, 1), class = sigma)
  ),
  sample_prior = "only",
  control = list(adapt_delta = .9)
)

```

```
## Warning: There were 1 divergent transitions after warmup. See
```

```
##      http://mc-stan.org/misc/warnings.html#divergent-transitions-
## after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: Examine the pairs() plot to diagnose sampling problems
```

We need to increase the `adapt_delta` parameter's default value from 0.8 to 0.9 to simulate the data avoiding warnings. Since Stan samples from the prior distributions in the same way that it samples from the posterior distribution, we should not ignore warnings to ensure that the model converged. In that respect, our custom function `normal_predictive_distribution()` defined in 3.2 has the advantage that, since it just relays on the `rnorm` function, it will always yield independent samples from the prior distribution and cannot fail to converge.

Plot the prior predictive distribution of means with the following code (the figure is not produced here, to conserve space):

```
pp_check(fit_prior_press_ln, type = "stat", stat = "mean") +
  coord_cartesian(xlim = c(0.001, 300000)) +
  scale_x_continuous("Response times [ms]",
    trans = "log",
    breaks = c(0.001, 1, 100, 1000, 10000, 100000),
    labels = c(
      "0.001", "1", "100", "1000", "10000",
      "100000"
    )
  ) +
  ggtitle("Prior predictive distribution of means")
```

To plot the distribution of minimum, and maximum values, we replace `mean` for `min`, and `max` respectively. The three statistics are displayed in Figure 3.12.

We see that the priors that we are using are still quite uninformative. The tails of the prior predictive distributions that correspond to our normal priors shown in Figure 3.12 are even further to the right, reaching more extreme values than for the prior predictive distributions generated by uniform priors (shown in Figure 3.11). Our new priors are still far from

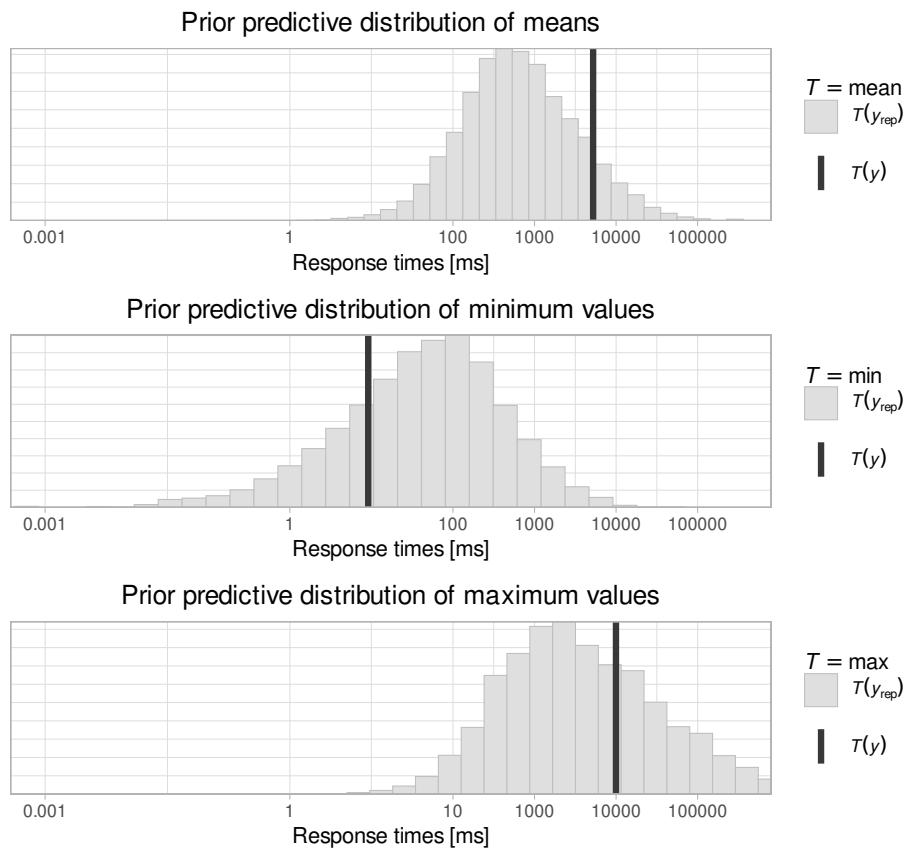


FIGURE 3.12: Prior predictive distribution of mean, maximum, and minimum values of the log-normal model with priors defined in Equation (3.14). The distributions of mean, minimum, and maximum values of the prior predictive distributions are labeled y_{rep} and the mean, minimum, and maximum values for the “reference” distribution, $\text{Uniform}(0, 10000)$, are labeled y . The x-axis is log-transformed.

perfectly encoding our prior knowledge. We could do more iterations of choosing priors and generating prior predictive distributions until we have priors that generate realistic data. However, given that the bulk of the distributions of mean, maximum, minimum values lie roughly in the correct order of magnitude, these priors are going to be acceptable. In general, we can use summary statistics (e.g., mean, median, min, max) to test whether the priors are in a plausible range. We can do this by defining, for the particular research problem we are studying, the extreme data that would be very implausible to ever observe (e.g., reading times at a word larger than one minute) and choosing priors such that such extreme response times occur only very rarely in the prior predictive distribution.

We can fit the model now; recall that both the distribution family and prior change in comparison to our previous example.

```
fit_press_ln <- brm(rt ~ 1,
  data = df_spacebar,
  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

## ...
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     5.12      0.01     5.10     5.13 1.00     3208     2371
## 
## Family Specific Parameters:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma       0.13      0.01     0.13     0.15 1.00     2748     2699
```

```
##  
## ...
```

If we want to know how long it takes 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(as_draws_df(fit_press_ln)$b_Intercept)
```

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

```
c(mean = mean(estimate_ms), quantile(estimate_ms, probs = c(.025, .975)))  
  
##   mean   2.5% 97.5%  
##   167   165   169
```

We can now check whether our predicted data sets look similar to the real data set. See Figure 3.13; compare this with the earlier Figure 3.9.

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

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

According to Figure 3.13, 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 data sets generated by different models and compare them to the observed statistics. We suspect that the normal distribution would generate response 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. We do this with `pp_check`, by using as stat either "min"

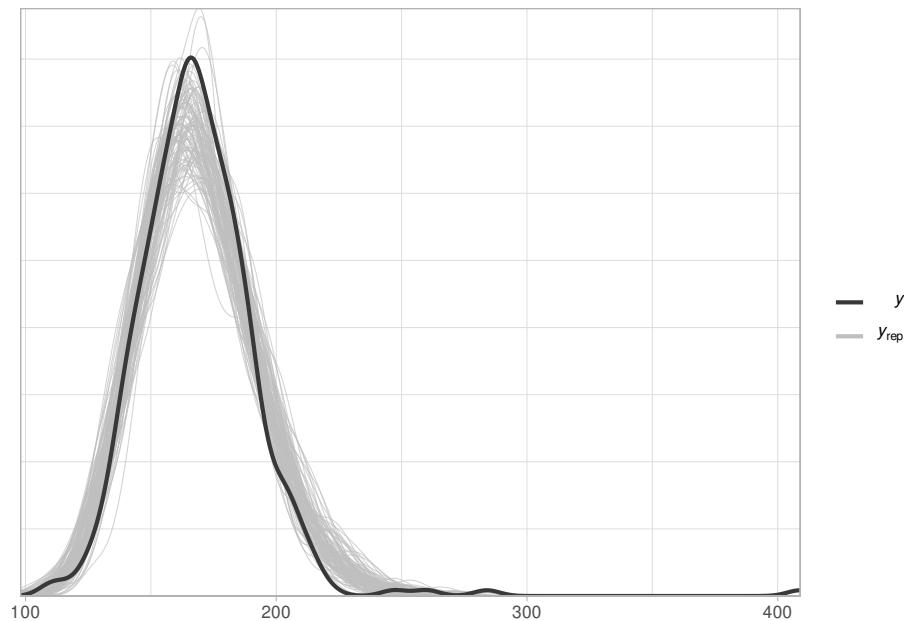


FIGURE 3.13: Posterior predictive distribution of `fit_noreading_ln`.

or "max" for both `fit_press`, and `fit_press_ln`; an example is shown below. The plots are shown in Figures 3.14 and 3.15.

```
pp_check(fit_press, type = "stat", stat = "min")
```

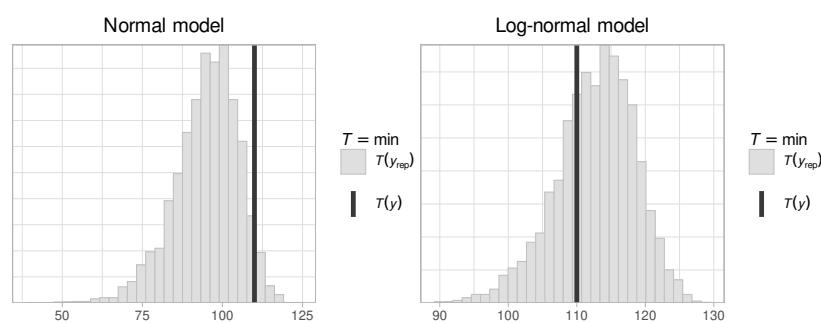


FIGURE 3.14: Distributions of minimum values in a posterior predictive check, using the normal and log-normal probability density functions. The minimum in the data is 110 ms.

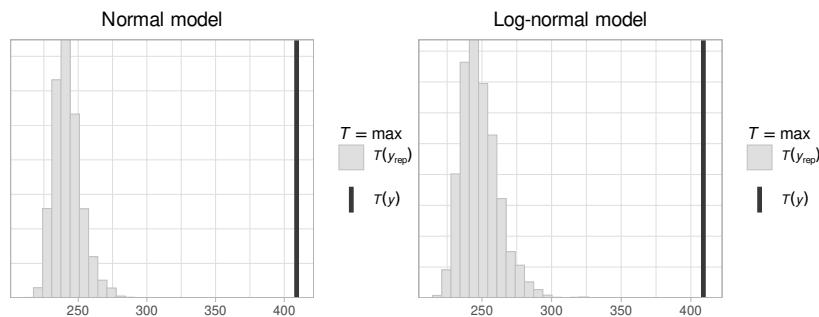


FIGURE 3.15: Distribution of maximum values in a posterior predictive check. The maximum in the data is 409 ms.

Figure 3.14 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 the normal one. Figure 3.15 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 response times are being generated by lapses of attention. This would mean that two probability distributions are mixed here; modeling this process involves more complex tools that we will treat in chapter 20.

This completes our introduction to `brms`. We are now ready to learn about more regression models.

3.6 Summary

In this chapter, we learned how to fit and interpret a Bayesian model with a normal likelihood. We looked at the effect of priors by means of prior predictive distributions and sensitivity analysis. We also looked at the fit of the posterior, by inspecting the posterior predictive distribution (which gives us some idea about the descriptive adequacy of the model). Furthermore, we learned how to fit a Bayesian model with a log-normal likelihood, and how to compare the predictive accuracy of different models.

3.7 Further reading

Sampling algorithms are discussed in detail in Gamerman and Lopes (2006). The sections on sampling from the short open-source book by Bob Carpenter, *Probability and Statistics: a simulation-based introduction* (<https://github.com/bob-carpenter/prob-stats>), and the sections on sampling algorithms in Lambert (2018) and Lynch (2007) are also very helpful.

3.8 Exercises

Exercise 3.1. A simple linear model.

- a. Fit the model `fit_press` with just a few iterations, say 50 iterations. What happens?
- b. Using normal distributions, choose priors that represent better **your** assumptions about response times.

Exercise 3.2. Revisiting the button-pressing example with different priors.

- a. Can you come up with very informative priors that bias the posterior in a noticeable way (use normal distributions for priors, not uniform priors)?
- b. Generate and plot prior predictive distributions based on this prior and plot them.
- c. Generate posterior predictive distributions based on this prior and plot them.

Exercise 3.3. Posterior predictive checks with a log-normal model.

- a. For the log-normal model `fit_press_ln`, change the prior of σ so that it is a log-normal distribution with location (μ) of -2 and scale (σ) of .5. What does such a prior imply about your belief regarding button-pressing times in milliseconds? Is it a good prior? Generate and plot prior predictive distributions. Do the

- new estimates change compared to earlier models when you fit the model?
- 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 response times in milliseconds?

Exercise 3.4. A skew normal distribution.

Would it make sense to use a “skew normal distribution” instead of the lognormal? The skew normal distribution has three parameters location ξ , scale ω , and shape α . The distribution is right skewed if $\alpha > 0$, is left skewed if $\alpha < 0$, and is identical to the regular normal distribution if $\alpha = 0$. For fitting this in `brms`, one needs to change `family` and set it to `skew_normal()`, and add a prior of `class = alpha` (location remains `class = Intercept` and scale, `class = sigma`).

- Fit this model with a prior that assigns approximately 95% of the prior probability mass of `alpha` to be between 0 and 10.
- Generate posterior predictive distributions and compare the posterior distribution of summary statistics of the skew normal with the normal and log-normal

4

Bayesian regression models

We generally run experiments because we are interested in the relationship between two or more variables. A regression will tell us how our *dependent variable*, also called the *response* or *outcome variable* (e.g., pupil size, response 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 (e.g., age). In this chapter we focus on simple regression models with different likelihood functions.

4.1 A first linear regression: 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 from one subject's pupil size of the control experiment by Wahn et al. (2016), averaged by trial. The data are available from `df_pupil` in the package `bcogsci`. In this experiment, a subject covertly tracked between zero and five objects among several randomly moving objects on a computer screen. This task is called multiple object tracking (or MOT; see Pylyshyn and Storm 1988). First, several objects appear on 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 subject need to indicate which objects were the targets. See Fig-

ure 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.

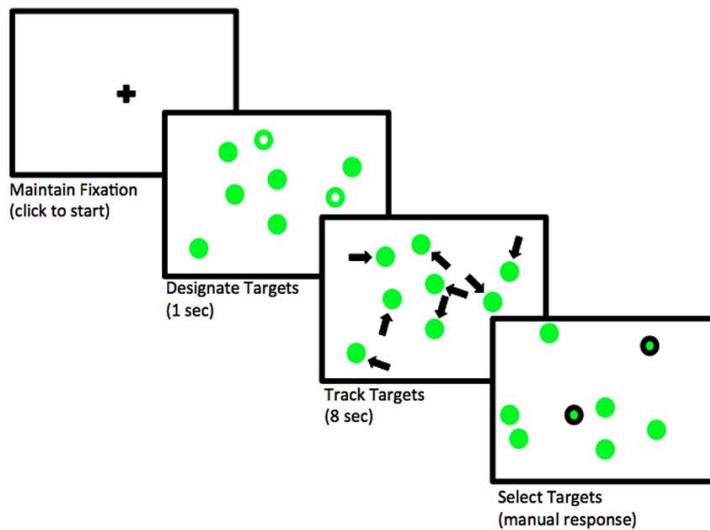


FIGURE 4.1: Flow of events in a trial where two objects need to be tracked. Adapted from Blumberg, Peterson, and Parasuraman (2015); licensed under CC BY 4.0.

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 distribution of pupil sizes. (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.

The generative probability density function will be as follows:

$$p_{\text{size}_n} \sim \text{Normal}(\alpha + c_{\text{load}}_n \cdot \beta, \sigma) \quad (4.1)$$

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 predictor, and `c_load` is our predictor that is multiplied by β . We will generally indicate with the prefix `c_`, that a predictor (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 had not 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 we can fit a frequentist model with `lm(p_size ~ 1 + c_load, data_set)`, when we fit a Bayesian model, we have to specify priors for each of the parameters.

For setting the priors, we need to do some research and find some information about pupil sizes. Although we might know that pupil diameters range between 2 to 4 mm in bright light to 4 to 8 mm in the dark (Spector 1990), 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 subject with no attentional load for the first 100 ms, measured every 10 ms, in the data frame `df_pupil_pilot` from the package `bcogsci`: This will give us some idea about the order of magnitude of our dependent variable.

```
data("df_pupil_pilot")
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.¹ 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 normal distribution and setting its standard deviation as 500.

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

Given that our predictor 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% credible interval limited by approximately $1000 \pm 2 \cdot 500 = [0, 2000]$ units. We can calculate this with more precision in R using the `qnorm` function:

```
qnorm(c(.025, .975), mean = 1000, sd = 500)
```

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

We know that the measurements of the pilot data are strongly correlated because they were taken 10 milliseconds apart. For this reason, they won't tell us how much the pupil size can vary. We set up quite an uninformative prior for σ that encodes our lack of precise information: σ is surely larger than zero and has to be in the order of magnitude of the pupil size with no load.

$$\sigma \sim \text{Normal}_+(0, 1000) \quad (4.3)$$

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% credible interval.

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

¹The 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 for the prior between 500-1500 would be fine if the standard deviation is large.

```
## [1] 31.3 2241.4
```

In order to compute the 95% credible interval, we used `qtnorm` from the `extraDistr` package rather than `qnorm()`. As mentioned earlier, the relevant command specification is `qtnorm(..., a = 0)`; recall that `a = 0` indicates a truncated normal distribution, truncated at the left by zero.

The mean of $Normal_+$, a normal distribution truncated at zero so as to allow for only positive values, does not coincide with its location indicated with the parameter μ (and neither does the standard deviation coincide with the scale, σ); see Box 4.1.

```
samples <- rtnorm(20000, mean = 0, sd = 1000, a = 0)
c(mean = mean(samples), sd = sd(samples))
```

```
## mean    sd
##  797   607
```

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 usually observe changes in pupil size in our day-to-day life), we expect them to be much smaller than the pupil size, so we use the following prior:

$$\beta \sim Normal(0, 100) \quad (4.4)$$

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 (it is centered at 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% credible interval.

```
c(qnorm(.025, mean = 0, sd = 100), qnorm(.975, mean = 0, sd = 100))
```

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

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

The priors we have specified here are relatively uninformative; as mentioned earlier, this is because we don't have much prior experience with pupil size studies. In other settings, we might have more prior knowledge and experience; in that case, we would use somewhat more principled priors. We will return to this point in the chapter on priors (chapter 6) and on a Bayesian workflow (chapter 7).

Box 4.1. Truncated distributions

Any distribution can be truncated. For a continuous distribution, the truncated version of the original distribution will have non-zero probability density values for a continuous subset of the original coverage. To make this more concrete, in our previous example, the normal distribution has coverage for values between minus infinity to plus infinity, and our truncated version Normal_+ has coverage between zero and plus infinity: all negative values have a probability density of zero. Let's see how we can generalize this to be able to understand any truncation of any continuous distribution. (For the discrete case, we can simply replace the integral with a sum, and replace PDF with PMF).

From the axiomatic definitions of probability, we know that the area below a PDF, $f(x)$, must be equal to one (section 1.1). More formally, this means that the integral of f evaluated as $f(-\infty < X < \infty)$ should be equal to one:

$$\int_{-\infty}^{\infty} f(x)dx = 1 \quad (4.5)$$

But if the distribution is truncated, f is going to be evaluated in some subset of its possible values, $f(a < X < b)$; in the specific case of Normal_+ , for example, $a = 0$, and $b = \infty$. In the general case, this means that the integral of the PDF evaluated for $a < X < b$ will lower than one unless $a = -\infty$ and $b = +\infty$.

$$\int_a^b f(x)dx < 1 \quad (4.6)$$

We want to ensure that we build a new PDF for the truncated distribution so that even though it has less coverage than the non-truncated version, it still integrates to one. To achieve this, we divide the “unnormalized” PDF by the total area of $f(a < X < b)$ (recall the discussion surrounding Equation (1.15)):

$$f_{[a,b]}(x) = \frac{f(x)}{\int_a^b f(x)dx} \quad (4.7)$$

The denominator of the previous equation is the difference between the CDF evaluated at $X = b$ and the CDF evaluated at $X = a$; this can be written as $F(b) - F(a)$:

$$f_{[a,b]}(x) = \frac{f(x)}{F(b) - F(a)} \quad (4.8)$$

For the specific case, where $f(x)$ is $\text{Normal}(x|0, \sigma)$ and we want the PDF of $\text{Normal}_+(x|0, \sigma)$, and thus $a = 0$ and $b = \infty$.

$$\text{Normal}_+(x|0, \sigma) = \frac{\text{Normal}(x|0, \sigma)}{1/2} \quad (4.9)$$

Because $F(X = b = \infty) = 1$ and $F(X = a = 0) = 1/2$.

You can verify this in R (and this is valid for any value of `sd`).

```
dnorm(1, mean = 0) * 2 == dtnorm(1, mean = 0, a = 0)
```

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

Unless the truncation of the normal distribution is symmetrical, the location, μ , of the truncated normal does not coincide with the mean, and for any type of truncation, the scale, σ , does not coincide with the standard deviation. Confusingly enough, the arguments of the family of functions `*tnorm` keep the names of the family of functions `*norm`, and the location is called `mean` and the scale `sd`.

For example, the mean of the truncated normal with boundaries a and b , given its location and scale is as follows:

$$E(X \mid a < X < b) = \mu + \sigma \frac{\phi(\alpha) - \phi(\beta)}{\Phi(\beta) - \Phi(\alpha)} \quad (4.10)$$

where $\alpha = (a - \mu)/\sigma$, $\beta = (b - \mu)/\sigma$, $\phi(X)$ is the PDF of the standard normal ($\mu = 0, \sigma = 1$) evaluated at X , and $\Phi(X)$ is the CDF of the standard normal evaluated at X .

We build a function in R that calculates the mean for any truncated normal as follows:

```
mean_n_ab <- function(mu = 0, sigma = 1, a = -Inf, b = Inf) {
  alpha <- (a - mu) / sigma
  beta <- (b - mu) / sigma
  mu + sigma * (dnorm(alpha) - dnorm(beta)) /
    (pnorm(beta) - pnorm(alpha))
}
```

We can try it in R for our $Normal_+(0, 1000)$:

```
mean_n_ab(mu = 0, sigma = 1000, a = 0)
## [1] 798
```

We get similar results calculating the average of 20000 samples.

```
mean(rtnorm(20000, mean = 0, sd = 1000, a = 0))
## [1] 804
```

4.1.2 The `brms` model

Before fitting the `brms` model of the effect of load on pupil size, load the data and center the predictor `load`:

```
data("df_pupil")
(df_pupil <- df_pupil %>%
  mutate(c_load = load - mean(load)))
```

```
## # A tibble: 41 x 5
##   subj trial  load p_size c_load
##   <int> <int> <int>  <dbl>  <dbl>
## 1    701     1     2  1021. -0.439
## 2    701     2     1   951. -1.44
## 3    701     3     5  1064.  2.56
## # ... with 38 more rows
```

Now fit the `brms` model:

```
fit_pupil <- brm(p_size ~ 1 + c_load,
  data = df_pupil,
  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 intercept we indicate this with `0 + ...` or `-1 + ...`. See also the Box 4.2 for more details about the treatment of the intercepts by `brms`. The priors are normal distributions for the intercept (α) and the slope (β), and a truncated normal distribution for the scale of the likelihood (σ), which in this case, since we are dealing with a normal distribution, it coincides with its the standard deviation. `brms` will automatically truncate that distribution and allow for only positive values.

Inspect the output of our model now. The posteriors and trace plots are shown in Figure 4.2; the figure is generated by typing:

```
plot(fit_pupil)
```

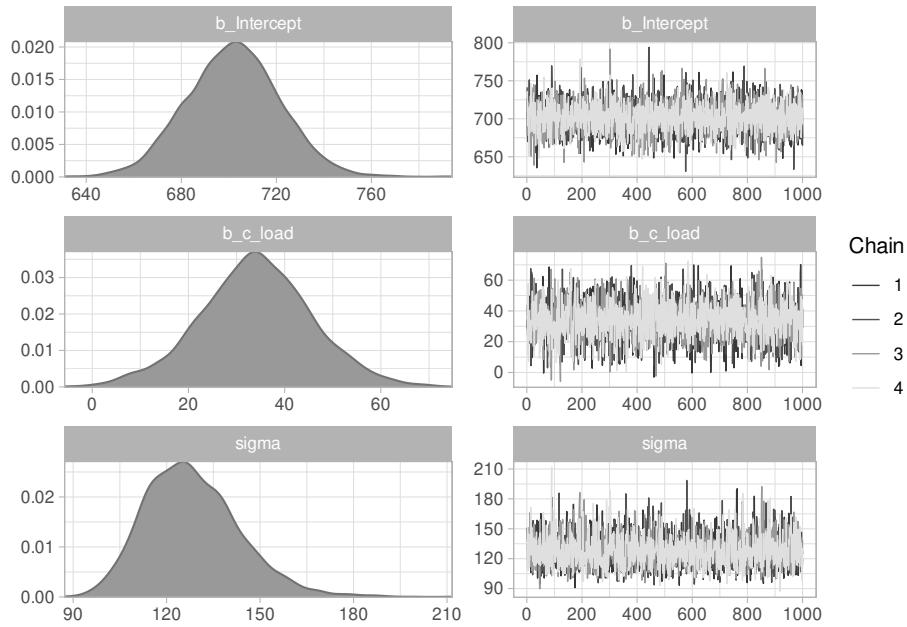


FIGURE 4.2: Posterior distributions of the parameters in the brms model `fit_pupil`, along with the corresponding trace plots.

```
fit_pupil
```

```
## ...
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept    701.93     19.74   663.94   740.40 1.00     3900    2796
## c_load       34.04     11.63    10.07   57.50 1.00     3710    2554
##
## Family Specific Parameters:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma      128.27     15.13   102.81   161.29 1.00     3434    2653
##
## ...
```

In the next section, we discuss how one can communicate the relevant information from the model.

Box 4.2. *Intercepts in brms*

When we set up a prior for the intercept in `brms`, we actually set a prior for an intercept assuming that all the predictors are centered. This means that when predictors are not centered (and only then), there is a mismatch between the interpretation of the intercept as returned in the output of `brms` and the interpretation of the intercept with respect to its prior specification. In this case, only the intercept in the output corresponds to the formula in the `brms` call. However, as we show below, when the intercept is much larger than the effects that we are considering in the formula (what we generally call β), this discrepancy hardly matters.

The reason for this mismatch when our predictors are uncentered is that `brms` increases sampling efficiency by automatically centering all the predictors internally (that is the population-level design matrix X is internally centered around its column means when `brms` fits a model). 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. This is in contrast with the centered version presented before, where the intercept represented the the pupil size for the average load of 2.44 (`c_load = 0`). The difference between the non-centered model (below) and the centered version presented before is depicted in Figure 4.3.

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:

```

prior_nc <- c(
  prior(normal(800, 200), class = b, coef = Intercept),
  prior(normal(0, 1000), class = sigma),
  prior(normal(0, 100), class = b, coef = load)
)

fit_pupil_non_centered <- brm(p_size ~ 0 + Intercept + load,
  data = df_pupil,
  family = gaussian(),
  prior = prior_nc
)

```

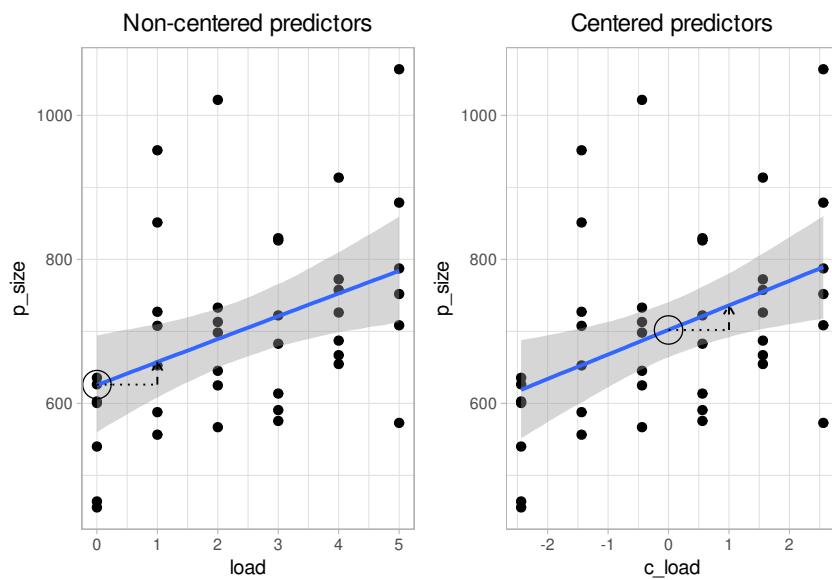


FIGURE 4.3: Regression lines for the non-centered and centered linear regressions. The intercept (or α) represented by a circle is positioned differently depending on the centering, whereas the slope (or β) represented by a vertical dashed line has the same magnitude in both models.

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 on 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. Once we use `0 + Intercept + ...`, the intercept is not calculated with predictors that are automatically centered any more.

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,
  variable = c("b_Intercept", "b_load"))

##           Estimate Est.Error   Q2.5 Q97.5
## b_Intercept     626.1      34.4 559.56 694.4
## b_load         31.8       11.6   8.36  54.4
```

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

```
fit_pupil_wrong <- brm(p_size ~ 1 + load,
  data = df_pupil,
  family = gaussian(),
  prior = prior_nc
)
```

What does it mean to set a prior to α_{centered} in a model that doesn’t include α_{centered} ?

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

$$\alpha + \text{load}_n \cdot \beta = \alpha_{\text{centered}} + (\text{load}_n - \text{mean}(\text{load})) \cdot \beta \quad (4.11)$$

The left side of Equation (4.11) 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 $\alpha_{centered}$.

$$\begin{aligned}\alpha + load_n \cdot \beta &= \alpha_{centered} + load_n \cdot \beta - mean(load) \cdot \beta \\ \alpha &= \alpha_{centered} - mean(load) \cdot \beta \\ \alpha + mean(load) \cdot \beta &= \alpha_{centered}\end{aligned}\tag{4.12}$$

That means that in the centered model, we are actually setting our prior to $\alpha + mean(load) \cdot \beta$. When β is very small (or the means of our predictors are very small because they might be almost centered), 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 is important to pay attention to what the parameters represent that we are setting priors on.

4.1.3 How to communicate the results?

We want to answer our research question “What is the effect of attentional load on the subject’s pupil size?” For that we’ll need to examine what happens with the posterior distribution of β , which is printed out as `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, 34.04, and we can be 95% certain that the value of β , given the model and the data, lies between 10.07 and 57.5.

We see that as the attentional load increases, the pupil size of the subject 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. (The intercept and the slopes are always preceded by `b_` in `brms`. One can see all the names of parameters being estimated with `variables()`.)

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

```
## [1] 0.999
```

This high probability does not mean that the effect of load is non-zero. It means instead that it's much more likely that the effect is positive rather than negative. In order to claim that the effect is likely to be non-zero, we would have to compare the model with an alternative model in which the model assumes that the effect of load is 0. We'll come back to this issue in the model comparison chapter 15.

4.1.4 Descriptive adequacy

Our model converged and we obtained a posterior distribution. There is, however, no guarantee that our model is good enough to represent our data. We can use posterior predictive checks to check the descriptive adequacy of the model.

Sometimes it's useful to customize the posterior predictive check to visualize the fit of our model. We iterate over the different loads (e.g., 0 to 4), and we show the prior predictive distributions based on 1000 simulations for each load together with the observed pupil sizes in Figure 4.4. We don't have enough data to derive a strong conclusion: both the predictive distributions and our data look very widely spread out, and it's 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.

```
for (l in 0:4) {
  df_sub_pupil <- filter(df_pupil, load == l)
  p <- pp_check(fit_pupil,
    type = "dens_overlay",
    ndraws = 100,
    newdata = df_sub_pupil
  ) +
    geom_point(data = df_sub_pupil, aes(x = p_size, y = 0.0001)) +
    ggtitle(paste("load: ", l)) +
    coord_cartesian(xlim = c(400, 1000))
```

```

print(p)
}

```

In Figure 4.5, we look instead at the distribution of a summary statistic, such as mean pupil size by load:

```

for (l in 0:4) {
  df_sub_pupil <- filter(df_pupil, load == l)
  p <- pp_check(fit_pupil,
    type = "stat",
    ndraws = 1000,
    newdata = df_sub_pupil,
    stat = "mean"
  ) +
    geom_point(data = df_sub_pupil, aes(x = p_size, y = 0.0001)) +
    ggtitle(paste("load: ", l)) +
    coord_cartesian(xlim = c(400, 1000))
  print(p)
}

```

Figure 4.5 shows that the observed means for no load and for a load of one are falling in the tails of the distributions. Although our model predicts a monotonic increase of pupil size, the data might be indicating that the relevant difference is simply between no load, and some load. 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 with this model, we are overinterpreting noise.

4.2 Log-normal model: Does trial affect response times?

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

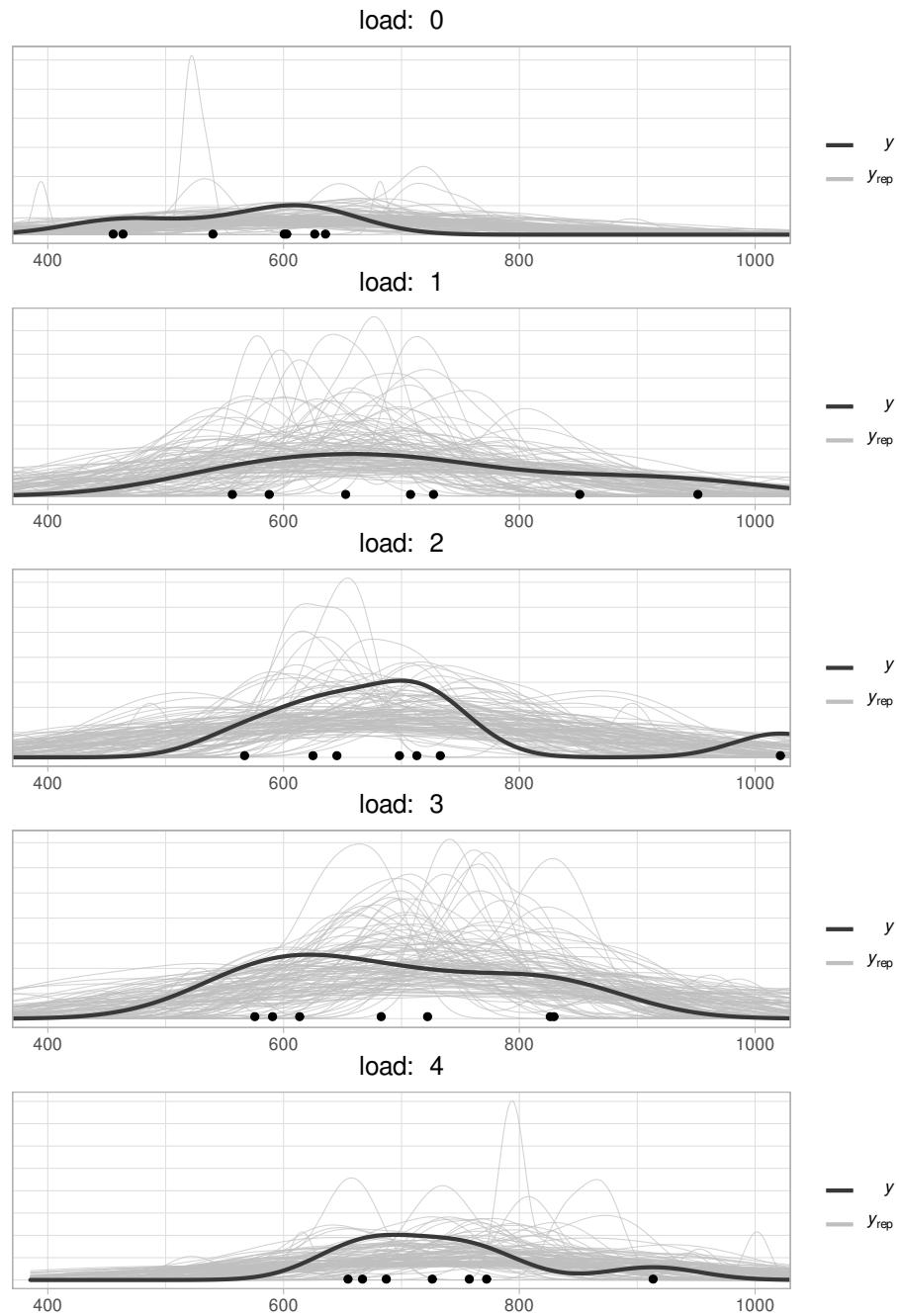


FIGURE 4.4: The plot shows 100 predicted distributions with the label y_{rep} , the distribution of pupil size data in black with the label y , and the observed pupil sizes in black dots for the five levels of attentional load.

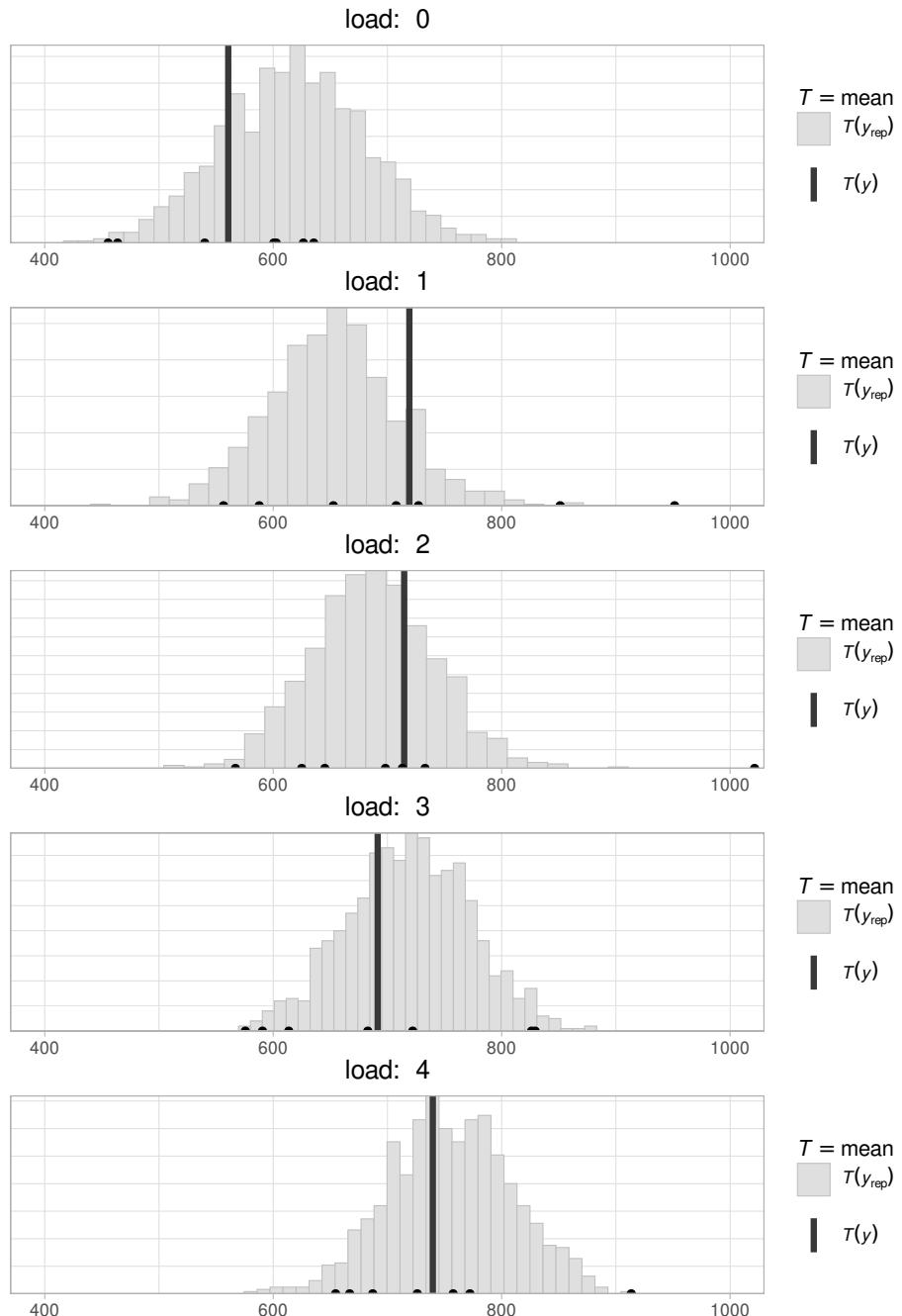


FIGURE 4.5: Distribution of posterior predicted means in gray and observed pupil size means in black lines by load.

the space bar. We'll use the same data set `df_spacebar` as before, and we'll center the column `trial`:

```
df_spacebar <- df_spacebar %>%
  mutate(c_trial = trial - mean(trial))
```

4.2.1 Likelihood and priors for the log-normal model

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

$$rt_n \sim \text{LogNormal}(\alpha + c_{\text{trial}}_n \cdot \beta, \sigma) \quad (4.13)$$

where $n = 1, \dots, N$, and rt is the dependent variable (response 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 .

$$\begin{aligned} \alpha &\sim \text{Normal}(6, 1.5) \\ \sigma &\sim \text{Normal}_+(0, 1) \end{aligned} \quad (4.14)$$

We still need a prior for β . 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 ; for details, see Box 4.3. We are going to try to understand how all our priors interact, by generating some prior predictive distributions. We start with the following prior centered in zero, a prior agnostic regarding the direction of the effect, which allows for both a slowdown ($\beta > 0$) or a speedup ($\beta < 0$):

$$\beta \sim \text{Normal}(0, 1) \quad (4.15)$$

This is our first attempt at a prior predictive distribution:

```
df_spacebar_ref <- df_spacebar %>%
  mutate(rt = runif(n(), 0, 10000))
fit_prior_press_trial <- brm(rt ~ 1 + c_trial,
```

```

data = df_spacebar_ref,
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)
),
sample_prior = "only",
control = list(adapt_delta = .9),
iter = 3000
)

```

In order to understand the type of data that we are assuming a priori with the prior of the parameter β , we'll plot the median difference between the effects at adjacent trials. As the prior of β gets wider we are going to observe larger differences between adjacent trials. The objective of the prior predictive checks is to calibrate the prior of β to obtain a plausible range of differences. We are going to plot a distribution of medians because they are less affected by the variance in the posterior predicted distribution than the distribution of mean differences; 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 . (Recall that the mean of log-normal distributed values depend on both the location, μ and the scale, σ , of the distribution.) To plot the median effect, we first define a function that calculates the difference between adjacent trials, and then applies the median to the result. We use that function in `pp_check` and we show the results in Figure 4.6. As expected, it is centered on zero (as our prior), but we see that the distribution of possible medians for the effect is too widely spread out and includes values that are too extreme.

```

median_effect <- function(x) {
  median(x - lag(x), na.rm = TRUE)
}
pp_check(fit_prior_press_trial,
  type = "stat",

```

```

stat = "median_effect",
# each bin has a width of 500ms
binwidth = 500) +
# cut the top of the plot to improve its scale
coord_cartesian(ylim = c(0, 50))

```

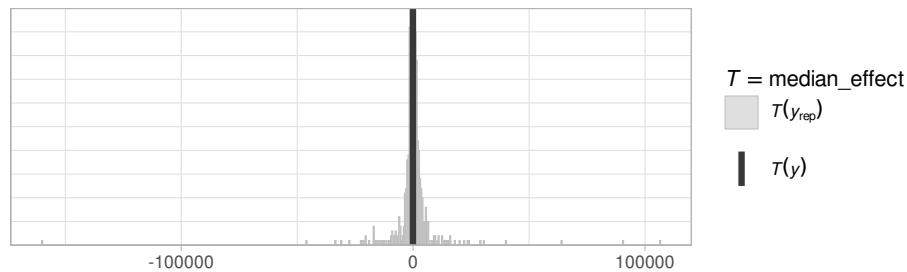


FIGURE 4.6: Prior predictive distribution of the median effect of the model defined in 4.2 with $\beta \sim \text{Normal}(0, 1)$.

We repeat the same procedure with $\beta \sim \text{Normal}(0, 0.01)$, and we plot it in Figure 4.7. The prior predictive distribution shows us that the prior is still quite vague, it is, however at least in the right order of magnitude.

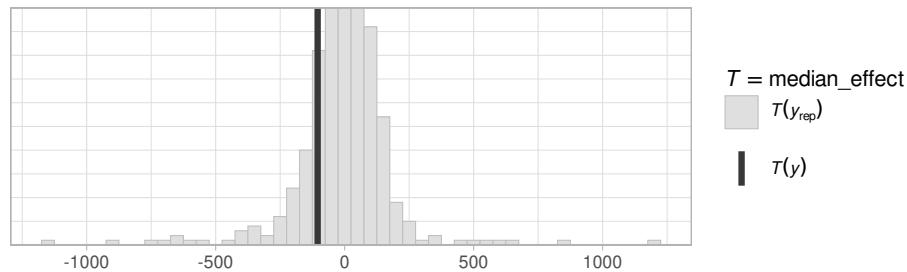


FIGURE 4.7: Prior predictive distribution of the median effect of the model defined in 4.2 with $\beta \sim \text{Normal}(0, 0.01)$.

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 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. We will return to the issue of prior selection in chapter 6.

Box 4.3. Understanding the Log-normal likelihood

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

$$\text{LogNormal}(y|\mu, \sigma) = f(z) = \frac{1}{\sqrt{2\pi\sigma^2}y} \exp\left(-\frac{(\log(y) - \mu)^2}{2\sigma^2}\right) \quad (4.16)$$

As explained in section 3.5.2, the model from Equation (4.13) is equivalent to the following:

$$\log(rt_n) \sim \text{Normal}(\alpha + c_{trial_n} \cdot \beta, \sigma) \quad (4.17)$$

The family of normal distributions is closed under linear transformations: that is, if X is normally distributed with mean μ and standard deviation σ , then (for any real numbers a and b), $aX + b$ is also normally distributed, with mean $a\mu + b$ (and standard deviation $\sqrt{a^2\sigma^2} = |a|\sigma$).

This means that, assuming $Z \sim \text{Normal}(\alpha, \sigma)$, Equation (4.17) can be re-written as follows:

$$\log(rt_n) = Z + c_{trial_n} \cdot \beta \quad (4.18)$$

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

$$\begin{aligned}
 rt_n &= \exp(Z + c_{trial}n \cdot \beta) \\
 rt_n &= \exp(Z) \cdot \exp(c_{trial}n \cdot \beta) \\
 rt_n &= Y \cdot \exp(c_{trial}n \cdot \beta)
 \end{aligned} \tag{4.19}$$

The last equation has two terms being multiplied, the first one, Y , is telling us that we are assuming that response times are log-normally distributed with a median of $\exp(\alpha)$, the second term, $\exp(c_{trial}n \cdot \beta)$ is telling us 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 response times for two adjacent trials (this is in contrast to what happens with an additive model such as normal likelihood); see Figure 4.8. This is because, unlike in the additive case, the intercept doesn't cancel out:

- Additive case:

$$\begin{aligned}
 (\alpha + trial_n \cdot \beta) - (\alpha + trial_{n-1} \cdot \beta) &= \\
 = \alpha - \alpha + (trial_n - trial_{n-1}) \cdot \beta &= \\
 = (trial_n - trial_{n-1}) \cdot \beta
 \end{aligned} \tag{4.20}$$

- Multiplicative case:

$$\begin{aligned}
 \exp(\alpha) \cdot \exp(trial_n \cdot \beta) - \exp(\alpha) \cdot \exp(trial_{n-1} \cdot \beta) &= \\
 = \exp(\alpha)(\exp(trial_n \cdot \beta) - \exp(trial_{n-1} \cdot \beta)) &= \\
 \neq (\exp(trial_n) - \exp(trial_{n-1})) \cdot \exp(\beta)
 \end{aligned} \tag{4.21}$$

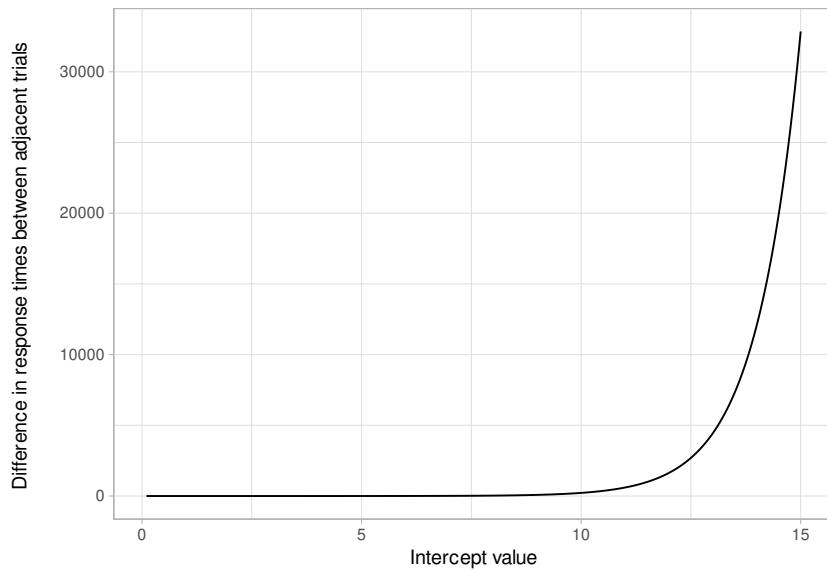


FIGURE 4.8: Fitted value of the difference in reaction time between two adjacent trials, when $\beta = 0.01$ and α lies between 0.1 and 15. The graph shows how changes in the intercept lead to changes in the difference in response times between trials, even if β is fixed.

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: Any (fixed) negative value for β will lead to exponential decay and any (fixed) positive value will lead to exponential growth; see Figure 4.9.

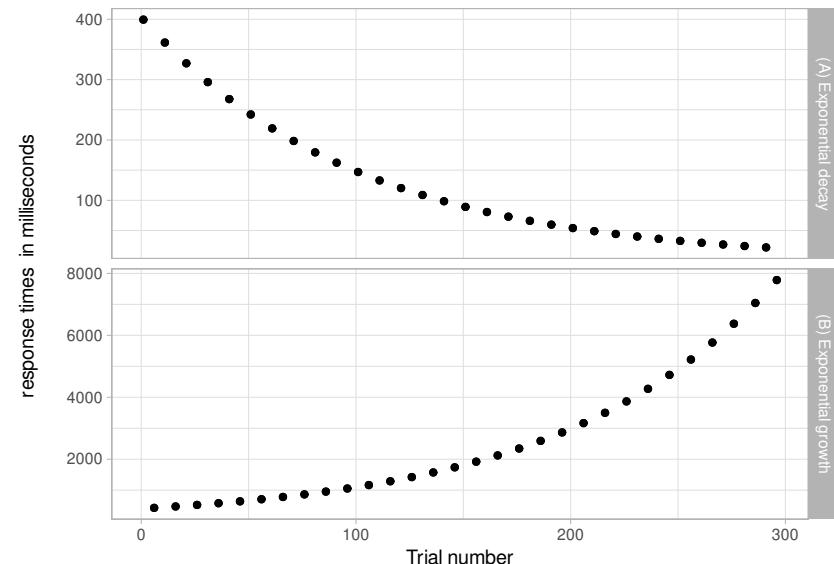


FIGURE 4.9: Fitted value of the dependent variable (response times in ms) as function of trial number, when (A) $\beta = -0.01$, exponential decay, and when (B) $\beta = 0.01$, exponential growth.

Can exponential growth or decay make sense? We need to consider that 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 or a floor effect: response times cannot truly be 0 milliseconds, or take several minutes. However, in our specific model, exponential growth or decay *by trial* is probably a bad approximation: We will predict that our subject 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:

$$rt_n \sim \text{LogNormal}(\alpha + c_{\log_trial_n} \cdot \beta, \sigma) \quad (4.22)$$

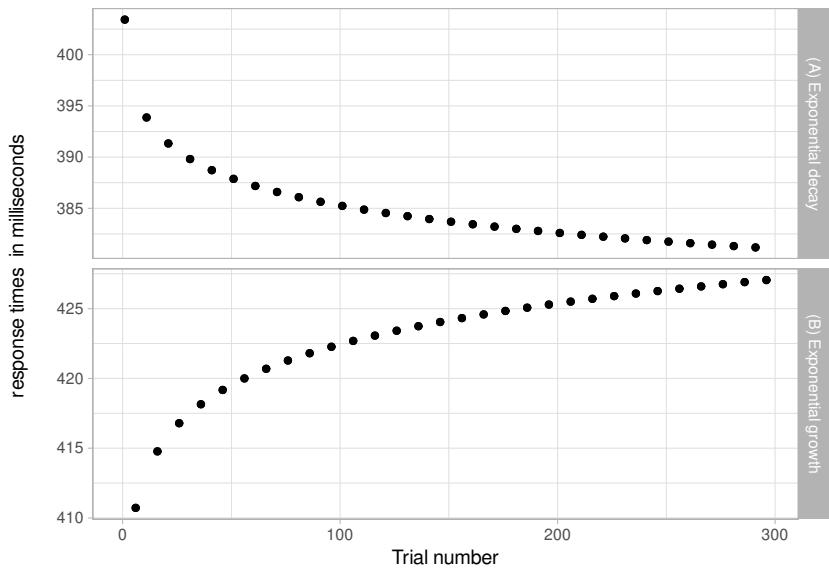


FIGURE 4.10: Fitted value of the dependent variable (response times in ms) as function of the natural logarithm of the trial number, when (A) $\beta = -0.01$, exponential decay, and when (B) $\beta = .01$, exponential growth.

Log-normal distributions everywhere

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, Stahel, and Abbt (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, Stahel, and Abbt (2001) point out that because a random variable that results from multiplying many independent variables has an approximate log-normal distribution, the most basic indicator of the importance of the log-normal distribution may be very gen-

eral: 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 response times? It has been long noticed that the log-normal distribution often provides a good fit to response times distributions (Brée 1975; Ulrich and Miller 1994). One advantage of assuming log-normally distributed response 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 increase with the mean, as has been observed in empirical distributions of response times (Wagenmakers, Grasman, and Molenaar 2005). Interestingly, it turns out that log-normal response times are also easily generated by certain process models. Ulrich and Miller (1993) show, for example, that models in which response 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 response times.

4.2.2 The `brms` model

We are now relatively satisfied with the priors for our model, and we can fit the model of the effect of trial as a button-pressing using `brms`. We need to specify that the family is `lognormal()`.

```
fit_press_trial <- brm(rt ~ 1 + c_trial,  
  data = df_spacebar,  
  family = lognormal(),
```

```

prior = c(
  prior(normal(6, 1.5), class = Intercept),
  prior(normal(0, 1), class = sigma),
  prior(normal(0, .01), 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 . These parameters are on the log scale:

```

posterior_summary(fit_press_trial)[, c("Estimate", "Q2.5", "Q97.5")]

##                 Estimate      Q2.5      Q97.5
## b_Intercept     5.118492   5.1061   5.131571
## b_c_trial       0.000522   0.0004   0.000643
## sigma          0.123329   0.1152   0.132614
## lp__        -1603.648696 -1606.7401 -1602.289496

```

The posterior distributions can be plotted to obtain a graphical summary of all the parameters in the model (Figure 4.11:

```
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, 0.00064].

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

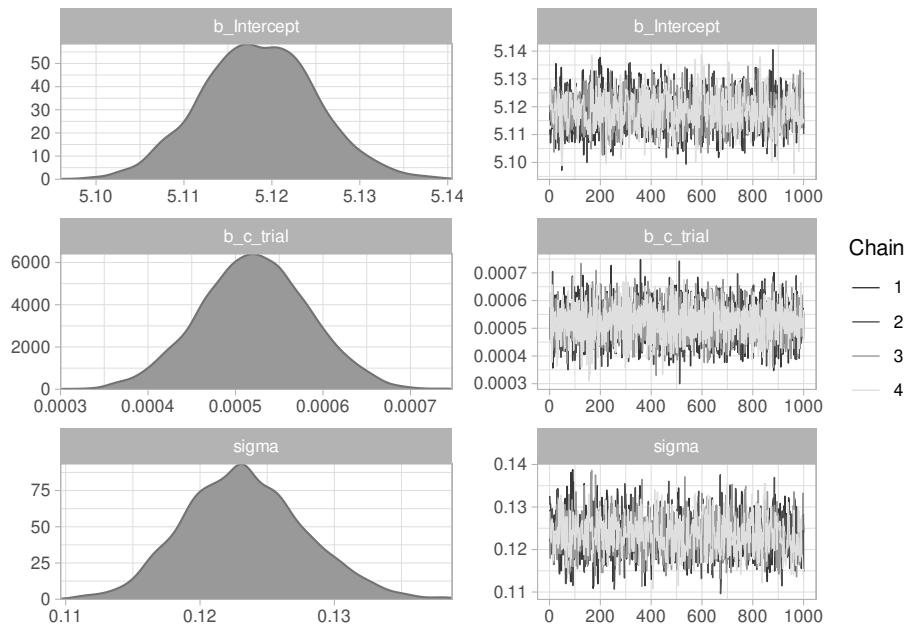


FIGURE 4.11: Posterior distributions of the model of the effect of trial on button-pressing.

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 certain estimate if we consider the difference between response 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 <- as_draws_df(fit_press_trial)$b_Intercept
beta_samples <- as_draws_df(fit_press_trial)$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(0.025, 0.975))
)

##   mean   2.5% 97.5%
## 0.0872 0.0667 0.1076

```

We will obtain different estimate if we consider the difference between the second and the first trial:

```

first_trial <- min(df_spacebar$c_trial)
second_trial <- min(df_spacebar$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(0.025, 0.975))
)

##   mean   2.5% 97.5%
## 0.0793 0.0621 0.0959

```

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 ($0.087 \cdot 100$), with a 95% credible interval ranging from 6.671 to 10.757. 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, Engelmann, and Vasishth (2017), Mahowald et al. (2016),

Nicenboim, Roettger, and Vasishth (2018), and Vasishth et al. (2013). We will discuss meta-analysis in later in the book, in chapter 13.

Sometimes, researchers are only interested in establishing that an effect is present or absent; 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, in order to answer such a question, a Bayes factor analysis must be carried out. We'll come back to this issue in the model comparison chapters 15–17.

4.3 Logistic regression: Does set size affect free recall?

In this section, we will learn how the principles we have learned so far can naturally extend to *generalized* linear models (GLMs). We focus on one special case of GLMs that has wide application in linguistics and psychology, logistic regression.

As an example data set, we look at a study investigating the capacity level of working memory. The data are a subset of a data set created by Oberauer (2019). Each subject 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 4.12. We will focus on the data from one subject.

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

The data can be found in `df_recall` in the package `bcogsci`. The code below loads the data, centers the predictor `set_size`, and briefly explores the data-set.

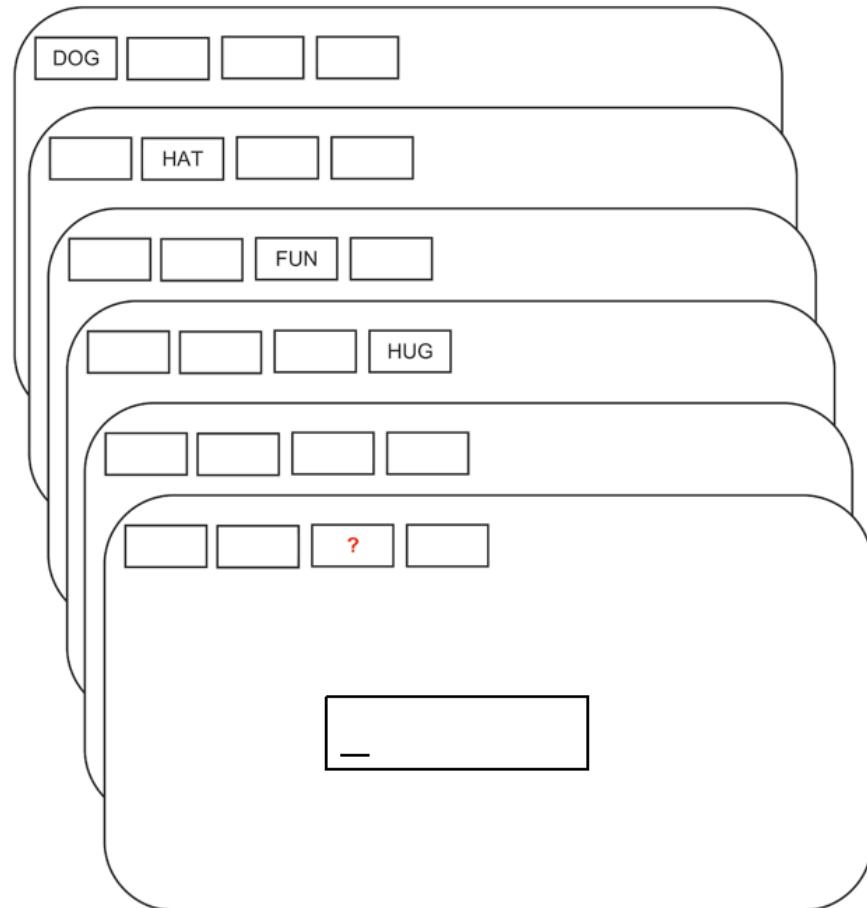


FIGURE 4.12: Flow of events in a trial with memory set size 4 and free recall. Adapted from Oberauer (2019); licensed under CC BY 4.0.

```
data("df_recall")
df_recall <- df_recall %>%
  mutate(c_set_size = set_size - mean(set_size))
# Set sizes in the data set:
df_recall$set_size %>%
  unique() %>% sort()
## [1] 2 4 6 8
```

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

```
## # A tibble: 4 x 2
## # Groups:   set_size [4]
##   set_size     n
##   <int> <int>
## 1       2     23
## 2       4     23
## 3       6     23
## # ... with 1 more row
```

Here, the column `correct` records the incorrect vs. correct responses with 0 vs 1, 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
```

```
## # A tibble: 92 x 8
##   subj  set_size correct trial session block tested c_set_size
##   <chr>    <int>    <int> <int>    <int> <int>    <dbl>
## 1 10        4        1      1      1      1      2      -1
## 2 10        8        0      4      1      1      8      3
## 3 10        2        1      9      1      1      2      -3
## # ... with 89 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, with a 50% probability of getting a 1 using `rbern` from the package `extraDistr`.

```
rbern(10, prob = 0.5)
```

```
## [1] 0 0 1 0 0 0 1 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, $n = 1, \dots, N$ 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 .

$$\text{correct}_n \sim \text{Bernoulli}(\theta_n) \quad (4.23)$$

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 $+\infty$, rather than being limited to zeros and ones.

The generalized linear modeling framework solves this problem by defining a *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.

$$\eta_n = g(\theta_n) = \log\left(\frac{\theta_n}{1 - \theta_n}\right) \quad (4.24)$$

The term $\frac{\theta_n}{1 - \theta_n}$ is called the *odds*.² The logit link function is therefore a log-odds; it maps probability values ranging from [0, 1] to real numbers ranging from $-\infty$ to $+\infty$. Figure 4.13 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.

²Odds 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.

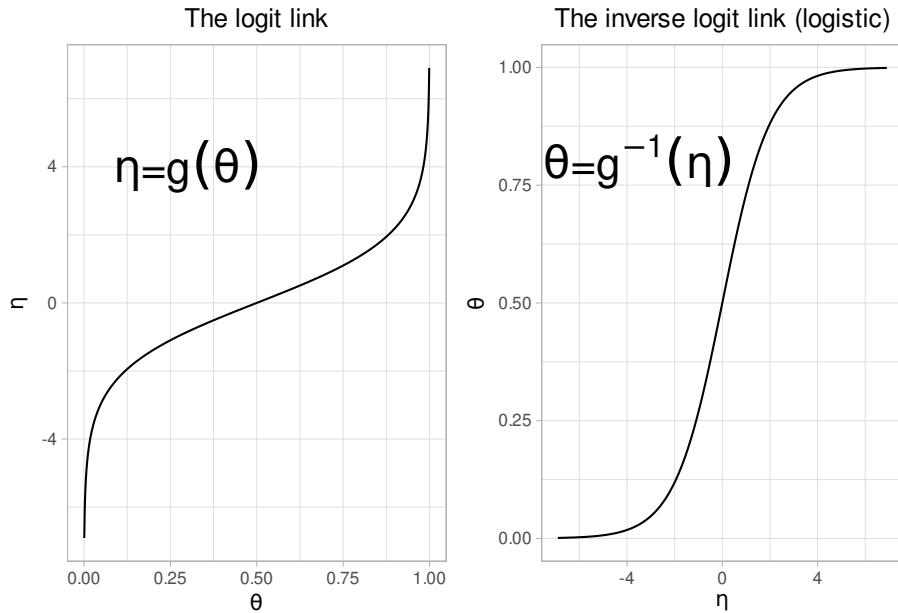


FIGURE 4.13: The logit and inverse logit (logistic) function.

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:

$$\eta_n = \ln \left(\frac{\theta_n}{1 - \theta_n} \right) = \alpha + \beta \cdot c_set_size \quad (4.25)$$

Unlike the linear models, the model is defined so that there is no residual error term (ε) in this model. Once η_n is estimated, one can solve the above equation for θ_n (in other words, we compute the inverse of the logit function and obtain the estimates on the probability scale). This gives the above-mentioned logistic regression function:

$$\theta_n = g^{-1}(\eta_n) = \frac{\exp(\eta_n)}{1 + \exp(\eta_n)} = \frac{1}{1 + \exp(-\eta_n)} \quad (4.26)$$

The last equality in the equation above arises by dividing both the numerator and denominator by $\exp(\eta_n)$.

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

$$\text{correct}_n \sim \text{Bernoulli}(\theta_n) \quad (4.27)$$

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 $\theta_n = \frac{\exp(\eta_n)}{1+\exp(\eta_n)}$. An example of this calculations will be shown in the next section.

4.3.2 Priors for the logistic regression

In order to decide on priors for α and β , we need to take into account that these parameters do not represent probabilities or proportions, but *log-odds*, the x-axis in Figure 4.13 (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 predictor, the intercept, α , represents the log-odds 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 log-odds 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 `qlogis(.5)` tells us that .5 corresponds to zero in log-odds space. How do we include a great deal of uncertainty? We could look at Figure 4.13, and decide on a standard deviation of 4 in a normal distribution centered in zero:

$$\alpha \sim \text{Normal}(0, 4) \quad (4.28)$$

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()

```

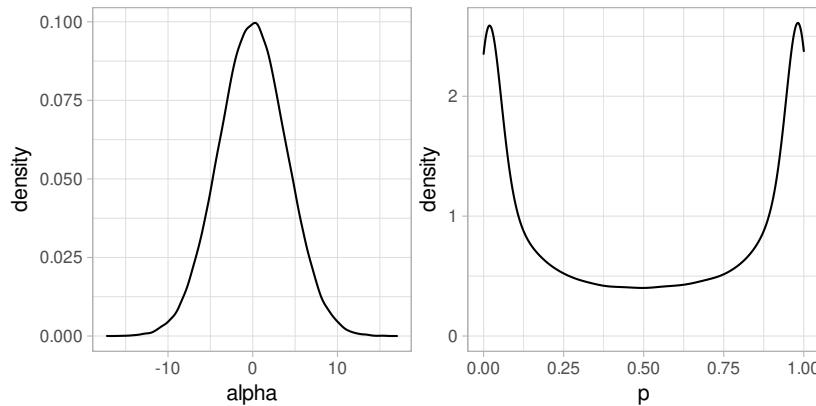


FIGURE 4.14: Prior for $\alpha \sim \text{Normal}(0, 4)$ in log-odds and in probability space.

Figure 4.14 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 4.15.

$$\alpha \sim \text{Normal}(0, 1.5) \quad (4.29)$$

We need to decide now on the prior for β , the effect in log-odds of increasing the set size. We could 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:

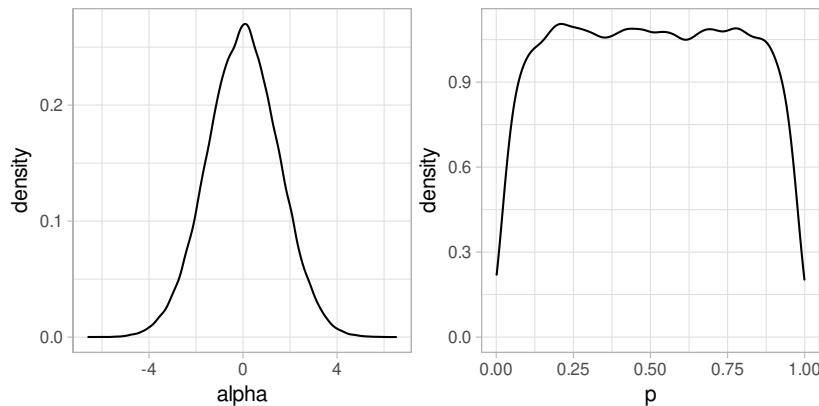


FIGURE 4.15: Prior for $\alpha \sim \text{Normal}(0, 1.5)$ in log-odds and in probability space.

- (a) $\beta \sim \text{Normal}(0, 1)$
- (b) $\beta \sim \text{Normal}(0, .5)$
- (c) $\beta \sim \text{Normal}(0, .1)$
- (d) $\beta \sim \text{Normal}(0, .01)$
- (e) $\beta \sim \text{Normal}(0, .001)$

In principle, we could produce the prior predictive distributions using `brms` with `sample_prior = "only"` and then `predict()`. The problem with our choice of priors here is that some of our priors are too uninformative; this becomes clear when we look at the prior predictive distributions. We can avoid using too uninformative a prior by performing prior predictive checks directly in R using the `r*` functions (e.g., `rnorm`, `rbinom`, etc.) together with loops. This method is not as simple as using the convenient functions provided by `brms`, but it can be very powerful. We show the prior predictive distributions in Figure 4.16, for the details on the implementation in R, see Box 4.4.

Box 4.4. *Prior predictive checks in R*

The following function is an edited version of the earlier `normal_predictive_distribution` from the Box 3.1 in section 3.2; it has

been edited to make it compatible with logistic regression and dependent on set size.

As we did before, our custom function uses the `purrr` function `map2_dfr`, which runs an efficient for-loop, iterating over two vectors (here `alpha_samples` and `beta_samples`), and builds a data frame with the output.

```
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,
                # 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),
                correct_pred = rbernoulli(N_obs, p = 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 for each set size:

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

Now, iterate over plausible standard deviations of β with the `purrr` function `map_dfr`, which iterates over one vector (here `sds_beta`), and also builds a data frame with the output.

```
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)
})
```

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, ")"))
```

Plot the accuracy in Figure 4.16 as follows.

```
mean_accuracy %>%
  ggplot(aes(accuracy)) +
  geom_histogram() +
  facet_grid(set_size ~ prior) +
  scale_x_continuous(breaks = c(0, .5, 1))
```

It's sometimes more useful to look at the predicted differences in

accuracy between set sizes. We calculate them as follows, and plot them in Figure 4.17.

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

diff_accuracy %>%
  ggplot(aes(diff_accuracy)) +
  geom_histogram() +
  facet_grid(diffsize ~ prior) +
  scale_x_continuous(breaks = c(-.5, 0, .5))
```

Figure 4.16 shows that, 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 1 is problematic: There is too much probability concentrated near 0 and 1 for set sizes of 2 and 8. 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 in Figure 4.17.

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 0.1. Under this reasoning, we settle on the following priors:

$$\begin{aligned}\alpha &\sim \text{Normal}(0, 1.5) \\ \beta &\sim \text{Normal}(0, 0.1)\end{aligned}\tag{4.30}$$

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`. We need to specify that the family is `bernoulli()`, and the link is `logit`.

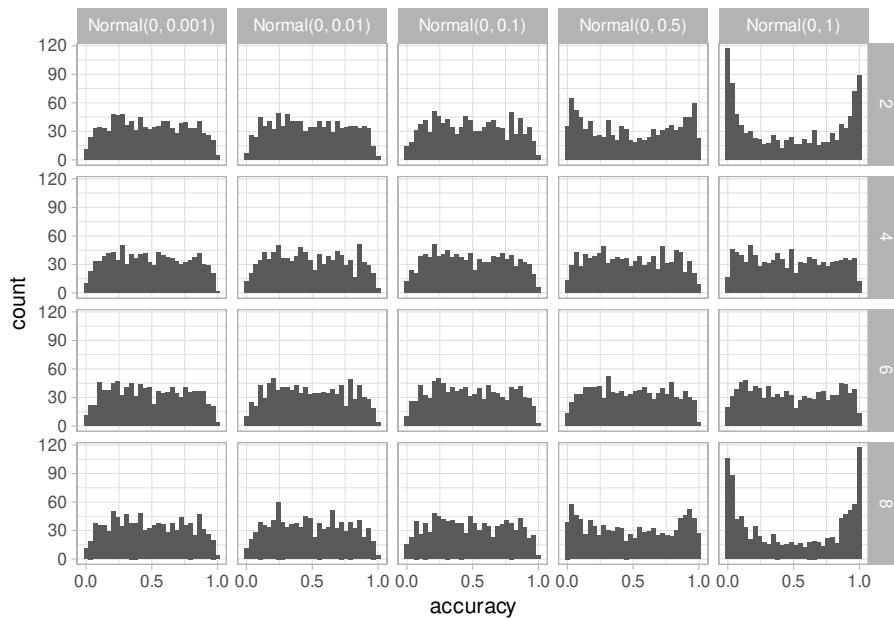


FIGURE 4.16: Prior predictive distribution of mean accuracy of the model defined in section 4.3, for different set sizes and different priors for β .

```
fit_recall <- brm(correct ~ 1 + c_set_size,
  data = df_recall,
  family = bernoulli(link = logit),
  prior = c(
    prior(normal(0, 1.5), class = Intercept),
    prior(normal(0, .1), 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, variable = c("b_Intercept", "b_c_set_size"))
```

	Estimate	Est.Error	Q2.5	Q97.5
## b_Intercept	1.910	0.3096	1.330	2.5488

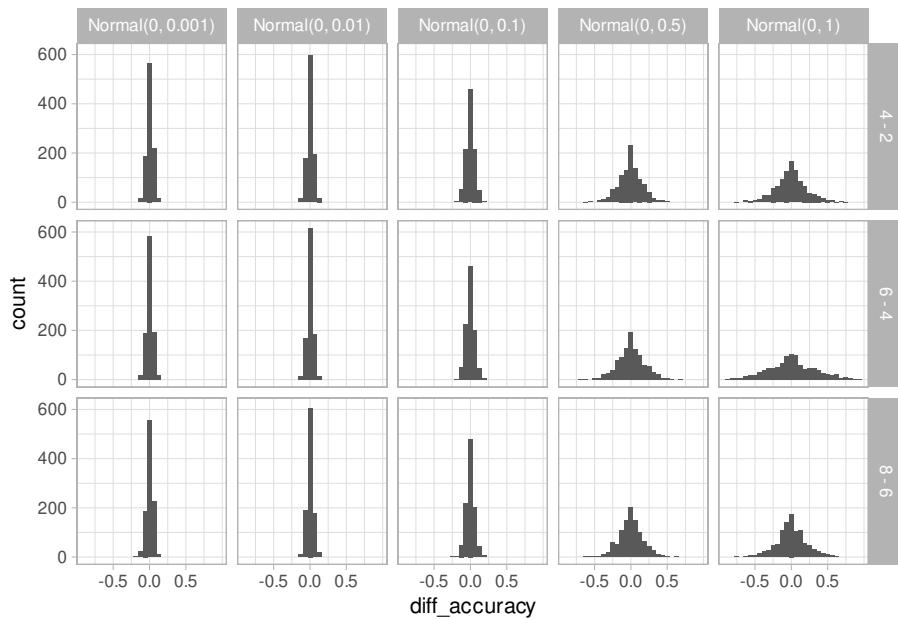


FIGURE 4.17: Prior predictive distribution of differences in mean accuracy between set sizes of the model defined in 4.3 for different priors for β .

```
## b_c_set_size -0.184 0.0818 -0.347 -0.0254
```

Inspecting `b_c_set_size`, we see that increasing the set size have a detrimental effect in recall, as we suspected.

Plot the posteriors as well (Figure 4.18):

```
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?

Here, we are in a situation analogous to the one we saw earlier 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: $\hat{\beta} = -0.184$, 95% CrI = $[-0.347, -0.025]$.

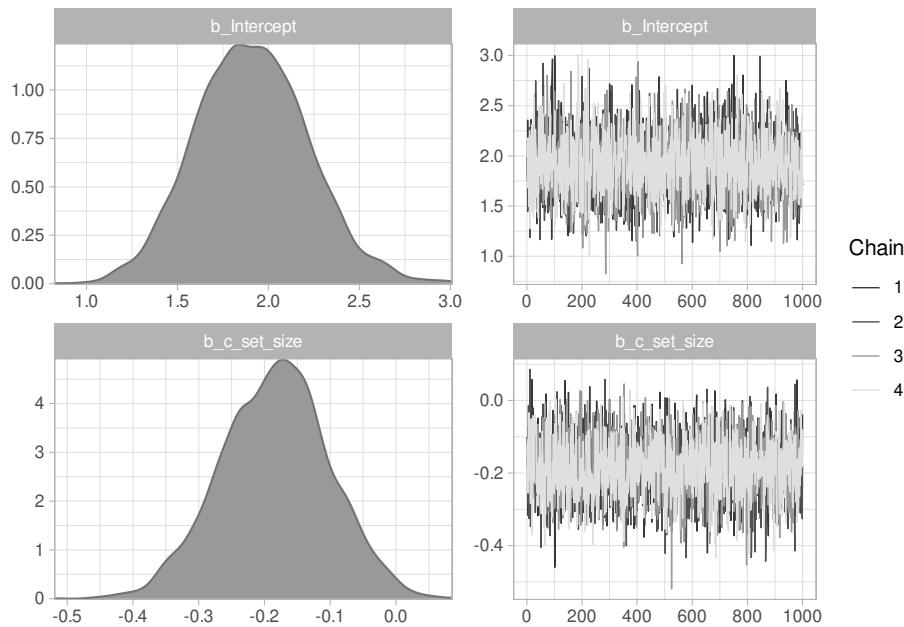


FIGURE 4.18: Posterior distributions of the parameters in the brms model `fit_recall`, along with the corresponding trace plots.

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

```
alpha_samples <- as_draws_df(fit_recall)$b_Intercept
av_accuracy <- plogis(alpha_samples)
c(mean = mean(av_accuracy), quantile(av_accuracy, c(0.025, 0.975)))
```

```
##   mean 2.5% 97.5%
## 0.867 0.791 0.927
```

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 <- as_draws_df(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(0.025, 0.975)))
```

```
##      mean     2.5%    97.5%
## -0.01909 -0.03782 -0.00291
```

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 -0.019 , 95% CrI = $[-0.038, -0.003]$. Recall that the average set size, 5, was not presented to the subject! We could alternatively look at the decrease in accuracy from a set size of 2 to 4:

```
effect_4m2 <-
  plogis(alpha_samples +
    (4 - mean(df_recall$set_size)) * beta_samples) -
  plogis(alpha_samples +
    (2 - mean(df_recall$set_size)) * beta_samples)
c(mean = mean(effect_4m2),
  quantile(effect_4m2, c(0.025, 0.975)))
```

```
##      mean     2.5%    97.5%
## -0.02984 -0.05528 -0.00561
```

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 investigate whether our model was right using another experiment. To make predictions for other set sizes, we extend our data set, 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 size in the data set). 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 4.19.

```
df_recall_ext <- df_recall %>%
  bind_rows(tibble(
    set_size = rep(c(3, 5, 7), 23),
    c_set_size = set_size -
      mean(df_recall$set_size),
    correct = 0
  ))
# nicer label for the facets:
set_size <- paste("set size", 2:8) %>%
  setNames(~3:3)
pp_check(fit_recall,
  type = "stat_grouped",
  stat = "mean",
  group = "c_set_size",
  newdata = df_recall_ext,
  facet_args = list(
    ncol = 1, scales = "fixed",
    labeller = as_labeller(set_size)
  ),
  binwidth = 0.02
)
```

We could now gather new data in an experiment that also shows set sizes of 3, 5, and 7. These data would be *held out* from the model `fit_recall`, since the model was fit when those data were not available. Verifying that the new observations fit in our already generated posterior predictive distribution would be a way to test genuine predictions from our model.

Having seen how we can fit simple regression models, we turn to hierarchical models in the next chapter.

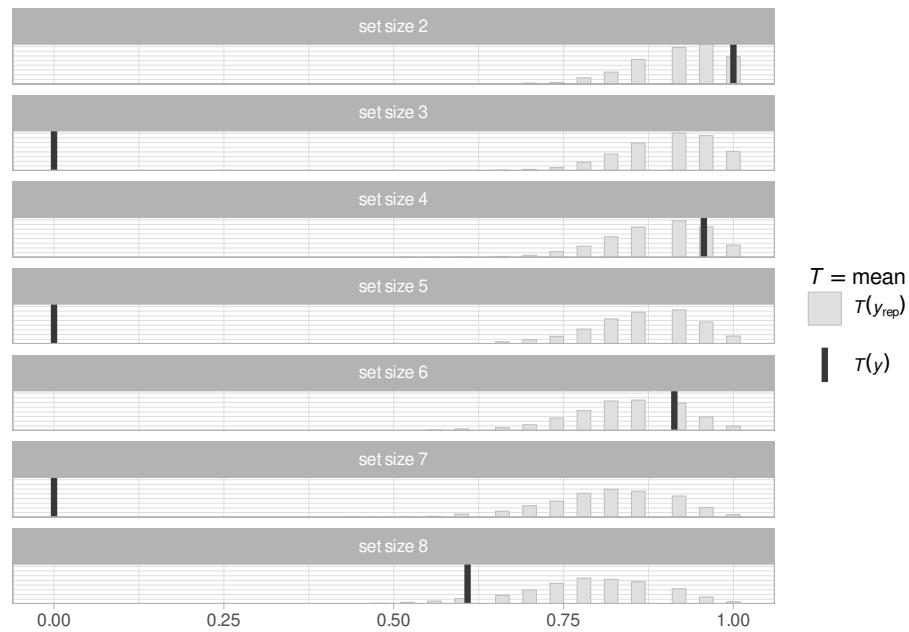


FIGURE 4.19: The distribution of posterior predicted mean accuracies for tested set sizes (2, 4, 6, and 8) and untested ones (3, 5, and 7) are labeled with y_{rep} . The observed mean accuracy, y , are only relevant for the tested set sizes.

4.4 Summary

In this chapter, we learned how to fit simple linear regression models and to fit and interpret models with a log-normal likelihood and logistic regression models. We investigated the prior specification for the models, using prior predictive checks, and the descriptive adequacy of the models using posterior predictive checks.

4.5 Further reading

Linear regression is discussed in several classic textbooks; these have largely a frequentist orientation, but the basic theory of linear modeling presented there can easily be extended to the Bayesian framework. An accessible textbook is by Dobson and Barnett (2011). Other useful textbooks on linear modeling are Harrell Jr (2015), Faraway (2016), Fox (2015), and Montgomery, Peck, and Vining (2012).

4.6 Exercises

Exercise 4.1. A simple linear regression: Power posing and testosterone.

Load the following data set:

```
data("df_powerpose")
head(df_powerpose)

##   id hptreat female age testm1 testm2
## 2 29    High   Male  19   38.7   62.4
## 3 30     Low Female  20   32.8   29.2
## 4 31    High Female  20   32.3   27.5
## 5 32     Low Female  18   18.0   28.7
## 7 34     Low Female  21   73.6   44.7
## 8 35    High Female  20   80.7  105.5
```

The data set, which was originally published in Carney, Cuddy, and Yap (2010) but released in modified form by Fosse (2016), shows the testosterone levels of 39 different individuals, before and after treatment, where treatment refers to each individual being assigned to a high power pose or a low power pose. In the original paper by Carney, Cuddy, and Yap (2010), the unit given for testosterone measurement (estimated from saliva samples) was picograms per milliliter (pg/ml). One picogram per milliliter is 0.001 nanogram per milliliter (ng/ml).

The research hypothesis is that on average, assigning a subject a high power pose vs. a low power pose will lead to higher testosterone levels after treatment. Assuming that you know nothing about normal ranges of testosterone using salivary measurement, choose an appropriate Cauchy prior (e.g., $\text{Cauchy}(2.5)$) for the target parameter(s).

Investigate this claim using a linear model and the default priors of `brms`. You'll need to estimate the effect of a new variable that encodes the change in testosterone.

Exercise 4.2. A first linear regression: Revisiting attentional load effect on pupil size.

Here, we revisit the analysis shown in the chapter, on how attentional load affects pupil size.

- a. Our priors for this experiment were quite arbitrary. How do the prior predictive distributions look like? Do they make sense?
- b. Is our posterior distribution sensitive to the priors that we selected? Perform a sensitivity analysis to find out whether the posterior is affected by our choice of prior for the σ .
- c. Our data set includes also a column that indicates the trial number. Could it be that trial has also an effect on the pupil size? As in `lm`, we indicate another main effect with a + sign. How would you communicate the new results?

Exercise 4.3. Log-normal model: Revisiting the effect of trial on response times.

We continue considering the effect of trial on response times.

- a. Estimate the slowdown in milliseconds between the last two times the subject pressed the space bar in the experiment.
- b. How would you change your model (keeping the log-normal likelihood) so that it includes centered log-transformed trial numbers or square-root-transformed trial numbers (instead of centered trial numbers)? Does the effect in milliseconds change?

Exercise 4.4. Logistic regression: Revisiting the effect of set size on free recall.

Our data set includes also a column coded as tested that indicates the position of the queued word. (In Figure 4.12 tested would be 3). Could it be that position also has an effect on recall accuracy? How would you incorporate this in the model? Verify the descriptive adequacy of our new model.

Exercise 4.5. Red is the sexiest color.

Load the following data set:

```
data("df_red")
head(df_red)

##      risk age red pink redorpink
## 8      0   19   0    0        0
## 9      0   25   0    0        0
## 10     0   20   0    0        0
## 11     0   20   0    0        0
## 14     0   20   0    0        0
## 15     0   18   0    0        0
```

The data set is from a study that contains information about the color of the clothing worn (red, pink, or red or pink) when the subject (female) is at risk of becoming pregnant (is ovulating, self-reported). The broader issue being investigated is whether women wear red more often when they are ovulating (in order to attract a mate). Using logistic regressions, fit three different models to investigate whether being ovulating increases the probability of wearing (a) red, (b) pink, or (c) either pink or red. Use priors that are reasonable (in your opinion).

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 subjects and experimental items (e.g., words, pictures, objects that are presented to the subjects). These clusters arise because we have multiple (repeated) observations for each subject, and for each item. If we want to incorporate this grouping structure in our analysis, we need to use a hierarchical model (also called multi-level or mixed model). This kind of clustering and hierarchical modeling is closely related to the idea of *exchangeability*.

Exchangeability, an important concept in hierarchical models: It is the Bayesian equivalent of the phrase “independent and identically distributed” that one regularly encounters in classical (i.e., frequentist) statistics textbooks.

Informally, the idea is as follows. Suppose we assign a numerical index to each of the levels of a group (e.g., to each subject). When the levels are exchangeable, we can reassigned the indices arbitrarily and lose no information. In hierarchical models, we treat the levels of the group as exchangeable, and observations within each level in the group 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 group level, these are predictors that indicate characteristics of the levels in the group (e.g., the working memory capacity score of each subject). 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 units in the group are exchangeable. This

is the reason why the item number is an appropriate cluster (the indexes are exchangeable), but trial number is not. In other words, if we permute the numbering of the items there is no loss of information, whereas if we change the numbering of the trials there might be (in case that, for example, as trial numbers increase, subjects get more experienced or fatigued). Even if we are not interested in the specific cluster-levels estimates, hierarchical models allow us to generalize to the underlying population (subjects, items) from which the clusters in the sample were drawn. For more on exchangeability, see Box 5.1 and consult the further reading at the end of the chapter.

Box 5.1. *Finitely exchangeable random variables*

Formally, we say that the random variables Y_1, \dots, Y_N are finitely exchangeable if, for any set of particular outcomes of an experiment y_1, \dots, y_N , the probability $p(y_1, \dots, y_N)$ that we assign to these outcomes is unaffected by permuting the labels given to the variables. In other words, for any permutation $\pi(n)$, where $n = 1, \dots, N$ (π is a function that takes as input the positive integer n and returns another positive integer. e.g., takes subject indexed as 1, and returns index 3), we can reasonably assume that $p(y_1, \dots, y_N) = p(y_{\pi(1)}, \dots, y_{\pi(N)})$. A simple example is a coin tossed twice. Suppose the first coin toss is $Y_1 = 1$, a heads, and the second coin toss is $Y_2 = 0$, a tails. If we are willing to assume that the probability of getting one heads is unaffected by whether it appears in the first or the second toss, i.e., $p(Y_1 = 1, Y_2 = 0) = p(Y_1 = 0, Y_2 = 1)$, then we assume that the indices are exchangeable.

One reason that exchangeability is important for Bayesian modeling is the following. Suppose that the parameters for each level in a group are θ_n , where the levels are labeled $n = 1, \dots, N$. An example of groups is subjects. Suppose also that the data y_n , where $n = 1, \dots, N$ are assumed to be generated as $y_n \sim \text{Normal}(\theta_n, \sigma)$. If the data y_n are exchangeable, the parameters θ_n are also exchangeable. The fact that the θ_n are exchangeable can be shown (Bernardo and Smith 2009) to be mathematically equivalent to assuming that they come from a common distribution, for example: $\theta_n \sim \text{Normal}(\theta, \tau)$.

The parameters θ and τ , called hyperparameters, are unknown and have prior distributions (hyperpriors) defined for them. This fact leads to a hierarchical relationship between the parameters: there is a common parameter θ for all the levels of a group, and the parameters θ_n are assumed to be generated as a function of this common parameter θ :

- $\theta_n \sim Normal(\theta, \tau)$
- $y_n \sim Normal(\theta_n, \sigma)$

Here, τ represents between-group variability, and σ represents within-group variability; as mentioned above, both parameters have hyperpriors defined for them. The parameter θ has a prior defined for it too. The first two priors are called hyperpriors.

- $p(\theta)$
- $p(\tau)$
- $p(\sigma)$

In such a model, information about θ_n comes from two sources:

- (a) from each of the observed y_n corresponding to the respective θ_n parameter, and
- (b) from the parameters θ and τ that led to all the other y_j (where $j \neq n$) being generated.

There are two other configurations possible that do not involve this hierarchical structure and which represent two extreme scenarios.

One is called the *no pooling model*; here, each y_i is assumed to be generated from an independent distribution: $y_i \sim Normal(\theta_i, \sigma_i)$; there is no common distribution that generates the θ_i parameters as in the hierarchical model.

The other configuration is called the *complete pooling model*, in which the data y_i are assumed to be generated from a single distribution: $y_i \sim Normal(\theta, \sigma)$.

The hierarchical model lies between these two extremes and for this

reason is sometimes called a *partial pooling model*. One way that the hierarchical model is often described is that the estimates θ_n “borrow strength” from the parameter θ (which represents the grand mean in the above example). The three scenarios described above are illustrated in Figures 5.1-5.3. An important practical consequence of partial pooling is the idea of “borrowing strength from the mean”: if we have very sparse data from a particular member of a group (e.g., missing data from a particular subject), the estimate θ_n of that particular group member n is determined by the parameter θ . In other words, when the data are sparse for group member n , the posterior estimate θ_n is determined largely by the prior $p(\theta)$. In this sense, even the frequentist hierarchical modeling software in R, `lmer` from the package `lme4`, is essentially Bayesian in formulation (except of course that there is no prior as such on θ).

We will be looking at each of these models in the coming sections.

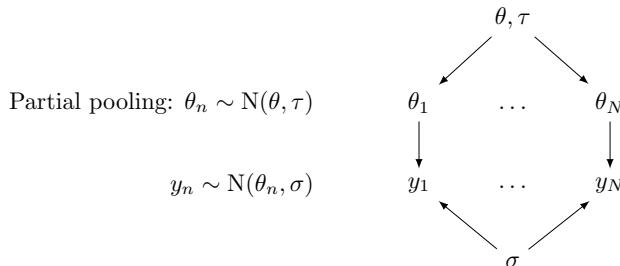


FIGURE 5.1: A directed acyclic graph illustrating a hierarchical model.

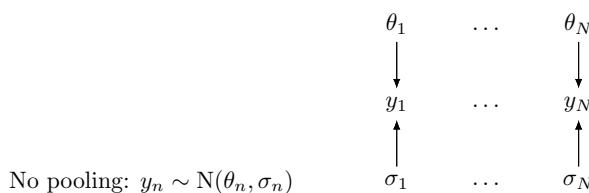


FIGURE 5.2: A directed acyclic graph illustrating a no pooling model.

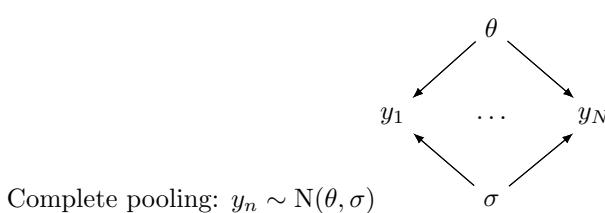


FIGURE 5.3: A directed acyclic graph illustrating a complete pooling model.

5.1 A hierarchical model with a normal likelihood: 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 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 ms after word onset over central parietal scalp sites (first reported in Kutas and Hillyard 1980, for semantic anomalies, and in 1984 for low predictable word; for a review, see Kutas and Federmeier 2011). The N400 is illustrated in Figure 5.4.

For example, in (1) below, 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 1.4).

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 com-

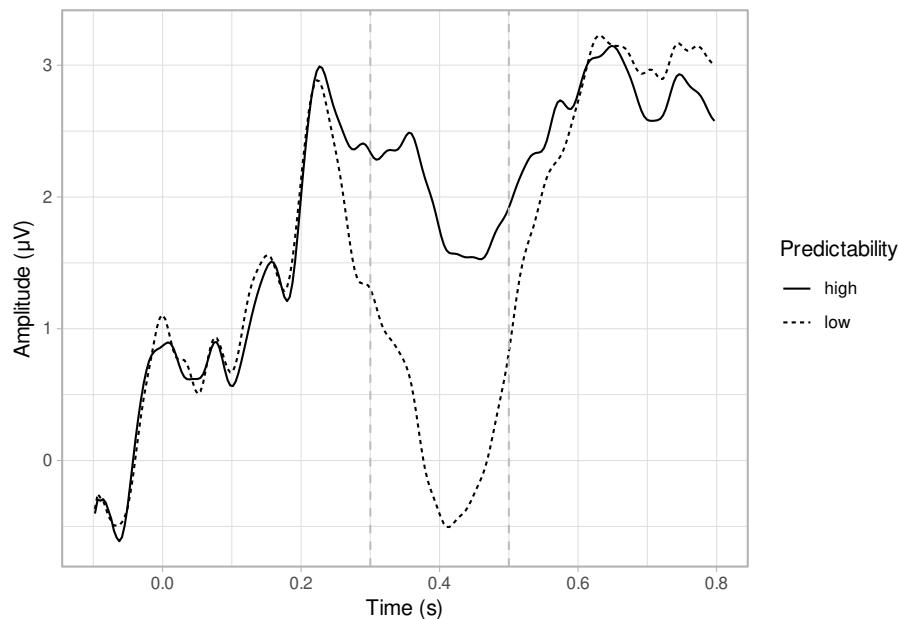


FIGURE 5.4: Typical ERP for the grand average across the N400 spatial window (central parietal electrodes: Cz, CP1, CP2, P3, Pz, P4, POz) for high and low predictability nouns (specifically from the constraining context of the experiment reported in Nicenboim, Vasishth, and Rösler 2020a). The x-axis indicates time in seconds and the y-axis indicates voltage in microvolts (unlike many EEG/ERP plots, the negative polarity is plotted downwards).

mon 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 the typical spatio-temporal window of the N400 (for example, see 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, Urbach, and Kutas (2005) with sentences like (2).

2. Example from DeLong, Urbach, and Kutas (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, Urbach, and Kutas 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 restrict the data set to the subjects from the Edinburgh lab, using the relevant subset containing the Edinburgh data from `df_eeg` in the `bcogsci` package. Center the cloze probability before using it as a predictor.

```
data("df_eeg")
(df_eeg <- df_eeg %>%
  mutate(c_cloze = cloze - mean(cloze)))
```

A tibble: 2,863 x 7
subj cloze item n400 cloze_ans N c_cloze
<dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 1 0 1 7.08 0 44 -0.476
2 1 0.03 2 -0.68 1 44 -0.446
3 1 1 3 1.39 44 44 0.524
... with 2,860 more rows

```
# Number of subjects
df_eeg %>%
  distinct(subj) %>%
  count()
```

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

One convenient aspect of using averages of EEG data is that they are roughly normally distributed. This allows us to use the Normal likelihood. Figure 5.5 shows the distribution of the data.

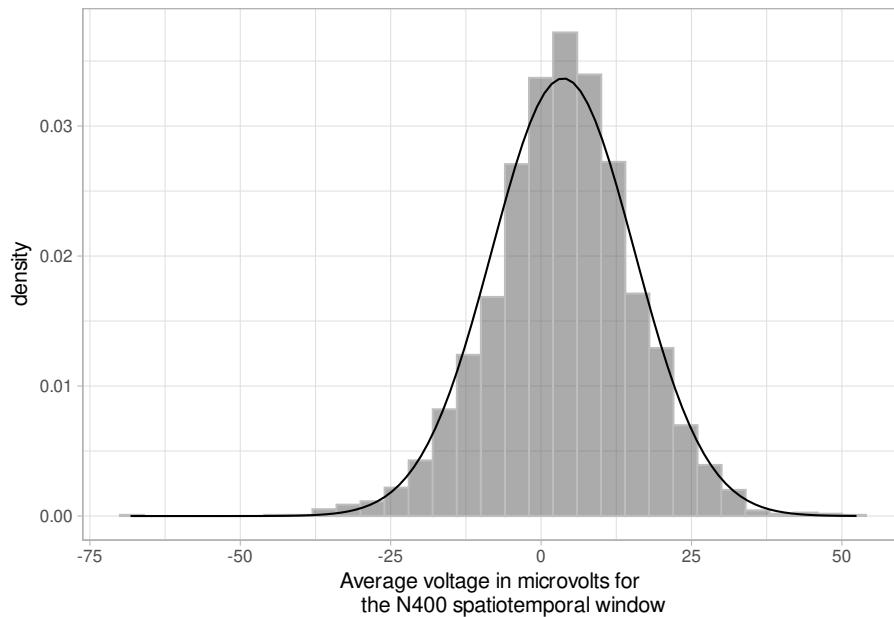


FIGURE 5.5: Histogram of the N400 averages for every trial, overlaid is a density plot of a normal distribution.

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.

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 of 0 and 0.1 is the

same as the difference in the signal when we compare nouns with cloze values of 0.1 and 0.2 (or 0.9 and 1). This is just 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 can now decide on a likelihood and priors.

5.1.1.2 Likelihood and priors

A normal likelihood seems reasonable for these data:

$$\text{signal}_n \sim \text{Normal}(\alpha + c_{\text{cloze}}_n \cdot \beta, \sigma) \quad (5.1)$$

where $n = 1, \dots, N$, and *signal* is the dependent variable (average signal in the N400 spatiotemporal window in microvolts). 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, $10\mu V$. As before, *a priori* we'll assume that effects can be negative or positive. We can quantify our prior knowledge regarding plausible values of β as normally distributed centered at zero with a standard deviation of $10\mu V$. (Other values such as $5\mu V$ would have been also reasonable, since it would entail that 95% of the prior mass probability is between -10 and 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 from the time window of interest, then the mean signal would be relatively close to 0. Since we know that 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 $10\mu V$.

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 50. Recall that since we truncate the distribution, the parameters location and scale do not correspond to the mean and standard deviation of the new distribution; see Box 4.1.

We can draw random samples from this truncated distribution and calculate their mean and standard deviation:

```
samples <- extraDistr::rtnorm(20000, mean = 0, sd = 50, a = 0)
c(mean = mean(samples), sd = sd(samples))
```

```
## mean    sd
## 40.3 30.4
```

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

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

##    2.5%   97.5%
##  1.54 113.13

# Analytically:
# c(qtnorm(.025, 0, 50, a = 0), qtnorm(.975, 0, 50, a = 0))
```

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

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

A model such as M_{cp} is sometimes called a *fixed-effects* model: all the parameters are fixed in the sense that 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)`.

We fit this model in `brms` as follows (the default family is `gaussian()` so we can omit it). As with the `lm` function in R, 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
)
```

For now, check the summary, and plot the posteriors of the model (Figure 5.6).

```
fit_N400_cp

## ...
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     3.67      0.22     3.24     4.10 1.00    4097    3249
## c_cloze      2.26      0.54     1.19     3.35 1.00    3766    2804
##
## Family Specific Parameters:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma       11.82      0.16    11.52    12.12 1.00    3643    2808
##
## ...

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 by subject (and also by the

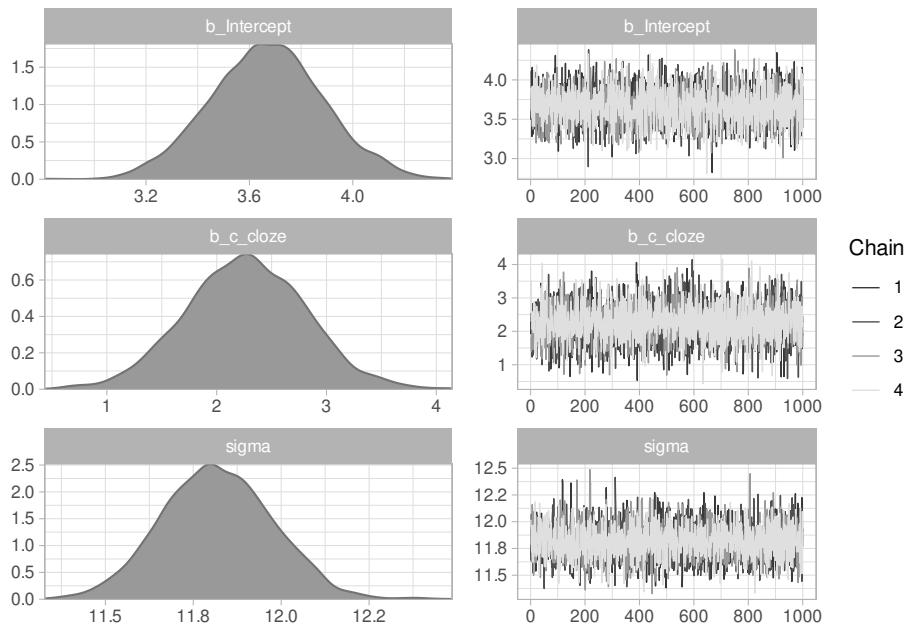


FIGURE 5.6: Posterior distributions of the complete pooling model, `fit_N400_cp`.

specific item, but we'll ignore this until section 5.1.4). It is reasonable to assume that EEG signals are more similar within subjects than between them. The following model assumes that each subject 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. Every subject's model is fit independently of the other subjects; the subjects have no parameters in common (an exception is the standard deviation; this is the same for all subjects).
3. There is a linear relationship between cloze and the EEG signal for the trial.

What likelihood and priors can we choose here?

¹For 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:

$$\text{signal}_n \sim \text{Normal}(\alpha_{\text{subj}[n]} + c_{\text{cloze}}_n \cdot \beta_{\text{subj}[n]}, \sigma) \quad (5.3)$$

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

We define the priors as follows:

$$\begin{aligned} \alpha_i &\sim \text{Normal}(0, 10) \\ \beta_i &\sim \text{Normal}(0, 10) \\ \sigma &\sim \text{Normal}_{+}(0, 50) \end{aligned} \quad (5.4)$$

In `brms`, such a model can be fit by removing the common intercept with the formula `n400 ~ 0 + factor(subj) + c_cloze:factor(subj)`.

This formula forces the model to estimate one intercept and one slope for *each* level of `subj`.² The by-subject intercepts are indicated with `factor(subj)` and the by-subject slopes with `c_cloze:factor(subj)`. It's very important to specify that `subject` should be treated as a factor and not as a number; we don't assume that subject number 3 will show 3 times more positive (or negative) average signal than subject number 1! 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. The parameters are independent from each other; it is only our previous knowledge (or prior beliefs) 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 setting 74 priors!

²If we don't remove the intercept, that is, if we use the formula `n400 ~ 1 + factor(subj) + c_cloze:factor(subj)`, with `factor(subj)` we are going to estimate the deviation between the first subject and each of the other subjects.

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

For this model, printing a summary means printing the 75 parameters ($\alpha_{1,\dots,37}$, $\beta_{1,\dots,37}$, and σ). We could do this as always by printing out the model results: just type `fit_N400_np`.

It may be easier to understand the output of the model by plotting $\beta_{1,\dots,37}$ using `bayesplot`. (`brms` also includes a wrapper for this function called `stanplot`). We can take a look at the internal names that `brms` gives to the parameters with `variables(fit_N400_np)`; they are `b_factorsubj`, 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 5.7. The subjects are ordered by the magnitude of their mean effects.

The model M_{np} does not estimate a unique population-level effect; instead, 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 $\hat{\beta}_{1,\dots,I}$, where I is the total number of subjects:

```
# parameter name of beta by subject:
ind_effects_np <- paste0(
  "b_factorsubj",
  unique(df_eeg$subj), ":c_cloze"
)
beta_across_subj <- as_draws_df(fit_N400_np) %>%
  #removes the meta data from the object
  as.data.frame() %>%
  select(ind_effects_np) %>%
  rowMeans()
```

```

## Note: Using an external vector in selections is ambiguous.
## i Use `all_of(ind_effects_np)` instead of `ind_effects_np` to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.

# Calculate the average of these estimates
(grand_av_beta <- tibble(
  mean = mean(beta_across_subj),
  lq = quantile(beta_across_subj, c(.025)),
  hq = quantile(beta_across_subj, c(.975))
))

## # A tibble: 1 x 3
##   mean     lq     hq
##   <dbl> <dbl> <dbl>
## 1  2.18  1.16  3.20

```

In Figure 5.7, the 95% credible interval of this overall mean effect is plotted as two vertical lines together with the effect of cloze probability for each subject (ordered by effect size). Here, rather than using a plotting function from `brms`, we can extract the summary of by-subject effects, reorder them by magnitude, and then plot the summary with a custom plot using `ggplot2`.

```

# make a table of beta's by subject
beta_by_subj <- posterior_summary(fit_N400_np,
  variable = ind_effects_np
) %>%
  as.data.frame() %>%
  mutate(subject = 1:n()) %>%
  ## reorder plot by magnitude of mean:
  arrange(Estimate) %>%
  mutate(subject = factor(subject, levels = subject))

```

The code below generates Figure 5.7.

```
# plot:
ggplot(
  beta_by_subj,
  aes(x = Estimate, xmin = Q2.5, xmax = Q97.5, y = subject)
) +
  geom_point() +
  geom_errorbar() +
  geom_vline(xintercept = grand_av_beta$mean) +
  geom_vline(xintercept = grand_av_beta$lq, linetype = "dashed") +
  geom_vline(xintercept = grand_av_beta$hq, linetype = "dashed") +
  xlab("By-subject effect of cloze probability in microvolts")
```

5.1.3 Varying intercepts and varying slopes model (M_v)

One major problem with the no-pooling model is that we completely ignore the fact that the subjects were doing the same experiment. We fit each subject's data ignoring the information available in the other subjects' data. The no-pooling model is very likely to *overfit* the individual subjects' data; we are likely to ignore the generalities of the data and we may end up overinterpreting noisy estimates from each subject's data. 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.

In the model that we fit next, we will assume that there is an overall effect that is common to the subjects and, importantly, that all subjects' parameters originate from one common (normal) distribution. This model specification will result in the estimation of posteriors for each subject being also influenced by what we know about all the subjects together. We begin with a hierarchical model with uncorrelated varying intercepts and slopes.³

³The analogous frequentist model can be fit using `lmer` from the package `lme4`, using `(1+c_cloze||subj)` or, equivalently, `(c_cloze||subj)` for the by-subject random effects.

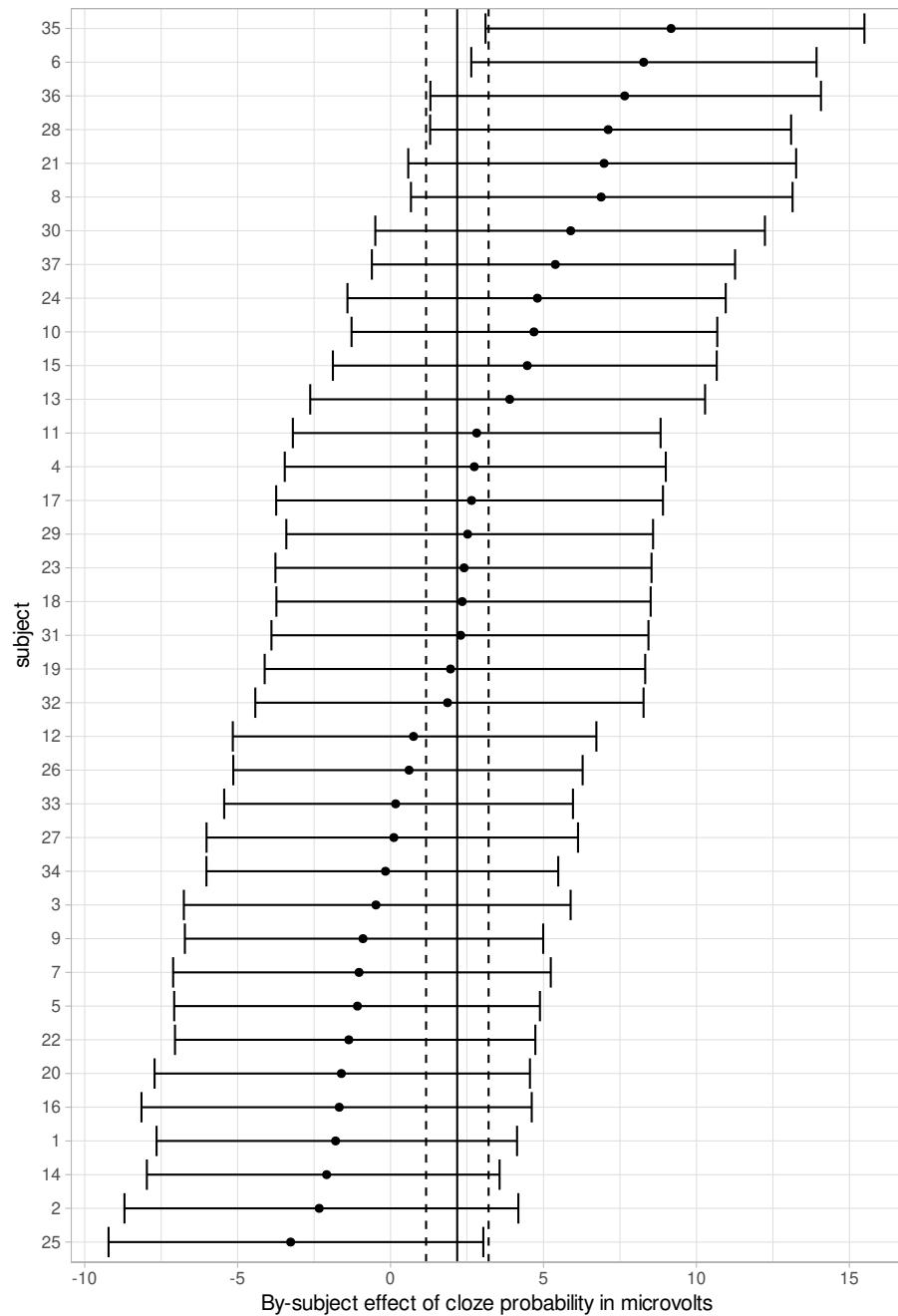


FIGURE 5.7: 95% credible intervals of the effect of cloze probability for each subject according to the no pooling model, `fit_N400_np`. The solid vertical line represents the mean over all the subjects; and the broken vertical lines mark the 95% credible interval for this mean.

5.1.3.1 Model assumptions

1. EEG averages for the N400 spatio-temporal window are normally distributed.
2. 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.
3. There is a linear relationship between cloze and the EEG signal.

5.1.3.2 Likelihood and priors

The likelihood now incorporates the assumption that both the intercept and slope are adjusted by subject.

$$\text{signal}_n \sim \text{Normal}(\alpha + u_{\text{subj}[n],1} + c_cloze_n \cdot (\beta + u_{\text{subj}[n],2}), \sigma) \quad (5.5)$$

$$\begin{aligned} \alpha &\sim \text{Normal}(0, 10) \\ \beta &\sim \text{Normal}(0, 10) \\ u_1 &\sim \text{Normal}(0, \tau_{u_1}) \\ u_2 &\sim \text{Normal}(0, \tau_{u_2}) \\ \tau_{u_1} &\sim \text{Normal}_+(0, 20) \\ \tau_{u_2} &\sim \text{Normal}_+(0, 20) \\ \sigma &\sim \text{Normal}_+(0, 50) \end{aligned} \quad (5.6)$$

In this model each subject has their own intercept adjustment, $u_{\text{subj},1}$, and slope adjustment, $u_{\text{subj},2}$.⁴ If $u_{\text{subj},1}$ is positive, the subject will have a more positive EEG signal than the grand mean average. If $u_{\text{subj},2}$ 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 (i.e., there will be a more positive effect of cloze probability on the N400). The parameters u are sometimes called random effects and thus a model with fixed effects (α and β) and

⁴The intercept adjustment is often called u_0 in statistics books, where the intercept might be called α or (sometimes also β_0), and thus u_1 refers to the adjustment to the slope. However, in this book, we start the indexing with 1 to be consistent with the Stan language.

random effects is called a mixed model. However, random effects have different meanings in different contexts. To avoid ambiguity, `brms` calls these random-effects parameters *group-level* effects. 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_1 . 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_2 . The standard deviations of these two adjustment terms, τ_{u_1} and τ_{u_2} , respectively, represent between subject variability; see Box 5.2.

Thus, the model M_v has three *standard deviations*: σ , τ_{u_1} and τ_{u_2} . 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_1} and τ_{u_2} characterize between-subject variability, and the variance component σ characterizes within-subject variability.

The by-subject adjustments u_1 and u_2 are parameters in the model, and therefore have priors defined on them.⁵ Parameters that appear in the prior specifications for parameters, such as τ_u , are often called *hyperparameters*,⁶ and the priors on such hyperparameters are called *hyperpriors*. Thus, the parameter u_1 has $Normal(0, \tau_{u_1})$ as a prior; τ_{u_1} is a hyperparameter, and the hyperprior on τ_{u_1} is $Normal(0, 20)$.⁷

We know that in general, in EEG experiments, 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 compari-

⁵By contrast, in the frequentist `lmer` model, the adjustments u_1 and u_2 are not parameters; they are called conditional modes; see Bates, Mächler, et al. (2015a).

⁶Another source of confusion here is that *hyperparameters* is also used in the machine learning literature with a different meaning.

⁷One could in theory keep going deeper and deeper, defining hyper-hyperpriors etc., but the model would quickly become impossible to fit.

son 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).

Box 5.2. 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:

$$\begin{aligned} \text{signal}_n &\sim \text{Normal}(\alpha_{\text{subj}[n]} + \beta_{\text{subj}[n]} \cdot \text{c_cloze}_n, \sigma) \\ \alpha_i &\sim \text{Normal}(\alpha, \tau_{u_1}) \\ \beta_i &\sim \text{Normal}(\beta, \tau_{u_2}) \end{aligned} \quad (5.7)$$

In fact, this is another common way to write the model.

- Why do the adjustments u have a normal distribution?

Mostly by convention, the adjustments u are assumed to come from a Normal distribution. Another reason is that if we don't know anything about the distribution besides its mean and variance, the normal distribution is the most conservative assumption (see 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_1 and u_2 ; this lack of correlation is indicated in `brms` using the double pipe `||`.⁸ In `brms`, we need to specify hyperpriors for τ_{u_1} and τ_{u_2} ; these are called `sd` in `brms`, to distinguish these standard deviations from the standard deviation of the residuals σ . As with the population-level effects, the by-subjects intercept adjustments are implicitly fit for the group-level effects and thus `(c_cloze || subj)` is equivalent to `(1 + c_cloze || subj)`. If we don't want an intercept we need to explicitly indicate it with `(0 + c_cloze || subj)` or

⁸The double pipe is also used in the same way in `lmer` from the package `lme4`.

$(-1 + c_cloze \mid\mid subj)$. Such a removal of the intercept is not normally done.

```
prior_v <-
  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 = subj),
    prior(normal(0, 20), class = sd, coef = c_cloze, group = subj)
  )
fit_N400_v <- brm(n400 ~ c_cloze + (c_cloze || subj),
  prior = prior_v,
  data = df_eeg
)
```

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_1} and τ_{u_2} 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`. The full summary can be printed out by typing:

```
fit_N400_v
```

Because the above command will result in some pages of output, it is easier to understand the summary graphically (Figure 5.8). Rather than the wrapper `plot()`, we use the original function of the package `bayesplot`, `mcmc_dens`, to only show density plots. We extract the first 6 parameters of the model with `variables(fit_N400_v)[1:6]`.

```
mcmc_dens(fit_N400_v, pars = variables(fit_N400_v)[1:6])
```

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 5.9. These are adjustments, $u_{1,1}, u_{1,2}, \dots, u_{1,37}$, and not the effect of the manipu-

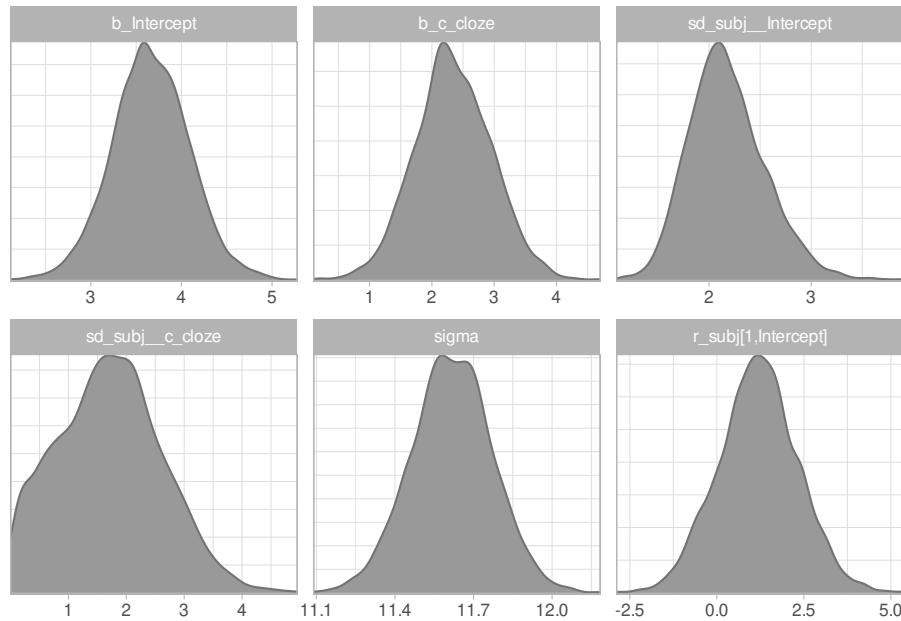


FIGURE 5.8: Posterior distributions of the parameters in the model fit_N400_v.

lation by subject, $\beta + [u_{1,1}, u_{1,2}, \dots, u_{1,37}]$. The code below produces Figure 5.9.

```
# make a table of u_2s
ind_effects_v <- paste0("r_subj[", unique(df_eeg$subj),
                         ",c_close]")
u_2_v <- posterior_summary(fit_N400_v, variable = ind_effects_v) %>%
  as_tibble() %>%
  mutate(subj = 1:n()) %>%
  ## reorder plot by magnitude of mean:
  arrange(Estimate) %>%
  mutate(subj = factor(subj, levels = subj))
# We plot:
ggplot(
  u_2_v,
  aes(x = Estimate, xmin = Q2.5, xmax = Q97.5, y = subj)
) +
```

```
geom_point() +
geom_errorbar() +
xlab("By-subject adjustment to the slope in 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. In Figure 5.10, we can see the shrinkage of the estimates in the varying intercepts model by comparing them with the estimates of the no pooling model (M_{np}). The code below produces Figure 5.10. This code is more involved since it requires us to build a data frame with the by-subject effects ($\beta + u_2$).

```
# Extract parameter estimates from the no pooling model:
par_np <- posterior_summary(fit_N400_np, variable = ind_effects_np) %>%
  as_tibble() %>%
  mutate(
    model = "No pooling",
    subj = unique(df_eeg$subj)
  )

# For the hierarchical model, the code is more complicated
# because we want the effect (beta) + adjustment.
# Extract the overall group level effect:
beta <- c(as_draws_df(fit_N400_v)$b_c_cloze)
# Extract the individual adjustments:
ind_effects_v <- paste0("r_subj[", unique(df_eeg$subj), ",c_cloze]")
adjustment <- as_draws_matrix(fit_N400_v, variable = ind_effects_v)
# Get the by subject effects in a data frame where each adjustment
# is in each column.
# Remove all the draws meta data by using as.data.frame
by_subj_effect <- as.data.frame(beta + adjustment)
# Summarize them by getting a table with the mean and the
```

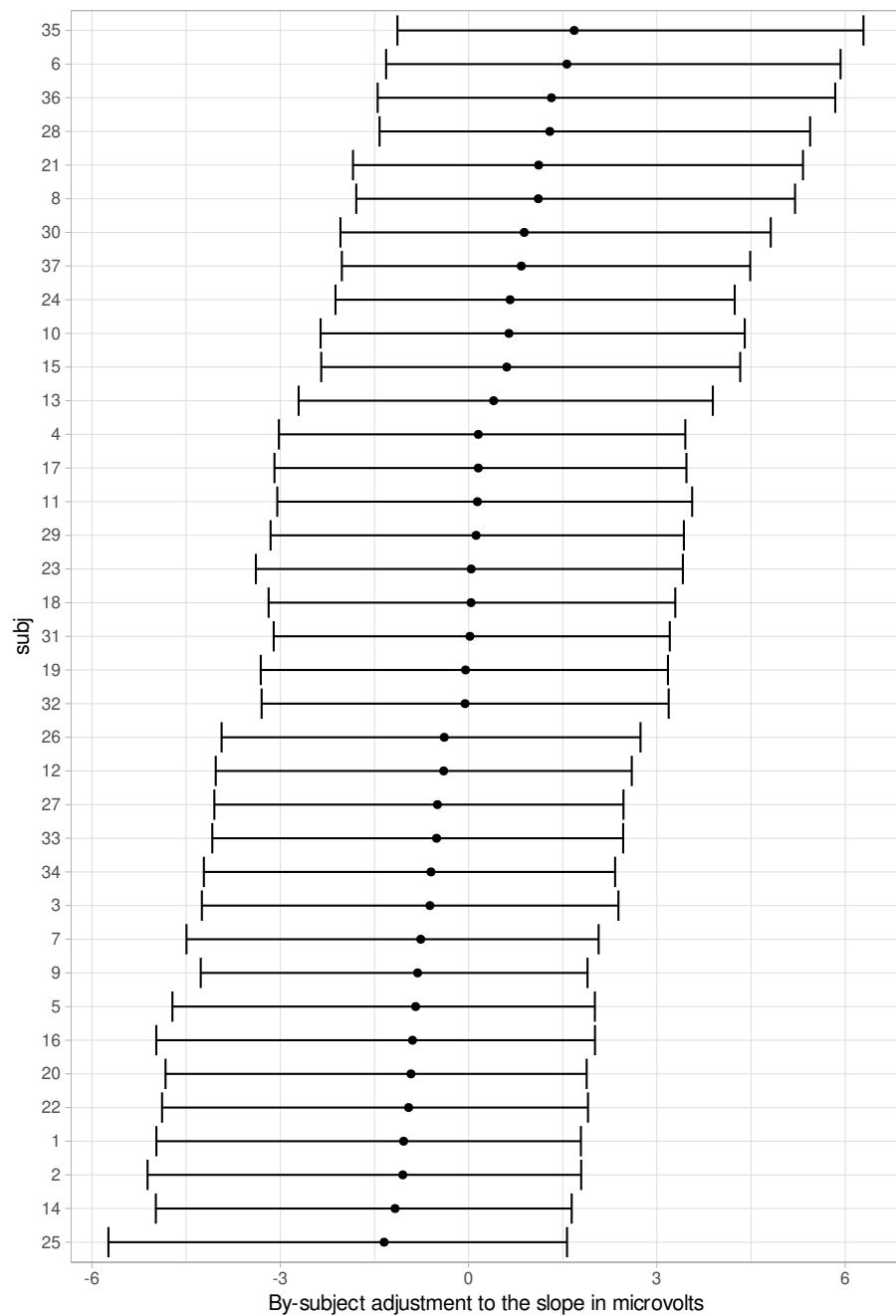


FIGURE 5.9: 95% credible intervals of adjustments to the effect of cloze probability for each subject ($u_{1,1..37}$) according to the varying intercept and varying slopes model, `fit_N400_v`.

```
# quantiles for each column and then binding them.
par_h <- lapply(by_subj_effect, function(x) {
  tibble(
    Estimate = mean(x),
    Q2.5 = quantile(x, .025),
    Q97.5 = quantile(x, .975)
  )
}) %>%
  bind_rows() %>%
  # Add a column to identify that the model,
  # and one with the subject labels:
  mutate(
    model = "Hierarchical",
    subj = unique(df_eeg$subj)
  )

# The mean and 95% CI of both models in one data frame:
by_subj_df <- bind_rows(par_h, par_np) %>%
  arrange(Estimate) %>%
  mutate(subj = factor(subj, levels = unique(.data$subj)))
```

```
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,
```

```

variable = "b_c_cloze")[, "Estimate"]
) +
geom_hline(
yintercept =
posterior_summary(fit_N400_v,
variable = "b_c_cloze")[, "Q2.5"],
linetype = "dotted", size = .5
) +
geom_hline(
yintercept =
posterior_summary(fit_N400_v,
variable = "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 intercepts (mean voltage) and slopes (effects of cloze) across subjects, but it has the implicit assumption that these varying intercepts and varying slopes 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 allows 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.

- 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 subject, and these two might be cor-

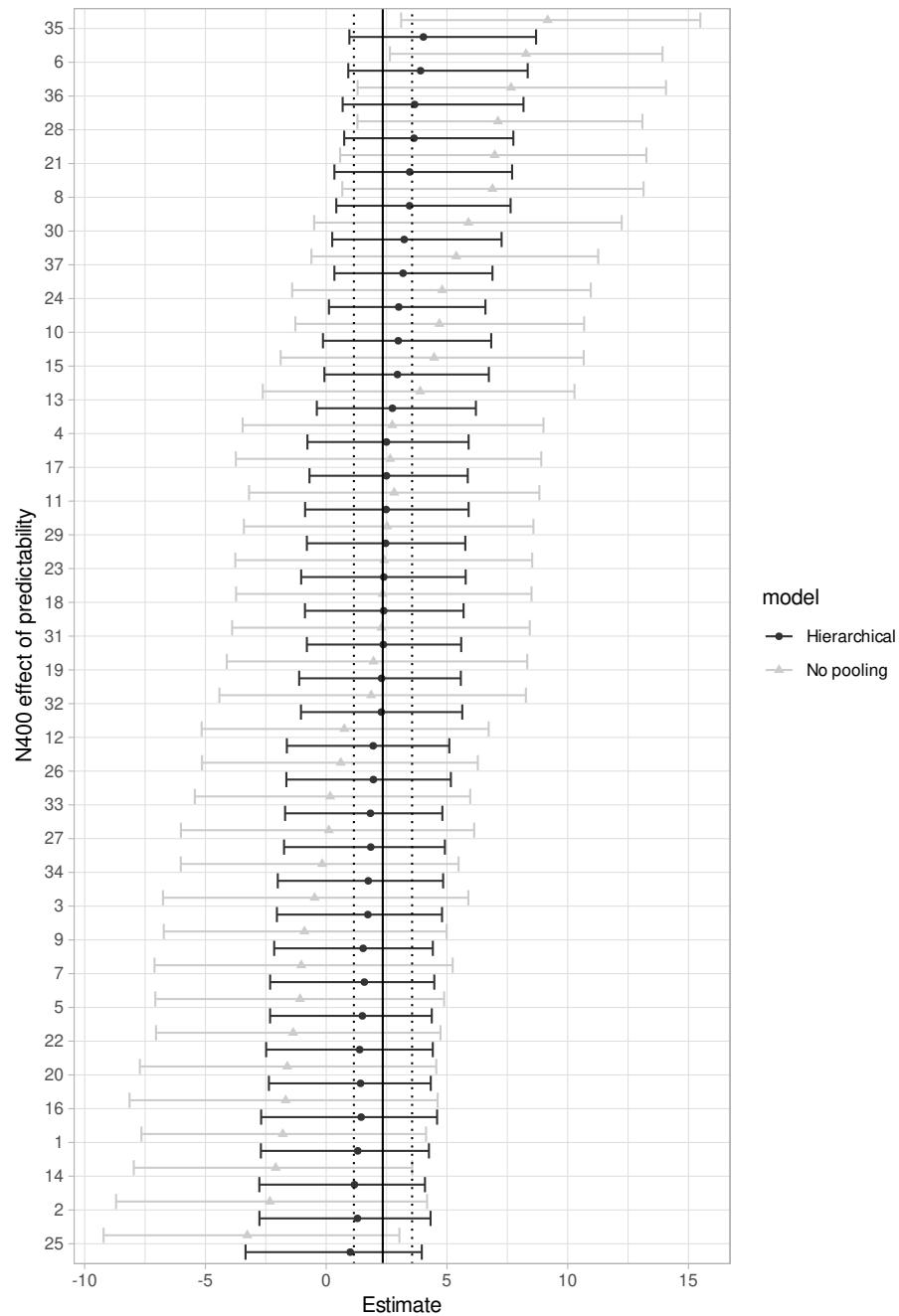


FIGURE 5.10: This plot compares the estimates of the effect of cloze probability for each subject between (i) the no pooling, `fit_N400_np` and (ii) the varying intercepts and varying slopes, hierarchical, model, `fit_N400_v`.

related, i.e., we assume group-level intercepts and slopes, and allow 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 M_v , which assumes no correlation between group-level intercepts and slopes (section 5.1.3):

$$\text{signal}_n \sim \text{Normal}(\alpha + u_{\text{subj}[n],1} + c_{\text{cloze}}_n \cdot (\beta + u_{\text{subj}[n],2}), \sigma) \quad (5.8)$$

The correlation is indicated in the priors on the adjustments for intercept u_1 and slopes u_2 .

- Priors:

$$\begin{aligned} \alpha &\sim \text{Normal}(0, 10) \\ \beta &\sim \text{Normal}(0, 10) \\ \sigma &\sim \text{Normal}_+(0, 50) \\ \begin{pmatrix} u_{i,1} \\ u_{i,2} \end{pmatrix} &\sim \mathcal{N} \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \boldsymbol{\Sigma}_u \right) \end{aligned} \quad (5.9)$$

In this model, a bivariate normal distribution generates the varying intercepts and varying slopes u ; this is an $n \times 2$ matrix. The variance-covariance matrix $\boldsymbol{\Sigma}_u$ defines the standard deviations of the varying intercepts and varying slopes, and the correlation between them. Recall from section 1.6.2 that the diagonals of the variance-covariance matrix contain the variances of the correlated random variables, and the off-diagonals contain the covariances. In this example, the covariance $\text{Cov}(u_1, u_2)$ between two variables u_1 and u_2 is defined as the product of their correlation ρ and their standard deviations τ_{u_1} and τ_{u_2} . In other words, $\text{Cov}(u_1, u_2) = \rho_u \tau_{u_1} \tau_{u_2}$.

$$\boldsymbol{\Sigma}_u = \begin{pmatrix} \tau_{u_1}^2 & \rho_u \tau_{u_1} \tau_{u_2} \\ \rho_u \tau_{u_1} \tau_{u_2} & \tau_{u_2}^2 \end{pmatrix} \quad (5.10)$$

In order to specify a prior for $\boldsymbol{\Sigma}_u$, we need priors for the standard deviations, τ_{u_1} and τ_{u_2} , 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 LKJ prior. The basic idea of the LKJ prior on the correlation matrix is that as its parameter, η (eta), increases, it

will favor correlations closer to zero.⁹ 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 through the wide spread of the prior between -1 and 1 . Thus, $\eta = 2$ gives us a regularizing, relatively uninformative or mildly informative prior.

Figure 5.11 shows a visualization of different parametrizations of the LKJ prior.

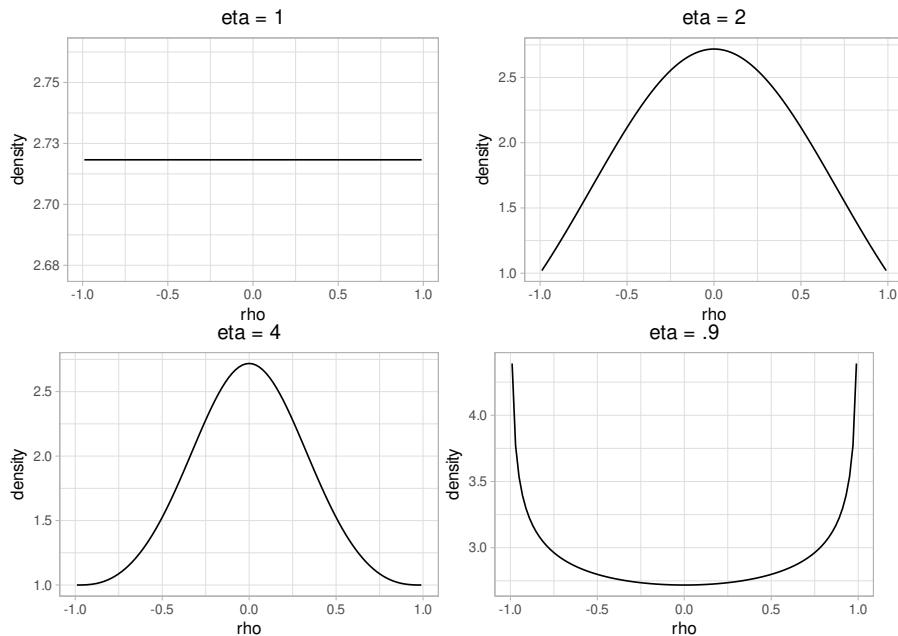


FIGURE 5.11: Visualization of the LKJ correlation distribution prior with four different values of the η parameter.

$$\begin{aligned}\tau_{u_1} &\sim Normal_+(0, 20) \\ \tau_{u_2} &\sim Normal_+(0, 20) \\ \rho_u &\sim LKJcorr(2)\end{aligned}\tag{5.11}$$

In our `brms` model, we allow a correlation between the by-subject inter-

⁹This is because an LKJ correlation distribution with a large η corresponds to a correlation matrix with values close to zero in the lower and upper triangles

cepts and slopes by using a single pipe | instead of the double pipe || that we used previously. This convention follows that in the frequentist `lmer` function. As before, the varying intercepts are implicitly fit.

Because we have a new parameter, the correlation ρ_u , we need to add a new prior for this correlation; in `brms`, this is achieved by addition a prior for the parameter type `cor`.

```
prior_h <- 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 = subj
  ),
  prior(normal(0, 20),
    class = sd, coef = c_cloze,
    group = subj
  ),
  prior(lkj(2), class = cor, group = subj)
)
fit_N400_h <- brm(n400 ~ c_cloze + (c_cloze | subj),
  prior = prior_h,
  data = df_eeg
)
```

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). As before we show the estimates graphically (Figure 5.12). One can access the complete summary as always with `fit_N400_h`.

```
plot(fit_N400_h, N = 6)
```

We are now half-way to what is called the maximal hierarchical model

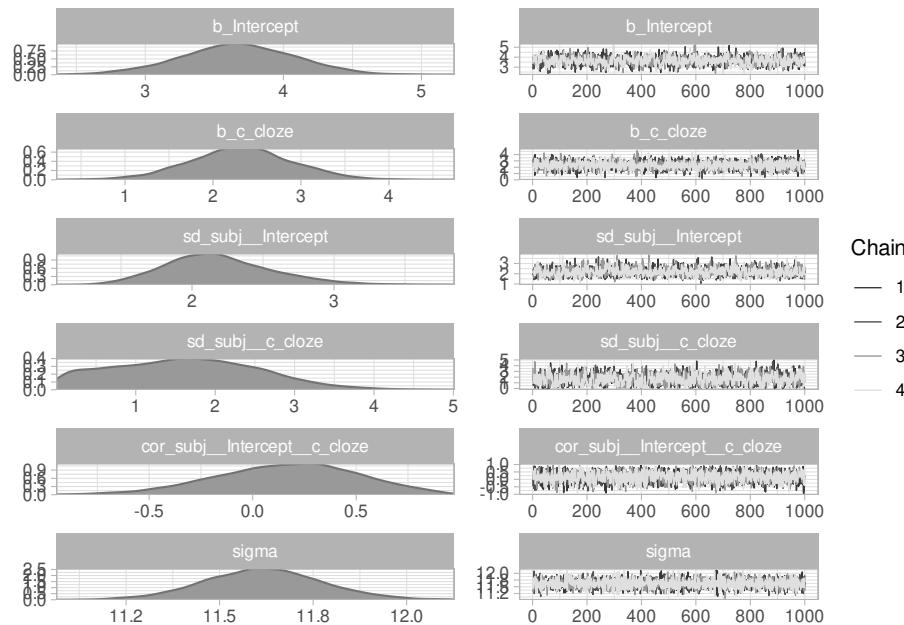


FIGURE 5.12: The posteriors of the parameters in the model `fit_N400_h`.

(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 intercepts (mean voltage) and slopes (effects of predictability) across subjects *and* across items. In typical Latin square designs, subjects and items are said to be *crossed random effects*—each subject sees exactly one instance of each item. 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 subject, 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 predictability 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:

$$\text{signal}_n \sim \text{Normal}(\alpha + u_{\text{subj}[n],1} + w_{\text{item}[n],1} + c_{\text{cloze}}_n \cdot (\beta + u_{\text{subj}[n],2} + w_{\text{item}[n],2}), \sigma) \quad (5.12)$$

- Priors:

$$\alpha \sim \text{Normal}(0, 10)$$

$$\beta \sim \text{Normal}(0, 10)$$

$$\sigma \sim \text{Normal}_+(0, 50)$$

$$\begin{pmatrix} u_{i,1} \\ u_{i,2} \end{pmatrix} \sim \mathcal{N} \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_u \right) \quad (5.13)$$

$$\begin{pmatrix} w_{j,1} \\ w_{j,2} \end{pmatrix} \sim \mathcal{N} \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_w \right)$$

We have added the index j , which represents each item, as we did with subjects; $\text{item}[n]$ indicates the item that corresponds to the observation in the n -th row of the data frame.

We have hyperparameters and hyperpriors as before:

$$\begin{aligned} \Sigma_u &= \begin{pmatrix} \tau_{u_1}^2 & \rho_u \tau_{u_1} \tau_{u_2} \\ \rho_u \tau_{u_1} \tau_{u_2} & \tau_{u_2}^2 \end{pmatrix} \\ \Sigma_w &= \begin{pmatrix} \tau_{w_1}^2 & \rho_w \tau_{w_1} \tau_{w_2} \\ \rho_w \tau_{w_1} \tau_{w_2} & \tau_{w_2}^2 \end{pmatrix} \end{aligned} \quad (5.14)$$

$$\begin{aligned}
 \tau_{u_1} &\sim \text{Normal}_+(0, 20) \\
 \tau_{u_2} &\sim \text{Normal}_+(0, 20) \\
 \rho_u &\sim \text{LKJcorr}(2) \\
 \tau_{w_1} &\sim \text{Normal}_+(0, 20) \\
 \tau_{w_2} &\sim \text{Normal}_+(0, 20) \\
 \rho_w &\sim \text{LKJcorr}(2)
 \end{aligned} \tag{5.15}$$

We set identical priors for by-items group-level effects as for by-subject group-level effects, but this only because we don't have any differentiated prior information about subject-level vs. item-level variation. 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:

```

prior_sih_full <-
  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 = subj
    ),
    prior(normal(0, 20),
      class = sd, coef = c_cloze,
      group = subj
    ),
    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)
)

fit_N400_sih <- brm(n400 ~ c_cloze + (c_cloze | subj) +
(c_cloze | item),
prior = prior_sih_full,
data = df_eeg
)

```

We can also simplify the call to `brms`, when we assign the same priors to the by-subject and by-item parameters:

```

prior_sih <-
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)
)

fit_N400_sih <- brm(n400 ~ c_cloze + (c_cloze | subj) +
(c_cloze | item),
prior = prior_sih,
data = df_eeg
)

```

We have new group-level effects in the summary, but again the estimate of the effect of cloze remains virtually unchanged (Figure 5.13).

```
fit_N400_sih
```

```
plot(fit_N400_sih, N = 9)
```

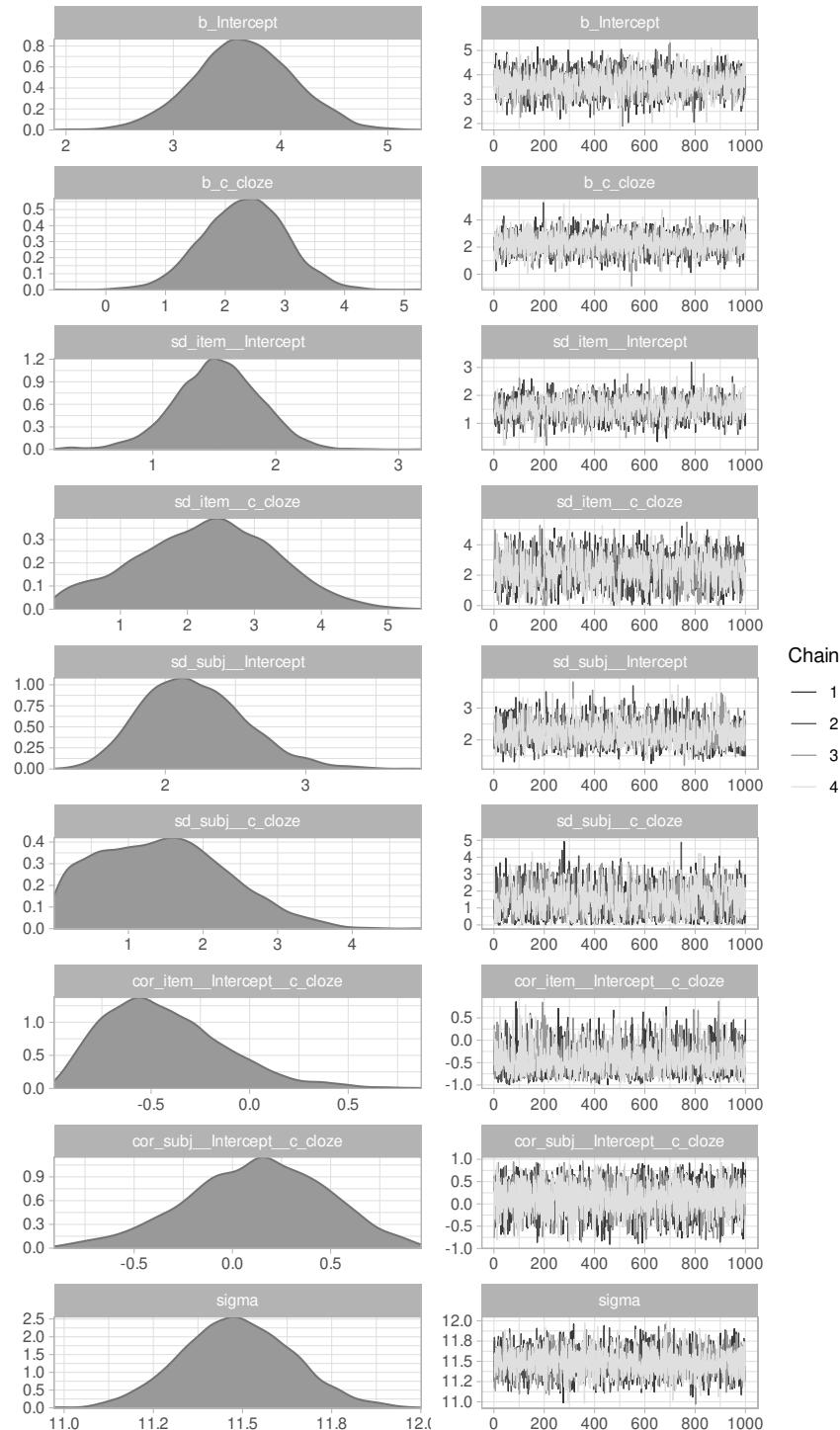


FIGURE 5.13: The posterior distributions of the parameters in the model fit_N400_si.h.

Box 5.3. The Matrix Formulation of Hierarchical Models (the Laird-Ware form)

We have been writing linear models as follows; where n refers to the row id in the data frame.

$$y_n \sim \text{Normal}(\alpha + \beta \cdot x_n) \quad (5.16)$$

This simple linear model can be re-written as follows:

$$y_n = \alpha + \beta \cdot x_n + \varepsilon_n \quad (5.17)$$

where $\varepsilon_n \sim \text{Normal}(0, \sigma)$.

The model does not change if α is multiplied by 1:

$$y_n = \alpha \cdot 1 + \beta \cdot x_n + \varepsilon_n \quad (5.18)$$

The above is actually n linear equations, and can be written compactly in matrix form:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix} \quad (5.19)$$

Consider this matrix in the above equation:

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \quad (5.20)$$

This matrix is called the model matrix or the design matrix; we will encounter it again in the contrast coding chapters, where it plays a crucial role. If we write the dependent variable y as a $n \times 1$ vector, the

above matrix as the matrix X (which has dimensions $n \times 2$, the intercept and slope parameters as a 2×1 matrix ζ , and the residual errors as an $n \times 1$ matrix, we can write the linear model very compactly:

$$y = X\zeta + \epsilon \quad (5.21)$$

The above matrix formulation of the linear model extends to the hierarchical model very straightforwardly. For example, consider the model M_{sih} that we just saw above. This model has the following likelihood:

$$\begin{aligned} signal_n \sim Normal(\alpha + u_{subj[n],1} + w_{item[n],1} \\ + c_cloze_n \cdot (\beta + u_{subj[n],2} + w_{item[n],2}), \sigma) \end{aligned} \quad (5.22)$$

The terms in the location parameter in the Normal likelihood can be re-written in matrix form, just like the linear model above. To see this, consider the fact that the location term

$$\alpha + u_{subj[n],1} + w_{item[n],1} + c_cloze_n \cdot (\beta + u_{subj[n],2} + w_{item[n],2}) \quad (5.23)$$

can be re-written as

$$\begin{aligned} \alpha \cdot 1 + u_{subj[n],1} \cdot 1 + w_{item[n],1} \cdot 1 + \\ \beta \cdot c_cloze_n + u_{subj[n],2} \cdot c_cloze_n + w_{item[n],2} \cdot c_cloze_n \end{aligned} \quad (5.24)$$

The above equation can in turn be written in matrix form:

$$\begin{aligned}
 & \begin{pmatrix} 1 & c_cloze_1 \\ 1 & c_cloze_2 \\ \vdots & \vdots \\ 1 & c_cloze_n \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} + \\
 & \begin{pmatrix} 1 & c_cloze_1 \\ 1 & c_cloze_2 \\ \vdots & \vdots \\ 1 & c_cloze_n \end{pmatrix} \begin{pmatrix} u_{subj[1],1} & u_{subj[2],1} & \dots & u_{subj[n],1} \\ u_{subj[1],2} & u_{subj[2],2} & \dots & u_{subj[n],2} \end{pmatrix} + \quad (5.25) \\
 & \begin{pmatrix} 1 & c_cloze_1 \\ 1 & c_cloze_2 \\ \vdots & \vdots \\ 1 & c_cloze_n \end{pmatrix} \begin{pmatrix} w_{item[1],1} & w_{item[2],1} & \dots & w_{item[n],1} \\ w_{item[1],2} & w_{item[2],2} & \dots & w_{item[n],2} \end{pmatrix}
 \end{aligned}$$

In this hierarchical model, there are three model matrices:

- the matrix associated with the intercept α and the slope β ; below, we call this the matrix X .
- the matrix associated with the by-subject varying intercepts and slopes; call this the matrix Z_u .
- the matrix associated with the by-item varying intercepts and slopes; call this the matrix Z_w .

The model can now be written very compactly in matrix form by writing these three matrices as follows:

$$\begin{aligned}
 X &= \begin{pmatrix} 1 & c_cloze_1 \\ 1 & c_cloze_2 \\ \vdots & \vdots \\ 1 & c_cloze_n \end{pmatrix} \\
 Z_u &= \begin{pmatrix} 1 & c_cloze_1 \\ 1 & c_cloze_2 \\ \vdots & \vdots \\ 1 & c_cloze_n \end{pmatrix} \\
 Z_w &= \begin{pmatrix} 1 & c_cloze_1 \\ 1 & c_cloze_2 \\ \vdots & \vdots \\ 1 & c_cloze_n \end{pmatrix}
 \end{aligned} \tag{5.26}$$

The location part of the model M_{sih} can now be written very compactly:

$$X\zeta + Z_u z_u + Z_w z_w \tag{5.27}$$

Here, ζ is a 2×1 matrix containing the intercept α and the slope β , and z_u and z_w are the intercept and slope adjustments by subject and by item:

$$\begin{aligned}
 z_u &= \begin{pmatrix} u_{subj[1],1} & u_{subj[2],1} & \dots & u_{subj[n],1} \\ u_{subj[1],2} & u_{subj[2],2} & \dots & u_{subj[n],2} \end{pmatrix} \\
 z_w &= \begin{pmatrix} w_{item[1],1} & w_{item[2],1} & \dots & w_{item[n],1} \\ w_{item[1],2} & w_{item[2],2} & \dots & w_{item[n],2} \end{pmatrix}
 \end{aligned} \tag{5.28}$$

In summary, the hierarchical model has a very general matrix formulation, called the Laird-Ware form (Laird and Ware 1982):

$$signal = X\zeta + Z_u z_u + Z_w z_w + \epsilon \tag{5.29}$$

The practical relevance of this matrix formulation is that we can define hierarchical models very compactly and efficiently in Stan by expressing the model in terms of the model matrices (Sorensen, Ho-

henstein, and Vasishth 2016). As an aside, notice that in the above example, $X = Z_u = Z_w$; but in principle one could have different model matrices for the fixed vs. random effects.

5.1.6 Beyond the maximal model—Distributional regression models

We can use posterior predictive checks to verify that our last model can capture the entire signal distribution. This is shown in Figure 5.14

```
pp_check(fit_N400_si, ndraws = 50, type = "dens_overlay")
```

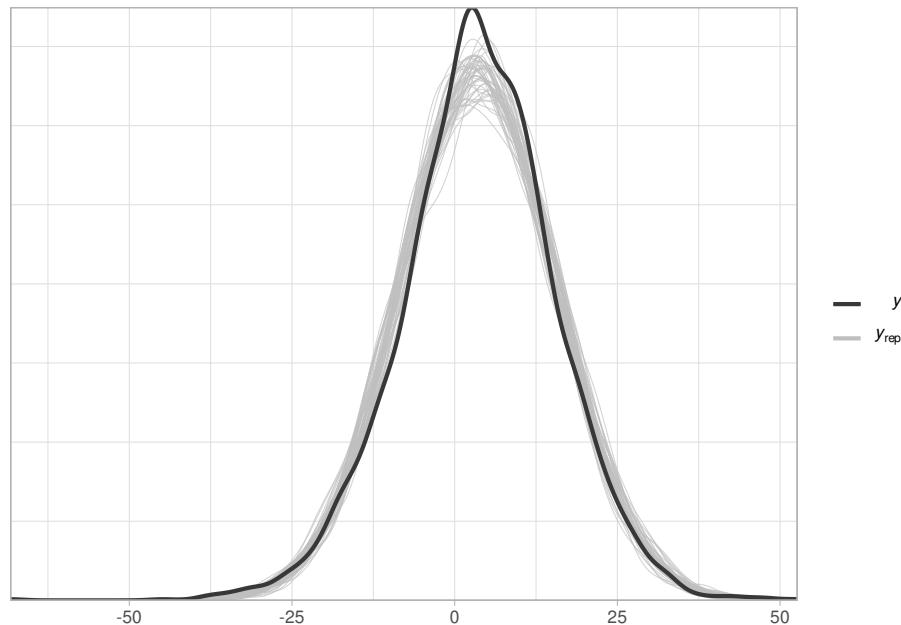


FIGURE 5.14: Overlay of densities from the posterior predictive distributions of the model `fit_N400_si`.

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 (Picton et al. 2000). Given that skin tissue is different between subjects, it could be the case that the level of noise varies by subject. It might be a good idea to verify that our model is good enough for capturing the by-subject variability. The code below produces Figure 5.15.

```
ppc_dens_overlay_grouped(df_eeg$n400,
  yrep =
    posterior_predict(fit_N400_si,
      ndraws = 100
    ),
  group = df_eeg$subj
) +
  xlab("Signal in the N400 spatiotemporal window")
```

Figure 5.15 hints that we might be misfitting some subjects: Some of the by-subject observed distributions of the EEG signal averages look much tighter than their corresponding posterior predictive distributions (e.g., subjects 3, 5, 9, 10, 14), whereas some other by-subject observed distributions look wider (e.g., subjects 25, 26, 27). Another approach to examine whether we misfit the by-subject noise level is to plot posterior distributions of the standard deviations and compare them with the observed standard deviation. This is achieved in the following code, which groups the data by subject, and shows the distribution of standard deviations. The result is shown in Figure 5.16. It is clear now that, for some subjects, the observed standard deviation lies outside the distribution of predictive standard deviations.

```
pp_check(fit_N400_si,
  type = "stat_grouped",
  ndraws = 1000,
  group = "subj",
  stat = "sd"
)
```

Why is our “maximal” hierarchical model misfitting the by-subject distribution of data? This is because, the 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, and are assumed to be constant across observations and clusters. This assumption is so common that researchers may not be aware that

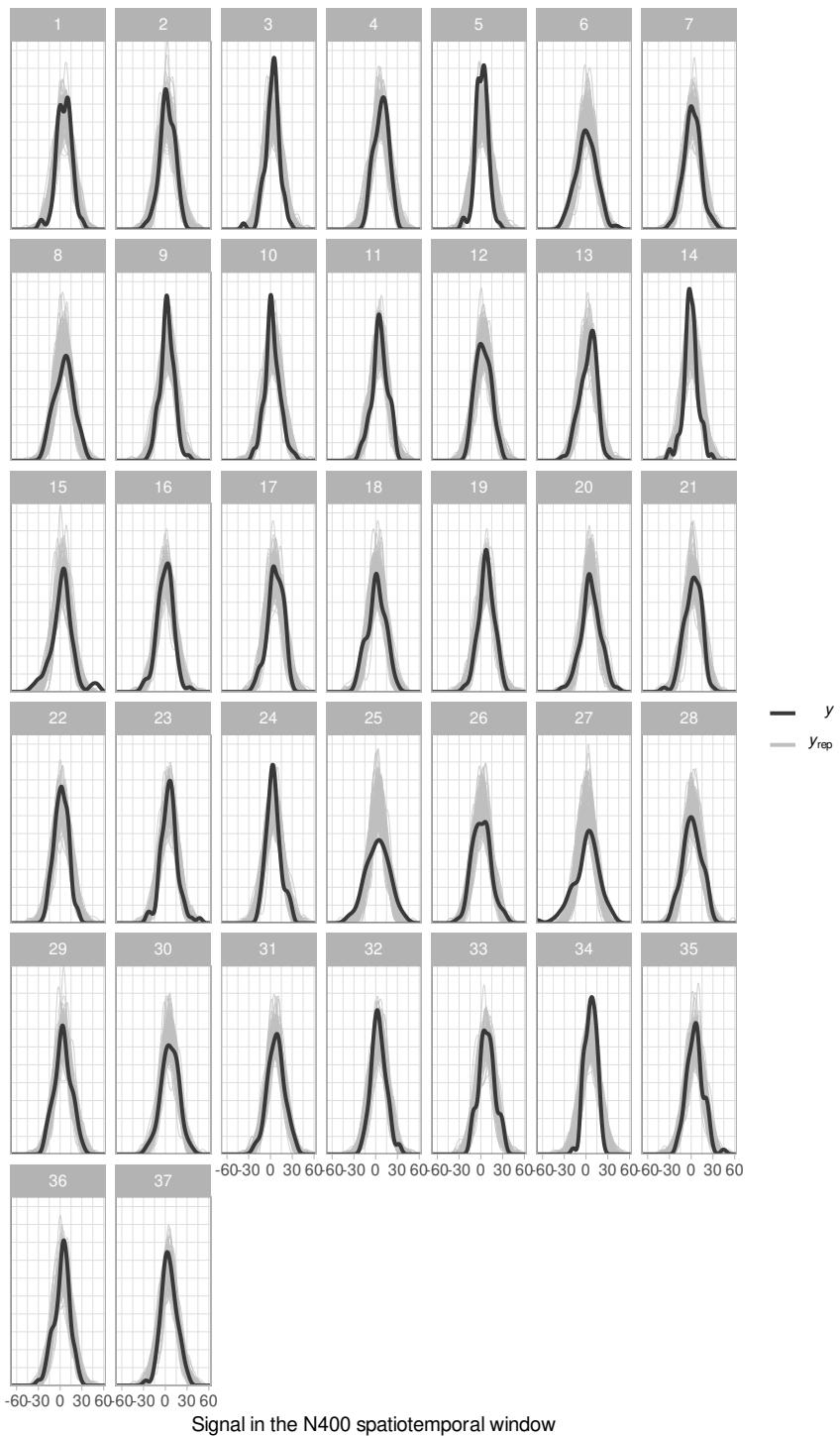


FIGURE 5.15: The plot shows 100 predicted distributions with the label y_{rep} and the distribution of the average signal data with the label y density plots for the 37 subjects that participated in the experiment.

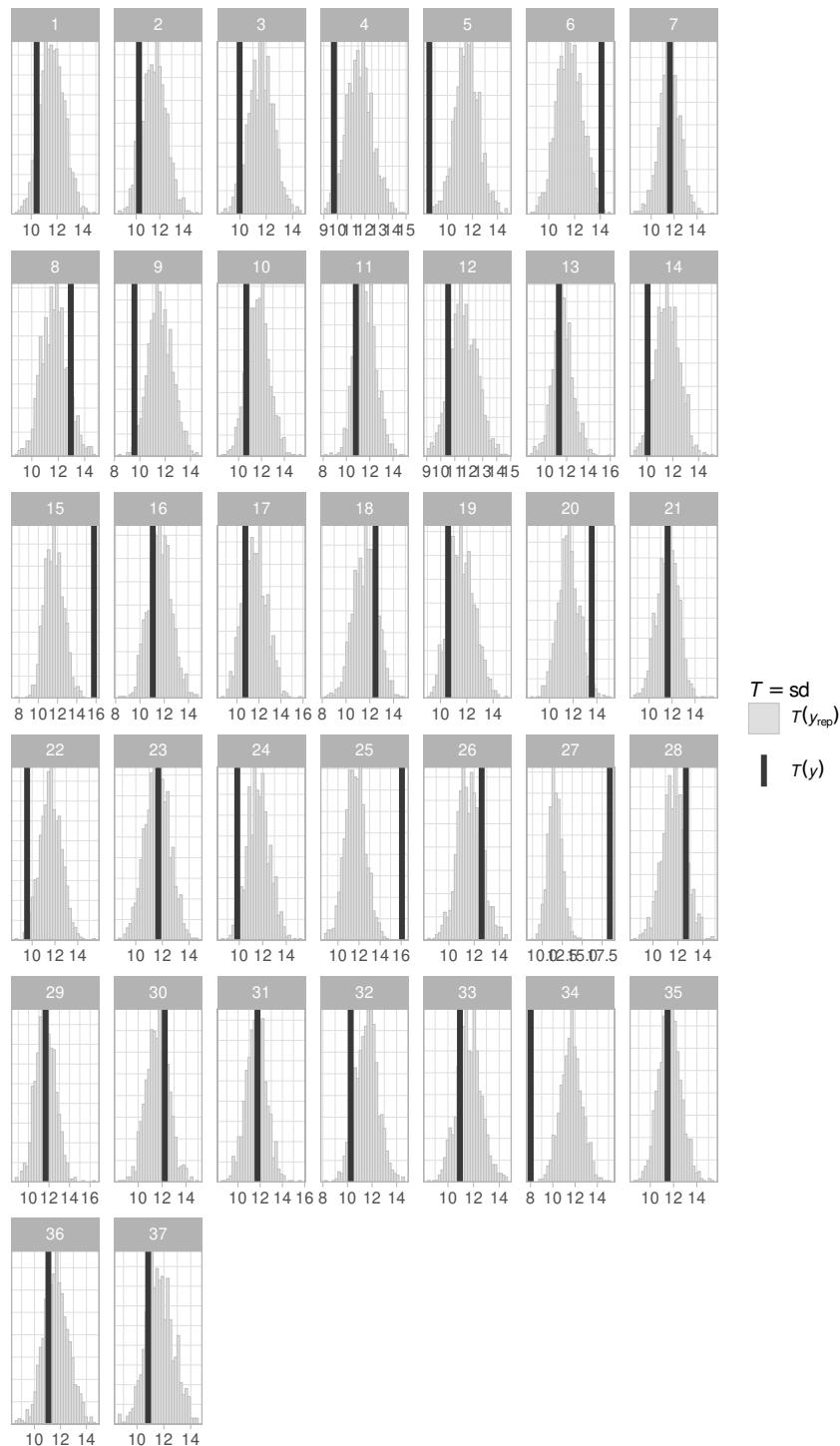


FIGURE 5.16: Distribution of posterior predicted standard deviations in gray and observed standard deviation in black lines by subject.

it is just an assumption. In the Bayesian framework, it is easy to change such default assumptions if necessary. Changing the assumption that all subjects have the same residual standard deviation leads to the distributional regression model. Such models can be fit in `brms`.¹⁰.

We are going to change our previous likelihood, so that the scale, σ has also a group-level effect structure. We exponentiate σ to make sure that the negative adjustments do not cause σ to become negative.

$$\begin{aligned} signal_n &\sim Normal(\alpha + u_{subj[n],1} + w_{item[n],1} + \\ &c_cloze_n \cdot (\beta + u_{subj[n],2} + w_{item[n],2}), \sigma_n) \quad (5.30) \\ \sigma_n &= \exp(\sigma_\alpha + \sigma_{u_{subj[n]}}) \end{aligned}$$

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 rather uninformative hyperpriors. Recall that everything is exponentiated when it goes inside the likelihood; that is why we use $\log(50)$ rather than 50 in σ .

$$\begin{aligned} \sigma_\alpha &\sim Normal(0, \log(50)) \\ \sigma_u &\sim Normal(0, \tau_{\sigma_u}) \quad (5.31) \\ \tau_{\sigma_u} &\sim Normal_+(0, 5) \end{aligned}$$

This model can be fit in `brms` using the internal function `brmsformula()`. This is a powerful function that extends the formulas that we used so far allowing for setting a hierarchical regression to any parameter of a model. This will allow us to set a by-subject hierarchical structure to the parameter σ . We also need to set new priors; these priors are identified by `dpar = sigma`.

```
prior_s <- c(
  prior(normal(0, 10), class = Intercept),
  prior(normal(0, 10), class = b),
```

¹⁰shorturl.at/brNS8

```

prior(normal(0, 20), class = sd),
prior(lkj(2), class = cor),
prior(normal(0, log(50)), class = Intercept, dpar = sigma),
prior(normal(0, 5),
      class = sd, group = subj,
      dpar = sigma
    )
)
fit_N400_s <- brm(brmsformula(
  n400 ~ c_cloze + (c_cloze | subj) + (c_cloze | item),
  sigma ~ 1 + (1 | subj)
),
prior = prior_s,
data = df_eeg
)

```

Inspect the output below; notice that our estimate for the effect of cloze remains very similar to that of the model `fit_N400_sih`.

Compare the two models' estimates:

```
posterior_summary(fit_N400_sih, variable = "b_c_cloze")
```

```
##           Estimate Est.Error Q2.5 Q97.5
## b_c_cloze     2.32     0.688  0.972  3.65
```

```
posterior_summary(fit_N400_s, variable = "b_c_cloze")
```

```
##           Estimate Est.Error Q2.5 Q97.5
## b_c_cloze     2.3       0.66  1.01  3.61
```

Nonetheless, Figure 5.17 shows that the fit of the model with respect to the by-subject variability is much better than before. Furthermore, Figure 5.18 shows that the observed standard deviations for each subject are well inside the posterior predictive distributions. The code below produces Figure 5.17.

```
ppc_dens_overlay_grouped(df_eeg$n400,
  yrep =
    posterior_predict(fit_N400_s,
      ndraws = 100
    ),
  group = df_eeg$subj
) +
  xlab("Signal in the N400 spatiotemporal window")
```

The model `fit_N400_s` raises the question: how much structure should we add to our statistical model? Should we also assume that σ can vary by items, and also by our experimental manipulation? Should we also have a maximal model 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 (we should include 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 the computations in a feasible time frame), and (iv) our domain knowledge.

Ultimately, all models are approximations (that's in the best case; often, they are plainly wrong) and we need to think carefully about which aspects 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 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 oversimplification does have an 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. For example, we can define “hybrid” models such as the linear ballistic accumulator (Brown and Heathcote 2008; and see Nicenboim 2018 for an implementation in Stan), where a great deal of cog-

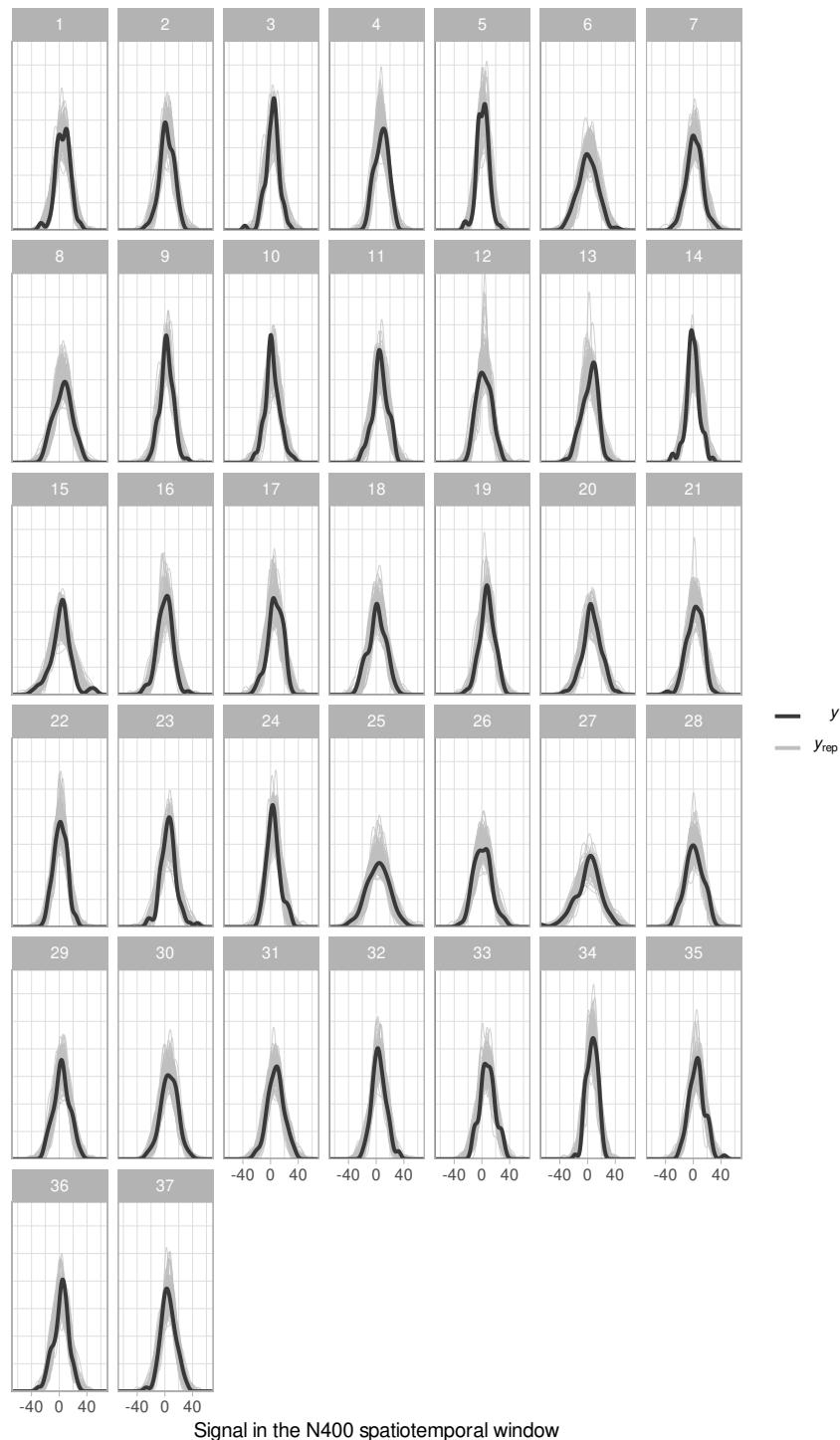


FIGURE 5.17: The gray density plots show 100 predicted distributions from a model that includes a hierarchical structure for σ . The black density plots show the distribution of the average signal data for the 37 subjects in the experiment.

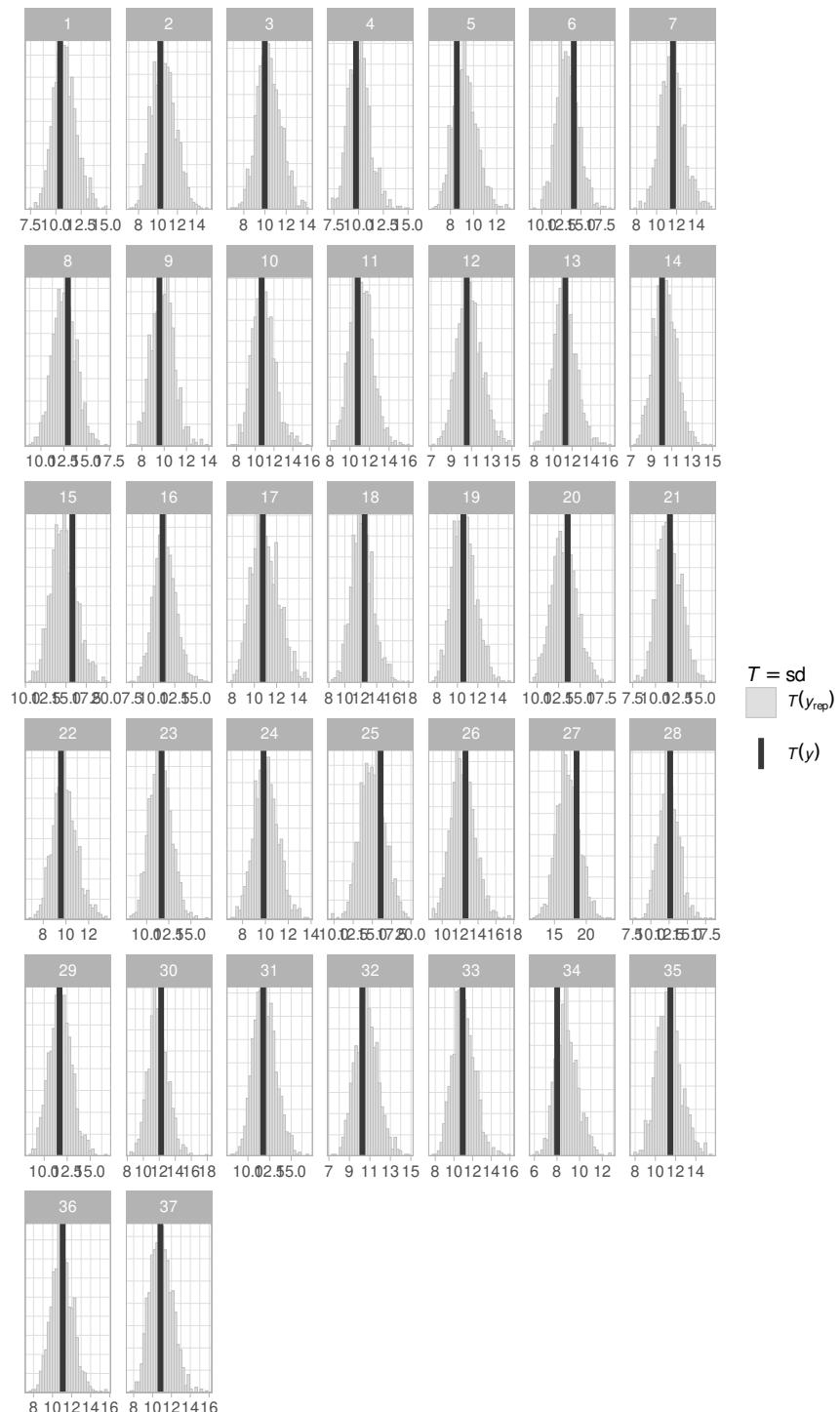


FIGURE 5.18: The gray lines show the distributions of posterior predicted standard deviations from a model that includes a hierarchical structure for σ , and observed mean standard deviations by subject (black vertical lines).

nitive 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 A hierarchical log-normal model: The Stroop effect

Next, using data from Ebersole et al. (2016), we illustrate some of the issues that arise with a log-normal likelihood in a hierarchical model. The data are from a Stroop task (Stroop 1935; for a review, see MacLeod 1991). We will analyze a subset of the data of 3337 subjects that participated in one variant of the Stroop task; this was part of a battery of tasks run in Ebersole et al. (2016).

For this variant of the Stroop task, subjects were presented with one word at the center of the screen (“red”, “blue”, or “green”). The word was written in either red, blue, or green color. In one third of the trials, the word matched the color of the text (“congruent” condition); and in the rest of the trials it did not match (“incongruent” condition). Subjects were instructed to only pay attention to the color that the word was written in, and press 1 if the color was red, 2 if it was blue, and 3 if it was green. In the incongruent condition, it is difficult to identify the color when it mismatches the word that is written on the screen. For example, it is hard to respond that the color is blue if the word written on the screen is green but the color it is presented in is blue; naming the color blue here is difficult in comparison to a baseline condition (the congruent condition), in which the word green appears in the color green. This increased difficulty in the incongruent condition is called the Stroop effect; the effect is extremely robust across variations in the task.

This task yields two measures: the accuracy of the decision made, and the time it took to respond. For the Stroop task, accuracy is usually almost at ceiling; to simplify the model, we will ignore accuracy. For a cognitive model that incorporates accuracy and response times into a model to analyze these Stroop data, see Nicenboim (2018).

5.2.1 A correlated varying intercept varying slopes log-normal model

If our theory only focuses on the difference between the response times for the “congruent” vs. “incongruent” condition, we can ignore the actual color presented and the word that was written. We can simply focus on whether a trial was congruent or incongruent. Define a predictor `c_cond` to represent these two conditions. For simplicity, we will also assume that all subjects share the same variance (as we saw in section 5.1.6, changing this assumption leads to distributional regression models).

The above assumptions mean that we are going to fit the data with the following likelihood. The likelihood function is identical to the one that we fit in section 5.1.4, except that here the location and scale are embedded in a log-normal likelihood rather than a normal likelihood. Equation (5.32) states that we are dealing with a hierarchical model with by-subjects varying intercepts and varying slopes model:

$$rt_n \sim \text{LogNormal}(\alpha + u_{subj[n],1} + c_cond_n \cdot (\beta + u_{subj[n],2}), \sigma) \quad (5.32)$$

In chapter 8, we will discuss the sum-contrast coding of the two conditions (`c_cond`). For now, it suffices to say that we assign a `+1` to `c_cond` for the “incongruent” condition, and a `-1` for the “congruent” condition (i.e., a sum-contrast coding). Under this contrast coding, if the posterior mean of the parameter β turns out to be positive, that would mean that the model predicts that the incongruent condition has slower reaction times than the congruent one. This is because on average the location of the log-normal likelihood for each condition will be as follows. In Equation (5.33), $\mu_{incongruent}$ refers to the mean reaction time of the incongruent condition, and $\mu_{congruent}$ refers to the mean reaction time of the congruent condition, and $\mu_{congruent}$ to the mean reaction time of the congruent condition.

$$\begin{aligned} \mu_{incongruent} &= \alpha + 1 \cdot \beta \\ \mu_{congruent} &= \alpha + -1 \cdot \beta \end{aligned} \quad (5.33)$$

We could have chosen to do the opposite contrast coding assignments: `-1` for the incongruent condition, and `+1` for congruent condition. In that case, if the posterior mean of the parameter β turns out to be positive, that would mean that the incongruent condition has a *faster* reaction time than

the congruent condition. Given that the Stroop effect is very robust, we do not expect such an outcome. In order to make the β parameter's mean easier to interpret, we have chosen the contrast coding where a positive sign on the mean of β implies that the incongruent condition has slower reaction times.

As always, we need priors for all the parameters in our model. For the population-level parameters (or fixed effects), we use the same priors as we did when we were fitting a regression with a log-normal likelihood in section 3.5.3.

$$\begin{aligned}\alpha &\sim \text{Normal}(6, 1.5) \\ \beta &\sim \text{Normal}(0, 0.01) \\ \sigma &\sim \text{Normal}_+(0, 1)\end{aligned}\tag{5.34}$$

Here, β represents, on the log scale, the change in the intercept α as a function of the experimental manipulation. In this model, β will probably be larger in magnitude than for the model that examined the difference in pressing the spacebar for two consecutive trials in section 3.5.3. We might need to examine the prior for β with predictive distributions, but we will delay this for now.

In contrast to our previous models, the intercept α is not the grand mean of the location. This is because the conditions were not balanced in the experiment (one-third of the conditions were congruent and two-thirds incongruent). The intercept could be interpreted here as the time (in log-scale) it takes to respond if we ignore the experimental manipulation.

Next, we turn our attention to the prior specification for the group-level parameters (or random effects). If we assume a possible correlation between by-subject intercepts and slopes, our model will have the following structure. In particular, we have to define priors for the parameters in the variance-covariance matrix Σ_u .

$$\begin{pmatrix} u_{i,1} \\ u_{i,2} \end{pmatrix} \sim \mathcal{N} \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_u \right)\tag{5.35}$$

$$\Sigma_u = \begin{pmatrix} \tau_{u_1}^2 & \rho_u \tau_{u_1} \tau_{u_2} \\ \rho_u \tau_{u_1} \tau_{u_2} & \tau_{u_2}^2 \end{pmatrix}\tag{5.36}$$

In practice, this means that we need priors for the by-subject standard deviations and correlations. For the variance components, we will set a similar prior as for σ . We don't expect the by-group adjustments to the intercept and slope to have more variance than the within-subject variance, so this prior will be quite conservative because they allow a large range of prior uncertainty. We assign the same prior for the correlations as we did in section 5.1.5.

$$\begin{aligned}\tau_{u_1} &\sim \text{Normal}_+(0, 1) \\ \tau_{u_2} &\sim \text{Normal}_+(0, 1) \\ \rho_u &\sim \text{LKJcorr}(2)\end{aligned}\tag{5.37}$$

We are now ready to fit the model. To speed up computation, we subset 50 subjects of the original data set; both the subsetted data and the original data set can be found in the package `bcogsci`. If we were analyzing these data for publication in a journal article or the like, we would obviously not subset the data.

We restrict ourselves to the correct trials only, and add a `c_cond` predictor, sum-coded as described earlier.

```
data("df_stroop")
(df_stroop <- df_stroop %>%
  mutate(c_cond = if_else(condition == "Incongruent", 1, -1)))

## # A tibble: 3,058 x 5
##   subj trial condition      RT c_cond
##   <dbl> <int> <chr>     <int>  <dbl>
## 1     1     1 Congruent    1484    -1
## 2     1     1 Incongruent  1316     1
## 3     1     2 Incongruent  628      1
## # ... with 3,055 more rows
```

Fit the model with 4000 iterations rather than with the default 2000 iterations by chain. If we were to run the model with the default number of iterations, the following warning would appear: `Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and media`

Running the chains for more iterations may help. See <http://mc-stan.org/misc/warnings.html#bulk-ess>.

```
fit_stroop <- brm(RT ~ c_cond + (c_cond | subj),
  family = lognormal(),
  prior =
  c(
    prior(normal(6, 1.5), class = Intercept),
    prior(normal(0, .01), class = b),
    prior(normal(0, 1), class = sigma),
    prior(normal(0, 1), class = sd),
    prior(lkj(2), class = cor)
  ),
  iter = 4000,
  data = df_stroop
)
```

We will focus on β (but you can verify that there is nothing surprising in the other parameters in the model `fit_stroop`).

```
posterior_summary(fit_stroop, variable = "b_c_cond")
```

```
##           Estimate Est.Error   Q2.5   Q97.5
## b_c_cond     0.027    0.00552  0.0157  0.0374
```

As shown in Figure 5.19, if we overlay the density plots for the prior and posterior distributions of β , it becomes evident that the prior might have been too restrictive: the posterior is relatively far from the prior, and the prior strongly down-weights the values that the posterior is centered around. Such a strong discrepancy between the prior and posterior can be investigated with a sensitivity analysis.

```
sample_b_post <- as_draws_df(fit_stroop)$b_c_cond
# We generate samples from the prior as well:
N <- length(sample_b_post)
sample_b_prior <- rnorm(N, 0, .01)
samples <- tibble(
```

```

sample = c(sample_b_post, sample_b_prior),
distribution = c(rep("posterior", N), rep("prior", N))
)
ggplot(samples, aes(x = sample, fill = distribution)) +
  geom_density(alpha = .5)

```

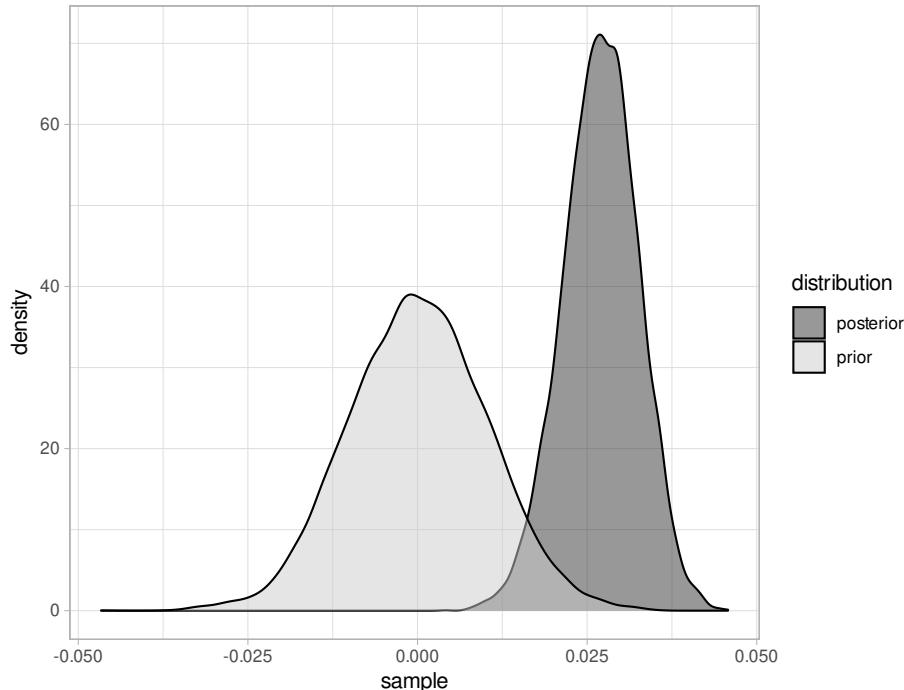


FIGURE 5.19: The discrepancy between the prior and the posterior distributions for the slope parameter in the model `fit_stroop`.

5.2.1.1 Sensitivity analysis

Here, the discrepancy evident in Figure 5.19 is investigated with a sensitivity analysis. We will examine what happens for the following priors for β . In the models we fit below, all the other parameters have the same priors as in the model `fit_stroop`; we vary only the priors for β . The different priors are:

- $\beta \sim \text{Normal}(0, 0.05)$
- $\beta \sim \text{Normal}(0, 0.1)$

TABLE 5.1: The summary (mean and 95% credible interval) for the posterior distribution of the slope in the model fit_stroop, given different priors on the slope parameter.

prior	Estimate	Q2.5	Q97.5
normal(0, 0.001)	0.001	-0.001	0.003
normal(0, 0.01)	0.027	0.016	0.037
normal(0, 0.05)	0.037	0.025	0.049
normal(0, 0.1)	0.037	0.025	0.049
normal(0, 1)	0.037	0.025	0.049
normal(0, 2)	0.038	0.026	0.050

- $\beta \sim Normal(0, 1)$
- $\beta \sim Normal(0, 2)$

We can summarize the estimates of β given different priors as shown in Table 5.1:

It might be easier to see how much the posterior difference between conditions changes depending on the prior. In order to answer this question, we need to remember that the median difference between conditions ($MedianRT_{diff}$) can be calculated as the difference between the exponents of each condition's medians:

$$\begin{aligned} MedianRT_{diff} &= MedianRT_{incongruent} - MedianRT_{congruent} \\ MedianRT_{diff} &= \exp(\alpha + \beta) - \exp(\alpha - \beta) \end{aligned} \quad (5.38)$$

Equation (5.38) gives us the posterior distributions of the median difference between conditions for the different models. We calculate the median difference rather than the mean difference because the mean depends on the parameter σ , but the median doesn't: The mean of a log-normal distribution is $\exp(\mu + \sigma^2/2)$, while the median is simply $\exp(\mu)$, see also 3.5.3.

Table 5.2 summarizes the posterior distributions under different priors using the means of the difference in medians, along with 95% credible intervals. It's important to realize that the use of mean to summarize the posterior distribution is orthogonal to our use of the median to summa-

TABLE 5.2: A summary, under a range of priors, of the posterior distributions of the mean difference between the two conditions, back-transformed to the millisecond scale.

prior	mean diff (ms)	Q2.5	Q97.5
normal(0, 0.001)	0.645	-1.51	2.77
normal(0, 0.01)	30.098	18.01	41.84
normal(0, 0.05)	41.458	27.68	55.42
normal(0, 0.1)	41.930	27.90	55.55
normal(0, 1)	41.980	28.14	55.82
normal(0, 2)	42.042	27.92	56.35

rize the response times by condition: In the first case, we use the median to summarize a group of *observations*, and in the second case, we use the mean to summarize a group of *samples* from the posterior—we could have summarized the samples from the posterior with its median instead of the mean.

Table 5.2 shows us that the posterior changes substantially when we use wider priors. It seems that the posterior is relatively unaffected when we use priors with a standard deviation larger than 0.05. However, if we assume a priori that the effect of the manipulation *must* be small, we will end up obtaining a posterior that is consistent with that belief. When we include less information about the possible effect sizes by using a less informative prior, we allow the data to influence the posterior more. A sensitivity analysis is always an important component of a good-quality Bayesian analysis.

Which analysis should one report after carrying out a sensitivity analysis? In the above example, the priors ranging from Normal(0,0.05) to Normal(0,2) show rather similar posterior distributions for the mean difference. The most common approach in Bayesian analysis is to report the results of such relatively uninformative priors (e.g., one could report the posterior associated with the Normal(0,2) here), because this kind of prior allows for a broader range of possible effects and is relatively agnostic. However, if there is a good reason to use a prior from a previous analysis, then of course it makes sense to report the analysis with the informative prior alongside an analysis with an uninformative prior. Reporting only

informative priors in a Bayesian analysis would generally not be a good idea. The issue is transparency: the reader should know what the posterior looks like for both an informative and an uninformative prior.

Another situation where posterior distributions associated with multiple priors should be reported is when one is carrying out an adversarial sensitivity analysis (Spiegelhalter, Abrams, and Myles 2004): one can take a group of agnostic, enthusiastic, and adversarial or skeptical priors that, respectively, reflect a non-committal a priori position, an informed position based on the researcher's prior beliefs, and an adversarial position based on a scientific opponent's beliefs. In such a situation, analyses using all three priors can be reported, so that the reader can determine how different prior beliefs influence the posterior. For an example of such an adversarial analysis, see Vasishth and Engelmann (2021). Finally, when carrying out hypothesis testing using Bayes factors, the choice of the prior on the parameter of interest becomes critically important; in that situation, it is very important to report a sensitivity analysis, showing the Bayes factor as well as a summary of the posterior distributions (Schad et al. 2021); we return to this point in chapter 16, which covers Bayes factors.

5.3 Why fitting a Bayesian hierarchical model is worth the effort

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. But there is a hidden cost to the relatively high speed furnished by the functions such as `lmer`. First, the model fit using `lmer` or the like makes many assumptions, but they are hidden from the user. This is not a problem for the knowledgeable modeler, but very dangerous for the naive user. A second conceptual problem is that the way frequentist models are typically used is to answer a binary question: is the effect “significant” or not? If a result is significant, the paper is considered worth publishing; if not, it is not. Although frequentist models can quickly answer the question

that the null hypothesis test poses, the frequentist test answers the wrong question. For discussion, see Vasishth and Nicenboim (2016).

Nevertheless, it is natural to ask why one should bother to go through all the trouble of fitting a Bayesian model. An important reason is the flexibility in model specification. The approach we have presented here can be used to extend essentially 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, Bürkner and Vuorre 2018), models with a shifted log-normal distribution (see Nicenboim, Logačev, et al. 2016; Rouder 2005), and distributional regression models (as shown in 5.1.6). By contrast, a frequentist model, although easy to fit quickly, forces the user to use an inflexible canned model, which may not necessarily make sense for their data.

This flexibility allows us also to go beyond the statistical models discussed before, and to develop complex hierarchical computational process models that are tailored to specific phenomena. An example are computational cognitive models, these 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 5.1.6, any parameter can have a group-level effect structure. Some examples of hierarchical computational cognitive models in psycholinguistics are Logačev and Vasishth (2016), Nicenboim and Vasishth (2018), Vasishth et al. (2017), Vasishth, Jaeger, and Nicenboim (2017), Lissón et al. (2021), Logacev and Dokudan (2021), Paape et al. (2021), Yadav, Smith, and Vasishth (2021a), and Yadav, Smith, and Vasishth (2021b). The hierarchical Bayesian modeling approach can even be extended to process models that cannot be expressed as a likelihood function, although in such cases one may have to write one's own sampler; for an example from psycholinguistics, see Yadav et al. (2021).¹¹ We discuss and implement in Stan some relatively simple computational cognitive models in chapters 18–21.

¹¹Most of the papers mentioned above provide example code using Stan or `brms`

5.4 Summary

This chapter presents two very commonly used classes of hierarchical model: those with normal and log-normal likelihoods. We saw several common variants of such models: varying intercepts, varying intercepts and varying slopes with or without a correlation parameter, and crossed random effects for subjects and items. We also experienced the flexibility of the Stan modeling framework through the example of a model that assumes a different residual standard deviation for each subject.

5.5 Further reading

Chapter 5 of Gelman et al. (2014) provides a rather technical but complete treatment of exchangeability in Bayesian hierarchical models. Bernardo and Smith (2009) is a brief but useful article explaining exchangeability, and Lunn et al. (2012) also has a helpful discussion that we have drawn on in this chapter. Gelman and Hill (2007) is a comprehensive treatment of hierarchical modeling, although it uses WinBUGS. Yarkoni (2020) discusses the importance of modeling variability in variables that researchers clearly intend to generalize over (e.g., stimuli, tasks, or research sites), and how under-specification of population-level (or random) effects imposes strong constraints on the generalizability of results. Sorensen, Hohenstein, and Vasishth (2016) provides an introduction, using Stan, to the Laird-Ware style matrix formulation (Laird and Ware 1982) of hierarchical models; this formulation has the advantage of flexibility and efficiency when specifying models in Stan syntax.

5.6 Exercises

Exercises with a normal likelihood

Exercise 5.1. A hierarchical model of cognitive load on pupil size.

As in section 4.1, we focus on the effect of cognitive load on pupil size, but this time we look at all the subjects of Wahn et al. (2016):

```
data("df_pupil_complete")
df_pupil_complete

## # A tibble: 2,228 × 4
##   subj trial  load p_size
##   <int> <int> <int>  <dbl>
## 1    1     1     2  1021.
## 2    1     2     1   951.
## 3    1     3     5  1064.
## # ... with 2,225 more rows
```

You should be able to now fit a “maximal” model (correlated varying intercept and slopes for subjects) assuming a normal likelihood. Base your priors in the priors discussed in section 4.1.

- (a) Examine the effect of load on pupil size, and the average pupil size. What do you conclude?
- (b) Do a sensitivity analysis for the prior on the intercept (α). What is the estimate of the effect (β) under different priors?
- (c) Is the effect of load consistent across subjects? Investigate this visually.

Exercises with a log-normal likelihood

Exercise 5.2. Are subject relatives easier to process than object relatives?

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?

Grodner and Gibson (2005) investigate an old claim in psycholinguistics that object relative clause (ORC) sentences are more difficult to process than subject relative clause (SRC) sentences. One explanation for this predicted difference is that the distance between the relative clause verb (*sent*

in the example below) and the head noun phrase of the relative clause (*reporter* in the example below) is longer in ORC vs. SRC. Examples are shown below. The relative clause is shown in square brackets.

(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, Jäger, and Vasishth (2020).

In the Grodner and Gibson data, the dependent measure is reading time at the relative clause verb, (e.g., *sent*) of different sentences with either ORC or SRC. The dependent variable is in milliseconds and was measured in a self-paced reading task. Self-paced reading is a task where subjects read a sentence or a short text word-by-word or phrase-by-phrase, pressing a button to get each word or phrase displayed; the preceding word disappears every time the button is pressed. In 6.1, we provide a more detailed explanation of this experimental method.

For this experiment, we are expecting longer reading times at the relative clause verbs of ORC sentences in comparison to the relative clause verb of SRC sentences.

```
data("df_gg05_rc")
df_gg05_rc

## # A tibble: 672 x 7
##   subj item condition     RT residRT qcorrect experiment
##   <int> <int> <chr>     <int>    <dbl>    <int> <chr>
## 1     1     1 objgap      320    -21.4       0 tedrg3
## 2     1     2 subjgap     424     74.7       1 tedrg2
## 3     1     3 objgap      309    -40.3       0 tedrg3
## # ... with 669 more rows
```

You should use a sum coding for the predictors. Here, object relative clauses ("objgaps") are coded +1, subject relative clauses -1.

```
df_gg05_rc <- df_gg05_rc %>%
  mutate(c_cond = 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 log-normal likelihood.

- (a) Examine the effect of relative clause attachment site (the predictor c_cond) on reading times RT (β).
- (b) Estimate the median difference between relative clause attachment sites in milliseconds, and report the mean and 95% CI.
- (c) Do a sensitivity analysis. What is the estimate of the effect (β) under different priors? What is the difference in milliseconds between conditions under different priors?

Exercise 5.3. Relative clause processing in Mandarin Chinese

Load the following two data sets:

```
data("df_gibsonwu")
gibsonwu <- df_gibsonwu
data("df_gibsonwu2")
gibsonwu2 <- df_gibsonwu2
```

The data are taken from two experiments that investigate (inter alia) the effect of relative clause type on reading time in Chinese. The data are from Gibson and Wu (2013) and Vasishth et al. (2013) respectively. The second data set is a direct replication attempt of the Gibson and Wu (2013) experiment.

Chinese relative clauses are interesting theoretically because they are prenominal: the relative clause appears before the head noun. For example, the English relative clauses shown above would appear in the following order in Mandarin. The square brackets mark the relative clause, and REL refers to the Chinese equivalent of the English relative pronoun *who*.

(2a) [The photographer *sent* to the editor] REL the *reporter* was hoping for a good story. (ORC)

(2b) [*sent* the photographer to the editor] REL the *reporter* who was hoping for a good story. (SRC)

As discussed in Gibson and Wu (2013), the consequence of Chinese relative clauses being prenominal is that the distance between the verb in relative clause and the head noun is larger in subject relatives than object relatives. Hsiao and Gibson (2003) were the first to suggest that the larger distance in subject relatives leads to longer reading time at the head noun. Under this view, the prediction is that subject relatives are harder to process than object relatives. If this is true, this is interesting and surprising because in most other languages that have been studied, subject relatives are easier to process than object relatives; so Chinese will be a very unusual exception cross-linguistically.

The data provided are for the critical region (the head noun; here, *reporter*). The experiment method is self-paced reading, so we have reading times in milliseconds. The second data set is a direct replication attempt of the first data set, which is from Gibson and Wu (2013).

The research hypothesis is whether the difference in reading times between object and subject relative clauses is negative. For the first data set (df_gibsonwu), investigate this question by fitting two “maximal” hierarchical models (correlated varying intercept and slopes for subjects and items). The dependent variable in both models is the raw reading time in milliseconds. The first model should use the normal likelihood in the model; the second model should use the log-normal likelihood. In both models, use ± 0.5 sum coding to model the effect of relative clause type. You will need to decide on appropriate priors for the various parameters.

- (a) Plot the posterior predictive distributions from the two models. What is the difference in the posterior predictive distributions of the two models; and why is there a difference?
- (b) Examine the posterior distributions of the effect estimate in the two models. Why are these different?
- (c) Given the posterior predictive distributions you plotted above,

why is the log-normal likelihood model better for carrying out inference and hypothesis testing?

Next, work out a normal approximation of the log-normal model's posterior distribution for the relative clause effect that you obtained from the above data analysis. Then use that normal approximation as an informative prior for the slope parameter when fitting a hierarchical model to the second data set. This is an example of incrementally building up knowledge by successively using a previous study's posterior as a prior for the next study; this is essentially equivalent to pooling both data sets (check that pooling the data gives you approximately the same posterior as the informative-prior model fit above).

Finally, as a sensitivity analysis, also fit a hierarchical model with the mildly informative prior $\text{Normal}(0,1)$. Do the posterior distributions of the relative clause effect differ in the second data set when one uses an informative prior vs. a mildly informative one like $\text{Normal}(0,1)$?

Exercise 5.4. Agreement attraction in comprehension

Load the following data:

```
data("df_dillonE1")
dillonE1 <- df_dillonE1
head(dillonE1)
```

```
##          subj      item    rt int    expt
## 49 dillonE11 dillonE119 2918 low dillonE1
## 56 dillonE11 dillonE119 1338 low dillonE1
## 63 dillonE11 dillonE119  424 low dillonE1
## 70 dillonE11 dillonE119  186 low dillonE1
## 77 dillonE11 dillonE119  195 low dillonE1
## 84 dillonE11 dillonE119 1218 low dillonE1
```

The data are taken from an experiment that investigate (inter alia) the effect of number similarity between a noun and the auxiliary verb in sentences like the following. There are two levels to a factor called Int(erference): low and high.

(3a) low: The key to the cabinet *are* on the table (3b) high: The key to the *cabinets* *are* on the table

Here, in (3b), the auxiliary verb *are* is predicted to be read faster than in (3a), because the plural marking on the noun *cabinets* leads the reader to think that the sentence is grammatical. (Both sentences are ungrammatical.) This phenomenon, where the high condition is read faster than the low condition, is called **agreement attraction**.

The data provided are for the critical region (the auxiliary verb *are*). The experiment method is eye-tracking; we have total reading times in milliseconds.

The research question is whether the difference in reading times between high and low conditions is negative.

- First, using a log-normal likelihood, fit a hierarchical model with correlated varying intercept and slopes for subjects and items. You will need to decide on the priors for the model.
- By simply looking at the posterior distribution of the slope parameter β , what would you conclude about the theoretical claim relating to agreement attraction?

Exercises with a logistic regression (Bernoulli likelihood).

Exercise 5.5. Attentional blink

The attentional blink (AB; first described by Raymond, Shapiro, and Arnell 1992; though it has been noticed before e.g., Broadbent and Broadbent 1987) refers to a temporary reduction in the accuracy of detecting a *probe* (e.g., a letter “X”) presented closely after a *target* that has been detected (e.g., a white letter). We will focus on the experimental condition of Experiment 2 of Raymond, Shapiro, and Arnell (1992). Subjects are presented with letters in rapid serial visual presentation (RSVP) at the center of the screen at a constant rate and are required to identify the only white letter (*target*) in the stream of black letters, and then to report whether the letter X (*probe*) occurred in the subsequent letter stream. The AB is defined as having occurred when the target is reported correctly but the report of the probe is inaccurate at a short *lag* or *target-probe* interval.

The data set `df_ab` is a subset of the data of this paradigm from a replication conducted by Grassi et al. (2021). In this subset, the probe was always present and the target was correctly identified. We want to find out how the lag affects the accuracy of the identification of the probe.

```
data("df_ab")
df_ab

## # A tibble: 2,101 × 4
##   subj  probe_correct trial    lag
##   <int>      <int> <int> <int>
## 1     1          0     2     5
## 2     1          1     4     4
## 3     1          1     8     6
## # ... with 2,098 more rows
```

Fit a logistic regression assuming a linear relationship between `lag` and accuracy (`probe_correct`). Assume a hierarchical structure with correlated varying intercept and slopes for subjects. You will need to decide on the priors for this model.

- (a) How is the accuracy of the probe identification affected by the lag? Estimate this in log-odds and percentages.
- (b) Is the linear relationship justified? Use posterior predictive checks to verify this.
- (c) Can you think about a better relationship between lag and accuracy? Fit a new model and use posterior predictive checks to verify if the fit improved.

Exercise 5.6. Is there a Stroop effect in accuracy?

Instead of the response times of the correct answers, we want to find out whether accuracy also changes by condition in the Stroop task. Fit the Stroop data with a hierarchical logistic regression (i.e., a Bernoulli likelihood with a logit link). Use the complete data set, `df_stroop_complete` which also includes incorrect answers, and subset it selecting the first 50 subjects.

Exercise 5.7. The grammaticality illusion

Load the following two data sets:

```
data("df_english")
english <- df_english
data("df_dutch")
dutch <- df_dutch
```

In an offline accuracy rating study on English double center-embedding constructions, Gibson and Thomas (1999) found that grammatical constructions (e.g., example 4a below) were no less acceptable than ungrammatical constructions (e.g., example 4b) where a middle verb phrase (e.g., *was cleaning every week*) was missing.

(4a) The apartment that the maid who the service had sent over was cleaning every week was well decorated.

(4b) *The apartment that the maid who the service had sent over — was well decorated

Based on these results from English, Gibson and Thomas (1999) proposed that working-memory overload leads the comprehender to forget the prediction of the upcoming verb phrase (VP), which reduces working-memory load. This came to be known as the *VP-forgetting hypothesis*. The prediction is that in the word immediately following the final verb, the grammatical condition (which is coded as +1 in the data frames) should be harder to read than the ungrammatical condition (which is coded as -1).

The design shown above is set up to test this hypothesis using self-paced reading for English (Vasishth et al. 2011), and for Dutch (Frank, Trompenaars, and Vasishth 2015). The data provided are for the critical region (the noun phrase, labeled NP1, following the final verb); this is the region for which the theory predicts differences between the two conditions. We have reading times in log milliseconds.

- (a) First, fit a linear model with a full hierarchical structure by subjects and by items for the English data. Because we have log mil-

liseconds data, we can simply use the normal likelihood (not the log-normal). What scale will the parameters be in, milliseconds or log milliseconds?

- (b) Second, using the posterior for the effect of interest from the English data, derive a prior distribution for the effect in the Dutch data. Then fit two linear mixed models: (i) one model with relatively uninformative priors for β (for example, $Normal(0, 1)$), and (ii) one model with the prior for β you derived from the English data. Do the posterior distributions of the Dutch data's effect show any important differences given the two priors? If yes, why; if not, why not?
- (c) Finally, just by looking at the English and Dutch posteriors, what can we say about the VP-forgetting hypothesis? Are the posteriors of the effect from these two languages consistent with the hypothesis?

6

The Art and Science of Prior Elicitation

Nothing strikes fear into the heart of the newcomer to Bayesian methods more than the idea of specifying priors for the parameters in a model. On the face of it, this concern seems like a valid one; how can one know what the plausible parameter values are in a model before one has even seen the data?

In reality, this worry is purely a consequence of the way we are normally taught to carry out data analysis, especially in areas like psychology and linguistics. Model fitting is considered to be a black-box activity, with the primary concern being whether the effect of interest is “significant” or “non-significant.” As a consequence of the training that we receive, we learn to focus on one thing (the p -value) and we learn to ignore the estimates that we obtain from the model; it becomes irrelevant whether the effect of interest has a mean value of 500 ms (in a reading study, say) or 10 ms; all that matters is whether it is a significant effect or not. In fact, the way many scientists summarize the literature in their field is by classifying studies into two bins: significant and non-significant. There are obvious problems with this classification method; for example, $p = 0.051$ might be counted as “marginally” significant, but $p = 0.049$ is never counted as marginally non-significant. Real-life examples of such a binary classification approach are Phillips, Wagers, and Lau (2011) and Hammerly, Staub, and Dillon (2019). Because the focus is on significance, we never develop a sense of what the estimates of an effect are likely to be in a future study. This is why, when faced with a prior-distribution specification problem, we are misled into feeling like we know nothing about the quantitative estimates relating to a problem we are studying.

Prior specification has a lot in common with something that physicists call a Fermi problem. As Von Baeyer (1988) describes it: “A Fermi problem has a characteristic profile: Upon first hearing it, one doesn’t have even the remotest notion what the answer might be. And one feels certain that too lit-

tle information exists to find a solution. Yet, when the problem is broken down into subproblems, each one answerable without the help of experts or reference books, an estimate can be made ...". Fermi problems in the physics context are situations where one needs ballpark (approximate) estimates of physical quantities in order to proceed with a calculation. The name comes from a physicist, Enrico Fermi; he developed the ability to carry out fairly accurate back-of-the-envelope calculations when working out approximate numerical values needed for a particular computation. Von Baeyer (1988) puts it well: "Prudent physicists—those who want to avoid false leads and dead ends—operate according to a long-standing principle: Never start a lengthy calculation until you know the range of values within which the answer is likely to fall (and, equally important, the range within which the answer is unlikely to fall)." As in physics, so in data analysis: as Bayesians, we need to acquire the ability to work out plausible ranges of values for parameters. This is a learnable skill, and improves with practice. With time and practice, we can learn to emulate prudent physicists.

As Spiegelhalter, Abrams, and Myles (2004) point out, there is no one "correct" prior distribution. One consequence of this fact is that a good Bayesian analysis always takes a range of prior specifications into account; this is called a sensitivity analysis. We have already seen examples of this, but more examples will be provided in this and later chapters.

Prior specification requires the estimation of probabilities. Human beings are not good at estimating probabilities, because they are susceptible to several kinds of biases (Kadane and Wolfson 1998; Spiegelhalter, Abrams, and Myles 2004). We list the most important ones that are relevant to cognitive science applications:

- Availability bias: Events that are more salient to the researcher are given higher probability, and events that are less salient are given lower probability.
- Adjustment and anchoring bias: One's initial assessment of the probability of an event can influence one's subsequent judgements, e.g., of uncertainty intervals—one's estimate of the uncertainty interval will tend to be influenced by one's initial assessment.
- Overconfidence: When eliciting uncertainty intervals from oneself, there is a tendency to specify too tight an interval.

- Hindsight bias: If one relies on the data to come up with a prior for the analysis of that very same data set, one's assessment is likely to be biased.

Although training can improve the natural tendency to be biased in these different ways, one must recognize that bias is inevitable when eliciting priors, either from oneself or from other experts; it follows that one should always define “a community of priors” (Kass and Greenhouse 1989): one should consider the effect of informed as well as skeptical or agnostic (uninformative) priors on the posterior distribution of interest. Incidentally, bias is not unique to Bayesian statistics; the same problems arise in frequentist data analysis. Even in frequentist analyses, the researcher always interprets the data in the light of their prior beliefs; the data never really “speak for themselves.” The great advantage that Bayesian methods have is that they allow us to formally take a range of (competing) prior beliefs formally into account in interpreting the data. We illustrate this point in the present chapter with some examples.

6.1 Eliciting priors from oneself for a self-paced reading study: A simple example

In section 3.4, we have already encountered a sensitivity analysis; there we used several priors to investigate how the posterior is affected. Here, we present another example of a sensitivity analysis; the problem we focus on is how to elicit priors from oneself for a particular research problem.

We will work out priors from first principles. Consider English subject vs. object relative clause processing differences in self-paced reading studies. The self-paced reading method is commonly used in psycholinguistics as a cheaper and faster substitute to eyetracking during reading. The subject is seated in front of a computer screen and is initially shown a series of broken lines that mask words from a complete sentence. The subject then unmasks the first word (or phrase) by pressing the space bar. Upon pressing the space bar again, the second word/phrase is unmasked and the first word/phrase is masked again; the time in milliseconds that elapsed between these two space-bar presses counts as the reading time

for the first word/phrase. In this way, the reading time for each successive word/phrase in the sentence is recorded. Usually, at the end of each trial, the subject is also asked a yes/no question about the sentence. This is intended to ensure that the subject is adequately attending to the meaning of the sentence.

A classic example of self-paced reading data appeared in Exercise 5.1. A hierarchical model that we could fit to such data would be the following. In chapter 5, we showed that for reading-time data, the log-normal likelihood is a better choice than a normal likelihood. In the present chapter, in order to make it easier for the reader to get started with thinking about priors, we use the normal likelihood instead of the log-normal.

The model below has varying intercepts and varying slopes for subjects and for items, but assumes no correlation between the varying intercepts and slopes. In the model shown below, we use “default” priors that the `brm` function assumes for all the parameters. We are only using default priors here as a starting point; in practice, we will **never** use default priors for a reported analysis. In the model output below, for brevity we will only display the summary of the posterior distribution for the slope parameter, which represents the difference between the two condition means.

```
data("df_gg05_rc")
df_gg05_rc <- df_gg05_rc %>%
  mutate(c_cond = if_else(condition == "objgap", 1/2, -1/2))
fit_gg05 <- brm(RT ~ c_cond + (1 + c_cond || subj) +
  (1 + c_cond || item), df_gg05_rc)

(default_b <- posterior_summary(fit_gg05,
  variable = "b_c_cond"))

##           Estimate Est.Error Q2.5 Q97.5
## b_c_cond     103      37.8  29.2   181
```

The estimates from this model are remarkably similar to those from a frequentist linear mixed model (Bates, Mächler, et al. 2015a):

```

fit_lmer <- lmer(RT ~ c_cond + (1 + c_cond || subj) +
  (1 + c_cond || item), df_gg05_rc)
b <- summary(fit_lmer)$coefficients["c_cond", "Estimate"]
SE <- summary(fit_lmer)$coefficients["c_cond", "Std. Error"]
## estimate of the slope and
## lower and upper bounds of the 95% CI:
(lmer_b <- c(b, b - (2 * SE), b + (2 * SE)))

```

[1] 102.3 29.9 174.7

The similarity between the estimates from the Bayesian and frequentist models is due to the fact that default priors, being relatively vague, don't influence the posterior much. This leads to the likelihood dominating in determining the posteriors. In general, such vague priors on the parameters will show a similar lack of influence on the posterior (Spiegelhalter, Abrams, and Myles 2004). We can quickly establish this in the above example by using another vague prior:

```

fit_gg05_unif <- brm(RT ~ c_cond + (1 + c_cond || subj) +
  (1 + c_cond || item),
prior = c(
  prior(uniform(-2000, 2000), class = Intercept),
  prior(uniform(-2000, 2000), class = b, coef = "c_cond"),
  prior(normal(0, 500), class = sd),
  prior(normal(0, 500), class = sigma)),
df_gg05_rc,
control = list(adapt_delta = 0.999,
  max_treedepth = 15))

## Warning: It appears as if you have specified a lower bounded prior on a parameter that has no nat
## If this is really what you want, please specify argument 'lb' of 'set_prior' appropriately.
## Warning occurred for prior
## b_c_cond ~ uniform(-2000, 2000)

## Warning: It appears as if you have specified an upper bounded prior on a parameter that has no na
## If this is really what you want, please specify argument 'ub' of 'set_prior' appropriately.
## Warning occurred for prior

```

TABLE 6.1: Estimates of the mean difference (with 95% confidence/credible intervals) between two conditions in a hierarchical model of English relative clause data from Grodner and Gibson, 2005, using (a) the frequentist hierarchical model, (b) a Bayesian model using default priors from the `brm` function, and (c) a Bayesian model with uniform priors.

model	mean	lower	upper
Frequentist	102	30	175
Default prior	103	29	181
Uniform	103	27	180

```
## b_c_cond ~ uniform(-2000, 2000)

(uniform_b <- posterior_summary(fit_gg05_unif, variable = c("b_c_cond")))

##           Estimate Est.Error Q2.5 Q97.5
## b_c_cond     103      38.2  26.8   180
```

We get warnings because `brms` wants us to discourage us from using a uniform prior, but the model converged.

As shown in Table 6.1, when we compare the means of the posteriors from this versus the other two model estimates shown above, we see that they are all very similar.

It is tempting for the newcomer to Bayesian statistics to conclude from Table 6.1 that default priors used in `brms`, or uniform priors, are good enough for fitting models. This conclusion would be incorrect. There are many reasons why a sensitivity analysis—which includes regularizing, relatively informative priors—is necessary in Bayesian modeling. First, relatively informative, regularizing priors must be considered in many cases to avoid convergence problems. In fact, in many cases the frequentist model fit in `lme4` will return estimates—such as ± 1 correlation estimates between varying intercepts and varying slopes—that are actually represent convergence failures (Bates, Kliegl, et al. 2015; Matuschek et al. 2017). In Bayesian models, unless we use regularizing priors that are at least mildly informative,

we will generally face similar convergence problems. Second, when computing Bayes factors, sensitivity analyses using increasingly informative priors is vital; see chapter @[\(ch:bf\)](#) for extensive discussion of this point. Third, as soon as we go beyond standard hierarchical models, and start fitting customized, more complex models, we will need more informative priors in order to improve the efficiency of the sampling and in some cases to avoid convergence problems (an example is finite mixture models in chapter [20](#)). Fourth, one of the greatest advantages of Bayesian models is that one can formally take into account conflicting or competing prior beliefs in the model, by eliciting informative priors from competing experts. Although such a use of informative priors is still rare in cognitive science, it can be of great value when trying to interpret a statistical analysis.

Given the importance of regularizing, informative priors, we consider next some informative priors that we could use in the given model. We unpack the process by which we could work these priors out from existing information in the literature.

Initially, when trying to work out some alternative priors for these parameters, we might think that we know absolutely nothing about the seven parameters in this model. But, as in Fermi problems, we actually know more than we realize. Let's think about the parameters one by one. For ease of exposition, we begin by writing out the model in mathematical form. n is the row id in the data-frame.

$$RT_n \sim Normal(\alpha + u_{subj[n],1} + w_{item[n],1} + c_cond_n \cdot (\beta + u_{subj[n],2} + w_{item[n],2}), \sigma) \quad (6.1)$$

where

$$\begin{aligned} u_1 &\sim Normal(0, \tau_{u_1}) \\ u_2 &\sim Normal(0, \tau_{u_2}) \\ w_1 &\sim Normal(0, \tau_{w_1}) \\ w_2 &\sim Normal(0, \tau_{w_2}) \end{aligned} \quad (6.2)$$

The parameters that we need to define priors for are the following: $\alpha, \beta, \tau_{u_1}, \tau_{u_2}, \tau_{w_1}, \tau_{w_2}, \sigma$.

We will proceed from first principles. Let's begin with the intercept, α ; under the sum-contrast coding used here, it represents the grand mean reading time in the data set. Ask yourself: What is the absolute minimum possible reading time? The answer is 0 ms; reading time cannot be negative. You have already eliminated half the real-number line as impossible values! Thus, one cannot really say that one knows *nothing* about the plausible values of mean reading times. Having eliminated half the real-number line, now ask yourself: what is a reasonable upper bound on reading time for an English ditransitive verb? Even after taking into account variations in word length and frequency, one minute (60 seconds) seems like too long; even 30 seconds seems unreasonably long to spend on a single word. As a first attempt at an approximation, somewhere between 2500 and 3000 ms might constitute a reasonable upper bound, with 3000 ms being less likely than 2500 ms.

Now consider what an approximate average reading time for a verb might be. One can arrive at such a ballpark number by asking oneself how fast one can read an abstract with, say, 500 words in it. Suppose that we estimate that we can read 500 words in 120 seconds (two minutes). Then, $120/500 = 0.24$ seconds is the time we would spend per word on average; this is 240 ms per word. Maybe two minutes for 500 words was too optimistic? Let's adjust the mean to 300 ms, instead of 240 ms. Such intuition-based judgments can be a valuable starting point for an analysis, as Fermi showed repeatedly in his work (Von Baeyer 1988). If one is uncomfortable consulting one's intuition about average reading times, or even as a sanity check to independently validate one's own intuitions, one can look up a review article on reading that gives empirical estimates (Rayner 1998).

One could express the above guesses as a normal distribution truncated at 0 ms on the ms scale, with mean 300 ms and standard deviation 1000 ms. An essential step in such an estimation procedure is to plot one's assumed prior distribution graphically to see if it seems reasonable: Figure 6.1 shows a graphical summary of this prior; it is generally a good idea to plot one's priors in order to assess them.

Once we plot the prior, one might conclude that the prior distribution is a bit too widely spread out to represent mean reading time per word. But for estimating the posterior distribution, it will rarely be harmful to allow a broader range of values than we strictly consider plausible (the situation

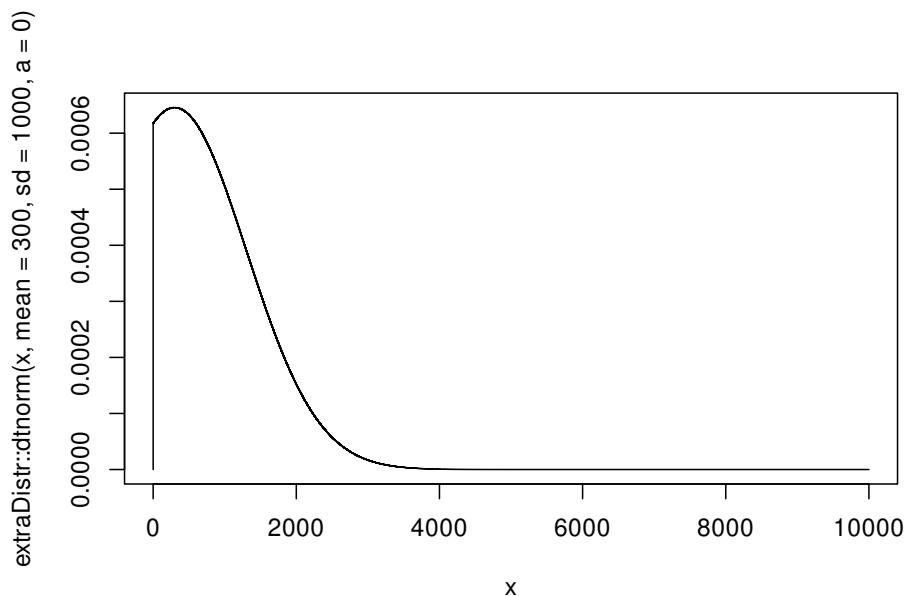


FIGURE 6.1: A truncated normal distribution representing a prior distribution on mean reading times.

is different when it comes to Bayes factors analyses, as we will see later—there, widely spread out priors can have a dramatic impact on the Bayes factor).

Another way to obtain a better feel for what plausible distributions of word reading times might be to just plot some existing data from published work. Figure 6.2 shows the reading time distributions from ten published studies.

Although our truncated normal distribution, $Normal_+(300, 1000)$, seems like a pretty wild guess, it actually is not terribly unreasonable given what we observe in these ten published self-paced reading studies. As shown in Figure 6.2, the distributions of mean reading times in these different self-paced reading studies from different languages (English, Persian, Dutch, Hindi, German, Spanish) fall within the prior distribution. These studies are not about relative clauses; but that doesn't matter, because we are just trying to come up with a prior distribution on average reading times for a word. We just want an approximate idea of the range of plausible mean reading times.

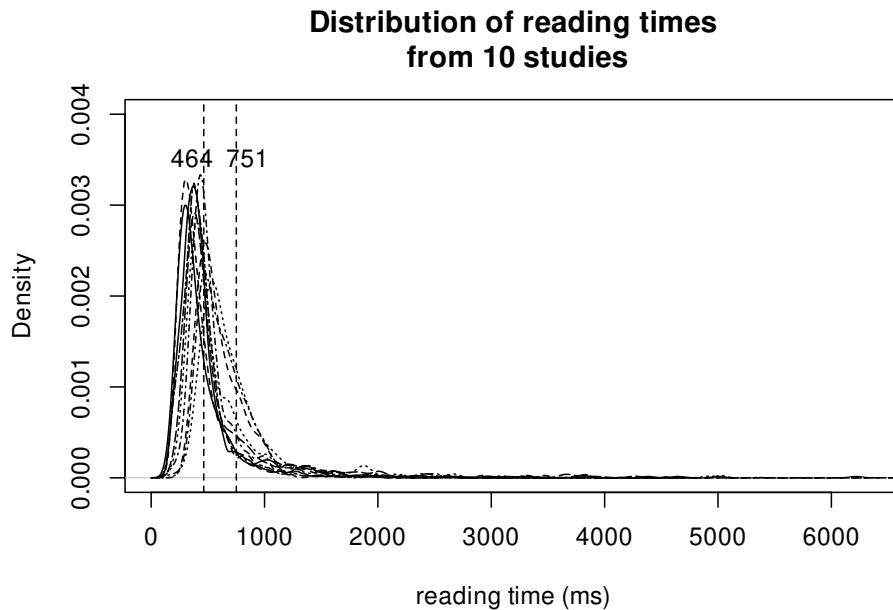


FIGURE 6.2: Distributions of reading times from ten self-paced reading studies. The two vertical lines mark the minimum and maximum means of the reading times in these ten studies.

The above prior specification for the intercept can (and must!) be evaluated in the context of the model using prior predictive checks. We have already encountered prior predictive checks in previous chapters; we will revisit them in detail in chapter 7. In the above data set on English relative clauses, we could check what the prior on the intercept implies in terms of the data generated by the model (see chapter 5 for examples). As we stress repeatedly throughout this book, sensitivity analysis is an integral component of Bayesian methodology. A sensitivity analysis should be used to work out what the impact is of a range of priors on the posterior distribution.

Having come up with some potential priors for the intercept, consider next the prior specification for the effect of relative clause type on reading time; this is the slope β in the model above. Theory suggests (see Grodner and Gibson 2005 for a review) that subject relatives in English should be easier to process than object relatives, at the relative clause verb. This means that a priori, we expect the difference between object and sub-

ject relatives to be positive in sign. What would be a reasonable mean for this effect? We can look at previous research to obtain some ballpark estimates.

For example, Just and Carpenter (1992) carried out a self-paced reading study on English subject and object relatives, and their Figure 2 (p. 130) shows that the difference between the two relative clause types at the relative clause verb ranges from about 10 ms to 100 ms (depending on working memory capacity differences in different groups of subjects). This is already a good starting point, but we can look at some other published data to gain more confidence about the approximate difference between the conditions. For example, Reali and Christiansen (2007) investigated subject and object relatives in four self-paced reading studies; in their design, the noun phrase inside the relative clause was always a pronoun, and they carried out analyses on the verb plus pronoun, not just the verb as in Grodner and Gibson (2005). We can still use the estimates from this study, because including a pronoun like “I”, “you”, or “they” in a verb region is not going to increase reading times dramatically. The hypothesis for Reali and Christiansen (2007) was that because object relatives containing a pronoun occur more frequently in corpora than subject relatives with a pronoun, the relative clause verb should be processed faster in object relatives than subject relatives (this is the opposite to the prediction for the reading times at the relative clause verb discussed in Grodner and Gibson 2005). The authors report comparisons for the pronoun and relative clause verb taken together (i.e., pronoun+verb in object relatives and verb+pronoun in subject relatives). In experiment 1, they report a -57 ms difference between object and subject relatives, with a 95% confidence interval ranging from -104 to -10 ms. In a second experiment, they report a difference of -53.5 ms with a 95% confidence interval ranging from -79 to -28 ms; in a third experiment, the difference was -32 ms [-48, -16]; and in a fourth experiment, -43 ms [-84, -2]. Thus, given these data from English, we may want to allow the prior values to range from a negative range to a positive range. Yet another study involved English relative clauses is by Fedorenko, Gibson, and Rohde (2006). In this self-paced reading study, Fedorenko and colleagues compared reading times within the entire relative clause phrase (the relative pronoun and the noun+verb sequence inside the relative clause). Their data show that object relatives are harder to process than subject relatives; the difference in means is 460 ms, with a con-

fidence interval [299, 621] ms. This difference is much larger than in the other studies mentioned above, but this is because of the long region of interest considered—it is well-known that the longer the reading/reaction time, the larger the standard deviation and therefore the larger the potential difference between means (Wagenmakers and Brown 2007).

This previous data from English gives us some empirical basis for assuming that the object minus subject relative clause difference in the Grodner and Gibson (2005) study on English could range from 10 to 100 ms or so. Although we expect the effect to be positive, perhaps we don't want to pre-judge this before we see the data. For this reason, we could decide on a $Normal(0, 50)$ prior on the slope parameter in the model. This prior, which implies that we are 95% certain that the range of values lies between -100 and $+100$ ms. This prior is specifically for the millisecond scale, and specifically for the case where the critical region is one word (the relative clause verb in English).

In this particular example, it makes sense to assume that large effects like 100 ms are unlikely; this is so even if we do occasionally see estimates as high as these in published data. We would be unwilling to take such large effects seriously because a major reason for observing overly large estimates in a one-word region of interest would be publication bias coupled with Type M error (Gelman and Carlin 2014). Published studies in psycholinguistics are often underpowered, which leads to exaggerated estimates being published (Type M error). Because big news effects are encouraged in major journals, overestimates tend to get published preferentially. See Vasishth et al. (2013), Vasishth, Mertzen, et al. (2018), Nicenboim, Vasishth, et al. (2018), and Jäger et al. (2020) for detailed discussion of this point in the context of psycholinguistics.

Of course, if our experiment is designed so that the critical region constitutes several words, as in the Fedorenko, Gibson, and Rohde (2006) study, then one would have to choose a prior with a larger mean and standard deviation.

A related important issue to consider when defining priors is the scale in which the parameter is defined. For example, if we were analyzing the Grodner and Gibson (2005) experiment using the log-normal likelihood, then the intercept and slope are on the log millisecond scale. A uniform

prior on the intercept and slope parameter imply rather strange priors on the millisecond scale. For example, suppose we assume that the intercept on the log ms scale has priors $\text{Normal}(0, 10)$ and the slope has a prior $\text{Normal}(0, 1)$. In the millisecond scale, the priors on the intercept and slope imply range from a very broad range of reading times, ranging from a very large negative value to a very large positive value, which obviously makes little sense:

```
intercept <- rnorm(100000, mean = 0, sd = 10)
slope <- rnorm(100000, mean = 0, sd = 1)
effect <- exp(intercept + slope/2) - exp(intercept - slope/2)
quantile(effect, prob = c(0.025, 0.975))

##          2.5%      97.5%
## -9316713  9510893
```

Similarly, if in a logistic regression, we assume that the intercept and slope on the log odds scale for the intercept and slope in a two-condition design are $\text{Normal}(0, 10)$ and $\text{Normal}(0, 1)$ respectively, this implies that, with ± 0.5 effect coding for the two conditions, on the probability scale the prior probabilities for the intercept and slope are as in Figure 6.3.

Figure 6.3 shows that the prior on the intercept seems quite unreasonable for most applications (0 and 1 are most likely values); the prior on the slope is not too unreasonable (although this depends on the particular question being studied). Of course, if there is sufficient data, the likelihood will dominate over the priors. In that case, even the relatively unreasonable prior on the intercept will lead to a posterior that will be mainly affected by the likelihood.

A frequently asked question from newcomers to Bayes is: what if I define a too restricted prior? Wouldn't that bias the posterior distribution? This concern is often raised by critics of Bayesian methods. The key point here is that a good Bayesian analysis always involves a sensitivity analysis, and also includes prior and posterior predictive checks under different priors. One should reject the priors that make no sense in the particular research problem we are working on, or which unreasonably bias the posterior. As one gains experience with Bayesian modeling, these concerns will recede

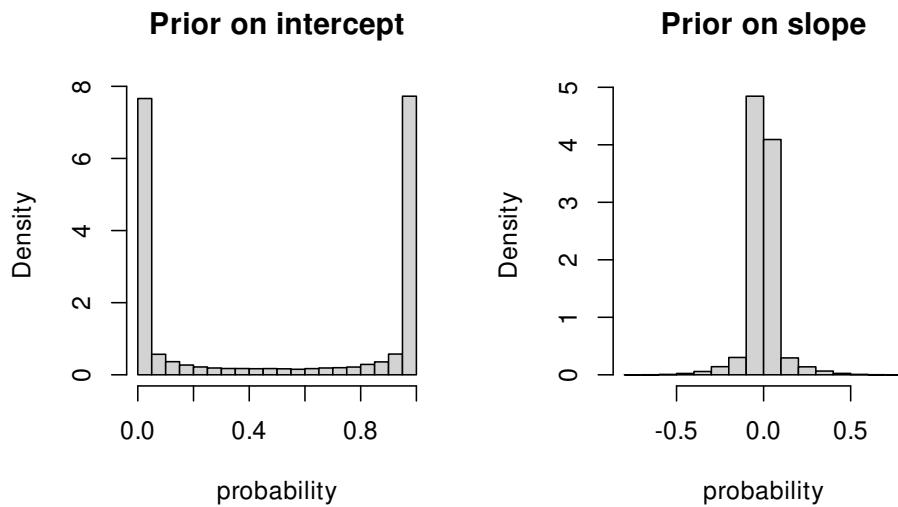


FIGURE 6.3: The effect of normal priors defined on log-odds space for the intercept and slope in a logistic regression, transformed to probability space.

as we come to understand how useful and important priors are for interpreting the data.

As an extreme example of an overly specific prior, if one were to define a $Normal(0, 10)$ prior for the α and/or β parameters on the millisecond scale for the Grodner and Gibson (2005) example above; that would definitely bias the posterior for the parameters. Let's check this:

```
restrictive_priors <- c(
  prior(normal(0, 10), class = Intercept),
  prior(normal(0, 10), class = b),
  prior(normal(0, 500), class = sd),
  prior(normal(0, 500), class = sigma)
)

fit_restrictive <- brm(RT ~ c_cond + (c_cond || subj) +
  ( c_cond || item),
  prior = restrictive_priors,
  # Increase the iterations to avoid warnings
```

```
    iter = 4000,
    df_gg05_rc)

summary(fit_restrictive)

## Family: gaussian
## Links: mu = identity; sigma = identity
## Formula: RT ~ c_cond + (c_cond || subj) + (c_cond || item)
## Data: df_gg05_rc (Number of observations: 672)
## Draws: 4 chains, each with iter = 4000; warmup = 2000; thin = 1;
##         total post-warmup draws = 8000
##
## Group-Level Effects:
## ~item (Number of levels: 16)
##             Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS
## sd(Intercept) 414.88     81.80   282.94   597.49 1.00      1068
## sd(c_cond)     86.28     42.49     8.95   174.76 1.00      1933
##                 Tail_ESS
## sd(Intercept)   2203
## sd(c_cond)      2830
##
## ~subj (Number of levels: 42)
##             Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS
## sd(Intercept) 162.52     24.30   121.39   216.79 1.00      2881
## sd(c_cond)     167.85     37.35   96.58   244.22 1.00      3130
##                 Tail_ESS
## sd(Intercept)   4363
## sd(c_cond)      3779
##
## Population-Level Effects:
##             Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     3.94     10.20   -16.00   23.95 1.00     16474     5078
## c_cond        5.74      9.74   -13.00   24.69 1.00     15465     5338
##
## Family Specific Parameters:
##             Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
```

```
## sigma    305.66      9.26   287.96   324.14 1.00    10715     6385
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

Here, we see that the overly specific priors on the intercept and slope will dominate in determining the posterior; such priors obviously make no sense. If there is ever any doubt about the implications of a prior, prior and posterior predictive checks should be used to investigate the implications.

Here, an important Bayesian principle is Cromwell's rule (Lindley 1991; Jackman 2009): we should generally allow for some uncertainty in our priors. A prior like $Normal(0, 10)$ or $Normal_+(0, 10)$ is clearly overly restrictive given what we've established about plausible values of the relative clause effect from existing data. A more reasonable but still quite tight prior would be $Normal(0, 50)$. In the spirit of Cromwell's rule, just to be conservative, we can allow larger possible effect sizes by adopting a prior such as $Normal(0, 75)$, and we allow the effect to be negative, even if theory suggests otherwise. Although there are no fixed rules for deciding on a prior, a sensitivity analysis will quickly establish whether the prior or priors chosen are biasing the posterior. One critical thing to remember related to Cromwell's rule is that if we categorically rule out a range of values a priori for a parameter by giving that range a probability of 0, the posterior will also never include that range of values, no matter what the data show. For example, in the Reali and Christiansen (2007) experiments, if we had used a truncated prior like $Normal_+(0, 50)$, the posterior can never show the observed negative sign on the effects as reported in the paper. As a general rule, therefore, one should allow the effect to vary in both directions, positive and negative. Sometimes unidirectional priors are justified; in those cases, it is of course legitimate to use them. An example is the prior on standard deviations (which cannot be negative).

Having defined the priors for the intercept and the slope, we are left with prior specifications for the variance component parameters. At least in psycholinguistics, the residual standard deviation is usually the largest source of variance; the by-subject intercepts' standard deviation is usually the next-largest value, and if experimental items are designed to have

minimal variance, then these are usually the smallest components. Here again, we can look at some previous data to get a sense of what the priors should look like. For example, we could use the estimates for the variance components from existing studies. Figure 6.4 shows the empirical distributions from 10 published studies. There are four classes of variance component: the subject and item intercept standard deviations, the standard deviations of slopes, and the standard deviations of the residuals. In each case, we can compute the means and standard deviations of each type of variance component, and use these to define normal distribution truncated at 0. These empirically estimated priors are shown in Figure 6.4. The priors are:

- Subject intercept SDs: $\text{Normal}_+(165, 55)$.
- Item intercept SDs: $\text{Normal}_+(49, 52)$.
- Slope SDs: $\text{Normal}_+(39, 58)$.
- Residual SDs: $\text{Normal}_+(392, 140)$.

One could use such empirically determined priors for the standard deviations, or one could just use the above estimates as a starting point, keeping Cromwell's rule in mind—it's better to have a little bit more uncertainty than warranted than too tight a prior. As mentioned earlier, an overly tight prior will ensure that the posterior is entirely driven by the prior. Again, prior predictive checks should be an integral part of the process of establishing a sensible set of priors for the variance components. This point about prior predictive checks will be elaborated on with examples in 7.

We now apply the relatively informative priors we came up with above to analyze the Grodner and Gibson (2005) data. Applying Cromwell's rule, we allow for a bit more uncertainty than our existing empirical data suggest. Specifically, we choose the following informative priors for the Grodner and Gibson (2005) data:

- $\alpha \sim \text{Normal}(500, 100)$
- $\beta \sim \text{Normal}(50, 50)$
- $\sigma_u, \sigma_w \sim \text{Normal}_+(0, 300)$
- $\sigma \sim \text{Normal}_+(0, 500)$

The first step is to check whether the prior predictive distribution makes sense:

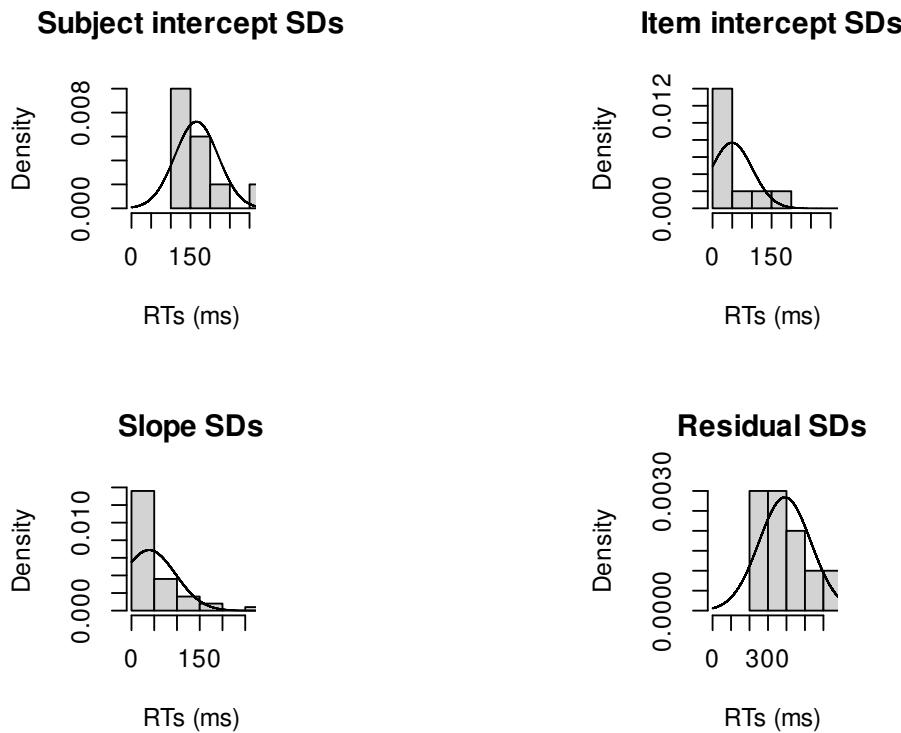


FIGURE 6.4: Empirical distributions of variance components from ten publishes studies. The solid lines represent normal distributions truncated at 0 ms.

Figure 6.5 shows that the prior predictive distributions are not too implausible, although they could be improved further. One problem is the normal distribution of the data; a log-normal distribution captures the shape of the distribution of the Grodner and Gibson (2005) data better than a normal distribution. The discrepancy between the Grodner and Gibson (2005) data and our prior predictive distribution implies that we might be using the wrong likelihood. Another problem is that the reading times in the prior predictive distribution can be negative—this is also a consequence of our using the wrong likelihood. An exercise in this chapter will ask you to fit a model with a log-normal likelihood and informative priors based on previous data. Nevertheless, since our running example uses a Gaussian likelihood on reading times in milliseconds, we can retain these priors for now.

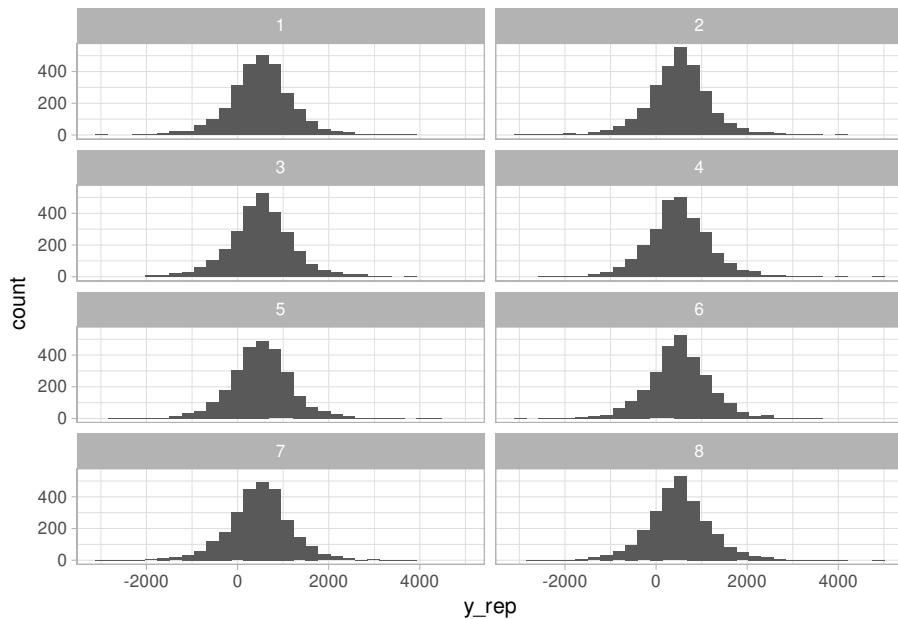


FIGURE 6.5: Prior predictive distributions from the model to be used for the Grodner and Gibson data analysis.

The sensitivity analysis could then be displayed, showing the posteriors under different prior settings. Figures 6.6 and 6.7 show the posteriors under two distinct sets of priors.

What can one do if one doesn't know absolutely anything about one's research problem? An example is the power posing data that we encountered in Chapter 4, in an exercise in section 4.6. Here, we investigated the change in testosterone levels after the subject was either asked to adopt a high power pose or a low power pose (a between-subjects design). Not being experts in this domain, we may find ourselves stumped for priors. In such a situation, it could be defensible to use vague priors like $\text{Cauchy}(0, 2.5)$. However, as discussed in a later chapter, if one is committed to doing a Bayes factor analysis, then we are obliged to think carefully about plausible a priori values of the effect. This would require consulting one or more experts. We turn next to the topic of eliciting priors from experts.

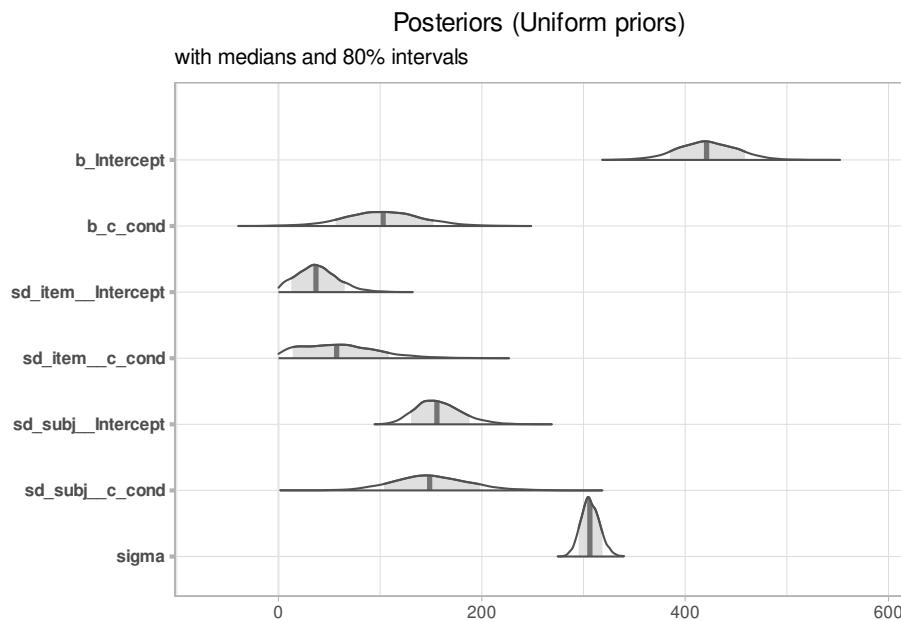


FIGURE 6.6: Posterior distributions of parameters for the English relative clause data, using uniform priors ($\text{Uniform}(0,2000)$) on the intercept and slope.

6.2 Eliciting priors from experts

It can happen that one is working on a research problem where either one's own prior knowledge is lacking, or we need to incorporate a range of competing prior beliefs into the analysis. In such situations, it becomes important to elicit priors from experts other than oneself. Although informal elicitation can be a perfectly legitimate approach, there does exist a well-developed methodology for systematically eliciting priors in Bayesian statistics (O'Hagan et al. 2006).

The particular method developed by O'Hagan and colleagues comes with an R package called 'SHELF', which stands for the Sheffield Elicitation Framework; the method was developed by statisticians at the University of Sheffield, UK. SHELF is available from <http://www.tonyohagan.co.uk/shelf/>.

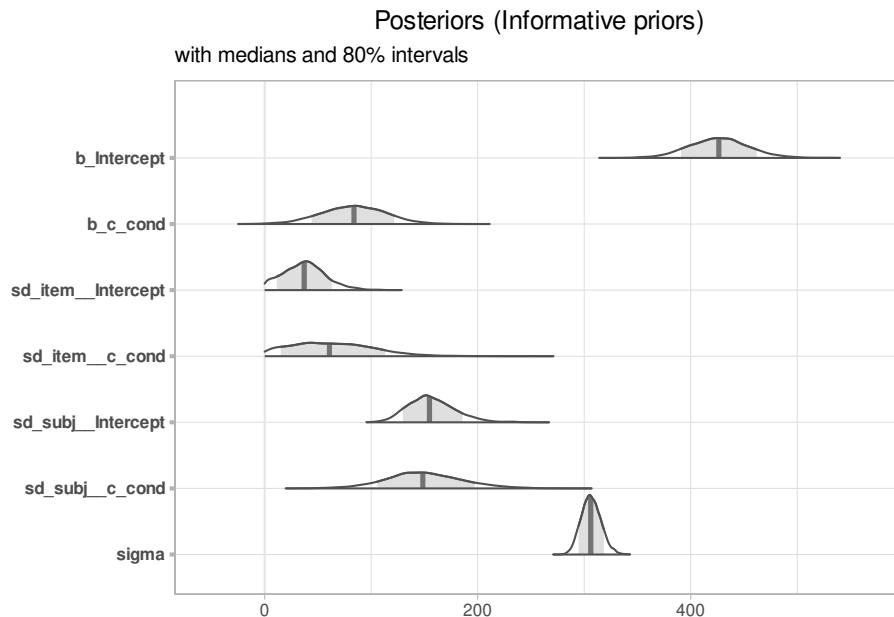


FIGURE 6.7: Posterior distributions of parameters for the English relative clause data, using relatively informative priors on the intercept and slope.

This framework comes with a detailed set of instructions and a fixed procedure for eliciting distributions. It also provides detailed guidance on documenting the elicitation process, thereby allowing a full record of the elicitation process to be created. The SHELF procedure works as follows. There is a facilitator and an expert (or a group of experts; we will consider the single expert case here, but one can easily extend the approach to multiple experts).

- A pre-elicitation form is filled out by the facilitator in consultation with the expert. This form sets the stage for the elicitation exercise and records some background information, such as the nature of the expertise of the assessor.
- Then, an elicitation method is chosen. Simple methods are the most effective. One effective approach is the quartile method. The expert first decides on a lower and upper limit of possible values for the quantity to be estimated; this minimizes the effects of the “anchoring and adjustment heuristic” (O’Hagan et al. 2006), whereby experts tend to anchor their subsequent estimates of quartiles based on their first judgement

of the median. Following this, a median value is decided on, and lower and upper quartiles are elicited. The SHELF package has functions to display these quartiles graphically, allowing the expert to adjust them at this stage if necessary. It is important for the expert to confirm that, in his/her judgement, the four partitioned regions that result have equal probability.

- The elicited distribution is then displayed as a density (several choices of probability density functions are available, but we will usually use the normal or the truncated normal in this chapter); this graphical summary serves to give feedback to the expert. The parameters of the distribution are also displayed. Once the expert agrees to the final density, the parameters can be considered the expert's judgement regarding the prior distribution of the bias. One can consult multiple experts and either combine their judgements into one prior, or consider each expert's prior separately in a sensitivity analysis.

When eliciting priors from more than one expert, one can elicit the priors separately and then use the priors separately in a sensitivity analysis. This approach takes each individual expert's opinion in interpreting the data and can be a valuable sensitivity analysis (for an example from psycholinguistics, see Vasishth and Engelmann 2021). Alternatively, one can pool the priors together (see Spiegelhalter, Abrams, and Myles 2004 for discussion) and create a single composite prior; this would amount to an average of the differing opinions about prior distributions. A third approach is to elicit a consensus prior by bringing all the experts together and eliciting a prior from the group in a single setting. Of course, these approaches are not mutually exclusive. One of the hallmark properties of Bayesian analysis is that the posterior distribution of the parameter of interest can be investigated in light of differing prior beliefs and the data (and of course the model).

6.3 Deriving priors from meta-analyses

Meta-analysis is a very useful tool for evidence synthesis; although meta-analysis has been used widely in clinical research (Higgins and Green 2008; Sutton et al. 2012; DerSimonian and Laird 1986; Normand 1999), it

is only rarely seen in psychology and linguistics. Random-effects meta-analysis (discussed in a later chapter in detail) is an especially useful tool in cognitive science. Meta-analysis is not a magic bullet; this is because of publication bias—usually only (supposedly) newsworthy results are published, leading to a skewed picture of the effects. As a consequence, meta-analysis will yield biased estimates; but they still tell us something about what we know so far from published studies. Some prior information is better than no information. As long as one remains aware of the limitations of meta-analysis, one can still use them effectively to study one's research questions.

We begin with observed effects y_n (e.g., estimated difference between two conditions) and their estimated standard errors (SEs); the SEs serve as an indication of the precision of the estimate, with larger SEs implying a low-precision estimate. Once we have collected the observed estimates (e.g., from published studies), we can define an assumed underlying generative process whereby each study $n = 1, \dots, N$ has an unknown true mean ζ_n :

$$y_n \sim \text{Normal}(\zeta_n, SE_n)$$

A further assumption is that each unknown true mean ζ_n in each study is generated from a distribution that has some true overall mean ζ , and standard deviation τ . The standard deviation τ reflects between-study variation, which could be due to different subjects being used in each study, different lab protocols, different methods, etc.

$$\zeta_n \sim \text{Normal}(\zeta, \tau)$$

This kind of meta-analysis is actually the familiar hierarchical model we have already encountered. As in hierarchical models, hyperpriors have to be defined for ζ and τ . A useful application of this kind of meta-analysis is to derive a posterior distribution for ζ based on the available evidence; this posterior can be used (e.g., with a normal approximation) as a prior for a future study.

A simple example is the published data on Chinese relative clauses; the data are from Vasishth (2015). Table 6.2 shows the mean difference between the object and subject relative, along with the standard error, that was derived from published reading studies on Chinese relatives.

Suppose that we want to do a new study investigating the difference be-

TABLE 6.2: The difference between object and subject relative clause reading times (effect), along with their standard errors (SE), from different published reading studies on Chinese relative clauses.

study.id	study	y	se
1	1 Hsiao et al 03	50.0	25.0
4	4 Vas. et al 13, E2	82.6	41.2
5	5 Vas. et al 13, E3	-109.4	54.8
6	6 Jaeg. et al 15, E1	55.6	65.1
7	7 Jaeg. et al 15, E2	81.9	36.3
9	9 Wu 09	50.0	23.0
10	10 Qiao et al 11, E1	-70.0	42.0
11	11 Qiao et al 11, E2	6.2	19.9
12	12 Lin & Garn. 11, E1	-100.0	30.0
14	14 Chen et al 08	75.0	35.5
15	15 C Lin & Bev. 06	100.0	80.0

tween object and subject relative clauses, and suppose that in the sensitivity analysis, one of the priors we want is an empirically justifiable informative prior. Of course, the sensitivity analysis will also contain relatively uninformative priors; we have seen examples of such priors in the previous chapters.

We can derive an empirically justified prior by conducting a random effects meta-analysis. We postpone discussion of how exactly to fit such a model to chapter 13; here, we simply report the posterior distribution of the overall effect ζ based on the prior data, ignoring the details of model fitting.

The posterior distribution of ζ is shown in Figure 6.8.

What we can derive from this posterior distribution of ζ is a normal approximation that represents what we know so far about Chinese relatives, based on the existing data. The mean of the posterior is 17, and the width of the 95% credible intervals is $25 - 1 = 24$. Since the 95% credible interval has a width that is approximately four times the standard deviation (assuming a normal distribution), we can work out the standard deviation by di-

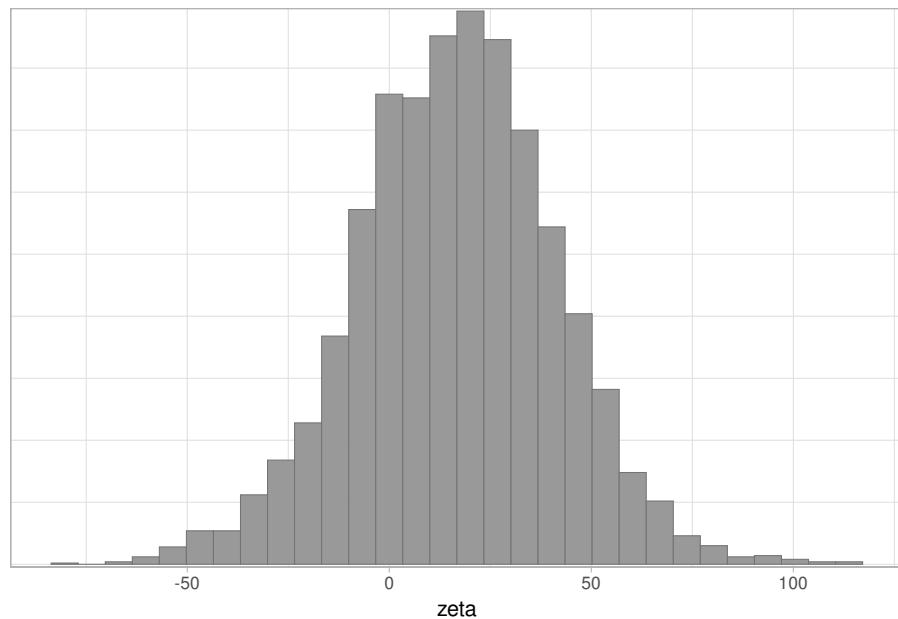


FIGURE 6.8: The posterior distribution of the difference between object and subject relative clause processing in Chinese relative clause data, computed from a random-effects meta-analysis using published Chinese relative clause data from reading studies.

viding the width by four: 6. Given these estimates, we could use a Normal distribution with mean 17 and standard deviation 6 as an informative prior in a sensitivity analysis.

As an example, we will analyze a data set from Gibson and Wu (2013) that was not part of the above meta-analysis, and we will use the meta-analysis posterior as an informative prior. First, we load the data and sum-code the predictor:

```
data("df_gibsonwu")
df_gibsonwu <- df_gibsonwu %>%
  mutate(c_cond = if_else(type == "obj-ext", 1/2, -1/2))
```

Because we will now use a log-normal likelihood for the reading time data, we need to work out what the meta-analysis posterior of ζ corresponds to on the log scale. The grand mean reading time of the Gibson and Wu (2013)

data on the log scale is 6.1. In order to arrive at approximately the mean difference of 17 ms, the log-scale value of the mean difference would be 0.041, with a 95% credible interval $[-0.079, 0.145]$, which implies a standard deviation of $(0.0145 - (-0.079))/4 = 0.2$. The calculations are shown below (see Box 4.3 in chapter 4 for an explanation of a model with a log-normal likelihood).

```
int <- mean(log(df_gibsonwu$rt))
b <- 0.041
exp(int + b/2) - exp(int - b/2)

## [1] 17.6

lower <- -0.079
exp(int + lower/2) - exp(int - lower/2)

## [1] -33.9

upper <- 0.145
exp(int + upper/2) - exp(int - upper/2)

## [1] 62.2
```

As always, we will do a sensitivity analysis, using (relatively) uninformative priors on the slope parameter ($\text{Normal}(0,1)$), and the meta-analysis prior.

```
## uninformative priors on the parameters of interest
## and on the variance components:
fit_gibsonwu_log <- brm(rt ~ c_cond +
                           (c_cond | subj) +
                           (c_cond | item),
                           family = lognormal(),
                           prior =
                           c(
                             prior(normal(6, 1.5), class = Intercept),
```

TABLE 6.3: A summary of the posteriors under a relatively uninformative prior and an informative prior based on a meta-analysis, for the Chinese relative clause data from Gibson and Wu, 2013.

Priors	Mean	Lower	Upper
Normal(0,1)	-0.07	-0.18	0.04
Normal(0.041, 0.2)	-0.06	-0.17	0.04

```

prior(normal(0, 1), class = b),
prior(normal(0, 1), class = sigma),
prior(normal(0, 1), class = sd),
prior(lkj(2), class = cor)
),
data = df_gibsonwu
)
## meta-analysis priors:
fit_gibsonwu_ma <- brm(rt ~ c_cond +
(c_cond | subj) +
(c_cond | item),
family = lognormal(),
prior =
c(
prior(normal(6, 1.5), class = Intercept),
prior(normal(0.041, 0.2), class = b),
prior(normal(0, 1), class = sigma),
prior(normal(0, 1), class = sd),
prior(lkj(2), class = cor)
),
data = df_gibsonwu
)

```

A summary of the posteriors under the Normal(0,1) and the meta-analysis prior is shown in Table 6.3. In this particular case, the posteriors are not strongly influenced by the two different priors.

TABLE 6.4: A summary of the posteriors under a (relatively) uninformative prior ($\text{Normal}(0,1)$), a prior based on the original data, and a meta-analysis prior, for data from a replication attempt of Gibson and Wu, 2013.

Priors	Mean	Lower	Upper
$\text{Normal}(0,1)$	-0.08	-0.21	0.05
$\text{Normal}(-0.07, 0.2)$	-0.08	-0.20	0.04
$\text{Normal}(0.041, 0.2)$	-0.07	-0.19	0.05

6.4 Using previous experiments' posteriors as priors for a new study

In a situation where we are attempting to replicate a previous study's results, we can use an informative prior for the analysis of the replication attempt by deriving a prior from the previous study's posterior distribution. In the previous chapter, we encountered this in one of the exercises: Given data on Chinese relatives (Gibson and Wu 2013), we want to replicate the effect with a new data set that has the same design but different subjects. The data from the replication attempt is from Vasishth et al. (2013).

The first dataset from Gibson and Wu (2013) was analyzed in the previous section using (relatively) uninformative priors. We can extract the mean and standard deviation of the posterior, and use that to derive an informative prior for the replication attempt.

Now, for the replication study, we can use this posterior (with a normal approximation), if we want to build on what we learned from the original Gibson and Wu (2013) study. As usual, we will do a sensitivity analysis: one model is fit with an uninformative prior on the parameter of interest, $\text{Normal}(0,1)$, as we did in the preceding section; and another model will be fit with the informative prior $\text{Normal}(-0.071, 0.21)$. For good measure, we can also include a model with a prior derived from the meta-analysis in the preceding section (the posterior of the ζ parameter).

Table 6.4 summarizes the different posteriors under the three prior specification. Again, in this case, the differences in the posteriors are small.

6.5 Summary

Working out appropriate priors for one's research problem is essentially a Fermi problem. One can use several different strategies for working out priors: introspection, a literature review, computing statistics from existing data, conducting a meta-analysis, using posteriors from existing data as priors for a new, closely related study, or formally eliciting priors from domain experts. If a prior specification is too vague, this can lead to slow convergence or convergence problems, and would lead to biased Bayes factors if these are computed from the model; and if a prior is too informative, this can bias the posterior. and can also lead to convergence problems. This inherent potential for bias in prior specification should be formally investigated using sensitivity analyses (with a collection of uninformative, skeptical, and informative priors of various types), and prior and posterior predictive checks. Although prior specification seems like a daunting task to the beginning student of Bayes, with time and experience one can develop a very well-informed set of priors for one's research problems.

6.6 Further reading

For interesting (and amusing) examples of Fermi solutions to questions, see <https://what-if.xkcd.com/84/>. Two important books, Mahajan (2010) and Mahajan (2014), unpack the art of approximation in mathematics and other disciplines; the approach presented in these books is closely related to the art of Fermi-style approximation. An excellent presentation of prior elicitation is in O'Hagan et al. (2006). Useful discussions about priors are provided in Lunn et al. (2012), Spiegelhalter, Abrams, and Myles (2004), Gelman, Simpson, and Betancourt (2017), and Simpson et al. (2017). The Stan website also includes some guidelines: Prior distributions for rstanarm models in <https://mc-stan.org/rstanarm/articles/priors.html>; and prior choice recommendations in <https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations>. Browne and Draper

(2006) and Gelman (2006) discuss prior specifications in hierarchical models.

7

Workflow

Although complex and powerful, Bayesian analysis tools (such as *brms*) are today easily accessible to lay users, and although these tools greatly facilitate Bayesian computations, the model specification is still (as it should be) the responsibility of the user. In the previous chapters (e.g., chapter 3), we have treated some steps needed to arrive at a useful and robust analysis. In this chapter, we bring these ideas together to delineate a principled approach to developing a workflow. This chapter is based on a recent introduction of a principled Bayesian workflow to cognitive science (Schad, Betancourt, and Vasishth 2019, for a revised published version see 2020).

Much research has been carried out in recent years to develop tools to ensure robust Bayesian data analyses (e.g., Gabry et al. 2017; Talts et al. 2018). One of the most recent products of this research has been the formulation of a principled Bayesian workflow for conducting a probabilistic analysis (Betancourt 2018; Schad, Betancourt, and Vasishth 2019). This workflow provides an initial coherent set of steps to take for a robust analysis, leaving room for further improvements and methodological developments. At an abstract level, parts of this workflow can be applied to any kind of data analysis, be it frequentist or Bayesian, be it based on sampling or on analytic procedures.

Here, we introduce parts of this principled Bayesian workflow by illustrating its use with experimental data from a reading-time experiment. We demonstrate how to implement analyses in R, and use the R package *brms* (Bürkner 2017) for statistical analysis. Some parts of this principled Bayesian workflow are specifically recommended when using advances/non-standard models.

A number of questions should be asked when fitting a model, and several checks should be performed to validate a probabilistic model. Before going into the details of this discussion, we first treat the process of model

building, and how different traditions have yielded different approaches to this questions.

7.1 Model building

One strategy for model building is to start with a minimal model that captures just the phenomenon of interest but not much other structure in the data. For example, this could be a linear model with just the factor or covariate of main interest. For this model, we perform a number of checks described in detail in the following sections. If the model passes all checks and does not show signs of inadequacy, then it can be applied in practice and we can be confident that the model provides reasonably robust inferences on our scientific question. However, if the model shows signs of trouble on one or more of these checks, then the model may need to be improved. Alternatively, we may need to be more modest with respect to our scientific question. For example, in a repeated measures dataset, we may be interested in estimating the correlation parameter between by-group adjustments (their random effects correlation) based on a sample of 30 subjects. If model analysis reveals that our sample size is not sufficiently large to estimate the correlation reliably, then we may need to either increase our sample size, or give up our plan.

During the model building process, we make use of an aspirational model \mathcal{M}_A : we mentally imagine a model with all the possible details that the phenomenon and measurement process contain; i.e., we imagine a model that one would fit if there were no limitations in resources, time, mathematical and computational tools, subjects, and so forth. It would contain all systematic effects that might influence the measurement process. For example, influences of time or heterogeneity across individuals. This should be taken to guide and inform model development; such a procedure prevents random walks in model space during model development. The model has to consider both the latent phenomenon of interest as well as the environment and experiment used to probe it.

In contrast to the aspirational model, the initial model \mathcal{M}_1 may only contain enough structure to incorporate the phenomenon of core scientific

interest, but none of the additional aspects/structures relevant for the modeling or measurement. The additional, initially left-out structures, which reflect the difference between the initial (\mathcal{M}_1) and the aspirational model (\mathcal{M}_A), can then be probed for using specifically designed summary statistics. These summary statistics can thus inform model expansion from the initial model \mathcal{M}_1 into the direction of the aspirational model \mathcal{M}_A . If the initial model proves inadequate, then the aspirational model and the associated summary statistics guide model development. If the expanded model is still not adequate, then another cycle of model development is conducted.

The range of prior and posterior predictive checks discussed in the following sections serve as a basis for a principled approach to model expansion. The notion of *expansion* is critical here. If an expanded model does not prove more adequate, one can always fall back to the previous model version.

As an alternative analysis strategy, a tradition in the cognitive and other experimental sciences relies on *maximal models* for a given experimental design (Barr et al. 2013). This maximal model contains all effects from experimental manipulations (main effects and interactions) as well as differences in these effects varying across subjects, items, or other random factors in the experimental design. That is, this model is maximal within the scope of a linear regression. Such a maximal model provides an alternative starting point for the principled Bayesian workflow. In this case, the focus does not lie so much on model expansion. Instead, for maximal models the workflow can be used to specify priors encoding domain expertise (possibly to ensure computational faithfulness and model sensitivity), and to ensure model adequacy. Some steps in the principled Bayesian workflow (e.g., the computationally more demanding steps of assessing computational faithfulness and model sensitivity) may even be performed only for models coded in Stan or only once for a given research program, where similar designs are repeatedly used. We will explain this in more detail below.

In the maximal model, “maximal” refers to maximal specification of the variance components within the scope of the linear regression approximation, not maximal with respect to the actual data generating process. Models are still bound by the linear regression structure and hence can-

not capture effects such as selection bias in the data, dynamical changes in processes across time, or measurement error. Importantly, however, these “maximal” models are not the aspirational model, which is an image of the true data generating process, but rather just an approximation. Indeed, aiming to formulate models closer to the aspiration model, which go beyond the linear model framework, may be one reason to consider investing in learning to express models directly within probabilistic programming languages such as Stan instead of the limited range of models provided in packages like *brms*.

Finally, sometimes the results from the Bayesian workflow will show that our experimental design or data is not sufficient to answer our scientific question at hand. In this case, ambition needs to be reduced, or new data needs to be collected, possibly with a different experimental design more sensitive to the phenomenon of interest.

One important development in open science practices is pre-registration of experimental analyses before the data are collected (Chambers 2019). This can be done using online-platforms such as the Open Science Foundation (<https://osf.io/>) or AsPredicted (<https://aspredicted.org/>) (but see Szollosi et al. 2019). What information can or should one document in preregistration of the Bayesian workflow? If one plans on using the maximal model for analysis, then this maximal model, including contrast coding (Schad et al. 2020), fixed effects, and random effects should be described. In the case of incremental model building, if a model isn’t a good fit to the data, then any resulting inference will be limited if not useless, so a rigid preregistration is useless unless one knows exactly what the model is. Thus, the deeper issue with preregistration is that a model cannot be confirmed until the phenomenon *and* experiment are all extremely well understood. One practical possibility is to describe the initial and the aspirational model, and the incremental strategy used to probe the initial model to move more towards the aspirational model. This can also include delineation of summary statistics that one plans to use for probing the tested models. Even if it is difficult to spell out the aspirational model fully, it can be useful to preregister the initial model, summary statistics, and the principles one intends to apply in model selection. While the maximal modeling approach clearly reflects confirmatory hypothesis testing, the incremental model building strategy towards the aspirational

model may be seen as lying at the boundary between confirmatory and exploratory, and becomes more confirmatory the more clearly the aspirational model can be spelled out *a priori*.

7.2 Principled questions on a model

What characterizes a useful probabilistic model? A useful probabilistic model should be consistent with domain expertise. Moreover, a useful probabilistic model should be rich enough to capture the structure of the true data generating process needed to answer scientific questions. When very complex or non-standard models are developed, there are two additional requirements that must be met (we will briefly touch upon these in the present chapter): it is key for the model to allow accurate posterior approximation, and the model must capture enough of the experimental design to give useful answers to our questions.

So what can we do aiming to meet these properties of our probabilistic model? In the following, we will outline a number of analysis steps to take and questions to ask in order to improve these properties for our model.

In a first step, we will use prior predictive checks to investigate whether our model is consistent with our domain expertise. Moreover, posterior predictive checks assess model adequacy for the given dataset, that is, they investigate the question whether the model captures the relevant structure of the true data generating process. We will also briefly discuss two additional steps that are computationally very expensive, and can be used e.g., when coding advanced/non-standard models, but which are also part of the principled workflow: this includes investigating computational faithfulness by studying whether posterior estimation is accurate, and it includes studying model sensitivity and the question whether we can recover model parameters with the given design and model.

7.2.1 Prior predictive checks: Checking consistency with domain expertise

The first key question for checking the model is whether the model and the distributions of prior parameters are consistent with domain expertise.

Prior distributions can be selected based on prior research or plausibility. However, for complex models it is often difficult to know which prior distributions should be chosen, and what consequences distributions of prior model parameters have for expected data. A viable solution is to use prior distributions to simulate hypothetical data from the model and to check whether the simulated data are plausible and consistent with domain expertise. This approach is often much easier to judge compared to assessing prior distributions in complex models directly.

In practice, this approach can be implemented by the following steps:

1. Take the prior $p(\Theta)$ and randomly draw a parameter set Θ_{pred} from it: $\Theta_{pred} \sim p(\Theta)$
2. Use this parameter set Θ_{pred} to simulate hypothetical data y_{pred} from the model: $y_{pred} \sim p(y | \Theta_{pred})$

To assess whether prior model predictions are consistent with domain expertise, it is useful to compute summary statistics of the simulated data $t(y_{pred})$. The distribution of these summary statistics can be visualized using, for example, histograms (see Figure 7.1). This can quickly reveal whether the data falls in an expected range, or whether a substantial amount of extreme data points are expected a priori. For example, in a study using self-paced reading times, “extreme” values may be considered to be reading times smaller than 50 ms or larger than 2000 ms. Reading times for a word larger than 2000 ms are not impossible, but would be implausible and largely inconsistent with domain expertise. Experience with reading studies shows that a small number of observations may actually take extreme values. However, if we observe a large number of extreme data points in the hypothetical data, and if these are inconsistent with domain expertise, the priors or the model should be adjusted so that they yield hypothetical data within the range of reasonable values.

Choosing good summary statistics is more an art than a science. The choice of summary statistics will be crucial, however, as they provide key markers of what we want the model to account for in the data. They should thus be carefully chosen and designed based on the expectations that we have about the true data generating process and about the kinds of structures and effects we expect the data may exhibit. Interestingly, summary

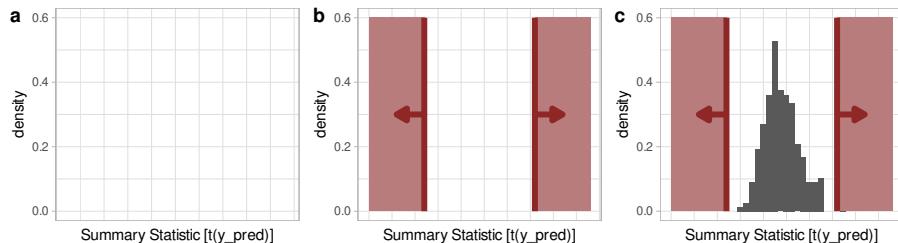


FIGURE 7.1: Prior predictive checks. a) In a first step, define a summary statistic that one wants to investigate. b) Second, define extremity thresholds (shaded areas), for which one does not expect a lot of prior data. c) Third, simulate prior model predictions for the data (histogram) and compare them with the extreme values (shaded areas).

statistics can also be used to critique the model: if someone wants to criticize an analysis, then they can formalize that criticism into a summary statistic they expect to show undesired behavior. Such criticism can serve as a very constructive way to write reviews in a peer-review setting. Here, we will show some examples of useful summary statistics below when discussing data analysis for a concrete example dataset.

Choosing good priors will be particularly relevant in cases where the likelihood is not precise (see Figure 7.2, in particular g-i). In linear mixed-effects models, for example, this often occurs in cases where a “maximal model” is fitted for a small dataset that does not constrain estimation of all random effects variance and covariance parameters.¹

In such situations, using a prior in a Bayesian analysis (or a more informative prior rather than a relatively uninformative one) should incorporate just enough domain expertise to suppress extreme, although not impossible parameter values. This may allow the model to be fit, as the posterior is now sufficiently constrained. Thus, introducing prior information in Bayesian computation allows us to fit and interpret models that cannot be validly estimated using frequentist tools.

¹In frequentist methods (such as implemented in the lme4 package in the lmer program), this problem manifests itself as problems with convergence of the optimizer, which indicates that the likelihood is too flat and that the parameter estimates are not constrained by the data.

A welcome side-effect of incorporating more domain expertise (into what still constitutes weakly informative priors) is thus more concentrated prior distributions, which can facilitate Bayesian computation. This allows more complex models to be estimated; that is, using prior knowledge can make it possible to fit models that could otherwise not be estimated using the available tools. In other words, incorporating prior knowledge can allow us to get closer to the aspirational model in the iterative model building procedure. Moreover, more informative priors also lead to faster convergence of MCMC algorithms.

```
## Warning: It is deprecated to specify `guide = FALSE` to remove a
## guide. Please use `guide = "none"` instead.
```

```
## Warning: It is deprecated to specify `guide = FALSE` to remove a
## guide. Please use `guide = "none"` instead.
```

```
## Warning: It is deprecated to specify `guide = FALSE` to remove a
## guide. Please use `guide = "none"` instead.
```

```
## Warning: It is deprecated to specify `guide = FALSE` to remove a
## guide. Please use `guide = "none"` instead.
```

```
## Warning: It is deprecated to specify `guide = FALSE` to remove a
## guide. Please use `guide = "none"` instead.
```

```
## Warning: It is deprecated to specify `guide = FALSE` to remove a
## guide. Please use `guide = "none"` instead.
```

```
## Warning: It is deprecated to specify `guide = FALSE` to remove a
## guide. Please use `guide = "none"` instead.
```

```
## Warning: It is deprecated to specify `guide = FALSE` to remove a
## guide. Please use `guide = "none"` instead.
```

```
## Warning: It is deprecated to specify `guide = FALSE` to remove a
## guide. Please use `guide = "none"` instead.
```

Incorporating more domain expertise into the prior also has crucial conse-

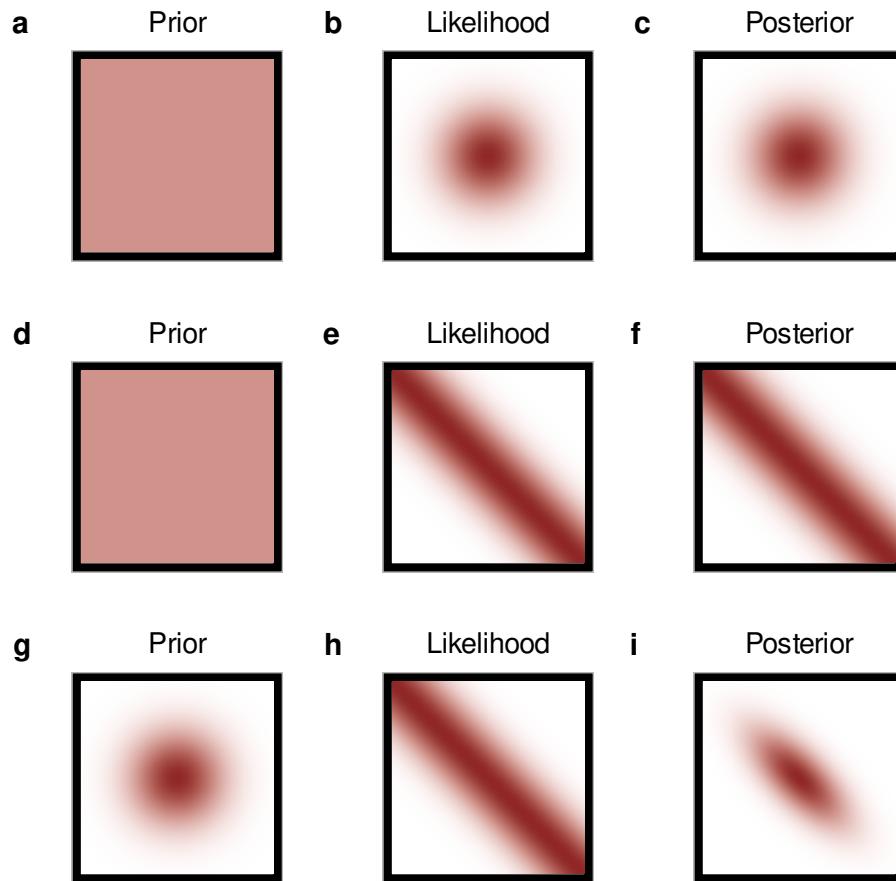


FIGURE 7.2: The role of priors for informative and uninformative data. a)-c) When the data provides good information via the likelihood (b), then a flat uninformative prior (a) is sufficient to obtain a concentrated posterior (c). d)-f) When the data does not sufficiently constrain the parameters through the likelihood (e), then using a flat uninformative prior (d) also leaves the posterior (f) widely spread out. g)-i) When the data does not constrain the parameter through the likelihood (h), then including domain expertise through an informative prior (g) can help to constrain the posterior (i) to reasonable values.

quences for Bayesian modeling when computing Bayes factors (see chapter 16, on Bayes factors). Bayes factors are highly sensitive to the prior, and in particular to the prior uncertainty. Priors are thus never uninformative when it comes to Bayes factors. Choosing (relatively) uninformative priors makes it very difficult to find posterior evidence in favor of an expanded model, and will often support the simpler model. For example, in nested model comparison of hierarchical models, large prior uncertainty implies the assumption that the effect of interest could be very (implausibly) large. Using Bayes factors for such nested model comparison with high prior uncertainty thus tests whether there is evidence for a very big effect size for the predictor term in question, which is usually not supported by the data (because the relatively uninformative prior covers implausibly large effect sizes). When using a weakly informative or even an informative prior with much smaller uncertainty, the Bayes factor tests whether there is evidence for a small effect of the additional predictor term, which is much more likely to be the case. Thus, using prior knowledge and specifying priors with reasonable uncertainty (rather than uninformative priors with large uncertainty) are crucial when carrying out model comparison using Bayes factors (Schad et al. 2021).

Importantly, this first step simulates from the *prior predictive distribution*, which specifies how the prior interacts with the likelihood. Mathematically, it computes an average (the integral) over different possible (prior) parameter values. The prior predictive distribution is (also see chapter 3):

$$\begin{aligned} p(\mathbf{y}_{pred}) &= \int p(\mathbf{y}_{pred}, \Theta) d\Theta = \int p(\mathbf{y}_{pred} | \Theta)p(\Theta) d\Theta \\ &= \int \text{likelihood}(\mathbf{y}_{pred} | \Theta) \cdot \text{prior}(\Theta) d\Theta \end{aligned} \tag{7.1}$$

As a concrete example, suppose we assume that our likelihood is a Normal distribution with mean μ and standard deviation σ . Suppose that we now define the following priors on the parameters: $\mu \sim \text{Normal}(0, 1)$, and $\sigma \sim \text{Uniform}(1, 2)$. We can generate the prior predictive distribution using the following steps:

- Do the following 100000 times:
 - Take one sample m from a $\text{Normal}(0, 1)$ distribution
 - Take one sample s from a $\text{Uniform}(1, 2)$ distribution

- Generate and save a data point from $\text{Normal}(m,s)$
- The generated data is the prior predictive distribution.

More complex generative processes involving repeated measures data can also be defined.

7.2.2 Computational faithfulness: Testing for correct posterior approximations

A key aim in Bayesian data analysis is to compute posterior expectations, such as the posterior mean or posterior credible intervals (quantiles) of some parameter. For some simple models and prior distributions, these posterior expectations can be computed exactly by analytical derivation. However, this is not possible in most interesting and realistic, more complex models, where analytical solutions cannot be computed. Instead, computational approximations are needed for estimation. Here, we use one option: while it is often not possible to compute the posterior exactly, it is possible to draw samples from it, and we accordingly use (Markov Chain Monte Carlo) sampling to approximate posterior expectations.

Approximations of posterior expectations can be inaccurate. For example, a computer program that is designed to sample from a posterior can be erroneous. This could involve an error in the specification of the likelihood (e.g., caused by an error in the R syntax formula), or insufficient sampling of the full density of the posterior. The sampler may be biased, sampling parameter values that are larger or smaller than the true posterior, or the variance of the posterior samples may be larger or smaller than the true posterior uncertainty. However, posterior sampling from simple and standard models should work properly in most cases. Thus, we think that in many applications, a further check of computational faithfulness may be asking for too much, and might need to be performed only once for a given research program, where different experiments are rather similar to each other. However, checking computational faithfulness can become an important issue when dealing with more advanced/non-standard models (such as those discussed in the later chapters of this book). Here, errors in the specification of the likelihood can occur more easily.

Given that posterior approximations can be inaccurate, it is important to

design a procedure to test whether the posterior approximation of choice is indeed accurate, e.g., that the software used to implement the sampling works without errors for the specific problem at hand. This checking can be performed using simulation-based calibration (SBC; Talts et al. 2018; Schad, Betancourt, and Vasishth 2019). This is a very simulation-intensive procedure, which can take a long time to run for considerably complex models and larger datasets. We do not discuss SBC in detail here, but refer the reader to its later treatment in chapter 19, where SBC is applied for models coded in Stan directly, as well as to the description in Schad, Betancourt, and Vasishth (2019).

Even powerful tools like Hamiltonian Monte Carlo (HMC) sampling do not always provide accurate posterior estimation. For a given experimental design, model, and dataset, it is therefore important to check computational faithfulness. Assuming that our posterior computations are accurate and faithful, we can take a next step, namely looking at the sensitivity of the model analyses.

7.2.3 Model sensitivity

What can we realistically expect from the posterior of a model, and how can we check whether these expectations are justified for the current setup? First, we might expect that the posterior recovers the true parameters generating the data without bias. That is, when we simulate hypothetical data based on a true parameter value, we may expect that the posterior mean exhibits the same value. However, for a given model, experimental design, and dataset, this expectation may or may not be justified. Indeed, parameter estimation for some, e.g., non-linear, models may be biased, such that the true value of the parameter can practically not be recovered from the data. At the same time, we might expect from the posterior that it is highly informative with respect to the parameters that generated the data. That is, we may hope for small posterior uncertainty (a small posterior standard deviation) relative to our prior knowledge. However, posterior certainty may sometimes be low. Some experimental designs, models, or datasets may yield highly uninformative estimates, where uncertainty is not reduced compared to our prior information. This can be the case when we have very little data, or when the experimental design does

not allow us to constrain certain model parameters; i.e., the model is too complex for the experimental design.

To study model sensitivity, one can investigate two questions about the model:

- 1) How well does the estimated posterior mean match the true simulating parameter?
- 2) How much is uncertainty reduced from the prior to the posterior?

To investigate these questions, it is again possible to perform extensive simulation studies. This is crucial to do for complex, non-standard, or cognitive models, but may be less important for simpler and more standard models. Indeed, the same set of simulations can be used that are also used in SBC. Therefore, both analyses can be usefully applied in tandem. Again, here we skip the details of how these computations can be implemented, and refer the interested reader to Schad, Betancourt, and Vasishth (2019).

7.2.4 Posterior predictive checks: Does the model adequately capture the data?

“All models are wrong but some are useful.” (Box 1979, 2). We know that our model probably does not fully capture the true data generating process, which is noisily reflected in the observed data. Our question therefore is whether our model is *close enough* to the true process that has generated the data, and whether the model is useful for informing our scientific question. To compare the model to the true data generating process (i.e., to the data), we can simulate data from the model and compare the simulated to the real data. This can be formulated via a *posterior predictive distribution* (see chapter 3): the model is fit to the data, and the estimated posterior model parameters are used to simulate new data.

Mathematically, the *posterior predictive distribution* is written:

$$p(\mathbf{y}_{pred} \mid \mathbf{y}) = \int p(\mathbf{y}_{pred} \mid \Theta) p(\Theta \mid \mathbf{y}) d\Theta \quad (7.2)$$

Here, the observed data \mathbf{y} is used to infer the posterior distribution over

model parameters, $p(\Theta \mid \mathbf{y})$. This is combined with the model or likelihood function, $p(\mathbf{y}_{pred} \mid \Theta)$, to yield new, now simulated, data, \mathbf{y}_{pred} . The integral $\int d\Theta$ indicates averaging across different possible values for the posterior model parameters (Θ).

As mentioned in chapter 3, we can't evaluate this integral exactly: Θ can be a vector of many parameters, making this a very complicated integral with no analytical solution. However, we can approximate it using sampling. Specifically, we can obtain samples from the posterior distribution, e.g., using HMC or a different MCMC sampling scheme. We can now use each of the posterior samples as parameters to simulate new data from the model. This procedure then approximates the integral and yields an approximation to the posterior predictive distribution.

To summarize, in the posterior predictive distribution, the model is fit to the data, and the estimated posterior model parameters are used to simulate new data. Critically, the question then is how close the simulated data is to the observed data.

One approach is to use features of the data that we care about, and to test how well the model can capture these features. Indeed, we had already defined summary statistics in the prior predictive checks. We can now compute these summary statistics for the data simulated from the posterior predictive distribution. This will yield a distribution for each summary statistic. In addition, we compute the summary statistic for the observed data, and can now check whether the data falls within the distribution of the model predictions (cf. Figure 7.3a), or whether the model predictions are far from the observed data (see Figure 7.3b). If the data is in the distribution of the model, then this supports model adequacy. If we observe a large discrepancy, then this indicates that our model likely misses some important structure of the true process that has generated the data, and that we have to use our domain expertise to further improve the model. Alternatively, a large discrepancy can be due to the data being an extreme observation, which was nevertheless generated by the process captured in our model. In general, we can't discriminate between these two possibilities. Consequently, we have to use our best judgement as to which possibility is more relevant, in particular changing the model only if the discrepancy is consistent with a known missing model feature.

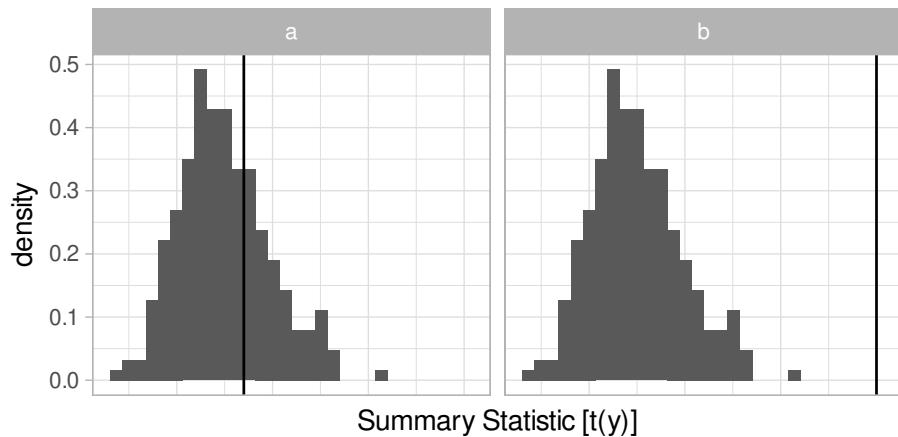


FIGURE 7.3: Posterior predictive checks. Compare posterior model predictions (histogram) with observed data (vertical line) for a specific summary statistic, $t(y)$. a) This displays a case where the observed summary statistic (vertical line) lies within the posterior model predictions (histogram). b) This displays a case where the summary statistic of the observed data (vertical line) lies clearly outside of what the model predicts a posteriori (histogram).

7.3 Exemplary data analysis

We perform an exemplary analysis of a dataset from Gibson and Wu (2013), a dataset that we have already encountered in previous chapters. The methodology they used is the familiar method (self-paced reading) that we encountered in earlier chapters. Gibson and Wu (2013) collected self-paced reading data using Chinese relative clauses. Relative clauses are sentences like: *The student who praised the teacher was very happy*. Here, the head noun, *student*, is modified by a relative clause *who...teacher*, and the head noun is the subject of the relative clause as well: the student praised the teacher. Such relative clauses are called subject relatives. By contrast, one can also have object relative clauses, where the head noun is modified by a relative clause which takes the head noun as an object. An example is: *The student whom the teacher praised was very happy*. Here, the teacher praised the student. Gibson and Wu (2013) were interested in testing the

hypothesis that Chinese shows an object relative (OR) processing advantage compared to the corresponding subject relative (SR). The theoretical reason for this processing advantage is that, in Chinese, the distance (which can be approximated by counting the number of words intervening) between the relative clause verb (*praised*) and the head noun is shorter in ORs than SRs. This prediction arises because, unlike English, the relative clause appears before the head noun in Chinese; see Gibson and Wu (2013) for a detailed explanation.

Their experimental design had one factor with two levels: (i) object relative sentences, and (ii) subject relative sentences. We use sum coding (-1, +1) for this factor, which we call ‘so’, an abbreviation for subject-object. Following Gibson and Wu (2013), we analyze reading time on the target word, which was the head noun of the relative clause. As mentioned above, in Chinese, the head noun appears after the relative clause. By the time the subject reads the head noun, they already know whether they are reading a subject or an object relative. Because the distance between the relative clause verb and the head noun is shorter in object relatives compared to subject relatives, reading the head noun is expected to be easier in object relatives.

The dataset contains reading time measurements in milliseconds from 37 subjects and from 15 items (there were 16 items originally, but one item was removed during analysis). The design is a classic repeated measures Latin square design.

7.3.1 Prior predictive checks

The first step in Bayesian data analysis is to specify the statistical model and the priors for the model parameters. As a statistical model, we use what is called the maximal model (Barr et al. 2013) for the design. Such a model includes fixed effects for the intercept and the slope (coded using sum contrast coding: +1 for object relatives, and -1 for subject relatives), correlated group-level intercepts and slopes for subjects, and correlated group-level intercepts and slopes for items. We define the likelihood as follows:

$$rt_n \sim LogNormal(\alpha + u_{subj[n],1} + w_{item[n],1} + so_n \cdot (\beta + u_{subj[n],2} + w_{item[n],2}), \sigma) \quad (7.3)$$

In `brms` syntax, the model would be specified as follows:

```
rt ~ so + ( so | subj) + (so | item), family = lognormal().
```

Because we assume a possible correlation between group-level (or random) effects, the adjustments to the intercept and slope for subjects and items have multivariate (in this case, bivariate) normal distributions with means zero, as in Equation (7.4).

$$\begin{aligned} \begin{pmatrix} u_{i,1} \\ u_{i,2} \end{pmatrix} &\sim \mathcal{N} \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_u \right) \\ \begin{pmatrix} w_{j,1} \\ w_{j,2} \end{pmatrix} &\sim \mathcal{N} \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_w \right) \end{aligned} \quad (7.4)$$

One possible standard setup for (relatively) uninformative priors which is sometimes used in reading studies (e.g., Paape, Nicenboim, and Vasishth 2017; Vasishth, Mertzen, et al. 2018) is as follows:

$$\alpha \sim Normal(0, 10) \quad (7.5)$$

$$\beta \sim Normal(0, 1) \quad (7.6)$$

$$\sigma \sim Normal_+(0, 1) \quad (7.7)$$

$$\tau_{\{1,2\}} \sim Normal_+(0, 1) \quad (7.8)$$

$$\rho \sim LKJ(2) \quad (7.9)$$

We define these priors in `brms` syntax as follows:

```
priors <- c(
  prior(normal(0, 10), class = Intercept),
  prior(normal(0, 1), class = b, coef = so),
  prior(normal(0, 1), class = sd),
  prior(normal(0, 1), class = sigma),
  prior(lkj(2), class = cor)
)
```

For the intercept (α) we use a normal distribution with mean 0 and standard deviation 10. This is on the log scale as we assume a lognormal distribution of reading times. That is, this approach assumes a priori that the intercept for reading times varies between 0 seconds and (one standard deviation) $\exp(10) = 22026$ ms (i.e., 22 sec) or (two standard deviations) $\exp(20) = 485165195$ ms (i.e., 135 hours). Going from seconds to hours within one standard deviation shows how uninformative this prior is.

Moreover, for the effect of linguistic manipulations on reading times (β), one common standard prior is to assume a mean of 0 and a standard deviation of 1 (also on the log scale). The prior on the effect size on log scale is a multiplicative factor, that is, the prediction for the effect size depends on the intercept. For an intercept of $\exp(6) = 403$ ms, a variation to one standard deviation above multiplies the base effect by 2.71, increasing the mean from 403 to $\exp(6) \times \exp(1) = 1097$. Likewise a variation to one standard deviation below multiplies the base effect by 1/2.71, decreasing the mean from 403 to $\exp(6) \times \exp(-1) = 148$. This effect size is strongly changed when assuming a different intercept: for a slightly smaller value for the intercept of $\exp(5) = 148$ ms, the expected condition difference is reduced to 37% (349 ms), and for a slightly larger value for the intercept of $\exp(7) = 1097$ ms, the condition difference is enhanced to 272% (2578 ms). Also see Box 4.3 in chapter 4 for an explanation about the non-linear behavior of the log-normal model.

Even though it seems $Normal(0, 1)$ is not entirely appropriate as a prior for the difference between object-relative and subject-relative sentences (i.e., the slope), we use it for illustrative purposes. We use the same $Normal(0, 1)$ prior for the τ parameter and σ . Finally, for the random effects correlation between the intercept and the slope, we use an lkj prior (Lewandowski, Kurowicka, and Joe 2009) with a relatively uninformative/regularizing prior parameter value of 2 (for visualization of the prior see Figure 7.4).

For the prior predictive checks, we use these priors to draw random parameter sets from the distributions, and to simulate hypothetical data using the statistical model. We load the data and code the predictor variable `so`. We next use `brms` to simulate prior predictive data from the hierarchical model.

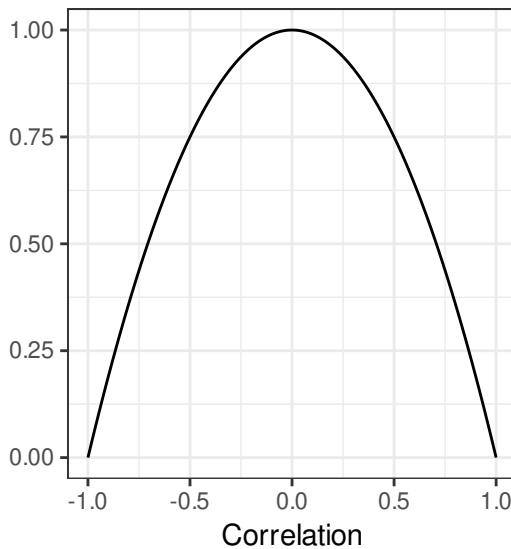


FIGURE 7.4: Shape of the LKJ prior. This is the prior density for the random effects correlation parameter, here used as a prior for the correlation between the effect size (*so*) and the intercept. The shape shows that correlation estimates close to zero are expected, and that very strong positive correlations (close to 1) or negative correlations (close to -1) are increasingly unlikely. Thus, correlation estimates are regularized towards values of zero.

```
data("df_gibsonwu")
df_gibsonwu <- df_gibsonwu %>%
  mutate(so = ifelse(type == "obj-ext", 1, -1))
set.seed(123)
fit_prior_gibsonwu <- brm(rt ~ so + (so | subj) + (so | item),
  data = df_gibsonwu,
  family = lognormal(),
  prior = priors,
  sample_prior = "only"
)
```

Based on the simulated data we can now perform prior predictive checks: we compute summary statistics, and plot the distributions of the sum-

mary statistic across simulated datasets. First, we visualize the distribution of the simulated data. For a single dataset, this could be visualized as a histogram. Here, we have a large number of simulated datasets, and thus a large number of histograms. We represent this uncertainty: for each bin, we plot the median as well as quantiles showing where 10%-90%, 20%-80%, 30%-70%, and 40%-60% of the histograms lie (for R code, see Schad, Betancourt, and Vasishth 2019). For the current prior data simulations, this shows (see Figure 7.5) that most of the hypothetical reading times are close to zero or larger than 2000 ms. It is immediately clear that the data predicted by this prior follows a very implausible distribution: it looks exponential; we would expect a lognormal distribution for reading times. Most data points take on extreme values.

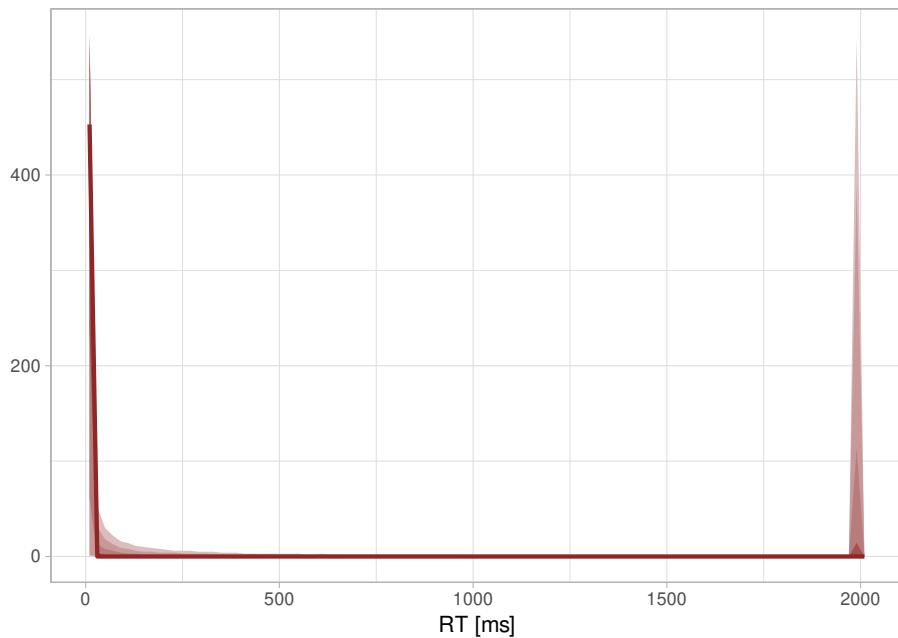


FIGURE 7.5: Prior predictive checks for a high-variance prior. Multivariate summary statistic: Distribution of histograms of reading times shows very short and also very long reading times are expected too frequently by the uninformative prior.

As an additional summary statistic, we take a look at the mean per simulated dataset and also at the variance in Figure 7.6. We create functions that use as summary statistics the mean and standard deviations; the

functions collapse all the values over 2000 ms, making the figure more readable. Then we use these summary statistics to visualize prior predictive checks using the `pp_check` function, where we enter the prior predictions (`fit_prior_gibsonwu`) and the relevant summary statistic (`mean_2000` and `sd_2000`).

```
mean_2000 <- function(x) {
  tmp <- mean(x)
  tmp[tmp > 2000] <- 2000
  tmp
}

sd_2000 <- function(x) {
  tmp <- sd(x)
  tmp[tmp > 2000] <- 2000
  tmp
}

pp_check(fit_prior_gibsonwu, type = "stat", stat = "mean_2000") +
  labs(x = "Mean RT [ms]") +
  theme(legend.position = "none")

pp_check(fit_prior_gibsonwu, type = "stat", stat = "sd_2000") +
  labs(x = "Standard Deviation RT [ms]") +
  theme(legend.position = "none")
```

The results, displayed in Figure 7.6, show that the mean (or the standard deviation) varies across a wide range, with a substantial number of datasets having a mean (or the standard deviation) larger than 2000 ms. Again, this reveals a highly implausible assumption about the intercept parameter.

Moreover, we also plot the size of the effect of object relative minus subject relative sentence as a measure of effect size with the following code. We first compute the summary statistic, here the difference in reading times between subject versus object relative sentences (i.e., variable `so`). Then we define difference-values larger than 2000 ms as 2000 ms, and values smaller than -2000 ms as -2000 ms because these represent implausibly large/small values and make it easier to read the plot. We plot

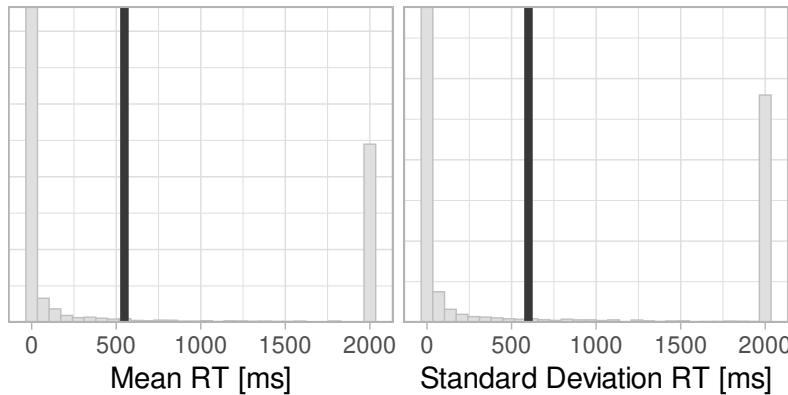


FIGURE 7.6: Prior predictive distribution of average reading times shows that extremely large reading times of more than 2000 ms are too frequently expected. Distribution of standard deviations of reading times shows that very large variances are over-expected in the priors. Values larger than 2000 are plotted at a value of 2000 for visualization.

these prior predictive data using `pp_check` by supplying the prior samples (`fit_prior_gibsonwu`) and the summary statistic (`effsize_2000`).

```
effsize_2000 <- function(x) {
  tmp <- mean(x[df_gibsonwu$so == +1]) -
    mean(x[df_gibsonwu$so == -1])
  tmp[tmp > +2000] <- +2000
  tmp[tmp < -2000] <- -2000
  tmp
}
pp_check(fit_prior_gibsonwu, type = "stat", stat = "effsize_2000") +
  labs(x = "Object - Subject [S-O RT]") +
  theme(legend.position = "none")
```

The results in Figure 7.7 show that our priors commonly assume differences in reading times between conditions of more than 2000 ms, which are larger than we would expect for a psycholinguistic manipulation of the kind investigated here. More specifically, given that we model reading times using a lognormal distribution, the expected effect size depends on the value for the intercept. For example, for an intercept of $\exp(1) = 2.7$

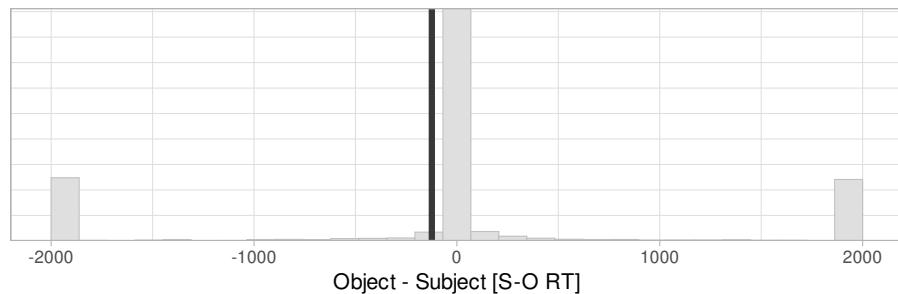


FIGURE 7.7: Prior predictive distribution of differences in reading times between object minus subject relatives shows that very large effect sizes are far too frequently expected. Values larger than 2000 (or smaller than -2000) are plotted at a value of 2000 (or -2000) for visualization.

ms and an effect size in log space of 1 (i.e., one standard deviation of the prior for the effect size), expected reading times for the two conditions are $\exp(1 - 1) = 1$ ms and $\exp(1 + 1) = 7$ ms. For an intercept of $\exp(10) = 22026$ ms, to the contrary, the corresponding reading times for the two conditions would be $\exp(10 - 1) = 8103$ ms and $\exp(10 + 1) = 59874$ ms.

This implies highly variable expectations for the effect size, including the possibility of very large effect sizes. If there is good reason to believe that the effect size is likely to be relatively small, priors with smaller expected effect sizes may be more reasonable. In chapter @[\(ch:priors\)](#) we discussed different methods for working out ballpark estimates of the range of variation in expected effect sizes.

It is also useful to look at individual-level differences in the effect of object versus subject relatives. Take a look at the subject with the largest (absolute) difference in reading times between object versus subject relatives in Figure 7.8.

Here, we first assign the prior simulated reading time to the data frame `df_gibsonwu` (terming it `rtfake`). Then we group the data frame by subject and by experimental condition (`so`), average prior predictive reading times for each subject and condition, and then compute the difference in reading times between subject versus object relative sentences for each subject (the subset where `so == 1` minus the subset where `so == -1`). Now,

we can take the absolute value of this difference (`abs(tmp$dif)`), and take the maximum or standard deviation across all subjects. Again, we set values larger than 2000 ms to a value of 2000 ms for better visibility.

```
effsize_max_2000 <- function(x) {
  df_gibsonwu$rtfake <- x
  tmp <- df_gibsonwu %>%
    group_by(subj, so) %>%
    summarize(rtfake = mean(rtfake)) %>%
    # calculates the difference between conditions:
    summarize(dif = rtfake[so == 1] - rtfake[so == -1])
  effect_size_max <- max(abs(tmp$dif), na.rm = TRUE)
  effect_size_max[effect_size_max > 2000] <- 2000
  effect_size_max
}

effsize_sd_2000 <- function(x) {
  df_gibsonwu$rtfake <- x
  tmp <- df_gibsonwu %>%
    group_by(subj, so) %>%
    summarize(rtfake = mean(rtfake)) %>%
    summarize(dif = rtfake[so == 1] - rtfake[so == -1])
  effect_size_SD <- sd(tmp$dif, na.rm = TRUE)
  effect_size_SD[effect_size_SD > 2000] <- 2000
  effect_size_SD
}

pp_check(fit_prior_gibsonwu,
            type = "stat",
            stat = "effsize_max_2000") +
  labs(x = "Max Effect Size [S-0 RT]") +
  theme(legend.position = "none")

pp_check(fit_prior_gibsonwu,
            type = "stat",
            stat = "effsize_sd_2000") +
  labs(x = "SD Effect Size [S-0 RT]") +
  theme(legend.position = "none")
```

The prior simulations in Figure 7.8 show common maximal effect sizes

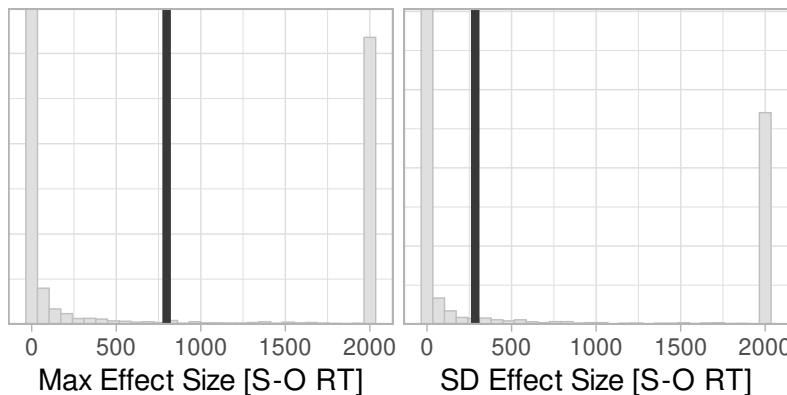


FIGURE 7.8: Maximal prior predicted effect size (object - subject relatives) across subjects again shows far too many extreme values and standard deviation of effect size (object - subject relatives) across subjects; again far too many extreme values are expected. Values > 2000 or < -2000 are plotted at a value of 2000 or -2000 for visualization.

of larger than 2000 ms, which is more than we would expect for observed data; similarly, the variance in hypothetical effect sizes is large, with many SDs larger than 2000 ms, and thus again takes many values that are inconsistent with our domain expertise about reading experiments.

7.3.2 Adjusting priors

Based on these analyses of prior predictive data, we can next use our domain expertise to refine our priors and adjust them to values for which we expect more plausible prior predictive hypothetical data as captured in the summary statistics.

First, we adapt the intercept; recall that in `@(ch:priors)` we made a first attempt at coming up with priors for the intercept; now we take our reasoning one step further. Given our prior knowledge about mean reading times (see the discussion in the previous chapter), we could choose a normal distribution in log-space with a mean of 6. This corresponds to an expected grand average reading time of $\exp(6) = 403$ ms. For the standard deviation, we use a value of $SD = 0.6$. For these prior values, we expect a strongly reduced mean reading time and a strongly reduced residual standard deviation in the simulated hypothetical data. Moreover, we expect

that implausibly small or large values for reading times will no longer be expected. For a visualization of the prior distribution of the intercept parameter in log-space and in ms-space see Figure 7.9a+b. Other values for the standard deviation that are close to 0.6, (e.g., SD = 0.5 or 0.7), may yield similar results. Our goal is not to specify a precise value, but rather to use prior parameter values that are qualitatively in line with our domain expertise about expected observed reading time data, and that do not produce highly implausible hypothetical data.

Next, for the effect of object minus subject relative sentences, we define a normally distributed prior with mean 0 and a much smaller standard deviation of 0.05. Again, we do not have precise information on the specific value for the standard deviation, but as we saw in chapter @*(ch:priors)*, we have some understanding of the range of variation in Chinese relatives. We expect a generally smaller effect size (see the meta-analysis in chapter @*(ch:priors)*), and we can check through prior predictive checks (data simulation and investigation of summary statistics) whether this yields a plausible pattern of expected results. Figures 7.9c+d show expected effects in log-scale and in ms-scale for a simple linear regression example.

In addition, we assume much smaller values for the standard deviations in how the intercept and the slope vary across subjects and across items of 0.1, and a smaller residual standard deviation of 0.5. Our expectation for the correlation between random effects is unchanged. In summary:

```
priors2 <- c(
  prior(normal(6, 0.6), class = Intercept),
  prior(normal(0, 0.05), class = b, coef = so),
  prior(normal(0, 0.1), class = sd),
  prior(normal(0, 0.5), class = sigma),
  prior(lkj(2), class = cor)
)
```

7.3.2.1 Prior predictive checks after increasing the informativity of the priors

We have now adjusted the priors increasing their informativity. This priors are still not too informative, but they are somewhat principled since they include some of the theory-neutral information that we have. Based

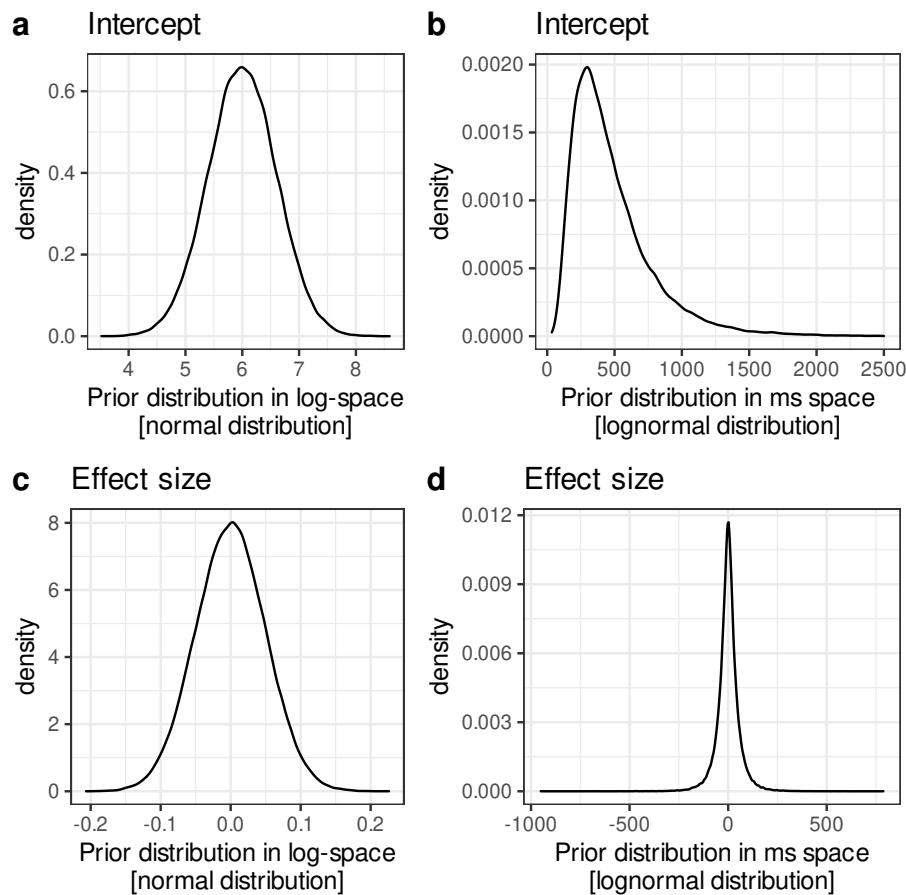


FIGURE 7.9: Prior distribution in log-space and in ms-space for a toy example of a linear regression. a) Displays the prior distribution of the intercept in log-space. b) Displays the prior distribution of the intercept in ms-space. c) Displays the prior distribution of the effect size in log-space. d) Displays the prior distribution of the effect size in ms-space.

on this new set of now weakly informative priors, we can again perform prior predictive checks as we did before. The new prior predictive checks are depicted in figure 7.10.

Figure 7.10a shows that now the distribution over histograms of the data looks much more reasonable, i.e., more like what we would expect for a histogram of observed data. Very small values for reading times are now rare, and not heavily inflated any more. Moreover, extremely large values for reading times larger than 2000 ms are rather unlikely.

We also take a look at the hypothetical average reading times (Figure 7.10b), and find that our expectations are now much more reasonable. We expect average reading times of around $\log(6) = 403$ ms. Most of the expected average reading times lie between 50 ms and 1500 ms, and only relatively few extreme values beyond these numbers are observed. The standard deviations of reading times are also in a much more reasonable range (see Figure 7.10d), with only very few values larger than the extreme value of 2000 ms.

As a next step, we look at the expected effect size (OR minus SR) in the hypothetical data (Figure 7.10c). Extreme values of larger or smaller than 2000 ms are now very rare, and most of the absolute values of expected effect sizes are smaller than 200 ms. More specifically, we also check the maximal effect size among all subjects (Figure 7.10e). Most of the distribution centers below a value of 1000 ms, reflecting a more plausible range of expected values. Likewise, the standard deviation of the psycholinguistically interesting effect size now rarely takes values larger than 500 ms (Figure 7.10f), reflecting more realistic a priori assumptions than in our initial (relatively) uninformative prior.

7.3.3 Computational faithfulness and model sensitivity

The next formal steps in the principled Bayesian workflow are to investigate computational faithfulness (using SBC) and model sensitivity. These allow the researcher to determine whether the posterior is estimated accurately for the given problem. Moreover, model sensitivity can be used to test whether parameter estimates are unbiased and whether anything can be learned by sampling data in the given design. Computational faithfulness (i.e., accurate posterior estimation) and model sensitivity need

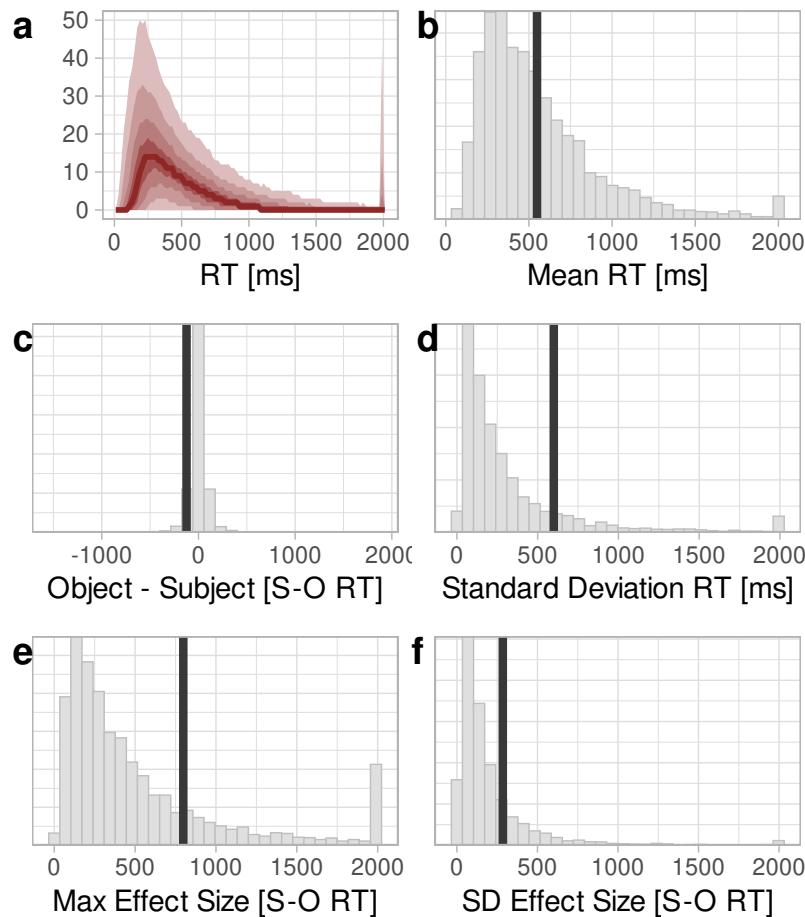


FIGURE 7.10: Prior predictive checks after adjusting the priors. The figures show prior predictive distributions. a) Histograms of reading times. Shaded areas correspond to 10-90 percent, 20-80 percent, 30-70 percent, and 40-60 percent quantiles across histograms; the solid line indicates the median across hypothetical datasets. This now provides a much more reasonable range of expectations. b)-f) Light blue/grey shows prior predictive distributions, dark blue/grey shows the data. b) Average log reading times now span a more reasonable range of values. c) Differences in reading times between object minus subject relatives; the values are now much more constrained without lots of extreme values. d) Standard deviations of reading times; in contrast to the relatively uninformative priors, values are in a reasonable range. e) Maximal effect size (object - subject relatives) across subjects; again, prior expectations are now much more reasonable compared to the uninformative prior. f) Standard deviation of effect size (object minus subject relative reading times) across subjects; this no longer shows a dominance of extreme values any more. a)-f) Values > 2000 or < -2000 are plotted at 2000 or -2000 for visualization.

to be checked for non-standard and more complex models, but for simpler/standard models may be performed only once for a research program where experimental designs and models are similar across studies. These steps are computationally very expensive, and can take a very long time to run for realistic datasets and models. For details on how to implement these steps, we refer the interested readers to Schad, Betancourt, and Vasishth (2019) and Betancourt (2018). We discuss a simplified version inspired by SBC for more advanced custom models implemented directly in Stan in chapter 19.

7.3.4 Posterior predictive checks: Model adequacy

Having examined the prior predictive data in detail, we can now take the observed data and perform posterior inference on it. We start by fitting a maximal *brms* model to the observed data.

```
fit_gibsonwu <- brm(rt ~ so + (1 + so | subj) + (1 + so | item),
  data = df_gibsonwu,
  family = lognormal(),
  prior = priors2
)
```

One could examine the posteriors from this model (we skip this step for brevity, but the reader should run the above code and examine the posterior summaries):

```
fit_gibsonwu
```

Figure 7.11 shows the posterior distribution for the slope parameter, which estimates the difference in reading times between object minus subject relative sentences.

```
mcmc_hist(fit_gibsonwu, pars = c("b_so")) +
  labs(x = "Object - subject relatives",
       title = "Posterior distribution")
```

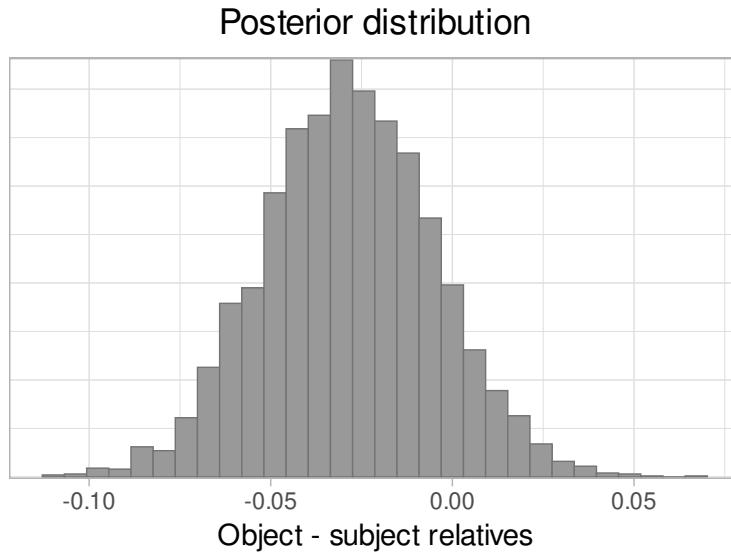


FIGURE 7.11: Posterior distribution for the slope parameter, estimating the difference in reading times between object relative minus subject relative sentences.

```
postgw <- as_draws_df(fit_gibsonwu)
mean(postgw$b_so < 0)
```

```
## [1] 0.892
```

Figure 7.11 shows that the reading times in object relative sentences tends to be slightly faster than in subject relative sentences ($p(b < 0) = 0.89$); this is as predicted by Gibson and Wu (2013). However, given the wide 95% confidence intervals; it is difficult to rule out the possibility that there is effectively no difference in reading time between the two conditions without doing model comparison (with Bayes factor or cross-validation).

To assess model adequacy, we perform posterior predictive checks. We simulate data based on posterior samples of parameters. This then allows us to investigate the simulated data by computing the summary statistics that we used in the prior predictive checks, and by comparing model predictions with the observed data.

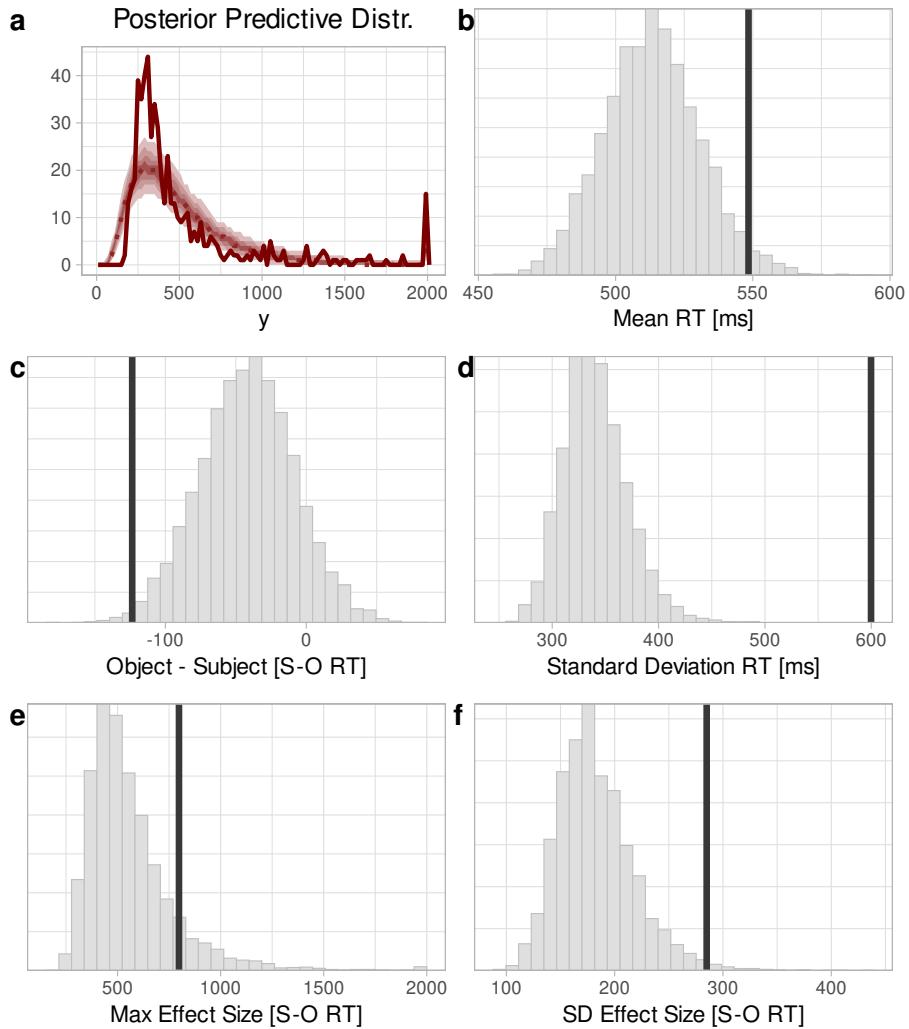


FIGURE 7.12: Posterior predictive checks for weakly informative priors. Distributions are over posterior predictive simulated data. a) Histograms of reading times. 10-90 percent, 20-80 percent, 30-70 percent, and 40-60 percent quantiles across histograms are shown as shaded areas; the median is shown as a dotted line and the observed data as a solid line. For illustration, values > 2000 are plotted as 2000; modeling was done on the original data. b)-f) Light blue/grey shows the posterior predictive distributions, dark blue/grey shows the data. b) Average reading times. c) Differences in reading times between object minus subject relatives. d) Standard deviations of reading times. e) Maximal effect size (object - subject relatives) across subjects. f) Standard deviation of effect size across subjects.

The results from these analyses show that the lognormal distribution (see Figure 7.12a) provides an approximation to the distribution of the data. However, although the fit looks reasonable, there is still systematic deviation from the data of the model's predictions. This deviation suggests that maybe a constant offset is needed in addition to the lognormal distribution. This can be implemented in `brms` by replacing the family specification `family = lognormal()` with the shifted version `family = shifted_lognormal()`, and motivates another round of model validation.

Next, for the other summary statistics, we first look at the distribution of means. The posterior predictive means capture the mean reading time in the observed data (i.e., the vertical line in Figure 7.12b); the data is not perfectly captured, but still in the distribution of the model. For the standard deviation we can see that the model posterior assumes too little posterior variation and that the model thus does not capture the standard deviation of the data well (Figure 7.12d). Figure 7.12c shows the effect size of object minus subject relative sentences predicted by model (histogram) and observed in the data (vertical line). Here, posterior model predictions for the effect are in line with the empirical data. However, again, the model is showing mostly smaller effect sizes than the data do. For the biggest effect among all subjects (Figure 7.12e) the model captures the data reasonably well. For the standard deviation of the effect across subjects (Figure 7.12f), the variation in the model is again a bit too small considering the variation in the data. Potentially, the lack of agreement between data and posterior predictive distributions might be due to the mismatch between the true distribution of the data and the lognormal likelihood that we assumed. This could be checked by running the model again and using a shifted lognormal instead of a lognormal likelihood.

7.4 Summary

In this chapter, we have introduced key questions to ask about a model and the inference process as discussed by Betancourt (2018) and by Schad, Betancourt, and Vasishth (2019), and have applied this to a dataset from an experiment involving a typical repeated measures experimental design used in cognitive psychology and psycholinguistics. Prior predictive

checks using analyses of simulated prior data suggest that, compared to previous applications in reading experiments (e.g., Nicenboim and Vasishth 2018), far more informative priors can and should be used. We demonstrated that including such additional domain knowledge into the priors leads to more plausible expected data. Moreover, incorporating more informative priors should also speed up the HMC sampling process. These more informative priors, however, may not alter posterior inferences much for the present design. Posterior predictive checks showed weak support for our statistical model, as the model only partially successfully recovered the tested summary statistics, suggesting that slightly more variable priors might be adequate. For inference on whether reading times differ between Chinese object versus subject relative sentences, a Bayes factor analysis would be needed to compare a null model assuming no effect to an alternative model assuming a difference between object versus subject relative sentences. As an aside, the published result in Gibson and Wu (2013) showed a statistically significant effect (using repeated measures ANOVA) and concluded that the effect was present.

In summary, this analysis provides an example and tutorial for using a principled Bayesian workflow (Betancourt 2018; Schad, Betancourt, and Vasishth 2019) in cognitive science experiments. The workflow reveals useful information about which (weakly informative) priors to use, and performs checks of the used inference procedures and the statistical model. The workflow provides a robust foundation for using a statistical model to answer scientific questions, and will be useful for researchers developing analysis plans as part of pre-registrations, registered reports, or simply as preparatory design analyses prior to conducting an experiment.

7.5 Further reading

Some important articles relating to developing a principled Bayesian workflow are by Betancourt (2018), Gabry et al. (2017), Gelman et al. (2020), and Talts et al. (2018). The `stantargets` R package provides tools for a systematic, efficient, and reproducible workflow (Landau 2021). Also recommended is the article on reproducible workflows by Wilson et al. (2017).

8

Contrast coding

Whenever one uses a categorical factor as a predictor in a Bayesian linear (mixed) model, for example when estimating the difference in a dependent variable between two or three experimental conditions, then it is necessary to code the discrete factor levels into numeric predictor variables. This coding is termed *contrast coding*. For example, in the previous chapter (section 5.2), we coded two experimental conditions as -1 and $+1$, i.e., implementing a sum contrast. Those *contrasts* are the values that we assign to predictor variables to encode specific comparisons between factor levels and to create predictor terms to estimate these comparisons in any type of regression, including Bayesian regressions.

Contrast coding in Bayesian models works more or less the same way as in frequentist models, and the same principles and tools can be used in both cases. This chapter will introduce contrast coding using Bayesian models. The descriptions are in large parts taken from Schad et al. (2020) (which is published under a CC-BY 4.0 license) and adapted for the current Bayesian context.

Consider a situation where we want to estimate differences in a dependent variable between three factor levels. An example could be differences in response times between three levels of word class (noun, verb, adjective). We might be interested in whether word class influences response times. In frequentist statistics, one way to approach this question would be to run an ANOVA and compute an omnibus F-test for whether word class explains response times. A Bayesian equivalent to the frequentist omnibus F-test is Bayesian model comparison (i.e., Bayes factors), where we might compare an alternative model including word class as a predictor term with a null model lacking this predictor. We will discuss such Bayesian model comparison using Bayes factors in a later chapter. However, if based on such omnibus approaches we find support for an influ-

ence of word class on response times, it remains unclear where this effect actually comes from, i.e., whether it originated from the nouns, verbs, or adjectives. However, scientists typically have a priori expectations about which groups differ from each other. In this chapter, we will show how to estimate specific comparisons directly in a Bayesian linear model. This gives the researcher a lot of control over Bayesian analyses. Specifically, we show how planned comparisons between specific conditions (groups) or clusters of conditions, are implemented as contrasts. This is a very effective way to align expectations with the statistical model. In Bayesian models, any specific comparisons can also be computed after the model is fit. However, coding a priori expectations into contrasts for model fitting will make it much more straightforward to estimate certain comparisons between experimental conditions, and to perform Bayesian model comparisons using Bayes factors to provide evidence for or against very specific hypotheses.

For this and the next chapter, although knowledge of the matrix formulation of the linear model is not necessary, for a deeper understanding of contrast coding some exposure to the matrix formulation is desirable. We discuss the matrix formulation in Box 5.3 and in the frequentist textbook (Vasishth, Schad, et al. 2021b).

8.1 Basic concepts illustrated using a two-level factor

We first consider the simplest case: suppose we want to compare the means of a dependent variable (DV) such as response times between two groups of subjects. R can be used to simulate data for such an example. Such simulated data is available in the R package `bcogsci` as the dataset `df_contrasts1`. The simulations assumed longer response times in condition F1 ($\mu_1 = 0.8$ sec) than F2 ($\mu_2 = 0.4$ sec). The data from the 10 simulated subjects are aggregated and summary statistics are computed for the two groups.

```
data("df_contrasts1")
df_contrasts1
```

```
## # A tibble: 10 x 3
#>   F          DV    id
#>   <fct>     <dbl> <int>
#> 1 F1        0.636     1
#> 2 F1        0.841     2
#> 3 F1        0.555     3
#> # ... with 7 more rows
```

```
str(df_contrasts1)
```

The results, displayed in Figure 8.1 and shown in Table 8.1, show that the assumed true condition means are exactly realized with the simulated data. The numbers are exact because the used `mvrnorm()` function (see `?df_contrasts1`) ensures that the data are generated so that the sample mean yields the true means for each level. In real datasets, of course, the sample means will vary from experiment to experiment.

A simple Bayesian linear model of DV on F using the function `brm` yields a straightforward estimate of the difference between the group means. We use relatively uninformative priors. The estimates for the fixed effects are presented below:

TABLE 8.1: Summary statistics per condition for the simulated data.

Factor	N data	Est. means	Std. dev.	Std. errors
F1	5	0.8	0.2	0.1
F2	5	0.4	0.2	0.1

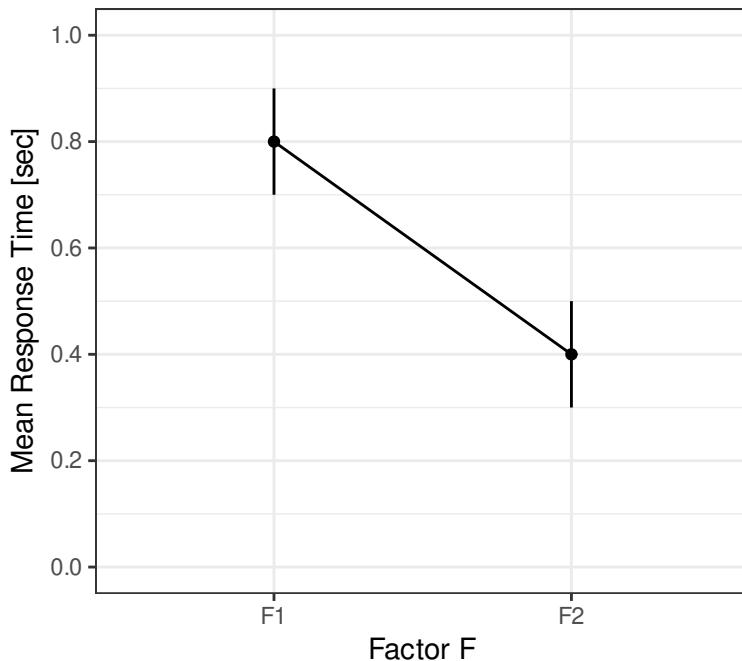


FIGURE 8.1: Means and standard errors of the simulated dependent variable (e.g., response times in seconds) in two conditions F1 and F2.

```
fit_F <- brm(DV ~ 1 + F,
  data = df_contrasts1,
  family = gaussian(),
  prior = c(
    prior(normal(0, 2), class = Intercept),
    prior(normal(0, 2), class = sigma),
    prior(normal(0, 1), class = b)
  )
)
```

```
fixef(fit_F)
```

	Estimate	Est.Error	Q2.5	Q97.5
## Intercept	0.80	0.11	0.57	1.01

```
## FF2      -0.39      0.16 -0.72 -0.09
```

Comparing the means for each condition with the coefficients (*Estimates*) reveals that (i) the intercept (0.8) is the mean for condition F1, $\hat{\mu}_1$; and (ii) the slope (FF2: -0.4) is the difference between the estimated means for the two groups, $\hat{\mu}_2 - \hat{\mu}_1$ (Bolker 2018):

$$\begin{aligned} \text{Intercept} &= \hat{\mu}_1 &= \text{estimated mean for F1} \\ \text{Slope (FF2)} &= \hat{\mu}_2 - \hat{\mu}_1 &= \text{estim. mean for F2} - \text{estim. mean for F1} \end{aligned} \quad (8.1)$$

The new information are the credible intervals for the difference between the two groups.

8.1.1 Default contrast coding: Treatment contrasts

How does the function `brm` arrive at these particular values for the intercept and slope? That is, why does the intercept assess the mean of condition F1 and how do we know the slope measures the difference in means between F2 – F1? This result is a consequence of the default contrast coding of the factor F. R assigns treatment contrasts to factors and orders their levels alphabetically. The first factor level (here: F1) is coded as 0 and the second level (here: F2) is coded as 1. This becomes clear when we inspect the current contrast attribute of the factor using the `contrasts` command:

```
contrasts(df_contrasts1$F)
```

```
##      F2
## F1  0
## F2  1
```

Why does this contrast coding yield these particular regression coefficients? Let's take a look at the regression equation. Let α represent the intercept, and β_1 the slope. Then, the simple regression above expresses the belief that the expected response time y is a linear function of the factor F. In a more general formulation, this is written as follows: y is a linear function of some predictor x with regression coefficients for the intercept, α , and for the factor, β_1 :

$$y = \alpha + \beta_1 x \quad (8.2)$$

This equation is part of the likelihood in a Bayesian model. So, if $x = 0$ (condition F1), y is $\alpha + \beta_1 \cdot 0 = \alpha$; and if $x = 1$ (condition F2), y is $\alpha + \beta_1 \cdot 1 = \alpha + \beta_1$.

Expressing the above in terms of the estimated coefficients:

$$\begin{aligned} \text{estim. value for F1} &= \hat{\mu}_1 = \hat{\alpha} = \text{Intercept} \\ \text{estim. value for F2} &= \hat{\mu}_2 = \hat{\alpha} + \hat{\beta}_1 = \text{Intercept} + \text{Slope (FF2)} \end{aligned} \quad (8.3)$$

It is useful to think of such unstandardized regression coefficients as difference scores; they express the increase in the dependent variable y associated with a change in the independent variable x of 1 unit, such as going from 0 to 1 in this example. The difference between condition means is $0.4 - 0.8 = -0.4$, which is the estimated regression coefficient $\hat{\beta}_1$. The sign of the slope is negative because we have chosen to subtract the larger mean F1 score from the smaller mean F2 score.

8.1.2 Defining comparisons

The analysis of the regression equation demonstrates that in the treatment contrast the intercept assesses the average response in the baseline condition, whereas the slope estimates the difference between condition means. However, these are just verbal descriptions of what each coefficient assesses. Is it also possible to formally write down what each coefficient assesses?

From the perspective of parameter estimation, the slope represents the effect of main interest, so we consider this first. The treatment contrast specifies that the slope β_1 estimates the difference in means between the two levels of the factor F. This can formally be written as:

$$\beta_1 = \mu_{F2} - \mu_{F1} \quad (8.4)$$

or equivalently:

$$\beta_1 = -1 \cdot \mu_{F1} + 1 \cdot \mu_{F2} \quad (8.5)$$

The ± 1 weights in the parameter estimation directly express which means are compared by the treatment contrast.

The intercept in the treatment contrast estimates a quantity that is usually of little interest: it estimates the mean in condition F1. Formally, the parameter α estimates the following quantity:

$$\alpha = \mu_{F1} \quad (8.6)$$

or equivalently:

$$\alpha = 1 \cdot \mu_{F1} + 0 \cdot \mu_{F2}. \quad (8.7)$$

The fact that the intercept term formally estimates the mean of condition F1 is in line with our previous derivation (see equation 8.1).

In R, factor levels are ordered alphabetically and by default the first level is used as the baseline in treatment contrasts. Obviously, this default mapping will only be correct for a given dataset if the levels' alphabetical ordering matches the desired contrast coding. When it does not, it is possible to re-order the levels. Here is one way of re-ordering the levels in R:

```
df_contrasts1$Fb <- factor(df_contrasts1$F,
  levels = c("F2", "F1"))
contrasts(df_contrasts1$Fb)
```

```
##     F1
## F2  0
## F1  1
```

This re-ordering did not change any data associated with the factor, only one of its attributes. With this new contrast attribute a simple Bayesian model yields the following result.

```
fit_Fb <- brm(DV ~ 1 + Fb,
  data = df_contrasts1,
  family = gaussian(),
```

```

prior = c(
  prior(normal(0, 2), class = Intercept),
  prior(normal(0, 2), class = sigma),
  prior(normal(0, 1), class = b)
)
)

fixef(fit_Fb)

##           Estimate Est.Error Q2.5 Q97.5
## Intercept     0.40      0.11 0.17  0.63
## FbF1         0.39      0.15 0.08  0.70

```

The model now estimates different quantities. The intercept now codes the mean of condition F_2 , and the slope measures the difference in means between F_1 minus F_2 . This represents an alternative coding of the treatment contrast.

Importantly, these model posteriors do not provide evidence for the hypothesis that the effect of factor F is different from zero. If the research focus is on such hypothesis testing, Bayesian hypothesis tests can be carried out using Bayes factors, by comparing a model containing a contrast of interest with a model lacking this contrast. We will discuss details of Bayesian hypothesis testing based on Bayes factors in chapter @[\(ch:bf\)](#).

8.1.3 Sum contrasts

Treatment contrasts are only one of many options. It is also possible to use sum contrasts, which code one of the conditions as -1 and the other as $+1$, effectively *centering* the effects at the grand mean (GM, i.e., the mean of the two group means). Here, we rescale the contrast to values of -0.5 and $+0.5$, which makes the estimated treatment effect the same as for treatment coding and easier to interpret.

To use this contrast in a linear regression, use the `contrasts` function:

```
contrasts(df_contrasts1$F) <- c(-0.5, +0.5)
fit_mSum <- brm(DV ~ 1 + F,
  data = df_contrasts1,
  family = gaussian(),
  prior = c(
    prior(normal(0, 2), class = Intercept),
    prior(normal(0, 2), class = sigma),
    prior(normal(0, 1), class = b)
  )
)
```

```
fixef(fit_mSum)
```

	Estimate	Est.Error	Q2.5	Q97.5
## Intercept	0.60	0.08	0.45	0.76
## F1	-0.39	0.16	-0.71	-0.06

Here, the slope ($F1$) again codes the difference of the groups associated with the first and second factor levels. It has the same value as in the treatment contrast. However, the intercept now represents the estimate of the average of condition means for $F1$ and $F2$, that is, the GM. This differs from the treatment contrast. For the scaled sum contrast:

$$\begin{aligned} \text{Intercept} &= (\hat{\mu}_1 + \hat{\mu}_2)/2 &= \text{estimated mean of } F1 \text{ and } F2 \\ \text{Slope } (F1) &= \hat{\mu}_2 - \hat{\mu}_1 &= \text{estim. mean for } F2 - \text{estim. mean for } F1 \end{aligned} \quad (8.8)$$

Why does the intercept assess the GM and why does the slope estimate the group difference? This is the result of rescaling the sum contrast. The first factor level ($F1$) was coded as -0.5 , and the second factor level ($F1$) as $+0.5$:

```
contrasts(df_contrasts1$F)
```

```
##   [,1]
## F1 -0.5
## F2  0.5
```

Let's again look at the regression equation to better understand what computations are performed. Again, α represents the intercept, β_1 represents the slope, and the predictor variable x represents the factor F. The regression equation is written as:

$$y = \alpha + \beta_1 x \quad (8.9)$$

The group of F1 subjects is then coded as -0.5 , and the response time for the group of F1 subjects is $\alpha + \beta_1 \cdot x_1 = 0.6 + (-0.4) \cdot (-0.5) = 0.8$. By contrast, the F2 group is coded as $+0.5$. By implication, the mean of the F2 group must be $\alpha + \beta_1 \cdot x_1 = 0.6 + (-0.4) \cdot 0.5 = 0.4$. Expressed in terms of the estimated coefficients:

$$\begin{aligned} \text{estim. value for F1} &= \hat{\mu}_1 = \hat{\alpha} - 0.5 \cdot \hat{\beta}_1 = \text{Intercept} - 0.5 \cdot \text{Slope (F1)} \\ \text{estim. value for F2} &= \hat{\mu}_2 = \hat{\alpha} + 0.5 \cdot \hat{\beta}_1 = \text{Intercept} + 0.5 \cdot \text{Slope (F1)} \end{aligned} \quad (8.10)$$

The unstandardized regression coefficient is a difference score: Taking a step of one unit on the predictor variable x , e.g., from -0.5 to $+0.5$, reflecting a step from condition F1 to F2, changes the dependent variable from 0.8 (for condition F1) to 0.4 (condition F2). This reflects a difference of $0.4 - 0.8 = -0.4$; this is again the estimated regression coefficient $\hat{\beta}_1$. Moreover, as mentioned above, the intercept now assesses the GM of conditions F1 and F2: it is in the middle between condition means for F1 and F2.

So far we gave verbal statements about what is estimated by the intercept and the slope in the case of the scaled sum contrast. It is possible to write these statements as formal parameter estimates. In sum contrasts, the slope parameter β_1 assesses the following quantity:

$$\beta_1 = -1 \cdot \mu_{F1} + 1 \cdot \mu_{F2} \quad (8.11)$$

This estimates the same quantity as the slope in the treatment contrast. The intercept, however, now assesses a different quantity: it estimates the average of the two conditions F1 and F2:

$$\alpha = 1/2 \cdot \mu_{F1} + 1/2 \cdot \mu_{F2} = \frac{\mu_{F1} + \mu_{F2}}{2} \quad (8.12)$$

In balanced data, i.e., in datasets where there are no missing data points, the average of the two conditions F1 and F2 is the GM. In unbalanced datasets, where there are missing values, this average is the weighted GM. To illustrate this point, consider an example with fully balanced data and two equal group sizes of 5 subjects for each group F1 and F2. Here, the GM is also the mean across all subjects. Next, consider a highly simplified unbalanced dataset, where in condition F1 two observations of the dependent variable are available with values of 2 and 3, and where in condition F2 only one observation of the dependent variable is available with a value of 4. In this dataset, the mean across all subjects is $\frac{2+3+4}{3} = \frac{9}{3} = 3$. However, the (weighted) GM as assessed in the intercept in a model using sum contrasts for factor F would first compute the mean for each group separately (i.e., $\frac{2+3}{2} = 2.5$, and 4), and then compute the mean across conditions $\frac{2.5+4}{2} = \frac{6.5}{2} = 3.25$. The GM of 3.25 is different from the mean across subjects of 3.

To summarize, treatment contrasts and sum contrasts are two possible ways to parameterize the difference between two groups; they generally estimate different quantities. Treatment contrasts compare one or more means against a baseline condition, whereas sum contrasts compare a condition's mean to the GM (which in the two-group case also implies estimating the difference between the two group means). One question that comes up here is: how does one know or how can one formally derive what quantities are estimated by a given set of contrasts? (In the context of Bayes factors, the question would be: what hypothesis test does the contrast coding encode?) This question will be discussed in detail below for the general case of any arbitrary contrast coding.

8.1.4 Cell means parameterization and posterior comparisons

One alternative option is to use what is called the cell means parameterization. In this approach, one does not estimate an intercept term, and then differences between factor levels. Instead, each free parameter is used to simply estimate the mean of one of the factor levels. As a consequence, no comparisons between condition means are estimated, but simply the mean of each experimental condition is estimated. Cell means parameterization is specified by explicitly removing the intercept term (which is added automatically in brms) by adding a `-1` in the regression formula:

```
fit_mCM <- brm(DV ~ -1 + F,
  data = df_contrasts1,
  family = gaussian(),
  prior = c(
    prior(normal(0, 2), class = sigma),
    prior(normal(0, 2), class = b)
  )
)
```

```
fixef(fit_mCM)
```

```
##      Estimate Std. Error Q2.5 Q97.5
## FF1       0.8       0.11  0.57  1.02
## FF2       0.4       0.11  0.18  0.61
```

Now, the regression coefficients (see the column labeled `Estimate`) estimate the mean of the first factor level (0.8) and the mean of the second factor level (0.4). This cell means parameterization usually does not allow us to make inferences about the hypotheses of interest using Bayes factors, as these hypotheses usually relate to differences between conditions rather than to whether each condition differs from zero. However, the cell means parameterization provides a good example demonstrating an advantage of Bayesian data analysis: In Bayesian models, it is possible to use the posterior samples to compute new estimates that were not directly contained in the fitted model. To implement this, we first extract the posterior samples from the `brm` model object:

```
df_postSamp <- as_draws_df(fit_mCM)
str(df_postSamp)
```

```
## #> #> draws_df [4,000 x 7] (S3: draws_df/draws/tbl_df/tbl/data.frame)
## #> $ b_FF1      : num [1:4000] 0.882 0.764 1.03 0.611 0.759 ...
## #> $ b_FF2      : num [1:4000] 0.279 0.532 0.461 0.409 0.391 ...
## #> $ sigma       : num [1:4000] 0.275 0.355 0.213 0.207 0.197 ...
## #> $ lp__        : num [1:4000] -4.64 -5.78 -6.23 -5.05 -3.04 ...
## #> $ .chain      : int [1:4000] 1 1 1 1 1 1 1 1 1 ...
```

```
## $ .iteration: int [1:4000] 1 2 3 4 5 6 7 8 9 10 ...
## $ .draw      : int [1:4000] 1 2 3 4 5 6 7 8 9 10 ...
```

In a second step, we can then compute comparisons from these posterior samples. For example, we can compute the difference between conditions F2 and F1. To do so, we simply take the posterior samples for each condition, and compute their difference.

```
df_postSamp$b_dif <- df_postSamp$b_FF2 - df_postSamp$b_FF1
```

This provides a posterior sample of the difference between conditions. It is possible to investigate this posterior sample by looking at its mean and 95% credibility intervals:

```
c(
  Estimate = mean(df_postSamp$b_dif),
  quantile(df_postSamp$b_dif, p = c(0.025, 0.975))
)
```

```
## Estimate      2.5%     97.5%
## -0.3992   -0.7115   -0.0971
```

Interestingly, this provides the same estimate of roughly -0.4 as we obtained previously when using the treatment contrast or the scaled sum contrasts in our Bayesian (brms) models. Thus, Bayesian models provide a lot of flexibility in computing new comparisons post-hoc from the posterior samples and in obtaining their posterior distributions. However, what these posterior computations do not provide directly are inferences on null hypotheses, i.e., they do not allow us to make inference on whether a given contrast is best explained by a null model assuming no difference, or by an alternative model assuming a difference between conditions.

8.2 The hypothesis matrix illustrated with a three-level factor

Consider an example with the three word classes nouns, verbs, and adjectives. We load simulated data from a lexical decision task with response

times as dependent variable. The research question is: do response times differ as a function of the between-subject factor word class with three levels: nouns, verbs, and adjectives? Here, just to illustrate the case of a three-level factor, we make the arbitrary assumption that nouns have longest response times and that adjectives the shortest response times. Word class is specified as a between-subject factor. In cognitive science experiments, word class will usually vary within subjects and between items. However, the within- or between-subjects status of an effect is independent of its contrast coding; we assume the manipulation to be between subjects for ease of exposition. The concepts presented here extend to repeated measures designs that are often analyzed using hierarchical Bayesian (linear mixed) models.

The following R code loads and displays the simulated data.

```
data("df_contrasts2")
head(df_contrasts2)

## # A tibble: 6 x 3
##   F       DV    id
##   <fct> <int> <int>
## 1 nouns    476     1
## 2 nouns    517     2
## 3 nouns    491     3
## # ... with 3 more rows
```

As shown in Table 8.2, the estimated means reflect our assumptions about the true means in the data simulation: Response times are longest for nouns and shortest for adjectives. In the following sections, we use this

TABLE 8.2: Summary statistics per condition for the simulated data.

Factor	N data	Est. means	Std. dev.	Std. errors
adjectives	4	400.2	19.9	9.9
nouns	4	500.0	20.0	10.0
verbs	4	450.2	20.0	10.0

dataset to illustrate **sum** contrasts. Furthermore, we will use an additional dataset to illustrate **repeated**, **Helmert**, **polynomial**, and **custom** contrasts. In practice, usually only one set of contrasts is selected when the expected pattern of means is formulated during the design of the experiment.

8.2.1 Sum contrasts

We begin with sum contrasts. Suppose that the expectation is that nouns are responded to slower and adjectives are responded to faster than the GM response time. Then, the research question could be: How much do nouns differ from the GM and how much do adjectives differ from the GM? And are the responses slower or faster than the GM? We want to estimate the following two quantities:

$$\beta_1 = \mu_1 - \frac{\mu_1 + \mu_2 + \mu_3}{3} = \mu_1 - GM \quad (8.13)$$

and

$$\beta_2 = \mu_2 - \frac{\mu_1 + \mu_2 + \mu_3}{3} = \mu_2 - GM \quad (8.14)$$

β_1 can also be written as:

$$\beta_1 = \mu_1 - \frac{\mu_1 + \mu_2 + \mu_3}{3} \quad (8.15)$$

$$\Leftrightarrow \beta_1 = \frac{2}{3}\mu_1 - \frac{1}{3}\mu_2 - \frac{1}{3}\mu_3 \quad (8.16)$$

Here, the weights $2/3, -1/3, -1/3$ are informative about how to combine the condition means to estimate the linear model coefficient.

β_2 is also rewritten as:

$$\beta_2 = \mu_2 - \frac{\mu_1 + \mu_2 + \mu_3}{3} \quad (8.17)$$

$$\Leftrightarrow \beta_2 = -\frac{1}{3}\mu_1 + \frac{2}{3}\mu_2 - \frac{1}{3}\mu_3 \quad (8.18)$$

Here, the weights are $-1/3, 2/3, -1/3$, and they again indicate how to combine the condition means for estimating the regression coefficient.

8.2.2 The hypothesis matrix

The weights of the condition means are not only useful to define parameter estimates. They also provide the starting step in a very powerful method which allows the researcher to generate the contrasts that are needed to estimate these comparisons in a linear model. That is, what we did so far is to explain some kinds of different contrast codings that exist and what comparisons are estimated by these contrasts. That is, if a certain dataset is given and the goal is to estimate certain comparisons, then the procedure would be to check whether any of the contrasts that we encountered above happen to estimate these comparisons of interest. Sometimes it suffices to use one of these existing contrasts. However, at other times, our research questions may not correspond exactly to any of the contrasts in the default set of standard contrasts provided in R. For these cases, or simply for more complex designs, it is very useful to know how contrast matrices are created. Indeed, a relatively simple procedure exists in which we write our comparisons formally, extract the weights of the condition means from the comparisons, and then automatically generate the correct contrast matrix that we need in order to estimate these comparisons in a linear model. Using this powerful method, it is not necessary to find a match to a contrast matrix provided by the family of functions in R starting with the prefix `contr`. Instead, it is possible to simply define the comparisons that one wants to estimate, and to obtain the correct contrast matrix for these in an automatic procedure. Here, for pedagogical reasons, we show some examples of how to apply this procedure in cases where the comparisons *do* correspond to some of the existing contrasts.

Defining a custom contrast matrix involves four steps:

1. Write down the estimated comparisons
2. Extract the weights and write them into what we will call a *hypothesis matrix* (and can also be viewed as a *comparison matrix*)
3. Apply the *generalized matrix inverse* to the hypothesis matrix to create the contrast matrix

4. Assign the contrast matrix to the factor and run the (Bayesian) model

Let us apply this four-step procedure to our example of the sum contrast. The first step, writing down the estimated comparisons, is shown above. The second step involves writing down the weights that each comparison gives to condition means. The weights for the first comparison are $wH01=c(+2/3, -1/3, -1/3)$, and the weights for the second comparison are $wH02=c(-1/3, +2/3, -1/3)$.

Before writing these into a hypothesis matrix, we also define the estimated quantity for the intercept term. The intercept parameter estimates the mean across all conditions:

$$\alpha = \frac{\mu_1 + \mu_2 + \mu_3}{3} \quad (8.19)$$

$$\alpha = \frac{1}{3}\mu_1 + \frac{1}{3}\mu_2 + \frac{1}{3}\mu_3 \quad (8.20)$$

This estimate has weights of 1/3 for all condition means. The weights from all three model parameters that were defined are now combined and written into a matrix that we refer to as the *hypothesis matrix* (Hc):

```
HcSum <- rbind(
  cH00 = c(adjectives = 1 / 3, nouns = 1 / 3, verbs = 1 / 3),
  cH01 = c(adjectives = +2 / 3, nouns = -1 / 3, verbs = -1 / 3),
  cH02 = c(adjectives = -1 / 3, nouns = +2 / 3, verbs = -1 / 3)
)
fractions(t(HcSum))
```

```
##           cH00  cH01  cH02
## adjectives  1/3   2/3  -1/3
## nouns       1/3  -1/3   2/3
## verbs       1/3  -1/3  -1/3
```

Each set of weights is first entered as a row into the matrix (command `rbind()`¹). However, we then switch rows and columns of the matrix for

¹The reason for this is that mathematically, individual comparisons in the hypothesis matrix are coded as rows rather than as columns (see Schad et al. 2020).

easier readability using the command `t()` (this transposes the matrix, i.e., switches rows and columns). The command `fractions()` from `MASS` library turns the decimals into fractions to improve readability.

Now that the condition weights have been written into the hypothesis matrix, the third step of the procedure is implemented: a matrix operation called the *generalized matrix inverse*² is used to obtain the contrast matrix that is needed to estimate these comparisons in a linear model. In R this next step is done using the function `ginv()` from the `MASS` package. Define a function `ginv2()` for nicer formatting of the output.³

```
ginv2 <- function(x) { # define a function to make the output nicer
  MASS:::fractions(provideDimnames(MASS:::ginv(x),
    base = dimnames(x)[2:1]
  ))
}
```

Applying the generalized inverse to the hypothesis matrix results in the new matrix `XcSum`. This is the contrast matrix X_c that estimates exactly those comparisons that were specified earlier:

```
(XcSum <- ginv2(HcSum))
```

```
##          cH00  cH01  cH02
## adjectives  1     1     0
## nouns       1     0     1
## verbs        1    -1    -1
```

This contrast matrix corresponds exactly to the sum contrasts described above. In the case of the sum contrast, the contrast matrix looks very different from the hypothesis matrix. The contrast matrix in sum contrasts codes with +1 the condition that is to be compared to the GM. The condi-

²At this point, there is no need to understand in detail what this means. We refer the interested reader to Schad et al. (2020). For a quick overview, we recommend a vignette explaining the generalized inverse in the `matlib` package (<https://cran.r-project.org/web/packages/matlib/vignettes/ginv.html>) (Friendly, Fox, and Chalmers 2020).

³The function `fractions()` from the `MASS` package is used to make the output more easily readable, and the function `provideDimnames()` is used to keep row and column names.

tion that is never compared to the GM is coded as -1 . Without knowing the relationship between the hypothesis matrix and the contrast matrix, the meaning of the coefficients is completely opaque.

To verify this custom-made contrast matrix, it is compared to the sum contrast matrix as generated by the R function `contr.sum()` in the `stats` package. The resulting contrast matrix is identical to the result when adding the intercept term, a column of ones, to the contrast matrix:

```
fractions(cbind(1, contr.sum(3)))
```

```
##   [,1] [,2] [,3]
## 1   1    1    0
## 2   1    0    1
## 3   1   -1   -1
```

In order to estimate model parameters, step four in our procedure involves assigning sum contrasts to the factor `F` in our example data, and running a (Bayesian) linear model. This allows use to estimate the regression coefficients associated with each contrast. We compare these to the data shown above (Table 8.2) to test whether the regression coefficients actually correspond to the differences of condition means, as intended. To define the contrast, it is necessary to remove the intercept term, as this is automatically added by the modeling function `brm()`.

```
contrasts(df_contrasts2$F) <- XcSum[, 2:3]
fit_Sum <- brm(DV ~ 1 + F,
  data = df_contrasts2,
  family = gaussian(),
  prior = c(
    prior(normal(500, 100), class = Intercept),
    prior(normal(0, 100), class = sigma),
    prior(normal(0, 100), class = b)
  )
<>)
```

```
fixef(fit_Sum)

##           Estimate Est.Error Q2.5 Q97.5
## Intercept    450.2     6.83 436.6 463.9
## FcH01       -49.2     9.71 -68.0 -29.1
## FcH02        49.1     9.61  29.9  67.2
```

The (Bayesian) linear model regression coefficients show the GM response time of 450 ms in the intercept. Remember that the first regression coefficient `FcH01` was designed to estimate the extent to which adjectives are responded to faster than the GM. The regression coefficient `FcH01` (`Estimate`) of -50 reflects the difference between adjectives (400 ms) and the GM of 450 ms. The second estimate of interest tells us the extent to which response times for nouns differ from the GM. The fact that the second regression coefficient `FcH02` is close to 50 indicates that response times for nouns (500 ms) slower than the GM of 450 ms. While the nouns are estimated to have 50 ms longer reading times than the GM, the reading times for adjectives are 50 ms faster than the GM.

We have now not only derived contrasts, parameter estimates, and comparisons for the sum contrast, we have also used a powerful and highly general procedure that is used to generate contrasts for many kinds of different comparisons and experimental designs.

8.2.3 Generating contrasts: The `hypr` package

To work with the four-step procedure, i.e., to flexibly design contrasts to estimate specific comparisons, we have developed the R package `hypr` (Maximilian M Rabe et al. 2020). This package allows the researcher to specify the desired comparisons, and based on these comparisons, it automatically generates contrast matrices that allow us to estimate these comparisons in linear models. The functions available in this package thus considerably simplify the implementation of the four-step procedure outlined above. Because `hypr` was originally written with the frequentist framework in mind, the comparisons are expressed as null hypotheses. In the Bayesian framework, these should be treated as comparisons between (bundles of) condition means.

To illustrate the functionality of the `hypr` package, we will use the two comparisons that we had defined and analyzed in the previous section:

$$\beta_1 = \mu_1 - \frac{\mu_1 + \mu_2 + \mu_3}{3} = \mu_1 - GM \quad (8.21)$$

and

$$\beta_2 = \mu_2 - \frac{\mu_1 + \mu_2 + \mu_3}{3} = \mu_2 - GM \quad (8.22)$$

These estimates are effectively comparisons between condition means or between bundles of condition means. That is, μ_1 is compared to the GM and μ_2 is compared to the GM. These two comparisons can be directly entered into R using the `hypr()` function from the `hypr` package. To do so, we use some labels to indicate factor levels. E.g., adjectives, nouns, and verbs can represent factor levels μ_1 , μ_2 , and μ_3 . The first comparison specifies that μ_1 is compared to $\frac{\mu_1 + \mu_2 + \mu_3}{3}$. This can be written as a formula in R: `adjectives ~ (adjectives + nouns + verbs)/3`. The second comparison is that μ_2 is compared to $\frac{\mu_1 + \mu_2 + \mu_3}{3}$, which can be written in R as `nouns ~ (adjectives + nouns + verbs)/3`.

```
HcSum <- hypr(
  b1 = adjectives ~ (adjectives + nouns + verbs) / 3,
  b2 = nouns ~ (adjectives + nouns + verbs) / 3,
  levels = c("adjectives", "nouns", "verbs")
)
HcSum
```

```
## hypr object containing 2 null hypotheses:
## H0.b1: 0 = (2*adjectives - nouns - verbs)/3
## H0.b2: 0 = (2*nouns - adjectives - verbs)/3
##
## Call:
## hypr(b1 = 2/3 * adjectives - 1/3 * nouns - 1/3 * verbs ~ 0, b2 = 2/3 *
##           nouns      -      1/3      *      adjectives      -
##           1/3      *      verbs ~ 0, levels = c("adjectives",
##           "nouns", "verbs"))
```

```

##  

## Hypothesis matrix (transposed):  

##          b1   b2  

## adjectives 2/3 -1/3  

## nouns      -1/3  2/3  

## verbs      -1/3 -1/3  

##  

## Contrast matrix:  

##          b1   b2  

## adjectives 1   0  

## nouns      0   1  

## verbs      -1  -1

```

The results show that the comparisons between condition means have been re-written into a form where 0 is coded on the left side of the equation, and the condition means together with associated weights are written on the right side of the equation. This presentation makes it easy to see the weights of the condition means to code a certain comparison. The next part of the results shows the hypothesis matrix, which contains the weights from the condition means. Thus, `hypr` takes comparisons between condition means as input, and automatically extracts the corresponding weights and encodes them into the hypothesis matrix. `hypr` moreover applies the generalized matrix inverse to obtain the contrast matrix from the hypothesis matrix. The different steps correspond exactly to the steps we had carried out manually in the preceding section. `hypr` automatically performs these steps for us. We can now extract the contrast matrix by a simple function call:

`contr.hypothesis(HcSum)`

```

##          b1   b2  

## adjectives 1   0  

## nouns      0   1  

## verbs      -1  -1  

## attr(,"class")  

## [1] "hypr_cmat" "matrix"    "array"

```

We can assign this contrast to our factor as we did before.

```
contrasts(df_contrasts2$F) <- contr.hypothesis(HcSum)
```

Now, we could again run the same Bayesian linear model. However, since the contrast matrix is now the same as used before, the Bayesian modeling results would also be exactly the same, and we therefore skip the model fitting for brevity.

The `hypr` package can be used to create contrasts for Bayesian models, where the focus lies on estimation of contrasts that code comparisons between condition means or bundles of condition means. Thus, the comparison that one specifies imply the estimation of a difference between condition means or bundles of condition means. We see this in the output of the `hypr()` function (see the first section of the results) - these formulate the comparison in a way that also illustrates the estimation of model parameters. I.e., the comparison $\mu_1 - \frac{\mu_1 + \mu_2 + \mu_3}{3}$ corresponds to a parameter estimate of $b_1 = 2/3*m1 - 1/3*m2 - 1/3*m3$, where $m1$ to $m3$ are the means for each of the conditions. The resulting contrasts will then allow us to estimate the specified differences between condition means or bundles of condition means.

8.3 Other types of contrasts: illustration with a factor with four levels

Here, we introduce repeated difference, Helmert, and polynomial contrasts. For these, it may be instructive to consider an experiment with one between-subject factor with four levels. We load a corresponding dataset, which contains simulated data about response times with a four-level between-subject factor. The sample sizes for each level and the means and standard errors are shown in Table 8.3, and the means and standard errors are also shown graphically in Figure 8.2.

```
data("df_contrasts3")
```

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

We assume that the four factor levels `F1` to `F4` reflect levels of word fre-

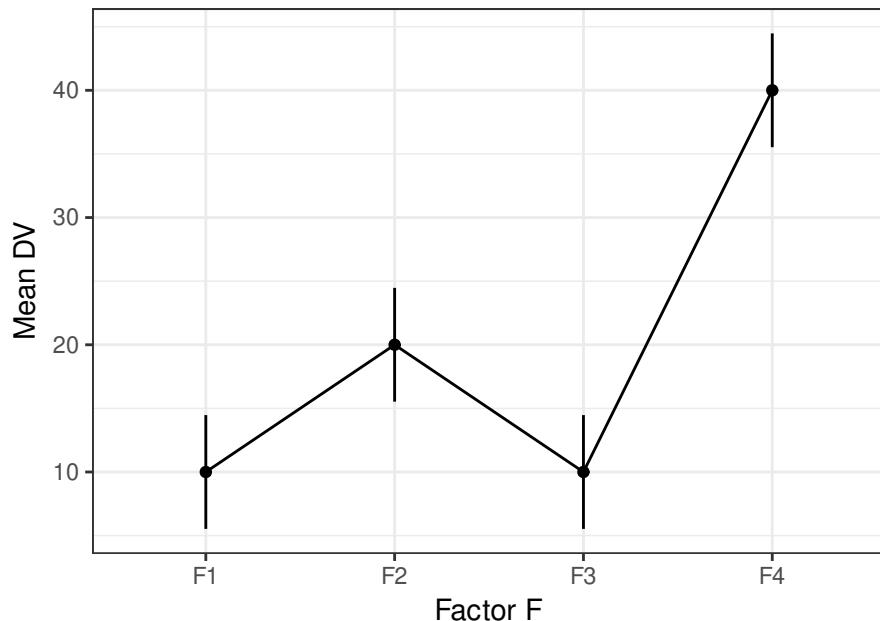


FIGURE 8.2: Means and error bars (showing standard errors) for a simulated dataset with one between-subjects factor with four levels.

quency, including the levels low, medium-low, medium-high, and high frequency words, and that the dependent variable reflects response time. (Qualitatively, the simulated pattern of results is actually empirically observed for word frequency effects on single fixation durations (Heister, Würzner, and Kliegl 2012).)

TABLE 8.3: Summary statistics per condition for the simulated data.

Factor	N data	Est. means	Std. dev.	Std. errors
F1	5	10.0	10.0	4.5
F2	5	20.0	10.0	4.5
F3	5	10.0	10.0	4.5
F4	5	40.0	10.0	4.5

8.3.1 Repeated contrasts

Arguably, the most popular contrast psychologists and psycholinguists are interested in is the comparison between neighboring levels of a factor. This type of contrast is called the repeated contrast. In our example, our research question might be whether the frequency level `low` leads to slower response times than frequency level `medium-low`, whether frequency level `medium-low` leads to slower response times than frequency level `medium-high`, and whether frequency level `medium-high` leads to slower response times than frequency level `high`.

Repeated contrasts are used to implement these comparisons. Consider first how to derive the contrast matrix for repeated contrasts, starting out by specifying the comparisons that are to be estimated. Importantly, this again applies the general strategy of how to translate (any) comparisons between groups or conditions into a set of contrasts, yielding a powerful tool of great value in many research settings. We follow the four-step procedure outlined above.

The first step is to specify our comparisons, and to write them down in a way such that their weights can be extracted easily. For a four-level factor, the three comparisons are:

$$\beta_{2-1} = -1 \cdot \mu_1 + 1 \cdot \mu_2 + 0 \cdot \mu_3 + 0 \cdot \mu_4 \quad (8.23)$$

$$\beta_{3-2} = 0 \cdot \mu_1 - 1 \cdot \mu_2 + 1 \cdot \mu_3 + 0 \cdot \mu_4 \quad (8.24)$$

$$\beta_{4-3} = 0 \cdot \mu_1 + 0 \cdot \mu_2 - 1 \cdot \mu_3 + 1 \cdot \mu_4 \quad (8.25)$$

Here, the μ_x are the mean response times in condition x . Each regression coefficient gives weights to the different condition means. For example, the first estimate (β_{2-1}) estimates the difference between condition mean for `F2` (μ_2) minus the condition mean for `F1` (μ_1), but ignores condition means for `F3` and `F4` (μ_3, μ_4). μ_1 has a weight of -1 , μ_2 has a weight of $+1$, and μ_3 and μ_4 have weights of 0 .

We can write these comparisons into hypr:

```

HcRep <- hypr(
  c2vs1 = F2 ~ F1,
  c3vs2 = F3 ~ F2,
  c4vs3 = F4 ~ F3,
  levels = c("F1", "F2", "F3", "F4")
)
HcRep

## hypr object containing 3 null hypotheses:
## H0.c2vs1: 0 = F2 - F1
## H0.c3vs2: 0 = F3 - F2
## H0.c4vs3: 0 = F4 - F3
##
## Call:
## hypr(c2vs1 = F2 ~ F1 ~ 0, c3vs2 = F3 ~ F2 ~ 0, c4vs3 = F4 ~ F3 ~
##       0, levels = c("F1", "F2", "F3", "F4"))
##
## Hypothesis matrix (transposed):
##   c2vs1 c3vs2 c4vs3
## F1 -1      0      0
## F2  1      -1     0
## F3  0      1     -1
## F4  0      0      1
##
## Contrast matrix:
##   c2vs1 c3vs2 c4vs3
## F1 -3/4   -1/2   -1/4
## F2  1/4    -1/2   -1/4
## F3  1/4    1/2    -1/4
## F4  1/4    1/2    3/4

```

The hypothesis matrix shows exactly the weights that we had written down above. Moreover, we see the contrast matrix. In the case of the repeated contrast, the contrast matrix again looks very different from the hypothesis matrix. In this case, the contrast matrix looks a lot less intuitive than the hypothesis matrix, and if one did not know the associated hypothesis matrix, it seems unclear what the contrast matrix would actu-

ally test. To verify this custom-made contrast matrix, we compare it to the repeated contrast matrix as generated by the R function `contr.sdif()` in the `MASS` package (Ripley 2019). The resulting contrast matrix is identical to our result:

```
MASS::fractions(MASS::contr.sdif(4))
```

```
##   2-1  3-2  4-3
## 1 -3/4 -1/2 -1/4
## 2  1/4 -1/2 -1/4
## 3  1/4  1/2 -1/4
## 4  1/4  1/2  3/4
```

We can thus use either approach (`hypr()` or `contr.sdif()`) to obtain the contrast matrix in this case. Next, we apply the repeated contrasts to the factor `F` in the example data and run a linear model. This allows us to estimate the regression coefficients associated with each contrast. These are compared to the data in Figure 8.2 to test whether the regression coefficients actually correspond to the differences between successive condition means, as intended.

```
contrasts(df_contrasts3$F) <- contr.hypothesis(HcRep)
```

```
fit_Rep <- brm(DV ~ 1 + F,
  data = df_contrasts3,
  family = gaussian(),
  prior = c(
    prior(normal(20, 50), class = Intercept),
    prior(normal(0, 50), class = sigma),
    prior(normal(0, 50), class = b)
  )
)
```

```
fixef(fit_Rep)
```

	Estimate	Est.Error	Q2.5	Q97.5
## Intercept	20.06	2.50	15.2	24.98
## Fc2vs1	9.89	6.86	-4.0	23.57

```
## Fc3vs2      -9.46      6.78 -22.9  3.96
## Fc4vs3      29.34     6.87  15.7 42.86
```

The results show that as expected, the regression coefficients reflect the differences that were of interest: the regression coefficient (`Estimate`) `Fc2vs1` has a value of 10, which exactly corresponds to the difference between the condition mean for `F2` (20) minus the condition mean for `F1` (10), i.e., $20 - 10 = 10$. Likewise, the regression coefficient `Fc3vs2` has a value of -10 , which corresponds to the difference between the condition mean for `F3` (10) minus the condition mean for `F2` (20), i.e., $10 - 20 = -10$. Finally, the regression coefficient `Fc4vs3` has a value of roughly 30, which reflects the difference between condition `F4` (40) minus condition `F3` (10), i.e., $40 - 10 = 30$. Thus, the regression coefficients estimate differences between successive or neighboring condition means.

8.3.2 Helmert contrasts

Another common contrast is the Helmert contrast. In a Helmert contrast for our 4-level factor, the first contrast compares level `F1` versus `F2`. The second contrast compares level `F3` to the average of the first two, i.e., $F3 \sim (F1 + F2)/2$. The third contrast then compares level `F4` to the average of the first three. We can code this contrast in `hypr`:

```
HcHel <- hypr(
  b1 = F2 ~ F1,
  b2 = F3 ~ (F1 + F2) / 2,
  b3 = F4 ~ (F1 + F2 + F3) / 3,
  levels = c("F1", "F2", "F3", "F4")
)
HcHel
```

```
## hypr object containing 3 null hypotheses:
## H0.b1: 0 = F2 - F1
## H0.b2: 0 = (2*F3 - F1 - F2)/2
## H0.b3: 0 = (3*F4 - F1 - F2 - F3)/3
##
## Call:
## hypr(b1 = F2 - F1 ~ 0, b2 = F3 - 1/2 * F1 - 1/2 * F2 ~ 0, b3 = F4 -
```

```

##      1/3 * F1 - 1/3 * F2 - 1/3 * F3 ~ 0, levels = c("F1", "F2",
## "F3", "F4"))
##
## Hypothesis matrix (transposed):
##   b1   b2   b3
## F1   -1 -1/2 -1/3
## F2    1 -1/2 -1/3
## F3    0   1 -1/3
## F4    0   0   1
##
## Contrast matrix:
##   b1   b2   b3
## F1 -1/2 -1/3 -1/4
## F2  1/2 -1/3 -1/4
## F3   0   2/3 -1/4
## F4   0   0   3/4

```

The classical Helmert contrast coded by the function `contr.helmert()` yields a similar but slightly different result:

`contr.helmert(4)`

```

## [,1] [,2] [,3]
## 1   -1   -1   -1
## 2    1   -1   -1
## 3    0    2   -1
## 4    0    0    3

```

These contrasts are scaled versions of our custom Helmert contrast. I.e., the first column of our custom Helmert contrast has to be multiplied by 2 to get the classical version, the second column has to be multiplied by 3, and the fourth column has to be multiplied by 4 to get to our custom Helmert contrast. In fact, the classical Helmert contrast does not directly estimate the differences as explained above, but it estimates scaled versions of these differences. This means that the estimates in the classical Helmert contrast do not directly estimate the condition differences as outlined above, but rather the scaled condition differences. Therefore, we suggest that our custom Helmert contrast defined using the `hypr` func-

tion is more appropriate and intuitive to use. Probably the only reason the classical Helmert contrast uses these scaled differences is that the rescaling yields an easier contrast matrix, which consists of integers rather than fractions. However, this shouldn't be a concern to us, and the intuitive estimates from our custom Helmert contrast seem much more relevant in Bayesian approaches today.

```

contrasts(df_contrasts3$F) <- contr.hypothesis(HcHel)
fit_Hel <- brm(DV ~ 1 + F,
  data = df_contrasts3,
  family = gaussian(),
  prior = c(
    prior(normal(20, 50), class = Intercept),
    prior(normal(0, 50), class = sigma),
    prior(normal(0, 50), class = b)
  )
)

## Compiling Stan program...
## Start sampling

fixef(fit_Hel)

##           Estimate Est.Error   Q2.5 Q97.5
## Intercept    20.03     2.41  15.27 24.83
## Fb1          9.80      6.82 -3.88 22.94
## Fb2         -4.99      6.02 -16.76  7.12
## Fb3         26.35      5.61  15.21 37.33

```

When we fit the Bayesian model using our custom Helmert contrast, we can see that the estimates reflect the comparisons outlined above. The first estimate `Fb1` has a value of roughly 10, reflecting the difference between conditions `F1` and `F2`. The second estimate `Fb2` has a value of 5, which reflects the difference between condition `F3` (10) and the average of the first two conditions ($(10 + 20)/2 = 15$). Last, the estimate `Fb3` reflects the difference between `F4` (40) minus the average of the first three, which is $(10 + 20 + 10)/3 = 13.3$, and is thus $40 - 13.3 = 26.7$.

Box 8.1. Treatment contrast with intercept as the grand mean.

Above, we have introduced the treatment contrast, where each contrast compares one condition to a baseline condition. We have discussed that the intercept in the treatment contrast estimates the condition mean for the baseline condition. There are some applications where this behavior may seem sub-optimal. This can be the case in experimental designs with multiple factors, where we may want to use centered contrasts (we will discuss this below). Moreover, the contrast coding of the fixed effects also defines what the random effects assess. If the intercept assesses the grand mean - rather than the baseline condition - in generalized linear mixed effects models, then the random intercepts reflect the grand mean variance, rather than the variance in the baseline condition.

Interestingly, it is possible to design a treatment contrast where the intercept reflects the grand mean. We implement this using the `hypr` package. The trick is to add the intercept explicitly as a comparison of the average of all four condition means:

```
HcTrGM <- hypr(
  b0 = ~ (F1 + F2 + F3 + F4) / 4,
  b1 = F2 ~ F1,
  b2 = F3 ~ F1,
  b3 = F4 ~ F1,
  levels = c("F1", "F2", "F3", "F4")
)
HcTrGM

## hypr object containing 4 null hypotheses:
## H0.b0: 0 = (F1 + F2 + F3 + F4)/4 (Intercept)
## H0.b1: 0 = F2 - F1
## H0.b2: 0 = F3 - F1
## H0.b3: 0 = F4 - F1
##
## Call:
```

```

## hypr(b0 = 1/4 * F1 + 1/4 * F2 + 1/4 * F3 + 1/4 * F4 ~ 0, b1 = F2 -
##      F1 ~ 0, b2 = F3 - F1 ~ 0, b3 = F4 - F1 ~ 0, levels = c("F1",
## "F2", "F3", "F4"))
##
## Hypothesis matrix (transposed):
##   b0  b1  b2  b3
## F1 1/4  -1  -1  -1
## F2 1/4   1   0   0
## F3 1/4   0   1   0
## F4 1/4   0   0   1
##
## Contrast matrix:
##   b0  b1  b2  b3
## F1   1 -1/4 -1/4 -1/4
## F2   1  3/4 -1/4 -1/4
## F3   1 -1/4  3/4 -1/4
## F4   1 -1/4 -1/4  3/4

```

The hypothesis matrix now explicitly codes the intercept as the first column, where all hypothesis weights are equal and sum up to one. This is coding the intercept. The other hypothesis weights are as expected for the treatment contrast. The contrast matrix now looks very different compared to the standard treatment contrast. Next, we fit a model with this adapted treatment contrast. The function `contr.hypothesis` automatically removes the intercept that is encoded in `HcTrGM`, since this is automatically added by `brms`.

```
contrasts(df_contrasts3$F) <- contr.hypothesis(HcTrGM)
fit_TrGM <- brm(DV ~ 1 + F,
  data = df_contrasts3,
  family = gaussian(),
  prior = c(
    prior(normal(20, 50), class = Intercept),
    prior(normal(0, 50), class = sigma),
    prior(normal(0, 50), class = b)
  )
)

## Compiling Stan program...

## Start sampling

fixef(fit_TrGM)

##           Estimate Est.Error   Q2.5   Q97.5
## Intercept    20.00     2.49  14.96  24.9
## Fb1          9.44     6.95 -4.08  23.6
## Fb2         -0.45     6.99 -13.80 13.4
## Fb3          29.25     6.79  15.52 42.3
```

The results show that the coefficients reflect comparisons of each condition F₂, F₃, and F₄ to the baseline condition F₁. However, the intercept now captures the grand mean across all four conditions of 20.

8.3.3 Contrasts in linear regression analysis: The design or model matrix

We have now discussed how different contrasts are created from the hypothesis matrix. However, we have not treated in detail how exactly contrasts are used in a linear model. Here, we will see that the contrasts for a factor in a linear model are just the same thing as continuous numeric predictors (i.e., covariates) in a linear/multiple regression analysis. That is, contrasts are the way to encode discrete factor levels into numeric predictor variables to use in linear/multiple regression analysis, by encoding

which differences between factor levels are estimated. The contrast matrix X_c that we have looked at so far has one entry (row) for each experimental condition. For use in a linear model, however, the contrast matrix is coded into a design or model matrix X , where each individual data point has one row. The design matrix X can be extracted using the function `model.matrix()`:

```
(contrasts(df_contrasts3$F) <- contr.hypothesis(HcRep)) # contrast matrix

##      c2vs1 c3vs2 c4vs3
## F1 -0.75  -0.5 -0.25
## F2  0.25  -0.5 -0.25
## F3  0.25   0.5 -0.25
## F4  0.25   0.5  0.75
## attr(),"class")
## [1] "hypr_cmat" "matrix"     "array"

covars <- model.matrix(~ 1 + F, df_contrasts3) # design matrix
(covars <- as.data.frame(covars))

##      (Intercept) Fc2vs1 Fc3vs2 Fc4vs3
## 1          1 -0.75  -0.5  -0.25
## 2          1 -0.75  -0.5  -0.25
## 3          1 -0.75  -0.5  -0.25
## 4          1 -0.75  -0.5  -0.25
## 5          1 -0.75  -0.5  -0.25
## 6          1  0.25  -0.5  -0.25
## 7          1  0.25  -0.5  -0.25
## 8          1  0.25  -0.5  -0.25
## 9          1  0.25  -0.5  -0.25
## 10         1  0.25  -0.5  -0.25
## 11         1  0.25   0.5  -0.25
## 12         1  0.25   0.5  -0.25
## 13         1  0.25   0.5  -0.25
## 14         1  0.25   0.5  -0.25
## 15         1  0.25   0.5  -0.25
## 16         1  0.25   0.5  0.75
```

```
## 17      1  0.25  0.5  0.75
## 18      1  0.25  0.5  0.75
## 19      1  0.25  0.5  0.75
## 20      1  0.25  0.5  0.75
```

For each of the 20 subjects, four numbers are stored in this model matrix. They represent the three values of three predictor variables used to predict response times in the task. Indeed, this matrix is exactly the design matrix X commonly used in multiple regression analysis, where each column represents one numeric predictor variable (covariate), and the first column codes the intercept term.

To further illustrate this, the covariates are extracted from this design matrix and stored separately as numeric predictor variables in the data-frame:

```
df_contrasts3[, c("Fc2vs1", "Fc3vs2", "Fc4vs3")] <- covars[, 2:4]
```

They are now used as numeric predictor variables in a multiple regression analysis:

```
fit_m3 <- brm(DV ~ 1 + Fc2vs1 + Fc3vs2 + Fc4vs3,
  data = df_contrasts3,
  family = gaussian(),
  prior = c(
    prior(normal(20, 50), class = Intercept),
    prior(normal(0, 50), class = sigma),
    prior(normal(0, 50), class = b)
  )
)
```

```
fixef(fit_m3)
```

	Estimate	Est.Error	Q2.5	Q97.5
## Intercept	19.99	2.46	15.19	24.84
## Fc2vs1	9.69	6.81	-3.77	23.11
## Fc3vs2	-9.46	6.94	-23.05	4.41

```
## Fc4vs3      29.31     6.98  15.21 42.83
```

The results show that the regression coefficients are the same as in the contrast-based analysis shown in the previous section. This demonstrates that contrasts serve to code discrete factor levels into a linear/multiple regression analysis by numerically encoding comparisons between specific condition means.

8.3.4 Polynomial contrasts

Polynomial contrasts are another option for analyzing factors. Suppose that we expect a linear trend across conditions, where the response increases by a constant magnitude with each successive factor level. This could be the expectation when four levels of a factor reflect decreasing levels of word frequency (i.e., four factor levels: high, medium-high, medium-low, and low word frequency), where one expects the lowest response for high frequency words, and successively higher responses for lower word frequencies. The effect for each individual level of a factor may not be strong enough for detecting it in the statistical model. Specifying a linear trend in a polynomial constraint allows us to pool the whole increase into a single coefficient for the linear trend, increasing statistical sensitivity for estimating/detecting the increase. Such a specification constrains the estimate to one interpretable parameter, e.g., a linear increase across factor levels. The larger the number of factor levels, the more parsimonious are polynomial contrasts compared to contrast-based specifications as introduced in the previous sections. Going beyond a linear trend, one may also have expectations about quadratic trends. For example, one may expect an increase only among very low frequency words, but no difference between high and medium-high frequency words.

```
Xpol <- contr.poly(4)
(contrasts(df_contrasts3$F) <- Xpol)
```

```
##          .L     .Q     .C
## [1,] -0.671  0.5 -0.224
## [2,] -0.224 -0.5  0.671
## [3,]  0.224 -0.5 -0.671
## [4,]  0.671  0.5  0.224
```

```
fit_Pol <- brm(DV ~ 1 + F,
  data = df_contrasts3,
  family = gaussian(),
  prior = c(
    prior(normal(20, 50), class = Intercept),
    prior(normal(0, 50), class = sigma),
    prior(normal(0, 50), class = b)
  )
)
```

```
fixef(fit_Pol)
```

	Estimate	Est.Error	Q2.5	Q97.5
## Intercept	19.98	2.40	15.38	24.8
## F.L	17.64	4.88	7.53	27.2
## F.Q	9.79	4.86	-0.13	19.1
## F.C	13.22	4.95	3.50	23.0

In this example, condition means increase across factor levels in a linear fashion, but there may also be quadratic and cubic trends.

8.3.5 An alternative to contrasts: monotonic effects

An alternative to specifying contrasts to estimate specific comparisons between factor levels is monotonic effects (https://paul-buerkner.github.io/brms/articles/brms_monotonic.html; Bürkner and Charpentier 2020). This simply assumes that the dependent variable increases (or decreases) in a monotonic fashion across levels of an ordered factor. In this kind of analysis, one does not define contrasts specifying differences between (groups of) factor levels. Instead, one estimates one parameter which captures the average increase (or decrease) in the dependent variable associated with two neighboring factor levels. Moreover, one estimates the percentages of the overall increase (or decrease) that is associated with each of the differences between neighboring factor levels (i.e., similar to simple difference contrasts, but measured in percentage increase, and assuming monotonicity, i.e., that the same increase or decrease is present for all simple differences).

To implement a monotonic analysis, we first code the factor F as being an ordered factor (i.e., `ordered=TRUE`). Then, we specify that we want to estimate a monotonic effect of F using the notation `mo(F)` in our call to `brms`:

```
df_contrasts3$F <- factor(df_contrasts3$F, ordered=TRUE)
fit_mo <- brm(DV ~ 1 + mo(F),
  data = df_contrasts3,
  family = gaussian(),
  prior = c(
    prior(normal(20, 50), class = Intercept),
    prior(normal(0, 50), class = sigma),
    prior(normal(0, 50), class = b)
  )
)
```

```
summary(fit_mo)
```

```
## Family: gaussian
## Links: mu = identity; sigma = identity
## Formula: DV ~ 1 + mo(F)
## Data: df_contrasts3 (Number of observations: 20)
## Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##         total post-warmup draws = 4000
##
## Population-Level Effects:
##             Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     9.39      4.29     0.46   17.53 1.00     1606     2121
## moF          9.42      2.50     4.04   14.18 1.00     1853     1719
##
## Simplex Parameters:
##             Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## moF1[1]      0.20      0.14     0.01     0.53 1.00     1915     1316
## moF1[2]      0.11      0.11     0.00     0.41 1.00     2862     2194
## moF1[3]      0.69      0.17     0.27     0.94 1.00     2143     2137
##
## Family Specific Parameters:
```

```

##      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     11.80      2.42     8.19    17.67 1.00     2273     2383
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).

```

The results show that there is an overall positive population-level effect of the factor F with an estimate (mo_F) of 9.42, reflecting an average increase in the dependent variables of roughly 9 with each level of F . The model summary shows estimates for the simplex parameters, which represent the ratios of the overall increase associated with F that can be attributed to each of the differences between neighboring factor levels. The results show that most of the increase is associated with $\text{mo}_{F1[3]}$, i.e., with the last difference, reflecting the difference between F_3 and F_4 , whereas the other two differences ($\text{mo}_{F1[1]}$, reflecting the difference between F_1 and F_2 , and $\text{mo}_{F1[2]}$, reflecting the difference between F_2 and F_3) are smaller. Comparing conditional effects between a model using polynomial contrasts and a model assuming monotonic effects makes it clear that the current model “forces” the effects to increase in a monotonic fashion; see Figure 8.3.

```

conditional_effects(fit_Pol)
conditional_effects(fit_mo)

```

This is regardless of the information provided in the data; see the posterior predictive checks in Figure 8.4.

```

pp_check(fit_Pol, type = "violin_grouped", group = "F", y_draw = "points") +
  theme(legend.position = "bottom") +
  coord_cartesian(ylim = c(-55, 105))
pp_check(fit_mo, type = "violin_grouped", group = "F", y_draw = "points") +
  theme(legend.position = "bottom") +
  coord_cartesian(ylim = c(-55, 105))

```

Interestingly, the monotonicity assumption is violated in the current dataset, since the mean is larger in condition F_2 than in condition F_3 . The

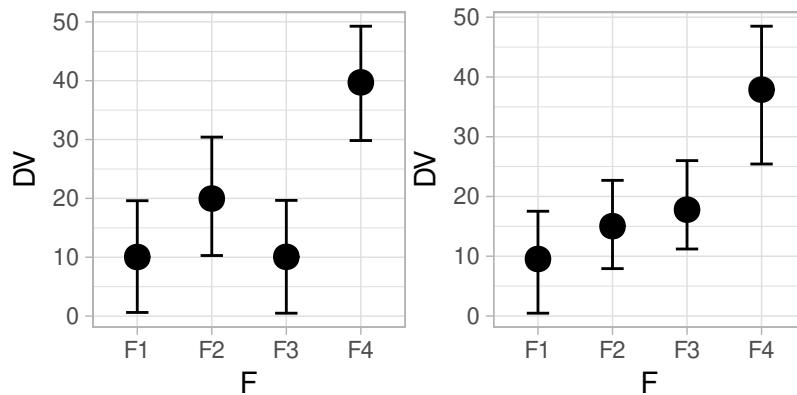


FIGURE 8.3: Conditional effects using the polynomial contrasts on the left side vs. assuming monotonic effects on the right side.

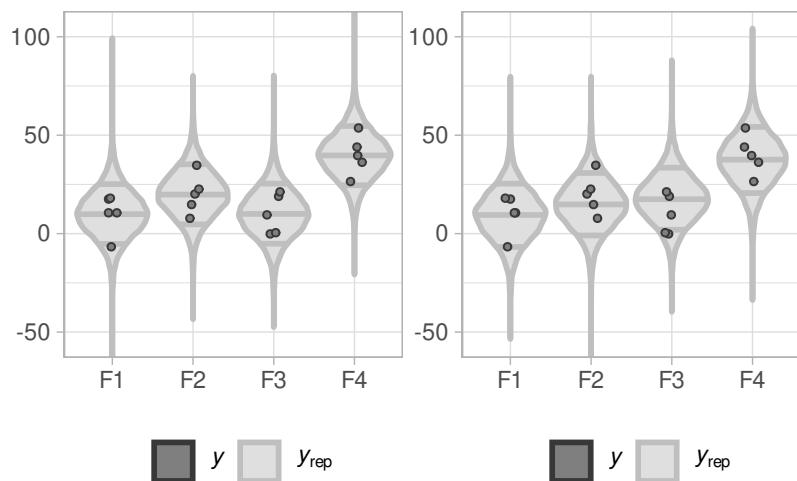


FIGURE 8.4: Posterior predictive distributions by condition using the polynomial contrasts on the left side vs. assuming monotonic effects on the right side.

monotonic model thus assumes this (negative) difference is due to chance; see Figure 8.4.

Estimating such monotonic effects provides an alternative to the contrast coding we treat in the rest of this chapter. It may be relevant when the specific differences between factor levels are not of interest, but when instead the goal is to estimate the overall monotonic effect of a factor and when this overall effect is not well approximated by a simple linear trend.

8.4 What makes a good set of contrasts?

For a factor with I levels one can make only $I - 1$ comparisons within a single model. For example, in a design with one factor with two levels, only one comparison is possible (between the two factor levels). The reason for this is that the intercept is also estimated. More generally, if we have a factor with I_1 and another factor with I_2 levels, then the total number of conditions is $I_1 \times I_2 = v$ (not $I_1 + I_2!$), which implies a maximum of $v - 1$ contrasts.

For example, in a design with one factor with three levels, A, B, and C, in principle one could make three comparisons (A vs. B, A vs. C, B vs. C). However, after defining an intercept, only two means can be compared. Therefore, for a factor with three levels, we define two comparisons within one statistical model.

One critical precondition for contrasts is that they implement different comparisons that are not collinear, that is, that none of the contrasts can be generated from the other contrasts by linear combination. For example, the contrast $c_1 = c(1, 2, 3)$ can be generated from the contrast $c_2 = c(3, 4, 5)$ simply by computing $c_2 - 2$. Therefore, contrasts c_1 and c_2 cannot be used simultaneously. That is, each contrast needs to encode some independent information about the data.

There are (at least) two criteria to decide what a good contrast is. First, *orthogonal contrasts* have advantages as they estimate mutually independent comparisons in the data (see Dobson and Barnett 2011, sec. 6.2.5, p. 91 for a detailed explanation of orthogonality). Second, it is crucial that con-

trasts are defined in a way such that they answer the research questions. This second point is crucial. One way to accomplish this is to use the hypothesis matrix to generate contrasts (e.g., via the `hypr` package), as this ensures that one uses contrasts that exactly estimate the comparisons of interest in a given study.

8.4.1 Centered contrasts

Contrasts are often constrained to be centered, such that the individual contrast coefficients c_i for different factor levels i sum to 0: $\sum_{i=1}^I c_i = 0$. This has advantages when estimating interactions with other factors or covariates (we discuss interactions between factors in a separate chapter below). All contrasts discussed here are centered except for the treatment contrast, in which the contrast coefficients for each contrast do not sum to zero:

```
colSums(contr.treatment(4))
```

```
## 2 3 4
## 1 1 1
```

Other contrasts, such as repeated contrasts, are centered and the contrast coefficients for each contrast sum to 0:

```
colSums(contr.sdif(4))
```

```
## 2-1 3-2 4-3
##    0    0    0
```

The contrast coefficients mentioned above appear in the contrast matrix. By contrast, the weights in the hypothesis matrix are always centered. This is also true for the treatment contrast. The reason is that they code comparisons between conditions or bundles of conditions. The only exception are the weights for the intercept, which are all the same and together always sum to 1 in the hypothesis matrix. This is done to ensure that when applying the generalized matrix inverse, the intercept results in a constant term with values of 1 in the contrast matrix. An important question concerns whether (or when) the intercept needs to be considered

in the generalized matrix inversion, and whether (or when) it can be ignored. This question is closely related to orthogonal contrasts, a concept we turn to below.

8.4.2 Orthogonal contrasts

Two centered contrasts c_1 and c_2 are orthogonal to each other if the following condition applies. Here, i is the i -th cell of the vector representing the contrast.

$$\sum_{i=1}^I c_{1,i} \cdot c_{2,i} = 0 \quad (8.26)$$

Orthogonality can be determined easily in R by computing the correlation between two contrasts. Orthogonal contrasts have a correlation of 0. Contrasts are therefore just a special case for the general case of predictors in regression models, where two numeric predictor variables are orthogonal if they are un-correlated.

For example, coding two factors in a 2×2 design (we return to this case in a section on designs with two factors below) using sum contrasts, these sum contrasts and their interaction are orthogonal to each other:

```
(Xsum <- cbind(F1 = c(1, 1, -1, -1),
                 F2 = c(1, -1, 1, -1),
                 F1xF2 = c(1, -1, -1, 1)))
```

```
##          F1  F2  F1xF2
## [1,]    1   1     1
## [2,]    1  -1    -1
## [3,]   -1   1    -1
## [4,]   -1  -1     1
```

```
cor(Xsum)
```

```
##          F1  F2  F1xF2
## F1      1   0     0
```

```
## F2      0   1      0
## F1xF2  0   0      1
```

The correlations between the different contrasts (i.e., the off-diagonals) are exactly 0. Sum contrasts coding one multi-level factor, however, are not orthogonal to each other:

```
cor(contr.sum(4))
```

```
##      [,1] [,2] [,3]
## [1,]  1.0  0.5  0.5
## [2,]  0.5  1.0  0.5
## [3,]  0.5  0.5  1.0
```

Here, the correlations between individual contrasts, which appear in the off-diagonals, deviate from 0, indicating non-orthogonality. The same is also true for treatment and repeated contrasts:

```
cor(contr.sdif(4))
```

```
##      2-1   3-2   4-3
## 2-1  1.000  0.577  0.333
## 3-2  0.577  1.000  0.577
## 4-3  0.333  0.577  1.000
```

```
cor(contr.treatment(4))
```

```
##      2     3     4
## 2  1.000 -0.333 -0.333
## 3 -0.333  1.000 -0.333
## 4 -0.333 -0.333  1.000
```

Orthogonality of contrasts plays a critical role when computing the generalized inverse. In the inversion operation, orthogonal contrasts are converted independently from each other. That is, the presence or absence of another orthogonal contrast does not change the resulting weights. In fact, for orthogonal contrasts, applying the generalized matrix inverse to

the hypothesis matrix simply furnishes a scaled version of the hypothesis matrix in the contrast matrix (for mathematical details see Schad et al. 2020). However, in Bayesian models, scaling is always important, since we need to interpret the scale in order to define priors or interpret posteriors. Therefore, when working with contrasts in Bayesian models, the generalized matrix inverse is always a good procedure to use.

8.4.3 The role of the intercept in non-centered contrasts

A related question concerns whether the intercept needs to be considered when computing the generalized inverse for a contrast. This is of key importance when using the generalized matrix inverse to define contrasts: the resulting contrast matrix and also the definition of estimates can completely change between a situation where the intercept is explicitly considered or not considered, and can thus change the resulting estimates in possibly unintended ways. Thus, if the definition of the intercept is incorrect, the estimates of slopes may also be wrong.

More specifically, it turns out that considering the intercept is necessary for contrasts that are not centered. This is the case for treatment contrasts which are not centered; e.g., the treatment contrast for two factor levels $c_{1vs0} = c(0, 1)$: $\sum_i c_i = 0 + 1 = 1$. One can actually show that the formula to determine whether contrasts are centered (i.e., $\sum_i c_i = 0$) is the same formula as the formula to test whether a contrast is “orthogonal to the intercept”. Remember that for the intercept, all contrast coefficients are equal to one: $c_{1,i} = 1$ (here, $c_{1,i}$ indicates the vector of contrast coefficients associated with the intercept). We enter these contrast coefficient values into the formula testing whether a contrast is orthogonal to the intercept (here, $c_{2,i}$ indicates the vector of contrast coefficients associated with some contrast for which we want to test whether it is “orthogonal to the intercept”): $\sum_i c_{1,i} \cdot c_{2,i} = \sum_i 1 \cdot c_{2,i} = \sum_i c_{2,i} = 0$. The resulting formula is: $\sum_i c_{2,i} = 0$, which is exactly the formula for whether a contrast is centered. Because of this analogy, treatment contrasts can be viewed to be ‘not orthogonal to the intercept.’ This means that the intercept needs to be considered when computing the generalized inverse for treatment contrasts. As we have discussed above, when the intercept is included in the hypothesis matrix, the weights for this intercept term should sum to

one, as this yields a column of ones for the intercept term in the contrast matrix.

We can see that considering the intercept makes a difference for the treatment contrast. First, we define the comparisons involved in a treatment contrast, where two experimental conditions b and c are each compared to a baseline condition a ($b \sim a$ and $c \sim a$). In addition, we explicitly code the intercept term, which involves a comparison of the baseline to 0 ($a \sim 0$). We take a look at the resulting contrast matrix:

```
hypr(int = a ~ 0, b1 = b ~ a, b2 = c ~ a)

## hypr object containing 3 null hypotheses:
## H0.int: 0 = a      (Intercept)
## H0.b1: 0 = b - a
## H0.b2: 0 = c - a
##
## Call:
## hypr(int = a ~ 0, b1 = b - a ~ 0, b2 = c - a ~ 0, levels = c("a",
## "b", "c"))
##
## Hypothesis matrix (transposed):
##   int b1 b2
## a  1  -1 -1
## b  0   1  0
## c  0   0  1
##
## Contrast matrix:
##   int b1 b2
## a  1   0  0
## b  1   1  0
## c  1   0  1

contr.treatment(c("a", "b", "c"))

##   b  c
## a  0  0
```

```
## b 1 0
## c 0 1
```

This shows a contrast matrix that we know from the treatment contrast. The intercept is coded as a column of 1s. And each of the comparisons is coded as a 1 in the condition which is compared to the baseline, and a 0 in other conditions. The point is here that this gives us the contrast matrix that is expected and known for the treatment contrast.

However, we can also ignore the intercept in the specification of the comparisons:

```
hypr(b1 = m1 ~ m0, b2 = m2 ~ m0)

## hypr object containing 2 null hypotheses:
## H0.b1: 0 = m1 - m0
## H0.b2: 0 = m2 - m0
##
## Call:
## hypr(b1 = m1 - m0 ~ 0, b2 = m2 - m0 ~ 0, levels = c("m0", "m1",
## "m2"))
##
## Hypothesis matrix (transposed):
##      b1 b2
## m0 -1 -1
## m1  1  0
## m2  0  1
##
## Contrast matrix:
##      b1   b2
## m0 -1/3 -1/3
## m1  2/3 -1/3
## m2 -1/3  2/3
```

Interestingly, the resulting contrast matrix now looks very different from the contrast matrix that we know from the treatment contrast. Indeed, this contrast also estimates a reasonable set of quantities. It again estimates whether the condition mean m_1 differs from the baseline and

whether m_2 differs from baseline. The intercept, however, now estimates the average dependent variable across all three conditions (i.e., the GM). This can be seen by explicitly adding a comparison of the average of all three conditions to Θ :

```
hypr(int = (m0 + m1 + m2) / 3 ~ 0, b1 = m1 ~ m0, b2 = m2 ~ m0)

## hypr object containing 3 null hypotheses:
## H0.int: 0 = (m0 + m1 + m2)/3 (Intercept)
## H0.b1: 0 = m1 - m0
## H0.b2: 0 = m2 - m0
##
## Call:
## hypr(int = 1/3 * m0 + 1/3 * m1 + 1/3 * m2 ~ 0, b1 = m1 - m0 ~
##       0, b2 = m2 - m0 ~ 0, levels = c("m0", "m1", "m2"))
##
## Hypothesis matrix (transposed):
##   int b1  b2
## m0 1/3 -1 -1
## m1 1/3  1  0
## m2 1/3  0  1
##
## Contrast matrix:
##   int  b1  b2
## m0    1 -1/3 -1/3
## m1    1  2/3 -1/3
## m2    1 -1/3  2/3
```

The resulting contrast matrix is now the same as when the intercept was ignored, which confirms that these both estimate the same comparison.

8.5 Computing condition means from estimated contrasts

As mentioned earlier, one advantage of Bayesian modeling is that based on the posterior samples, it is possible to very flexibly compute new com-

parisons and estimates. Above (see section 8.1.4), we had discussed the case where the Bayesian model estimated the condition means instead of contrasts by removing the intercept from the brms model (the formula in brms was: $DV \sim -1 + F$). This allowed us to get posterior samples from each condition mean, and then to compute any possible comparison between condition means by subtracting the corresponding samples.

Importantly, posterior samples for the condition means can also be obtained after fitting a model with contrasts. We illustrate this here for the case of sum contrasts. Let's use our above example of a design where we assess response times (in milliseconds, DV) for three different word classes adjectives, nouns, and verbs, that is, for a 3-level factor F . In the above example, factor F was coded using a sum contrast, where the first contrast coded the difference of adjectives from the grand mean, and the second contrast coded the difference of nouns from the grand mean. This was the corresponding contrast matrix:

```
contrasts(df_contrasts2$F) <- contr.hypothesis(HcSum)
contrasts(df_contrasts2$F)
```

```
##          b1  b2
## adjectives  1  0
## nouns       0  1
## verbs       -1 -1
```

We had estimated a brms model for this data. The posterior estimates show results for the intercept (which is estimated to be 450 ms) and for our two coded comparisons. The effect F_{CH01} codes our first comparison that response times for adjectives differ from the grand mean, and show an estimate that response times for adjectives are about 50 ms shorter than the grand mean. Moreover, the effect F_{CH02} codes our second comparison that response times for nouns differ from the grand mean, and show the estimate that response times for nouns are 50 ms longer than the grand mean.

```
fixef(fit_Sum)
```

```
##          Estimate Est.Error Q2.5 Q97.5
```

```

## Intercept      450.2      6.83 436.6 463.9
## FcH01        -49.2      9.71 -68.0 -29.1
## FcH02         49.1      9.61  29.9  67.2

```

However, of course other comparisons might be of interest to us as well. For example, we might be interested in estimating how strongly response times for verbs differ from the grand mean.

To do so, one possible first step is to obtain the posteriors for the response times in each of the three conditions. How can this be done? The first step is to again extract the posterior samples from the model:

```
df_postSamp_Sum <- as_draws_df(fit_Sum)  
str(df_postSamp_Sum)
```

```
## #> #> #> draws_df [4,000 x 8] (S3: draws_df/draws/tbl_df/tbl/data.frame)
## #> #> $ b_Intercept: num [1:4000] 450 444 457 457 456 ...
## #> #> $ b_FcH01     : num [1:4000] -56.7 -49.5 -46.5 -53.8 -51.5 ...
## #> #> $ b_FcH02     : num [1:4000] 53.3 58.7 57.6 43.1 50.9 ...
## #> #> $ sigma       : num [1:4000] 18.2 20.7 22.1 21.7 23.3 ...
## #> #> $ lp__        : num [1:4000] -70.6 -71.8 -71.8 -71.7 -71.1 ...
## #> #> $ .chain      : int [1:4000] 1 1 1 1 1 1 1 1 1 1 ...
## #> #> $ .iteration : int [1:4000] 1 2 3 4 5 6 7 8 9 10 ...
## #> #> $ .draw       : int [1:4000] 1 2 3 4 5 6 7 8 9 10 ...
```

We can see the samples for our first contrast (`b_FcH01`) and for our second contrast (`b_FcH02`). How can we now compute the posterior samples for each of the condition means, i.e., for adjectives, nouns, and verbs? For this, we need to take another look at the contrast matrix.

```
contrasts(df_contrasts2$F)
```

```
##          b1 b2  
## adjectives 1 0  
## nouns      0 1  
## verbs      -1 -1
```

It tells us how the condition means are computed. For adjectives (see the

first row of the contrast matrix), we can see that the response time is computed by taking 1 times the coefficient for b_1 (i.e., F_{CH01}) and 0 times the coefficient for b_2 (i.e., F_{CH02}). Thus, response times for adjectives are simply the samples for the b_1 (i.e., F_{CH01}) contrast. The contrast matrix does not show the intercept term, which is implicitly added. Thus, we also have to add the estimates for the intercept. Thus, the condition mean for adjectives is computed as $b_adjectives \leftarrow b_Intercept + b_FcH01$:

```
df_postSamp_Sum$b_adjectives <-  
  df_postSamp_Sum$b_Intercept + df_postSamp_Sum$b_FcH01
```

Similarly, we can obtain the posterior samples for the response times for nouns. The computation can be seen from the second row of the contrast matrix, which shows that the contrast b_1 (i.e., F_{CH01}) has weight 0 times, whereas the contrast b_2 (i.e., F_{CH02}) has weight 1. Adding the intercept thus gives:

```
df_postSamp_Sum$b_nouns <-  
  df_postSamp_Sum$b_Intercept + df_postSamp_Sum$b_FcH02
```

Finally, we want to obtain posterior samples for the average response times for verbs. For verbs, the third row of the contrast matrix shows two times a -1. Thus, contrasts b_1 (i.e., F_{CH01}) and b_2 (i.e., F_{CH02}) have to be subtracted from the intercept:

```
df_postSamp_Sum$b_verbs <-  
  df_postSamp_Sum$b_Intercept - df_postSamp_Sum$b_FcH01 -  
  df_postSamp_Sum$b_FcH02
```

This yields posterior samples for the mean response times for verbs.

We can now look at the posterior means and 95% credible intervals for adjectives, nouns, and verbs by computing the means and quantiles across all computed samples.

```
postTab <- df_postSamp_Sum %>%
  # removes the meta data:
  as.data.frame() %>%
  select(b_adjectives, b_nouns, b_verbs) %>%
  # transform from wide to long with tidyR:
  pivot_longer(cols = everything(),
               names_to = "condition",
               values_to = "samp") %>%
  group_by(condition) %>%
  summarize(
    post_mean = round(mean(samp)),
    `2.5%` = round(quantile(samp, p = 0.025)),
    `97.5%` = round(quantile(samp, p = 0.975))
  )
```

postTab

```
## # A tibble: 3 x 4
##   condition     post_mean `2.5%` `97.5%
##   <chr>          <dbl>    <dbl>    <dbl>
## 1 b_adjectives     401     378     424
## 2 b_nouns          499     475     523
## 3 b_verbs          450     426     474
```

The results show that as expected the posterior mean for adjectives is 400 ms, for nouns it is 500 ms, and for verbs, the posterior mean is 450 ms. Moreover, we have now posterior credible intervals for each of these estimates.

In fact, `brms` has a very convenient built-in function that allows us to compute these nested effects automatically (`robust = FALSE` shows the posterior mean; by default `brms` shows the posterior median). Notice that you need to add a `[]` after the function call, otherwise `brms` will plot the results.

```
conditional_effects(fit_Sum, robust = FALSE)[]
```

```
## $F
##          F DV cond__ effect1__ estimate__ se__ lower__ upper__
## 1 adjectives 450      1 adjectives      401 11.7     378     424
## 2 nouns      450      1 nouns        499 12.0     475     523
## 3 verbs      450      1 verbs        450 12.0     426     474
```

The same function allows us to visualize the effects, as shown in Figure 8.5.

```
conditional_effects(fit_Sum, robust = FALSE)
```

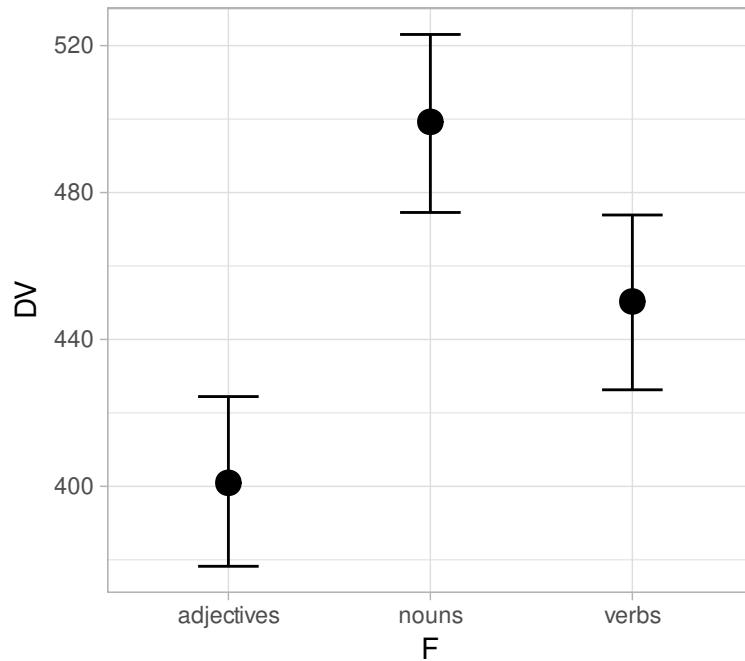


FIGURE 8.5: Estimated condition means, computed from a brms model fitted with a sum contrast.

Importantly, coming back to our hand-crafted computations, the computed posterior samples can be used to compute additional comparisons. For example, we might be interested in how much response times for verbs differ from the grand mean. This can be computed based on the samples for the condition means: we first compute the grand mean from the

three condition means, $b_{GM} \leftarrow (b_{adjectives} + b_{nouns} + b_{verbs})/3$, and then we compare this to the estimate for verbs.

```
df_postSamp_Sum$GM <-  
  (df_postSamp_Sum$b_adjectives +  
   df_postSamp_Sum$b_nouns +  
   df_postSamp_Sum$b_verbs) / 3  
df_postSamp_Sum$b_FcH03 <-  
  df_postSamp_Sum$b_verbs - df_postSamp_Sum$GM  
c(  
  post_mean = mean(df_postSamp_Sum$b_FcH03),  
  quantile(df_postSamp_Sum$b_FcH03, p = c(0.025, 0.975))  
)  
  
## post_mean      2.5%     97.5%  
##      0.158    -19.313    19.458
```

The results show that reading times for verbs are quite the same as the grand mean, with a posterior mean estimate for the differences of nearly 0 ms, and with a 95% credible interval ranging between -20 and +20 ms.

The key message here is that based on the contrast matrix, it is possible to compute posterior samples for the condition means, and then to compute any arbitrary further comparisons or contrasts. We want to stress again that just obtaining the posterior distribution of a comparison does not allow us to argue that we have evidence for the effect; to argue that we have evidence for an effect being present/absent, we need Bayes factors. But the approach we outline above does allow us to obtain posterior means and credible intervals for arbitrary comparisons.

We briefly show how to compute posterior samples for condition means for one more example contrast, namely for repeated contrasts. Here, the contrast matrix is:

```
contrasts(df_contrasts3$F) <- contr.hypothesis(HcRep)  
contrasts(df_contrasts3$F)  
  
##      c2vs1 c3vs2 c4vs3
```

```

## F1 -0.75 -0.5 -0.25
## F2 0.25 -0.5 -0.25
## F3 0.25 0.5 -0.25
## F4 0.25 0.5 0.75

```

The model estimates were:

```
fixef(fit_Rep)
```

	Estimate	Est.Error	Q2.5	Q97.5
## Intercept	20.06	2.50	15.2	24.98
## Fc2vs1	9.89	6.86	-4.0	23.57
## Fc3vs2	-9.46	6.78	-22.9	3.96
## Fc4vs3	29.34	6.87	15.7	42.86

We first obtain the posterior samples for the contrasts:

```
df_postSamp_Rep <- as_draws_df(fit_Rep)  
str(df_postSamp_Rep)
```

```

## #> #> draws_df [4,000 x 9] (S3: draws_df/draws/tbl_df/tbl/data.frame)
## #> $ b_Intercept: num [1:4000] 20.4 16.7 20.3 23.5 14.4 ...
## #> $ b_Fc2vs1   : num [1:4000] 17.01 15.41 4.82 16.02 5.15 ...
## #> $ b_Fc3vs2   : num [1:4000] -14.06 -10.49 -9.82 -13.4 -14.4 ...
## #> $ b_Fc4vs3   : num [1:4000] 20.7 24.6 31.5 27.1 42.5 ...
## #> $ sigma       : num [1:4000] 11.71 10.29 12.19 9.38 12.18 ...
## #> $ lp__        : num [1:4000] -96.4 -95.6 -95.3 -95.8 -98.8 ...
## #> $ .chain      : int [1:4000] 1 1 1 1 1 1 1 1 1 ...
## #> $ .iteration  : int [1:4000] 1 2 3 4 5 6 7 8 9 10 ...
## #> $ .draw       : int [1:4000] 1 2 3 4 5 6 7 8 9 10 ...

```

Then we compute the posterior samples for condition F1. First, we have to add the intercept. Then, we can see in the contrast matrix that to compute the condition mean for F1, we have to add up all contrasts while using the weights $c(-3/4, -1/2, -1/4)$ for each of the three contrasts (see first row of the contrast matrix). Thus, the posterior samples are computed as follows:

```
df_postSamp_Rep$b_F1 <- df_postSamp_Rep$b_Intercept +
-3 / 4 * df_postSamp_Rep$b_Fc2vs1 +
-1 / 2 * df_postSamp_Rep$b_Fc3vs2 +
-1 / 4 * df_postSamp_Rep$b_Fc4vs3
```

The other condition means are computed correspondingly:

```
df_postSamp_Rep$b_F2 <- df_postSamp_Rep$b_Intercept +
1 / 4 * df_postSamp_Rep$b_Fc2vs1 +
-1 / 2 * df_postSamp_Rep$b_Fc3vs2 +
-1 / 4 * df_postSamp_Rep$b_Fc4vs3
df_postSamp_Rep$b_F3 <- df_postSamp_Rep$b_Intercept +
1 / 4 * df_postSamp_Rep$b_Fc2vs1 +
1 / 2 * df_postSamp_Rep$b_Fc3vs2 +
-1 / 4 * df_postSamp_Rep$b_Fc4vs3
df_postSamp_Rep$b_F4 <- df_postSamp_Rep$b_Intercept +
1 / 4 * df_postSamp_Rep$b_Fc2vs1 +
1 / 2 * df_postSamp_Rep$b_Fc3vs2 +
3 / 4 * df_postSamp_Rep$b_Fc4vs3
```

Now we can look at the posterior means and credible intervals:

```
postTab <- df_postSamp_Rep %>%
  as.data.frame() %>%
  select(b_F1, b_F2, b_F3, b_F4) %>%
  pivot_longer(cols = everything(),
               names_to = "condition",
               values_to = "samp") %>%
  group_by(condition) %>%
  summarize(
    post_mean = round(mean(samp)),
    `2.5%` = round(quantile(samp, p = 0.025)),
    `97.5%` = round(quantile(samp, p = 0.975))
  )
```

```
postTab
```

```
## # A tibble: 4 x 4
##   condition post_mean `2.5%` `97.5%
##   <chr>          <dbl>    <dbl>    <dbl>
## 1 b_F1           10      0       20
## 2 b_F2           20     11       30
## 3 b_F3           10      1       20
## # ... with 1 more row
```

We can verify that `brms` function return the same values:

```
conditional_effects(fit_Rep, robust = FALSE)[]
```

```
## $F
##   F DV cond__ effect1__ estimate__ se__ lower__ upper__
## 1 F1 20    1      F1        10.0 4.90  0.365  19.7
## 2 F2 20    1      F2        19.9 4.81  10.678 29.5
## 3 F3 20    1      F3        10.5 4.97  0.792  20.4
## 4 F4 20    1      F4        39.8 4.90 29.994 49.3
```

The posterior means reflect exactly the means in the data (for comparison see Figure 8.2 and Table 8.3). However, we now have posterior samples for each of the conditions and can compute posterior credible intervals as well as new comparisons between conditions.

8.6 Summary

Contrasts in Bayesian models work in exactly the same way as in frequentist models. Contrasts provide a way to tell the model how to code factors into numeric covariates. That is, they provide a way to define which comparisons between which condition means or bundles of condition means should be estimated in the Bayesian model. There are a number of default contrasts, like treatment contrasts, sum contrasts, repeated contrasts, or Helmert contrasts, that are known to estimate specific comparisons be-

tween condition means. A much more powerful procedure is to use the generalized matrix inverse, e.g., as implemented in the `hypr` package, to derive contrasts automatically after specifying the comparisons that a contrast should estimate. We have seen that in Bayesian models, it is quite straightforward to compute posterior samples for new contrasts post-hoc, after the model is fit. However, specifying precise contrasts is still of key importance when doing model comparisons (via Bayes factors) to answer the question of whether the data provide evidence for an effect of interest. If the effect of interest relates to a factor, then it has to be defined using contrast coding.

8.7 Further reading

A good discussion on contrast coding appears in Chapter 15 of Baguley (2012). A book-length treatment is by Rosenthal, Rosnow, and Rubin (2000). A brief discussion on contrast coding appears in Venables and Ripley (2002).

8.8 Exercises

Exercise 8.1. Contrast coding for a four-condition design

Load the following data. These data are from Experiment 1 in a set of reading studies on Persian (Safavi, Husain, and Vasishth 2016). This is a self-paced reading study on particle-verb constructions, with a 2×2 design: distance (short, long) and predictability (predictable, unpredictable). The data are from a critical region in the sentence. All the data from the Safavi, Husain, and Vasishth (2016) paper are available from <https://github.com/vasishth/SafaviEtAl2016>.

```
library(bcogsci)
data("df_persianE1")
```

```
dat1 <- df_persianE1  
head(dat1)  
  
##      subj item   rt distance  predictability  
## 60      4    6  568     short  predictable  
## 94      4   17  517     long  unpredictable  
## 146     4   22  675     short  predictable  
## 185     4    5  575     long  unpredictable  
## 215     4    3  581     long  predictable  
## 285     4    7 1171     long  predictable
```

The four conditions are:

- Distance=short and Predictability=unpredictable
- Distance=short and Predictability=predictable
- Distance=long and Predictability=unpredictable
- Distance=long and Predictability=predictable

The researcher wants to do the following sets of comparisons between condition means:

Compare the condition labeled Distance=short and Predictability=unpredictable with each of the following conditions:

- Distance=short and Predictability=predictable
- Distance=long and Predictability=unpredictable
- Distance=long and Predictability=predictable

Questions:

- Which contrast coding is needed for such a comparison?
- First, define the relevant contrast coding. Hint: You can do it by creating a condition column labeled a,b,c,d and then use a built-in contrast coding function.
- Then, use the `hypr` library function to confirm that your contrast coding actually does the comparison you need.
- Fit a simple linear model with the above contrast coding and display the slopes, which constitute the relevant comparisons.
- Now, compute each of the four conditions' means and check that the

slopes from the linear model correspond to the relevant differences between means that you obtained from the data.

Exercise 8.2. Helmert coding for a four-condition design.

Load the following data:

```
library(bcogsci)
data("df_polarity")
head(df_polarity)
```

	subject	item	condition	times	value
## 1	1	6	f	SFD	328
## 2	1	24	f	SFD	206
## 3	1	35	e	SFD	315
## 4	1	17	e	SFD	265
## 5	1	34	d	SFD	252
## 6	1	7	a	SFD	156

The data come from an eyetracking study in German reported in Vasishth et al. (2008). The experiment is a reading study involving six conditions. The sentences are in English, but the original design was involved German sentences. In German, the word *durchaus* (certainly) is a positive polarity item: in the constructions used in this experiment, *durchaus* cannot have a c-commanding element that is a negative polarity item licensor. Here are the conditions:

- Negative polarity items
 - (a) Grammatical: No man who had a beard was ever thrifty.
 - (b) Ungrammatical (Intrusive NPI licensor): A man who had no beard was ever thrifty.
 - (c) Ungrammatical: A man who had a beard was ever thrifty.
- Positive polarity items
 - (d) Ungrammatical: No man who had a beard was certainly thrifty.
 - (e) Grammatical (Intrusive NPI licensor): A man who had no beard was certainly thrifty.
 - (f) Grammatical: A man who had a beard was certainly thrifty.

We will focus only on re-reading time in this dataset. Subset the data so that we only have re-reading times in the data-frame:

```
dat2 <- subset(df_polarity, times == "RRT")
head(dat2)
```

```
##      subject item condition times value
## 6365       1    20        b   RRT   240
## 6366       1     3        c   RRT  1866
## 6367       1    13        a   RRT   530
## 6368       1    19        a   RRT   269
## 6369       1    27        c   RRT   845
## 6370       1    26        b   RRT   635
```

The comparisons we are interested in are:

- What is the difference in reading time between negative polarity items and positive polarity items?
- Within negative polarity items, what is the difference between grammatical and ungrammatical conditions?
- Within negative polarity items, what is the difference between the two ungrammatical conditions?
- Within positive polarity items, what is the difference between grammatical and ungrammatical conditions?
- Within positive polarity items, what is the difference between the two grammatical conditions?

Use the `hypr` package to specify the comparisons specified above, and then extract the contrast matrix. Finally, specify the contrasts to the condition column in the data frame. Fit a linear model using this contrast specification, and then check that the estimates from the model match the mean differences between the conditions being compared.

Exercise 8.3. Number of possible comparisons in a single model.

- How many comparisons can one make in a single model when there is a single factor with four levels? Why can we not code four comparisons?
- How many comparisons can one code in a model where there are two factors, one with 3 levels and one with 2 levels?

- How about a model for a $2 \times 2 \times 3$ design?

9

Contrast coding for designs with two predictor variables

Chapter 8 provides a basic introduction into contrast coding in situations where there is one predictor variable, i.e., one factor, which can be tested using one specified contrast matrix. Here, we will investigate how contrast coding generalizes to situations where there is more than one predictor variable. This could either be a situation where two factors are present or where one factor is paired with a continuous predictor variable, i.e., a covariate. We first discuss contrast coding for the case of two factors (for 2×2 designs; see section 9.1) and then go on to investigate situations where one predictor is a factor and the other predictor is a covariate (see section 9.2). Moreover, one problem in the analysis of interactions occurs in situations where the model is not linear, but has some non-linear link function, such as e.g., in logistic models or when assuming a log-normally distributed dependent variable. In these situations, the model makes predictions for each condition (i.e., design cell) at the latent level of the linear model. However, sometimes it is important to translate these model predictions to the level of the observations (e.g., to probabilities in a logistic regression model). We will discuss how this can be implemented in section 9.3. We begin by treating contrast coding in a factorial 2×2 design.

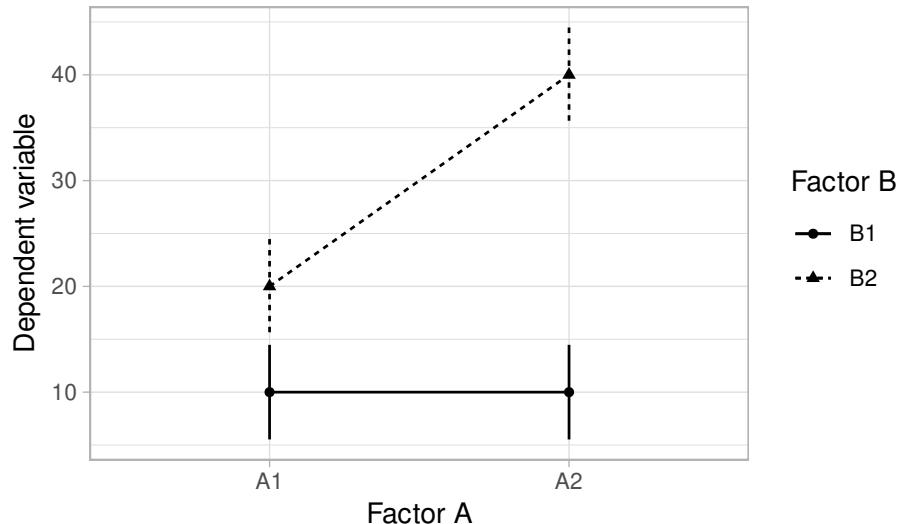
9.1 Contrast coding in a factorial 2×2 design

In chapter 8 in section 8.3, we used a data set with one 4-level factor. Here, we assume that the same four means come from an $A(2) \times B(2)$ between-subject-factor design rather than an $F(4)$ between-subject-factor design. Load the simulated data and show summary statistics in Table 9.1 and in

TABLE 9.1: Summary statistics per condition for the simulated data.

Factor A	Factor B	N data	Means	Std. dev.	Std. errors
A1	B1	5	10	10	4.5
A1	B2	5	20	10	4.5
A2	B1	5	10	10	4.5
A2	B2	5	40	10	4.5

Figure 9.1. The means and standard deviations are exactly the same as in Figure 8.2 and in Table 8.3.

**FIGURE 9.1:** Means and error bars (showing standard errors) for a simulated data set with a two-by-two between-subjects factorial design.

In order to carry out a 2×2 ANOVA-type (main effects and interaction) analysis, one needs sum contrasts in the linear model. (This is true for factors with two levels, but does not generalize to factors with more levels.)

```
# define sum contrasts:
contrasts(df_contrasts4$A) <- contr.sum(2)
contrasts(df_contrasts4$B) <- contr.sum(2)
```

TABLE 9.2: Regression analysis with sum contrasts.

Predictor	Estimate	Std. Error	t	p
(Intercept)	20	2.24	8.94	0.00
A1	-5	2.24	-2.24	0.04
B1	-10	2.24	-4.47	0.00
A1:B1	5	2.24	2.24	0.04

```
# frequentist LM
fit_AB_mr.sum <- lm(DV ~ 1 + A * B, data = df_contrasts4)
```

```
# Bayesian LM
fit_AB.sum <- brm(DV ~ 1 + A * B,
  data = df_contrasts4,
  family = gaussian(),
  prior = c(
    prior(normal(20, 50), class = Intercept),
    prior(normal(0, 50), class = sigma),
    prior(normal(0, 50), class = b)
  )
)
```

```
fixef(fit_AB.sum)
```

```
##           Estimate Est.Error   Q2.5 Q97.5
## Intercept    20.02     2.54 14.84 25.04
## A1          -5.02     2.50 -9.99 -0.21
## B1          -9.96     2.48 -15.03 -4.98
## A1:B1        5.00     2.53  0.07 10.09
```

Next, we reproduce the $A(2) \times B(2)$ - ANOVA with contrasts specified for the corresponding one-way $F(4)$ ANOVA, that is by treating the $2 \times 2 = 4$ condition means as four levels of a single factor F. In other words, we go back to the data frame simulated for the analysis of repeated contrasts

(see chapter 8, section 8.3). We first define weights for condition means according to our hypotheses, invert this matrix, and use it as the contrast matrix for factor F in a LM. We define weights of 1/4 and –1/4. We do so because (a) we want to compare the mean of two conditions to the mean of two other conditions (e.g., factor A compares $\frac{F1+F2}{2}$ to $\frac{F3+F4}{2}$). Moreover, (b) we want coefficients to code half the difference between condition means, reflecting sum contrasts. Together (a+b), this yields weights of $1/2 \cdot 1/2 = 1/4$. The resulting contrast matrix contains contrast coefficients of +1 or –1, showing that we successfully implemented sum contrasts. The results are identical to the previous models.

```
t(fractions(HcInt <- rbind(
  A = c(F1 = 1 / 4, F2 = 1 / 4, F3 = -1 / 4, F4 = -1 / 4),
  B = c(F1 = 1 / 4, F2 = -1 / 4, F3 = 1 / 4, F4 = -1 / 4),
  AxB = c(F1 = 1 / 4, F2 = -1 / 4, F3 = -1 / 4, F4 = 1 / 4)
)))
##      A      B     AxB
## F1  1/4  1/4  1/4
## F2  1/4 -1/4 -1/4
## F3 -1/4  1/4 -1/4
## F4 -1/4 -1/4  1/4

(XcInt <- ginv2(HcInt))

##      A     B   AxB
## F1  1   1   1
## F2  1  -1  -1
## F3 -1   1  -1
## F4 -1  -1   1

contrasts(df_contrasts3$F) <- XcInt

fit_F4.sum <- brm(DV ~ 1 + F,
  data = df_contrasts3,
```

```

family = gaussian(),
prior = c(
  prior(normal(20, 50), class = Intercept),
  prior(normal(0, 50), class = sigma),
  prior(normal(0, 50), class = b)
)
)

fixef(fit_F4.sum)

##           Estimate Est.Error   Q2.5 Q97.5
## Intercept    20.01     2.43 15.26 24.82
## FA          -4.95     2.41 -9.61 -0.12
## FB          -9.99     2.45 -14.75 -5.21
## FAXB         4.99     2.41   0.31  9.89

```

This shows that it is possible to specify the contrasts not only for each factor (e.g., here in the 2×2 design) separately. Instead, one can also pool all experimental conditions (or design cells) into one large factor (here factor F with 4 levels), and specify the contrasts for the main effects and for the interactions in the resulting one large contrast matrix simultaneously.

In this approach, it can again be very useful to apply the `hypr` package to construct contrasts for a 2×2 design. The first hypothesis estimates the main effect A, i.e., it compares the average of F_1 and F_2 to the average of F_3 and F_4 . The second parameter estimates the main effect B, i.e., it compares the average of F_1 and F_3 to the average of F_2 and F_4 . We code direct differences between the averages, i.e., we implement scaled sum contrasts instead of sum contrasts. This is shown below: the contrast matrix contains coefficients of $+1/2$ and $-1/2$ instead of $+1$ and -1 . The interaction term estimates the difference between differences, i.e., the difference between $F_1 - F_2$ and $F_3 - F_4$.

```

hAxB <- hypr(
  A = (F1 + F2) / 2 ~ (F3 + F4) / 2,
  B = (F1 + F3) / 2 ~ (F2 + F4) / 2,

```

```
AxB = (F1 - F2) / 2 ~ (F3 - F4) / 2
)
hAxB
```

```
## hypr object containing 3 null hypotheses:
## H0.A: 0 = (F1 + F2 - F3 - F4)/2
## H0.B: 0 = (F1 + F3 - F2 - F4)/2
## H0.AxB: 0 = (F1 - F2 - F3 + F4)/2
##
## Call:
## hypr(A = 1/2 * F1 + 1/2 * F2 - 1/2 * F3 - 1/2 * F4 ~ 0, B = 1/2 *
##       F1 + 1/2 * F3 - 1/2 * F2 - 1/2 * F4 ~ 0, AxB = 1/2 * F1 -
##       1/2 * F2 - 1/2 * F3 + 1/2 * F4 ~ 0, levels = c("F1", "F2",
##       "F3", "F4"))
##
## Hypothesis matrix (transposed):
##   A     B     AxB
## F1  1/2  1/2  1/2
## F2  1/2 -1/2 -1/2
## F3 -1/2  1/2 -1/2
## F4 -1/2 -1/2  1/2
##
## Contrast matrix:
##   A     B     AxB
## F1  1/2  1/2  1/2
## F2  1/2 -1/2 -1/2
## F3 -1/2  1/2 -1/2
## F4 -1/2 -1/2  1/2
```

```
contrasts(df_contrasts3$F) <- contr.hypothesis(hAxB)
```

```
fit_F4hypr <- brm(DV ~ 1 + F,
  data = df_contrasts3,
  family = gaussian(),
  prior = c(
```

```

prior(normal(20, 50), class = Intercept),
prior(normal(0, 50), class = sigma),
prior(normal(0, 50), class = b)
)
)

fixef(fit_F4hypr)

##           Estimate Est.Error   Q2.5   Q97.5
## Intercept    20.03     2.42  15.18  24.83
## FA          -9.94     4.71 -19.05 -0.43
## FB         -19.86     4.80 -29.35 -10.31
## FAxB         9.93     4.80    0.34  19.74

```

The results show that the estimates have half the size as compared to the sum contrasts - this is the result of the scaling that we applied. I.e., the main effects now directly estimate the difference between averages. However, both contrasts provide the exact same hypothesis tests. Thus, the hypr package can be used to code hypotheses in a 2×2 design.

An alternative way to code main effects and interactions is to use the `ifelse` command in R. For example, if we want to use ± 1 sum contrasts in the above example, we can specify the contrasts for the main effects as vectors:

```

A <- ifelse(df_contrasts3$F %in% c("F1", "F2"), -1, 1)
B <- ifelse(df_contrasts3$F %in% c("F1", "F3"), -1, 1)

```

Now, defining the interaction is simply a matter of multiplying the two vectors:

```
AxB <- A * B
```

If one uses $\pm 1/2$ coding when using this approach, if one wants the interaction term to be on the same scale as the main effects, it would need to be multiplied by 2:

```
A <- ifelse(df_contrasts3$F %in% c("F1", "F2"), -1 / 2, 1 / 2)
B <- ifelse(df_contrasts3$F %in% c("F1", "F3"), -1 / 2, 1 / 2)
## scale has changed:
(AxB <- A * B)

## [1]  0.25  0.25  0.25  0.25  0.25 -0.25 -0.25 -0.25 -0.25 -0.25 -
0.25
## [12] -0.25 -0.25 -0.25 -0.25  0.25  0.25  0.25  0.25  0.25  0.25

## same scale as main effects:
(AxB <- A * B * 2)

## [1]  0.5  0.5  0.5  0.5  0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5 -0.5
## [14] -0.5 -0.5  0.5  0.5  0.5  0.5  0.5
```

This kind of vector-based contrast coding is convenient for more complex designs, such as $2 \times 2 \times 2$ factorial designs.

9.1.1 Nested effects

One can specify hypotheses that do not correspond directly to main effects and interaction of the traditional ANOVA. For example, in a 2×2 experimental design, where factor A codes word frequency (low/high) and factor B is part of speech (noun/verb), one can test the effect of word frequency within nouns and the effect of word frequency within verbs. Formally, A_{B1} versus A_{B2} are nested within levels of B . Said differently, simple effects of factor A are tested for each of the levels of factor B . In this version, we test whether there is a main effect of part of speech (B ; as in traditional ANOVA). However, instead of also estimating the second main effect word frequency, A , and the interaction, we estimate (1) whether the two levels of word frequency, A , differ for the first level of B (i.e., nouns) and (2) whether the two levels of word frequency, A , differ for the second level of B (i.e., verbs). In other words, we estimate whether there are differences for A in each of the levels of B . Often researchers have hypotheses about these differences, and not about the interaction.

```
t(fractions)HcNes <- rbind(  
  B = c(F1 = 1 / 2, F2 = -1 / 2, F3 = 1 / 2, F4 = -1 / 2),  
  B1xA = c(F1 = -1, F2 = 0, F3 = 1, F4 = 0),  
  B2xA = c(F1 = 0, F2 = -1, F3 = 0, F4 = 1)  
)
```

```
##      B      B1xA B2xA  
## F1  1/2   -1     0  
## F2 -1/2    0    -1  
## F3  1/2    1     0  
## F4 -1/2    0     1
```

```
(XcNes <- ginv2(HcNes))
```

```
##      B      B1xA B2xA  
## F1  1/2 -1/2     0  
## F2 -1/2    0 -1/2  
## F3  1/2  1/2     0  
## F4 -1/2    0  1/2
```

```
contrasts(df_contrasts3$F) <- XcNes
```

```
fit_Nest <- brm(DV ~ 1 + F,  
  data = df_contrasts3,  
  family = gaussian(),  
  prior = c(  
    prior(normal(20, 50), class = Intercept),  
    prior(normal(0, 50), class = sigma),  
    prior(normal(0, 50), class = b)  
  )  
)
```

```
fixef(fit_Nest)
```

```
##             Estimate Est.Error Q2.5 Q97.5
```

```

## Intercept      20.0      2.48  15.08  25.0
## FB            -19.9     4.91 -29.55 -10.2
## FB1xA         0.0      7.06 -14.56  14.2
## FB2xA         19.4     7.05  5.59  32.9

```

Regression coefficients estimate the GM, the difference for the main effect of word frequency (A) and the two differences (for B ; i.e., simple main effects) within levels of word frequency (A).

These custom nested contrasts' columns are scaled versions of the corresponding hypothesis matrix. This is the case because the columns are orthogonal. It illustrates the advantage of orthogonal contrasts for the interpretation of regression coefficients: the underlying hypotheses being tested are already clear from the contrast matrix.

There is also a built-in R formula specification for nested designs. The order of factors in the formula from left to right specifies a top-down order of nesting within levels, i.e., here factor A (word frequency) is nested within levels of the factor B (part of speech). This yields the exact same result as our previous result based on custom nested contrasts:

```

contrasts(df_contrasts4$A) <- c(-0.5, +0.5)
contrasts(df_contrasts4$B) <- c(+0.5, -0.5)
fit_Nest2 <- brm(DV ~ 1 + B / A,
  data = df_contrasts4,
  family = gaussian(),
  prior = c(
    prior(normal(20, 50), class = Intercept),
    prior(normal(0, 50), class = sigma),
    prior(normal(0, 50), class = b)
  )
)

```

```
fixef(fit_Nest2)
```

	Estimate	Est.Error	Q2.5	Q97.5
## Intercept	20.00	2.47	15.11	24.80
## B1	-19.97	4.97	-29.78	-9.91

```
## BB1:A1      -0.02      6.76 -13.51 13.39
## BB2:A1      19.56     6.97   5.68 32.58
```

In cases such as these, where A_{B1} vs. A_{B2} are nested within levels of B , it is necessary to include the effect of B (part of speech) in the model, even if one is only interested in the effect of A (word frequency) within levels of B (part of speech). Leaving out factor B in this case would increase posterior uncertainty in the case of fully balanced data, and can lead to biases in parameter estimation in the case the data are not fully balanced.

Again, we show how nested contrasts can be easily implemented using `hypr`:

```
hNest <- hypr(
  B = (F1 + F3) / 2 ~ (F2 + F4) / 2,
  B1xA = F3 ~ F1,
  B2xA = F4 ~ F2
)
hNest

## hypr object containing 3 null hypotheses:
## H0.B: 0 = (F1 + F3 - F2 - F4)/2
## H0.B1xA: 0 = F3 - F1
## H0.B2xA: 0 = F4 - F2
##
## Call:
## hypr(B = 1/2 * F1 + 1/2 * F3 - 1/2 * F2 - 1/2 * F4 ~ 0, B1xA = F3 -
##       F1 ~ 0, B2xA = F4 - F2 ~ 0, levels = c("F1", "F2", "F3",
##       "F4"))
##
## Hypothesis matrix (transposed):
##      B    B1xA  B2xA
## F1  1/2    -1     0
## F2 -1/2     0    -1
## F3  1/2     1     0
## F4 -1/2     0     1
##
## Contrast matrix:
```

```

##      B      B1xA B2xA
## F1  1/2 -1/2    0
## F2 -1/2    0 -1/2
## F3  1/2  1/2    0
## F4 -1/2    0  1/2

contrasts(df_contrasts3$F) <- contr.hypothesis(hNest)

```

```

fit_NestHypr <- brm(DV ~ 1 + F,
  data = df_contrasts3,
  family = gaussian(),
  prior = c(
    prior(normal(20, 50), class = Intercept),
    prior(normal(0, 50), class = sigma),
    prior(normal(0, 50), class = b)
  )
)

```

```
fixef(fit_NestHypr)
```

	Estimate	Est.Error	Q2.5	Q97.5
## Intercept	19.98	2.47	15.06	25.0
## FB	-19.74	4.98	-29.48	-10.2
## FB1xA	0.23	7.02	-13.49	14.2
## FB2xA	19.66	6.80	6.16	32.8

Of course, we can also ask the reverse question: Are there differences for part of speech (B) in the levels of word frequency (A ; in addition to estimating the main effect of word frequency, A)? That is, do nouns differ from verbs for low-frequency words (B_{A1}) and do nouns differ from verbs for high-frequency words (B_{A2})?

```

hNest2 <- hypr(
  A = (F1 + F2) / 2 ~ (F3 + F4) / 2,
  A1xB = F2 ~ F1,

```

```
A2xB = F4 ~ F3
)
hNest2
```

```
## hypr object containing 3 null hypotheses:
## H0.A: 0 = (F1 + F2 - F3 - F4)/2
## H0.A1xB: 0 = F2 - F1
## H0.A2xB: 0 = F4 - F3
##
## Call:
## hypr(A = 1/2 * F1 + 1/2 * F2 - 1/2 * F3 - 1/2 * F4 ~ 0, A1xB = F2 -
##       F1 ~ 0, A2xB = F4 - F3 ~ 0, levels = c("F1", "F2", "F3",
##       "F4"))
##
## Hypothesis matrix (transposed):
##   A   A1xB A2xB
## F1  1/2   -1    0
## F2  1/2    1    0
## F3 -1/2    0   -1
## F4 -1/2    0    1
##
## Contrast matrix:
##   A   A1xB A2xB
## F1  1/2 -1/2    0
## F2  1/2  1/2    0
## F3 -1/2    0 -1/2
## F4 -1/2    0  1/2
```

```
contrasts(df_contrasts3$F) <- contr.hypothesis(hNest2)
```

```
fit_Nest2Hypr <- brm(DV ~ 1 + F,
  data = df_contrasts3,
  family = gaussian(),
  prior = c(
    prior(normal(20, 50), class = Intercept),
```

```

prior(normal(0, 50), class = sigma),
prior(normal(0, 50), class = b)
)
)

fixef(fit_Nest2Hypr)

##           Estimate Est.Error   Q2.5 Q97.5
## Intercept    20.05     2.45 15.08 25.0
## FA          -9.93     4.82 -19.55 -0.5
## FA1xB        9.89     6.86 -3.89 23.4
## FA2xB       29.34     6.88 15.52 42.4

```

Regression coefficients estimate the GM, the difference for the main effect of word frequency (A) and the two part of speech effects (for B ; i.e., simple main effects) within levels of word frequency (A).

9.1.2 Interactions between contrasts

Importantly, in a 2×2 experimental design, the results from sum contrasts are equivalent to typical ANOVA results that we see in frequentist analyses. This means that sum contrasts assess the main effects and the interactions. One interesting question arises here is: what would happen in a 2×2 design if we had used treatment contrasts instead of sum contrasts? Is it still possible to meaningfully interpret the results from the treatment contrasts in a simple 2×2 design?

This leads us to a very important principle in interpreting results from contrasts: When interactions between contrasts are included in a model, then the results for one contrast actually depend on the specification of the other contrast(s) in the analysis! This may be counter-intuitive at first. However, it is very important and essential to keep in mind when interpreting results from contrasts. How does this work in detail?

The general rule to remember is that the effect of one contrast measures its effect at the location 0 of the other contrast(s) in the analysis. What does that mean? Let us consider the example that we use two treatment

contrasts in a 2×2 design. Let's take a look at the effect of factor A. How can we interpret what this measures or tests? This effect actually tests the effect of factor A at the “location” where factor B is coded as 0. Factor B is coded as a treatment contrast, that is, it codes a zero at its baseline condition, which is B1. Thus, the effect of factor A tests the effect of A nested within the baseline condition of B, i.e., a simple effect. We take a look at the data presented in Figure 9.1, what this nested effect should be. Figure 9.1 shows that the effect of factor A nested in B1 is 0. If we now compare this to the results from the linear model, it is indeed clear that the effect of factor A is exactly estimated as 0. As expected, when factor B is coded as a treatment contrast, the effect of factor A estimates the effect of A nested within the baseline level of factor B.

Next, consider the effect of factor B. According to the same logic, this effect estimates the effect of factor B at the “location” where factor A is 0. Factor A is also coded as a treatment contrast, that is, it codes its baseline condition A1 as 0. The effect of factor B estimates the effect of B nested within the baseline condition of A. Figure 9.1 shows that this effect should be 10.

How do we know what the “location” is, where a contrast applies? For the treatment contrasts discussed here, it is possible to reason this through because all contrasts are coded as 0 or 1. However, how is it possible to derive the “location” in general? What we can do is to look at the hypotheses tested by the treatment contrasts (or the comparisons that are estimated) in the presence of an interaction between them by using the generalized matrix inverse. We go back to the default treatment contrasts. Then we extract the contrast matrix from the design matrix:

```
contrasts(df_contrasts4$A) <- contr.treatment(2)
contrasts(df_contrasts4$B) <- contr.treatment(2)

XcTr <- df_contrasts4 %>%
  group_by(A, B) %>%
  summarise() %>%
  model.matrix(~ 1 + A * B, .) %>%
  as.data.frame() %>%
  as.matrix()
```

```
rownames(XcTr) <- c("A1_B1", "A1_B2", "A2_B1", "A2_B2")
XcTr

##          (Intercept) A2 B2 A2:B2
## A1_B1         1  0  0     0
## A1_B2         1  0  1     0
## A2_B1         1  1  0     0
## A2_B2         1  1  1     1
```

This shows the treatment contrast for factors A and B, and their interaction. We can now assign this contrast matrix to a `hypr` object. `hypr` automatically converts the contrast matrix into a hypothesis matrix, such that we can read from the hypothesis matrix which comparison are being estimated by the different contrasts.

```
htr <- hypr() # initialize empty hypr object
cmat(htr) <- XcTr # assign contrast matrix to hypr object
htr # look at the resulting hypothesis matrix

## hypr object containing 4 null hypotheses:
## H0.(Intercept): 0 = A1_B1                               (Intercept)
##           H0.A2: 0 = -A1_B1 + A2_B1
##           H0.B2: 0 = -A1_B1 + A1_B2
##           H0.A2:B2: 0 = A1_B1 - A1_B2 - A2_B1 + A2_B2
##
## Call:
## hypr(`(Intercept)` = A1_B1 ~ 0, A2 = -A1_B1 + A2_B1 ~ 0, B2 = -
## A1_B1 +
##           A1_B2 ~ 0, `A2:B2` = A1_B1 - A1_B2 -
## A2_B1 + A2_B2 ~ 0, levels = c("A1_B1",
## "A1_B2", "A2_B1", "A2_B2"))
##
## Hypothesis matrix (transposed):
##          (Intercept) A2 B2 A2:B2
## A1_B1  1         -1 -1  1
## A1_B2  0         0  1 -1
## A2_B1  0         1  0 -1
```

```
## A2_B2 0          0 0 1
##
## Contrast matrix:
##      (Intercept) A2 B2 A2:B2
## A1_B1 1          0 0 0
## A1_B2 1          0 1 0
## A2_B1 1          1 0 0
## A2_B2 1          1 1 1
```

The same result is obtained by applying the generalized inverse to the contrast matrix (this is what `hypr` does as well). An important fact is that when we apply the generalized inverse to the contrast matrix, we obtain the corresponding hypothesis matrix (for details see Schad et al. 2020).

```
t(ginv2(XcTr))
```

```
##      (Intercept) A2 B2 A2:B2
## A1_B1 1          -1 -1 1
## A1_B2 0          0 1 -1
## A2_B1 0          1 0 -1
## A2_B2 0          0 0 1
```

As discussed above, the effect of factor A estimates its effect nested within the baseline level of factor B. Likewise, the effect of factor B estimates its effect nested within the baseline level of factor A.

How does this work for sum contrasts? They do not have a baseline condition that is coded as 0. In sum contrasts, however, the average of the contrast coefficients is 0. Therefore, effects estimate the average effect across factor levels, i.e., they estimate main effect. This is what is typically also tested in standard ANOVA. Let's look at the example shown in Table 9.2: given that factor B has a sum contrast, the main effect of factor A is tested as the average across levels of factor B. Figure 9.1 shows that the effect of factor A in level B1 is $10 - 10 = 0$, and in level B2 it is $20 - 40 = -20$. The average effect across both levels is $(0 - 20)/2 = -10$. Due to the sum contrast coding, we have to divide this by 2, yielding an expected effect of $-10/2 = -5$. This is exactly what the effect of factor A measures (see Table 9.2, Estimate for A1).

Similarly, factor B tests its effect at the location 0 of factor A. Again, 0 is exactly the mean of the contrast coefficients from factor A, which is coded as a sum contrast. Therefore, factor B tests the effect of B averaged across factor levels of A, i.e., the main effect of B. For factor level A1, factor B has an effect of $10 - 20 = -10$. For factor level A2, factor B has an effect of $10 - 40 = -30$. The average effect is $(-10 - 30)/2 = -20$, which again needs to be divided by 2 due to the sum contrast. This yields exactly the estimate of -10 that is also reported in Table 9.2 (Estimate for B1).

Again, we look at the hypothesis matrix for the main effects and the interaction:

```
contrasts(df_contrasts4$A) <- contr.sum(2)
contrasts(df_contrasts4$B) <- contr.sum(2)
XcSum <- df_contrasts4 %>%
  group_by(A, B) %>%
  summarise() %>%
  model.matrix(~ 1 + A * B, .) %>%
  as.data.frame() %>%
  as.matrix()
rownames(XcSum) <- c("A1_B1", "A1_B2", "A2_B1", "A2_B2")

hsum <- hypr() # initialize empty hypr object
cmat(hsum) <- XcSum # assign contrast matrix to hypr object
hsum # look at the resulting hypothesis matrix
```

```
## hypr object containing 4 null hypotheses:
## H0.(Intercept): 0 = (A1_B1 + A1_B2 + A2_B1 + A2_B2)/4 (Intercept)
##           H0.A1: 0 = (A1_B1 + A1_B2 - A2_B1 - A2_B2)/4
##           H0.B1: 0 = (A1_B1 - A1_B2 + A2_B1 - A2_B2)/4
##           H0.A1:B1: 0 = (A1_B1 - A1_B2 - A2_B1 + A2_B2)/4
##
## Call:
## hypr(`(Intercept)` = 1/4 * A1_B1 + 1/4 * A1_B2 + 1/4 * A2_B1 +
##       1/4 * A2_B2 ~ 0, A1 = 1/4 * A1_B1 + 1/4 * A1_B2 - 1/4 * A2_B1 -
##       1/4 * A2_B2 ~ 0, B1 = 1/4 * A1_B1 - 1/4 * A1_B2 + 1/4 * A2_B1 -
##       1/4 * A2_B2 ~ 0, `A1:B1` = 1/4 * A1_B1 - 1/4 * A1_B2 - 1/4 *
```

```

##      A2_B1 + 1/4 * A2_B2 ~ 0, levels = c("A1_B1", "A1_B2", "A2_B1",
## "A2_B2"))
##
## Hypothesis matrix (transposed):
##      (Intercept) A1     B1    A1:B1
## A1_B1   1/4        1/4   1/4
## A1_B2   1/4        1/4 -1/4 -1/4
## A2_B1   1/4       -1/4   1/4 -1/4
## A2_B2   1/4      -1/4  -1/4   1/4
##
## Contrast matrix:
##      (Intercept) A1 B1 A1:B1
## A1_B1   1        1  1  1
## A1_B2   1        1 -1 -1
## A2_B1   1       -1  1 -1
## A2_B2   1       -1 -1  1

```

This shows that each of the effects now does not compute nested comparisons any more, but that they rather test their effect averaged across conditions of the other factor. The averaging involves using weights of $1/2$. Moreover, the regression coefficients in the sum contrast measure half the distance between conditions, leading to weights of $1/2 \cdot 1/2 = 1/4$.

The general rule to remember from these examples is that when interactions between contrasts are estimated, what an effect of a factor estimates depends on the contrast coding of the other factors in the design! The effect of a factor estimates the effect nested within the location zero of the other contrast(s) in an analysis. If another contrast is centered, and zero is the average of this other contrasts' coefficients, then the contrast of interest tests the average or main effect, averaged across the levels of the other factor. Importantly, this property holds only when the interaction between two contrasts is included into a model. If the interaction is omitted and only effects are estimated, then there is no such “action at a distance”.

This may be a very surprising result for interactions of contrasts. However, it is also essential to interpreting contrast coefficients involved in interactions. It is particularly relevant for the analysis of the default treat-

ment contrast, where the main effects estimate nested effects rather than average effects.

9.2 One factor and one covariate

9.2.1 Estimating a group difference and controlling for a covariate

In this section we treat the case where there are again two predictor variables for one dependent variable, but where one predictor variable is a discrete factor, and the other is a continuous covariate. Let's assume we have measured some response time (RT), e.g., in a lexical decision task. We want to predict the response time based on each subject's IQ, and we expect that higher IQ leads to shorter response times. Moreover, we have two groups of each 30 subjects. These are coded as factor F, with factor levels F1 and F2. We assume that these two groups have obtained different training programs to optimize their response times on the task. Group F1 obtained a control training, whereas group F2 obtained training to improve lexical decisions. We want to test whether the training for better lexical decisions in group F2 actually leads to shorter response times compared to the control group F1. This is our main question of interest here, i.e., whether the training program in F2 leads to faster response times compared to the control group F1. We load the data, which is an artificially simulated data set.

Our main effect of interest is the factor F. We want to test its effect on response times and code it using scaled sum contrasts, such that negative

parameter estimates would yield support for our hypothesis that response times are faster in the training group F2:

```
(contrasts(df_contrasts5$F) <- c(-0.5, +0.5))
```

```
## [1] -0.5  0.5
```

We run a brms model to estimate the effect of factor F, i.e., how strongly the response times in the two groups differ from each other.

```
fit_RT_F <- brm(RT ~ 1 + F,
  data = df_contrasts5,
  family = gaussian(),
  prior = c(
    prior(normal(200, 50), class = Intercept),
    prior(normal(0, 50), class = sigma),
    prior(normal(0, 50), class = b)
  )
)
```

```
fixef(fit_RT_F)
```

	Estimate	Est.Error	Q2.5	Q97.5
Intercept	212.3	5.16	202.2	222.47
F1	-23.9	10.34	-44.4	-4.01

We find (see model estimates and data shown in Figure 9.2) that response times in group F2 are roughly 25 ms faster than in group F1 (Estimate of -24). This suggests that as expected, the training program that group F2 obtained seems to be successful in speeding up response times. Recall that one cannot just look at the 95% credible interval and check whether zero is outside the interval to declare that we have found an effect. To make a discovery claim, we need to run a Bayes factor analysis on this data set to directly test this hypothesis, and this may or may not provide evidence for a difference in response times between groups.

However, let's assume we have allocated subjects to the two groups randomly. Let's say that we also measured the IQ of each person using an IQ

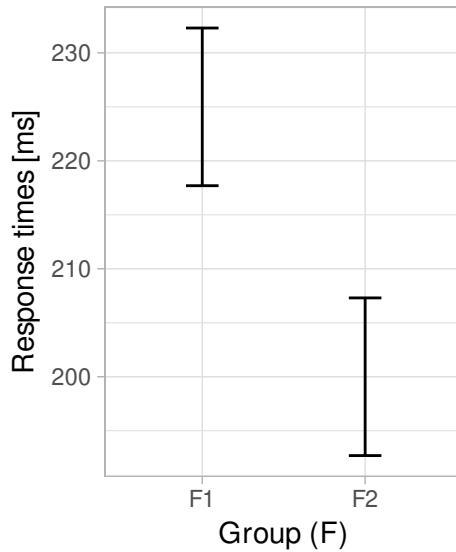


FIGURE 9.2: Means and error bars (showing standard errors) for a simulated data set of response times for two different groups of subjects, who have obtained a training in lexical decisions (F2) versus have obtained a control training (F1).

test. We did so, because we expected that IQ could have a strong influence on response times, and we wanted to control for this influence. We now can check whether the two groups had the same average IQ.

```
df_contrasts5 %>%
  group_by(F) %>%
  summarize(M.IQ = mean(IQ))
```

```
## # A tibble: 2 × 2
##   F      M.IQ
##   <fct> <dbl>
## 1 F1     85
## 2 F2    115
```

Interestingly, group F2 did not only obtain an additional training and had faster response times, but group F2 also had a higher IQ (mean of 115) on average than group F1 (mean IQ = 85). Thus, the random allocation of sub-

jects to the two groups seems to have created - by chance - a difference in IQs. Now we can ask the question: why may response times in group F₂ be faster than in group F₁? Is this because of the training program in F₂? Or is this simply because the average IQ in group F₂ was higher than in group F₁? To investigate this question, we add both predictor variables simultaneously in a brms model. Before we enter the continuous IQ variable, we center it, by subtracting its mean. Centering covariates is generally good practice. Moreover, it is often important to z-transform the covariate, i.e., to not only subtract the mean, but also to divide by its standard deviation (this can be done as follows: `df_contrasts5$IQ.s <- scale(df_contrasts5$IQ)`). The reason why this is often important is that the sampler doesn't work well if predictors have different scales. For the simple models we use here, the sampler works without z-transformation. However, for more realistic and more complex models, z-transformation of covariates is often very important.

```
df_contrasts5$IQ.c <- df_contrasts5$IQ - mean(df_contrasts5$IQ)
fit_RT_F_IQ <- brm(RT ~ 1 + F + IQ.c,
  data = df_contrasts5,
  family = gaussian(),
  prior = c(
    prior(normal(200, 50), class = Intercept),
    prior(normal(0, 50), class = sigma),
    prior(normal(0, 50), class = b)
  )
)
```

```
fixef(fit_RT_F_IQ)
```

	Estimate	Est.Error	Q2.5	Q97.5
## Intercept	212.43	4.83	202.90	221.92
## F1	6.09	13.92	-20.85	33.53
## IQ.c	-1.05	0.34	-1.71	-0.39

The results from the brms model now show that the difference in response times between groups (i.e., factor F) is not estimated to be -25 ms any more, but instead, the estimate is about +7 ms, and the 95% credible in-

terval spans the range -20 to 33 . Thus, there doesn't seem to be much reason to believe any more that the groups would differ. At the same time, we see that the predictor variable IQ shows a negative effect (Estimate = -1 with 95% credible interval: -1.7 to -0.4), suggesting that - as expected - response times seem to be faster in subjects with higher IQ.

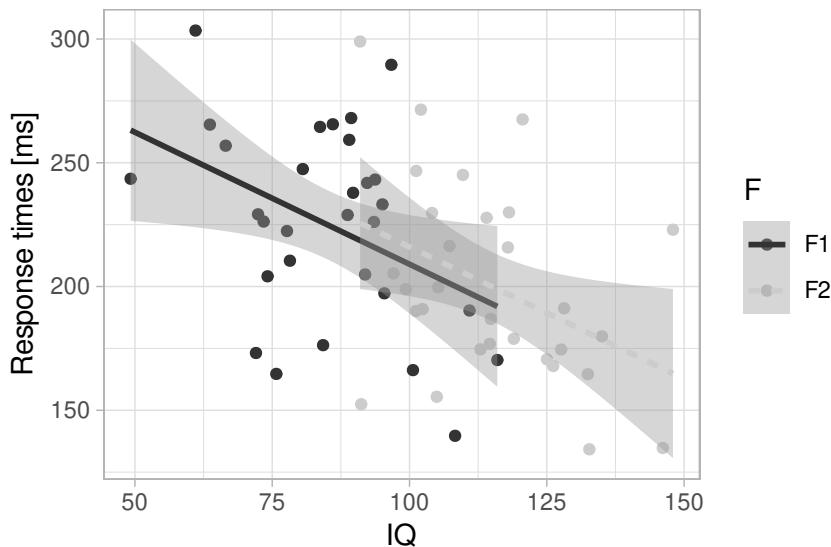


FIGURE 9.3: Response times as a function of individual IQ for two groups with a lexical decision training (F2) versus a control training (F1). Points indicate individual subjects, and lines with error bands indicate linear regression lines.

This result can also be seen in Figure 9.3, which shows that response times decrease with increasing IQ, as suggested by the brms model. However, the heights of the two regression lines do not differ from each other, consistent with the observation in the brms model that the effect of factor F did not seem to differ from zero. That is, factor F in the brms model estimates the difference in height of the regression line between both groups. That the height does not differ and the effect of F is estimated close to zero suggests that in fact group F2 showed faster response times not because of their additional training program. Instead, they had faster response times simply because their IQ was by chance higher on average compared to the control group F1. This analysis is the Bayesian equivalence of the

frequentist “analysis of covariance” (ANCOVA), where it’s possible to test a group difference after “controlling for” the influence of a covariate.

Importantly, we can see in Figure 9.3 that the two regression lines for the two groups are exactly parallel to each other. That is, the influence of IQ on response times seems to be exactly the same in both groups. This is actually a prerequisite for the ANCOVA analysis that needs to be checked in the data. That is, if we want to test the difference between groups after controlling for a covariate (here IQ), we have to test whether the influence of the covariate is the same in both groups. We can investigate this by including an interaction term between the factor and the covariate in the brms model:

```
fit_RT_FxIQ <- brm(RT ~ 1 + F * IQ.c,
  data = df_contrasts5,
  family = gaussian(),
  prior = c(
    prior(normal(200, 50), class = Intercept),
    prior(normal(0, 50), class = sigma),
    prior(normal(0, 50), class = b)
  )
)
```

```
fixef(fit_RT_FxIQ)
```

```
##           Estimate Est.Error   Q2.5   Q97.5
## Intercept    212.45     6.87 198.57 225.84
## F1            6.13    13.63 -21.85  33.28
## IQ.c         -1.05     0.33  -1.72  -0.41
## F1:IQ.c       0.00     0.67  -1.31   1.32
```

The estimate for the interaction (the term “F1:IQ.c”) is very small here (close to 0) and the 95% credible intervals clearly overlap with zero, showing that the two regression lines are estimated to be very similar, or parallel, to each other. If this is the case, then it is possible to correct for IQ when testing the group difference.

9.2.2 Estimating differences in slopes

We now take a look at a different data set.

```
data("df_contrasts6")
levels(df_contrasts6$F) <- c("simple", "complex")
str(df_contrasts6)

## # tibble [60 x 4] (S3:tbl_df/tbl/data.frame)
## $ F : Factor w/ 2 levels "simple","complex": 1 1 1 1 1 1 1 1 1 ...
## $ RT: num [1:60] 223 200 152 206 203 ...
## $ IQ: num [1:60] 99.3 109.5 76.3 87.1 87.6 ...
## $ id: Factor w/ 60 levels "1","2","3","4",...: 1 2 3 4 5 6 7 8 9 10 ...
```

This again contains data from response times (RT) in two groups. Let's assume the two groups have performed two different response time tasks, where one simple RT task doesn't rely on much cognitive processing (group "simple"), whereas the other task is more complex and depends on complex cognitive operations (group "complex"). We therefore expect that RTs in the simple task should be independent of IQ, whereas in the complex task, individuals with a high IQ should be faster in responding compared to individuals with low IQ. Thus, our primary hypothesis of interest states that the influence of IQ on RT differs between conditions. This means that we are interested in the difference between slopes. A slope in a linear regression assesses how strongly the dependent variable (here RT) changes with an increase of one unit on the covariate (here IQ), it thus assesses how "steep" the regression line is. Our hypothesis thus states that the regression lines differ between groups.

The results, displayed in Figure 9.4, suggest that the data seem to support our hypothesis. For the subjects performing the complex task, response times seem to decrease with increasing IQ, whereas for subjects performing the simple task, response times seem to be independent of IQ. As stated before, our primary hypothesis relates to the difference in slopes. Statistically speaking, this is assessed in the interaction between the factor and the covariate. Thus, we run a brms model where the interaction is included. Importantly, we first use scaled sum contrasts for the group effect, and again center the covariate IQ.

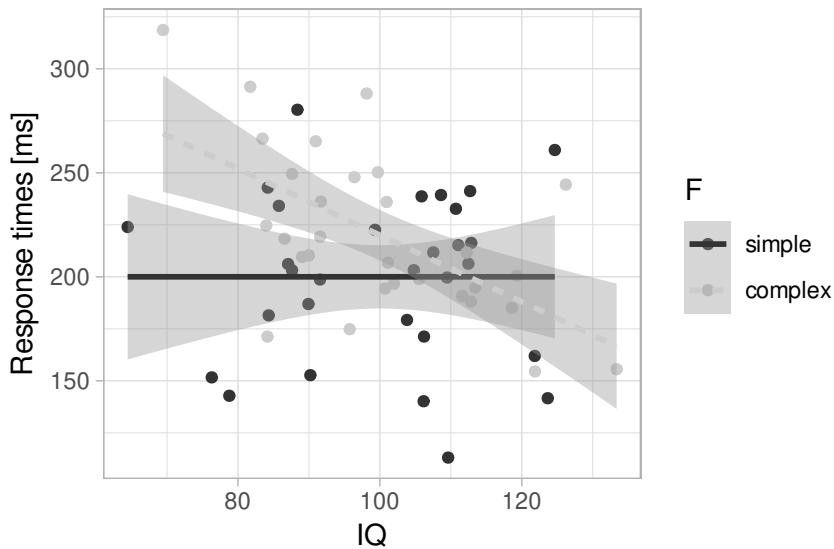


FIGURE 9.4: Response times as a function of individual IQ for two groups performing a simple versus a complex task. Points indicate individual subjects, and lines with error bands indicate linear regression lines.

```

contrasts(df_contrasts6$F) <- c(-0.5, +0.5)
df_contrasts6$IQ.c <- df_contrasts6$IQ - mean(df_contrasts6$IQ)
fit_RT_FxIQ2 <- brm(RT ~ 1 + F * IQ.c,
  data = df_contrasts6,
  family = gaussian(),
  prior = c(
    prior(normal(200, 50), class = Intercept),
    prior(normal(0, 50), class = sigma),
    prior(normal(0, 50), class = b)
  )
)

fixef(fit_RT_FxIQ2)

##           Estimate Est.Error   Q2.5   Q97.5
## Intercept     210.0      4.90 200.59 219.85

```

```
## F1          19.2      9.59  -0.62  38.14
## IQ.c       -0.8      0.33  -1.45  -0.14
## F1:IQ.c    -1.6      0.66  -2.87  -0.28
```

We can see that the main effect of IQ (term “IQ.c”) is negative (-0.8) with 95% credible intervals -1.5 to -0.2 , suggesting that overall response times decrease with increasing IQ. However, this is qualified by the interaction term, which is estimated to be negative (-1.6), with 95% credible intervals -2.9 to -0.3 . This suggests that the slope in the complex group (which was coded as $+0.5$ in the scaled sum contrast) is more negative than the slope in the simple group (which was coded as -0.5 in the scaled sum contrast). Thus, the interaction assesses the difference between slopes.

We can also run a model, where the nested slopes are estimated, i.e., the slope of IQ in the simple group and the slope of IQ in the complex group. This can be implemented by using the nested coding that we learned about in the previous section:

```
fit_RT_FnIQ2 <- brm(RT ~ 1 + F / IQ.c,
  data = df_contrasts6,
  family = gaussian(),
  prior = c(
    prior(normal(200, 50), class = Intercept),
    prior(normal(0, 50), class = sigma),
    prior(normal(0, 50), class = b)
  )
)
```

```
fixef(fit_RT_FnIQ2)
```

	Estimate	Est.Error	Q2.5	Q97.5
## Intercept	210.00	4.84	200.25	219.36
## F1	19.09	9.54	0.70	37.67
## Fsimple:IQ.c	0.01	0.47	-0.91	0.94
## Fcomplex:IQ.c	-1.59	0.46	-2.50	-0.71

Now we see that the slope of IQ in the simple group (“Fsimple:IQ.c”) is

estimated to be 0, with credible intervals clearly including zero. By contrast, the slope in the complex group ("Fcomplex:IQ.c") is estimated as -1.6 (95% CrI = $[-2.5, -0.7]$). This is consistent with our hypothesis that high IQ speeds up response times for the complex but not for the simple task. (To obtain evidence for this effect, we need Bayes factors, which we discuss in a separate chapter.) We can also see from the nested analysis that the difference in slopes between conditions is $-1.6 - 0.0 = -1.6$. This is exactly the value for the interaction term that we estimated in the previous model, demonstrating that interaction terms assess the difference between slopes; i.e., they estimate in how far the regression lines in the two conditions are parallel, with an estimate of 0 indicating perfectly parallel lines.

Interestingly, we can compute posterior samples for the nested slopes from the model with the interaction. That is, we can take the model that estimates main effects and the interaction, and compute posterior samples for the slope of IQ in the simple task and the slope of IQ in the complex task. First, we extract the posterior samples from the model.

```
df_postSamp_RT_FxIQ2 <- as_draws_df(fit_RT_FxIQ2)
str(df_postSamp_RT_FxIQ2)

## #> #> ## draws_df [4,000 x 9] (S3: draws_df/draws/tbl_df/tbl/data.frame)
## #> #> $ b_Intercept: num [1:4000] 211 207 209 208 210 ...
## #> #> $ b_F1       : num [1:4000] 0.086 5.42 3.284 34.142 24.202 ...
## #> #> $ b_IQ.c     : num [1:4000] -1.404 -1.175 -1.351 -0.281 -1.611 ...
## #> #> $ b_F1:IQ.c  : num [1:4000] -1.523 -1.595 -0.976 -2.132 -2.347 ...
## #> #> $ sigma      : num [1:4000] 37.6 37.9 36.5 38.1 40.2 ...
## #> #> $ lp__        : num [1:4000] -324 -322 -323 -323 -324 ...
## #> #> $ .chain      : int [1:4000] 1 1 1 1 1 1 1 1 1 ...
## #> #> $ .iteration : int [1:4000] 1 2 3 4 5 6 7 8 9 10 ...
## #> #> $ .draw       : int [1:4000] 1 2 3 4 5 6 7 8 9 10 ...
```

Then, we take a look at the contrast coefficients for the group factor:

```
contrasts(df_contrasts6$F)
```

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

```
## simple -0.5
## complex 0.5
```

They show a value of -0.5 for the simple group. Thus, to compute the slope for the simple group we have to take the overall slope for IQ.c and subtract -0.5 times the estimate for the interaction:

```
df_postSamp_RT_FxIQ2$b_IQ.c_simple <-
  df_postSamp_RT_FxIQ2$b_IQ.c - 0.5 * df_postSamp_RT_FxIQ2$b_F1:IQ.c`
```

Likewise, to estimate the slope for the complex group we have to take the overall slope for IQ.c and add $+0.5$ times the estimate for the interaction:

```
df_postSamp_RT_FxIQ2$b_IQ.c_complex <-
  df_postSamp_RT_FxIQ2$b_IQ.c + 0.5 * df_postSamp_RT_FxIQ2$b_F1:IQ.c`
```

```
c(
  IQc.c_simple = mean(df_postSamp_RT_FxIQ2$b_IQ.c_simple),
  IQc.c_complex = mean(df_postSamp_RT_FxIQ2$b_IQ.c_complex)
)
```

```
## IQc.c_simple IQc.c_complex
##      -0.00193     -1.59874
```

The results show that the posterior means for the slope of IQ.c are 0 and -1.6 for the simple and the complex groups, as we had found above in the nested analysis.

In most situations one should always center covariates before including them into a model. If covariates are not centered, then the effects (here the effect for the factor) cannot be interpreted as main effects any more.

Interestingly, one can also do analyses with interactions between a covariate and a factor, but by using different contrast codings. For example, if we use treatment contrasts for the factor, then the main effect of IQ.c assesses not the average slope of IQ.c across conditions, but instead the nested slope of IQ.c within the baseline group of the treatment contrast. The interaction still assesses the difference in slopes between groups. In

a situation where there are more than two groups, when one estimates the interaction of contrasts with a covariate, then the contrasts define which slopes are compared with each other in the interaction terms. For example, when using sum contrasts in an example where the influence of IQ is measured on response times for nouns, verbs, and adjectives, then there are two interaction terms: these assess (1) whether the slope of IQ for nouns is different from the average slope across conditions, and (2) whether the slope of IQ for verbs is different from the average slope across conditions. If one uses repeated contrasts in a situation where the influence of IQ on response times is estimated for word frequency conditions “low”, “medium-low”, “medium-high”, and “high”, then there are three interaction terms (one for each contrast). The first interaction term estimates the difference in slopes between “low” and “medium-low” word frequencies, the second interaction term estimates the difference in slopes between “medium-low” and “medium-high” word frequencies, and the third interaction term estimates the difference in slopes between “medium-high” and “high” word frequency conditions. Thus, the logic of how contrasts specify certain comparisons between conditions extends directly to the situation where differences in slopes are estimated.

9.3 Interactions in generalized linear models (with non-linear link functions) and non-linear models

Next, we look at generalized linear models, where a linear predictor is passed through a non-linear link function to predict the dependent variable. Examples for generalized linear models include logistic regression models and models assuming a Poisson distribution. Even though a log-normal model is a linear model on a log-transformed dependent variable, the same techniques apply to this type of model since the logarithm is not linear. Here, we treat an example with a logistic model in a 2×2 factorial between-subject design. The logistic model has the following non-linear link function called the logistic function: $p(y = 1 | x, b) = \frac{1}{1+\exp(-\eta)}$, where η is the latent linear predictor. For example, in our 2×2 factorial design with main effects A and B and their interaction, η is computed as

TABLE 9.3: Summary statistics per condition for the simulated data.

Factor A	Factor B	N data	Means
A1	B1	50	0.2
A1	B2	50	0.5
A2	B1	50	0.2
A2	B2	50	0.8

a linear combination of the intercept plus the main effects and their interaction: $\eta = 1 + \beta_A x_A + \beta_B x_B + \beta_{A \times B} x_{A \times B}$.

Thus, there is a latent level of linear predictions (η), which are then passed through a non-linear link function to predict the probability that the observed data is a success ($p(y = 1)$). We will use this logistic model to analyze an example data set where the dependent variable is dichotomous, coded as either a 1 (indicating success) or a 0 (indicating failure).

We load a simulated data set where the dependent variable codes whether a subject performed a task successfully ($pDV = 1$) or not ($pDV = 0$). Moreover, the data set has two between-subject factors A and B. The means for each of the four conditions are shown in Table 9.3.

```
data("df_contrasts7")
str(df_contrasts7)

### tibble [200 x 4] (S3: tbl_df/tbl/data.frame)
### $ A   : Factor w/ 2 levels "A1","A2": 1 1 1 1 1 1 1 1 1 1 ...
### $ B   : Factor w/ 2 levels "B1","B2": 1 1 1 1 1 1 1 1 1 1 ...
### $ pDV: int [1:200] 0 0 0 1 0 0 0 0 0 0 ...
### $ id  : int [1:200] 1 2 3 4 5 6 7 8 9 10 ...
```

To analyze this data, we use scaled sum contrasts, as we had done above for the 2×2 design with response times as the dependent variable; this allows us to interpret the coefficients directly as main effects. Next, we fit a brms model. The model specification is the same as the model with response times - with two differences: First, the `family` argument is now specified as `family = bernoulli(link = "logit")` to indicate the logistic

model. Moreover, we do not specify a prior for `sigma`, since there is no residual standard deviation in a logistic model.

```
contrasts(df_contrasts7$A) <- c(-0.5, +0.5)
contrasts(df_contrasts7$B) <- c(-0.5, +0.5)
fit_pDV_AB.sum <- brm(pDV ~ 1 + A * B,
  data = df_contrasts7,
  family = bernoulli(link = "logit"),
  prior = c(
    prior(normal(0, 3), class = Intercept),
    prior(normal(0, 3), class = b)
  )
)

fixef(fit_pDV_AB.sum)
```

	Estimate	Est.Error	Q2.5	Q97.5
## Intercept	-0.36	0.17	-0.70	-0.03
## A1	0.70	0.34	0.03	1.39
## B1	2.10	0.34	1.45	2.78
## A1:B1	1.36	0.67	0.08	2.67

Multiplying the main effects, which are coded as ± 0.5 , with each other will lead to a weight of 0.25 in the interaction. Because this changes the scale, it may be better to use ± 1 coding, creating vectors for the main effects:

```
ME_A <- ifelse(df_contrasts7$A == "A1", 1, -1)
ME_B <- ifelse(df_contrasts7$B == "B1", 1, -1)
```

However, in the present example, we continue with the ± 0.5 coding, because as long as one is aware of the scaling, one can interpret the main effects and interactions appropriately.

The results from this analysis show that the estimates for the two main effects ("A1" and "B1") as well as the interaction ("A1:B1") are positive and the 95% credible intervals do not include zero. If we want to make a discovery

claim, we would need to perform Bayes factor analyses to investigate the evidence that there is for each of the effects.

Next, we discuss how we can obtain model predictions for each of the four experimental conditions for this generalized linear model. To obtain such predictions, we first take a look at the contrast matrix. We simultaneously have contrasts for two main effects and one interaction:

```
tab7 <- df_contrasts7 %>%
  group_by(A, B) %>%
  summarize() %>%
  model.matrix(~ A * B, data = .) %>%
  as.data.frame()
row.names(tab7) <- c("A1_B1", "A1_B2", "A2_B1", "A2_B2")
```

tab7

```
##          (Intercept)    A1    B1 A1:B1
## A1_B1           1 -0.5 -0.5  0.25
## A1_B2           1 -0.5  0.5 -0.25
## A2_B1           1  0.5 -0.5 -0.25
## A2_B2           1  0.5  0.5  0.25
```

We obtain the posterior samples for the estimates from the model:

```
df_postSamp_pDV <- as_draws_df(fit_pDV_AB.sum)
str(df_postSamp_pDV)

## #> #> draws_df [4,000 x 8] (S3: draws_df/draws/tbl_df/tbl/data.frame)
## #> $ b_Intercept: num [1:4000] -0.4106 -0.4529 -0.1708 -0.1594 -
## #> 0.0549 ...
## #> $ b_A1        : num [1:4000] 1.095 0.419 1.095 1.415 0.899 ...
## #> $ b_B1        : num [1:4000] 2.25 2.34 1.54 1.67 1.88 ...
## #> $ b_A1:B1    : num [1:4000] 1.564 0.719 1.811 2.435 1.522 ...
## #> $ lp__        : num [1:4000] -119 -120 -121 -124 -120 ...
## #> $ .chain      : int [1:4000] 1 1 1 1 1 1 1 1 1 ...
## #> $ .iteration : int [1:4000] 1 2 3 4 5 6 7 8 9 10 ...
```

```
## $ .draw      : int [1:4000] 1 2 3 4 5 6 7 8 9 10 ...
```

From these, we can compute the posterior samples for the linear predictions for each group. We see in the contrast matrix how we have to combine the posterior samples for the intercept, main effects, and interaction to obtain latent linear predictions for each condition. The first condition (design cell A1, B1) has a weight of 1 for the intercept, and then weights of -0.5 (for the main effect of A), -0.5 (for the main effect of B), and 0.25 (for the interaction). The posterior samples for the other conditions are computed accordingly.

```
df_postSamp_pDV$A1_B1 <-
  1 * df_postSamp_pDV$b_Intercept +
  -0.5 * df_postSamp_pDV$b_A1 +
  -0.5 * df_postSamp_pDV$b_B1 +
  +0.25 * df_postSamp_pDV$b_A1:B1` 

df_postSamp_pDV$A1_B2 <-
  1 * df_postSamp_pDV$b_Intercept +
  -0.5 * df_postSamp_pDV$b_A1 +
  +0.5 * df_postSamp_pDV$b_B1 +
  -0.25 * df_postSamp_pDV$b_A1:B1` 

df_postSamp_pDV$A2_B1 <-
  1 * df_postSamp_pDV$b_Intercept +
  +0.5 * df_postSamp_pDV$b_A1 +
  -0.5 * df_postSamp_pDV$b_B1 +
  -0.25 * df_postSamp_pDV$b_A1:B1` 

df_postSamp_pDV$A2_B2 <-
  1 * df_postSamp_pDV$b_Intercept +
  +0.5 * df_postSamp_pDV$b_A1 +
  +0.5 * df_postSamp_pDV$b_B1 +
  +0.25 * df_postSamp_pDV$b_A1:B1`
```

Now, we have computed posterior samples for estimates of the latent linear predictor η for each experimental condition. We can look at the posterior means:

```
colMeans(as.data.frame(df_postSamp_pDV)[, c("A1_B1", "A1_B2", "A2_B1", "A2_B2")])
```

```
##      A1_B1      A1_B2      A2_B1      A2_B2
## -1.42153  0.00259 -1.39858  1.38601
```

This shows that these values are not on the scale of probabilities. Instead, they are on the scale of the latent linear predictor η . However, for presentation and interpretation of the results, it might be much more informative to look at the condition means in terms of the probabilities of success in each of the four conditions. Given that we have the linear predictions for each condition, this can be easily computed by sending all posterior samples for the linear predictions through the link function. Applying the logistic function (`plogis(.)` in R) transforms the linear predictors to the probability scale:¹

```
df_postSamp_pDV$p_A1_B1 <- plogis(df_postSamp_pDV$A1_B1)
df_postSamp_pDV$p_A1_B2 <- plogis(df_postSamp_pDV$A1_B2)
df_postSamp_pDV$p_A2_B1 <- plogis(df_postSamp_pDV$A2_B1)
df_postSamp_pDV$p_A2_B2 <- plogis(df_postSamp_pDV$A2_B2)
```

Now, we have posterior samples for each condition on the probability scale. We can take a look at the posterior means, and see that these closely correspond to the probabilities in the data that we have seen above in Table 9.3.

```
colMeans(as.data.frame(df_postSamp_pDV)[, c(
  "p_A1_B1", "p_A1_B2",
  "p_A2_B1", "p_A2_B2"
)])
```

```
## p_A1_B1 p_A1_B2 p_A2_B1 p_A2_B2
## 0.200   0.501   0.204   0.794
```

Of course, the advantage is that we now have posterior samples for these conditions available, and can compute posterior 95% credible intervals

¹The same (with lower precision) can be achieved using `1/1+exp(-.)`.

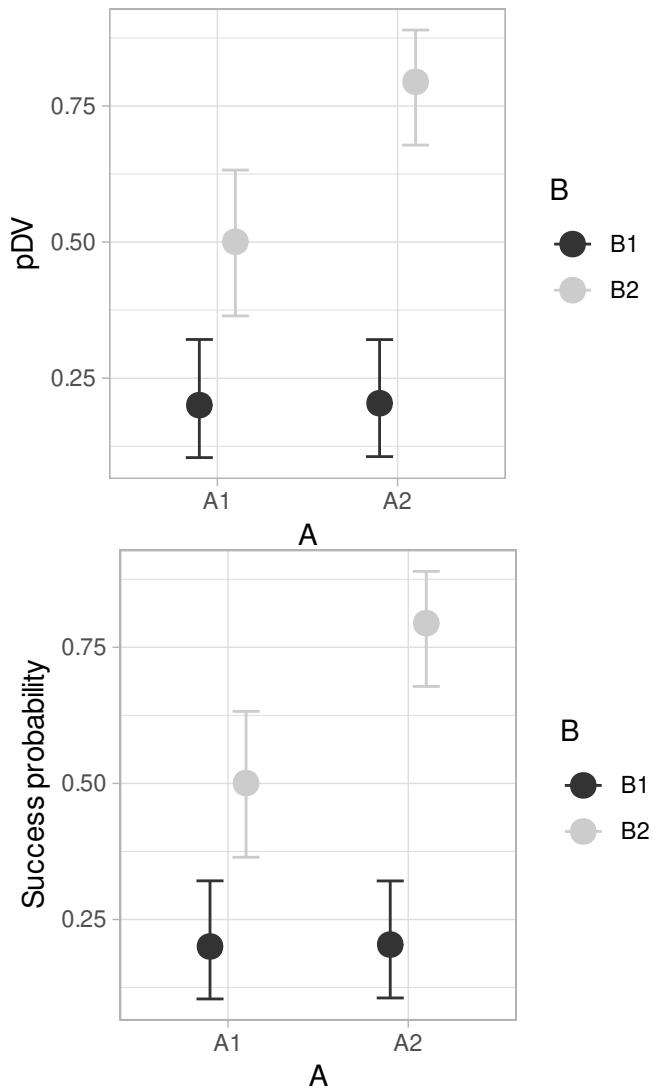
(also see Figure ??). Rather than do it manually, the function `conditional_effects` can do this for us. By default, all main effects and two-way interactions estimated in the model are shown (this can be changed by including, for example, `effects = "A:B"`).

```
conditional_effects(fit_pDV_AB.sum, robust = FALSE)[]

## $A
##   A   pDV  B cond__ effect1__ estimate__    se__ lower__ upper__
## 1 A1 0.425 B1      1       A1     0.200 0.0560  0.104  0.321
## 2 A2 0.425 B1      1       A2     0.204 0.0557  0.106  0.321
##
## $B
##   B   pDV  A cond__ effect1__ estimate__    se__ lower__ upper__
## 1 B1 0.425 A1      1       B1     0.200 0.0560  0.104  0.321
## 2 B2 0.425 A1      1       B2     0.501 0.0693  0.364  0.632
##
## $`A:B`
##   A   B   pDV cond__ effect1__ effect2__ estimate__    se__ lower__ upper__
## 1 A1 B1 0.425      1       A1       B1     0.200 0.0560  0.104
## 2 A1 B2 0.425      1       A1       B2     0.501 0.0693  0.364
## 3 A2 B1 0.425      1       A2       B1     0.204 0.0557  0.106
## 4 A2 B2 0.425      1       A2       B2     0.794 0.0550  0.678
##
##   upper__
## 1  0.321
## 2  0.632
## 3  0.321
## 4  0.890
```

We plot the two-way interaction using `brms` embedding the `conditional_effects()` call in `plot().[[1]]`. This allows us to select the first (and here the only) `ggplot` element and to customize it.

```
plot(conditional_effects(fit_pDV_AB.sum, effects = "A:B", robust = FALSE))[[1]] +
  labs(
    y = "Success probability"
  )
```



9.4 Summary

To summarize, we have seen interesting results for contrasts in the context of 2×2 designs, where depending on the contrast coding, the factors estimated nested effects (treatment contrasts) or main effects (sum contrasts). We also saw that it is possible to code contrasts for a 2×2 design,

by creating one factor comprising all design cells, and by specifying all effects of interest in one large contrast matrix. In designs with one factor and one covariate it is possible to control group-differences for differences in the covariate (ANCOVA), or to test whether regression slopes are parallel in different experimental conditions. Last, in generalized linear models with non-linear link functions it is possible to obtain posterior samples not only on the latent scale of linear predictors, but also on the scale of the response.

9.5 Further readings

Analysis of variance is discussed in detail in Maxwell, Delaney, and Kelley (2017). A practical book on ANOVA using R is Faraway (2002).

9.6 Exercises

Exercise 9.1. ANOVA coding for a four-condition design.

Load the following data. These data are from Experiment 1 in a set of reading studies on Persian (Safavi, Husain, and Vasishth 2016); we encountered these data in the preceding chapter's exercises.

```
library(bcogsci)
data("df_persianE1")
dat1 <- df_persianE1
head(dat1)

##      subj item    rt distance predability
##  60      4     6   568    short  predictable
##  94      4    17   517     long  unpredictable
## 146      4    22   675    short  predictable
## 185      4     5   575     long  unpredictable
## 215      4     3   581     long  predictable
```

```
## 285     4     7 1171      long    predictable
```

The four conditions are:

- Distance=short and Predictability=unpredictable
- Distance=short and Predictability=predictable
- Distance=long and Predictability=unpredictable
- Distance=long and Predictability=predictable

For the data given above, define an ANOVA-style contrast coding, and compute main effects and interactions. Check with `hypr` what the hypothesis tests are with an ANOVA coding, and write down the null hypotheses.

Exercise 9.2. ANOVA and nested comparisons in a $2 \times 2 \times 2$ design

Load the following data set. This is $2 \times 2 \times 2$ design from Jäger et al. (2020), with the factors Grammaticality (grammatical vs. ungrammatical), Dependency (Agreement vs. Reflexives), and Interference (Interference vs. no interference). The experiment is a replication attempt of Experiment 1 reported in Dillon et al. (2013).

```
library(bcogsci)
data("df_dillonrep")
```

- The grammatical conditions are a,b,e,f. The rest of the conditions are ungrammatical.
- The agreement conditions are a,b,c,d. The other conditions are reflexives.
- The interference conditions are a,d,e,h, and the others are the no-interference conditions.

The dependent measure of interest is TFT (total fixation time, in milliseconds).

Using a linear model, do a main effects and interactions ANOVA contrast coding, and obtain an estimate of the main effects of Grammaticality, Dependency, and Interference, and all interactions. You may find it easier to code the contrasts coding the main effects as +1, -1, using `ifelse` in R to code vectors corresponding to each main effect. This will make the specification of the interactions easy.

The researchers had a further research hypothesis: in ungrammatical sentences only, agreement would show an interference effect but reflexives would not. In grammatical sentences, both agreement and reflexives are expected to show interference effects. This kind of research question can be answered with nested contrast coding.

To carry out the relevant nested contrasts, define contrasts that estimate the effects of

- grammaticality
- dependency type
- the interaction between grammaticality and dependency type
- reflexives interference within grammatical conditions
- agreement interference within grammatical conditions
- reflexives interference within ungrammatical conditions
- agreement interference within ungrammatical conditions

Do the estimates match expectations? Check this by computing the condition means and checking that the estimates from the models match the relevant differences between conditions or clusters of conditions.



Part III

Advanced models with Stan



10

Introduction to the probabilistic programming language Stan

Stan is a probabilistic programming language for statistical inference written in C++ that can be accessed through several interfaces (e.g., R, Python, Matlab, etc.). Stan uses an advanced dynamic Hamiltonian Monte Carlo algorithm (Betancourt 2016) based on a variant of the No-U-Turn sampler (known as NUTS: Hoffman and Gelman 2014), which is, in general, more efficient than the traditional Gibbs sampler used in other probabilistic languages such as (Win)BUGS (Lunn et al. 2000) and JAGS (Plummer 2016). In this part of the book, we will focus on the package `rstan` (Guo, Gabry, and Goodrich 2019) that integrates Stan (Carpenter et al. 2017) with R (R Core Team 2019).

In order to understand how to fit a model in Stan and the difficulties we might face, a minimal understanding of the Stan sampling algorithm is needed. Stan takes advantage of the fact that the *shape* of the posterior distribution is perfectly determined by the priors and the likelihood we have defined, that is the unnormalized posterior distribution, the upper part of the Bayes rule (abbreviated as *up*). This is because the denominator, or marginal likelihood, “only” constitutes a normalizing constant:

$$p(\Theta|y) = \frac{p(y|\Theta) \cdot p(\Theta)}{p(y)} \quad (10.1)$$

$$up(\Theta|y) = p(y|\Theta) \cdot p(\Theta) \quad (10.2)$$

Thus the unnormalized posterior is proportional to the posterior distribution:

$$up(\Theta|y) \propto p(\Theta|y) \quad (10.3)$$

(The notation $up(\cdot)$ that we are using here is not standard in statistics; we are using it only for pedagogical convenience.)

Stan sampler uses Hamiltonian dynamics and treats the vector of parameters, Θ (that could range from a vector containing a couple of parameters, e.g., $\langle \mu, \sigma \rangle$, to a vector of hundreds of parameters in hierarchical models), as the position of a frictionless particle that glides on the *negative logarithm of the unnormalized posterior*. That means that high probability places are valleys and low probability places are peaks in this space.¹ However, Stan doesn't just let the particle glide until the bottom of this space. If we let that happen, we would find the mode of the posterior distribution, rather than samples. Stan uses a complex algorithm to determine the weight of the particle and the momentum that we apply to it, as well as when to stop the particle trajectory to take a sample. Because we need to know the speed of this particle, Stan needs to be able to calculate the derivative of the log unnormalized posterior with respect to the parameters (recall that speed is the first derivative of position). This means that if the parameter space is differentiable (is relatively smooth, and does not have any break or angle) and if the parameters of the Stan algorithm are well adjusted—as should happen in the warm-up period—these samples are going to represent samples of the true posterior distribution. Bear in mind that the geometry of the posterior has a big influence on whether the algorithm will converge (fast) or not: If the space is very flat, because there isn't much data and the priors are not informative, then the particle may need to glide for a long time before it gets to a high probability area; if there are several valleys (multimodality) the particle may never leave the vicinity of one of them; and if the space is funnel shaped, the particle may never explore the funnel. One of the reasons for the difficulties in exploring complicated spaces is that the continuous path of the “particle” is discretized and divided into steps, and the *step size* is optimized for the entire posterior space. In spaces that are too complex, such as a funnel, a step size might be too small to explore the wide part of the funnel, but too large to explore the narrow part; we will deal with this problem in section 11.1.2.

¹In the specific case of a model with two parameters, e.g., $\langle \mu, \sigma \rangle$, the physical analogy works quite well: The $\langle x, y \rangle$ coordinates of the particle would be determined by $\langle \mu, \sigma \rangle$, and its z coordinate would be established by the height of the unnormalized posterior.

Although our following example assumes a vector of two parameters and thus a simple geometry, real world examples can easily have hundreds of parameters defining an unnormalized posterior space with hundreds of dimensions.

One question that might arise here is the following: Given that we already know the shape of the posterior, why do we need samples? After all, the posterior is just the unnormalized posterior multiplied by some number, the normalizing constant.

To make this discussion concrete, let's say that we have a subject that participates in a memory test, and in each trial we get a noisy score from their true working memory score. We assume that at each trial, the score is elicited with normally distributed noise. If we want to estimate the score and how much the noise makes it vary from trial to trial, we are assuming a normal likelihood and we want to estimate its mean and standard deviation.

We will use simulated data produced by a normal distribution with a true mean of 3 and a true standard deviation of 10:

```
Y <- rnorm(n = 100, mean = 3, sd = 10)
head(Y)
```

```
## [1] 23.65 24.94 4.57 -5.64 4.65 -3.53
```

As always, given our prior knowledge, we decide on priors. In this case, we use a log-normal prior for the standard deviation, σ , since it can only be positive, but except for that, the prior distributions are quite arbitrary in this example.

$$\begin{aligned}\mu &\sim \text{Normal}(0, 20) \\ \sigma &\sim \text{LogNormal}(3, 1)\end{aligned}\tag{10.4}$$

The unnormalized posterior will be the product of the likelihood of each data point times the prior for each parameter:

$$up(\mu, \sigma | y) = \prod_n^{100} Normal(y_n | \mu, \sigma) \cdot Normal(\mu | 0, 20) \cdot LogNormal(\sigma | 3, 1) \quad (10.5)$$

where $y = 23.647, 24.936, \dots$

We can also define the unnormalized posterior, $up(\cdot)$, as a function in R:

```
up <- function(y, mu, sigma) {
  dnorm(x = mu, mean = 0, sd = 20) *
  dlnorm(x = sigma, mean = 3, sd = 1) *
  prod(dnorm(x = y, mean = mu, sd = sigma))
}
```

For example, if we want to know the unnormalized posterior density for the vector of parameters $\langle \mu, \sigma \rangle = \langle 0, 5 \rangle$, we do the following:

```
up(y = Y, mu = 0, sigma = 5)
## [1] 5.59e-201
```

The shape of the unnormalized posterior density is completely defined and it will look like Figure 10.1.

Why is the shape of the unnormalized posterior density not enough? The main reason is that unless we already know which probability distribution we are dealing with (e.g., normal, Bernoulli, etc.) or we can easily integrate it (which only happens in a handful of cases) we cannot do much with the analytical form of the unnormalized posterior: We cannot calculate credible intervals, or know how likely it is that the true score is above or below zero, and even the mean of the posterior is impossible to calculate. This is because the unnormalized posterior distribution represents the general shape of the posterior distribution. With just the shape of an unknown distribution, we can only answer the following question: What is the most (or least) likely value of the vector of parameters? We can answer this question by searching for the highest (or lowest) place in that shape. This leads us to the estimate of the maximum a posteriori (MAP), which is the Bayesian counterpart of the maximum likelihood estimate

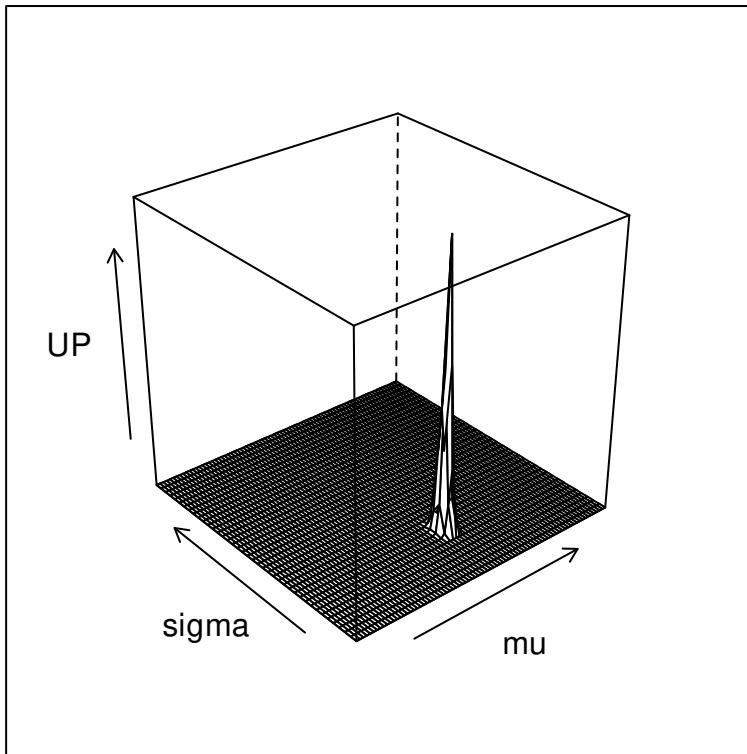


FIGURE 10.1: The unnormalized posterior defined by Equation (10.5)

(MLE). (However, if we can recognize the shape as a distribution, we are in a different situation. In that case, we might know already the formulas for the expectation, variance, etc. This is what we did in chapter 2, but it is an unusual situation in realistic analyses.) As we mentioned before, if we want to get posterior density values, we need the denominator of the Bayes rule (or marginal likelihood), $p(y)$, which requires integrating the unnormalized posterior. Even this is not too useful if we want to communicate findings, almost every summary statistic require us to solve more integrals, and except for a handful of cases, these integrals might not have an analytical solution.

As we mentioned before, the only summary that we can get with an unnormalized posterior shape (that we don't recognize as a familiar distribution) is its mode (or the MAP: the highest point in Figure 10.1, or the lowest point of the negative log unnormalized log posterior). However, even

calculating the mode is not always trivial. In simple cases as this one, one can calculate it analytically; but in more complex cases relatively complicated algorithms are needed.

If we want to be able to calculate summary statistics of the posterior distribution (mean, quantiles, etc.), we are going to need samples from this distribution. This is because with enough samples of a probability distribution, we can achieve very good approximations of summary statistics. Stan will take care of returning samples from the posterior distribution, if the log unnormalized posterior distribution is differentiable and can be expressed as follows:²

$$\log(up(\Theta|y)) = \sum_n \log(p(y_n|\Theta)) + \sum_q \log(p(\Theta_q)) \quad (10.6)$$

where n indicates each data point and q each parameter. In our case, this corresponds to the following:

$$\begin{aligned} \log(up(\mu, \sigma|y)) = & \sum_n^{100} \log(Normal(y_n|\mu, \sigma)) + \log(Normal(\mu|0, 20)) \\ & + \log(LogNormal(\sigma|3, 1)) \end{aligned} \quad (10.7)$$

In the following sections, we'll see how we can implement this model and many others in Stan.

10.1 Stan syntax

A Stan program is usually saved as a `.stan` file and accessed through R (or other interfaces) and it is organized into a sequence of optional and obligatory blocks, which must be written in order. The Stan language is different from R and it is loosely based on C++; one important aspect to pay attention to is that every statement ends in a semi-colon, `;`. Blocks (`{}`) do not

²Incidentally, this `log(up())` is the variable `target` in a Stan model and `lp__` in its output; see Box 10.1.

end in semi-colons. Some functions in Stan are written in the same way as in R (e.g., `mean`, `sum`, `max`, `min`). But some are different; when in doubt, Stan documentation (<https://mc-stan.org/users/documentation/>) can be extremely helpful. In addition, the package `rstan` provides the function `lookup()` to look up for translations of functions. For example, in 4.3, we saw that the R function `plogis()` is needed to convert from log-odds to probability space. If we need it in a Stan program, we can look for it in the following way:

```
lookup(plogis)
```

##	StanFunction	Arguments	ReturnType
## 227	<code>inv_logit</code>	(T x)	R
## 260	<code>log_inv_logit</code>	(T x)	R
## 261	<code>logistic_cdf</code> (reals y, reals mu, reals sigma)		real
## 262	<code>logistic_lccdf</code> (reals y , reals mu, reals sigma)		real
## 263	<code>logistic_lcdf</code> (reals y , reals mu, reals sigma)		real

There are three columns in the output of this call. The first one indicates Stan function names, the second one their arguments with their type, and the third one the type they return. Unlike R, Stan is strict with the type of the variables.³ If we need to decide on the function, we'll probably need to go to the Stan documentation, and figure out which is the one that matches our specific needs (it would be `inv_logit()`).

Another important difference with R is that every variable needs to be declared at the beginning of a block with its type (real, integer, vector, matrix, etc.). The next two sections exemplify these details through basic Stan programs.

³In these output, there are some types that are new to the R user (but they are also used in C++): `reals` indicates that any of `real`, `real[]`, `vector`, or `row_vector`. A return type `R` with an input type `T` indicates that the type of the output of the function is the same as type of the argument.

10.2 A first simple example with Stan: Normal likelihood

Let's fit a Stan model to estimate the simple example given at the introduction of this chapter, where we simulate data from a normal distribution with a true mean of 3 and a true standard deviation of 10:

```
Y <- rnorm(n = 100, mean = 3, sd = 10)
```

As we said before Stan code is organized in blocks. The first block indicates what constitutes “data” for the model:

```
data {
    int<lower = 1> N; // Total number of trials
    vector[N] y; // Score in each trial
}
```

The variable of type `int` (integer) represents the number of trials. In addition to the type, some constraints can be indicated with `lower` and `upper`. In this case, `N` can't be smaller than 1. These constraints serve as a sanity check; if they are not satisfied, we get an error and the model won't run. The data are stored in a vector of length `N`, unlike R, vectors (and matrices and arrays) need to be defined with their dimensions. Comments are indicated with `//` rather than `#`.

The next block indicates the parameters of the model:

```
parameters {
    real mu;
    real<lower = 0> sigma;
}
```

The two parameters are real numbers, and `sigma` is constrained to be positive.

Finally, we indicate the prior distributions and likelihood functions in the model block:

```
model {
    // Priors:
```

```

target += normal_lpdf(mu | 0, 20);
target += lognormal_lpdf(sigma | 3, 1);
// Likelihood:
for(i in 1:N)
    target += normal_lpdf(y[i] | mu, sigma);
}

```

The variable `target` is a reserved word in Stan; every statement with `target +=` adds terms to the unnormalized *log* posterior probability. We do this because adding to the unnormalized *log* posterior means to multiply a term in the numerator of the unnormalized posterior. As we explained before, Stan uses the shape of the unnormalized posterior to sample from the actual posterior distribution. See Box 10.1 for a more detailed explanation.

Box 10.1. What does `target` do?

We can exemplify how `target` works with one hypothetical iteration of the sampler.

In every iteration where the sampler explores the posterior space, `mu` and `sigma` acquire different values (this is where Stan algorithm stops the movement of the particle in the Hamiltonian space). Say that in an iteration, `mu = 3.595` and `sigma = 10.588`. Then the following happens in the model block:

1. At the beginning of the iteration, `target` is zero.
2. The transformations that the sampler *automatically* does are taken into account. In our case, although `sigma` is constrained to be positive in our model, inside Stan's sampler it is transformed to an “unconstrained” space amenable to Hamiltonian Monte Carlo. That is, Stan samples from an auxiliary parameter that ranges from minus infinity to infinity, which is equivalent to `log(sigma)`. This auxiliary parameter is then exponentiated, when it is incorporated into our model. Because of the mismatch between the constrained parameter space that we defined

and the unconstrained space that its converted to by Stan, an adjustment to the unnormalized posterior is required and added *automatically*. The reasons for this requirement are somewhat complex and will be discussed in section 12. In this particular case, this adjustment (which is the log absolute value of the Jacobian determinant), is equivalent to adding `log(sigma) = 2.36` to `target`.

3. After `target += normal_lpdf(mu | 0, 20);` the log of the density of $Normal(0, 20)$ is evaluated at a given sample of `mu` (specifically 3.595) and this is added to `target`. In R, this would be `dnorm(x = 3.595, mean = 0, sd = 20, log = TRUE)`, which is equal to -3.931. Thus, `target` should be $-3.931 + 2.36 = -1.571$.

4. After `target += lognormal_lpdf(sigma | 3, 1);` we add the log of the density of $LogNormal(3, 1)$ evaluated at 10.588 to the previous value of the target. In R, this would be `dlnorm(x = 10.588, mean = 3, sd = 1, log = TRUE)`, which is equal to -3.484. Thus, `target` should be updated to $-1.571 + -3.484 = -5.055$.

5. After each iteration of the for-loop in the model block, we add to the `target` the log density of $Normal(3.595, 10.588)$ evaluated at each of the values of `Y`. In R, this would be to add `sum(dnorm(Y, 3.595, 10.588, log = TRUE))` (which is equal to -368.771) to the current value of `target` $-5.055 + -368.771 = -373.826$.

This means that for the coordinates `<mu = 3.595, sigma = 10.588>`, the height of the unnormalized posterior would be the value `exp(target) = exp(-373.826) = 4.46 \times 10^{-163}`. Incidentally, the value of `target` is returned as `lp_` (log probability) in an object storing a fit model with Stan.

It is possible to expose the value of `target`, by printing `target()` inside a Stan model. The value of `target` after each iteration is named `lp_` in the Stan object. This can be useful to troubleshoot a problematic model.

We didn't use curly brackets with the for-loop; this is a common practice

if the for-loop has only one line, but brackets can be added and are obligatory if the for-loop spans several lines.

It's also possible to avoid the for-loop since many functions are vectorized in Stan:

```
model {
    // Priors:
    target += normal_lpdf(mu | 0, 20);
    target += lognormal_lpdf(sigma | 3, 1);
    // Likelihood:
    target += normal_lpdf(y | mu, sigma);
}
```

The for-loop and vectorized versions are giving us the exact same output: The for-loop version evaluated the log-likelihood at each value of y and added it to `target`. The vectorized version does not create a vector of log-likelihoods, instead it sums the log-likelihood evaluated at each element of y and then it adds that to `target`.

The complete model that we will fit looks like this:

```
data {
    int<lower = 1> N; // Total number of trials
    vector[N] y; // Score in each trial
}
parameters {
    real mu;
    real<lower = 0> sigma;
}
model {
    // Priors:
    target += normal_lpdf(mu | 0, 20);
    target += lognormal_lpdf(sigma | 3, 1);
    // Likelihood:
    target += normal_lpdf(y | mu, sigma);
}
```

You can save the above code as `normal.stan`. Alternatively, you can use the version stored in the package `bcogsci`. (`?stan-normal` provides some docu-

mentation for the model.) You can access the code of the models of this book by using `system.file("stan_models", "name_of_the_model.stan", package = "bcogsci")`.

```
normal <- system.file("stan_models",
                      "normal.stan",
                      package = "bcogsci")
```

This command just points to a text file that the package `bcogsci` stored in your system. You can open it to read the code (with any text editor, or `readLines()` in R). You'll need to compile this code and run it with `stan()`.

Stan requires the data to be in a list object in R. Below we fit the model with the default number of chains and iterations.

```
lst_score_data <- list(y = Y, N = length(Y))
# Fit the model with the default values of number of
# chains and iterations: chains = 4, iter = 2000
fit_score <- stan(
  file = normal,
  data = lst_score_data
)
# alternatively:
# stan(file = "normal.stan", data = lst_score_data)
```

We can inspect how well the chains mixed in Figure 10.2. The chains for each parameter should look like a “fat hairy caterpillar” (Lunn et al. 2012); see section 3.1.1.1.2 for a brief discussion of convergence.

```
traceplot(fit_score, pars = c("mu", "sigma"))
```

We can see a summary of the posterior by either printing out the model fit, or by plotting it. The summary displayed by the function `print` includes means, standard deviations (`sd`), quantiles, Monte Carlo standard errors for the mean of the posterior (`se_mean`), split Rhats, and effective sample sizes (`n_eff`). The summaries are computed after removing the warmup and merging together all chains. The `se_mean` is unrelated to the `se` of an es-

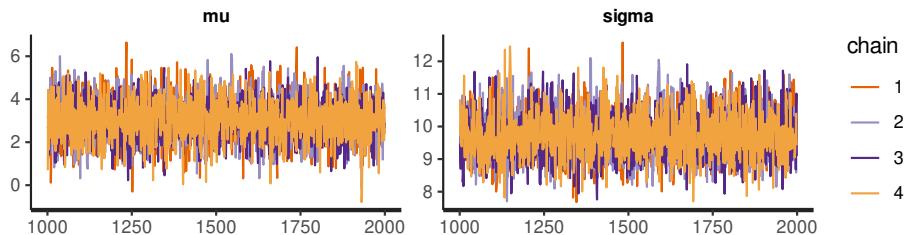


FIGURE 10.2: Traceplots of `mu` and `sigma` from the model `fit_score`.

timate in the parallel frequentist model. Similarly to a large effective sample size, small Monte Carlo standard errors indicate an “efficient” sampling procedure: with a large value of `n_eff` and a small value for `se_mean` we can be relatively sure of the reliability of the mean of the posterior. However, what constitutes a large or small `se_mean` is harder to define (see Vehtari, Gelman, et al. 2019 for a more extensive discussion).⁴

```
print(fit_score, pars = c("mu", "sigma"))
```

```
## Inference for Stan model: normal.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##          mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
## mu     3.02    0.02 0.97 1.13 2.36 3.02  3.65  4.97  3400     1
## sigma  9.73    0.01 0.71 8.45 9.23 9.70 10.19 11.22  3213     1
##
## Samples were drawn using NUTS(diag_e) at Thu Nov 18 11:56:21 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

After transforming the `stanfit` object into a data frame, it’s possible to provide summary plots as the one shown in 10.3. The package `bayesplot` (Gabry and Mahr 2019) is a wrapper around `ggplot2` (Wickham, Chang,

⁴We simplify the output of `print` in the text after this call, by actually calling `summary(fit, pars = pars, probs = c(0.025, 0.975))$summary`.

et al. 2019) and has several convenient functions to plot the samples. Bayesplot functions for posterior summaries start with `mcmc_`:

```
df_fit_score <- as.data.frame(fit_score)
mcmc_hist(df_fit_score, pars = c("mu", "sigma"))
```

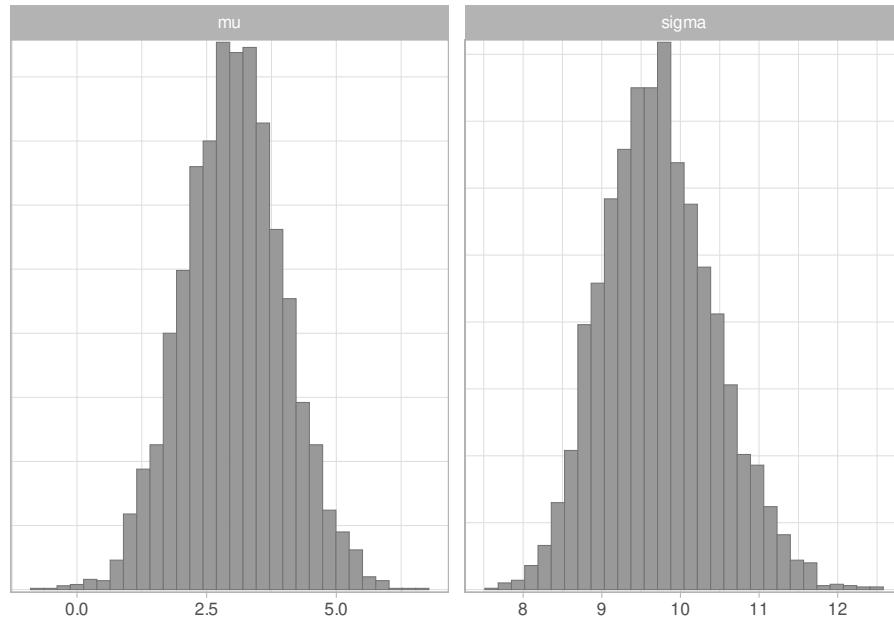


FIGURE 10.3: Histograms of the samples of the posterior distributions of `mu` and `sigma` from the model `fit_score`.

There are also several ways to get the samples for other summaries or customized plots, depending on whether we want a list, a data frame, or an array.

```
# extract from rstan is sometimes overwritten by
# a tidyverse version, we make sure that it's right one:
rstan::extract(fit_score) %>%
  str()

## List of 3
## $ mu    : num [1:4000(1d)] 3.59 1.72 2.83 3.63 2.36 ...
```

```

##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ iterations: NULL
##   $ sigma: num [1:4000(1d)] 10.59 9.4 9.14 9.64 9.45 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ iterations: NULL
##   $ lp__: num [1:4000(1d)] -374 -374 -373 -373 -373 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ iterations: NULL

as.data.frame(fit_score) %>%
  str(list.len = 5)

## 'data.frame':    4000 obs. of  3 variables:
## $ mu     : num  1.2 1.27 1.78 1.93 2.04 ...
## $ sigma: num  9.37 9.48 9.76 9.61 9.83 ...
## $ lp__  : num  -375 -374 -374 -373 -373 ...

as.array(fit_score) %>%
  str()

## num [1:1000, 1:4, 1:3] 1.2 1.27 1.78 1.93 2.04 ...
## - attr(*, "dimnames")=List of 3
##   ..$ iterations: NULL
##   ..$ chains    : chr [1:4] "chain:1" "chain:2" "chain:3" "chain:4"
##   ..$ parameters: chr [1:3] "mu" "sigma" "lp__"

```

Box 10.2. An alternative R interface to Stan: cmdstanr

For the time being, there are two major nuisances with `rstan`, (i) the R code interfaces directly with C++ creating installation problems in many systems, (ii) `rstan` releases lag behind Stan language releases considerably preventing the user from taking advantage from the latest features of Stan. The package `cmdstanr` (<https://mc-stan.org/cmdstanr/>) is a lightweight interface to Stan for R that solves these problems. The downside is that being lightweight some functionality of `rstan` is (still) lost, such as looking up functions with `lookup`,

exposing functions with `expose_stan_function`, as well as using the fitted model with the `bridgesampling` package to generate Bayes factors. Furthermore, the package `cmdstanr` is currently under development and the application programming interface (API) might still change. However, the user interested in an easy (and painless) installation and the latest features of Stan might find it useful.

Once `cmdstanr` is installed, we can use it as follows:

First create a new `CmdStanModel` object from a file containing a Stan program using `cmdstan_model()`

```
normal <- system.file("stan_models",
                      "normal.stan",
                      package = "bcogsci")
normal_mod <- cmdstan_model(normal)
```

The object `normal_mod` is an R6 reference object (<https://r6.r-lib.org/>). This class of object behaves similarly to objects in object oriented programming languages, such as python. Methods are accessed using `$` (rather than `.` as in python).

To sample, use the `$sample()` method. The `data` argument accepts a list (as we used in `stan()` from `rstan`). However, many of the arguments of `$sample` have different names than the ones used in `stan()` from the `rstan` package:

```
lst_score_data <- list(y = Y, N = length(Y))
fit_normal_cmd <- normal_mod$sample(
  data = lst_score_data,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  iter_warmup = 1000,
  iter_sampling = 1000)
```

To show the posterior summary, access the method `$summary()` of the object `fit_normal_cmd`.

```
fit_normal_cmd$summary()
```

Access the samples of `fit_normal_cmd` using `$draws()`.

```
fit_normal_cmd$draws(variables = "mu")
```

The vignette of <https://mc-stan.org/cmdstanr/> shows more use cases, and how the samples can be transformed into other formats (data frame, matrix, etc) together with the package `posterior` (<https://mc-stan.org/cmdstanr/>).

10.3 Another simple example: Cloze probability with Stan with the Binomial likelihood

Let's fit a Stan model (`binomial_cloze.stan`) to estimate the cloze probability of a word given its context: that is, what is the probability of an upcoming word given its previous context; the model that is detailed in [2.2](#) and was fit in [3.1](#). We want to estimate the cloze probability of “*umbrella*”, θ , given the following data: “*umbrella*” was answered 80 out of 100 trials. We assume a Binomial distribution as the likelihood function, and $Beta(a = 4, b = 4)$ as a prior distribution for the cloze probability.

```
data {
  int<lower = 1> N; // Total number of answers
  int<lower = 0, upper = N> k; // Number of times umbrella was answered
}
parameters {
  // theta is a probability, it has to be constrained between 0 and 1
  real<lower = 0, upper = 1> theta;
}
model {
  // Prior on theta:
```

```

target += beta_lpdf(theta | 4, 4);
// Likelihood:
target += binomial_lpmf(k | N, theta);
}

```

There is only one parameter in this model, cloze probability represented with the parameter `theta`, which is a real number constrained between 0 and 1. Another difference between this and the previous example is that the likelihood function ends with `_lpmf` rather than with `_lpdf`. This is because Stan differentiates between distributions of continuous variables, i.e., probability density functions (PDF), and distributions of discrete variables, i.e., probability mass functions (PMF).

```

lst_cloze_data <- list(k = 80, N = 100)
binomial_cloze <- system.file("stan_models",
                               "binomial_cloze.stan",
                               package = "bcogsci")
fit_cloze <- stan(
  file = binomial_cloze,
  data = lst_cloze_data
)

```

We print the summary of the posterior distribution of θ below, and we show its posterior distribution graphically (see Figure 10.4).

```

print(fit_cloze, pars = c("theta"))

##          mean 2.5% 97.5% n_eff Rhat
## theta 0.78  0.7  0.85  1396     1

df_fit_cloze <- as.data.frame(fit_cloze)
mcmc_dens(df_fit_cloze, pars = "theta") +
  geom_vline(xintercept = mean(df_fit_cloze$theta))

```

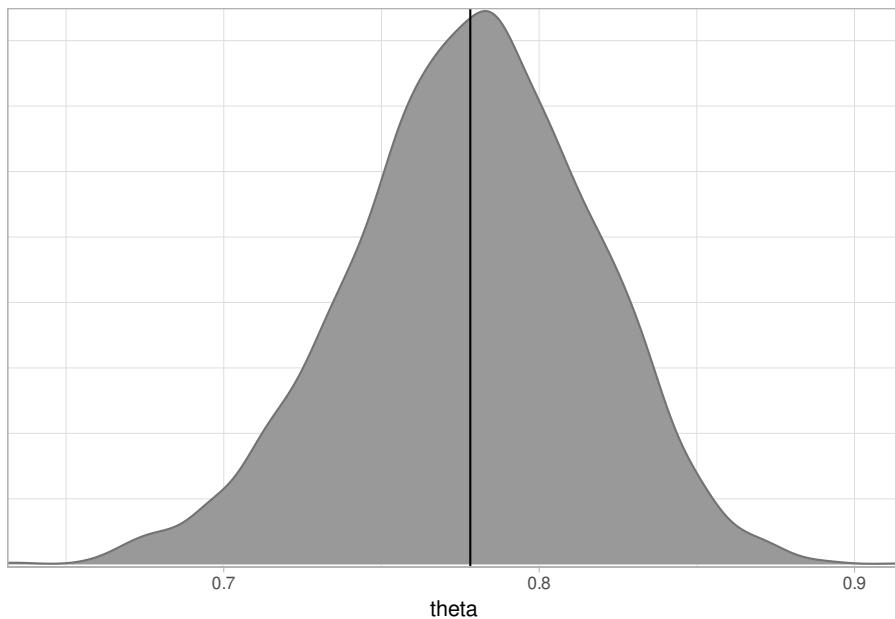


FIGURE 10.4: Posterior distribution of the cloze probability of umbrella; parameter θ .

10.4 Regression models in Stan

In the following sections, we will revisit and expand on some of the examples that we fit with `brms` in chapter 4.

10.4.1 A first linear regression in Stan: Does attentional load affect pupil size?

As in section 4.1, we focus on the effect of cognitive load on one subject's pupil size with a subset of the data of Wahn et al. (2016). We use the following likelihood and priors. For details about our decision on priors and likelihood, see 4.1.

$$\begin{aligned}
 p_size_n &\sim Normal(\alpha + c_load_n \cdot \beta, \sigma) \\
 \alpha &\sim Normal(1000, 500) \\
 \beta &\sim Normal(0, 100) \\
 \sigma &\sim Normal_+(0, 1000)
 \end{aligned} \tag{10.8}$$

The Stan model `pupil_model.stan` follows:

```
data {
    int<lower=1> N;
    vector[N] p_size;
    vector[N] c_load;
}
parameters {
    real alpha;
    real beta;
    real<lower = 0> sigma;
}
model {
    // priors:
    target += normal_lpdf(alpha | 1000, 500);
    target += normal_lpdf(beta | 0, 100);
    target += normal_lpdf(sigma | 0, 1000)
        - normal_lccdf(0 | 0, 1000);
    // likelihood
    target += normal_lpdf(p_size | alpha + c_load * beta, sigma);
}
```

Because we are fitting a regression, we use the location (μ) of the likelihood function to regress `p_size` with the following equation `alpha + c_load * beta`, where both `p_size` and `c_load` are vectors defined in the data block. The following line accumulates the log-likelihood of every observation:

```
target += normal_lpdf(p_size | alpha + c_load * beta, sigma);
```

This is equivalent to and slightly faster than the following lines:

```
for(n in 1:N)
    target += normal_lpdf(p_size[n] | alpha + c_load[n] * beta, sigma);
```

A statement that requires some explanation is the following:

```
target += normal_lpdf(sigma | 0, 1000)
    - normal_lccdf(0 | 0, 1000);
```

As in our original example in 4.1, we are assuming a truncated normal dis-

tribution as a prior for σ . Not only are we setting a lower boundary to the parameter with `lower = 0`, but we are also “correcting” its prior distribution by subtracting `normal_lccdf(0 | 0, 1000)`, where `lccdf` stands for log complement of a cumulative distribution function. Once we add a lower boundary, the probability mass under *half* of the “regular” normal distribution should be one, that is, when we integrate from zero (rather than from minus infinity) to infinity. As we saw in Box 4.1, we need to normalize the PDF by dividing it by the difference of its CDF evaluated in the new boundaries ($a = 0$ and $b = \infty$ in our case):

$$f_{[a,b]}(x) = \frac{f(x)}{F(b) - F(a)} \quad (10.9)$$

This equation in log-space is:

$$\log(f_{[a,b]}(x)) = \log(f(x)) - \log(F(b) - F(a)) \quad (10.10)$$

In Stan $\log(f(x))$ corresponds to `normal_lpdf(x | ...)`, and $\log(F(x))$ to `normal_lcdf(x|...)`. Because in our example $b = \infty$, $F(b) = 1$, we are dealing with the complement of the log CDF evaluated at $a = 0$, $\log(1 - F(0))$, that is why we use `normal_lccdf(0 | ...)`.

To be able to fit the model, Stan requires the data to be input as a list: First, we load the data and center the dependent variable in a data frame and then we create a list.

```
df_pupil <- df_pupil %>%
  mutate(c_load = load - mean(load))
```

```
ls_pupil <- list(
  p_size = df_pupil$p_size,
  c_load = df_pupil$c_load,
  N = nrow(df_pupil)
)
pupil_model <- system.file("stan_models",
  "pupil_model.stan",
  package = "bcogsci")
```

```
)
fit_pupil <- stan(pupil_model,
  data = ls_pupil
)
```

Check the traceplots in 10.5.

```
traceplot(fit_pupil, pars = c("alpha", "beta", "sigma"))
```

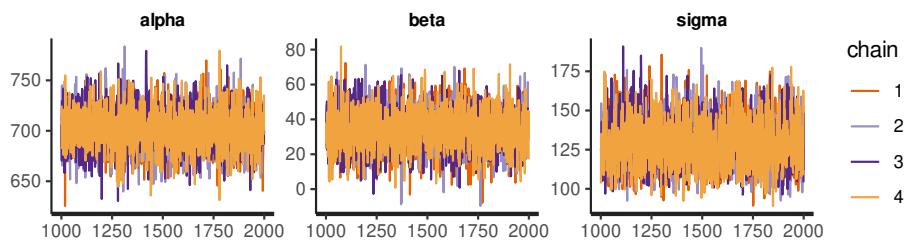


FIGURE 10.5: Traceplots of `alpha`, `beta`, and `sigma` from the model `fit_pupil`.

Examine some summaries of the marginal posterior distributions of the parameters of interest:

```
print(fit_pupil, pars = c("alpha", "beta", "sigma"))
```

```
##          mean 2.5% 97.5% n_eff Rhat
## alpha    701   660  742.2  3452     1
## beta     34    11   57.3  3647     1
## sigma   129   103  161.2  3300     1
```

Plot the posterior distributions in 10.6.

```
df_fit_pupil <- as.data.frame(fit_pupil)
mcmc_hist(fit_pupil, pars = c("alpha", "beta", "sigma"))
```

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.

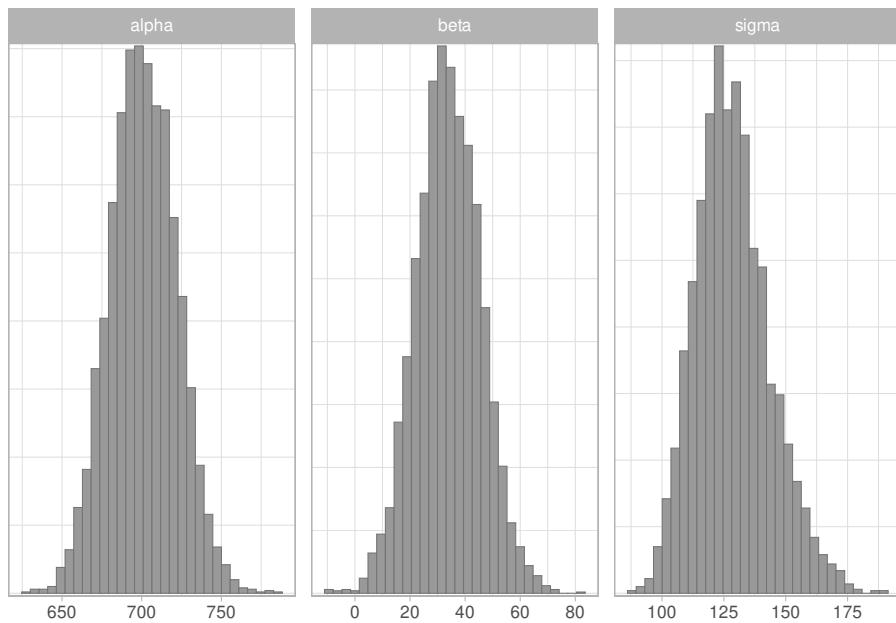


FIGURE 10.6: Histograms of the posterior samples of `alpha`, `beta`, and `sigma` from the model `fit_pupil`.

```
# We are using df_fit_pupil and not the "raw" Stanfit object.
mean(df_fit_pupil$beta > 0)
```

```
## [1] 0.998
```

If we want to generate prior or posterior predictive distributions, we can either create our own functions in R with the `purrr` function `map_dfr` (or a for-loop) as we did in section 4.2 with the function `lognormal_model_pred()`. Alternatively, we can use the generated `quantities` block in our model:

```
data {
  int<lower = 1> N;
  vector[N] c_load;
  int<lower= 0, upper = 1> onlyprior;
  vector[N] p_size;
}
```

```

parameters {
    real alpha;
    real beta;
    real<lower = 0> sigma;
}
model {
    // priors including all constants
    target += normal_lpdf(alpha | 1000, 500);
    target += normal_lpdf(beta | 0, 100);
    target += normal_lpdf(sigma | 0, 1000)
        - normal_lccdf(0 | 0, 1000);
    if (!onlyprior)
        target += normal_lpdf(p_size | alpha + c_load * beta, sigma);
}
generated quantities {
    real p_size_pred[N];
    p_size_pred = normal_rng(alpha + c_load * beta, sigma);
}

```

For most of the probability functions, there is a matching pseudorandom number generator (PRNG) with the suffix `_rng`. Here we are using the vectorized function `normal_rng`. Once `p_size_pred` is declared as an array of size `N`, the following statement generates N predictions (for each iteration of the sampler):

```
p_size_pred = normal_rng(alpha + c_load * beta, sigma);
```

At the moment not all the PRNG are vectorized, but the ones that are only allow for arrays and, confusingly enough, not vectors. We define arrays by indicating a type, and then between brackets, the length of each dimension. For example to define an array of real numbers with three dimension of length 6, 7, and 10 we write `real var[6, 7, 10]`. Vectors and matrices are also valid types for an array. See Box 10.3 for more about the difference between arrays and vectors, and other algebra types. We also included a data variable called `onlyprior`, this is an integer that can only be set to 1 (TRUE) or 0 (FALSE). When `onlyprior = 1`, the likelihood is omitted from the model, `p_size` is ignored, and `p_size_pred` is the prior predictive distribution. When `onlyprior = 0`, the likelihood is incorporated in the

model (as it is in the original code `pupil_model.stan`) using `p_size`, and `p_size_pred` is the posterior predictive distribution.

If we want posterior predictive distributions, we fit the model to the data and set `onlyprior = 0`, if we want prior predictive distributions, we sample from the priors and set `onlyprior = 0`. Then we use `bayesplot` functions to visualize predictive checks.

For posterior predictive checks, we would do the following:

```
ls_pupil <- list(  
  onlyprior = 0,  
  p_size = df_pupil$p_size,  
  c_load = df_pupil$c_load,  
  N = nrow(df_pupil)  
)  
pupil_gen <- system.file("stan_models",  
  "pupil_gen.stan",  
  package = "bcogsci")  
fit_pupil <- stan(file = pupil_gen, data = ls_pupil)
```

Store the predicted pupil sizes in `yrep_pupil`. This variable contains an $N_{samples} \times N_{observations}$ matrix, that is, each row of the matrix is a draw from the posterior predictive distribution, i.e., a vector with one element for each of the data points in `y`.

```
yrep_pupil <- extract(fit_pupil)$p_size_pred  
dim(yrep_pupil)
```

```
## [1] 4000 41
```

Predictive checks functions in `bayesplot` (starting with `ppc_`) require a vector with the observations in the first argument and a matrix with the predictive distribution as its second argument. As an example, in Figure 10.7 we use an overlay of densities and we draw only 50 elements (that is 50 predicted datasets).

```
ppc_dens_overlay(df_pupil$p_size, yrep = yrep_pupil[1:50, ])
```

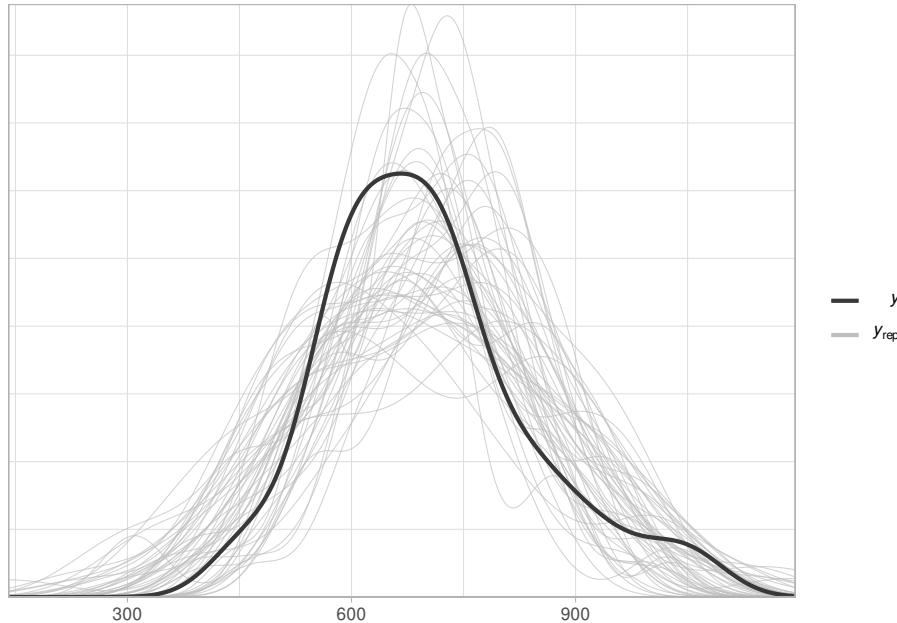


FIGURE 10.7: Posterior predictive check showing 50 predicted density plots from the model `fit_pupil` against the observed data.

For prior predictive distributions, we simply set `onlyprior = 1`. The observations (`p_size`) are ignored by the model, but are required by the data block in Stan. If we haven't collected data yet, we could include a vector of zeros.

```
ls_pupil_prior <- list(  
  onlyprior = 1,  
  p_size = df_pupil$p_size,  
  # or: p_size = rep(0, nrow(df_pupil)),  
  c_load = df_pupil$c_load,  
  N = nrow(df_pupil)  
)  
prior_pupil <- stan(pupil_gen,  
  data = ls_pupil_prior,
```

```
control = list(adapt_delta = 0.90)
)
```

We need to increase the `adapt_delta` parameter's default value from 0.80 to 0.90 to simulate the data to avoid divergent transitions. It is important to highlight that we cannot safely ignore the warnings of the last model, even if we are not fitting data. This is so because in practice one is still sampling a density using Hamiltonian Monte Carlo, and thus the prior sampling process can break in the same ways as the posterior sampling process. Predictive checks in `bayesplot`, as the one shown in Figure 10.8, require (at least for now), an argument `y` with data. If we haven't collected data yet, we can, for example, use it to provide plausible or implausible values that we want to compare to the prior predictive realizations. In the following example, we set `y` to be a uniform distribution that ranges between 0 and 1000; this is the distribution density drawn with the darker color in the plot. (Alternatively, we can use `ggplot` and manually build the plot.) The uniformly distributed data (the dark line) don't look particularly uniform here; this is because we are simulating only 41 data points.

```
yrep_prior_pupil <- extract(prior_pupil)$p_size_pred
ppc_dens_overlay(
  runif(ls_pupil_prior$N, 0, 1000),
  yrep_prior_pupil[1:50, ]
)
```

Box 10.3. Matrix, vector, or array in Stan?

Stan contains three basic linear algebra types, `vector`, `row_vector`, and `matrix`. But Stan also allows for building arrays of any dimension from any type of element (integer, real, etc.). This means that there are several ways to define one-dimensional N-sized containers of real numbers,

```
real a[N];
vector[N] a;
```

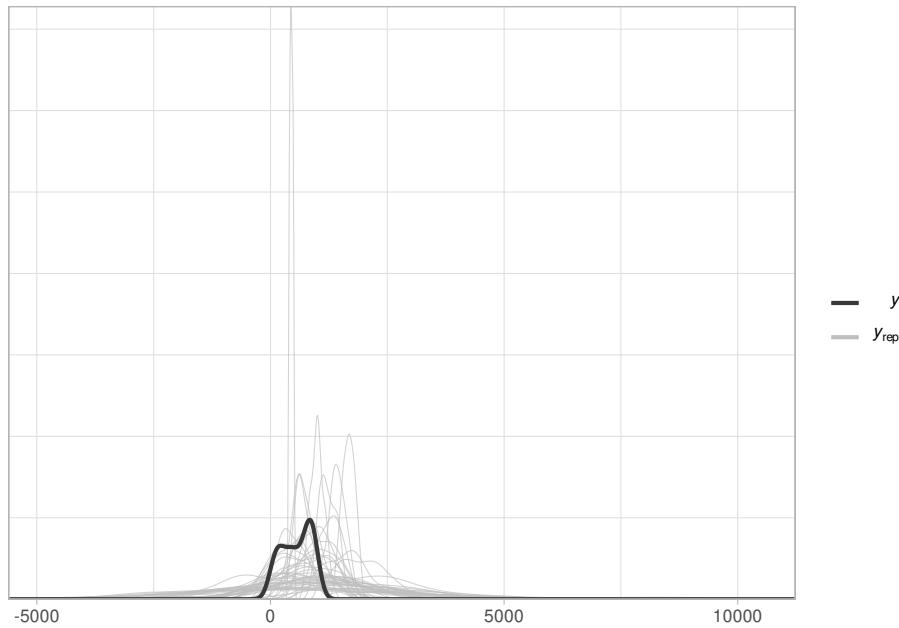


FIGURE 10.8: Prior predictive check showing 50 predicted density plots from the model `fit_pupil`; the darker density plot depicts a uniform distribution (from 0 to 1000) of synthetic observations.

```
row_vector[N] a;
```

as well as, two-dimensional $N_1 \times N_2$ -sized containers of real numbers:

```
real m[N1, N2];
matrix[N1, N2] m;
vector[N2] b[N1];
row_vector[N2] b[N1];
```

These distinctions affect either what we can do with these variables, or the speed of our model, and sometimes are interchangeable. Matrix algebra is only defined for (row) vectors and matrices, that is we cannot multiply arrays. The following line requires all the one-dimensional containers (`p_size` and `c_load`) to be defined as vectors (or `row_vectors`):

```
vector[N] mu = alpha + c_load * beta;
```

Many “vectorized” operation are also valid for arrays, that is, `normal_lpdf`, accepts (row) vectors (as we did in our code) or arrays as in the next example. There is of course no point in converting a vector to an array as follows, but this shows that Stan allows both type of one-dimensional containers.

```
real mu[N]= to_array_1d(alpha + c_load * beta);
target += normal_lpdf(p_size | mu, sigma);
```

By contrast, the outcome of “vectorized” pseudorandom number generator (`_rng`) functions can only be stored in an array. The following example shows the only way to vectorize this type of function:

```
real p_size_pred[N] = normal_rng(alpha + c_load * beta, sigma);
```

Alternatively, one can always use a `for`-loop, and it won’t matter if `p_size_pred` is an array or a vector:

```
vector[N] p_size_pred;
for(n in 1:N)
    p_size_pred[n] = normal_rng(alpha + c_load[n] * beta, sigma);
```

See also Stan’s manual section on matrices, vector, and arrays:

https://mc-stan.org/docs/2_26/stan-users-guide/matrices-vectors-and-arrays.html

10.4.2 Interactions in Stan: Does attentional load interact with trial number affecting pupil size?

We’ll expand the previous model to also include the effect of (centered) trial and its interaction with cognitive load on one subject’s pupil size. Our new likelihood will look as follows:

$$p_size_n \sim Normal(\alpha + c_load_n \cdot \beta_1 + c_trial \cdot \beta_2 + c_load \cdot c_trial \cdot \beta_3, \sigma) \quad (10.11)$$

Define priors for all the new β s. Since we don’t have more information about the new predictors, they are sampled from identical prior distributions:

$$\begin{aligned}
 \alpha &\sim \text{Normal}(1000, 500) \\
 \beta_1 &\sim \text{Normal}(0, 100) \\
 \beta_2 &\sim \text{Normal}(0, 100) \\
 \beta_3 &\sim \text{Normal}(0, 100) \\
 \sigma &\sim \text{Normal}_+(0, 1000)
 \end{aligned} \tag{10.12}$$

The following Stan model, `pupil_int1.stan`, is the direct translation of the new priors and likelihood.

```

data {
    int<lower = 1> N;
    vector[N] c_load;
    vector[N] c_trial;
    vector[N] p_size;
}
parameters {
    real alpha;
    real beta1;
    real beta2;
    real beta3;
    real<lower = 0> sigma;
}
model {
    // priors including all constants
    target += normal_lpdf(alpha | 1000, 500);
    target += normal_lpdf(beta1 | 0, 100);
    target += normal_lpdf(beta2 | 0, 100);
    target += normal_lpdf(beta3 | 0, 100);
    target += normal_lpdf(sigma | 0, 1000)
        - normal_lccdf(0 | 0, 1000);
    target += normal_lpdf(p_size | alpha + c_load * beta1 + c_trial * beta2 +
        c_load .* c_trial * beta3, sigma);
}

```

When there are matrices or vectors involved, `*` indicates matrix multiplication whereas `.*` indicates element-wise multiplication; in R `%%*` indi-

cates matrix multiplication whereas \star indicates element-wise multiplication.

There is, however, an alternative notation that can simplify our code. In the following likelihood, p_size is a vector of N observations (in this case 41), X is the model matrix with a dimension of $N \times N_{pred}$ (in this case 41×3), and β a vector of N_{pred} (in this case, 3) rows. Assuming that β is a vector, we indicate with one line that each beta is sampled from identical prior distributions.

$$\begin{aligned} p_size &\sim Normal(\alpha + X \cdot \beta, \sigma) \\ \beta &\sim Normal(0, 100) \\ \sigma &\sim Normal_+(0, 1000) \end{aligned} \tag{10.13}$$

The translation into Stan code is the following:

```
data {
    int<lower = 1> N;
    int<lower = 0> K;    // number of predictors
    matrix[N, K] X;    // model matrix
    vector[N] p_size;
}
parameters {
    real alpha;
    vector[K] beta;
    real<lower = 0> sigma;
}
model {
    // priors including all constants
    target += normal_lpdf(alpha | 1000, 500);
    target += normal_lpdf(beta | 0, 100);
    target += normal_lpdf(sigma | 0, 1000)
        - normal_lccdf(0 | 0, 1000);
    target += normal_lpdf(p_size | alpha + X * beta, sigma);
}
```

For some likelihood functions, Stan provides a more efficient implementation of the linear regression than the one manually written in the previous code. It's critical to understand that, in general, a more efficient im-

lementation should not only be faster, but should also achieve the same number of effective samples (or more) than a less efficient implementation (and should also show convergence). In this case, we can achieve that using `_glm` functions. We can replace the last line with the following statement (the order of the arguments is important):⁵

```
target += normal_id_glm_lpdf(p_size | X, alpha, beta, sigma);
```

The most optimized model, `pupil_int.stan`, includes this last statement. We prepare the data as follows: First create a centered version of trial, `c_trial` and load `c_load`, then use the function `model.matrix` to create the `X` matrix that contains in each column our predictors and omits the intercept with `0 +`.

```
df_pupil <- df_pupil %>%
  mutate(
    c_trial = trial - mean(trial),
    c_load = load - mean(load)
  )
X <- model.matrix(~ 0 + c_load * c_trial, df_pupil)
ls_pupil_X <- list(
  p_size = df_pupil$p_size,
  X = X,
  K = ncol(X),
  N = nrow(df_pupil)
)
```

```
pupil_int <- system.file("stan_models",
  "pupil_int.stan",
  package = "bcogsci"
)
fit_pupil_int <- stan(pupil_int,
  data = ls_pupil_X
)
```

⁵An extra boost in efficiency can be obtained in regular regressions where `x` and `y` are data (rather than parameters as in cases of missing data or measurement error), since this function can be executed on a GPU.

```
print(fit_pupil_int, pars = c("alpha", "beta", "sigma"))

##          mean   2.5%  97.5% n_eff Rhat
## alpha    699.35 667.78 732.81  4290    1
## beta[1]  31.36  12.40  50.62  4977    1
## beta[2] -5.81 -8.46 -3.06  5529    1
## beta[3] -1.84 -3.43 -0.30  5261    1
## sigma    104.43  82.55 133.40  3457    1
```

In 10.9, we plot here the 95% CrI of the parameters of interest. We use `regex_pars`, rather than `pars`, because we want to capture `beta[1]`, `beta[2]`, and `beta[3]`; `regex_pars` use regular expressions to select the parameters (for information about regular expressions in R see <https://stat.ethz.ch/R-manual/R-devel/library/base/html/regex.html>)

```
df_fit_pupil_int <- as.data.frame(fit_pupil_int)
mcmc_intervals(fit_pupil_int,
  regex_pars = "beta",
  prob_outer = .95,
  prob = .8,
  point_est = "mean"
)
```

10.4.3 Logistic regression in Stan: Does set size and trial affect free recall?

We revisit and expand on the analysis presented in 4.3 of a subset of the data of Oberauer (2019). In this example, we will investigate whether the length of a list and trial number affect the probability of correctly recalling a word.

As in section 4.3, we assume a Bernoulli likelihood with a logit link function, and the following priors (recall that the logistic function is the inverse of the logit).

$$\begin{aligned} \text{correct}_n &\sim \text{Bernoulli}(\text{logistic}(\alpha + X \cdot \beta)) \\ \alpha &\sim \text{Normal}(0, 1.5) \\ \beta &\sim \text{Normal}(0, 0.1) \end{aligned} \tag{10.14}$$

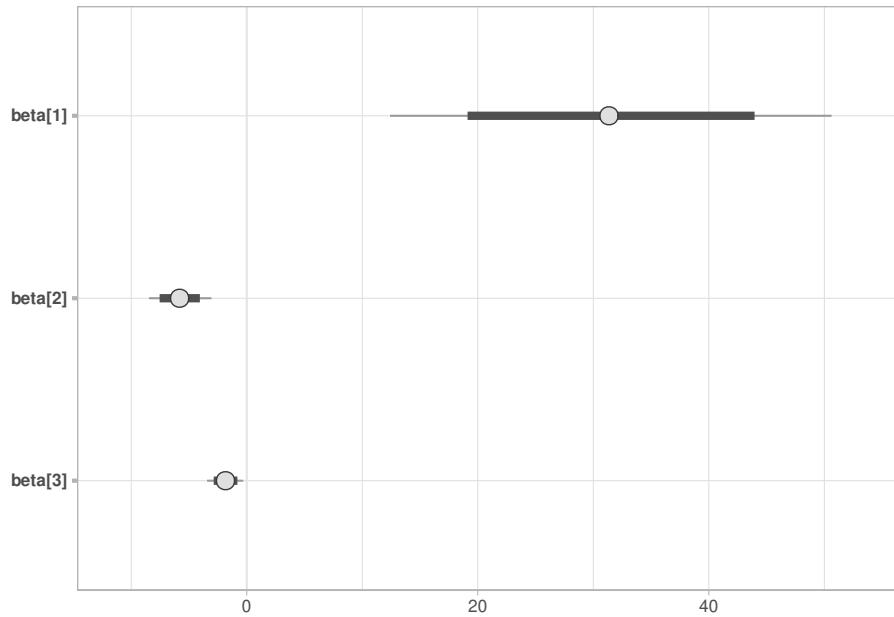


FIGURE 10.9: 95% CrI of the effect of load, β_1 , the effect of trial β_2 , and their interaction β_3 .

Where β is a vector of size $K = 2$, $\{\beta_0, \beta_1\}$. Below in `recall.stan` we present the most efficient way to code this in Stan.

```
data {
    int<lower = 1> N;
    int<lower=0> K;    // number of predictors
    matrix[N, K] X;    // model matrix
    int correct[N];
}
parameters {
    real alpha;
    vector[K] beta;
}
model {
    // priors including all constants
    target += normal_lpdf(alpha | 0, 1.5);
    target += normal_lpdf(beta | 0, .1);
    target += bernoulli_logit_glm_lpmf(correct | X, alpha, beta);
```

```
}
```

The dependent variable, `correct`, is an array of integers rather than a vector; this is because vectors are always composed of real numbers, but the Bernoulli likelihood only accepts the integers 1 or 0. As in the previous example, we are taking advantage of the `_glm` functions. A less efficient but more transparent option would be to replace the last statement with:

```
target += bernoulli_logit_lpmf(correct | alpha + X * beta);
```

We might want to use `bernoulli_logit_lpmf` if we want to define a non-linear relationship between the predictors that are outside the generalized linear model framework. One example would be the following:

```
target += bernoulli_logit_lpmf(correct | alpha + exp(X * beta));
```

Another more flexible possibility when we want to indicate a Bernoulli likelihood is to use `bernoulli_lpmf` and add the link manually. The last statement of `recall.stan` would become the following:

```
target += bernoulli_lpmf(correct | inv_logit(alpha + X * beta));
```

The function `bernoulli_lpmf` can be useful if one wants to try other link functions; see exercise 10.4.

Finally, the most transparent form (but less efficient) would be the following for-loop:

```
for (n in 1:N)
  target += bernoulli_lpmf(correct[n] | inv_logit(alpha + X[n] * beta));
```

To fit the model as `recall.stan`, prepare the data by centering the trial number first:

```
df_recall <- df_recall %>%
  mutate(c_set_size = set_size - mean(set_size),
        c_trial = trial - mean(trial))
```

Next, we create the model matrix, X , and prepare the data as a list. As in section 10.4.2, we exclude the intercept from the matrix x using `0 + ...`. This is because the Stan code that we are using already takes into account that the first column in the model matrix is going to be a vector of ones.

```
X <- model.matrix(~ 0 + c_set_size * c_trial, df_recall)
ls_recall <- list(
  correct = df_recall$correct,
  X = X,
  K = ncol(X),
  N = nrow(df_recall)
)

recall <- system.file("stan_models",
                      "recall.stan",
                      package = "bcogsci")
fit_recall <- stan(
  file = recall,
  data = ls_recall
)
```

After fitting the model we can print and plot summaries of the posterior distribution.

```
print(fit_recall, pars = c("alpha", "beta"))
```

```
##          mean 2.5% 97.5% n_eff Rhat
## alpha     1.99  1.39  2.62  3697    1
## beta[1] -0.19 -0.35 -0.03  3761    1
## beta[2] -0.02 -0.09  0.05  3496    1
## beta[3]  0.00 -0.03  0.04  3076    1
```

In Figure 10.10, we plot the 95% CrI of the parameters of interest.

```
df_fit_recall <- as.data.frame(fit_recall)
mcmc_intervals(df_fit_recall,
  regex_pars = "beta",
  prob_outer = .95,
  prob = .8,
```

```
    point_est = "mean"
)
```

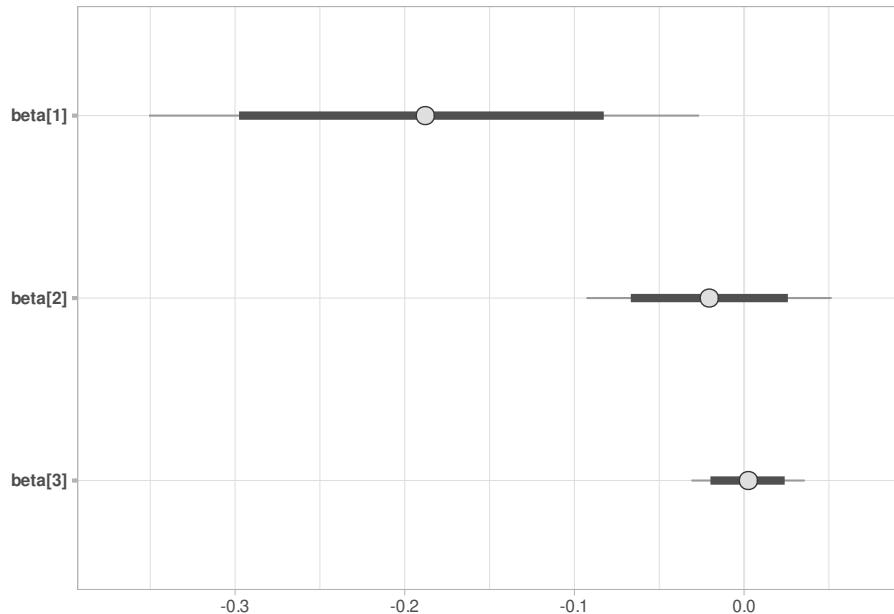


FIGURE 10.10: 95% credible intervals of the `beta` parameters of `fit_recall` model.

As we did in 4.3.4, we might want to communicate the posterior in proportions rather than in log-odds (as seen in the parameters `beta`). We can do this in R manipulating the dataframe `df_fit_recall`, or extracting the samples with `extract(fit_recall)`. Another alternative presented here is by using the generated quantities block. To make the code more compact we declare the type of each variable and store its content in the same line in `recall_prop.stan`.

```
generated quantities {
  real average_accuracy = inv_logit(alpha);
  vector[K] change_acc = inv_logit(alpha) - inv_logit(alpha - beta);
}
```

Recall that due to the non-linearity of the scale, the effects depend on the average accuracy, and the set size and trial that we start from (in this case

we are examining the change of one unit from the average set size and the average trial).

```
recall_prop <- system.file("stan_models",
                             "recall_prop.stan",
                             package = "bcogsci")

fit_recall <- stan(
  file = recall_prop,
  data = ls_recall
)
```

The plot in 10.11 now shows how the average accuracy deteriorates when the subject is exposed to a set size larger than the average by one, a trial further than the middle one, and the interaction of both.

```
df_fit_recall <- as.data.frame(fit_recall) %>%
  rename(
    set_size = `change_acc[1]`,
    trial = `change_acc[2]`,
    interaction = `change_acc[3]`
  )
mcmc_intervals(df_fit_recall,
  pars = c("set_size", "trial", "interaction"),
  prob_outer = .95,
  prob = .8,
  point_est = "mean"
) +
  xlab("Change in accuracy")
```

The plot in 10.11 is showing us that our model is estimating that by increasing the set size by one unit, the recall accuracy of the single subject is deteriorated by 2%. In contrast, there is hardly any trial effect or interaction between trial and set size.

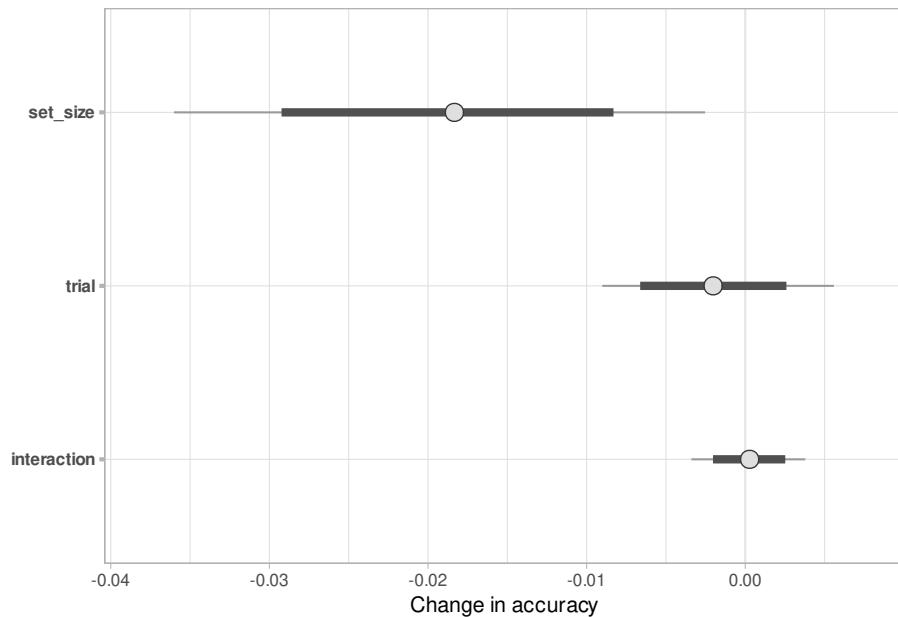


FIGURE 10.11: Effect of set size, trial, and their interaction on the average accuracy of recall.

10.5 Summary

This chapter introduced basic Stan syntax for fitting some standard linear models. Example code covered the normal, binomial, Bernoulli, and log-normal likelihoods. We also saw how to express regression models using the matrix model in Stan syntax.

10.6 Further reading

For further reading on the Hamiltonian Monte Carlo algorithm, see the rather technical review of Betancourt (2017), or the more conceptual introduction provided by Monnahan, Thorson, and Branch (2017). A useful article with example R code is Neal (2011). A detailed walk-through on its implementation is also provided in Chapter 41 of MacKay (2003). The

Stan documentation is an excellent source for further study: <https://mc-stan.org/users/documentation/>.

10.7 Exercises

Exercise 10.1. A very simple model.

In this exercise we revisit the model from 3.1.1.1. We assume the following:

1. There is a true underlying time, μ , that the subject 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 response times are generally skewed; we fix this assumption later).

That is the likelihood for each observation n will be:

$$rt_n \sim Normal(\mu, \sigma) \quad (10.15)$$

- a. Decide on *appropriate* priors and fit this model in Stan. Data can be found in `df_spacebar`
- b. Change the likelihood for a log-normal distribution and change the priors. Fit the model in Stan.

Exercise 10.2. Incorrect Stan model.

We want to fit both response times and accuracy with the same model. We simulate the data as follows:

```
N <- 500
df_sim <- tibble(
  rt = rlnorm(N, mean = 6, sd = .5),
  correct = rbern(N, prob = .85)
)
```

We build the following model:

```
data {  
    int<lower = 1> N;  
    vector[N] rt;  
    int correct[N];  
}  
parameters {  
    real<lower = 0> sigma;  
    real theta;  
}  
model {  
    target += normal_lpdf(mu | 0, 20);  
    target += lognormal_lpdf(sigma | 3, 1)  
    for(n in 1:N)  
        target += lognormal_lpdf(rt[n] | mu, sigma);  
    target += bernoulli_lpdf(correct[n] | theta);  
}
```

Why is this model not working?

```
ls_sim <- list(  
    rt = df_sim$rt,  
    correct = df_sim$correct  
)  
incorrect <- system.file("stan_models", "incorrect.stan", package = "bcogsci")  
fit_sim <- stan(incorrect, data = ls_sim)  
  
## SYNTAX ERROR, MESSAGE(S) FROM PARSER:  
## Variable "mu" does not exist.  
##   error in 'model8ac955214c714_incorrect' at line 11, column 26  
##   -----  
##       9: }  
##      10: model {  
##      11:   target += normal_lpdf(mu | 0, 20);
```

```

##          ^
##      12: target += lognormal_lpdf(sigma | 3, 1)
## -----
##  

## Error in stanc(file = file, model_code = model_code, model_name = model_name, : failed to parse

```

Try to make it run. (Hint: There are several problems.)

Exercise 10.3. Using Stan documentation.

Edit the simple example with Stan from section 10.2, and replace the normal distribution with a skew normal distribution. (Don't forget to add a prior to the new parameter, and check Wikipedia and Stan documentation for more information about the distribution).

Fit the following data:

```
Y <- rnorm(1000, mean = 3, sd = 10)
```

Does the estimate of the new parameter make sense?

Exercise 10.4. The probit link function as an alternative to the logit function.

The probit link function is the inverse of the CDF of the standard normal distribution ($Normal(0, 1)$). Since the CDF of the standard normal is usually denoted with the Greek letter Φ (Phi), the probit is denoted as Φ^{-1} . Refit the model presented in 10.4.3 changing the logit link function for the probit link (that is transforming the regression to a constrained space using `Phi()` in Stan).

You will probably see the following while the model runs, this is because the probit link is less numerically stable (i.e., under and overflows) than the logit link in Stan. Don't worry, it is good enough for this exercise.

```

Rejecting initial value:
Log probability evaluates to log(0), i.e. negative infinity.
Stan can't start sampling from this initial value.

```

- a. Do the results change if the coefficients α and β change?
- b. Do the results in probability space change?

Exercise 10.5. Examining the position of the queued word on recall.

Refit the model presented in 10.4.3 and examine whether, set size, trial effects, the position of the queued word (tested in the dataset), and their interaction affect free recall. (Tip: You can do this exercise without changing the Stan code.).

How does the accuracy change from position one to position two?

Exercise 10.6. The conjunction fallacy.

Paolacci, Chandler, and Ipeirotis (2010) examined whether the results of some classic experiments differ between a university pool population and subjects recruited from Mechanical Turk. We'll examine whether the results of the conjunction fallacy experiment (or Linda problem: Tversky and Kahneman 1983) are replicated for both groups.

```
data("df_fallacy")
df_fallacy
```

```
## # A tibble: 268 x 2
##   source answer
##   <chr>   <int>
## 1 mturk     1
## 2 mturk     1
## 3 mturk     1
## # ... with 265 more rows
```

The conjunction fallacy shows that people often fail to regard a combination of events as less probable than a single event in the combination. Tversky and Kahneman (1983)

Linda is 31 years old, single, outspoken, and very bright. She majored in philosophy. As a student, she was deeply concerned with issues of discrimination and social justice, and also participated in anti-nuclear demonstrations.

Which is more probable?

- a. *Linda is a bank teller.*
- b. *Linda is a bank teller and is active in the feminist movement.*

The majority of those asked chose option b. Even if it's less probably ($\Pr(a \wedge b) \leq \Pr(b)$) The dataset is named `df_fallacy` and it indicates with 0 option "a" and with 1 option b. Fit a logistic regression in Stan and report:

- a. The estimated overall probability of answering (b) ignoring the group.
- b. The estimated overall probability of answering (b) for each group.

11

Complex models and reparametrization

Now that we know how to fit simple regression models using Stan syntax, we can now turn to more complex cases, such as the hierarchical models we fit in `brms` in chapter 5. Fitting such models in Stan allows us a great deal of flexibility. In exchange, however, we need to think on how exactly we code the model. In some cases, two pieces of computer code that are mathematically equivalent might behave differently due to the computer's limitations; in this chapter, we will learn some of the more common techniques needed to optimize the model's behavior. In particular, we will learn how to deal with convergence problems using what is called the non-centered reparameterization. Another very important topic we cover here is defining priors for the correlation parameters in the variance-covariance matrix of the random effects.

11.1 Hierarchical models with Stan

In the following sections, we will revisit and expand on some of the examples from chapter 5.

11.1.1 Varying intercept model with Stan

Recall that in section 5.1 we fit models to investigate the effect of cloze probability on EEG averages in the N400 spatiotemporal time window. For our first model, we'll make the (implausible) assumption that only the average signal varies across subjects, but all subjects share the same effect of cloze probability. This means that the likelihood incorporates the assumption that the intercept, α , is adjusted with the term u_i for each subject.

$$\text{signal}_n \sim \text{Normal}(\alpha + u_{\text{subj}[n]} + c_cloze_n \cdot \beta, \sigma) \quad (11.1)$$

$$\begin{aligned} \alpha &\sim \text{Normal}(0, 10) \\ \beta &\sim \text{Normal}(0, 10) \\ u &\sim \text{Normal}(0, \tau_u) \\ \tau_u &\sim \text{Normal}_+(0, 20) \\ \sigma &\sim \text{Normal}_+(0, 50) \end{aligned} \quad (11.2)$$

Here n represents each observation, the n th row in the data frame and $\text{subj}[n]$ is the subject that corresponds to observation n . We present the mathematical notation of the likelihood with “multiple indexing”: the index of u is provided by the vector subj .

Before we discuss the Stan implementation, let’s see what the location of the normal likelihood, the vector μ , looks like. There are in total 2863 observations; that means that $\mu = \{\mu_1, \mu_2, \dots, \mu_{2863}\}$. We have 37 subjects which means that $u = \{u_1, u_2, \dots, u_{37}\}$. The following equation shows that the use of multiple indexing allows us to have a vector of adjustments with only 37 different elements, with a total length of 2863. In the equation below, the multiplication operator \circ is the Hadamard product: when we write $X \circ B$, both X and B have the same dimensions $m \times n$, and each cell in location $[i, j]$ (where $i = 1, \dots, m$, and $j = 1, \dots, n$) in X and B are multiplied together to give a matrix that also has dimensions $m \times n$.

$$\begin{aligned}
\boldsymbol{\mu} &= \begin{bmatrix} \mu_1 \\ \mu_2 \\ \dots \\ \mu_{101} \\ \mu_{102} \\ \dots \\ \mu_{215} \\ \mu_{216} \\ \mu_{217} \\ \dots \\ \mu_{1000} \\ \dots \\ \mu_{2863} \end{bmatrix} = \begin{bmatrix} \alpha \\ \alpha \\ \dots \\ \alpha \\ \alpha \\ \dots \\ \alpha \\ \alpha \\ \alpha \\ \alpha \\ \dots \\ \alpha \end{bmatrix} + \begin{bmatrix} u_{subj[1]} \\ u_{subj[2]} \\ \dots \\ u_{subj[101]} \\ u_{subj[102]} \\ \dots \\ u_{subj[215]} \\ u_{subj[216]} \\ u_{subj[217]} \\ \dots \\ u_{subj[1000]} \\ \dots \\ u_{i[2863]} \end{bmatrix} + \begin{bmatrix} ccloze_1 \\ ccloze_2 \\ \dots \\ ccloze_{101} \\ ccloze_{102} \\ \dots \\ ccloze_{215} \\ ccloze_{216} \\ ccloze_{217} \\ \dots \\ ccloze_{1000} \\ \dots \\ ccloze_{2863} \end{bmatrix} \circ \begin{bmatrix} \beta \\ \beta \\ \dots \\ \beta \\ \beta \\ \dots \\ \beta \\ \beta \\ \beta \\ \beta \\ \dots \\ \beta \end{bmatrix} \\
&= \begin{bmatrix} \alpha \\ \alpha \\ \dots \\ \alpha \\ \alpha \\ \dots \\ \alpha \\ \alpha \\ \alpha \\ \dots \\ \alpha \\ \dots \\ \alpha \end{bmatrix} + \begin{bmatrix} u_1 \\ u_1 \\ \dots \\ u_2 \\ u_2 \\ \dots \\ u_3 \\ u_3 \\ u_3 \\ \dots \\ u_{13} \\ \dots \\ u_{37} \end{bmatrix} + \begin{bmatrix} -0.476 \\ -0.446 \\ \dots \\ -0.206 \\ 0.494 \\ \dots \\ -0.136 \\ 0.094 \\ 0.294 \\ \dots \\ 0.524 \\ \dots \\ 0.494 \end{bmatrix} \circ \begin{bmatrix} \beta \\ \beta \\ \dots \\ \beta \\ \beta \\ \dots \\ \beta \\ \beta \\ \beta \\ \beta \\ \dots \\ \beta \end{bmatrix} \tag{11.3}
\end{aligned}$$

In this model each subject has their own intercept adjustment u_i , with i indexing the subjects. If u_i is positive, the subject has a more positive EEG signal than the average over all the subjects; if u_i is negative, then the subject has a more negative EEG signal than the average; and if u_i is 0, then the subject has the same EEG signal as the average. As we discussed in section 5.1.3, 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 of 0), whatever the subjects have in common “goes” to α , and u only “absorbs” the differences between subjects through the variance component τ_u .

We implement this in Stan in the `hierarchical1.stan`:

```
data {
    int<lower=1> N;
    vector[N] signal;
    int<lower = 1> N_subj;
    vector[N] c_cloze;
    // The following line creates an array of integers;
    int<lower = 1, upper = N_subj> subj[N];
}
parameters {
    real<lower = 0> sigma;
    real<lower = 0> tau_u;
    real alpha;
    real beta;
    vector[N_subj] u;
}
model {
    target += normal_lpdf(alpha | 0, 10);
    target += normal_lpdf(beta | 0, 10);
    target += normal_lpdf(sigma | 0, 50) -
        normal_lccdf(0 | 0, 50);
    target += normal_lpdf(tau_u | 0, 20) -
        normal_lccdf(0 | 0, 20);
    target += normal_lpdf(u | 0, tau_u);
    target += normal_lpdf(signal | alpha + u[subj] +
        c_cloze * beta, sigma);
}
```

In the previous Stan code we use `int<lower = 1, upper = N_subj> subj[N];` to define a one-dimensional *array* of `N` elements that contains integers (bounded between 1 and `N_subj`). As we explain in Box 10.3, the difference between vectors and one-dimensional arrays is that vectors can only contain real numbers and can be used with matrix algebra functions, and arrays can contain any type but can't be used in matrix algebra. We use `normal_lpdf` rather than `normal_glm_lpdf` since at the moment there is no efficient likelihood implementation of hierarchical generalized linear models.

The following code centers the predictor cloze and stores the data required by the Stan model in a list. Because we are using `subj` as a vector of indices, we need to be careful to have integers starting from 1 and ending in `N_subj` without skipping any number (but the order won't matter).¹

```
data("df_eeg")
df_eeg <- df_eeg %>%
  mutate(c_cloze = cloze - mean(cloze))
ls_eeg <- list(
  N = nrow(df_eeg),
  signal = df_eeg$n400,
  c_cloze = df_eeg$c_cloze,
  subj = df_eeg$subj,
  N_subj = max(df_eeg$subj)
)
```

Fit the model:

```
hierarchical1 <- system.file("stan_models",
  "hierarchical1.stan",
  package = "bcogsci"
)
fit_eeg1 <- stan(
  file = hierarchical1,
  data = ls_eeg
)
```

Summary of the model:

```
print(fit_eeg1, pars = c("alpha", "beta", "sigma", "tau_u"))
```

```
##          mean  2.5% 97.5% n_eff Rhat
## alpha    3.64  2.79  4.48   1790     1
## beta     2.32  1.28  3.35   4996     1
```

¹If this is not the case, we can do `as.numeric(as.factor(df_eeg$subj))` to transform the data appropriately.

```
## sigma 11.64 11.34 11.96 5946    1
## tau_u  2.19  1.54  3.00 2344    1
```

11.1.2 Uncorrelated varying intercept and slopes model with Stan

In the following model, we relax the strong assumption that every subject will be affected equally by the manipulation. For ease of exposition, we start by assuming that the adjustments for the intercept and slope are not correlated, as we did in section 5.1.3.

$$\text{signal}_n \sim \text{Normal}(\alpha + u_{\text{subj}[n],1} + c_{\text{cloze}}_n \cdot (\beta + u_{\text{subj}[n],2}), \sigma) \quad (11.4)$$

$$\begin{aligned} \alpha &\sim \text{Normal}(0, 10) \\ \beta &\sim \text{Normal}(0, 10) \\ u_1 &\sim \text{Normal}(0, \tau_{u_1}) \\ u_2 &\sim \text{Normal}(0, \tau_{u_2}) \\ \tau_{u_1} &\sim \text{Normal}_+(0, 20) \\ \tau_{u_2} &\sim \text{Normal}_+(0, 20) \\ \sigma &\sim \text{Normal}_+(0, 50) \end{aligned} \quad (11.5)$$

We implement this in Stan in `hierarchical2.stan`:

```
data {
  int<lower=1> N;
  vector[N] signal;
  int<lower = 1> N_subj;
  vector[N] c_cloze;
  int<lower = 1, upper = N_subj> subj[N];
}
parameters {
  real<lower = 0> sigma;
  vector<lower = 0>[2] tau_u;
  real alpha;
  real beta;
  matrix[N_subj, 2] u;
}
```

```

model {
    target += normal_lpdf(alpha| 0,10);
    target += normal_lpdf(beta | 0,10);
    target += normal_lpdf(sigma | 0, 50) -
        normal_lccdf(0 | 0, 50);
    target += normal_lpdf(tau_u[1] | 0, 20) -
        normal_lccdf(0 | 0, 20);
    target += normal_lpdf(tau_u[2] | 0, 20) -
        normal_lccdf(0 | 0, 20);
    target += normal_lpdf(u[, 1]| 0, tau_u[1]);
    target += normal_lpdf(u[, 2]| 0, tau_u[2]);
    target += normal_lpdf(signal | alpha + u[subj, 1] +
        c_cloze .* (beta + u[subj, 2]), sigma);
}

```

In the previous model, we assign the same prior distribution to both `tau_u[1]` and `tau_u[2]`, and thus in principle we could have written the two statements in one (we multiply by 2 because there are two PDFs that need to be corrected for the truncation):

```

target += normal_lpdf(tau_u | 0, 20) -
    2 * normal_lccdf(0 | 0, 20);

```

Fit the model as follows:

```

hierarchical2 <- system.file("stan_models",
    "hierarchical2.stan",
    package = "bcogsci"
)
fit_eeg2 <- stan(
    file = hierarchical2,
    data = ls_eeg
)

```

```

## Warning: There were 2 chains where the estimated Bayesian Fraction of Missing Information was low
## http://mc-stan.org/misc/warnings.html#bfmi-low

## Warning: Examine the pairs() plot to diagnose sampling problems

```

```

## Warning: The largest R-hat is 1.1, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#r-hat

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may
## be unreliable. Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess

## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail
## quantiles may be unreliable. Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess

```

We see that there are warnings. As we increase the complexity and the number of parameters, the sampler has a harder time exploring the parameter space.

Show the summary of the model below:

```
print(fit_eeg2, pars = c("alpha", "beta", "tau_u", "sigma"))
```

```

##          mean 2.5% 97.5% n_eff Rhat
## alpha     3.65  2.75  4.50    918 1.00
## beta      2.33  1.08  3.54   3326 1.00
## tau_u[1]  2.17  1.54  2.98   2727 1.00
## tau_u[2]  1.72  0.31  3.39     63 1.09
## sigma    11.62 11.31 11.92   4998 1.00

```

We see that `tau_u[2]` has a low number of effective samples (`n_eff`).

Plot a traceplot displayed in Figure 11.1:

```
traceplot(fit_eeg2, pars = c("alpha", "beta", "tau_u", "sigma"))
```

Figure 11.1 shows that the chains of the parameter `tau_u[2]` are not mixing properly. This parameter is specially problematic because there are not enough data from each subject to estimate this parameter accurately, its estimated mean is quite small (in comparison with `sigma`), it's bounded by zero, and there is a dependency between this parameter and `u[, 2]`. This makes the exploration by the sampler quite hard.

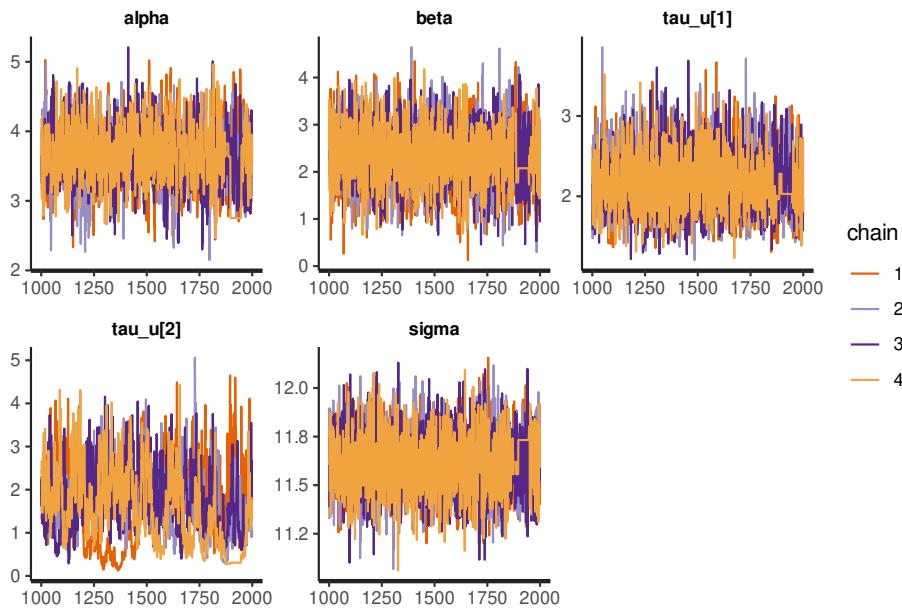


FIGURE 11.1: Traceplots of `alpha`, `beta`, `tau_u`, and `sigma` from the model `fit_eeg2`.

Pairs plots can be useful to uncover pathologies in the sampling, since we can visualize correlations between samples, which are in general problematic. The following code creates a pair plot where we see the samples of `tau_u[2]` against some of the adjustments to the slope u ; see Figure 11.2.

```
pairs(fit_eeg2, pars = c("tau_u[2]", "u[1,2]", "u[2,2]", "u[3,2]"))
```

Compare with `tau_u[1]` plotted against the by-subject adjustments to the intercept. In Figure 11.3, instead of funnels we see blobs, indicating no strong correlation between the parameters.

```
pairs(fit_eeg2, pars = c("tau_u[1]", "u[1,1]", "u[2,1]", "u[3,1]"))
```

In contrast to Figure 11.3 that shows blob-shaped clouds of samples, Figure 11.2 shows a relatively strong correlation between the samples of some of the parameters of the model, in particular τ_2 and the samples of $u_{i,2}$. This strong correlation is hindering the exploration of the sampler lead-

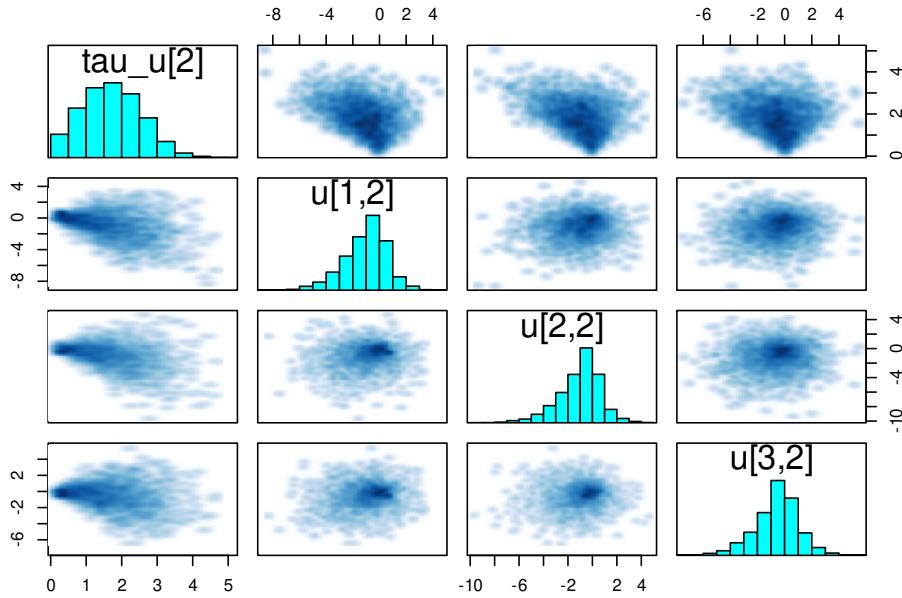


FIGURE 11.2: Pair plots showing a relatively strong correlations (funnel-shaped clouds of samples) between the samples of τ_2 and some of the by-subject adjustments to the slope.

ing to the warnings we saw in Stan. However, the problem that the sampler faces is, in fact, more serious than what our initial plots show. Stan samples in an *unconstrained* space where all the parameters can range from minus infinity to infinity, and then transforms back the parameters to the constrained space that we specified where, for example, a standard deviation parameter is restricted to be positive. This means that Stan is actually sampling from an auxiliary parameter equivalent to $\log(\tau_{\text{u}}[2])$ rather than from $\tau_{\text{u}}[2]$. We can use `mcmc_scatter` to see the actual funnel; see Figure 11.4.

```
mcmc_pairs(as.array(fit_eeg2),
  pars = c("tau_u[2]", "u[1,2"]),
  transform = list(`tau_u[2]` = "log")
)
```

At the neck of the funnel, $\tau_{\text{u}}[2]$ is close to zero (and $\log(\tau_{\text{u}}[2])$ is a negative number) and thus the adjustment u is constrained to be near

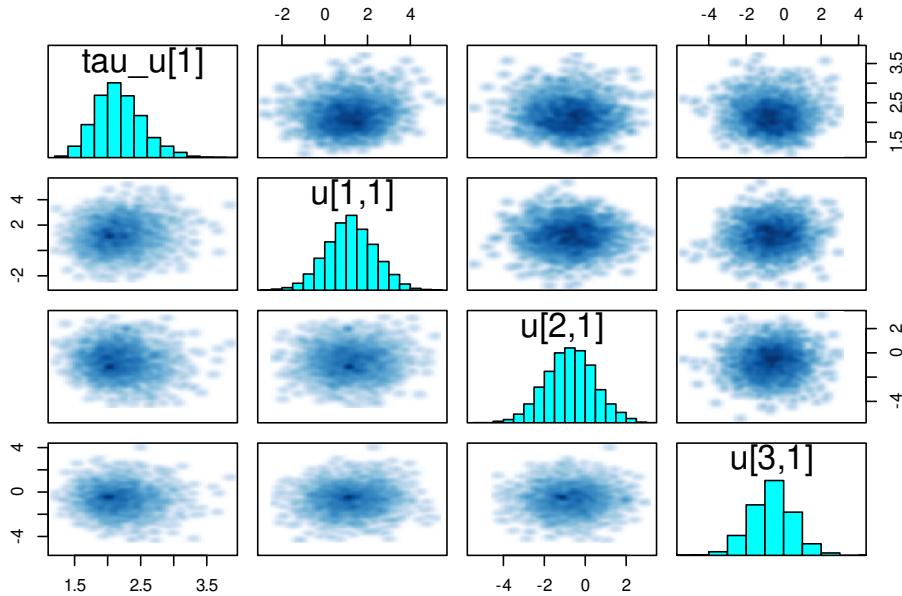


FIGURE 11.3: Pair plots showing no strong correlation (blob-shaped clouds of samples) between the samples of τ_1 and some of the by-subject adjustments to the intercept.

o. This is a problem because a step size that's optimized to work well in the broad part of the funnel will fail to work appropriately in the neck of the funnel and vice versa; see also Neal's funnel (Neal 2003) and the optimization chapter of Stan's manual (Stan Development Team 2021, sec 22.7). There are two options, we might just remove the by-subject varying slope since it's not giving us much information anyway, or we can alleviate this problem by re-parameterizing the model. In general, this is the trickiest and probably most annoying part of modeling. A model can be theoretically and mathematically sound, but still fail to converge. The best advice to solve this type of problem is to start small with simulated data where we know the true values of the parameters, and increase the complexity of the models gradually. Although in this example the problem was clearly in the parameterization of $\tau_u[2]$, in many cases the biggest hurdle is to identify where the problem lies. Fortunately, the issue with $\tau_u[2]$ is a common problem which is easy to solve by using a non-centered parametrization (Papaspiliopoulos, Roberts, and Sköld 2007). The following Box ex-

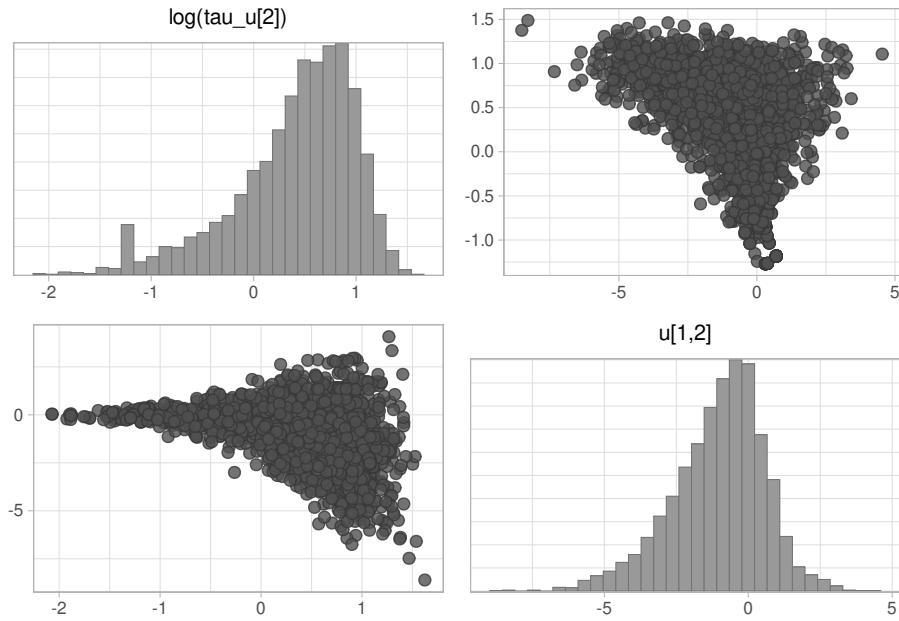


FIGURE 11.4: Pair plots showing a strong correlation (funnel-shaped clouds of samples) between the samples of τ_2 and one of the by-subject adjustments to the intercept ($u_{1,2}$).

plains the specific re-parametrization we use for the improved version of our Stan code.

Box 11.1. *A simple non-centered re-parametrization*

The sampler can explore the parameter space more easily if its step size is appropriate for all the parameters. This is achieved when there are no strong correlations between parameters. We want to assume the following

$$u_2 \sim \text{Normal}(0, \tau_{u_2}) \quad (11.6)$$

where u_2 is the column vector of $u_{i,2}$'s. The index i refers to the subject id.

We can transform u_2 to z-scores as follows. This amounts to center-

ing the parameter, so we can call this this centered parameteriza-tion.

$$z_{u_2} = \frac{u_2 - 0}{\tau_{u_2}} \quad (11.7)$$

where

$$z_{u_2} \sim Normal(0, 1) \quad (11.8)$$

Now z_{u_2} is easier to sample because it doesn't depend on other pa-rameters (in particular, it is no longer conditional on τ) and its scale is 1. Once we have sampled this centered parameter, we can derive the actual parameter we care about by carrying out the reverse op-eration, which is called a non-centered parameterization:

$$u_2 = z_{u_2} \cdot \tau_{u_2} \quad (11.9)$$

A question that might be raised here is whether using a non-centered parametrization is always a good idea. Betancourt and Girolami (2013) point out that the extremeness of the correlation depends on the amount of data, and the efficacy of the parametrization de-pends on the relative strength of the data. When there is enough data this parametrization is unnecessary and it may be more efficient to use the centered parameterization. However, cases where there is enough data to render this parametrization useless are also cases where the partial pooling of the hierarchical models isn't needed in the first place. Although data from conventional lab experiments in psychology, psycholinguistics, and related areas seem to benefit from the non-centered parametrization, the jury is still out for larger data sets with thousands of subjects from crowdsourcing websites.

From a mathematical point of view, the following model is equivalent to the one described in Equations (11.4) and (11.11). However, as discussed previously, the computational implementation of the “new” model is more efficient. The following model includes the re-parametrization of both adjustments, u_1 , and u_2 , although the re-parametrization of u_1 is not strictly necessary (we didn't see any problems in the trace plots), it won't hurt either and the Stan code will be simpler.

$$\text{signal}_n \sim \text{Normal}(\alpha + u_{\text{subj}[n],1} + c_{\text{cloze}}_n \cdot (\beta + u_{\text{subj}[n],2}), \sigma) \quad (11.10)$$

$$\begin{aligned}\alpha &\sim \text{Normal}(0, 10) \\ \beta &\sim \text{Normal}(0, 10) \\ z_{u_1} &\sim \text{Normal}(0, 1) \\ z_{u_2} &\sim \text{Normal}(0, 1) \\ \tau_{u_1} &\sim \text{Normal}_+(0, 20) \\ \tau_{u_2} &\sim \text{Normal}_+(0, 20) \\ \sigma &\sim \text{Normal}_+(0, 50) \\ u_1 &= z_{u_1} \cdot \tau_{u_1} \\ u_2 &= z_{u_2} \cdot \tau_{u_2}\end{aligned}\quad (11.11)$$

The following Stan code (`hierarchical3.stan`) uses the previous reparametrization, and introduces some new Stan functions: `to_vector()` converts a matrix into a long column vector (in column-major order, that is, concatenating the columns from left to right); and `std_normal_lpdf()` implements the log PDF of a standard normal distribution, a normal distribution with location 0 and scale 1. This function is just a more efficient version of `normal_lpdf(... | 0, 1)`. We also introduce a new optional block called `transformed parameters`, with each iteration of the sampler, the values of the parameters (i.e., `alpha`, `beta`, `sigma`, and `z`) are available at the `transformed parameters` block, and we can derive new auxiliary variables based on them. In this case, we use `z_u` and `tau_u` to obtain `u`, that then becomes available in the `model` block.

```
data {
    int<lower=1> N;
    vector[N] signal;
    int<lower = 1> N_subj;
    vector[N] c_cloze;
    int<lower = 1, upper = N_subj> subj[N];
}
parameters {
    real<lower = 0> sigma;
```

```

    vector<lower = 0>[2] tau_u;
    real alpha;
    real beta;
    matrix[N_subj, 2] z_u;
}
transformed parameters {
    matrix[N_subj, 2] u;
    u[, 1] = z_u[, 1] * tau_u[1];
    u[, 2] = z_u[, 2] * tau_u[2];
}
model {
    target += normal_lpdf(alpha | 0, 10);
    target += normal_lpdf(beta | 0, 10);
    target += normal_lpdf(sigma | 0, 50) -
        normal_lccdf(0 | 0, 50);
    target += normal_lpdf(tau_u[1] | 0, 20) -
        normal_lccdf(0 | 0, 20);
    target += normal_lpdf(tau_u[2] | 0, 20) -
        normal_lccdf(0 | 0, 20);
    target += std_normal_lpdf(to_vector(z_u));
    target += normal_lpdf(signal | alpha + u[subj, 1] +
        c_cloze .* (beta + u[subj, 2]), sigma);
}

```

By re-parametrizing the model we can also optimize it more, we can convert the matrix z_u into a long column vector allowing us to use a single call of `std_normal_lpdf`. We fit the model named `hierarchical3.stan`.

```

hierarchical3 <- system.file("stan_models",
    "hierarchical3.stan",
    package = "bcogsci"
)
fit_eeg3 <- stan(
    file = hierarchical3,
    data = ls_eeg
)

```

We verify that the model worked as expected by printing its summary and traceplots; see Figure 11.5.

```
print(fit_eeg3, pars = c("alpha", "beta", "tau_u", "sigma"))
```

```
##          mean 2.5% 97.5% n_eff Rhat
## alpha     3.64 2.77 4.46 1632 1.00
## beta      2.32 1.11 3.53 4251 1.00
## tau_u[1]  2.18 1.53 3.02 1288 1.01
## tau_u[2]  1.72 0.11 3.47 1317 1.00
## sigma    11.62 11.32 11.92 5931 1.00
```

```
traceplot(fit_eeg3, pars = c("alpha", "beta", "tau_u", "sigma"))
```

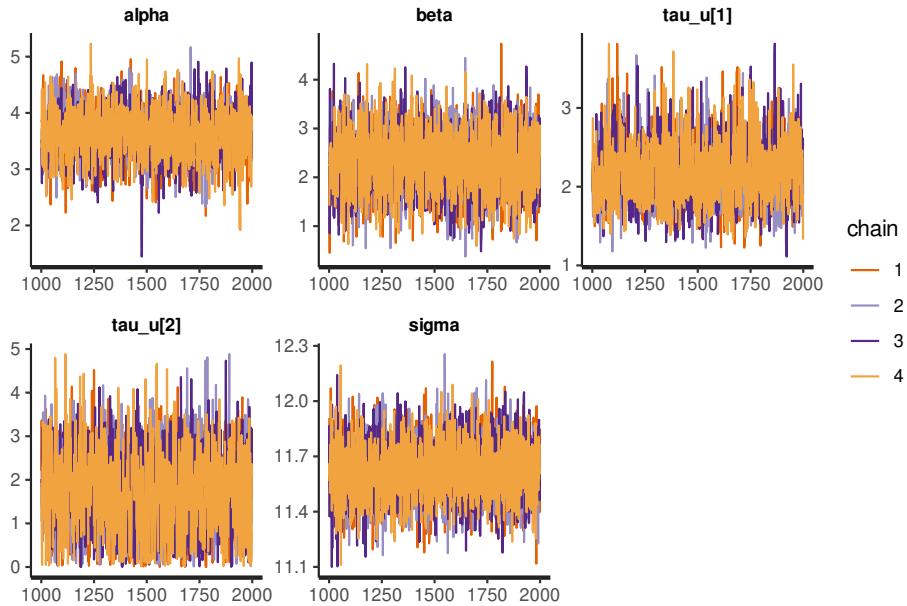


FIGURE 11.5: Traceplots of `alpha`, `beta`, `tau_u`, and `sigma` from the model `fit_eeg3`.

Although the samples of `tau_u[2]` are still correlated with the adjustments for the slope, `u[,2]`, these latter parameters are not the ones explored by the model, the auxiliary parameters, `z_u`, are the relevant ones for the

sampler. The plots 11.6 and 11.7 show that although $\log(\tau_{\text{u}}[2])$ and $u[1,2]$ are still correlated, $\log(\tau_{\text{u}}[2])$ and $z_{\text{u}}[1,2]$ are not.

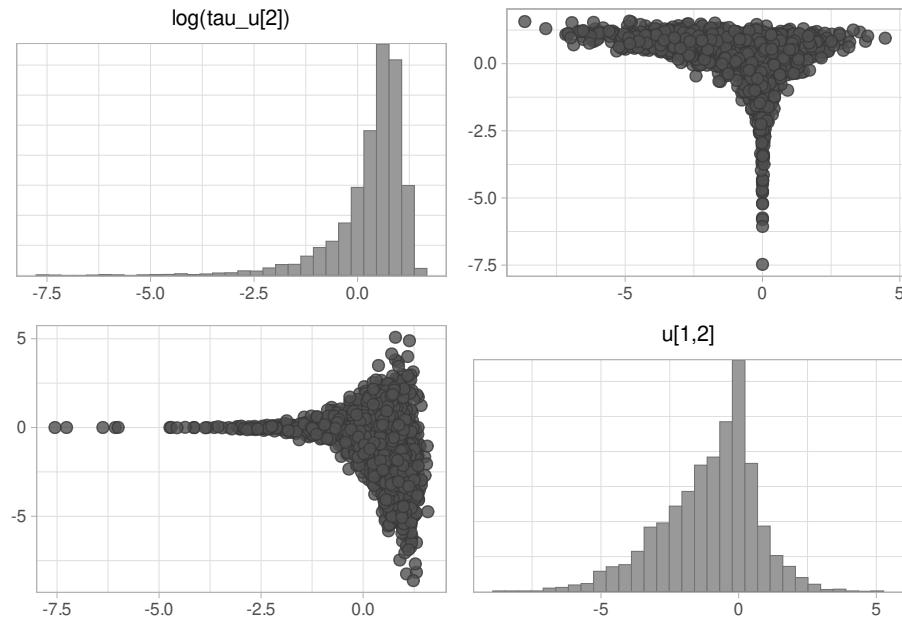


FIGURE 11.6: Pair plots showing a clear correlations (funnel-shaped clouds) between the samples of τ_2 and some of the by-subject adjustments to the slope.

11.1.3 Correlated varying intercept varying slopes model

For the model with correlated varying intercepts and slopes, the likelihood remains identical to the model without a correlation between group-level intercepts and slopes. Priors and hyperpriors change to reflect the potential correlation between by-subject adjustments to intercepts and slopes:

$$\text{signal}_n \sim \text{Normal}(\alpha + u_{\text{subj}[n],1} + c_cloze_n \cdot (\beta + u_{\text{subj}[n],2}), \sigma) \quad (11.12)$$

The correlation is indicated in the priors on the adjustments for vector of by-subject intercepts u_1 and the vector of by-subject slopes u_2 .

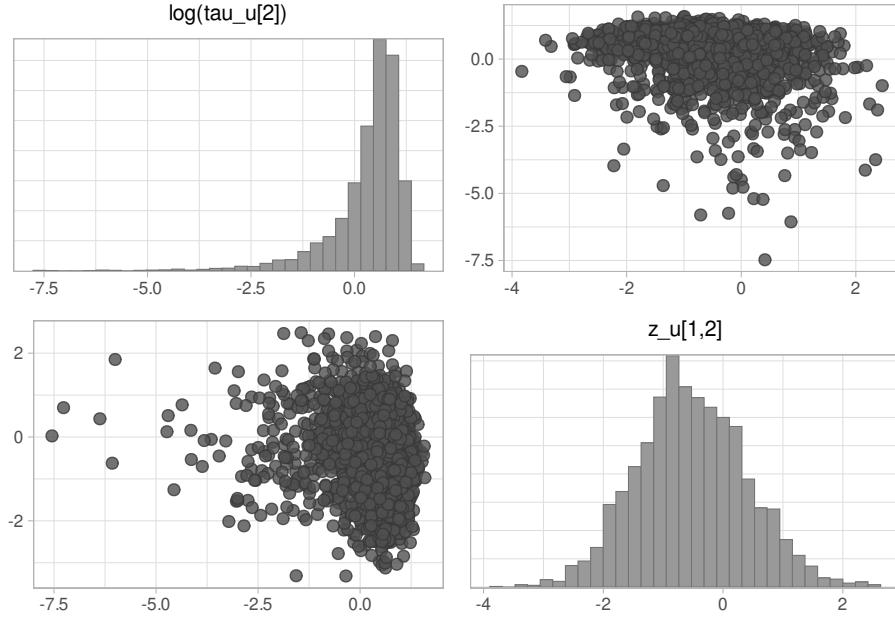


FIGURE 11.7: Pair plots showing no clear correlation between the samples of τ_2 and some of the by-subject auxiliary parameters (z_u) used to build the adjustments to the slope.

- Priors:

$$\begin{aligned} \alpha &\sim \text{Normal}(0, 10) \\ \beta &\sim \text{Normal}(0, 10) \\ \sigma &\sim \text{Normal}_+(0, 50) \\ \begin{pmatrix} u_{i,1} \\ u_{i,2} \end{pmatrix} &\sim \mathcal{N} \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_u \right) \end{aligned} \quad (11.13)$$

where $i = \{1, \dots, N_{subj}\}$

$$\Sigma_u = \begin{pmatrix} \tau_{u_1}^2 & \rho_u \tau_{u_1} \tau_{u_2} \\ \rho_u \tau_{u_1} \tau_{u_2} & \tau_{u_2}^2 \end{pmatrix} \quad (11.14)$$

$$\begin{aligned} \tau_{u_1} &\sim \text{Normal}_+(0, 20) \\ \tau_{u_2} &\sim \text{Normal}_+(0, 20) \\ \rho_u &\sim LKJcorr(2) \end{aligned} \quad (11.15)$$

As a first attempt, we write this model following the mathematical

notation as closely as possible. We'll see that this will be problematic in terms of efficient sampling and convergence. In this Stan model (`hierarchical_corr.stan`), we use some new functions and types:

- `corr_matrix[n] M;` defines a (square) matrix of n rows and n columns called M , symmetrical around a diagonal of ones.
- `rep_row_vector(X, n)` creates a row vector with n columns filled with X .
- `quad_form_diag(M, V)` creates a *quadratic form* using the column vector V as a diagonal matrix (a matrix with all zeros except for its diagonal), this function corresponds in Stan to: `diag_matrix(V) * M * diag_matrix(V)` and in R to `diag(V) %*% M %*% diag(V)`. This computes a variance-covariance matrix from the vector of standard deviations and the correlation matrix (recall the generation of multivariate data in section 1.6.3).

```
data {
    int<lower=1> N;
    vector[N] signal;
    int<lower = 1> N_subj;
    vector[N] c_cloze;
    int<lower = 1, upper = N_subj> subj[N];
}
parameters {
    real<lower = 0> sigma;
    vector<lower = 0>[2] tau_u;
    real alpha;
    real beta;
    matrix[N_subj, 2] u;
    corr_matrix[2] rho_u;
}
model {
    target += normal_lpdf(alpha | 0, 10);
    target += normal_lpdf(beta | 0, 10);
    target += normal_lpdf(sigma | 0, 50) -
        normal_lccdf(0 | 0, 50);
    target += normal_lpdf(tau_u[1] | 0, 20) -
        normal_lccdf(0 | 0, 20);
    target += normal_lpdf(tau_u[2] | 0, 20) -
        normal_lccdf(0 | 0, 20);
```

```

target += lkj_corr_lpdf(rho_u | 2);
for(i in 1:N_subj)
    target += multi_normal_lpdf(u[i,] |
                                rep_row_vector(0, 2),
                                quad_form_diag(rho_u, tau_u));
target += normal_lpdf(signal | alpha + u[subj, 1] +
                      c_cloze .* (beta + u[subj, 2]), sigma);
}

```

Problematic aspects of the first model presented in section 11.1.2 (before the reparameterization), that is, dependencies between parameters, are also present here. Fit the model as follows:

```

hierarchical_corr <- system.file("stan_models",
                                 "hierarchical_corr.stan",
                                 package = "bcogsci"
)
fit_eeg_corr <- stan(
    file = hierarchical_corr,
    data = ls_eeg
)

## Warning: There were 1 divergent transitions after warmup. See
##          http://mc-stan.org/misc/warnings.html#divergent-transitions-
##          after-warmup
## to find out why this is a problem and how to eliminate them.

## Warning: There were 23 transitions after warmup that exceeded the maximum treedepth. Increase m
##          http://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded

## Warning: There were 4 chains where the estimated Bayesian Fraction of Missing Information was l
##          http://mc-stan.org/misc/warnings.html#bfmi-low

## Warning: Examine the pairs() plot to diagnose sampling problems

## Warning: The largest R-hat is NA, indicating chains have not mixed.
## Running the chains for more iterations may help. See
##          http://mc-stan.org/misc/warnings.html#r-hat

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians ma

```

```
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess

## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess
```

As we expected, there are warnings and bad mixing of the chains for $\tau_u[2]$; see also the traceplot in Figure 11.8.

```
print(fit_eeg_corr, pars = c("alpha", "beta", "tau_u", "sigma"))
```

```
##          mean 2.5% 97.5% n_eff Rhat
## alpha     3.72 2.83 4.50   124 1.04
## beta      2.29 1.17 3.49  1075 1.01
## tau_u[1]  2.24 1.53 2.99    83 1.06
## tau_u[2]  1.35 0.02 3.47    17 1.31
## sigma     11.62 11.32 11.92  4862 1.00
```

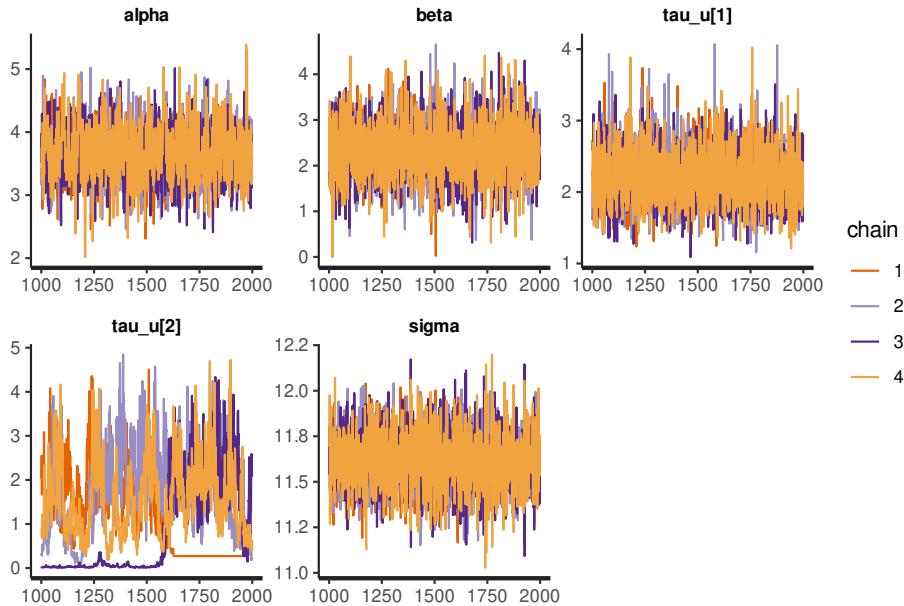


FIGURE 11.8: Traceplots of α , β , τ_u , and σ from the model `fit_eeg_corr`.

The problem (which can also be discovered in a pairs plot) is the same one that we saw before: There is a strong correlation between `tau_u[2]` (in fact, `log(tau_u[2])`, which is the parameter dimension that the sampler considers) and `u`, creating a funnel.

The solution to this problem is the reparametrization of this model. The non-centered parametrization for this type of model is called the Cholesky factorization. The mathematics and the intuition behind this parametrization are explained in Box 11.2.

Box 11.2. Cholesky factorization

First, some definitions that we will need below. A matrix is square if the number of rows and columns is identical. A square matrix A is symmetric if $A^T = A$, i.e., if transposing the matrix gives the matrix back. Suppose that A is a known matrix with real numbers. If x is a vector of variables with length p (a $p \times 1$ matrix), then $x^T Ax$ is called a quadratic form in x ($x^T Ax$ will be a scalar, 1×1). If $x^T Ax > 0$ for all x , then A is a positive definite matrix. If $x^T Ax \geq 0$ for all x , then A is positive semi-definite.

We encountered correlation matrices first in section 1.6.3. A correlation matrix is always symmetric, has 1's along the diagonal, and real values ranging between -1 and 1 on the off-diagonals. Given a correlation matrix ρ_u that is positive definite or semi-definite, we can decompose it into a lower triangular matrix L_u such that $L_u L_u^T = \rho_u$. The matrix L_u is called the Cholesky factor of ρ_u . Intuitively, you can think of L_u as the matrix equivalent of the square root of ρ_u . More details on the Cholesky factorization are in Gentle (2007).

$$L_u = \begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix} \quad (11.16)$$

For a model without a correlation between adjustments for the intercept and slope, we assumed that adjustments u_1 and u_2 were generated by two independent normal distributions. But now we want those adjustments to be correlated. We can use the Cholesky fac-

torization to generate correlated random variables in the following way.

1. Generate uncorrelated vectors, z_{u_1} and z_{u_2} , for each vector of adjustments u_1 and u_2 , as sampled from $Normal(0, 1)$:

$$\begin{aligned} z_{u_1} &\sim Normal(0, 1) \\ z_{u_2} &\sim Normal(0, 1) \end{aligned}$$

2. By multiplying the Cholesky factor by our z 's, generate a matrix that contains two row vectors of correlated variables (with standard deviation of 1).

$$L_u \cdot z_u = \begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix} \begin{pmatrix} z_{u_{1,subj=1}} & z_{u_{1,subj=2}} & \dots & z_{u_{1,subj=N_{subj}}} \\ z_{u_{2,subj=1}} & z_{u_{2,subj=2}} & \dots & z_{u_{2,subj=N_{subj}}} \end{pmatrix}$$

$$L_u \cdot z_u = \begin{pmatrix} l_{11} \cdot z_{u_{1,1}} + 0 \cdot z_{u_{2,1}} & \dots & l_{11} \cdot z_{u_{1,N_{subj}}} \\ l_{21} \cdot z_{u_{1,1}} + l_{22} \cdot z_{u_{2,1}} & \dots & l_{21} \cdot z_{u_{1,N_{subj}}} + l_{22} \cdot z_{u_{2,N_{subj}}} \end{pmatrix}$$

A very informal explanation of why this works is that we are making the variable that corresponds to the slope to be a function of a scaled version of the intercept.

3. The last step is to scale the previous matrix to the desired standard deviation. We define the diagonalized matrix $diag_matrix(\tau_u)$ as before:

$$\begin{pmatrix} \tau_{u_1} & 0 \\ 0 & \tau_{u_2} \end{pmatrix}$$

Then pre-multiply it by the correlated variables with SD of 1 from before:

$$\mathbf{u} = \text{diag_matrix}(\tau_u) \cdot \mathbf{L}_u \cdot \mathbf{z}_u =$$

$$\begin{pmatrix} \tau_{u_1} & 0 \\ 0 & \tau_{u_2} \end{pmatrix} \begin{pmatrix} l_{11} \cdot z_{u_{1,1}} & \dots \\ l_{21} \cdot z_{u_{1,1}} + l_{22} \cdot z_{u_{2,1}} & \dots \end{pmatrix}$$

$$\begin{pmatrix} \tau_{u_1} \cdot l_{11} \cdot z_{u_{1,1}} & \tau_{u_1} \cdot l_{11} \cdot z_{u_{1,2}} & \dots \\ \tau_{u_2} \cdot (l_{21} \cdot z_{u_{1,1}} + l_{22} \cdot z_{u_{2,1}}) & \tau_{u_2} \cdot (l_{21} \cdot z_{u_{1,2}} + l_{22} \cdot z_{u_{2,2}}) & \dots \end{pmatrix}$$

It might be helpful to see how one would implement this in R:

- Let's assume a correlation of 0.8.

```

rho <- .8
# Correlation matrix
(rho_u <- matrix(c(1, rho, rho, 1), ncol = 2))

##      [,1] [,2]
## [1,]  1.0  0.8
## [2,]  0.8  1.0

# Cholesky factor:
# (Transpose it so that it looks the same as in Stan)
(L_u <- t(chol(rho_u)))

##      [,1] [,2]
## [1,]  1.0  0.0
## [2,]  0.8  0.6

# Verify that we recover rho_u,
# Recall that %*% indicates matrix multiplication
L_u %*% t(L_u)

##      [,1] [,2]
## [1,]  1.0  0.8

```

```
## [2,] 0.8 1.0

1 Generate uncorrelated z from a standard normal distribution assuming
only 10 subjects.

N_subj <- 10
(z_u1 <- rnorm(N_subj, 0, 1))

## [1] -0.2542 -0.1733  0.5189  1.0515 -0.0886  1.5616  0.9669
## [8] -0.0501 -1.5723 -0.3225

(z_u2 <- rnorm(N_subj, 0, 1))

## [1] -1.4407  0.6259  0.5980  0.2818  0.3068  0.0187 -0.3614
## [8] -0.8778 -2.2983  0.0235
```

2 Create matrix of correlated parameters:

First, create a matrix with the uncorrelated parameters:

```
# matrix z_u
(z_u <- matrix(c(z_u1, z_u2), ncol = N_subj, byrow = TRUE))

##      [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]
## [1,] -0.254 -0.173  0.519  1.052 -0.0886  1.5616  0.967
## [2,] -1.441  0.626  0.598  0.282  0.3068  0.0187 -0.361
##      [,8]   [,9]   [,10]
## [1,] -0.0501 -1.57 -0.3225
## [2,] -0.8778 -2.30  0.0235
```

Then, generate correlated parameters by pre-multiplying the z_u matrix with L_u .

```
L_u %*% z_u

##      [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]   [,10]
```

```
## [1,] -0.254 -0.173 0.519 1.05 -0.0886 1.56 0.967 -0.0501 -
1.57 -0.322
## [2,] -1.068  0.237 0.774 1.01  0.1132 1.26 0.557 -0.5668 -
2.64 -0.244
```

3 Use the following diagonal matrix to scale the z_u

```
tau_u1 <- .2
tau_u2 <- .01
(diag_matrix_tau <- diag(c(tau_u1, tau_u2)))
```

```
##      [,1] [,2]
## [1,] 0.2  0.00
## [2,] 0.0  0.01
```

4 Finally, generate the adjustments for each subject u :

```
(u <- diag_matrix_tau %*% L_u %*% z_u)

##      [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
## [1,] -0.0508 -0.03466 0.10378 0.2103 -0.01772 0.3123
## [2,] -0.0107  0.00237 0.00774 0.0101  0.00113 0.0126
##      [,7]     [,8]     [,9]     [,10]
## [1,] 0.19338 -0.01001 -0.3145 -0.06449
## [2,] 0.00557 -0.00567 -0.0264 -0.00244
```

The rows are correlated ~.8

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

```
## [1] 0.92
```

The tau's can be recovered as well:

```
sd(u[1, ])
```

```
## [1] 0.178
```

```
sd(u[2, ])
```

```
## [1] 0.0116
```

The reparameterization of the model, which allows for a correlation between adjustments for the intercepts and slopes, in `hierarchical_corr2.stan` is shown below. The code implements the following new types and functions:

- `cholesky_factor_corr[2] L_u`, which defines L_u as a lower triangular (2×2) matrix which has to be the Cholesky factor of a correlation.
- `diag_pre_multiply(tau_u, L_u)` which makes a diagonal matrix out of the vector `tau_u` and multiplies it by `L_u`.
- `to_vector(z_u)` makes a long vector out the matrix `z_u`.
- $L_u \sim lkj_corr_cholesky(2)$ is the Cholesky factor associated with the lkj correlation distribution. It implies that $L_u * L_u' = \rho_u \sim lkj_corr(2.0)$. The symbol `'` indicates transposition (in R, this corresponds to the function `t(.)`).

```
data {
    int<lower=1> N;
    vector[N] signal;
    int<lower = 1> N_subj;
    vector[N] c_cloze;
    int<lower = 1, upper = N_subj> subj[N];
}
parameters {
    real<lower = 0> sigma;
    vector<lower = 0>[2] tau_u;
    real alpha;
    real beta;
    matrix[2, N_subj] z_u;
    cholesky_factor_corr[2] L_u;
}
transformed parameters {
    matrix[N_subj, 2] u;
    u = (diag_pre_multiply(tau_u, L_u) * z_u)';
}
```

```

model {
    target += normal_lpdf(alpha| 0,10);
    target += normal_lpdf(beta | 0,10);
    target += normal_lpdf(sigma | 0, 50) -
        normal_lccdf(0 | 0, 50);
    target += normal_lpdf(tau_u[1] | 0, 20) -
        normal_lccdf(0 | 0, 20);
    target += normal_lpdf(tau_u[2] | 0, 20) -
        normal_lccdf(0 | 0, 20);
    target += lkj_corr_cholesky_lpdf(L_u | 2);
    target += std_normal_lpdf(to_vector(z_u));
    target += normal_lpdf(signal | alpha + u[subj, 1] +
        c_cloze .* (beta + u[subj, 2]), sigma);
}
generated quantities {
    corr_matrix[2] rho_u= L_u * L_u';
    vector[N_subj] effect_by_subj;
    for(i in 1:N_subj)
        effect_by_subj[i] = beta + u[i, 2];
}

```

In this Stan model, we also created an `effect_by_subj` in the generated quantities. This would allow us to plot or to summarize by-subject effects of cloze probability.

One can recover the correlation parameter by adding in the generated quantities section a 2×2 matrix `rho_u`, defined as `rho_u = L_u * L_u'`.

Fit the new model:

```

hierarchical_corr2 <- system.file("stan_models",
    "hierarchical_corr2.stan",
    package = "bcogsci"
)
fit_eeg_corr2 <- stan(
    file = hierarchical_corr2,
    data = ls_eeg
)

```

The Cholesky matrix has elements which are always zero or one, and thus the variance within and between chains (and therefore Rhat) are not defined. However, the rest of the parameters of the model have an appropriate number of effective sample size (more than 10% of the total number of post-warmup samples), Rhats are close to one, and the chains are mixing well; see also the traceplots in Figure 11.9.

```
print(fit_eeg_corr2,
      pars =
        c("alpha", "beta", "tau_u", "rho_u", "sigma", "L_u")
)

##           mean 2.5% 97.5% n_eff Rhat
## alpha      3.65  2.83  4.45  1154   1
## beta       2.34  1.18  3.55  3190   1
## tau_u[1]   2.20  1.56  3.00  1440   1
## tau_u[2]   1.60  0.10  3.44  1011   1
## rho_u[1,1] 1.00  1.00  1.00  NaN    NaN
## rho_u[1,2]  0.17 -0.57  0.79  3096   1
## rho_u[2,1]  0.17 -0.57  0.79  3096   1
## rho_u[2,2]  1.00  1.00  1.00  689    1
## sigma     11.62 11.32 11.94  5343   1
## L_u[1,1]   1.00  1.00  1.00  NaN    NaN
## L_u[1,2]   0.00  0.00  0.00  NaN    NaN
## L_u[2,1]   0.17 -0.57  0.79  3096   1
## L_u[2,2]   0.91  0.59  1.00  1886   1
```

Is there a correlation between the by-subject intercept and slope?

Let's visualize some of the posteriors in Figure 11.10 with the following code:

```
mcmc_hist(as.data.frame(fit_eeg_corr2),
          pars = c("beta", "rho_u[1,2]")
)
```

The posterior distribution of the correlation parameter shows that one can't really know whether the by-subject intercepts and slopes are cor-

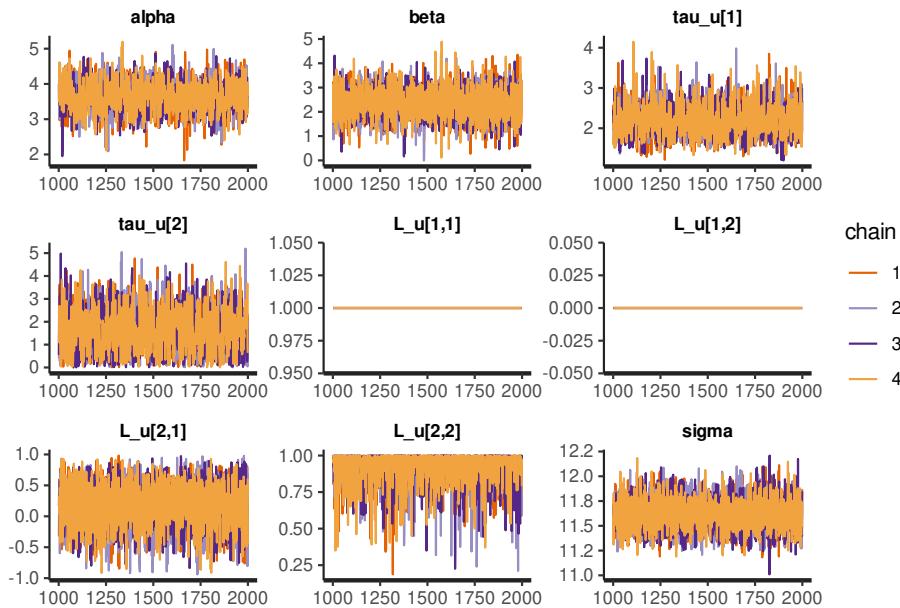


FIGURE 11.9: Traceplots of α , β , τ_u , and σ from the model `fit_eeg_corr`.

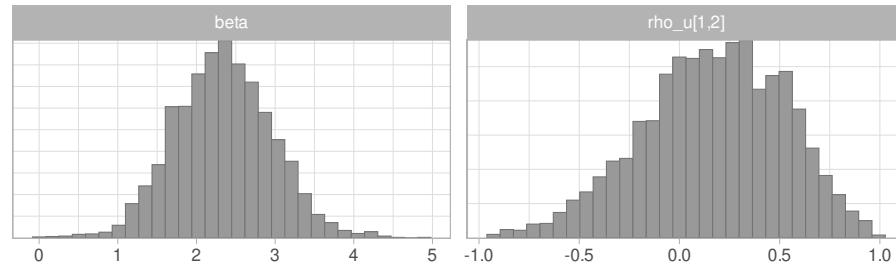


FIGURE 11.10: Histograms of the samples of the posteriors of β and $\rho_{u[1,2]}$ from the model `fit_eeg_corr2`.

related. The broad spread of the posterior indicates that we don't have enough data to estimate this parameter accurately: the posterior ranges from -1 to 1 , and is basically just reflecting the prior specification (the LKJcorr prior with parameter 2).

11.1.4 By-subject and by-items correlated varying intercept varying slopes model

We extend the previous model by adding by-items intercepts and slopes, and priors and hyperpriors that reflect the potential correlation between by-items adjustments to intercepts and slopes:

$$\begin{aligned} signal_n &\sim Normal(\alpha + u_{subj[n],1} + w_{item[n],1} + \\ & \quad c_cloze_n \cdot (\beta + u_{subj[n],2} + w_{item[n],2}), \sigma) \end{aligned} \quad (11.17)$$

The correlation is indicated in the priors on the adjustments for the vectors representing the varying intercepts u_1 and varying slopes u_2 for subjects, and the varying intercepts w_1 and varying slopes w_2 for items.

- Priors:

$$\begin{aligned} \alpha &\sim Normal(0, 10) \\ \beta &\sim Normal(0, 10) \\ \sigma &\sim Normal_+(0, 50) \end{aligned} \quad (11.18)$$

$$\begin{pmatrix} u_{i,1} \\ u_{i,2} \end{pmatrix} \sim \mathcal{N} \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_u \right) \quad (11.19)$$

$$\Sigma_u = \begin{pmatrix} \tau_{u_1}^2 & \rho_u \tau_{u_1} \tau_{u_2} \\ \rho_u \tau_{u_1} \tau_{u_2} & \tau_{u_2}^2 \end{pmatrix} \quad (11.20)$$

$$\Sigma_w = \begin{pmatrix} \tau_{w_1}^2 & \rho_w \tau_{w_1} \tau_{w_2} \\ \rho_w \tau_{w_1} \tau_{w_2} & \tau_{w_2}^2 \end{pmatrix} \quad (11.21)$$

$$\begin{aligned} \tau_{u_1} &\sim Normal_+(0, 20) \\ \tau_{u_2} &\sim Normal_+(0, 20) \\ \rho_u &\sim LKJcorr(2) \end{aligned} \quad (11.21)$$

$$\begin{aligned}\tau_{w_1} &\sim \text{Normal}_+(0, 20) \\ \tau_{w_2} &\sim \text{Normal}_+(0, 20) \\ \rho_w &\sim \text{LKJcorr}(2)\end{aligned}\tag{11.22}$$

The translation to Stan (hierarchical_corr_by.stan) looks as follows:

```
data {
    int<lower=1> N;
    vector[N] signal;
    int<lower = 1> N_subj;
    int<lower = 1> N_item;
    vector[N] c_cloze;
    int<lower = 1, upper = N_subj> subj[N];
    int<lower = 1, upper = N_item> item[N];
}
parameters {
    real<lower = 0> sigma;
    vector<lower = 0>[2] tau_u;
    vector<lower = 0>[2] tau_w;
    real alpha;
    real beta;
    matrix[2, N_subj] z_u;
    matrix[2, N_item] z_w;
    cholesky_factor_corr[2] L_u;
    cholesky_factor_corr[2] L_w;
}
transformed parameters {
    matrix[N_subj, 2] u;
    matrix[N_item, 2] w;
    u = (diag_pre_multiply(tau_u, L_u) * z_u)';
    w = (diag_pre_multiply(tau_w, L_w) * z_w)';
}
model {
    target += normal_lpdf(alpha | 0, 10);
    target += normal_lpdf(beta | 0, 10);
    target += normal_lpdf(sigma | 0, 50) -
        normal_lccdf(0 | 0, 50);
```

```

target += normal_lpdf(tau_u | 0, 20) -
    2 * normal_lccdf(0 | 0, 20);
target += normal_lpdf(tau_w | 0, 20) -
    2* normal_lccdf(0 | 0, 20);
target += lkj_corr_cholesky_lpdf(L_u | 2);
target += lkj_corr_cholesky_lpdf(L_w | 2);
target += std_normal_lpdf(to_vector(z_u));
target += std_normal_lpdf(to_vector(z_w));
target += normal_lpdf(signal | alpha + u[subj, 1] + w[item, 1] +
    c_cloze .* (beta + u[subj, 2] + w[item, 2]), sigma);
}
generated quantities {
    corr_matrix[2] rho_u = L_u * L_u';
    corr_matrix[2] rho_w = L_w * L_w';
}

```

Add item as a number to the data and store it in a list:

```

df_eeg <- df_eeg %>%
    mutate(item = as.numeric(as.factor(item)))

ls_eeg <- list(
    N = nrow(df_eeg),
    signal = df_eeg$n400,
    c_cloze = df_eeg$c_cloze,
    subj = df_eeg$subj,
    item = df_eeg$item,
    N_subj = max(df_eeg$subj),
    N_item = max(df_eeg$item)
)

```

Fit the model:

```

hierarchical_corr_by <- system.file("stan_models",
    "hierarchical_corr_by.stan",
    package = "bcogsci"
)

```

```
fit_eeg_corr_by <- stan(
  file = hierarchical_corr_by,
  data = ls_eeg
)
```

Print the summary:

```
print(fit_eeg_corr_by,
  pars =
    c("alpha", "beta", "sigma", "tau_u", "tau_w", "rho_u", "rho_w")
)
```

	mean	2.5%	97.5%	n_eff	Rhat
## alpha	3.66	2.76	4.58	2395	1.00
## beta	2.32	0.99	3.65	4506	1.00
## sigma	11.49	11.19	11.80	6880	1.00
## tau_u[1]	2.21	1.56	3.01	1667	1.00
## tau_u[2]	1.54	0.11	3.35	1207	1.00
## tau_w[1]	1.52	0.85	2.18	1345	1.00
## tau_w[2]	2.21	0.19	4.24	1010	1.01
## rho_u[1,1]	1.00	1.00	1.00	NaN	NaN
## rho_u[1,2]	0.14	-0.61	0.78	4388	1.00
## rho_u[2,1]	0.14	-0.61	0.78	4388	1.00
## rho_u[2,2]	1.00	1.00	1.00	405	1.00
## rho_w[1,1]	1.00	1.00	1.00	NaN	NaN
## rho_w[1,2]	-0.42	-0.90	0.30	2562	1.00
## rho_w[2,1]	-0.42	-0.90	0.30	2562	1.00
## rho_w[2,2]	1.00	1.00	1.00	602	1.00

The correlations of interest are `rho_u[1,2]` and `rho_w[1,2]`; the summary above shows that the data are far too sparse to get tight estimates of these parameters: both posteriors are widely spread out.

This completes our review of hierarchical models and their implementation in Stan. The importance of coding directly a hierarchical model in Stan rather than using `brms` is that this increases the flexibility of the type of models that we can fit. In fact, we will see in chapter 18–21 that the

same “machinery” can be used to have hierarchical parameters in cognitive models.

11.2 Summary

In this chapter, we learned to fit the four standard types of hierarchical models that we encountered in earlier chapters:

- The by-subjects varying intercepts model.
- The by-subjects varying intercepts and varying slopes model without any correlation.
- The by-subjects varying intercepts and varying slopes model with correlation.
- The hierarchical model, with a full variance covariance matrix for both subjects and items.

We also learned some important and powerful tools for making the Stan models more efficient at sampling: the non-centered parameterization and the Cholesky factorization. One important takeaway was that if data are sparse, the posteriors will just reflect the priors. We saw examples of this situation when investigating the posteriors of the correlation parameters.

11.3 Further reading

Gelman and Hill (2007) provides a comprehensive introduction to Bayesian hierarchical models, although that edition does not use Stan but rather WinBUGS. Sorensen, Hohenstein, and Vasishth (2016) is a short tutorial on hierarchical modeling using Stan, especially tailored for psychologists and linguists. The Stan documentation, consisting of a User’s Guide and the Language Reference Manual are important starting points for going deeper into Stan programming: <https://mc-stan.org/users/documentation/>.

11.4 Exercises

Exercise 11.1. Log-normal model in Stan

Refit the Stroop example from section 5.2 in Stan (`df_stroop`).

Assume the following likelihood and priors:

$$rt_n \sim \text{LogNormal}(\alpha + u_{subj[n],1} + c_cond_n \cdot (\beta + u_{subj[n],2}), \sigma) \quad (11.23)$$

$$\begin{aligned} \alpha &\sim \text{Normal}(6, 1.5) \\ \beta &\sim \text{Normal}(0, .1) \\ \sigma &\sim \text{Normal}_+(0, 1) \end{aligned} \quad (11.24)$$

$$\begin{aligned} \tau_{u_1} &\sim \text{Normal}_+(0, 1) \\ \tau_{u_2} &\sim \text{Normal}_+(0, 1) \\ \rho_u &\sim \text{LKJcorr}(2) \end{aligned} \quad (11.25)$$

Exercise 11.2. By subject and by items hierarchical model with a log-normal likelihood.

We'll revisit the question “Are subject relatives easier to process than object relatives?” Refit in Stan the exercise 5.2.

Exercise 11.3. A hierarchical logistic regression with Stan.

We'll revisit the question “Is there a Stroop effect in accuracy?”. Refit in Stan the exercise 5.6.

Exercise 11.4. Distributional regression model of the effect of cloze probability on the N400

In section 5.1.6, we saw how to fit a distributional regression model. We might want to extend this approach to Stan. Fit the EEG data to a hierarchical model with by-subject and by-items varying intercept and slopes,

and in addition assume that the variance component of the model can vary by subject.

$$\begin{aligned} signal_n &\sim Normal(\alpha + u_{subj[n],1} + w_{item[n],1} + \\ & c_cloze_n \cdot (\beta + u_{subj[n],2} + w_{item[n],2}), \sigma_n) \\ \sigma_n &= \exp(\alpha_\sigma + u_{\sigma_{subj[n]}}) \end{aligned} \quad (11.26)$$

$$\begin{aligned} \alpha_\sigma &\sim Normal(0, log(50)) \\ u_\sigma &\sim Normal(0, \tau_{u_\sigma}) \\ \tau_{u_\sigma} &\sim Normal_+(0, 5) \end{aligned} \quad (11.27)$$

To fit this model, take into account that `sigma` is now a vector, and it's transformed parameter which depends on two parameters: `alpha_sigma` and the vector with `N_subj` elements `u_sigma`. In addition, `u_sigma` depends on the hyperparameter `tau_u_sigma` (τ_{u_σ}). (Using non-centered parametrization for `u_sigma` speeds up the model fit considerably).



12

Custom distributions in Stan

CHAPTER IN PREPARATION

Stan includes a large number of distributions, but what happens if we need a distribution that is not provided? In many cases, we can simply add it using the (ever growing number of) functions available in Stan language.

12.1 A change of variables with reciprocal normal distribution

In previous chapters, when faced with response times, we assumed a log-normal distribution. The log-normal distribution moves the inferences about the location parameter into a multiplicative frame (see Box 4.3). Another alternative, however, is to assume a “reciprocal”-normal distribution (rec-normal) of response times; this is to assume that the reciprocal of the response times are normally distributed.

$$\begin{aligned} 1/y &\sim \text{Normal}(\mu, \sigma) \\ y &\sim \text{RecNormal}(\mu, \sigma) \end{aligned} \tag{12.1}$$

An interesting aspect of the rec-normal is that it affords an interpretation of the location parameter in terms of rate or speed rather than time. As it happens with the log-normal, both the location, μ and scale, σ are not

in the same scale of the dependent variable, they are in the same scale as $1/y$.

By setting $\mu = .002$ and $\sigma = .0004$ we generate data that looks right-skewed and not unlike a distribution of response times, the code below shows some summary statistics of the generated data.

```
N <- 10000
mu <- .002
sigma <- .0004
rt <- 1/rnorm(N, mu, sigma)
c(mean(rt), sd(rt), min(rt), max(rt))
```

```
## [1] 524 124 284 2646
```

Figure 12.1 shows the distribution generated with the code shown above.

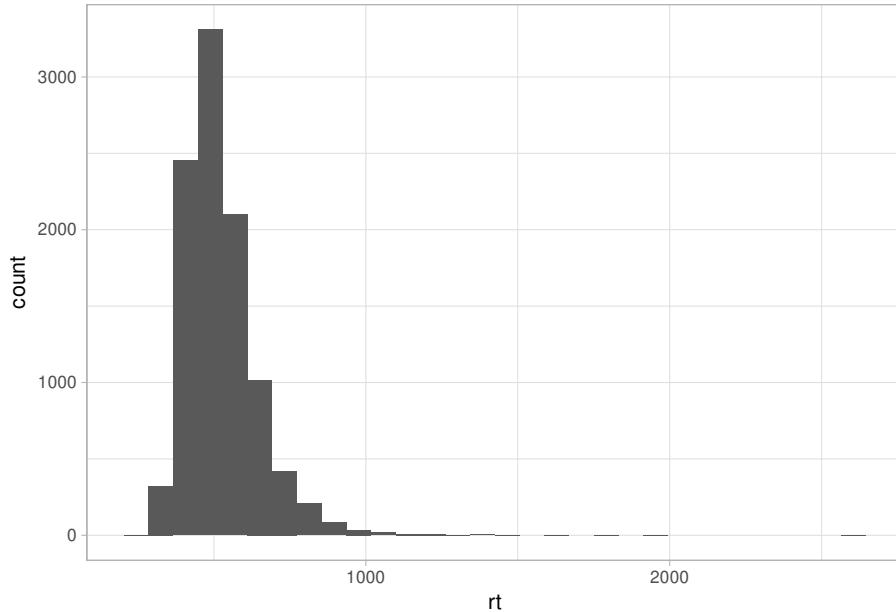


FIGURE 12.1: Distribution of synthetic data with a rec-normal distribution with parameters $\mu = .002$ and $\sigma = .0004$.

We could fit the response times in a simple model with a normal likeli-

hood, where we store the reciprocal of a vector of response times ($1/RT$) in the variable `recRT`, rather than the “raw” response times:

```
model {
  \\ priors go here
  target += normal_lpdf(recRT | mu, sigma);
}
```

One issue here is that the parameters of the likelihood, `mu` and `sigma` are going to be very far away from the unit scale (0.002 and 0.004 respectively). Due to the way Stan’s sampler is built, parameters that are too small (<<1) or too large (>>1) can cause convergence problems. A straightforward solution is to fit the normal distribution to parameters in reciprocal of seconds rather than milliseconds. We can do that using the `transformed parameters` block. Although we use `mu` and `sigma` to fit the data, the priors are on `mu_s` and `sigma_s`.

```
data {
  int<lower = 1> N;
  vector[N] recRT;
}
parameters {
  real mu_s;
  real<lower = 0> sigma_s;
}
transformed parameters {
  real mu = mu_s / 1000;
  real sigma = sigma_s / 1000;
}
model {
  target += normal_lpdf(mu_s | 2, 2);
  target += lognormal_lpdf(sigma_s | log(.5), 1);
  target += normal_lpdf(recRT | mu, sigma);
}
```

Fit and display the summary of the previous model:

```

normal_recrt <- system.file("stan_models",
                            "normal_recrt.stan",
                            package = "bcogsci")
fit_rec <- stan(normal_recrt, data = list(N = N, recRT = 1/rt))

print(fit_rec, pars = c("mu", "sigma"), digits = 4)

##          mean   2.5% 97.5% n_eff Rhat
## mu      0.0020 0.0020 0.0020  4518    1
## sigma  0.0004 0.0004 0.0004  3048    1

```

Is a rec-normal likelihood more appropriate than a log-normal likelihood? We won't be able to compare the models, because the dependent variables are different: we cannot compare reciprocal response times with untransformed response times. Model comparison with cross validation or Bayes factor can only compare models with the same dependent variables.

This requires us to do a *changes of variables*. The change of variables requires an adjustment to the (unnormalized log) posterior probability to account for the distortion caused by the transform.¹ The probability must be scaled by a *Jacobian* adjustment, which, in the univariate case as this one, is the absolute value of the derivative of the transform.²

$$\begin{aligned}
 p(RT_n | \mu, \sigma) &= Normal(1/RT_n | \mu, \sigma) \left| \frac{d}{dRT_n} 1/RT_n \right| = \\
 &= Normal(1/RT_n | \mu, \sigma) \cdot |-1/RT_n^2| = \\
 &= Normal(1/RT_n | \mu, \sigma) \cdot 1/RT_n^2
 \end{aligned} \tag{12.2}$$

If we omit the Jacobian adjustment, we are essentially fitting the wrong model, and how bad our results would be depend on the how large the Jacobian adjustment is for our model.

¹Not every transformation is valid, univariate changes of variables must be monotonic and differentiable, multivariate changes of variables must be injective and differentiable.

²In the multivariate case, it is equal to the absolute determinant of the Jacobian, the matrix of all its first-order partial derivatives, of the transform.

Because Stan works in log-space, rather than multiplying the Jacobian adjustment, we add its logarithm to the log-probability density ($\log(1/RT_n^2) = -2 \cdot \log(RT_n)$).

$$\log(Normal(1/RT_n|\mu, \sigma)) - 2 \cdot \log(RT_n) \quad (12.3)$$

We do this in Stan for RT_1 to RT_N as follows:

```
target += normal_lpdf(1 ./ RT | mu, sigma)
- sum(2 * log(RT));
```

It's important to consider when we write this in Stan that vectorized _lpdf functions are already summing together the log-PDF evaluated at each element of the vector, and `1 ./ RT` yields the reciprocal of each element of `RT`. If `1 ./ RT` is a vector of data with N elements $\{1/RT_1, 1/RT_2, \dots, 1/RT_N\}$ then doing `normal_lpdf(1 ./ RT | mu, sigma)` in Stan is equivalent to the following equation:

$$\begin{aligned} & \log(Normal(1/RT_1|\mu, \sigma)) + \dots + \log(Normal(1/RT_N|\mu, \sigma)) \\ &= \sum_n^N \log(Normal(1/RT_n|\mu, \sigma)) \end{aligned} \quad (12.4)$$

In contrast, because `2 * log(RT)` in Stan creates a vector such as $\{2 \cdot \log(RT_1); 2 \cdot \log(RT_2); \dots; 2 \cdot \log(RT_N)\}$, we need to sum the elements in this case.

Before we fit the model with the change of variables, we are going to truncate the distribution. We didn't encounter negative values in our synthetic data, but this was the case because the distribution was not too spread, that is, the scale was much smaller than the location ($\sigma \ll \mu$). In principle, however, since we could generate negative values, we truncate the underlying normal distribution to solve this problem. The reciprocal truncated Normal distribution has been argued as an appropriate model of response times, neural inter-spike intervals, and latency distributions of saccades in a simple optimality model in which reward is maximized to yield to an optimal response rate (Harris et al. 2014; Harris and Waddington 2012).

Because we have N observations, the truncation consists of adding `- N`

* `normal_lccdf(0 | mu, sigma)` to target. The complete model, `reconormal_rt.stan`, including the truncation looks as follows.

```
data {
    int<lower = 1> N;
    vector[N] RT;
}
parameters {
    real mu_s;
    real<lower = 0> sigma_s;
}
transformed parameters {
    real mu = mu_s / 1000;
    real sigma = sigma_s / 1000;
}
model {
    target += normal_lpdf(mu_s | 2, 2);
    target += lognormal_lpdf(sigma_s | log(.5), 1);
    target += normal_lpdf(1 ./ RT | mu, sigma)
        - N * normal_lccdf(0 | mu, sigma)
        - sum(2 * log(RT));
}
```

Fit the model to the data.

```
reconormal_rt <- system.file("stan_models",
                             "reconormal_rt.stan",
                             package = "bcogsci")
fit_rec <- stan(reconormal_rt, data = list(N = N, RT = rt))
```

Print the posterior.

```
print(fit_rec, pars = c("mu", "sigma"), digits = 4)
```

```
##          mean   2.5%  97.5% n_eff Rhat
## mu      0.0020 0.0020 0.0020  3823     1
## sigma  0.0004 0.0004 0.0004  2901     1
```

We get the same results as before, but now we could potentially compare this model to one that assumes a log-normal likelihood, for example.

An important question is the following: Does every operation on parameters requires a Jacobian adjustment? Essentially if we assign a distribution to a parameter and *then* we apply some mathematical operation we are doing a *transformation* and this does not require a Jacobian adjustment. This is the case when we only have parameters at the left of the pipe | in a PDF or PMF. Alternatively, if we apply a mathematical operation on a parameter *first*, and we assign a distribution to the modified parameter afterwards, we have a change of variables that requires a Jacobian adjustment. This is the case when we have some operation over a parameter (or a transformed parameter) at the left of the pipe | in the PDF or PMF. This is the reason for `1 ./ RT` requiring a Jacobian adjustment, but not for `mu_s`, `mu`, `sigma_s`, or `sigma_s` in the previous model.

As a last step, we can encapsulate our new distribution in a function, and we also create a random number generator function. We do this in a block called functions. To create a function, we need to specify the type of every argument and the type that the function returns.

As a simple example, if we would like a function to center a vector, we would do it as follows:

```
functions {
vector center(vector x) {
    vector centered;
    centered = centered - mean(centered);
    return centered;
}
data {
...
}
parameters {
...
}
model {
...
}
```

```
}
```

We want to create a log(PDF) function, similar to the native Stan functions that end in `_lpdf`. Our function will take as arguments a vector `RT`, and real numbers `mu` and `sigma`. In `_lpdf` functions, (some of) the arguments (e.g., `RT`, `mu`, and `sigma`) can be vectorized, but the output of the function is always a real number: the sum of the log(PDF) evaluated at every value of the random variable at the left of the pipe. As we show below, to do that we just move the content of `target` inside our new function. However, unlike native `lpdf` functions, we cannot overload our function, that is, we cannot have the same function name for different vectorizations in the arguments. See further reading for more about Stan functions.

For our custom random number generator function ending `_rng`, we have the added complication that the function is truncated. We opt for a not that efficient implementation for the sake of clarity. We simply generate random values from a normal distribution, we do the reciprocal transformation and if they are above zero we return them, otherwise we draw a new number. A more efficient, but more complex implementation can be found in section 18.10 of the Stan's user guide (Stan Development Team 2021).

The complete code can be found in `reconormal_rt_f.stan` and it is shown below.

```
functions {
    real reconormal_lpdf(vector RT, real mu, real sigma){
        real lpdf;
        lpdf = normal_lpdf(1 ./ RT | mu, sigma)
        - num_elements(RT) * normal_lccdf(0 | mu, sigma)
        - sum(2 * log(RT));
        return lpdf;
    }
    real reconormal_rng(real mu, real sigma){
        real pred_rt = 0;
        while (pred_rt <= 0)
            pred_rt = 1 / normal_rng(mu, sigma);
        return pred_rt;
    }
}
```

```

}

data {
    int<lower = 1> N;
    vector[N] RT;
}

parameters {
    real mu_s;
    real<lower = 0> sigma_s;
}

transformed parameters {
    real mu = mu_s / 1000;
    real sigma = sigma_s / 1000;
}

model {
    target += normal_lpdf(mu_s | 2, 2);
    target += lognormal_lpdf(sigma_s | log(.5), 1);
    target += rechnormal_lpdf(RT | mu, sigma);
}

generated quantities {
    real rt_pred[N];
    for (n in 1:N)
        rt_pred[n] = rechnormal_rng(mu, sigma);
}

```

Fit the model:

```

rechnormal_rt_f <- system.file("stan_models",
                               "rechnormal_rt_f.stan",
                               package = "bcogsci")
fit_rec <- stan(rechnormal_rt_f, data = list(N = N, RT = rt))

```

Print the summary:

```

print(fit_rec, pars = c("mu", "sigma"), digits = 4)

```

```

##          mean   2.5%  97.5% n_eff Rhat
## mu     0.0020 0.0020 0.0020  4246     1

```

```
## sigma 0.0004 0.0004 0.0004 2437     1
```

The model converged, but did it work? At the very least we expect that the synthetic data that we used to test the model shows a perfect fit when we posterior predictive checks. This is because we are fitting the data with the same function that we use to generate the data. Figure 12.2 shows that this is the case.

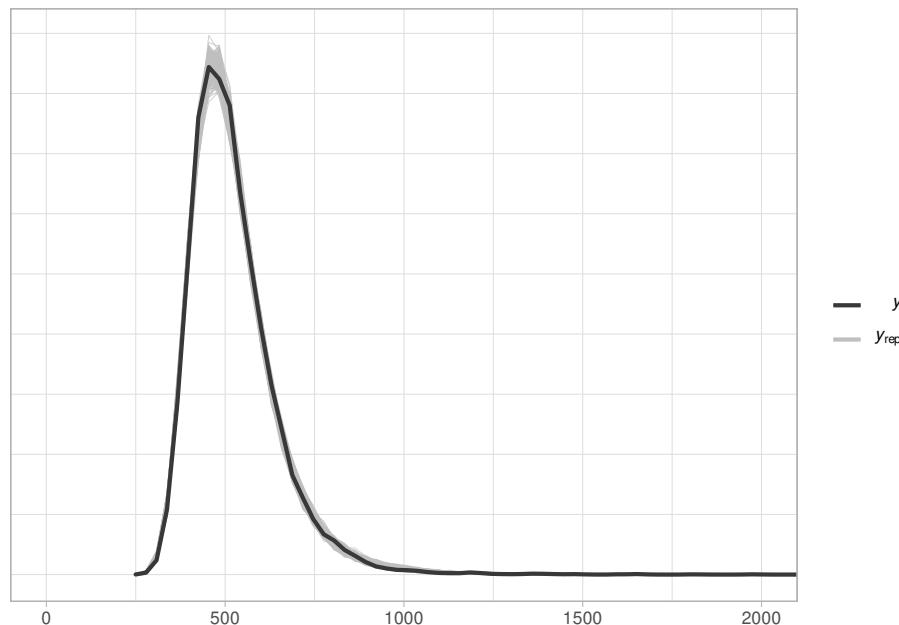


FIGURE 12.2: Posterior predictive check of `fit_rec` in comparison with the synthetic data.

Furthermore, the true values of the parameters μ and σ should be well inside their posterior distribution. We investigate this by plotting the true values of the parameters together with their posterior distributions using the function `mcmc_recover_hist` of `bayesplot` as shown below:

```
post_rec <- as.data.frame(fit_rec) %>%
  select("mu", "sigma")
mcmc_recover_hist(post_rec, true = c(mu, sigma))
```

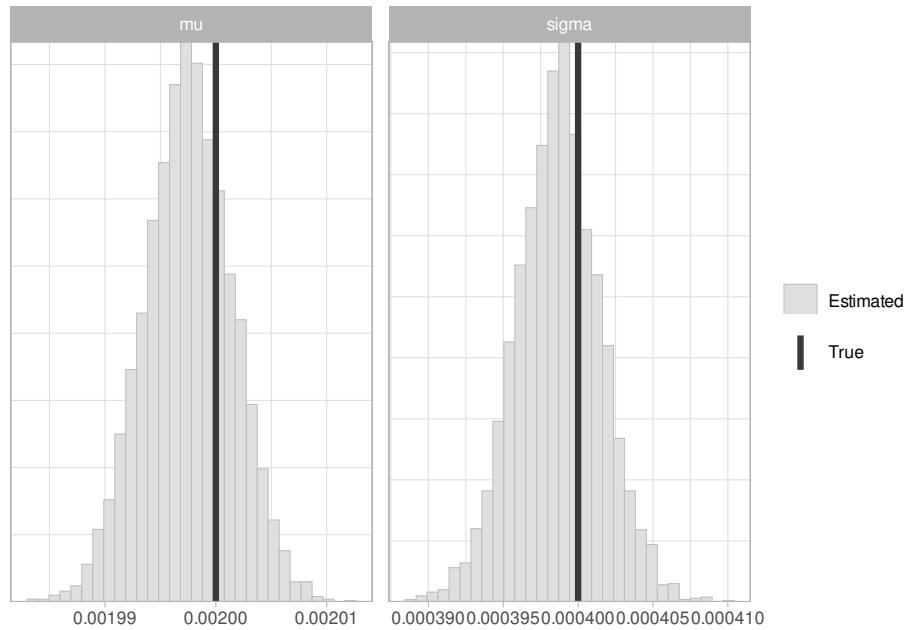


FIGURE 12.3: Posterior distributions of the parameters of `fit_rec` together with their true values.

12.1.1 Simulation based calibration

```
functions {
    real reccnormal_lpdf(vector RT, real mu, real sigma){
        real lpdf;
        lpdf = normal_lpdf(1 ./ RT | mu, sigma)
        - num_elements(RT) * normal_lccdf(0 | mu, sigma)
        - sum(2 * log(RT));
        return lpdf;
    }
    real reccnormal_rng(real mu, real sigma){
        real pred_rt = 0;
        while (pred_rt <= 0)
            pred_rt = 1 / normal_rng(mu, sigma);
        return pred_rt;
    }
}
data {
```

```
int<lower = 1> N;
real mu_s_loc;
real mu_s_scale;
real sigma_s_loc;
real sigma_s_scale;
}
transformed data { // these adhere to the conventions above
    vector[N] RT;
    real mu_ = normal_rng(mu_s_loc, mu_s_loc)/ 1000;
    real sigma_ = lognormal_rng(sigma_s_loc, sigma_s_scale) /1000;
    for(n in 1:N)
        RT[n] = rechnormal_rng(mu_, sigma_);
    }
parameters {
    real mu_s;
    real<lower = 0> sigma_s;
}
transformed parameters {
    real mu = mu_s / 1000;
    real sigma = sigma_s / 1000;
}
model {
    target += normal_lpdf(mu_s | mu_s_loc, mu_s_loc);
    target += lognormal_lpdf(sigma_s | sigma_s_loc, sigma_s_scale);
    target += rechnormal_lpdf(RT | mu, sigma);
}
generated quantities {
    vector[N] RT_ = RT;
    vector[2] pars_;
    int ranks_[2] = {mu > mu_, sigma > sigma_};
    vector[N] log_liks;
    pars_[1] = mu_;
    pars_[2] = sigma_;
    for(n in 1:N)
        log_liks[n] = rechnormal_lpdf([RT[n]]' | mu, sigma);
}
```

```

recnormal_sbc <- system.file("stan_models",
                               "recnormal_sbc.stan",
                               package = "bcogsci")

mod_recnormal_sbc <- stan_model(recnormal_sbc)
output_sbc <- sbc(mod_recnormal_sbc, data = list(N = 100,
                                              mu_s_loc = 2,
                                              mu_s_scale = 2,
                                              sigma_s_loc = log(.5),
                                              sigma_s_scale = 1),
                                              M = 5)

```

TO FINISH

12.2 A custom distribution: Re-implementing the exponential distribution manually

There are cases when one needs a distribution that is not included in Stan. Many times one can find the PDF (or a CDF) derived in a paper, and one needs to implement it manually by writing the log(PDF) in Stan language. Even though the exponential distribution is included in Stan, we demonstrate how we would include it step-by-step as if it weren't available. This example extends what is demonstrated in Stan's user guide (Stan Development Team 2021).

The exponential distribution is many times used to model waiting times until a certain event, and its PDF is the following

$$f(x, \lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0, \\ 0 & x < 0. \end{cases} \quad (12.5)$$

where parameter λ is often called the rate parameter and needs to be positive. A higher rate (λ) parameter leads to shorter waiting times on average, in fact the mean of the distribution is $1/\lambda$. The exponential distribution has the key property of being *memoryless*. The memoryless property means that the system “forgets” in which state it is, and the current state (still waiting, or the event happened) does not depend on how much time has elapsed already. As an example, the conditional probability that a certain event will happen in the next 100 ms is the same regardless if we have been waiting already 1000 ms, 10 ms, or 0 ms. Whereas the exponential distribution is not commonly used for modeling response times in cognitive science, it has been used in the past (Ashby and Townsend 1980; Ashby 1982). We focus on this distribution, however, due to its simple analytical form.

As a first step, we make our own version of the PDF in R, and we'll check that it integrates to 1 to verify that this is actually a proper distribution (and we haven't introduced a typo in the formula).

To avoid underflow, that is getting a zero instead of a very small number, we'll work in log-scale. This will also be useful when we implement this function in Stan, thus the log(PDF) is

$$\log(f(x, \lambda)) = \log(\lambda) - \lambda x \quad (12.6)$$

where $x > 0$.

Implement this in R with the same arguments as the `d*` function family: if `log = TRUE` the output is a log density, and call it `dexp2`.

```
dexp2 <- function(x, lambda = 1, log = FALSE){
  log_density <- log(lambda) - lambda * x
  if(log == FALSE) {
    exp(log_density)
  } else {
    log_density
  }
}
```

Verify that integrates to 1 for some parameter values (here $\lambda = 1$ and $\lambda = 20$):

```
dexp2_l1 <- function(x) dexp2(x, 1)
integrate(dexp2_l1, lower = 0, upper = Inf)

## 1 with absolute error < 0.000057

dexp2_l20 <- function(x) dexp2(x, 20)
integrate(dexp2_l20, lower = 0, upper = Inf)

## 1 with absolute error < 0.00000001
```

To test our function we'll also need to generate random values from the exponential distribution. If the quantile function of the distribution exists, the inverse transform sampling is a relatively straightforward way to get pseudo random numbers sampled from a target distribution.

TO COMPLETE EXPLANATION HERE

In this case, the quantile function has been derived already for us and it's:

$$-\log(1 - p)/\lambda \quad (12.7)$$

But it can also be derived by inverting the CDF; given the CDF of the exponential distribution:

$$F(x, \lambda) = 1 - e^{-\lambda x} \quad (12.8)$$

we need to find the value of x :

$$\begin{aligned}
 p &= 1 - e^{-\lambda x} \\
 p - 1 &= -e^{-\lambda x} \\
 -p + 1 &= e^{-\lambda x} \\
 \log(1 - p) &= -\lambda x \\
 -\log(1 - p)/\lambda &= x
 \end{aligned} \tag{12.9}$$

Write the quantile function (or inverse CDF) and random number generator for the exponential distribution in R:

```

qexp2 <- function(p, lambda = 1) {
  - log( 1 - p ) / lambda
}
rexp2 <- function(n, lambda = 1){
  u <- runif(n, 0, 1)
  qexp2(u, lambda)
}

```

The Stan model, `exponential.stan`, is relatively faithful to the R code, but follows Stan conventions: The function `exp_lpdf` stores the sum of the log(PDF) evaluated at each value of `x`, this would be analogous to do `sum(dexp2(x, lambda, log = TRUE))`; the function `exp_rng` implements a non-vectorized version of `rexp2` using the auxiliary function `exp_icdf` which is similar to `qexp2`.

```

functions {
  real exp_lpdf(vector x, real lambda){
    vector[num_elements(x)] lpdf = log(lambda) - lambda * x;
    return sum(lpdf);
  }
  real exp_icdf(real p, real lambda){
    return - log(1 - p) / lambda;
  }
  real exp_rng(real lambda){
    real u = uniform_rng(0, 1);
    return exp_icdf(u, lambda);
  }
}

```

```

}
data {
    int<lower = 1> N;
    vector[N] RT;
}
parameters {
    real<lower = 0> lambda;
}
model {
    target += normal_lpdf(lambda | 0, .1) -
        normal_lccdf(0 | 0, .1);
    target += exp_lpdf(RT | lambda);
}
generated quantities {
    real rt_pred[N];
    for (n in 1:N)
        rt_pred[n] = exp_rng(lambda);
}

```

We are now ready to generate synthetic data and fit the distribution in Stan.

Generate 1000 observations.

```

N <- 1000
lambda <- 1/200
rt <- rexp2(N, lambda)

exponential <- system.file("stan_models",
                            "exponential.stan",
                            package = "bcogsci")
fit_exp <- stan(exponential, data = list(N = N, RT = rt))

```

Print the summary:

```
print(fit_exp, pars = c("lambda"))
```

```
##          mean 2.5% 97.5% n_eff Rhat
## lambda 0.01    0  0.01  1415    1
```

We expect that the model will pass posterior predictive checks coded below and shown in Figure 12.4.

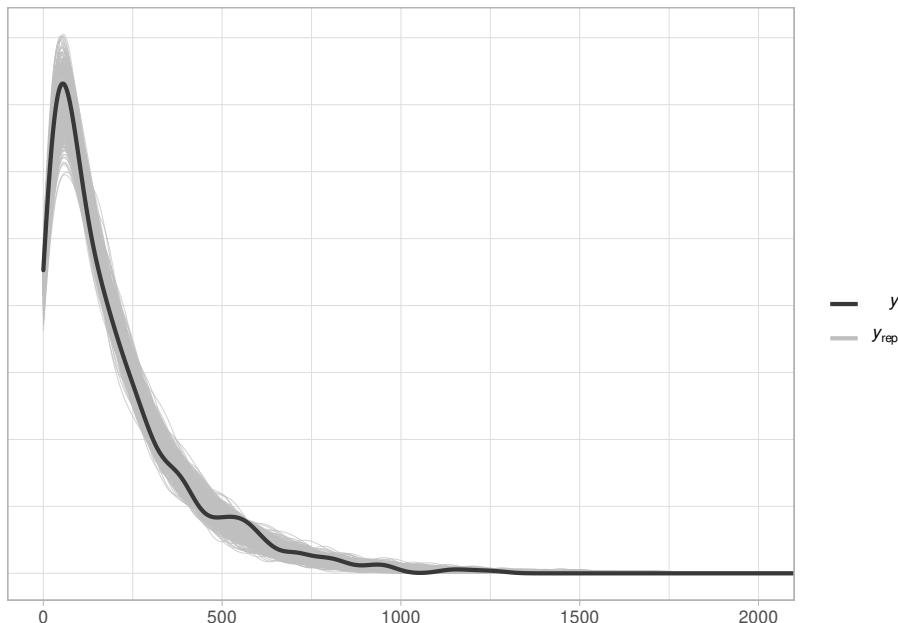


FIGURE 12.4: Posterior predictive check of `fit_exp` in comparison with the synthetic data.

Check that the true values of the parameter λ and is well inside its posterior distribution, and shown in Figure 12.5

```
post_exp <- as.data.frame(fit_exp) %>%
  select("lambda")
mcmc_recover_hist(post_exp, true = lambda)
```

SBC?

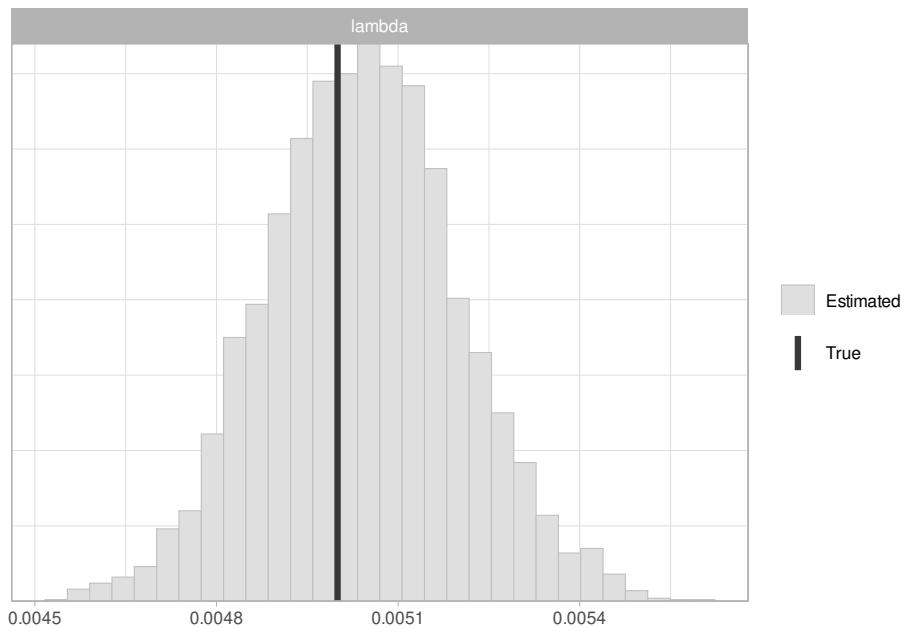


FIGURE 12.5: Posterior distributions of the parameters of `fit_exp` together with their true values.

12.3 Further reading

Jacobian adjustments are an ongoing source of confusion and there are several posts in blogs and study cases that try to clarify them:

- https://rstudio-pubs-static.s3.amazonaws.com/486816_440106f76c944734a7d4c84761e37388.html
- https://betanalpha.github.io/assets/case_studies/probability_theory.html#42_probability_density_functions
- <https://jsocolar.github.io/jacobians/#fn1>
- <https://mc-stan.org/documentation/case-studies/mle-params.html>

Custom functions in general and custom probability functions are treated in chapters 18 and 19 of the Stan user's guide (Stan Development Team 2021).



Part IV

Other useful models



13

Meta-analysis and measurement error models

In this chapter, we introduce two relatively underutilized models that are potentially very important for cognitive science: meta-analysis and measurement-error models.

Meta-analysis can be very informative when carrying out systematic reviews, and measurement-error models are able to take into account uncertainty in one's dependent or independent variable (or both). What's common to these two classes of model is that they both assume that the n -th measured data point y_n has an unknown true value of a parameter, say ζ_n (pronounced *zeta en*), that is measured with some uncertainty that can be represented by the standard error SE_n of the measurement y_n :

$$y_n \sim Normal(\zeta_n, SE_n)$$

In both classes of model, the goal is to obtain a posterior distribution of a latent parameter ζ which is assumed to generate the ζ_n , with some standard deviation τ :

$$\zeta_n \sim Normal(\zeta, \tau)$$

The main parameter of interest is usually ζ , but τ , which quantifies between-study variability or the noise in the measurement process, and the posterior distributions of ζ_n , can also be informative. The above model specification should remind you of the hierarchical models we saw in earlier chapters.

13.1 Meta-analysis

Once a number of studies have accumulated on a particular topic, it can be very informative to synthesize the data. Here is a commonly used approach: a random-effects meta-analysis.

13.1.1 A meta-analysis of similarity-based interference in sentence comprehension

The model is set up as follows. For each study n , let effect_n be the effect of interest, and let SE_n be the standard error of the effect. A concrete example from a recent meta-analysis is the effect of similarity-based interference in sentence comprehension (Jäger, Engelmann, and Vasishth 2017); when two nouns are more similar to each other, there is greater processing difficulty (i.e., longer reading times in milliseconds) when an attempt is made to retrieve one of the nouns to complete a linguistic dependency. The estimate of the effect and its standard error is the information we have from each study n .

First, we load the data, and we add an id that identifies each experiment.

```
data("df_sbi")
(df_sbi <- df_sbi %>%
  mutate(study_id = 1:n()))

## # A tibble: 12 x 4
##   publication      effect      SE study_id
##   <chr>          <int>    <int>     <int>
## 1 VanDyke07E1LoSem     13     30         1
## 2 VanDyke07E2LoSem     37     21         2
## 3 VanDyke07E3LoSem     20     11         3
## # ... with 9 more rows
```

We begin with the assumption that there is a true (unknown) effect ζ_n that lies behind each of these studies. Each of the observed effects has an uncertainty associated with it, SE_n . We can therefore assume that each observed effect, effect_n , is generated as follows:

$$\text{effect}_n \sim \text{Normal}(\zeta_n, SE_n) \quad (13.1)$$

Each study is assumed to have a different true effect ζ_n because each study will have been carried out under different conditions: in a different lab with different protocols and workflows, with different subjects, with different languages, with slightly different experimental designs, etc.

Further, each of the true underlying effects ζ_n has behind it some true unknown value ζ . We can write this as:

$$\zeta_n \sim \text{Normal}(\zeta, \tau) \quad (13.2)$$

τ is the between-study standard deviation; this expresses the assumption that there will be some variability between the true effects ζ_n , simply because of differences between the way the different experiments were conducted.

To summarize the model:

- effect_n is the observed effect (in this example, in milliseconds) in the n -th study.
- ζ_n is the true (unknown) effect in each study.
- ζ is the true (unknown) effect, to be estimated by the model.
- SE_n is the true standard deviation of the sampling distribution; each SE_n is estimated from the standard error available from the study n .
- The parameter τ represents between-study standard deviation.

We can construct a hierarchical model as follows:

$$\begin{aligned} \text{effect}_n &\sim \text{Normal}(\zeta_n, SE_n) \quad n = 1, \dots, N_{\text{studies}} \\ \zeta_n &\sim \text{Normal}(\zeta, \tau) \\ \zeta &\sim \text{Normal}(0, 100) \\ \tau &\sim \text{Normal}_+(0, 100) \end{aligned} \quad (13.3)$$

The priors are based on domain knowledge; we know that the interference effect is unlikely to be larger than 100 ms. Of course, a sensitivity analysis is necessary (but skipped here).

This model can be implemented in brms in a relatively straightforward

way as shown below. We show the Stan version later in the chapter, since it presents some interesting challenges that can be useful for the reader interested in deepening their Stan modeling knowledge.

13.1.1.1 brms version of the meta-analysis model

First, define the priors:

```
priors <- c(
  prior(normal(0, 100), class = Intercept),
  prior(normal(0, 100), class = sd)
)
```

Fit the model as follows. Because of our relatively uninformative priors and the few data points, the models of this chapter require us to tune the `control` parameter, increasing `adapt_delta` and `max_treedepth`.

```
fit_sbi <- brm(effect | resp_se(`SE`, sigma = FALSE) ~ 1
  + (1 | study_id),
  data = df_sbi,
  prior = priors,
  control = list(
    adapt_delta = .999,
    max_treedepth = 12
  )
)
```

The posterior of ζ and τ are summarized below as `Intercept` and `sd(Intercept)`.

```
fit_sbi

## ...
## Group-Level Effects:
## ~study_id (Number of levels: 12)
##             Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS
## sd(Intercept)   11.60      7.90     0.63    29.59 1.00      736
##                 Tail_ESS
```

```

## sd(Intercept)      1683
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     13.32      6.24     3.00    27.90 1.00     1551     1526
##
## Family Specific Parameters:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma       0.00      0.00     0.00     0.00   NA       NA       NA
##
## ...

```

The `sigma` parameter does not play any role in this model, but appears in the `brms` output anyway. In the model specification, `sigma` was explicitly removed by writing `sigma = FALSE`. For this reason, we can ignore that parameter in the model summary output above. Box 13.1 explains what happens if we set `sigma = TRUE`.

As theory predicts, the overall effect from these studies has a positive sign.

One advantage of such a meta-analysis is that the posterior can now be used as an informative prior for a future study. This is especially important when doing an analysis using Bayes factors. But this meta-analysis posterior could also be used as an informative prior in a future experiment; that would allow the researcher to build on what is known so far from published studies.

Box 13.1. What happens if we set `sigma = TRUE`?

If we set `sigma = TRUE`, we won't be able to get estimates for ζ_n , since they are implicitly handled. The model presented formally in (13.3) is equivalent to the following one in (13.4). A critical difference is that ζ_n does not appear anymore.

$$\begin{aligned}
 \text{effect}_n &\sim Normal(\zeta_n, \sqrt{\tau^2 + SE_n^2}) \\
 \zeta &\sim Normal(0, 100) \\
 \tau &\sim Normal_+(0, 100)
 \end{aligned} \tag{13.4}$$

This works because of the following property of normally distributed random variables:

If X and Y are two independent random variables, and

$$\begin{aligned} X &\sim \text{Normal}(\mu_X, \sigma_X) \\ Y &\sim \text{Normal}(\mu_Y, \sigma_Y) \end{aligned} \tag{13.5}$$

then, the sum of these two random variables

$$Z = X + Y \tag{13.6}$$

has the distribution

$$Z \sim \text{Normal}(\mu_X + \mu_Y, \sqrt{\sigma_X^2 + \sigma_Y^2}) \tag{13.7}$$

In our case, we set $U_n \sim \text{Normal}(0, SE_n)$ then, given that $\zeta_n \sim \text{Normal}(\zeta, \tau)$, we unpack $\text{Normal}(\zeta_n, SE_n)$ and we build an equation analogous to (13.7):

$$\text{effect}_n = U_n + \zeta_n \tag{13.8}$$

The sum of these two random variable in effect_n will be

$$\text{effect}_n \sim \text{Normal}(\zeta, \sqrt{SE^2 + \tau^2}) \tag{13.9}$$

We can fit this in `brms` as follows. Importantly, in this model specification, one should not include the `+ (1 | study_id)`, and the prior for τ should now be specified to `sigma`.

```

priors2 <- c(
  prior(normal(0, 100), class = Intercept),
  prior(normal(0, 100), class = sigma)
)
fit_sbi_sigma <- brm(effect | resp_se(`SE`, sigma = TRUE) ~ 1,
  data = df_sbi,
  prior = priors2,
  control = list(
    adapt_delta = .999,
    max_treedepth = 12
  )
)

```

There are slight differences with `fit_sbi` due to the different parametrization and the sampling process, but the results are very similar:

```

posterior_summary(fit_sbi_sigma,
  variable = c("b_Intercept", "sigma"))

##           Estimate Est.Error Q2.5 Q97.5
## b_Intercept     13.4      6.18 2.574 26.7
## sigma          12.0      7.59 0.776 29.8

```

If we are not interested in the underlying effects in each study, this parametrization of the meta-analysis can be faster and more robust (i.e., it has less potential convergence issues). A major drawback is that we can no longer display a forest plot as we do in Figure 13.1.

Another interesting by-product of a random-effects meta-analysis is the possibility of displaying a forest plot (Figure 13.1). A forest plot shows the meta-analytic estimate (the parameter `b_Intercept` in `brms`) alongside the original estimates effect_n (and their SE_n) and the posterior distributions of the ζ_n for each study (we reconstruct these estimates by adding `b_Intercept` to the parameters starting with `r_` in `brms`). The original estimates are the ones fed to the model as data and the posterior distributions of the ζ_n are calculated, as in previous hierarchical models, after the infor-

mation from all studies is pooled together. The ζ_n estimates are shrunken estimates of each study's (unknown) true effect, shrunk towards the grand mean ζ , and weighted by the standard error observed in each study n . The ζ_n for a particular study is shrunk more towards the grand mean ζ when the study's standard error is large (i.e., when the estimate is very imprecise). The code below shows how to build a forest plot step by step.

```
# First, change the format of the data
# so that it looks like the output of brms
df_sbi <- df_sbi %>%
  mutate(
    Q2.5 = effect - 2 * SE,
    Q97.5 = effect + 2 * SE,
    Estimate = effect,
    type = "original"
  )

# Extract the meta-analytical estimate:
df_Intercept <- posterior_summary(fit_sbi,
                                    variable = c("b_Intercept")) %>%
  as.data.frame() %>%
  mutate(publication = "M.A. estimate", type = "")

# For the pooled estimated effect of the individual studies
# we need to do the following:
# meta-analytical estimate (intercept) + adjustments:
# 1. extract the meta-analytical estimate:
intercept <- c(as_draws_df(fit_sbi)$b_Intercept)
# 2. extract the adjustments
adj_names <- paste0("r_study_id[",
                     unique(df_sbi$study_id),
                     ",Intercept]")
adj <- as.matrix(as_draws_df(fit_sbi)[adj_names])

## Warning: Dropping 'draws_df' class as required metadata was removed.
```

```
# 3. add them:
by_study <- as_tibble(intercept + adj)
# Summarize them by getting a table with the mean and the
# quantiles for each column and then binding them.
df_model <- lapply(by_study, function(x) {
  tibble(
    Estimate = mean(x),
    Q2.5 = quantile(x, .025),
    Q97.5 = quantile(x, .975)
  )
}) %>%
bind_rows() %>%
# Add a column to identify the posteriors,
# and another column to id the publication:
mutate(
  type = "adjusted",
  publication = df_sbi$publication
)
# Bind the original data,
# the adjusted estimates and the m.a. estimate:
bind_rows(df_sbi, df_model, df_Intercept) %>%
# Plot:
ggplot(aes(
  x = Estimate,
  y = publication,
  xmin = Q2.5,
  xmax = Q97.5,
  color = type
)) +
  geom_point(position = position_dodge(.5)) +
  geom_errorbar(position = position_dodge(.5)) +
  # Add the meta-analytical estimate and CrI
  geom_vline(xintercept = df_Intercept$Q2.5,
             linetype = "dashed",
             alpha = .3) +
  geom_vline(xintercept = df_Intercept$Q97.5,
```

```

    linetype = "dashed",
    alpha = .3) +
  geom_vline(xintercept = df_Intercept$Estimate,
    linetype = "dashed",
    alpha = .5) +
  scale_color_discrete(breaks = c("adjusted","original"))

```

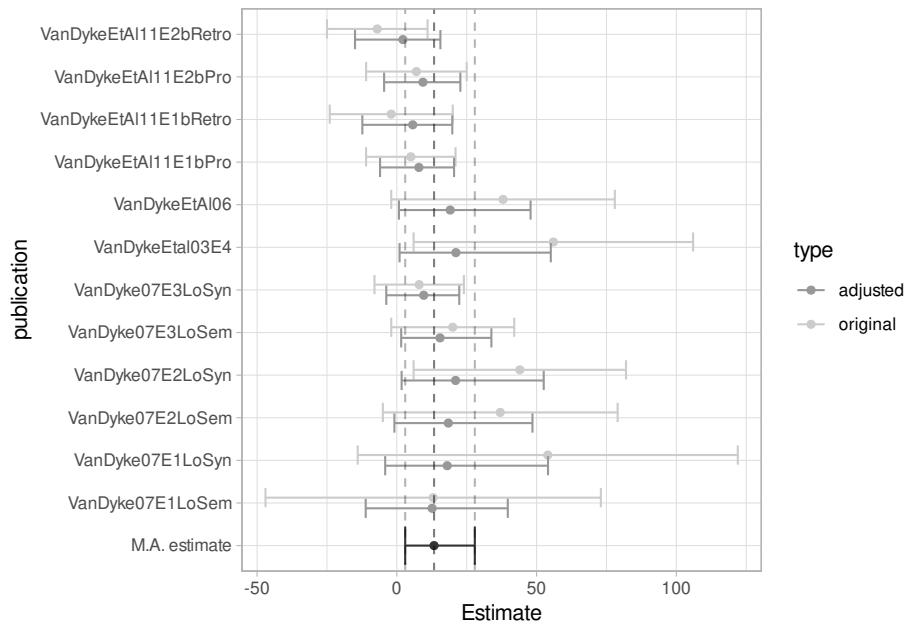


FIGURE 13.1: Forest plot showing the original and the adjusted estimates computed from each study from the random-effects meta-analysis. The error bars on the original estimates show 95% confidence intervals, and those on the adjusted estimates show 95% credible intervals.

It is important to keep in mind that a meta-analysis is always going to yield biased estimates as long as we have publication bias: if a field has a tendency to allow only “big news” studies to be published, then the literature that will appear in the public domain will be biased, and any meta-analysis based on such information will be biased. Despite this limitation, a meta-analysis is still a useful way to synthesize the known evidence; one just has to remember that the estimate from the meta-analysis is likely to be biased.

13.1.1.2 Stan version of the meta-analysis model

Even though `brms` can handle meta-analyses, fitting them in Stan allows us for more flexibility, which might be necessary in some cases. As a first attempt we could build a model that follows closely the formal specification given in (13.3).

```
data {
  int<lower=1> N;
  vector[N] effect;
  vector[N] SE;
  vector[N] study_id;
}
parameters {
  real zeta;
  real<lower = 0> tau;
  vector[N] zeta_n;
}
model {
  target += normal_lpdf(effect | zeta_n, SE);
  target += normal_lpdf(zeta_n | zeta, tau);
  target += normal_lpdf(zeta | 0, 100);
  target += normal_lpdf(tau | 0, 100)
    - normal_lccdf(0 | 0, 100);
}
```

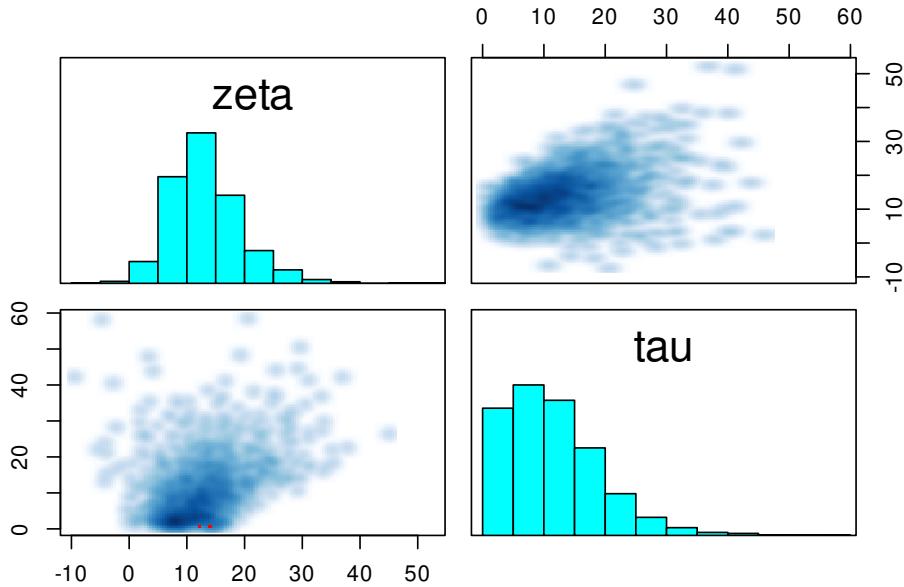
Fit the model as follows:

```
ma0 <- system.file("stan_models",
                    "meta-analysis0.stan",
                    package = "bcogsci")
ls_sbi <- list(N = nrow(df_sbi),
                effect = df_sbi$effect,
                SE = df_sbi$SE,
                study_id = df_sbi$study_id)
fit_sbi0 <- stan(
  file = ma0,
  data = ls_sbi,
```

```
control = list(  
    adapt_delta = .999,  
    max_treedepth = 12  
)  
  
## Warning: There were 3 divergent transitions after warmup. See  
##     http://mc-stan.org/misc/warnings.html#divergent-transitions-  
after-warmup  
## to find out why this is a problem and how to eliminate them.  
  
## Warning: There were 2 chains where the estimated Bayesian Fraction of Missing Information was lo  
## http://mc-stan.org/misc/warnings.html#bfmi-low  
  
## Warning: Examine the pairs() plot to diagnose sampling problems  
  
## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians ma  
## Running the chains for more iterations may help. See  
## http://mc-stan.org/misc/warnings.html#bulk-ess  
  
## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail  
## Running the chains for more iterations may help. See  
## http://mc-stan.org/misc/warnings.html#tail-ess
```

We see that there are warnings. We can use pairs plots as in 11.1.2 to uncover pathologies in the sampling. Here we see the samples of `zeta` and `tau` are highly correlated:

```
pairs(fit_sbi0, pars = c("zeta", "tau"))
```



We face a similar problem as we faced in 11.1.2, namely, the sampler cannot properly explore the neck of the funnel-shaped space because, because of the strong correlation between the parameters. The solution is, as in 11.1.2, a non-centered parametrization. We re-write (13.3) as follows:

$$\begin{aligned}
 z_n &\sim \text{Normal}(0, 1) \\
 \zeta_n &= z_n \cdot \tau + \zeta \\
 \text{effect}_n &\sim \text{Normal}(\zeta_n, SE_n) \\
 \zeta &\sim \text{Normal}(0, 100) \\
 \tau &\sim \text{Normal}_+(0, 100)
 \end{aligned} \tag{13.10}$$

This works because if $X \sim \text{Normal}(a, b)$ and $Y \sim \text{Normal}(0, 1)$, then $X = a + Y \cdot b$. You can re-visit chapter 11.1.2 for more details.

We translate (13.10) into Stan code as follows in `meta-analysis1.stan`:

```

data {
  int<lower=1> N;
  vector[N] effect;
  vector[N] SE;
  vector[N] study_id;
}

```

```

parameters {
    real zeta;
    real<lower = 0> tau;
    vector[N] z;
}
transformed parameters {
    vector[N] zeta_n = z * tau + zeta;
}
model {
    target += normal_lpdf(effect | zeta_n, SE);
    target += std_normal_lpdf(z);
    target += normal_lpdf(zeta | 0, 100);
    target += normal_lpdf(tau | 0, 100)
        - normal_lccdf(0 | 0, 100);
}

```

The model converges with values virtually identical to the ones of the `brms` model.

```

ma1 <- system.file("stan_models",
                    "meta-analysis1.stan",
                    package = "bcogsci")

```

```

fit_sbi1 <- stan(
    file = ma1,
    data = ls_sbi,
    control = list(
        adapt_delta = .999,
        max_treedepth = 12
    )
)

```

```

print(fit_sbi1, pars = c("zeta", "tau"))

```

```

##      mean 2.5% 97.5% n_eff Rhat
## zeta 13.1 2.70 27.1 1226     1
## tau  10.8 0.48 27.2   875     1

```

We can also reparametrize the model slightly differently, if we set $U_n \sim Normal(0, SE_n)$ then,

$$\text{effect}_n = U_n + \zeta_n \quad (13.11)$$

Then, given that $\zeta_n \sim Normal(\zeta, \tau^2)$,

$$\text{effect}_n \sim Normal(\zeta, \sqrt{SE^2 + \tau^2}) \quad (13.12)$$

See Box 13.1 if it's not clear why this reparametrization works.

This is equivalent to the `brms` model where `sigma = TRUE`. As with `brms`, we lose the possibility of estimating the posterior of the true effect of the individual studies.

We write this in Stan as follows in `meta-analysis2.stan`:

```
data {
    int<lower=1> N;
    vector[N] effect;
    vector[N] SE;
    vector[N] study_id;
}
parameters {
    real zeta;
    real<lower = 0> tau;
}
model {
    target += normal_lpdf(effect | zeta, sqrt(square(SE) + square(tau)));
    target += normal_lpdf(zeta | 0, 100);
    target += normal_lpdf(tau | 0, 100)
        - normal_lccdf(0 | 0, 100);
}
```

Fit it below:

```
ma2 <- system.file("stan_models",
                    "meta-analysis2.stan",
```

```

    package = "bcogsci")
fit_sbi2 <- stan(
  file = ma2,
  data = ls_sbi,
  control = list(adapt_delta = .9)
)
print(fit_sbi2, pars = c("zeta", "tau"))

##      mean 2.5% 97.5% n_eff Rhat
## zeta 13.5 2.70 29.1   615 1.01
## tau  11.9 0.67 30.8   841 1.01

```

13.2 Measurement-error models

Measurement error models deal with the situation where some predictor or the dependent variable, or both, are observed with measurement error. This measurement error could arise because a variable is an average (i.e., its standard error can also be estimated), or because we know that our measurement is noisy due to limitations of our equipment (e.g., delays in the signal from the keyboard to the motherboard, impedance in the electrodes in an EEG system, etc.).

13.2.1 Accounting for measurement error in a voice onset time model

As a motivating example, consider the following data. We have mean *voice onset time* (VOT) data from male and female native speakers of Mandarin, along with the standard errors of the mean VOT. VOT is the amount of time that elapses after a stop consonant is released until the vocal cords start vibrating. In addition, for each subject, their average vowel duration is calculated from a separate set of data; the standard error of the average vowel duration is also available.

Of interest here is the question whether the average time taken to pro-

duce a vowel affects VOT. See Vasishth, Nicenboim, et al. (2018) for more details; the data used here are from that paper.

```
data("df_VOTmandarin")
head(df_VOTmandarin)

##   subject meanVOT meanvdur sevdur seVOT c_meanvdur
## 1     F01    105.7      160    11.6  3.79     -5.312
## 2     F02     86.7      177    13.7  4.16     11.536
## 3     F03     97.8      166    14.7  4.62      0.879
## 4     F04     84.9      192    13.4  4.68     26.191
## 5     F05     84.6      164    13.0  4.49     -1.109
## 6     F06     98.6      210    13.8  4.10     44.113
```

At first glance, the relationship between mean vowel duration and centered mean VOT looks quite tantalizing; see Figure 13.2.

```
ggplot(df_VOTmandarin, aes(y = meanVOT, x = c_meanvdur)) +
  geom_point() +
  geom_smooth(method = "lm")
```

A simple linear model shows a somewhat weak association between mean VOT and centered mean vowel duration. The priors are relatively arbitrary but they are in the right order of magnitude given that we know that we are dealing with short times in milliseconds.

```
priors <- c(
  prior(normal(0, 200), class = Intercept),
  prior(normal(0, 10), class = b),
  prior(normal(0, 50), class = sigma)
)
fit_mvot <- brm(meanVOT ~ c_meanvdur,
  data = df_VOTmandarin,
  family = gaussian(),
  prior = priors
)
```

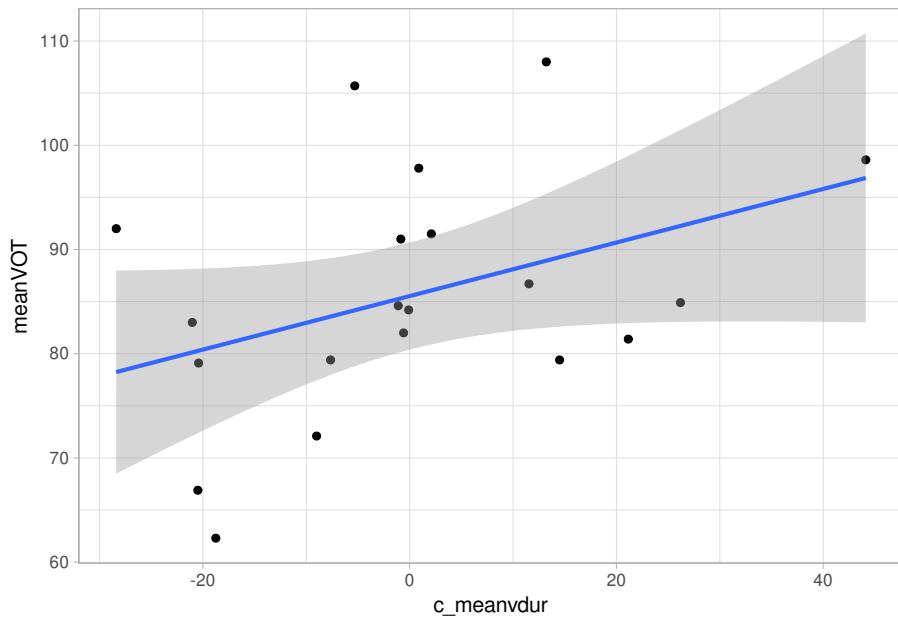


FIGURE 13.2: The relationship between mean voice onset time and centered mean vowel duration

Figure 13.3 shows the posterior distribution of the slope in this model. Most of the probability mass is positive, suggesting a possible (small) effect of vowel duration on mean voice onset time. The code to generate the figure appears below. We use `brms` function `mcmc_plot`, and we set `regex = TRUE` to indicate that the `variable` argument should be treated as a regular expression (in this case, it will capture all the variables starting with `b_c_meanvdur`).

```
mcmc_plot(fit_mvot, variable = "^b_c_meanvdur", regex = TRUE, type = "hist")
```

In many practical research problems, researchers will often take average measurements like these and examine the correlation between them. However, each of those data points is being measured with some error (uncertainty), and that uncertainty is not being taken into account in the model. Ignoring this uncertainty leads to over-enthusiastic inferences. A measurement-error model takes this uncertainty into account.

The measurement error model is stated as follows. There is assumed to

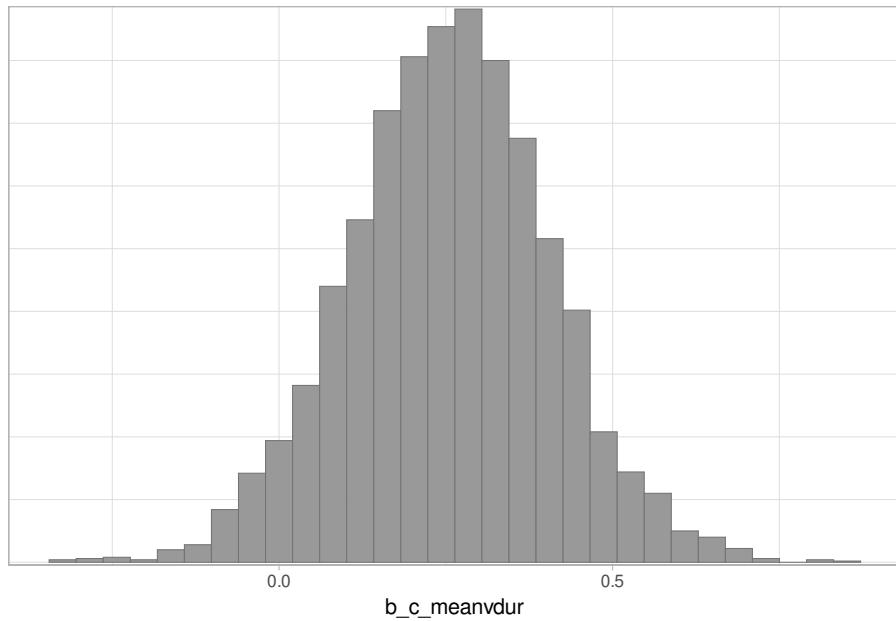


FIGURE 13.3: The posterior distribution of the slope in the linear model, modeling the effect of centered mean vowel duration on mean voice onset time.

be a true unobserved value $y_{n,TRUE}$ for the dependent variable, and a true unobserved value $x_{n,TRUE}$ for the predictor, where n is indexing the observation number. The observed values y_n and the predictor x_n are assumed to be generated with some error:

$$\begin{aligned} y_n &\sim \text{Normal}(y_{n,TRUE}, SE_y) \\ x_n &\sim \text{Normal}(x_{n,TRUE}, SE_x) \end{aligned} \tag{13.13}$$

The regression is fit to the (unknown) *true* values of the dependent and independent variables:

$$y_{n,TRUE} \sim \text{Normal}(\alpha + \beta x_{n,TRUE}, \sigma) \tag{13.14}$$

In addition, there is also an unknown standard deviation (standard error) of the latent unknown means that generate the underlying vowel duration means. I.e., we assume that each of the observed centered vowel du-

rations has some true underlying mean, χ with some variability around some single true mean, τ . This is very similar to the meta-analysis situation we saw earlier: $\zeta_n \sim Normal(\zeta, \tau)$, where ζ_n was the true latent mean of each study, and ζ was the (unknown) true value of the parameter, and τ was the between-study variability.

$$x_{n,TRUE} \sim Normal(\chi, \tau) \quad (13.15)$$

The goal of the modeling is to obtain posterior distributions for the intercept and slope α and β (and the residual error standard deviation σ).

We need to decide on priors for all the parameters now. Since our measurements of vowel duration are centered, we assume that their true value is also close to zero, but in general, we set relatively vague priors, which can still be considered *regularizing priors*. Given how much we need to increase `adapt_delta` and `max_treedepth`, our posterior is hard to explore, and more precise priors would probably have been better (based on expert opinion from a phonology researcher). These priors, however, are used just to demonstrate the point that in situations where not much is known about a research question, one could use such vague priors.

$$\begin{aligned} \alpha &\sim Normal(50, 50) \\ \beta &\sim Normal(0, 10) \\ \chi &\sim Normal(0, 10) \\ \sigma &\sim Normal_+(0, 30) \\ \tau &\sim Normal_+(0, 30) \end{aligned} \quad (13.16)$$

13.2.1.1 brms version of measurement error in voice onset time model

```
priors_me <- c(
  prior(normal(50, 50), class = Intercept),
  prior(normal(0, 10), class = b),
  prior(normal(0, 10), class = meanme),
  prior(normal(0, 30), class = sdme),
  prior(normal(0, 30), class = sigma)
)
```

Here the parameter with class `meanme` and `sdme` refer to the unknown mean and standard deviation (standard error) of the latent unknown means that generate the underlying vowel duration means, χ and τ in (13.16). Once we decide on the priors, we use `resp_se(.)` with `sigma = TRUE` (i.e, we don't estimate $y_{n,TRUE}$ explicitly) and we use `me(meandur, sevdur)` to indicate that the dependent variable `meandur` is measured with error and `sevdur` is its SE. Because of our relatively uninformative priors and the few data points, this model requires us to both tune the `control` parameter increasing `adapt_delta` and `max_treedepth`, and increase the number of iterations.

```
fit_mvotme <- brm(meanVOT | resp_se(seVOT, sigma = TRUE) ~
  me(c_meanvdur, sevdur),
  data = df_VOTmandarin,
  family = gaussian(),
  prior = priors_me,
  control = list(max_treedepth = 15,
                 adapt_delta = .99999),
  iter = 5000
)
```

```
fit_mvotme
```

```
## ...
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS
## Intercept     85.40      4.69    77.69    93.97 1.00     2639
## mec_meanvdursevdur   0.61      1.24    -0.45     3.00 1.01      923
##           Tail_ESS
## Intercept      1300
## mec_meanvdursevdur    875
##
## Family Specific Parameters:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     9.00      3.22     2.07    15.27 1.00     1578     1127
##
```

```
## ...
```

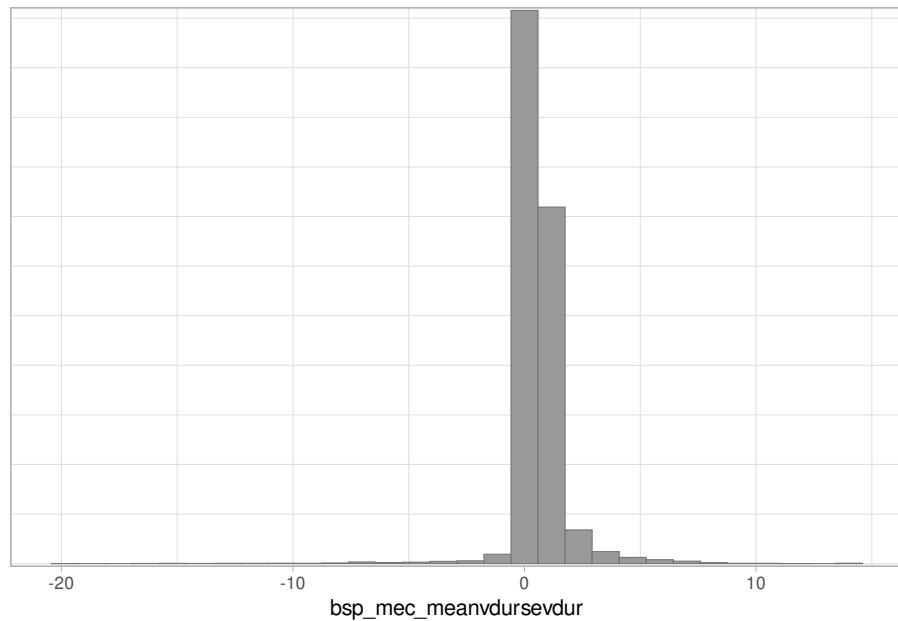


FIGURE 13.4: The posterior distribution of the slope in the measurement error model, modeling the effect of centered mean vowel duration on mean voice onset time.

The posterior for the slope is plotted in Figure 13.4; this figure shows that the association between VOT and vowel duration is much weaker once measurement error is taken into account: the posterior is much more uncertain (much more widely distributed) than in the simple linear model we fit above (compare Figures 13.4 and ??). The conclusion here cannot be that there is no association between mean VOT and mean vowel duration; as often happens, we just don't know enough to make a conclusive claim. As data analysts, we can and should be open about the fact that more data would be needed to make a conclusive claim.

13.2.1.2 Stan version of measurement error in voice onset time model

As it happened when we modeled the meta-analysis, the main difficulty for modeling measurement error models directly in Stan is that we need to reparametrize the models to avoid correlations between samples of dif-

ferent parameters. The two changes that we need to do to the parametrization of our model presented in (13.16) are the following.

1. Sample from an auxiliary parameter z_n rather than directly from $x_{n,TRUE}$, as we did in (13.10):

$$\begin{aligned} z_n &\sim \text{Normal}(0, 1) \\ x_{n,TRUE} &= z_n \cdot \tau + \chi \\ x_n &\sim \text{Normal}(x_{n,TRUE}, SE_x) \end{aligned} \tag{13.17}$$

2. Don't model $y_{n,TRUE}$ explicitly as in (??); rather take into account the SE and the variation on $y_{n,TRUE}$ in the following way:

$$y_n \sim \text{Normal}(\alpha + \beta x_{n,TRUE}, \sqrt{SE_y^2 + \sigma^2}) \tag{13.18}$$

We are now ready to write this in Stan; the code is in the model called `me.stan`:

```
data {
    int<lower=1> N;
    vector[N] x;
    vector[N] SE_x;
    vector[N] y;
    vector[N] SE_y;
}
parameters {
    real alpha;
    real beta;
    real chi;
    real<lower = 0> sigma;
    real<lower = 0> tau;
    vector[N] z;
}
transformed parameters {
    vector[N] x_true = z * tau + chi;
}
model {
```

```

target += normal_lpdf(x | x_true, SE_x);
target += normal_lpdf(y | alpha + beta * x_true,
                      sqrt(square(SE_y) + square(sigma)));
target += std_normal_lpdf(z);
target += normal_lpdf(alpha | 50, 50);
target += normal_lpdf(beta | 0, 10);
target += normal_lpdf(chi | 0, 10);
target += normal_lpdf(sigma| 0, 30)
    - normal_lccdf(0 | 0, 30);
target += normal_lpdf(tau | 0, 30)
    - normal_lccdf(0 | 0, 30);
}

```

Fit it below. As with `brms`, we are required to tune the `control` parameter and to increase the number of iterations:

```

me <- system.file("stan_models",
                  "me.stan",
                  package = "bcogsci")
ls_me <- list(N = nrow(df_VOTmandarin),
               y = df_VOTmandarin$meanVOT,
               SE_y = df_VOTmandarin$seVOT,
               x = df_VOTmandarin$c_meanvdur,
               SE_x = df_VOTmandarin$sevdur)
fit_mvotme_stan <- stan(
  file = me,
  data = ls_me,
  control = list(max_treedepth = 15,
                 adapt_delta = .99999),
  iter = 5000
)

```

```
print(fit_mvotme_stan , pars = c("alpha", "beta", "sigma"))
```

```

##          mean 2.5% 97.5% n_eff Rhat
## alpha  85.55 78.37  94.1   915 1.00
## beta   0.64 -0.57   3.2   326 1.02

```

```
## sigma 8.93 1.19 15.6 1062 1.00
```

13.3 Summary

This chapter introduced two statistical tools that are potentially of great relevance to cognitive science: random-effects meta-analysis and measurement error models. Despite the inherent limitations of meta-analysis, these should be used routinely to accumulate knowledge through systematic evidence synthesis. Measurement errors can also prevent over-enthusiastic conclusions that are often made based on noisy data.

13.4 Further reading

For some examples of Bayesian meta-analyses in psycholinguistics, see Vasishth et al. (2013), Jäger, Engelmann, and Vasishth (2017), Nicenboim, Roettger, and Vasishth (2018), Nicenboim, Vasishth, and Rösler (2020a), and Bürki et al. (2020). A frequentist meta-analysis of priming effects in psycholinguistics appears in Mahowald et al. (2016). Sutton et al. (2012) and Higgins and Green (2008) are two useful general introductions that discuss systematic reviews, meta-analysis, and evidence synthesis; these two references are from medicine, where meta-analysis is more widely used than in cognitive science. A potentially important article for meta-analysis introduces a methodology for modeling bias, to adjust for different kinds of bias in the data (Turner et al. 2008).

13.5 Exercises

Exercise 13.1. Meta-analysis data of picture-word interference data

Load the following data set:

```
data("df_buerki")
head(df_buerki)

##           study   d    se study_id
## 1 Collina 2013 Exp.1 a  24 13.09     1
## 2 Collina 2013 Exp.1 b -25 17.00     2
## 3 Collina 2013 Exp.2  46 22.79     3
## 4 Mahon 2007 Exp.1   17 12.24     4
## 5 Mahon 2007 Exp.2   57 13.96     5
## 6 Mahon 2007 Exp. 4  17  8.01     6
```

```
df_buerki <- subset(df_buerki, se > 0.60)
```

The data are from Bürki et al. (2020). We have a summary of the effect estimates (d) and standard errors (se) of the estimates from 162 published experiments on a phenomenon called **semantic picture-word interference**. We removed an implausibly low SE in the code above, but the results don't change regardless of whether we keep them or not, because we have a lot of studies.

In this experimental paradigm, subjects are asked to name a picture while ignoring a distractor word (which is either related or unrelated to the picture). The word can be printed on the picture itself, or presented auditorily. The dependent measure is the response latency, or time interval between the presentation of the picture and the onset of the vocal response. Theory says that distractors that come from the same semantic category as the picture to be named lead to a slower response than when the distractor comes from a different semantic category.

Carry out a random effects meta-analysis using brms and display the posterior distribution of the effect, along with the posterior of the between study standard deviation.

Choose $\text{Normal}(0,100)$ priors for the intercept and between study sd parameters. You can also try vague priors (sensitivity analysis). Examples would be:

- $\text{Normal}(0,200)$

- $\text{Normal}(0, 400)$

Exercise 13.2. Measurement error model for English VOT data

Load the following data:

```
data("df_VOTenglish")
head(df_VOTenglish)
```

```
##   subject meanVOT seVOT meanvdur sevdur
## 1     F01    108.1  4.56      171   11.7
## 2     F02     92.5  4.62      189   12.7
## 3     F03     82.6  3.13      171   10.0
## 4     F04     88.3  3.21      168   11.8
## 5     F05     94.6  3.67      166   15.0
## 6     F06     75.9  3.70      176   12.9
```

You are given mean voice onset time (VOT) data (with SEs) in milliseconds for English, along with mean vowel durations (with SEs) in milliseconds. Fit a measurement-error model investigating the effect of mean vowel duration on mean VOT duration. First plot the relationship between the two variables; does it look like there is an association between the two?

Then use brms with measurement error included in both the dependent and independent variables. You can use the same priors as in the chapter, but also do a sensitivity analysis to check the influence of the priors on the posteriors of the relevant parameters.



14

SAT



Part V

Model comparison and hypothesis testing



15

Introduction to model comparison

A key goal of cognitive science is to decide which theory under consideration accounts for the experimental data better. This can be accomplished by implementing the theories (or some aspects of them) as Bayesian models and comparing their predicting power. Thus, model comparison and hypothesis testing are closely related ideas. There are two Bayesian perspectives on model comparison: a *prior* predictive perspective based on the Bayes factor using marginal likelihoods, and a *posterior* predictive perspective based on cross-validation. The main characteristic difference between the prior predictive approach (Bayes factor) versus the posterior predictive approach (cross-validation) is the following: The Bayes factor examines how well the model (*prior and likelihood*) explains the experimental data. By contrast, the posterior predictive approach assesses model predictions for held-out data after seeing most of the data.

That is, the predictive accuracy of the Bayes factor is only based on its prior predictive distribution. In Bayes factor analyses, the prior model predictions are used to evaluate the support that the data give to the model. By contrast, in cross-validation, the model is fit to a large subset of the data (i.e., the training data). The posterior distributions of the parameters of this fitted model are then used to make predictions for held-out or validation data, and model fit is assessed on this subset of the data. Typically, this process is repeated several times, until the entire dataset is assessed as held-out data. This attempts to assess whether the model will generalize to truly new, unobserved data. Of course, the held-out data is usually not “truly new” because it is part of the data that was collected, but at least it is data that the model has not been exposed to. That is, the predictive accuracy of cross-validation methods is based on how well the posterior predictive distribution that is fit to most of the data (i.e., the training data) characterizes out-of-sample data (i.e., the test or held-out data).

The prior predictive distribution is obviously highly sensitive to the priors: it evaluates the probability of the observed data under prior assumptions. By contrast, the posterior predictive distribution is less dependent on the priors because the priors are combined with the likelihood (and are thus less influential, given sufficient data) before making predictions for held-out validation data.

Jaynes (2003, chap. 20) compares these two perspectives to “a cruel realist” and “a fair judge”: According to Jaynes, Bayes factor adopts the posture of a cruel realist, who “judge[s] each model taking into account the prior information we actually have pertaining to it; that is, we penalize a model if we do not have the best possible prior information about its parameters, although that is not really a fault of the model itself.” In contrast, cross-validation adopts the posture of a scrupulously fair judge, “who insists that fairness in comparing models requires that each is delivering the best performance of which it is capable, by giving each the best possible prior probability for its parameters (similarly, in Olympic games we might consider it unfair to judge two athletes by their performance when one of them is sick or injured; the fair judge might prefer to compare them when both are doing their absolute best).”

Regardless of whether we use Bayes factor or cross-validation or any other method for model comparison, there are several important points that one should keep in mind:

1. Although the objective of model comparison might ultimately be to find out which of the models under consideration generalizes better, this generalization can only be done well within the range of the observed data (see Vehtari and Lampinen 2002; Vehtari and Ojanen 2012). That is, if one hypothesis implemented as the model \mathcal{M}_1 shows to be superior to a second hypothesis, implemented as the model \mathcal{M}_2 , according to Bayes factor and/or cross-validation and evaluated with young Western University student population, this doesn’t mean that \mathcal{M}_1 will be superior to \mathcal{M}_2 when it is evaluated with a broader population (and in fact it seems that many times it won’t, see Henrich, Heine, and Norenzayan 2010). However, if we can’t generalize even within the range of the observed data (e.g., University

students), there is no hope to generalize outside of that range (e.g., non-University students). Navarro (2019) argues that one of the most important functions of a model is to encourage directed exploration of new territory; our view is that this makes sense only if historical data is also accounted for. In practice, what that means for us is that evaluating a model's performance should be carried out using historical benchmark data in addition to any new data one has; just using isolated pockets of new data to evaluate a model is not convincing. For an example from psycholinguistics of model evaluation using historical benchmark data, see Engelmann, Jäger, and Vasishth (2020).

2. Model comparison can provide a quantitative way to evaluate models, but this cannot replace understanding the qualitative patterns in the data (see, e.g., Navarro 2019). A model can provide a good fit by behaving in a way that contradicts our substantive knowledge. For example, Lissón et al. (2021) examine two computational models of sentence comprehension. One of the models yielded higher predictive accuracy when the parameter that is related to the probability of correctly comprehending a sentence was higher for impaired subjects (individuals with aphasia) than for the control population. This contradicts domain knowledge—impaired subjects are generally observed to show worse performance than unimpaired control subjects—and led to a re-evaluation of the model.
3. Model comparison is based on finding the most “useful model” for characterizing our data, but neither the Bayes factor or cross-validation (nor any other method that we are aware of) guarantees selecting the model closest to the truth (even with enough data). This is related to our previous point: A model that’s closest to the true generating data process is not guaranteed to produce the best (prior or posterior) predictions, and a model with a clearly wrong generating data process is not guaranteed to produce poor (prior or posterior) predictions (see Wang and Gelman 2014, for an example with cross-validation; and Navarro 2019 for a toy example with Bayes factor).
4. One should also check that the precision (the uncertainty) of the

data being modeled is high; if an effect is being modeled that has high uncertainty in the data, then any measure of model fit can be uninformative because we don't have accurate estimates of the effect. In the Bayesian context, this implies that the prior predictive and posterior predictive distributions of the effects generated by a model should be theoretically plausible and reasonably constrained, and the target data being modeled should have as high precision as possible. Later in this part of the book, we will discuss the adverse impact of imprecision in the data on model comparison. We will show that, in the face of low precision, we generally won't learn much from model comparison.

15.1 Further reading

Roberts and Pashler (2000) and Pitt and Myung (2002) argue for the need of going beyond “a good fit” (this is a good posterior predictive check in the context of Bayesian data analysis) and argue for the need of model comparison and a focus on measuring the generalizability of a model. Navarro (2019) deals with the problematic aspects of model selection in the context of psychological literature and cognitive modeling. Fabian Dablander's blog post, <https://fabiandablander.com/r/Law-of-Practice.html>, shows a very clear comparison between Bayes factor and PSIS-LOO-CV.

16

Bayes factors

This chapter is based on a longer manuscript available on arXiv: Schad et al. (2021). Bayesian approaches provide tools for different aspects of data analysis. A key contribution of Bayesian data analysis is that it provides probabilistic ways to quantify the evidence that data provide in support of one model or another. Models provide ways to implement scientific hypotheses; as a consequence, model comparison and hypothesis testing a closely related. There are two kinds of hypotheses: point hypotheses, which hypothesize that a model parameter has as specific point value - such as e.g., zero. By contrast, range hypotheses specify that a parameter exists and is needed to explain the data, but they do not specify the parameter value, which can be estimated from data. Bayesian hypothesis testing of range hypotheses is implemented using Bayes factors (Rouder, Haaf, and Vandekerckhove 2018; Schönbrodt and Wagenmakers 2018; Wagenmakers et al. 2010; Kass and Raftery 1995; Quentin F Gronau et al. 2017a; Jeffreys 1939), which quantify evidence in favor of one statistical (or computational) model over another. Point hypotheses are the norm in frequentist hypothesis testing, and can be implemented in Bayesian analyses using posterior density ratios. This chapter will focus on Bayes factors as the way to compare models and to obtain evidence about (range) hypotheses.

There are subtleties associated with Bayes factors that are not widely appreciated. For example, the results of Bayes factor analyses are highly sensitive to and crucially depend on prior assumptions about model parameters (we will illustrate this below), which can vary between experiments/research problems and even differ subjectively between different researchers. Many authors use or recommend so-called default prior distributions, where the prior parameters are fixed, and are independent of the scientific problem in question (Hammerly, Staub, and Dillon 2019; Navarro 2015). However, default priors result in an overly simplistic per-

spective on Bayesian hypothesis testing, and can be misleading. For this reason, even though leading experts in the use of Bayes factor, such as Rouder et al. (2009), often provide default priors for computing Bayes factors, they also make it clear that: “simply put, principled inference is a thoughtful process that cannot be performed by rigid adherence to defaults” (Rouder et al. 2009, 235). However, this observation does not seem to have had much impact on how Bayes factors are used in, e.g., psychology; the use of default priors when computing Bayes factor seems to be widespread.

Given the key influence of priors on Bayes factors, defining priors becomes a central issue when using Bayes factors. The priors determine which models will be compared.

In this chapter, we demonstrate how Bayes factors should be used in practical settings in cognitive science. In doing so, we demonstrate the strength of this approach and some important pitfalls that researchers should be aware of.

16.1 Hypothesis testing using the Bayes factor

16.1.1 Marginal likelihood

Bayes' rule can be written with reference to a specific statistical model \mathcal{M}_1 .

$$p(\Theta \mid y, \mathcal{M}_1) = \frac{p(y \mid \Theta, \mathcal{M}_1)p(\Theta \mid \mathcal{M}_1)}{p(y \mid \mathcal{M}_1)} \quad (16.1)$$

Here y refers to the data and Θ is a vector of parameters; for example, this vector could include the intercept, slope, and variance component in a linear regression model.

$P(y \mid \mathcal{M}_1)$ is the marginal likelihood, and is a single number that tells you the likelihood of the observed data y given the model \mathcal{M}_1 (and only in the discrete case, it tells you the probability of the observed data y given the model; see also section 1.7). Because in general it's not a probability, it

should be interpreted relative to another marginal likelihood (evaluated at the same y).

In frequentist statistics, it's also common to quantify evidence for the model by determining the maximum likelihood, that is, the likelihood of the data given the best-fitting model parameter. Thus, the data is used twice: once for fitting the parameter, and then for evaluating the likelihood. Importantly, this inference completely hinges upon this best-fitting parameter to be a meaningful value that represents well what we know about the parameter, and doesn't take the uncertainty of the estimates into account. Bayesian inference quantifies the uncertainty that is associated with a parameter, that is, one accepts that the knowledge about the parameter value is uncertain. Computing the marginal likelihood entails computing the likelihood given all plausible values for the model parameter.

One difficulty in the above equation showing Bayes' rule is that the marginal likelihood $P(y | \mathcal{M}_1)$ in the denominator cannot be easily computed in Bayes' rule:

$$p(\Theta | y, \mathcal{M}_1) = \frac{p(y | \Theta, \mathcal{M}_1)p(\Theta | \mathcal{M}_1)}{p(y | \mathcal{M}_1)} \quad (16.2)$$

The marginal likelihood is not a function: It does not depend on the model parameters Θ ; the parameters are “marginalized” or integrated out:

$$P(y | \mathcal{M}_1) = \int p(y | \Theta, \mathcal{M}_1)p(\Theta | \mathcal{M}_1)d\Theta \quad (16.3)$$

The likelihood is evaluated for every possible parameter value (that is what the integral does), weighted by the prior plausibility and summed together. For this reason, *the prior is as important as the likelihood*. Equation (16.3) also looks almost identical to the prior predictive distribution from section 3.2 (that is, the predictions that the model makes before seeing any data). The prior predictive distribution is repeated below for convenience:

$$\begin{aligned} p(\mathbf{y}_{pred}) &= p(y_{pred_1}, \dots, y_{pred_n}) \\ &= \int_{\Theta} p(y_{pred_1} | \Theta) \cdot p(y_{pred_2} | \Theta) \cdots p(y_{pred_N} | \Theta)p(\Theta) d\Theta \end{aligned} \quad (16.4)$$

However, while the prior predictive distribution describes possible observations, the marginal likelihood is evaluated on the actually observed data.

Let's compute the Bayes factor for a very simple example case. We assume a study where we assess the number of "successes" observed in a fixed number of trials. For example, we have 80 "successes" out of 100 trials. A simple model of this data can be built by assuming, as we did in section 1.4, that the data are distributed according to a binomial distribution. In a binomial distribution, n independent experiments are performed, where the result of each experiment is either a "success" or "no success" with probability θ . The binomial distribution is the probability distribution of the number of successes k (number of "success" responses) in this situation for a given sample of experiments X .

Suppose now that we have prior information about the probability parameter θ . As we explained in section 2.2, a typical prior distribution for θ is a Beta distribution. The Beta distribution defines a probability distribution on the interval $[0, 1]$, which is the interval on which the probability θ is defined. It has two parameters a and b , which determine the shape of the distribution. The prior parameters a and b can be interpreted as the *a priori* number of "successes" versus "failures". These could be based on previous evidence, or on the researcher's beliefs, drawing on their domain knowledge (O'Hagan et al. 2006).

Here, to illustrate the calculation of the Bayes factor, we assume that the parameters of the Beta distribution are $a = 4$ and $b = 2$. As mentioned above, these parameters can be interpreted as representing "success" ($n = 4$ prior observations), and "no success" ($n = 2$ prior observations). The resulting prior distribution is visualized in Figure 16.1. A $Beta(a = 4, b = 2)$ prior on θ amounts to a mildly informative prior with some, but no clear prior evidence for more than 50% of success.

To compute the marginal likelihood, equation (16.3) shows that we need to multiply the likelihood with the prior. The marginal likelihood is then the area under the curve, that is, the likelihood averaged across all possible values for the model parameter (the probability of success).

Based on this data, likelihood, and prior we can calculate the marginal

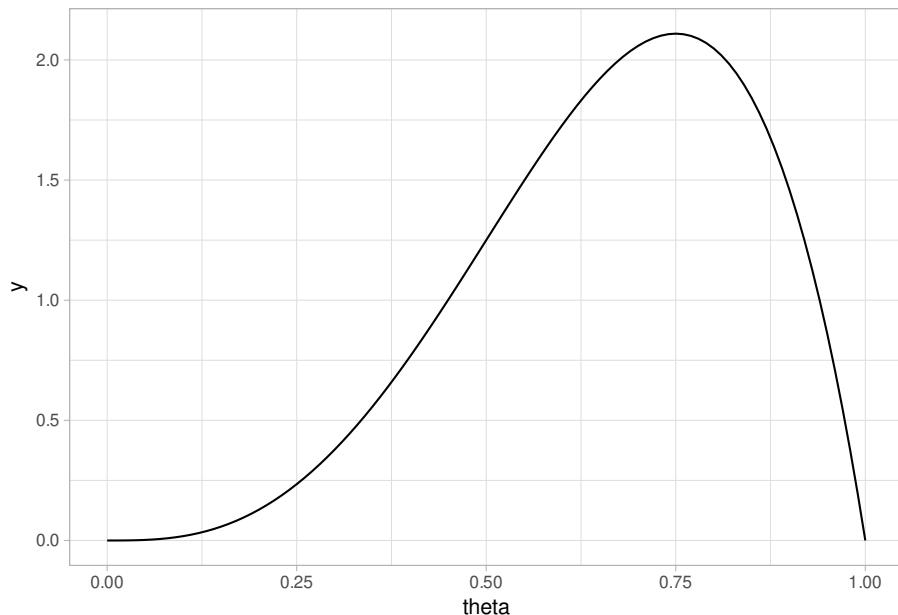


FIGURE 16.1: Beta distribution with parameters $a = 4$ and $b = 2$.

likelihood, that is, this area under the curve, in the following way using R:¹

```
# First we multiply the likelihood with the prior
plik1 <- function(theta) {
  dbinom(x = 80, size = 100, prob = theta) *
    dbeta(x = theta, shape1 = 4, shape2 = 2)
}
# Then we integrate (compute the area under the curve):
(MargLik1 <- integrate(f = plik1, lower = 0, upper = 1)$value)

## [1] 0.02
```

Importantly, one would prefer a model that gives a higher marginal likelihood, i.e., a higher likelihood of observing the data after integrating out

¹Given that the posterior is analytically available for Beta-distributed priors for the Binomial distribution, we could alternatively compute the posterior first, and then integrate out the probability p . For increased generalization to the following examples, here we show the version where the likelihood and the prior are multiplied.

the influence of the model parameter(s) (here: θ). A model will yield a high marginal likelihood if it makes a high proportion of good predictions (i.e., model 2 in Figure 16.2; Figure adapted from Bishop (2006)). Model predictions are normalized, that is, the total probability that models assign to different expected data patterns is the same for all models. Models that are too flexible (model 3 in Figure 16.2) will divide their prior predictive probability density across all of their predictions. Such models can predict many different outcomes. Thus, they likely can also predict the actually observed outcome. However, due to the normalization, they cannot predict it with high probability, because they also predict all kinds of other outcomes. This is true for both models with priors that are too wide or for models with too many parameters. Bayesian model comparison automatically penalizes such complex models, which is called the “Occam factor” (MacKay 2003).

By contrast, good models (Figure 16.2, model 2) will make very specific predictions, where the specific predictions are consistent with the observed data. Here, all the predictive probability density is located at the “location” where the observed data fall, and little probability density is located at other places, providing good support for the model. Of course, specific predictions can also be wrong, when expectations differ from what the observed data actually look like (Figure 16.2, model 1).

Having a natural Occam factor is good for posterior inference, i.e., for assessing how much (continuous) evidence there is for one model or another. However, it doesn’t necessarily imply good decision making or hypothesis testing, i.e., to make discrete decisions about which model explains the data best, or on which model to base further actions.

Here, we provide two examples of more flexible models. First, the following model assumes the same likelihood and the same distribution function for the prior. However, we assume a less informative prior, with prior parameters $a = 1$ and $b = 1$ (i.e., only one prior “success” and one prior “failure”), which provides more prior spread than the first model. Again, we can formulate our model as multiplying the likelihood with the prior, and integrate out the influence of the parameter θ :

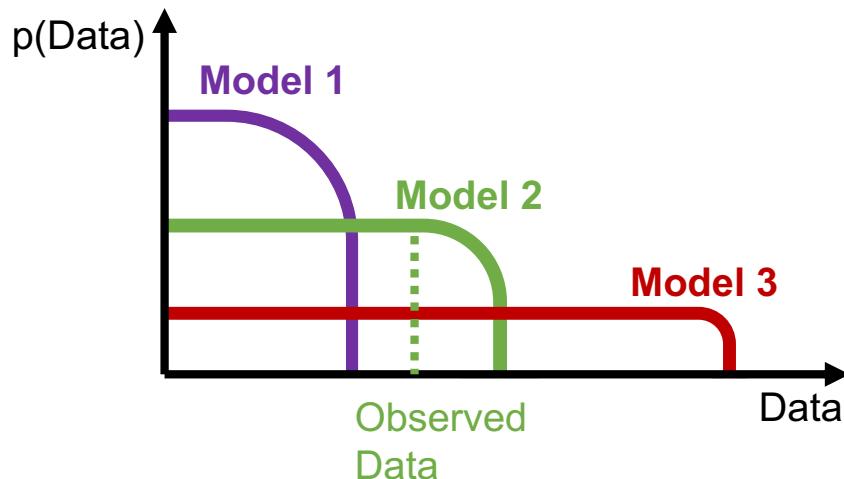


FIGURE 16.2: Shown are the schematic probabilities, $p(\text{Data})$, that each of three models assigns to different possible data. The total probability each model assigns to the data is equal to one, i.e., the areas under the curves of all three models are the same. Model 1 (violet) assigns all probability to a narrow range of data, and can predict this data with high probability (low complexity model). Model 3 (red) assigns its probability to a large range of different possible outcomes, but predicts each individual data with low probability (high complexity model). Model 2 (green) takes an intermediate position (intermediate complexity). The vertical dashed line (green) illustrates where the actual empirically observed data fall. The data most support model 2, since this model predicts the data with highest probability. The figure is closely designed after Figure 3.13 in Bishop (2006).

```

plik2 <- function(theta) {
  dbinom(x = 80, size = 100, prob = theta) *
  dbeta(x = theta, shape1 = 1, shape2 = 1)
}
(MargLik2 <- integrate(f = plik2, lower = 0, upper = 1)$value)

## [1] 0.0099

```

We can see that this second model is more flexible: due to the more spread-out prior, it is compatible with a larger range of possible observed data patterns. However, when we integrate out the θ parameter to obtain the marginal likelihood, we can see that this flexibility also comes with a cost: the model has a smaller marginal likelihood (0.00990) than the first model (0.01998). Thus, on average (averaged across all possible values of θ) the second model performs worse in explaining the specific data that we observed compared to the first model, and has less support from the data.

A model might be more “complex” because it has a more spread-out prior, or alternatively because it has a more complex likelihood function, which uses a larger number of parameters to explain the same data. Here we implement a third model, which assumes a more complex likelihood by using a Beta-Binomial distribution. The Beta-Binomial distribution is equivalent to the Binomial distribution, with one important difference: In the Binomial distribution the probability of success θ is fixed across trials. In the Beta-Binomial distribution, the probability of success is fixed for each trial, but is drawn from a Beta distribution across trials. Thus, θ can differ between trials. In the Beta-Binomial distribution, we thus assume that the likelihood function is a combination of a Binomial distribution and a Beta distribution of the probability θ , which yields:

$$p(X = k | a, b) = \frac{B(k + a, n - k + b)}{B(a, b)} \quad (16.5)$$

What is important here is that this more complex distribution has two parameters (a and b ; rather than one, θ) to explain the same data. We assume log-normally distributed priors for the a and b parameters, with location zero and scale 100, which correspond to the mean (0) and standard deviation (100) of the distribution on the log scale. The likelihood of this combined Beta-Binomial distribution is given by the R-function `dbbinom()` in the package `extraDistr`. We can now write down the likelihood times the priors (given as log-normal densities, `dlnorm()`), and integrate out the influence of the two free model parameters a and b using numerical integration (applying `integrate` twice):

```
plik3 <- function(a, b) {
  extraDistr::dbbinom(x = 80, size = 100, alpha = a, beta = b) *
```

```

dlnorm(x = a, meanlog = 0, sdlog = 100) *
dlnorm(x = b, meanlog = 0, sdlog = 100)
}

# Compute marginal likelihood by applying integrate twice
f <- function(b) integrate(function(a) plik3(a, b), lower = 0, upper = Inf)$value
# integrate requires a vectorized function:
(MargLik3 <- integrate(Vectorize(f), lower = 0, upper = Inf)$value)

## [1] 0.00000707

```

The results show that this third model has an even smaller marginal likelihood compared to the first two (0.000007). With its two parameters a and b , this third model has a lot of flexibility to explain a lot of different patterns of observed empirical results. However, again, this increased flexibility comes at a cost, and the simple pattern of observed data does not seem to require such complex model assumptions. The small value for the marginal likelihood indicates that this complex model has less support from the data.

That is, for this present simple example case, we would prefer model 1 over the other two, since it has the largest marginal likelihood (0.01998), and we would prefer model 2 over model 3, since the marginal likelihood of model 2 (0.00990) is larger than that of model 3 (0.000007). The decision about which model is preferred is based on comparing the marginal likelihoods.

16.1.2 Bayes factor

The Bayes factor is a measure of relative evidence, the comparison of the predictive performance of one model against another one. This comparison is a ratio of marginal likelihoods:

$$BF_{12} = \frac{P(y | \mathcal{M}_1)}{P(y | \mathcal{M}_2)} \quad (16.6)$$

BF_{12} indicates the extent to which the data are more probable under \mathcal{M}_1 over \mathcal{M}_2 , or in other words, which of the two models is more likely to have generated the data, or the relative evidence that we have for \mathcal{M}_1 over \mathcal{M}_2 .

Values larger than one indicate evidence in favor of \mathcal{M}_1 , smaller than one indicate evidence in favor of \mathcal{M}_2 , and values close to one indicate that the evidence is inconclusive. This model comparison does not depend on a specific parameter value. Instead, all possible prior parameter values are taken into account simultaneously. This is in contrast with the likelihood ratio test, as it is explained in Box 16.1

Box 16.1. Likelihood ratio vs Bayes Factor.

The likelihood ratio test is a very similar, but frequentist, approach to model comparison and hypothesis testing, which also compares the probability for the data given two different models. We show this here to highlight the similarities and differences between frequentist and Bayesian hypothesis testing. In contrast to the Bayes factor, the likelihood ratio test depends on the “best” (i.e., the maximum likelihood) estimate for the model parameter(s), that is, the model parameter θ occurs on the right side of the semi-colon in the equation for each likelihood. (An aside: we do not use a conditional statement, i.e., the vertical bar, when talking about likelihood in the frequentist context; instead, we use a semi-colon. This is because the statement $f(y | \theta)$ is a conditional statement, implying that θ has a probability density function associated with it; in the frequentist framework, parameters cannot have a pdf associated with them, they are assumed to have fixed, point values.)

$$LikRat = \frac{P(y; \hat{\theta}_1, \mathcal{M}_1)}{P(y; \hat{\theta}_2, \mathcal{M}_2)} \quad (16.7)$$

That means that in the likelihood ratio test, each model is tested on its ability to explain the data using this “best” estimate for the model parameter (here $\hat{\theta}$). That is, the likelihood ratio test reduces the full range of possible parameter values to a point value, leading to overfitting the model to the maximum likelihood estimate (MLE). If the MLE badly misestimates the true value of the parameter, due to Type M error (Gelman and Carlin 2014), tough luck. By contrast, the Bayes factor involves range hypotheses, which are implemented via inte-

grals over the model parameter; that is, it uses marginal likelihoods that are averaged across all possible posterior values of the model parameter(s). Thus, if, due to Type M error, the best point estimate (the MLE) for the model parameter(s) is not very representative of the possible values for the model parameter(s), then Bayes factors will be superior to the likelihood ratio test. An additional difference, of course, is that Bayes factors rely on priors for estimating each model's parameter(s), whereas the frequentist likelihood ratio test does not consider priors in the estimation of the best-fitting model parameter(s). As we show later, this has far-reaching consequences for Bayes factor-based model comparison; for a more extensive exposition, see Schad et al. (2021).

For the Bayes factor, a scale (see Table 16.1) has been proposed to interpret Bayes factors according to the strength of evidence in favor of one model (corresponding to some hypothesis) over another (Jeffreys 1939); but this scale should not be regarded as a hard and fast rule with clear boundaries.

TABLE 16.1: Bayes factor scale as proposed by Jeffreys (1939).

BF_{12}	Interpretation
> 100	Extreme evidence for \mathcal{M}_1 .
$30 - 100$	Very strong evidence for \mathcal{M}_1 .
$10 - 30$	Strong evidence for \mathcal{M}_1 .
$3 - 10$	Moderate evidence for \mathcal{M}_1 .
$1 - 3$	Anecdotal evidence for \mathcal{M}_1 .
1	No evidence.
$\frac{1}{1} - \frac{1}{3}$	Anecdotal evidence for \mathcal{M}_2 .
$\frac{1}{3} - \frac{1}{10}$	Moderate evidence for \mathcal{M}_2 .
$\frac{1}{10} - \frac{1}{30}$	Strong evidence for \mathcal{M}_2 .
$\frac{1}{30} - \frac{1}{100}$	Very strong evidence for \mathcal{M}_2 .
$< \frac{1}{100}$	Extreme evidence for \mathcal{M}_2 .

So if we go back to our previous example, we can calculate BF_{12} , BF_{13} , and BF_{23} . (BF_{21} is simply $\frac{1}{BF_{12}}$).

$$BF_{12} = \frac{\text{marginal likelihood model 1}}{\text{marginal likelihood model 2}} = \frac{MargLik1}{MargLik2} = 2 \quad (16.8)$$

$$BF_{13} = \frac{MargLik1}{MargLik3} = 2825.4 \quad (16.9)$$

$$BF_{32} = \frac{MargLik3}{MargLik2} = 0.001 = \frac{1}{BF_{23}} = \frac{1}{1399.9} \quad (16.10)$$

However, if we want to know, given the data y , what the probability for model \mathcal{M}_1 is, or how much more probable model \mathcal{M}_1 is than model \mathcal{M}_2 , then we need the prior odds, that is, we need to specify how probable \mathcal{M}_1 is compared to \mathcal{M}_2 *a priori*.

$$\frac{p(\mathcal{M}_1 | y)}{p(\mathcal{M}_2 | y)} = \frac{p(\mathcal{M}_1)}{p(\mathcal{M}_2)} \times \frac{P(y | \mathcal{M}_1)}{P(y | \mathcal{M}_2)} \quad (16.11)$$

$$\text{Posterior odds}_{12} = \text{Prior odds}_{12} \times BF_{12} \quad (16.12)$$

The Bayes factor tells us, given the data and the priors, by how much we need to update our relative belief between the two models. However, **the Bayes factor alone cannot tell us which one of the models is the most probable**. Given our priors for the models and the Bayes factor, we can calculate the odds between the models.

Here we compute posterior model probabilities for the case where we compare two models against each other. However, posterior model probabilities can also be computed for the more general case, where more than two models are considered:

$$p(\mathcal{M}_1 | y) = \frac{p(y | \mathcal{M}_1)p(\mathcal{M}_1)}{\sum_n p(y | \mathcal{M}_n)p(\mathcal{M}_n)} \quad (16.13)$$

For simplicity, we mostly constrain ourselves to two models. (However, the sensitivity analyses we carry out below compare more than two models.)

Importantly, Bayes factors (and posterior model probabilities) tell us how

much evidence the data (and priors) provide in favor of one model or another. That is, they allow us to perform inferences on the model space, i.e., to learn how much each hypothesis is consistent with the data. A completely different issue, however, is the question of how to perform (discrete) decisions based on continuous evidence. The question here is: which hypothesis should one choose to maximize utility? While Bayes factors have a clear rationale and justification in terms of the (continuous) evidence they provide, there is not a clear and direct mapping from inferences to how to perform decisions based on them. To derive decisions based on posterior model probabilities, utility functions are needed. Indeed, the utility of different possible actions (i.e., to accept and act based on one hypothesis or another) can differ quite dramatically in different situations. For example, for a researcher trying to implement a life-saving therapy, erroneously rejecting this new therapy could have high negative utility, whereas erroneously adopting the new therapy may have little negative consequences. By contrast, erroneously claiming a new discovery in fundamental research may have bad consequences (low utility), whereas erroneously missing a new discovery claim may be less problematic if further evidence can be accumulated. Thus, Bayesian evidence (in the form of Bayes factors or posterior model probabilities) must be combined with utility functions in order to perform decisions based on them. For example, this could imply specifying the utility of a true discovery (U_{TD}) and the utility of a false discovery (U_{FD}). Calibration (i.e., simulations) can then be used to derive decisions that maximize overall utility (see Schad et al. 2021).

The question now is how do we extend this method to models that we care about, i.e., that represent more realistic data analysis situations. In cognitive science, we typically fit fairly complex hierarchical models with many variance components. The major problem is that we won't be able to calculate the marginal likelihood for hierarchical models (or any other complex model) either analytically or just using the R functions shown above. There are two very useful methods for calculating the Bayes factor for complex models: the Savage–Dickey density ratio method (Dickey, Lientz, and others 1970; Wagenmakers et al. 2010) and bridge sampling (Bennett 1976; Meng and Wong 1996). The Savage–Dickey density ratio method is a straightforward way to compute the Bayes factor, but it is limited to nested models. The current implementation of the Savage–Dickey

method in brms can be unstable, especially in cases where the posterior is far away from zero. Bridge sampling is a much more powerful method, but it requires many more samples than what is normally required for parameter estimation. We will use bridge sampling from the `bridgesampling` package (Quentin F Gronau et al. 2017b; Quentin F Gronau, Singmann, and Wagenmakers 2017) with the function `bayes_factor()` to calculate the Bayes factor in the first examples.

16.2 Examining the N400 effect with Bayes factor

In section 5.1 we estimated the effect of cloze probability on the N400 average signal. This yielded a posterior credible interval for the effect of cloze probability. It is certainly possible to check whether e.g., the 95% posterior credible interval overlaps with zero or not. However, such estimation cannot really answer the following question: How much evidence do we have in support for an effect? A 95% credible interval that doesn't overlap with zero, or a high probability mass away from zero may hint that the predictor may be needed to explain the data, but it is not really answering how much evidence we have in favor of an effect (for discussion, see Royall 1997; Wagenmakers et al. 2019; Rouder, Haaf, and Vandekerckhove 2018).

This is a very important point. Indeed, this is often overlooked in the literature, and many papers misuse 95% posterior credible intervals to argue that there is evidence for or against an effect. In the past, we have also misused posterior credible intervals in this way (and even recommended this incorrect interpretation in, for example, Nicenboim and Vasishth 2016).

The reason why the 95% posterior credible interval does not answer the question about evidence for the H_1 or H_0 is that we do not explicitly consider and quantify the possibility that the parameter estimate is zero: we do not quantify the likelihood of the data under the assumption that the effect is absent. The Bayes factor answers this question about the evidence in favor of an effect by explicitly conducting a model comparison. We will compare a model that assumes the presence of an effect, with a null model that assumes no effect.

As we saw before, the Bayes factor is highly sensitive to the priors. In the

example presented above, both models are identical except for the effect of interest, β , and so the prior on this parameter will play a major role in the calculation of the Bayes factor.

Next, we will run a hierarchical model which includes random intercepts and slopes by items and by subjects. We will use relatively informative priors on all the parameters – this speeds up computation and implies realistic expectations about the parameters. However, the prior on β will be **crucial** for the calculation of the Bayes factor.

Building good priors is a challenging task. Indeed, it is one of the crucial steps involved in a principled Bayesian workflow (Schad, Betancourt, and Vasishth 2020), which we will discuss in a later chapter.

One possible way we can build a good prior for the parameter β estimating the influence of cloze probability here is the following. The reasoning below is based on domain knowledge; but there is room for differences of opinion here. In a realistic data analysis situation, we would carry out a sensitivity analysis using a range of priors to determine the extent of influence of the priors.²

1. One may want to be agnostic regarding the direction of the effect; that means that we will center the prior of β on zero by specifying that the mean of the prior distribution is zero. However, we are still not sure about the variance of the prior on β .
2. One would need to know a bit about the variation on the dependent variable that we are analyzing. After re-analyzing the data from a couple of EEG experiments available from osf.io, we can say that for N400 averages, the standard deviation of the signal is between 8-15 microvolts (Nicenboim, Vasishth, and Rösler 2020b).
3. Based on published estimates of effects in psycholinguistics, we can conclude that they are generally rather small, often repre-

²A formal methodology for eliciting priors (including from oneself) is discussed in O'Hagan et al. (2006). There is also a very useful R package, SHELF Oakley and O'Hagan (2010), which is very helpful in visualizing priors in preparation for a sensitivity analysis. Schad, Betancourt, and Vasishth (2020) provide some suggested guidelines for a principled investigation of priors in Bayesian analysis.

senting between 5%-30% of the standard deviation of the dependent variable.

4. The effect of noun predictability on the N400 is one of the most reliable and strongest effects in neurolinguistics (together with the P600 that might even be stronger), and the slope β represents the average change in voltage when moving from a cloze probability of zero to one – the strongest prediction effect.

An additional and highly recommended way to obtain good priors (Schad, Betancourt, and Vasishth 2020, also see the chapter later in the present book on a principled Bayesian workflow) is to perform prior predictive checks. Here, the idea is to simulate data from the model and the priors, and then to analyze the simulated data using summary statistics. For example, it would be possible to compute the summary statistic of the difference in the N400 between high versus low cloze probability. The simulations would yield a distribution of differences. Arguably, this distribution of differences, that is, the data analyses of the simulated data, are much easier to judge for plausibility than the prior parameters specifying prior distributions. That is, we might find it easier to judge whether a difference in voltage between high and low cloze probability is plausible rather than judging the parameters of the model. For reasons of brevity, we skip performing these calculations here (but see the later chapter on developing a workflow for details).

Instead, we will start with the prior $\beta \sim Normal(0, 5)$ (since 5 microvolts is roughly 30% of 15, which is the upper bound of the expected standard deviation of the EEG signal).

```
priors1 <- c(
  prior(normal(2, 5), class = Intercept),
  prior(normal(0, 5), class = b),
  prior(normal(10, 5), class = sigma),
  prior(normal(0, 2), class = sd),
  prior(lkj(4), class = cor)
)
```

We load the dataset on N400 amplitudes, which has data on cloze probabilities (Nieuwland et al. 2018). We mean-center the cloze probability mea-

sure to make the intercept and the random intercepts easier to interpret (i.e., after centering, they represent the grand mean and the average variability around the grand mean across subjects or items).

```
data(df_eeg)
df_eeg <- df_eeg %>% mutate(c_cloze = cloze - mean(cloze))
```

We will need a large number of samples to be able to get stable estimates of the Bayes factor with bridge sampling, for this reason a large number of sampling iterations ($n = 20000$) is specified. Because otherwise we see warnings, we also set `adapt_delta = 0.9` to ensure that the posterior sampler is working correctly. For Bayes factors analyses, it's necessary to set the argument `save_pars = save_pars(all = TRUE)`. This setting is a precondition for later performing Bridge sampling for computing the Bayes factor.

```
fit_N400_h_linear <- brm(n400 ~ c_cloze +
  (c_cloze | subj) + (c_cloze | item),
prior = priors1,
warmup = 2000,
iter = 20000,
cores = 4,
control = list(adapt_delta = 0.9),
save_pars = save_pars(all = TRUE),
data = df_eeg
)
```

Next, take a look at the population-level results from the Bayesian modeling.

```
fixef(fit_N400_h_linear)

##           Estimate Est.Error Q2.5 Q97.5
## Intercept     3.65      0.45  2.76  4.53
## c_cloze      2.33      0.65  1.05  3.59
```

We can now take a look at the estimates and at the credible intervals. The

effect of Cloze probability (`c_cloze`) is 2.33 with a 95% credible interval ranging from 1.05 to 3.59. While this provides an initial hint that highly probable words may elicit a stronger N400 compared to low probable words, by just looking at the posterior there is no way to quantify evidence for the question whether this effect is different from zero. Model comparison is needed to answer this question.

To this end, we run the model again, now without the parameter of interest, i.e., the null model. This is a model where our prior for β is that it is exactly zero.

```
fit_N400_h_null <- brm(n400 ~ 1 +
  (c_cloze | subj) + (c_cloze | item),
prior = priors1[priors1$class != "b", ],
warmup = 2000,
iter = 20000,
cores = 4,
control = list(adapt_delta = 0.9),
save_pars = save_pars(all = TRUE),
data = df_eeg
)
```

Now everything is ready to compute the log marginal likelihood, that is, the probability of the data given the model, after integrating out the model parameters. In the toy examples shown above, we had used the R-function `integrate()` to perform this integration. This is not possible for the more realistic and more complex models that are considered here because the integrals that have to be solved are too high-dimensional and complex for these simple functions to do the job. Instead, a standard approach to sampling realistic complex models is to use bridge sampling (Quentin F Gronau et al. 2017b; Quentin F Gronau, Singmann, and Wagenmakers 2017). We perform this integration using the function `bridge_sampler()` for each of the two models:

```
margLogLik_linear <- bridge_sampler(fit_N400_h_linear, silent = TRUE)
margLogLik_null <- bridge_sampler(fit_N400_h_null, silent = TRUE)
```

This gives us the marginal log likelihoods for each of the models. From these, we can compute the Bayes factors. The function `bayes_factor()` is a convenient function to calculate the Bayes factor.

```
(BF_ln <- bayes_factor(margLogLik_linear, margLogLik_null))
```

```
## Estimated Bayes factor in favor of x1 over x2: 50.96782
```

Alternatively, the Bayes factor can be computed manually as well. First, we compute the difference in marginal log likelihoods, then we transform this difference in log likelihoods to the likelihood scale (`exp()`). A difference in exponential scale is the same as a ratio in non-exponentiated scale: $\exp(a-b) = \exp(a)/\exp(b)$, that is, it computes the Bayes factor. However, the values `exp(ml1)` and `exp(ml2)` are too small to be represented accurately by R. Therefore, for numerical reasons, it is important to take the difference first and to compute the exponential afterwards `exp(a-b)`, i.e., `exp(margLogLik_linear$logml - margLogLik_null$logml)`, which yields the same result as the `bayes_factor()` command.

```
exp(margLogLik_linear$logml - margLogLik_null$logml)
```

```
## [1] 51
```

The Bayes factor is quite large in this example, and provides strong evidence for the alternative model, which includes a coefficient representing the effect of cloze probability. That is, under the criteria shown in Table 16.1, the Bayes factor provides strong evidence for an effect of cloze probability.

In this example, there was good prior information about the free model parameter β . However, what happens if we are not sure about the prior for the model parameter? It might happen that we compare the null model with a very “bad” alternative model, because our prior for β is not appropriate.

For example, assuming that we do not know much about N400 effects, or that we do not want to make strong assumptions, we might be inclined to use very vague and uninformative priors. For example, these could look as follows (where all the priors except for b remain identical):

```
priors_vague <- c(
  prior(normal(2, 5), class = Intercept),
  prior(normal(0, 500), class = b),
  prior(normal(10, 5), class = sigma),
  prior(normal(0, 2), class = sd),
  prior(lkj(4), class = cor)
)
```

We can use these uninformative priors in the Bayesian model:

```
fit_N400_h_linear_vague <- brm(n400 ~ c_cloze +
  (c_cloze | subj) + (c_cloze | item),
prior = priors_vague,
warmup = 2000,
iter = 20000,
cores = 4,
control = list(adapt_delta = 0.9),
save_pars = save_pars(all = TRUE),
data = df_eeg
)
```

Interestingly, we can still estimate the effect of cloze probability fairly well:

```
posterior_summary(fit_N400_h_linear_vague, variable = "b_c_cloze")
```

```
##           Estimate Est.Error Q2.5 Q97.5
## b_c_cloze     2.37     0.652  1.07  3.64
```

Next, we again perform the bridge sampling for the alternative model.

```
margLogLik_linear_vague <- bridge_sampler(fit_N400_h_linear_vague, silent = TRUE)
```

We compute the Bayes factor for the alternative over the null model, BF_{10} :

```
(BF_lnVague <- bayes_factor(margLogLik_linear_vague, margLogLik_null))  
  
## Estimated Bayes factor in favor of x1 over x2: 0.56000
```

This is easier to read as the evidence for null model over the alternative:

```
1 / BF_lnVague[[1]]
```

```
## [1] 1.79
```

The results show that there is no evidence in favor of the effect of Cloze probability. The reason for that is that priors are never uninformative when it comes to Bayes factors. The wide prior specifies that both very small and very large effect sizes are possible (with some considerable probability), but there is relatively little evidence in the data for such large effect sizes.

Indeed, very recently, Uri Simonsohn has criticized Bayes factors because they might provide evidence in favor of the null and against a very specific alternative model, when the researchers only knew the direction of the effect (see <https://datacolada.org/78a>). This can happen when very uninformative vague priors are used.

One way to overcome this problem is to actually try to learn about the effect size that we are investigating. This can be done by first running an exploratory experiment and analysis without computing any Bayes factor, and then use the posterior distribution derived from this first experiment to calibrate the priors for the next confirmatory experiment where we do use the Bayes factor (see Verhagen and Wagenmakers 2014 for a Bayes Factor test calibrated to investigate replication success).

Another possibility is to examine a lot of different alternative models, where each model uses different prior assumptions. This way, it's possible to investigate the extent to which the Bayes factor results depend on, or are sensitive to, the prior assumptions. This is an instance of a sensitivity analysis. Recall that the model is the likelihood *and* the priors. We can therefore compare models that only differ in the prior (for an example involving EEG and predictability effects, see Nicenboim, Vasishth, and Rösler 2020b).

16.2.1 Sensitivity analysis

Here, we perform a sensitivity analysis by examining Bayes factors for several models. Each model has the same likelihood but a different prior for β . For all of the priors we assume a normal distribution with a mean of zero. Assuming a mean of zero means that we do not make any assumption a priori that the effect differs from zero. If the effect should differ from zero, we want the data to tell us that. What differs between the different priors is their standard deviation. That is, what differs is the amount of uncertainty about the effect size that we allow for in the prior. A large standard deviation allows for very large effect sizes, whereas a small standard deviation implies the assumption that we expect the effect not to be very large. Although a model with a wide prior (i.e., large standard deviation) also allocates prior probability to small effect sizes, it allocates much less probability to small effect sizes compared to a model with a narrow prior. Thus, if the effect size in the observed data is actually small, then a model with a narrow prior (small standard deviation) will have a better chance of detecting the effect.

Next, we try out a range of standard deviations, ranging from 1 to a much wider prior that has a standard deviation of 100. In practice, for the experiment method we are discussing here, it would not be a good idea to define very large standard deviations such as 100 microvolts, since they imply unrealistically large effect sizes. However, we include such a large value here just for illustration. Such a sensitivity analysis takes a very long time: here, we are running 11 models, where each model involves a lot of iterations to obtain stable Bayes factor estimates.

```
prior_sd <- c(1, 1.5, 2, 2.5, 5, 8, 10, 20, 40, 50, 100)
BF <- c()
for (i in 1:length(prior_sd)) {
  psd <- prior_sd[i]
  # for each prior we fit the model
  fit <- brm(n400 ~ c_cloze + (c_cloze | subj) + (c_cloze | item),
  prior =
  c(
    prior(normal(2, 5), class = Intercept),
    set_prior(paste0("normal(0,", psd, ")"), class = "b"),
    prior(normal(0, 100), class = "sigma")
  )
  BF[[i]] <- bayes_factor(fit)
}
```

```

prior(normal(10, 5), class = sigma),
prior(normal(0, 2), class = sd),
prior(lkj(4), class = cor)
),
warmup = 2000,
iter = 20000,
cores = 4,
control = list(adapt_delta = 0.9),
save_pars = save_pars(all = TRUE),
data = df_eeg
)
# for each model we run a bridge sampler
lml_linear_beta <- bridge_sampler(fit, silent = TRUE)
# we store the Bayes factor compared to the null model
BF <- c(BF, bayes_factor(lml_linear_beta, lml_null)$bf)
}
BFs <- tibble(beta_sd = prior_sd, BF)

```

For each model, we run bridge sampling and we compute the Bayes factor of the model against our baseline or null model, which does not contain a fixed effect of Cloze probability (BF_{10}). Next, we need a way to visualize all the Bayes factors. We plot them in Figure 16.3 as a function of the prior width.

This figure clearly shows that the Bayes factor provides evidence for the alternative model; that is, it provides evidence that the fixed effect Cloze probability is needed to explain the data. This can be seen as the Bayes factor is quite large for a range of different values for the prior standard deviation. The Bayes factor is largest for a prior standard deviation of 2.5, suggesting a rather small size of the effect of Cloze probability. If we assume gigantic effect sizes a priori (e.g., standard deviations of 50 or 100), then the evidence for the alternative model is weaker. Conceptually, the data do not fully support such big effect sizes, but start to favor the null model relatively more, when such big effect sizes are tested against the null. Overall, we can conclude that the data provide evidence for a not too large but robust influence of Cloze probability on the N400 amplitude.

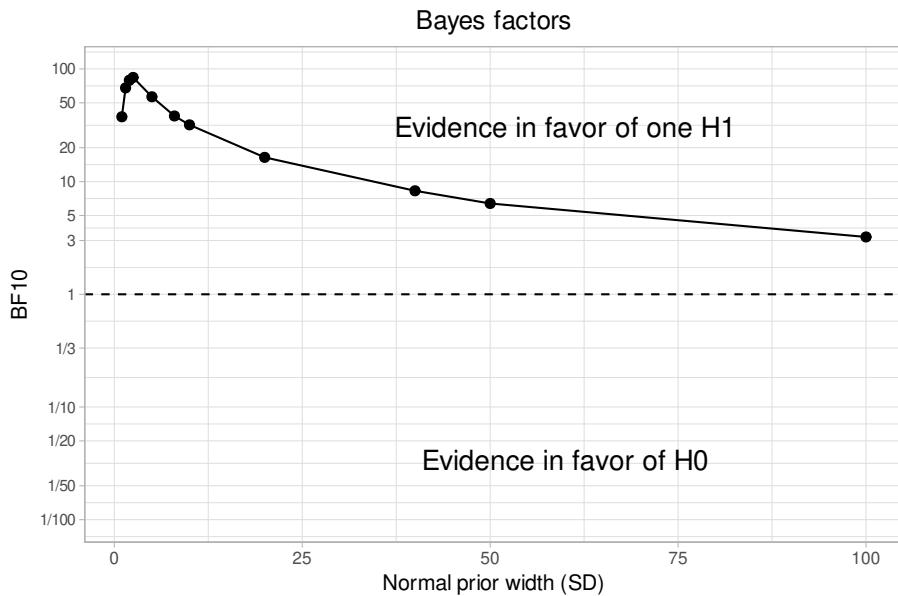


FIGURE 16.3: Prior sensitivity analysis for the Bayes factor

16.2.2 Non-nested models

One important advantage of Bayes factors is that they can be used to compare models that are not nested. In nested models, the simpler model is a special case of the more complex and general model. For example, our previous model of Cloze probability was a general model, allowing different influences of Cloze probability on the N400. We compared this to a simpler more specific null model, where the influence of Cloze probability was not included, which means that the regression coefficient (fixed effect) for Cloze probability was assumed to be set to zero. Such nested models can be compared using frequentist methods such as the likelihood ratio test (ANOVA).

By contrast, the Bayes factor also makes it possible to compare non-nested models. An example of a non-nested model would be a case where we log-transform the Cloze probability variable before using it as a predictor. A model with log Cloze probability as a predictor is not a special case of a model with linear cloze probability as predictor. These are just different, alternative models. With Bayes factors, we can compare these

non-nested models with each other to determine which receives more evidence from the data.

To do so, we first log-transform the Cloze probability variable. Some Cloze probabilities in the data set are equal to zero. This creates a problem when taking logs, since a log of zero is minus infinity; a value of the predictor variable that we cannot deal with. We are going to overcome this problem by “smoothing” the Cloze probability in this example. We use additive smoothing (also called Laplace or Lidstone smoothing; Lidstone 1920; Chen and Goodman 1999) with pseudocounts set to one, this means that the smoothed probability is calculated as the number of responses with a given gender plus one divided by the total number of responses plus two.

Box 16.2. Bayesian interpretation of additive smoothing

TO-DO

```
df_eeg <- df_eeg %>%
  mutate(
    scloze = (cloze_ans + 1) / (N + 2),
    c_logsclcloze = log(scloze) - mean(log(scloze))
  )
```

Next, we center the predictor variable, and we scale it to the same standard deviation as the linear Cloze probabilities. To implement this scaling, we first divide the centered smoothed log cloze probability variable by its standard deviation (i.e., effectively creating z-scaled values). As a next step, we multiply the z-scaled values by the standard deviation of the non-transformed cloze probability variable. This way, both predictors (log cloze and cloze) have the same standard deviation. We therefore expect them to have a similar impact on the N400. As a result of this transformation, we can therefore use the same priors for both variables (given that we currently have no specific information about the effect of log Cloze probability versus linear Cloze probability):

```
df_eeg <- df_eeg %>%
  mutate(c_logs cloze = scale(c_logs cloze) * sd(c_cloze))
```

Then, we can run a linear mixed-effects model with log Cloze probability instead of linear Cloze probability, and we again carry out bridge sampling.

```
fit_N400_h_log <- brm(n400 ~ c_logs cloze +
  (c_logs cloze | subj) + (c_logs cloze | item),
  prior = priors1,
  warmup = 2000,
  iter = 20000,
  cores = 4,
  control = list(adapt_delta = 0.9),
  save_pars = save_pars(all = TRUE),
  data = df_eeg
)
```

```
margLogLik_log <- bridge_sampler(fit_N400_h_log, silent = TRUE)
```

Now, we can compare the linear and the log model to each other using Bayes factors.

```
(BF_log_lin <- bayes_factor(margLogLik_log, margLogLik_linear))
```

```
## Estimated Bayes factor in favor of x1 over x2: 6.04762
```

The results show a Bayes factor of 6 of the log model over the linear model. This shows some evidence that log cloze probability is a better predictor of N400 amplitudes than linear cloze probability. Importantly, this analysis demonstrates that model comparisons using Bayes factor are not limited to nested models, but can also be used for non-nested models.

16.3 The influence of the priors on Bayes factors: beyond the effect of interest

We saw above that the width (or standard deviation) of the prior distribution for the effect of interest had a strong impact on the results from Bayes factor analyses (see sensitivity analysis). Thus, one question is whether only the prior for the effect of interest is important, or whether priors for other model parameters can also impact resulting Bayes factors in an analysis. It turns out that priors for other model parameters can also be important and impact Bayes factors, especially when there are non-linear components in the model, such as in generalized linear mixed effects models. We investigate this issue by using a simulated data set on a variable that has a Bernoulli distribution; in each trial, subjects can perform either successfully ($pDV = 1$) on a task, or not ($pDV = 0$). The simulated data is from a factorial experimental design, with one between-subject factor F with 2 levels (F1 and F2), and Table 16.2 shows success probabilities for each of the experimental conditions.

```
data("df_BF")
str(df_BF)

## # tibble [100 x 3] (S3:tbl_df/tbl/data.frame)
## $ F : Factor w/ 2 levels "F1","F2": 1 1 1 1 1 1 1 1 1 1 ...
## $ pDV: int [1:100] 1 1 1 1 1 1 1 1 1 1 ...
## $ id : int [1:100] 1 2 3 4 5 6 7 8 9 10 ...
```

Our question now is whether there is evidence for a difference in success

TABLE 16.2: Summary statistics per condition for the simulated data.

Factor A	N	data	Means
F1	50		0.98
F2	50		0.70

probabilities between groups F₁ and F₂. As contrasts for the factor F, we use scaled sum coding (-0.5, +0.5).

```
contrasts(df_BF$F) <- c(-0.5, +0.5)
```

Next, we proceed to specify our priors. Let's assume that for the difference between groups (F₁ versus F₂), we define a normally distributed prior with a mean of 0 and a standard deviation of 0.5. Thus, we do not specify a direction of the difference a priori, and assume not too large effect sizes. We can now run two logistic brms models, one with the group factor F included, and one without the factor F, and compute Bayes factors using bridge sampling to obtain the evidence that the data provide for the alternative hypothesis that a group difference exists between levels F₁ and F₂.

So far, we have only specified the prior for the effect size. The question we are asking now is whether priors on other model parameters can impact the Bayes factor computations for testing the group effect. Specifically, can the prior for the intercept influence the Bayes factor for the group difference? The results show that yes, such an influence can take place in some situations. Let's have a look at this in more detail. Let's assume that we compare two different priors for the intercept. We specify each as a normal distribution with a standard deviation of 0.1, thus, specifying relatively high certainty a priori where the intercept of the data will fall. The only difference that we now specify, is that one time, the prior mean (on the latent logistic scale) is set to 0, corresponding to a prior mean probability of 0.5. In the other condition, we specify a prior mean of 2, corresponding to a prior mean probability of 88%. When we look at the data (see Table 16.2) we see that the prior mean of 0 (i.e., prior probability for the intercept of 0.5) is not very compatible with the data, whereas the prior mean of 2 (i.e., a prior probability for the intercept of 0.88) is quite closely aligned with the actual data.

We now compute Bayes factors for the group difference (F₁ versus F₂) by using these different priors for the intercept. Thus, we first fit a null (H₀) and alternative (H₁) model under the assumption of a false prior belief (mean = 0), and perform bridge sampling for these models:

```

# set prior
priors_logit1 <- c(
  prior(normal(0, 0.1), class = Intercept),
  prior(normal(0, 0.5), class = b)
)
# Bayesian GLM: H0
fit_pDV_H0 <- brm(pDV ~ 1,
  data = df_BF,
  family = bernoulli(link = "logit"),
  prior = priors_logit1[-2, ],
  save_pars = save_pars(all = TRUE)
)
# Bayesian GLM: H1
fit_pDV_H1 <- brm(pDV ~ 1 + F,
  data = df_BF,
  family = bernoulli(link = "logit"),
  prior = priors_logit1,
  save_pars = save_pars(all = TRUE)
)
# Bridge sampling
mLL_binom_H0 <- bridge_sampler(fit_pDV_H0, silent = TRUE)
mLL_binom_H1 <- bridge_sampler(fit_pDV_H1, silent = TRUE)

```

Next, we again prepare computation of Bayes factors, by again running the null (H_0) and the alternative (H_1) model, now assuming a more realistic prior for the intercept (prior mean = 2).

```

``{r bfho12, echo=TRUE, message=FALSE, results="hide", eval =
!file.exists("dataR/mLL_binom_H0_2.RDS")
!file.exists("dataR/mLL_binom_H1_2.RDS")}

priors_logit2 <- c( prior(normal(2, 0.1), class = Intercept), prior(normal(0,
0.5), class = b) ) # Bayesian GLM: H0 fit_pDV_Ho_2 <- brm(pDV ~ 1,
data = df_BF, family = bernoulli(link = "logit"), prior = priors_logit2[-2,
], save_pars = save_pars(all = TRUE) ) # Bayesian GLM: H1 fit_pDV_H1_2
<- brm(pDV ~ 1 + F, data = df_BF, family = bernoulli(link = "logit"),
prior = priors_logit2, save_pars = save_pars(all = TRUE) ) # Bridge

```

```
sampling mLL_binom_H0_2 <- bridge_sampler(fit_pDV_H0_2, silent = TRUE)
mLL_binom_H1_2 <- bridge_sampler(fit_pDV_H1_2, silent = TRUE) "
```

Based on these models and bridge samples, we can now compute the Bayes factors in support for H1 (i.e., in support of a group-difference between F1 and F2). We can do so for the unrealistic prior for the intercept (prior mean of 0) and the more realistic prior for the intercept (prior mean of 2).

```
(BF_binom_H1_H0 <- bayes_factor(mLL_binom_H1, mLL_binom_H0))
```

```
## Estimated Bayes factor in favor of x1 over x2: 7.19449
```

```
(BF_binom_H1_H0_2 <- bayes_factor(mLL_binom_H1_2, mLL_binom_H0_2))
```

```
## Estimated Bayes factor in favor of x1 over x2: 29.60257
```

The results show that with the realistic prior for the intercept (prior mean = 2), the evidence for the H1 is quite strong, with a Bayes factor of $BF_{10} = 29.6$. With the unrealistic prior for the intercept (i.e., prior mean = 0), by contrast, the evidence for the H1 is much reduced, $BF_{10} = 7.2$, and now only modest in nature.

These results show that when performing Bayes factor analyses, not only can the priors for the effect of interest (here the group difference) impact the results, under certain circumstances priors for other model parameters can too, such as the prior mean for the intercept here. Such an influence will not always be strong, and can sometimes be negligible. There may be many situations, where the exact specification of the intercept does not have much of an effect on the Bayes factor for a group difference. However, such influences can in principle occur, especially in models with non-linear components. Therefore, it is very important to be careful in specifying realistic priors for all model parameters, also including the intercept. A good way to judge whether prior assumptions are realistic and plausible is prior predictive checks, where we simulate data based on the priors and the model and judge whether the simulated data is plausi-

ble and realistic. We perform such prior predictive checks in detail in the chapter on Bayesian workflow.

16.4 Bayes factor in Stan

The package `bridgesampling` allows for a straightforward calculation of Bayes factor for Stan models as well. All the limitations and caveats of Bayes factor discussed (16) apply to Stan code as much as they apply to `brms` code.³

An advantage of using Stan in comparison with `brms` is Stan's flexibility. We revisit the model implemented before in 10.4.2, and we want to assess the evidence for a *positive* effect of attentional load on pupil size against a similar model that assumes no effect. To do this, we assume the following likelihood:

$$p_size_n \sim Normal(\alpha + c_load_n \cdot \beta_1 + c_trial \cdot \beta_2 + c_load \cdot c_trial \cdot \beta_3, \sigma) \quad (16.14)$$

Define priors for all the β s as before, with the difference that we assume that β_1 can only be positive:

$$\begin{aligned} \alpha &\sim Normal(1000, 500) \\ \beta_1 &\sim Normal_+(0, 100) \\ \beta_2 &\sim Normal(0, 100) \\ \beta_3 &\sim Normal(0, 100) \\ \sigma &\sim Normal_+(0, 1000) \end{aligned} \quad (16.15)$$

The following Stan model is the direct translation of the new priors and likelihood.

³Stan also allows for specifying priors and likelihood with the notation `parameter ~ pdf_name(...)` rather than `target += pdf_name_lpdf(parameter | ...)` (as we have been doing in this book). However the former notation doesn't drop normalizing constants for the pdfs that it evaluates and it's not compatible with the calculation of Bayes factor with bridge sampling.

```

data {
    int<lower = 1> N;
    vector[N] c_load;
    vector[N] c_trial;
    vector[N] p_size;
}
parameters {
    real alpha;
    real<lower = 0> beta1;
    real beta2;
    real beta3;
    real<lower = 0> sigma;
}
model {
    // priors including all constants
    target += normal_lpdf(alpha | 1000, 500);
    target += normal_lpdf(beta1 | 0, 100) -
        normal_lccdf(0 | 0, 100);
    target += normal_lpdf(beta2 | 0, 100);
    target += normal_lpdf(beta3 | 0, 100);
    target += normal_lpdf(sigma | 0, 1000)
        - normal_lccdf(0 | 0, 1000);
    target += normal_lpdf(p_size | alpha + c_load * beta1 + c_trial * beta2 +
        c_load .* c_trial * beta3, sigma);
}

```

We fit it with 20000 iterations to ensure that the Bayes factor is stable (and we increase its `adapt_delta` parameter to avoid warnings):

```

data("df_pupil")
df_pupil <- df_pupil %>%
    mutate(c_load = load - mean(load),
          c_trial = trial - mean(trial))
ls_pupil <- list(
    p_size = df_pupil$p_size,
    c_load = df_pupil$c_load,
    c_trial = df_pupil$c_trial,

```

```
N = nrow(df_pupil)
)
pupil_pos <- system.file("stan_models", "pupil_pos.stan", package = "bcogsci")
fit_pupil_int_pos <- stan(
  file = pupil_pos,
  data = ls_pupil,
  warmup = 1000,
  iter = 20000,
  control = list(adapt_delta = .95)
)
```

The null model that we defined has $\beta_1 = 0$ and is written in Stan as follows:

```
data {
  int<lower = 1> N;
  vector[N] c_load;
  vector[N] c_trial;
  vector[N] p_size;
}
parameters {
  real alpha;
  real beta2;
  real beta3;
  real<lower = 0> sigma;
}
model {
  // priors including all constants
  target += normal_lpdf(alpha | 1000, 500);
  target += normal_lpdf(beta2 | 0, 100);
  target += normal_lpdf(beta3 | 0, 100);
  target += normal_lpdf(sigma | 0, 1000)
    - normal_lccdf(0 | 0, 1000);
  target += normal_lpdf(p_size | alpha + c_trial * beta2 +
    c_load .* c_trial * beta3, sigma);
}
generated quantities{
  real log_likelihood[N];
```

```

for (n in 1:N){

pupil_null <- system.file("stan_models", "pupil_null.stan", package = "bcogsci")
fit_pupil_int_null <- stan(
  file = pupil_null,
  data = ls_pupil,
  warmup = 1000,
  iter = 20000
)

```

We compare the models with bridge sampling:

```

lml_pupil <- bridge_sampler(fit_pupil_int_pos, silent = TRUE)
lml_pupil_null <- bridge_sampler(fit_pupil_int_null, silent = TRUE)
BF_att <- bridgesampling::bf(lml_pupil, lml_pupil_null)

```

BF_att

```
## Estimated Bayes factor in favor of lml_pupil over lml_pupil_null: 25.17274
```

We find that the data is 25.173 more likely under a model that assumes a positive effect of load than under a model that assumes no effect.

16.5 Bayes factors in theory and in practice

16.5.1 Bayes factors in theory: Stability and accuracy

One question that we can ask here is how stable and accurate the estimates of Bayes factors are. Importantly, the bridge sampling algorithm used to compute the Bayes factors needs a lot of posterior MCMC samples to obtain stable estimates of the Bayes factor. Running bridge sampling based on a too small an effective sample size (related to the number of posterior MCMC samples) will yield unstable estimates of the Bayes factor, such that repeated computations will yield different results. Moreover, even if the Bayes factor is approximated in a stable way, it is unclear

whether this approximate Bayes factor is equal to the true Bayes factor, or whether there is bias in the computation such that the approximate Bayes factor has a wrong value. We will investigate this below.

16.5.1.1 Instability due to the effective number of posterior samples

The number of posterior samples chosen in the Hamiltonian Markov Chain Monte Carlo (MCMC) sampler (called by `brms`), that is, the algorithms used to compute or approximate the posterior, can have a strong impact on the robustness of the results of the bridge sampling algorithm (i.e., on the resulting Bayes factor). Bridge sampling is a form of density estimation for which we have no good theoretical guarantees of MCMC sampling since the densities are no expectations. In the analyses presented above, we set the number of MCMC samples to a very large number of $n = 20000$ iterations. The sensitivity analysis therefore took a considerable amount of time. Indeed, the results from this analysis were stable, as we will show below.

Running the same analysis with less MCMC samples will induce some instability in the Bayes factor estimates based on the bridge sampling, such that running the same analysis twice would yield different results for the Bayes factor. Moreover, bridge sampling in itself may be unstable and may return different results for different bridge sampling runs on the same posterior MCMC samples (just because of different starting values). This is very concerning, as the results reported in a paper might not be stable if the number of posterior samples or effective sample size is not large enough. Indeed, the default number of posterior samples in `brms` is `iter = 2000` (and the default number of warmup samples is `warmup = 1000`). These defaults were not set to support bridge sampling, i.e., they were not defined for computation of densities to support Bayes factors. Instead, they are valid for posterior inference on expectations (e.g., posterior means) for models that are not too complex. However, when using these defaults for estimation of densities and the computation of Bayes factors, instabilities can arise.

For illustration, we perform the same sensitivity analysis again, now using the default number of 2000 iterations in `brms`. The posterior sampling process now runs much quicker. Moreover, we check the stability of the Bayes factors in the sensitivity analyses by repeating both sensitivity anal-

yses (with $n = 20000$ iterations and with the default number of $n = 2000$ iterations) a second time, to see whether the results for Bayes factors are stable.

```
## Warning: Ignoring unknown aesthetics: linetype
## Warning: Ignoring unknown aesthetics: shape
```

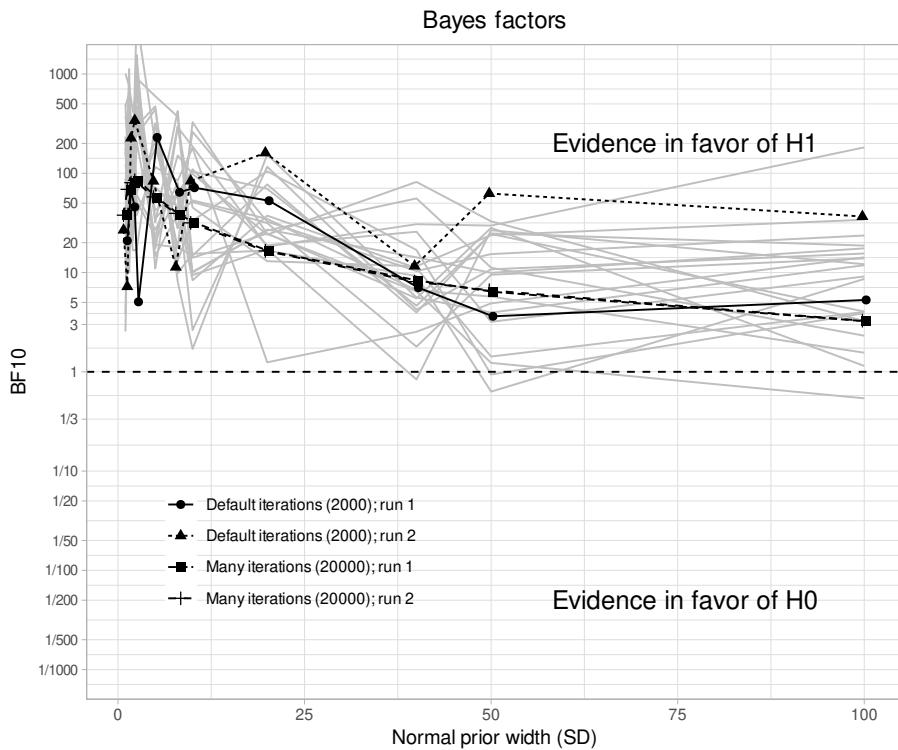


FIGURE 16.4: Effect of the number of samples on a prior sensitivity analysis for the Bayes factor. Grey lines show 20 runs with default number of iterations (2000).

The results displayed in Figure 16.4 show for the default number of iterations (i.e., number of posterior samples = 2000) that the resulting Bayes factors are highly unstable. They clearly deviate from the Bayes factors estimated with 20000 iterations, resulting in very unstable estimates. By contrast, the analyses using 20000 iterations provide nearly the exact same results in both analyses. The two lines lie virtually directly on top of each other; the points are jittered horizontally for better visibility.

This result demonstrates that it is necessary to use a large number of iterations when computing Bayes factors using `brms` and `bridge_sampler()`. In practice, one should compute the sensitivity analysis (or at least one of the models or priors) twice (as we did here) to make sure that the results are stable and sufficiently similar, in order to provide a good basis for reporting results.

By contrast, Bayes factors based on the Savage–Dickey method (as implemented in `brms`) can be unstable even when using a large number of posterior samples. This problem can arise especially when the posterior is very far from zero, and thus very large or very small Bayes factors are obtained. Because of this instability of the Savage–Dickey method in `brms`, we recommend not to use it. Instead, we recommend using bridge sampling, and to check the stability of the estimates by running the posterior sampling and bridge sampling at least twice to compare Bayes factors in order to check whether they are stable.

16.5.1.2 Inaccuracy of Bayes factor estimates: Does the estimate approximate the true Bayes factor well?

An important point about approximate estimates of Bayes factors using bridge sampling is that there are no strong guarantees for their accuracy. That is, even if we can show that the approximated Bayes factor estimate using bridge sampling is stable (i.e., when using sufficient effective samples, see the analyses above), even then it remains unclear whether the Bayes factor estimate actually is close to the true Bayes factor. Bridge sampling is a form of density estimation. Thus, bridge sampling does not rely on posterior expectations. Therefore, the accuracy of bridge sampling is unclear. In principle, it could very well be that the stably estimated Bayes factors based on bridge sampling are in fact biased, i.e., that they are not close to the correct (true) Bayes factor, but that the estimation exhibits bias and yields a different value. The technique of simulation-based calibration (SBC; Talts et al. 2018; Schad, Betancourt, and Vasishth 2020) can be used to investigate this question. We ask and investigate this question next (for details, see Schad et al. (2021)).

The procedure we take to analyze the accuracy of Bayes factor estimates using bridge sampling is simulation-based calibration (SBC; Talts et al. 2018; Schad, Betancourt, and Vasishth 2020; Schad et al. 2021). In this ap-

proach, the priors are used to simulate artificial data. Then, posterior inference is done on the artificial, simulated data, and the posterior can be compared to the prior. If the posteriors are equal to the priors, then this supports accurate computations. Applied to Bayes factor analyses, one defines a prior on the hypothesis space, i.e., one defines the prior probabilities for a null and an alternative model, specifying how likely each model is a priori. From these priors, one can randomly draw one hypothesis (model), e.g., $nsim = 500$ times. Thus, in each of 500 draws one randomly chooses one model (either H_0 or H_1), with the probabilities given by the model priors. For each draw, one first samples model parameters from their prior distributions, and then uses these sampled model parameters to simulate artificial data. For each simulated artificial data set, one can then compute marginal likelihoods and Bayes factor estimates using posterior HMC sampling and bridge sampling, and one can then compute the posterior probabilities for each hypothesis (i.e., how likely each model is a posteriori). As the last, and critical step in SBC, one can then compare the posterior model probabilities to the prior model probabilities. A key result in SBC is that if the computation of marginal likelihoods and posterior model probabilities is performed accurately by the bridge sampling procedure, i.e., without bias, that is, if the Bayes factor estimate is close to the true Bayes factor, then the posterior model probabilities should be the same as the prior model probabilities.

We here perform this SBC approach. Across the 500 simulations, we systematically vary the prior model probability from zero to one. For each of the 500 simulations we sample a model (hypothesis) from the model prior, then sample parameters from the priors over parameters, use the sampled parameters to simulate fake data, fit the null and the alternative model on the simulated data, perform bridge sampling for each model, compute the Bayes factor estimate between them, and compute posterior model probabilities. If the bridge sampling works accurately, then the posterior model probabilities should be the same as the prior model probabilities. Given that we varied the prior model probabilities from zero to one, the posterior model probabilities should also vary from zero to one. In Figure 16.5, we plot the posterior model probabilities as a function of the prior model probabilities. If the posterior probabilities are the same as the priors, then the local regression line and all points should lie on the diagonal.

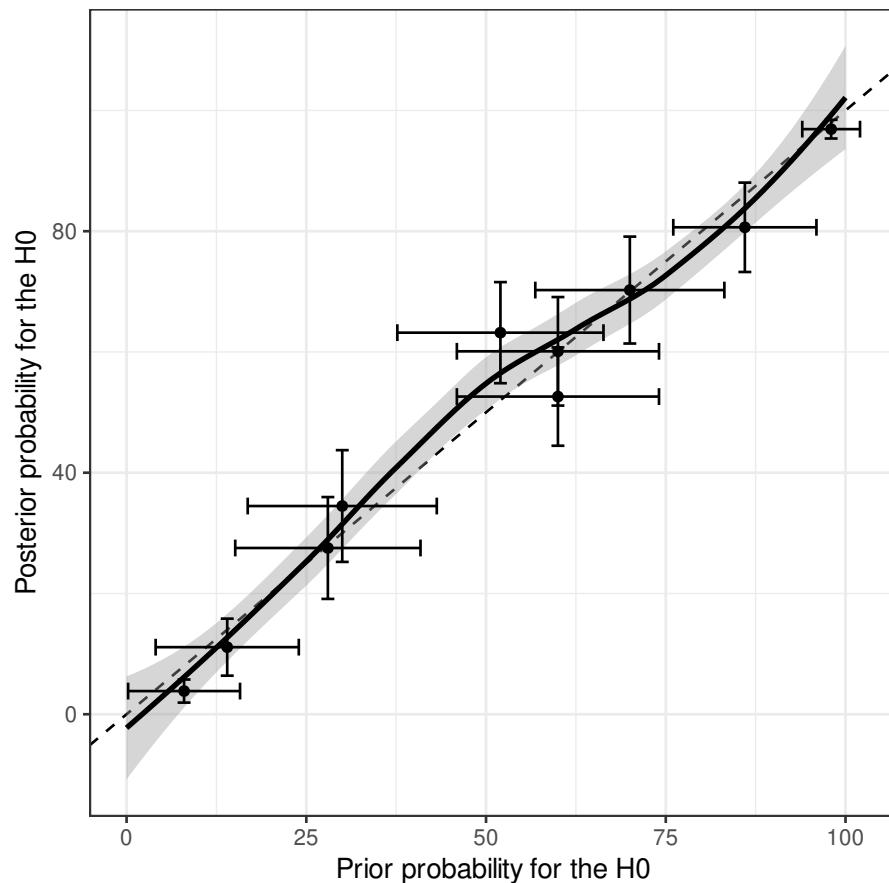


FIGURE 16.5: Posterior probabilities for the H_0 are plotted as a function of prior probabilities for the H_0 . If the approximation of the Bayes factor using bridge sampling is unbiased, then the data should be aligned along the diagonal (see dashed black line). The thick black line is a prediction from a local regression analysis. The points are average posterior probabilities as a function of a priori selected hypotheses for 50 simulation runs each. Errorbars represent 95 percent confidence intervals.

The results of this analysis in Figure 16.5 show that the local regression line is very close to the diagonal, and that the data points (each summarizing results from 50 simulations, with means and confidence intervals) also lie close to the diagonal. This importantly demonstrates that the estimated posterior model probabilities are close to their a priori values. This result shows that posterior model probabilities, which are based on the Bayes factor estimates from the bridge sampling, are unbiased for a large range of different a priori model probabilities.

This result is very important as it shows one example case where the Bayes factor approximation is accurate. Importantly, however, of course this demonstration is valid only for this one specific application case, i.e., with a particular data set, particular generalized linear mixed-effects models, specific priors for the parameters, and a specific comparison between nested models (for details on the underlying data, models, and priors see Schad et al. (2021)). Strictly speaking - if one wants to be sure that the Bayes factor estimate is accurate for a particular data analysis, then such a SBC validation analysis would have to be computed for every data analysis (for details, including code, on how to perform such an SBC see Schad et al. (2021)). However, the fact that the SBC yields such promising results for this first application case also gives some hope that the bridge sampling may be accurate also for other comparable data analysis situations.

Based on these results on the average theoretical performance of Bayes factor estimation, we next turn to a different issue - i.e., how Bayes factors depend on and vary with varying data, leading to bad performance in individual cases despite good average performance.

16.5.2 Bayes factors in practice: Variability with the data

16.5.2.1 Variation associated with the data (subjects, items, and residual noise)

A second, and very different, source limiting robustness of Bayes factor estimates derives from the variability that is observed with the data, i.e., among subjects, items, and residual noise. Thus, when repeating an experiment a second time in a replication analysis, using different subjects and items, will lead to different outcomes of the statistical analysis every time a new replication run is conducted. This limit to robustness is well known in frequentist analyses, as the “dance of p-values” (Cumming

2014), where over repeated replication attempts, p-values are not consistently significant across studies. Instead, the results yield highly different p-values each time a study is re-run. This can also be observed when simulating data from some known truth and re-running analyses on simulated data sets.

This same type of variability should also be present in Bayesian analyses (also see <https://daniellakens.blogspot.com/2016/07/dance-of-bayes-factors.html>). Here we investigate this type of variability in Bayes factor analyses by looking at a new example data analysis: we look at research on sentence comprehension, and specifically on effects of cue-based retrieval interference.

16.5.2.2 Example: Facilitatory interference effects

In the following, we will look at experimental studies that investigated cognitive mechanisms underlying a well-studied phenomenon in sentence comprehension. The example we consider here is the agreement attraction configuration below, where the ungrammatical sentence (2) seems more grammatical than the equally ungrammatical sentence (1):

- (1) The key to the cabinet are in the kitchen.
- (2) The key to the cabinets are in the kitchen.

Both sentences are ungrammatical because the subject (“key”) does not agree with the verb in number (“are”). Sentences such as (2) are often found to have shorter reading times at the verb (“are”) compared to (1) (for a meta-analysis see Jäger, Engelmann, and Vasishth 2017). Such shorter reading times are sometimes referred to as “facilitatory interference” (Dillon 2011). One proposal explaining the shorter reading times is that the attractor word (here, cabinets) agrees locally in number with the verb, leading to an illusion of grammaticality. This is an interesting phenomenon as the plural versus singular feature of the attractor noun (“cabinet/s”) is not the subject, and therefore, under the rules of English grammar, is not supposed to agree with the number marking on the verb. That agreement attraction effects are consistently observed indicates that some non-compositional processes are taking place. An account of agreement attraction effects in language processing that is based on a full computational implementation (which is in the ACT-R framework, Anderson et al.

(2004)), explains such agreement attraction effects in ungrammatical sentences as a result of retrieval-based working memory mechanisms (Engelmann, Jäger, and Vasishth 2020; cf. Hammerly, Staub, and Dillon 2019). Agreement attraction in ungrammatical sentences has been investigated many times in similar experimental setups with different dependent measures such as self-paced reading and eye-tracking. It is generally believed to be a robust empirical phenomenon, and we choose it for analysis here because it provides an example of a relatively robust effect in cognitive science.

Here, we look at a self-paced reading study on agreement attraction in Spanish by Lago et al. (2015). We estimate a fixed effect for the experimental condition agreement attraction (x ; i.e., sentence type), against a null model where the fixed effect of sentence type is excluded. For the agreement attraction effect of sentence type, we use sum contrast coding (i.e., -1 and +1). We run a multilevel model with the following formula in `brms`: $rt \sim 1 + x + (1 + x | subj) + (1 + x | item)$, where rt is reading time, we have random variation associated with subjects and with items, and we assume that reading times follow a log-normal distribution: `family = lognormal()`.

As a next step, we determine priors for the analysis of these data.

16.5.2.3 Determine priors using meta-analysis

One good way to obtain priors for Bayesian analyses, and specifically for Bayes factor analyses, is to use results from meta-analyses on the subject. Here, we take the prior for the experimental manipulation of agreement attraction from the published meta-analysis (Jäger, Engelmann, and Vasishth 2017).⁴

The mean effect size (difference in reading time between the two experimental conditions) in the meta-analysis is -22 milliseconds (ms), with 95% CI = [-36 – 9] (cf., Jäger, Engelmann, and Vasishth 2017, Table 4). This means that on average, the target word (i.e., the verb) in sentences such as (2) is on average read 22 milliseconds faster than in sentences such

⁴This meta-analysis already includes the data that we want to make inference about; thus, this meta-analysis estimate is not really the right estimate to use, since it involves using the data twice. We ignore this detail here because our goal is simply to illustrate the approach.

as (1). The size of the effect is measured on the millisecond scale, assuming a normal distribution of effect sizes across studies.

However, individual reading times usually do not follow a normal distribution. Instead, a better assumption about the distribution of reading times is a log-normal distribution. This is what we will assume in the `brms` model. Therefore, to use the prior from the meta-analysis in the Bayesian analysis, we have to transform the prior values from the millisecond scale to log millisecond scale.

We have performed this transformation in Schad et al. (2021). Based on these calculations, the prior for the experimental factor of interference effects is set to a normal distribution with mean = -0.03 and standard deviation = 0.009. For the other model parameters, we use mildly informative priors based on our recent analysis of a principled Bayesian workflow (Schad, Betancourt, and Vasishth 2020).

```
priors <- c(
  prior(normal(6, 0.5), class = Intercept),
  prior(normal(-0.03, 0.009), class = b),
  prior(normal(0, 0.5), class = sd),
  prior(normal(0, 1), class = sigma),
  prior(lkj(2), class = cor)
)
```

16.5.2.4 Running a hierarchical Bayesian analysis

We now run a `brms` model on the data. We use a large number of iterations (`iter = 10000`) such that we can compute stable Bayes factor estimates. We use bridge sampling to estimate the Bayes factor of the alternative model, which includes a fixed effect for the experimental condition agreement attraction (`x`; i.e., sentence type), against the null model where the fixed effect of sentence type is excluded. For the agreement attraction effect of sentence type, we use sum contrast coding (i.e., -1 and +1).

We estimate Bayes factors between a full model, where the effect of agreement attraction is included, and a null model, where the effect of agreement attraction is absent, using the command

`bayes_factor(lml_m1_lagoE1, lml_m0_lagoE1)`. It computes the Bayes factor BF_{10} , that is, the evidence of the alternative over the null.

We first show the results from the posterior analyses:

```
round(fixef(m1_lagoE1), 3)
```

```
##           Estimate Est.Error Q2.5 Q97.5
## Intercept     6.02      0.06  5.90  6.13
## x            -0.03      0.01 -0.04 -0.01
```

They show that for the fixed effect `x`, capturing the agreement attraction effect, the 95% credible interval does not overlap with zero. This indicates that there is some hint that the effect may have the expected negative direction, reflecting shorter reading times in the plural condition. As mentioned earlier, this does not provide a direct test of the hypothesis that the effect exists and is not zero. This is not tested here, because we did not specify the null hypothesis of zero effect explicitly. We can, however, make inference on this null hypothesis by using the Bayes factor. We now look at the Bayes factor of the alternative model compared to the null model (BF_{10}):

```
h_lagoE1$bf
```

```
## [1] 6.29
```

It shows a Bayes factor of 6, suggesting that there is some support for the alternative model, which contains the fixed effect of agreement attraction. That is, this provides evidence for the alternative hypothesis that there is a difference between the experimental conditions, i.e., a facilitatory effect in the plural condition of the size derived from the meta analysis.

16.5.2.5 Variability of the Bayes factor: Posterior simulations

One way to investigate how variable the outcome of Bayes factor analyses can be (given that the Bayes factor is computed in a stable and accurate way), is to run posterior simulations based on a fitted model. That is, one can assume that the truth is approximately known (as approximated by the posterior model fit), and that based on this “truth” several artificial

data sets are simulated. Computing the Bayes factor analysis again on the simulated data can provide some insight into how variable the Bayes factor will be in a situation where the “true” data generating process is always the same, and where variations in Bayes factor results have to be attributed to random noise in subjects, items, residual variation, and to uncertainty about the precise true parameter values.

We can take the Bayesian hierarchical model fitted to the data from Lago et al. (2015), and run posterior predictive simulations. In these simulations, one takes posterior samples for the model parameters (i.e., $p(\Theta | y)$), and for each posterior sample of the model parameters, one can simulate new data \tilde{y} from the model $p(\tilde{y} | \Theta)$. That is, posterior predictive simulations are a Bayesian way to perform artificial data simulation. The posterior predictive distribution is written as: $p(\tilde{y} | y) = \int p(\tilde{y} | \Theta)p(\Theta | y)d\Theta$. Posterior predictive simulations from the fitted `brms` model can be performed using the `brms`-function `posterior_predict()`.

```
pred_lagoE1 <- posterior_predict(m1_lagoE1)
```

The question that we are interested in here now is, how much information is contained in this posterior simulated data. That is, we can run Bayesian models on this posterior simulated data and compute Bayes factors to test whether in the simulated data there is evidence for agreement attraction effects. Of great interest to us is then the question of how variable the results of these Bayes factor analyses will be across different simulated replications of the same study.

We now perform this analysis for 50 different artificial data sets simulated from the posterior predictive distribution. For each of these data sets, we can proceed in exactly the same way as we did for the real observed experimental data. That is, 50 times, we again fit the same `brms` model, now to the simulated data, and using the same prior as before. For each simulated data set, we use bridge sampling to compute the Bayes factor of the alternative model compared to a null model where the agreement attraction effect (fixed effect predictor of sentence type, `x`) is set to 0. For each simulated posterior predictive data set, we store the resulting Bayes factor. We again use the prior from the meta analysis.

16.5.2.6 Visualize distribution of Bayes factors

We can now visualize the distribution of Bayes factors (BF_{10}) across posterior predictive distributions by plotting a histogram. Values larger than one in this histogram indicate evidence for the alternative model (H_1) that agreement attraction effects exist (i.e., the sentence type effect is different from zero), and Bayes factor values smaller than one indicate evidence for the null model (H_0) that no agreement attraction effect exists (i.e., the difference in reading times between experimental conditions is zero).

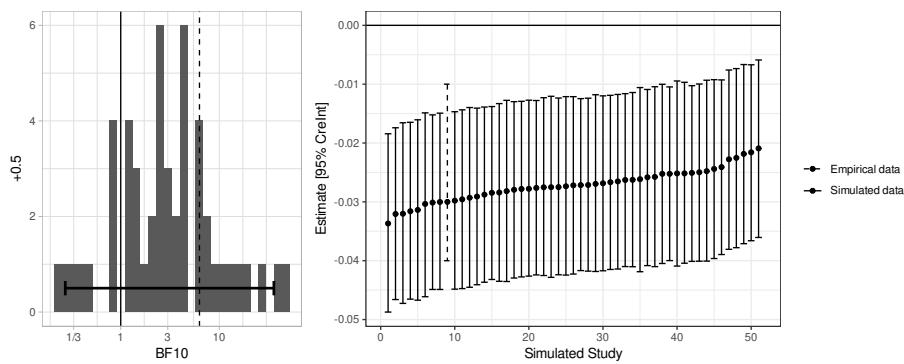


FIGURE 16.6: Left panel: Histogram of Bayes factors (BF_{10}) of the alternative model over the null model in 50 simulated data sets. The vertical solid black line shows equal evidence for both hypotheses; the dashed red line shows the Bayes factor computed from the empirical data; the horizontal error bar shows 95 percent of all Bayes factors. Right panel: Estimates of the facilitatory effect of retrieval interference and 95 percent credible intervals across all simulations (solid lines) and the empirically observed data (dashed line).

The results show that the Bayes factors are quite variable. Although all data sets are simulated from the same posterior predictive distribution, the Bayes factor results are as different as providing moderate evidence for the null model ($BF_{10} < 1/3$) or providing strong evidence for the alternative model ($BF_{10} > 10$). The bulk of the simulated data sets provide moderate or anecdotal evidence for the alternative model. That is, much like the “dance of p-values” (Cumming 2014), this analysis reveals a “dance of the Bayes factors” with simulated repetitions of the same study. The variability in these results shows that a typical cognitive or psycholinguistic

tic data set is not necessarily highly informative for drawing firm conclusions about the hypotheses in question.

What is driving these differences in the Bayes factors between simulated data sets? One obvious reason why the outcomes may be so different is that the difference in reading times between the two sentence types, that is, the experimental effect that we wish to make inferences about, may vary based on the noise and uncertainty in the posterior predictive simulations. It is therefore interesting to plot the Bayes factors from this simulated data set as a function of the difference in simulated reading times between the two sentence types as estimated in the Bayesian model. That is, we extract the estimated mean difference in reading times at the verb between plural and singular attractor conditions from the fixed effects of the Bayesian model, and plot the Bayes factor as a function of this difference (together with 95% credibility intervals).

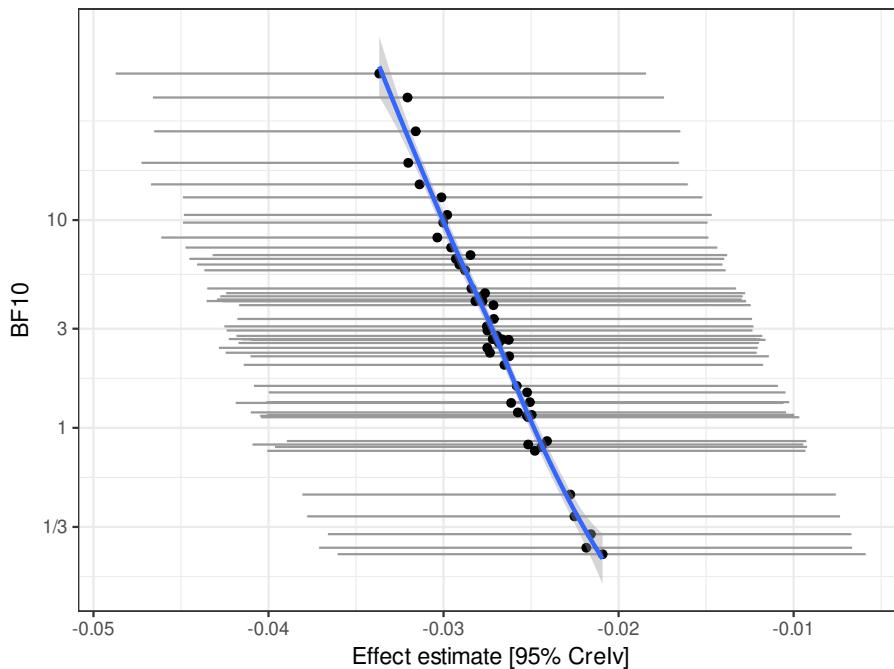


FIGURE 16.7: Bayes factor (BF₁₀) as a function of the estimate (with 95 percent credible intervals) of the facilitatory effect of retrieval interference across 50 simulated data sets. The prior is from a meta analysis.

The results (displayed in Figure 16.7) show that the mean difference in reading times between experimental conditions varies across posterior predictive simulations. This indicates that the experimental data and design contain a limited amount of information about the effect of interest. Of course, if the data is not stable, Bayes factor analyses based on this simulated data cannot be stable across simulations either. Accordingly, as is clear from Figure 16.7, the difference in mean reading times between experimental conditions is indeed a main driving force for the Bayes factor calculations (other model parameters don't show a close association; Schad et al. 2021).

In Figure 16.7, as the difference between reading times becomes more negative, that is, the faster the plural noun condition (i.e., “cabinets” in the example; sentence 2) is read compared to the singular noun condition (i.e., “cabinet”; example sentence 1), the larger the Bayes factor BF_{10} becomes, indicating that the evidence in favor of the alternative model increases. By contrast, when the difference between reading times becomes less negative, i.e., the plural condition (sentence 2) is not read much faster than the singular condition (sentence 1), then the Bayes factor BF_{10} decreases to values smaller than 1. Importantly, this behavior occurs because we are using our informative priors from the meta analysis, where the prior mean for the agreement attraction effect is not centered at a mean of zero, but has a negative value (i.e., a prior mean of -0.03). Therefore, differences in reading times that are less negative / more positive than this prior mean are more in line with a null model of no effect. This also leads to the striking observation that the 95% credible intervals are quite consistent and all do not overlap with zero, whereas the Bayes factor results are far more variable. This should alarm researchers who use the 95% credible interval to decide whether an effect is present or not, i.e., to make a discovery claim.

Computing Bayes factors for such a prior with a non-zero mean asks the very specific question of whether the data provide more evidence for the effect size obtained from the meta-analysis compared to the absence of any effect.

The important lesson to learn from this analysis is that Bayes factors can be quite variable for different data sets assessing the same phenomenon. Individual data sets in the cognitive sciences often do not contain a lot

of information about the phenomenon of interest, even when - as is the case here with agreement attraction - the phenomenon is thought to be a relatively robust phenomenon. For a more detailed investigation of how Bayes factors can vary with data, in both simulated and real replication studies, we refer the reader to Schad et al. (2021).

16.6 Summary

Bayes factors are a very important tool in Bayesian data analysis. They allow the researcher to quantify the evidence in favor of certain effects in the data by comparing a full model, which contains a parameter corresponding to the effect of interest, with a null model, that does not contain that parameter. We saw that Bayes factor analyses are highly sensitive to priors specified for the parameters; this is true both for the parameter corresponding to the effect of interest, but also sometimes for priors relating to other parameters in the model, such as the intercept. It is therefore very important to perform prior predictive checks to select good and plausible priors. Moreover, sensitivity analyses, where Bayes factors are investigated for differing prior assumptions, should be standardly reported in any analysis involving Bayes factors. We studied theoretical aspects of Bayes factors and saw that bridge sampling requires a very large effective sample size in order to obtain stable results for approximate Bayes factors. Therefore, one should always perform a Bayes factor analysis at least twice to ensure that the results are stable. Bridge sampling comes with no strong guarantees concerning its accuracy, and we saw that simulation-based calibration can be used to evaluate the accuracy of Bayes factor estimates. Last, we learned that Bayes factors can strongly vary with the data. In the cognitive sciences, the data are — even for relatively robust effects — often not stable due to small effect sizes and limited sample size. Therefore, also the resulting Bayes factors can strongly vary with the data. As a consequence, only large effect sizes, large sample studies, and/or replication studies can lead to reliable inferences from empirical data in the cognitive sciences.

16.7 Further reading

- A detailed explanation on how bridge sampling works can be found in Quentin F Gronau et al. (2017b), and more details about the bridgesampling package can be found in Quentin F Gronau, Singmann, and Wagenmakers (2017).
 - Wagenmakers et al. (2010) provides a complete tutorial and the mathematical proof of the Savage-Dickey method.
 - For a Bayes Factor Test calibrated to investigate replication success, see Verhagen and Wagenmakers (2014).
 - An argument against null hypothesis testing with Bayes Factors appears in this blog post by Andrew Gelman: <https://statmodeling.stat.columbia.edu/2019/09/10/i-hate-bayes-factors-when-theyre-used-for-null-hypothesis-significance-testing/>
 - An argument in favor of null hypothesis testing with Bayes Factor as an approximation (but assuming realistic effects): <https://statmodeling.stat.columbia.edu/2018/03/10/incorporating-bayes-factor-understanding-scientific-information-replication-crisis/>
 - A visualization of the distinction between Bayes factor and k-fold cross-validation is in a blog post by Fabian Dablander, <https://tinyurl.com/47n5cte4>.
-

16.8 Exercises

Exercise 16.1.

Is there evidence for differences in the effect of Cloze probability among the subjects? Use Bayes factor to compare the log Cloze probability model that we examined in section 16.2.2 with a similar model but that incorporates the strong assumption of no difference between subjects for the effect of Cloze ($\tau_{u_2} = 0$).

Exercise 16.2. Is there evidence for the claim that subject relative clause are easier to process than object relative clauses?

Consider again the reading time data coming from Experiment 1 of Grodner and Gibson (2005) presented in exercise 5.2. Try to quantify the evidence against the null model (no population-level reading times difference between SRC and ORC) relative to the following alternative models:

- a. $\beta \sim Normal(0, 1)$
- b. $\beta \sim Normal(0, .1)$
- c. $\beta \sim Normal(0, .01)$
- d. $\beta \sim Normal_+(0, 1)$
- e. $\beta \sim Normal_+(0, .1)$
- f. $\beta \sim Normal_+(0, .01)$

(A $Normal_+(.)$ prior can be set in brms by defining a lower boundary as 0, with the argument `lb = 0`.)

What is the Bayes factor against the null and in favor of the alternative models a-f?

Exercise 16.3. Is there evidence for the claim that sentences with subject relative clauses are easier to comprehend?

Consider now the question response accuracy of the data of Experiment 1 of Grodner and Gibson (2005).

- a. Compare a model that assumes that RC type affects question accuracy on the population and by-subjects and by-items with a *null model* that assumes that there is no population-level present.
- b. Compare a model that assumes that RC type affects question accuracy on the population and by-subjects and by-items with another *null model* that assumes that there is no population-level or group-level present, that is no by-subject or by-item effect. What's the meaning of the results of the Bayes factor analysis.

Assume that for the effect of RC on question accuracy, $\beta \sim Normal(0, .1)$ is a reasonable prior, and that for all the variance components, the same prior, $\tau \sim Normal(0, 1)$, is a reasonable prior.

Exercise 16.4. Bayes factor and bounded parameters using Stan.

Re fit the data of a single subject pressing a button repeatedly from 4.2 from `data("df_spacebar")` coding it in Stan.

Start by assuming the following likelihood and priors:

$$rt_n \sim LogNormal(\alpha + c_{trial_n} \cdot \beta, \sigma) \quad (16.16)$$

$$\begin{aligned} \alpha &\sim Normal(6, 1.5) \\ \beta &\sim Normal_+(0, .1) \\ \sigma &\sim Normal_+(0, 1) \end{aligned} \quad (16.17)$$

Use the Bayes factor to answer the following questions:

- a. Is there evidence for any effect of trial number in comparison with no effect?
- b. Is there evidence for a positive effect of trial number (as the subject reads further, they slowdown) in comparison with no effect?
- c. Is there evidence for a negative effect of trial number (as the subject reads further, they speedup) in comparison with no effect?
- d. Is there evidence for a positive effect of trial number in comparison with a negative effect?

(Expect very large Bayes factors in this exercise.)

17

Cross-validation

A popular way to evaluate and compare models is on their ability to make predictions for “out-of-sample data”, that is, future or unseen observations, using what we learned from the observed data. We can use cross-validation to test which of the models under consideration is able to learn the most from our data in order to make the better predictions. However, in cognitive science, our objective will rarely be to predict future observations, but rather to compare how well different models fare in accounting for our own observations. The objective of cross-validation is to avoid over optimistic predictions based on using the data to estimate the parameters of our model, and then using these estimates to predict the same data, that is, to use the data twice. The basic idea behind cross-validation is that we fit the models with a large subset of the data, the *training set*, and then we predict a smaller part of the data, the *held-out set*. In order to treat the entire data set as a held-out set, and to evaluate the predictive accuracy using every observation, we keep rotating the training set and the held-out set until we evaluated how well we predict our entire data set.

17.1 Expected log predictive density of a model

In order to compare the quality of the posterior predictions of two models, we will use a *utility* function or a *scoring rule* (see Gneiting and Raftery 2007 for a review on scoring rules). The logarithmic score rule (Good 1952), shown in (17.1), has been proposed as a good rule for assessing the posterior predictive distribution of a candidate model \mathcal{M}_1 given the data y , because it takes into account the uncertainty of the predictions (in comparison with, for example, the mean square error). If new observations

are well-accounted by the posterior predictive distribution, then the density of the posterior predictive distribution is high and so is its logarithm.

$$u(\mathcal{M}_1, y_{pred}) = \log p(y_{pred} | y, \mathcal{M}_1) \quad (17.1)$$

Unlike in the Bayes factor, the prior is absent from (17.1). However, the prior does have a role here: The posterior predictive distribution is based on the posterior distribution $p(\Theta | y)$ (where Θ is a vector of all the parameters of the model), which in turn depends on both priors and likelihood together according to Bayes' rule. Recall Equation (3.10) of section 3.5, repeated here for convenience:

$$p(y_{pred} | y) = \int_{\Theta} p(y_{pred} | \Theta) p(\Theta | y) d\Theta \quad (17.2)$$

Although it is implicit in Equation (17.2), we are conditioning on the model under consideration:

$$p(y_{pred} | y, \mathcal{M}_1) = \int_{\Theta} p(y_{pred} | \Theta, \mathcal{M}_1) p(\Theta | y, \mathcal{M}_1) d\Theta \quad (17.3)$$

The unobserved data, y_{pred} are unknown to the utility function, so the utility function as presented in (17.1) cannot be evaluated. For this reason, we marginalize over *all possible future data* (calculating $E[\log p(y_{pred} | y, \mathcal{M}_1)]$); this expression is called the expected log predictive density of model \mathcal{M}_1 :

$$elpd = u(\mathcal{M}_1) = \int_{y_{pred}} p_t(y_{pred}) \log p(y_{pred} | y, \mathcal{M}_1) dy_{pred} \quad (17.4)$$

where p_t is the true data generating distribution. If we consider a set of models, the model with the highest *elpd* is the model with the predictions that are the closest to the ones of the true data generating process.¹ The intuition behind Equation (17.4) is that we are evaluating the predictive distribution of \mathcal{M}_1 over all possible future data weighted by how likely the

¹Maximizing the *elpd* in (17.4) is also equivalent to minimizing the Kullback–Leibler (KL) divergence from the true data generating distribution $p_t(y_{pred})$ to the *posterior predictive distribution* of the candidate model \mathcal{M}_1 .

future data is according to its true distribution. This means that observations that are very likely according to the true model will have a higher weight than unlikely ones.

But we don't know the true data-generating distribution, p_t ! If we knew it, we wouldn't be looking for the best model, since p_t is the best model.

We can use the observed data distribution as a proxy for the true data generation distribution. So instead of weighting the predictive distribution by the true density of all possible future data, we just use the N observations that we have. We can do that because our observations are presumed to be samples from the true distribution of the data: Under this assumption, observations with higher likelihood according to the true distribution of the data will also be more commonly obtained. This means that instead of integrating, we sum the posterior predictive density of the observations and we give to each observation the same weight; this is valid because observations that are more common will be already appearing more often (see also Box 17.1). This quantity is called *log pointwise predictive density* or *lpd* (without the $1/N$ in Vehtari, Gelman, and Gabry 2017b):

$$lpd = \frac{1}{N} \sum_{n=1}^N \log p(y_n | y, \mathcal{M}_1) \quad (17.5)$$

The *lpd* is an overestimate of *elpd* for actual future data, because the parameters of the posterior predictive distribution are estimated with the same observations that we are considering out-of-sample. Incidentally, this also explains why posterior predictive checks are generally optimistic and good fits cannot be taken too seriously, but they do capture very strong model misspecifications.²

However, we can obtain a less optimistic estimate of the predictive performance of a model using cross-validation (Geisser and Eddy 1979) as we explain next (there are, however, also other alternatives to cross-validation; these are presented in Vehtari and Ojanen 2012).

²The double use of the data is also a problem when one relies on information criteria like the Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC).

Box 17.1. How do we get rid of the integral in the approximation of elpd?

As an example, let's imagine that we have N observations in an experiment. The true generative process (which is always unknown to us) is a Beta distribution as follows:

$$p_t(y) = \text{Beta}(y|1, 3) \quad (17.6)$$

We set N and we can only observe y :

```
N <- 10000
y_data <- rbeta(N, 1, 3)
head(y_data)

## [1] 0.2527 0.8196 0.0278 0.3778 0.5326 0.4972
```

Let's say that we fit the Bayesian model, \mathcal{M}_1 , and somehow, after getting the posterior distribution, we are able derive the analytical form of its posterior predictive distribution for the model:

$$p(y_{pred} | y, \mathcal{M}_1) = \text{Beta}(y_{pred} | 2, 2) \quad (17.7)$$

This distribution will tell us how likely different future observations will be, and it also entails that our future observations will be bounded by 0 and 1. (Any observation outside this range will have a probability density of zero).

Imagine that we could know the true distribution of the data, p_t , which is conveniently close to our posterior predictive distribution. This means that equation (17.4), repeated below, is simple enough, and we know all its terms:

$$\text{elpd} = u(\mathcal{M}_1) = \int_{y_{pred}} p_t(y_{pred}) \log p(y_{pred} | y, \mathcal{M}_1) dy_{pred} \quad (17.8)$$

We can compute this quantity in R. Notice that we don't introduce

the data at any point. However, the data had to be used when p , the posterior predictive distribution, was derived; we skipped that step here.

```
# True distribution
p_t <- function(y) dbeta(y, 1, 3)
# Predictive distribution
p <- function(y) dbeta(y, 2, 2)
# Integration
integrand <- function(y) p_t(y) * log(p(y))
integrate(f = integrand, lower = 0, upper = 1)

## -0.375 with absolute error < 0.00000068
```

Because we'll never know p_t , we'll approximate this integral using our data, y_{data} . We see below that we can approximate the integration without any reference to p_t as we did in (17.5):

```
1/N * sum(log(p(y_data)))
```



```
## [1] -0.367
```

The main problem with this approach is that we are using y_{data} twice, once to derive p , the predictive posterior distribution, and once for the approximation of elpd . We'll see that cross-validation approaches rely on deriving the posterior predictive distribution with part of the data, and estimating the approximation to elpd with unseen data. (Don't worry that we don't know the analytical form of posterior predictive distribution, we saw that we could generate samples from that distribution based on the likelihood and posterior samples.)

17.2 K-fold and leave-one-out cross-validation

The basic idea of K-fold cross-validation (K-fold-CV) is to split the N observations of our data in K subsets, such that each subset is used as a valida-

tion (or held-out) set, D_k , while the remaining set (the training set), D_{-k} is used for estimating the parameters and approximating p_t , the true data distribution. The leave-one-out cross-validation (LOO-CV) method represent a special case of K-fold-CV where the training set only excludes one observation ($K = N$). We estimate \widehat{elpd} as follows.

$$\widehat{elpd} = \frac{1}{N} \sum_{n=1}^N \log p(y_n | D_{\setminus n}, \mathcal{M}_1) \quad (17.9)$$

In Equation (17.9), each observation, y_n , belongs to a certain “validation” fold, D_k , and the predictive accuracy of y_n is evaluated based on a posterior predictive model trained on the set, $D_{\setminus n}$, which is the complete data set excluding the validation fold that contains the observation n . This means that the posterior predictive distribution is used to evaluate y_n , even though the posterior predictive distribution was derived without having information from that y_n -th observation (in other words, the model was trained without that observation in the subset of the data, D_{-k}). In K-fold-CV, several observations are held out in same (validation) fold. This means that the held-out observations are split among K folds, and $D_{\setminus n}$, the data used to derive the posterior predictive distribution, contain only a proportion of the observations; this proportion is $(1 - 1/K)$. By contrast, in leave-one-out cross-validation, the held-out data set includes only one observation. That is, $D_{\setminus n}$ contains the entire data set except for one data point, y_n , with $n = 1, \dots, N$. Box 17.2 explains the algorithm in detail.

Vehtari, Gelman, and Gabry (2017b) define the expected log *pointwise* predictive density of the observation y_n as follows:

$$\widehat{elpd}_n = \log p(y_n | D_{\setminus n}, \mathcal{M}_1) \quad (17.10)$$

This quantity indicates the predictive accuracy of the model \mathcal{M}_1 for a single observation, and it is reported in the package `loo` and also in `brms`. In addition, the `loo` package uses the sum of the expected log pointwise predictive density, $\sum \widehat{elpd}_n$ (Equation (17.9) without $\frac{1}{N}$) as a measure of predictive accuracy (this is referred as `elpd_loo` or `elpd_kfold` by `loo` and `brms` packages). For model comparison, the difference between the $\sum \widehat{elpd}_n$ of

competing models can be computed, including the standard deviation of the sampling distribution of the difference. It's important to notice that we are calculating an approximation to the expectation that we actually want to compute, \widehat{elpd} , and thus we always need to consider its inherent randomness (Vehtari, Simpson, et al. 2019).

Unlike what is common with information criterion methods (such as Akaike Information Criterion, AIC, and Deviance Information Criterion, DIC), higher \widehat{elpd} means higher predictive accuracy. An alternative to using \widehat{elpd} is to examine $-2 \times \widehat{elpd}$, which is equivalent to deviance, and is called the LOO Information Criterion (LOOIC).

The approximation to the true data generating distribution is worse when fewer observations are used, and thus ideally we would set $K = N$, and thus computing LOO-CV rather than K-fold-CV. The main advantage of LOO-CV is its robustness, since the training set is as similar as possible to the real data, while the same observations are never used simultaneously for training and evaluating the predictions. A major disadvantage is the computational burden (Vehtari and Ojanen 2012), since we need to fit a model as many times as the number of observations. The package `loo` provides an approximation to LOO-CV, Pareto smoothed importance sampling leave-one-out (PSIS-LOO; Vehtari and Gelman 2015; Vehtari, Gelman, and Gabry 2017b), which as we show next is relatively straightforward to use in `brms` and in Stan models (see <https://mc-stan.org/loo/articles/loo2-with-rstan.html>). However, in some cases, its estimates can be unreliable, which is indicated by the estimated shape parameter \hat{k} of the generalized Pareto distribution. In those cases, where one or several pointwise predictive density have associated large (larger than 0.5 or 0.7, see <https://mc-stan.org/loo/reference/pareto-k-diagnostic.html>) \hat{k} , either (i) the problematic predictions can be refitted with exact LOO-CV, (ii) one can try some additional computations using the existing posterior sample based on the moment matching approximation (see <https://mc-stan.org/loo/articles/loo2-moment-matching.html> and Paananen et al. 2021), or (iii) alternatively one can abandon PSIS-LOO-CV and use K-fold-CV, with K typically set to 10.

One of the main disadvantages of cross-validation (in comparison with Bayes factor at least) is that the numerical difference in predictive accuracy is hard to interpret. As a rule of thumb, it has been suggested that if

the `elpd` difference (`elpd_diff` in `loo` package) is less than 4, the difference is small, and if it is larger than 4, one should compare that difference to its standard error (`se_diff`).³

Box 17.2. The cross-validation algorithm

Here we spell out the Bayesian cross-validation algorithm in detail:

1. Split the data pseudo-randomly into K held-out or validation sets D_k , (where $k = 1, \dots, K$) that are a fraction of the original data, and K training sets, D_{-k} . The length of the held-out data-vector D_k is approximately $1/K$ -th the size of the full data set. In general, it is common to use $K = 10$ for K-fold-CV, and K should be set to the number of observations for LOO-CV.
2. Fit K models using each of the K training sets, and obtain posterior distributions $p_{-k}(\Theta) = p(\Theta | D_{-k})$, where Θ is the vector of model parameters.
3. Each posterior distribution $p(\Theta | D_{-k})$ is used to compute the predictive accuracy for each held-out data-point y_n :

$$\widehat{\text{elpd}}_n = \log p(y_n | D_{-k}) \quad (17.11)$$

Given that the posterior distribution $p(\Theta | D_{-k})$ is summarized by S samples, the log predictive density for each data point y_n in a subset k can be approximated as follows:

$$\widehat{\text{elpd}}_n = \log \left(\frac{1}{S} \sum_{s=1}^S p(y_n | \Theta^{k,s}) \right) \quad (17.12)$$

where $\Theta^{k,s}$ corresponds to the sample s of the posterior of the model fit to the validation set $-k$.

³See <https://avehtari.github.io/modelselection/CV-FAQ.html>

5. We obtain the $\text{elpd}_{k\text{fold}}$ (or elpd_{loo}) for all the held-out data points by summing up the $\widehat{\text{elpd}}_n$:

$$\text{elpd}_{k\text{fold}} = \sum_{n=1}^N \widehat{\text{elpd}}_n \quad (17.13)$$

We can also compute the standard deviation of the sampling distribution (the standard error) by multiplying the standard deviation (or square root of variance) of the N components by \sqrt{N} . Letting \widehat{ELPD} be the vector $\widehat{\text{elpd}}_1, \dots, \widehat{\text{elpd}}_N$, we can write:

$$\text{se}(\widehat{\text{elpd}}) = \sqrt{N \text{Var}(\widehat{ELPD})} \quad (17.14)$$

The difference between the $\text{elpd}_{k\text{fold}}$ of two competing models, \mathcal{M}_1 and \mathcal{M}_2 , is a measure of relative predictive performance. We can also compute the standard error of their difference using the formula discussed in Vehtari, Gelman, and Gabry (2017b).

$$\text{se}(\widehat{\text{elpd}}_{\mathcal{M}_1} - \widehat{\text{elpd}}_{\mathcal{M}_2}) = \sqrt{N \text{Var}(\widehat{ELPD}_{\mathcal{M}_1} - \widehat{ELPD}_{\mathcal{M}_2})} \quad (17.15)$$

17.3 Testing the N400 effect using cross-validation

Similarly as we did in section 16.2 with the Bayes factor, we revisit section 5.1, where we estimated the effect of cloze probability on the N400 average signal. We consider two models here, a model that includes the effect of cloze probability, such as `fit_N400_sihi` from section 5.1.5, and a null model.

We can verify that the likelihood that we fit was appropriate for a hierarchical model that includes an effect of cloze probability as follows:

```
formula(fit_N400_sihi)
```

```
## n400 ~ c_cloze + (c_cloze | subj) + (c_cloze | item)
```

In contrast to situation with Bayes factor, priors are less critical for cross-validation. Priors are only important in cross-validation to the extent that they affect parameter estimation: As we saw previously, very narrow priors can bias the posterior; and unrealistically wide priors can lead to convergence problems. The number of samples is also less critical than with Bayes factor, most of the uncertainty in the estimates of the \widehat{elpd} is due to the number of observations. However, a very small number of samples can affect the \widehat{elpd} because the posterior estimation will be affected by the small sample size. We update our previous formula to define a null model as follows:

```
fit_N400_sihi_null <- update(fit_N400_sihi, ~ . - c_cloze)
```

17.3.1 Cross-validation with PSIS-LOO

Estimating \widehat{elpd} using PSIS-LOO is very straightforward with `brms`, which uses the package `loo` as a back-end. There is no need to refit the model, and `loo` takes care of the applying the PSIS approximation to derive estimates and standard errors.

```
(loo_sihi <- loo(fit_N400_sihi))

##
## Computed from 4000 by 2863 log-likelihood matrix
##
##          Estimate    SE
## elpd_loo -11092.2 46.7
## p_loo      80.9  2.7
## looic     22184.5 93.4
## -----
## Monte Carlo SE of elpd_loo is 0.1.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
(loo_sih_null <- loo(fit_N400_sih_null))

##
## Computed from 4000 by 2863 log-likelihood matrix
##
##           Estimate    SE
## elpd_loo -11094.9 46.5
## p_loo      89.4   3.0
## looic     22189.8 93.1
## -----
## Monte Carlo SE of elpd_loo is 0.2.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

The function `loo` reports three quantities with their standard error:

1. `elpd_loo` is the sum of pointwise predictive accuracy (a larger, less negative number indicates better predictions).
2. `p_loo` is an estimate of effective complexity of the model; asymptotically and under certain regularity conditions, `p_loo` can be interpreted as the effective number of parameters. If `p_loo` is larger than the number of data points or parameters, this may indicate a severe model misspecification.
3. `looic` is simply $-2 \times \text{elpd}_{\text{loo}}$, the *elpd* on the deviance scale. This is called the information criterion is mainly provided for historical reasons: other information criteria like AIC (Akaike Information Criterion) and DIC (Deviance Information Criterion) are commonly used in model selection.

It's important to bear in mind that the PSIS-LOO approximation to LOO can only be trusted if Pareto k estimates (\hat{k}) are smaller than 0.7. To compare the models, we need to take a look at the difference between `elpd_loo` and the standard error of that difference:

```
loo_compare(loo_sih, loo_sih_null)

##               elpd_diff se_diff
## fit_N400_sih      0.0      0.0
## fit_N400_sih_null -2.7     2.6
```

Although the model that includes cloze probability as a predictor has higher predictive accuracy, the difference is smaller than 4 and it's smaller than two SE. This means that from the perspective LOO-CV, both models are almost indistinguishable! In fact, the same will happen if we compare the model with logarithmic predictability to the linear or null model; see exercise 17.1.

We could also check whether the alternative model is making good predictions *somewhere* by examining the difference in pointwise predictive accuracy as a function of, for example, cloze probability. In the following plot we subtract the predictive accuracy of the alternative model by the accuracy of the null model; we can interpret now larger differences as an advantage for the alternative model. However, we see that as far as *posterior predictive accuracy* goes, both models are quite similar. Figure 17.1 shows that the difference in predictive accuracy is symmetrical with respect to the zero; as we go further from the mean cloze (which is around 0.5), the differences in predictions are larger but they span over positive and negative values.

The following code stores the difference in predictive accuracy of the models in a variable and plots it in Figure 17.1.

```
df_eeg <- mutate(df_eeg,
  diff_elpd = loo_sih$pointwise[, "elpd_loo"] -
    loo_sih_null$pointwise[, "elpd_loo"]
)
ggplot(df_eeg, aes(x = cloze, y = diff_elpd)) +
  geom_point(alpha = .4, position = position_jitter(w = .001, h = 0))
```

This is unsettling because the effect of cloze probability on the N400 has been replicated in numerous studies. We would expect to see that, similar to the Bayes factor, cross-validation techniques will also show that a

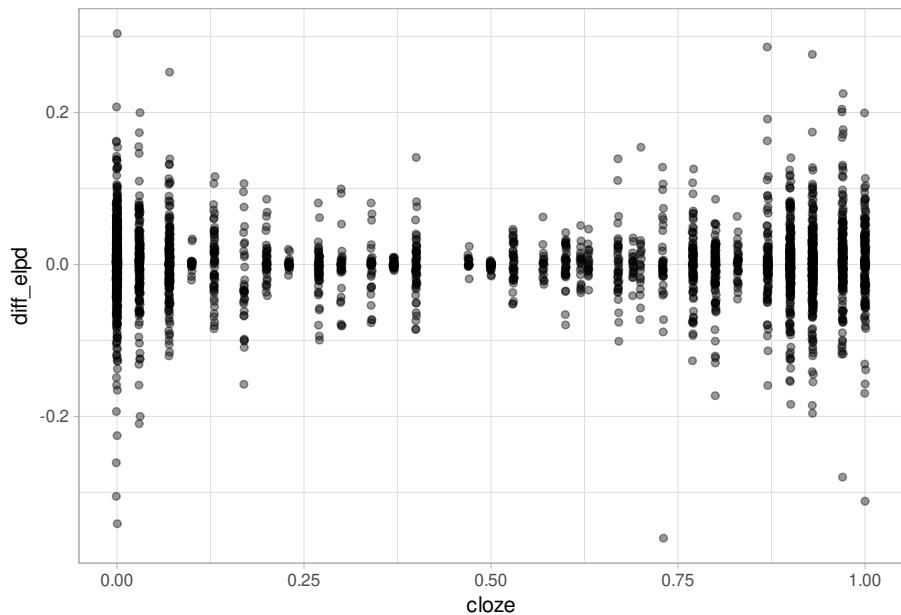


FIGURE 17.1: Difference in predictive accuracy between a model including the effect of cloze and a null model. A larger (more positive) difference indicates an advantage for the model that includes the effect of cloze.

model that includes cloze probability as a predictor is superior to a model without it. Before we discuss why we don't see a large difference, we'll try our luck with K-fold-CV.

17.3.2 Cross-validation with K-fold

Estimating $elpd$ using k-fold-CV has the advantage of omitting one layer of approximations: the $elpd$ based on PSIS-LOO-CV is an approximation of the $elpd$ based on *exact* LOO-CV (and we saw how any cross-validation approach gave as an approximation to the true $elpd$). This means that we don't need to worry about \hat{k} . However, K-fold also uses a reduced training set in comparison with LOO, worsening the approximation to the true generating process p_t .

Because we divide our data into folds, we need to think about the way we split the data: We could do it randomly, but we take the risk that in some of the training sets, observations from a subject, for example, would be

completely absent. This will lead to large differences in predictive accuracy between folds. We can avoid that by using stratification: we split the observations into groups ensuring that relative category frequencies are approximately preserved. We do this with the `kfold()` function, available in the package `brms`, by setting `folds = "stratified"` and `group = "subj"`, by default k is set to 10, but that can be changed.

```
kfold_sih <- kfold(fit_N400_sih,
                      folds = "stratified",
                      group = "subj")
```

```
kfold_sih_null <- kfold(fit_N400_sih_null,
                           folds = "stratified",
                           group = "subj")
```

Running K-fold CV takes some time since each model is refit K times. We can now inspect the *elpd* values:

```
kfold_sih
```

```
##  
## Based on 10-fold cross-validation  
##  
##           Estimate    SE  
## elpd_kfold -11097.4 46.6  
## p_kfold      85.7  3.5  
## kfoldic     22194.8 93.1
```

```
kfold_sih_null
```

```
##  
## Based on 10-fold cross-validation  
##  
##           Estimate    SE  
## elpd_kfold -11098.7 46.5  
## p_kfold      93.3  3.7
```

```
## kfoldic      22197.5 93.0
```

We compare the two models using `loo_compare` (regardless if we have used PSIS-LOO-CV or K-fold-CV):

```
loo_compare(kfold_sih, kfold_sih_null)
```

```
##                   elpd_diff se_diff
## fit_N400_sih      0.0      0.0
## fit_N400_sih_null -1.3      4.1
```

We see that, in this case, the results with K-fold-CV and PSIS-LOO-CV are quite similar: We can't really distinguish between the two models.

17.3.3 Leave-one-group-out cross-validation

An alternative to split the observations randomly using stratification is to treat naturally occurring clusters as folds, this is leave-one-group-out cross-validation (LOGO-CV). By doing this we can interpret the output of cross-validation as the capacity of the models for generalizing to unseen clusters. We implement LOGO-CV with subjects.

```
logo_sih <- kfold(fit_N400_sih,
                     group = "subj")
```

```
logo_sih_null <- kfold(fit_N400_sih_null,
                         group = "subj")
```

Running LOGO CV with subjects takes some time since each model is re-fit as many times as we have subjects, in this case 37 times! We can now inspect the `elpd` estimates and evaluate which model generalizes better to unseen subjects.

We compare the models using `loo_compare`.

```
loo_compare(logo_sih, logo_sih_null)
```

```
##                   elpd_diff se_diff
```

```
## fit_N400_sihi      0.0      0.0
## fit_N400_sihi_null -1.5     2.3
```

As before, and as with PSIS-LOO-CV, K-fold-CV, we can't distinguish between the two models.

17.4 Comparing different likelihoods with cross-validation

We now compare two models with different likelihoods. In section 3.5.3 from chapter 3, we saw how a log-normal distribution was a more appropriate likelihood than a normal distribution for response times data. This was because response times are bounded by zero and right skewed, unlike the symmetrical normal distribution. Now we'll use PSIS-LOO-CV to compare the predictive accuracy of the Stroop model from section 5.2 in chapter 5, which assumed a log-normal likelihood to fit response times of correct responses with a similar model which assumes a normal likelihood.

We load the data from `bcogsci`, create a sum coded predictor (see chapter 8 for more details), and fit the model as in section 5.2.

```
data("df_stroop")
df_stroop <- df_stroop %>%
  mutate(c_cond = if_else(condition == "Incongruent", 1, -1))
```

```
fit_stroop <- brm(RT ~ c_cond + (c_cond | subj),
  family = lognormal(),
  prior =
  c(
    prior(normal(6, 1.5), class = Intercept),
    prior(normal(0, 1), class = b),
    prior(normal(0, 1), class = sigma),
    prior(normal(0, 1), class = sd),
    prior(lkj(2), class = cor)
  ),
```

```
  data = df_stroop  
}
```

We can calculate the elpd_{loo} for the original model with the log-normal likelihood.

```
loo_stroop_log <- loo(fit_stroop)  
  
loo_stroop_log  
  
##  
## Computed from 4000 by 3058 log-likelihood matrix  
##  
## Estimate SE  
## elpd_loo -19858.7 93.6  
## p_loo 60.5 4.1  
## looic 39717.4 187.1  
## -----  
## Monte Carlo SE of elpd_loo is 0.1.  
##  
## Pareto k diagnostic values:  
##                                     Count Pct. Min. n_eff  
## (-Inf, 0.5]   (good)    3057 100.0% 1069  
## (0.5, 0.7]   (ok)      1 0.0% 203  
## (0.7, 1]     (bad)      0 0.0% <NA>  
## (1, Inf)    (very bad) 0 0.0% <NA>  
##  
## All Pareto k estimates are ok (k < 0.7).  
## See help('pareto-k-diagnostic') for details.
```

The summary shows that k estimates are ok.

Now we fit a similar model where we assume that the likelihood is a normal distribution. It's important now to change the priors since they are in a different scale (namely, in milliseconds). We choose reasonable but wide priors. We can do a sensitivity analysis, if we are unsure about the priors.

However, unlike what happened with the Bayes factor in chapter 16, priors are going to affect cross-validation based model comparison only as far as they have a noticeable effect on the posterior distribution.

```
fit_stroop_normal <- brm(RT ~ c_cond + (c_cond | subj),  
  family = gaussian(),  
  prior =  
    c(  
      prior(normal(400, 600), class = Intercept),  
      prior(normal(0, 100), class = b),  
      prior(normal(0, 300), class = sigma),  
      prior(normal(0, 300), class = sd),  
      prior(lkj(2), class = cor)  
    ),  
  data = df_stroop  
)
```

If we try to obtain elpd based on PSIS-LOO-CV, we'll find several large \hat{k} values.

```
loo_stroop_normal <- loo(fit_stroop_normal)
```

loo_stroop_normal

```

## (-Inf, 0.5] (good)    3054 99.9% 1305
## (0.5, 0.7] (ok)       1   0.0% 84
## (0.7, 1] (bad)       2   0.1% 43
## (1, Inf) (very bad)  1   0.0% 3
## See help('pareto-k-diagnostic') for details.

```

We can try the model matching approximation for problematic observations, by setting `moment_match = TRUE` in the `loo()` call (and we also need to fit the model with `save_pars = save_pars(all = TRUE)`). In this particular case, this approximation won't solve our problem. We skip that step here. We can use exact LOO (rather than its approximation) for the problematic observations. By setting `reloo = TRUE`, we re-fit the 3 problematic observations with \hat{k} values over 0.7 using exact LOO-CV.

```
loo_stroop_normal <- loo(fit_stroop_normal, reloo = TRUE)
```

```
loo_stroop_normal
```

```

##
## Computed from 4000 by 3058 log-likelihood matrix
##
##           Estimate      SE
## elpd_loo -21702.3  589.5
## p_loo     230.7   180.2
## looic    43404.7 1178.9
## -----
## Monte Carlo SE of elpd_loo is 0.3.
##
## Pareto k diagnostic values:
##                               Count Pct. Min. n_eff
## (-Inf, 0.5] (good)    3057 100.0% 1
## (0.5, 0.7] (ok)       1   0.0% 77
## (0.7, 1] (bad)       0   0.0% <NA>
## (1, Inf) (very bad)  0   0.0% <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.

```

We are ready to compare the models.

```
loo_compare(loo_stroop_log, loo_stroop_normal)

##          elpd_diff se_diff
## fit_stroop      0.0     0.0
## fit_stroop_normal -1843.6    533.8
```

Here cross-validation shows a clear advantage for the model with the log-normal likelihood. We visualize the pointwise predictive accuracy in Figure 17.2. At first glance it might seem that the advantage of the log-normal likelihood lies in being able to capture extremely slow observations.

```
df_stroop <- mutate(df_stroop,
  diff_elpd = loo_stroop_log$pointwise[, "elpd_loo"] -
    loo_stroop_normal$pointwise[, "elpd_loo"]
)
ggplot(df_stroop, aes(x = RT, y = diff_elpd)) +
  geom_point(alpha = .4)
```

We zoom in to visualize the pointwise predictive accuracy for observations with response times smaller than 2 seconds in 17.3. Here, we see that the advantage of the log-normal likelihood lies in being able to account for most of the observations in the data set, which occur around the 500 ms.

```
ggplot(df_stroop, aes(x = RT, y = diff_elpd)) +
  geom_point(alpha = .3) +
  geom_hline(yintercept = 0, linetype = "dashed") +
  coord_cartesian(xlim = c(0, 2000), ylim = c(-10, 10))
```

17.5 Issues with cross-validation

Sivula, Magnusson, and Vehtari (2020) analyzed the behavior of the uncertainty estimate of the *elpd* in typical problem settings. Although they fo-

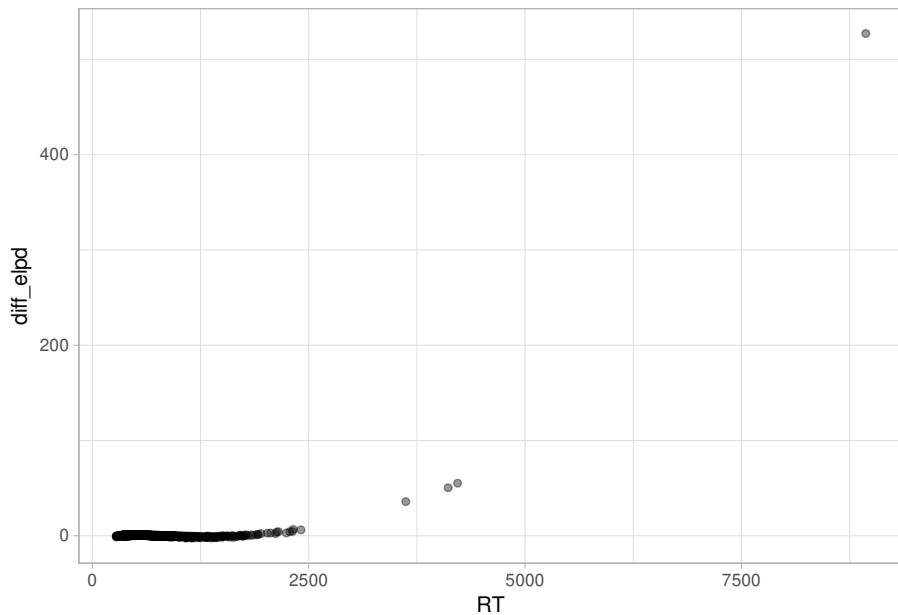


FIGURE 17.2: Difference in predictive accuracy between a Stroop model with a log-normal likelihood vs. a model with a normal likelihood. A larger (more positive) difference indicates an advantage for the model with the log-normal likelihood.

cus on LOO-CV, the consequences are the same for K-fold-CV (and cross-validation in a non-Bayesian context). Sivula, Magnusson, and Vehtari (2020) identified three cases where the uncertainty estimates can perform badly:

1. the models make very similar predictions,
2. the number of observations is small, and
3. the models are misspecified with outliers (influential extreme values) in the data.

When the models make similar predictions (as it is the case with nested models that we saw in earlier model comparisons) and there is not much difference in predictive performance of the models, the uncertainty estimates will behave badly making cross-validation less useful for separating very small effect sizes (from zero effect sizes). In addition, small differences in the predictive performance cannot reliably be detected by cross-

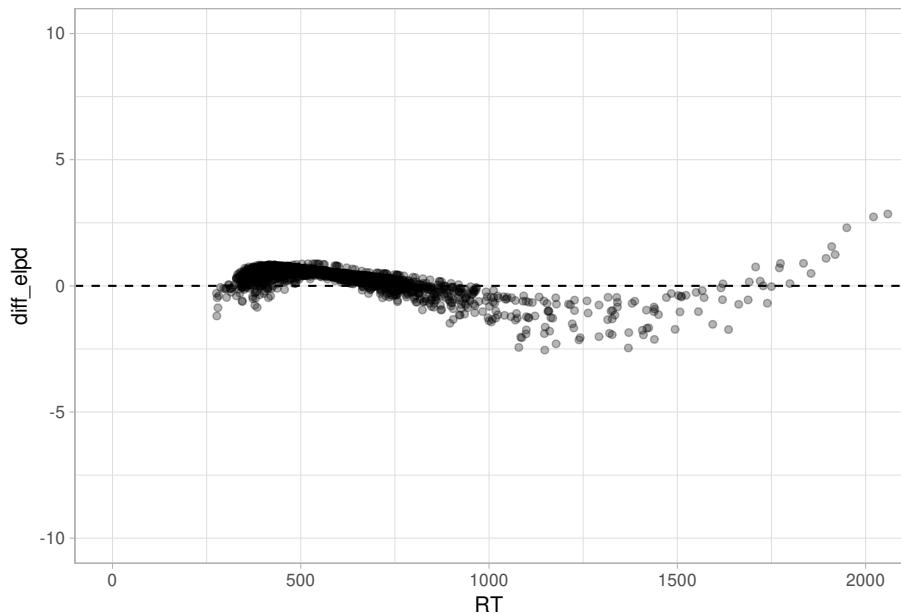


FIGURE 17.3: Difference in predictive accuracy between a Stroop model with a log-normal likelihood vs. a model with a normal likelihood for observations smaller than 2 seconds. A larger (more positive) difference indicates an advantage for the model with the log-normal likelihood.

validation if the number of observations is small (say, around 100 observations). However, if the predictions are very similar, Sivula, Magnusson, and Vehtari (2020) show that the same problems persist even with a larger data set.

One of the issues that cross-validation methods face when they are used to compare nested models lies in the way we approximate the exact $elpd$: In cross-validation approximations, we use out-of-sample observations which are not part of the model that we are fitting. Every time we evaluate the predictive accuracy of an observation, we ignore modeling assumptions. One of the weaknesses of cross-validation is the high variance in the approximation of the integral over the unknown true data distribution, p_t (Vehtari and Ojanen 2012, sec. 4).

Curiously, cross-validation methods are sometimes criticized because with too much data they will give undue preference to the complex model

in comparison to a true simpler model (Gronau and Wagenmakers 2018). This might be true for toy examples where we can have unlimited observations and we compare a “wrong” model with the true model.⁴ However, the problems that we face in practice are often very different: This is because the true model is unknown and very likely not under consideration in our comparison (also see Navarro 2019). In our experience, we are very far from the asymptotic behavior of cross-validation that guarantees undue preference to a more complex model in comparison to a true simpler model. In contrast, the main weakness of cross-validation lies in its lack of assumptions, which prevents it from selecting a more complex model rather than a simple one when there is only a modest gain in predictions.

An alternative to the cross-validation approach discussed here for nested models is the projection predictive method (Piironen, Paasiniemi, and Vehtari 2020). However, this approach (which is less general since it is valid only for generalized linear models) has a somewhat different objective. In the projection predictive method, we first, build the most complete predictive model, the *reference model*, and then we look for a simpler model that gives as similar predictions as the reference model. The idea is that for a given complexity (number of predictors), the model with the smallest predictive discrepancy to the reference model should be selected. See <https://github.com/stan-dev/projpred> for an implementation of this approach. Thus this approach focuses on model simplification rather than on model comparison.

For models that are badly misspecified, the bias in the uncertainty makes their comparison unreliable as well. In this case, posterior predictive checks and possible model refinements are worth considering before carrying out model comparison.

If there are a large number of observations and the models under consideration are different enough from each other, the differences in predictive accuracy will dwarf the variance in the estimate of *elpd* and cross-validation can be very useful (see also Piironen and Vehtari 2017), as we saw in section 17.4. When models are very different, one advantage of

⁴If the true model is under consideration among the models being compared, we are under an M_{closed} scenario. However, this is rarely realistic. The most common case is an M_{open} scenario, where the true model is not included in the set of models being compared.

cross-validation methods in comparison with the Bayes factor is that the selection of priors is less critical in cross-validation. It is sometimes hard to decide on priors that encode our knowledge for one model, and this difficulty is exacerbated when we want to assign comparable prior information to models with different number of parameters that might be in a different scale. Given that cross-validation methods are less sensitive to prior specification, different models can be compared on the same footing. See Nicenboim and Vasishth (2018) for an example from psycholinguistics where K-fold-CV does help in distinguishing between models.

17.6 Cross-validation in Stan

We can also use PSIS-LOO-CV and K-fold-CV with our Stan models, but we should be careful in storing the appropriate log-likelihood in the generated quantities block.

17.6.1 PSIS-LOO-CV in Stan

As we explained earlier, PSIS-LOO (as implemented in the package `loo`) approximates the likelihood of the held-out data based on the observed data: it's faster (because we only fit one model), and it only requires a minimal modification of the Stan code we need to fit a model. Stan by default only saves the sum of the log likelihood of each observation (in the parameter `lp__`). If we want to store the log-likelihood of each observation, we have to do this in the generated quantities block.

We revisit the model implemented before in section 10.4.2, which was evaluated using the Bayes factor in the previous chapter (chapter 16). Now, we want to compare the predictive performance of a model that assumes an effect of attentional load on pupil size against a similar model that assumes no effect. To do this, we assume the following likelihood:

$$p_{size_n} \sim Normal(\alpha + c_load_n \cdot \beta_1 + c_trial \cdot \beta_2 + c_load \cdot c_trial \cdot \beta_3, \sigma) \quad (17.16)$$

Define priors for all the β s as before.

$$\begin{aligned}\alpha &\sim \text{Normal}(1000, 500) \\ \beta_{\{1,2,3\}} &\sim \text{Normal}(0, 100) \\ \sigma &\sim \text{Normal}_+(0, 1000)\end{aligned}\tag{17.17}$$

We prepare the data as in 10.4.2:

```
df_pupil <- df_pupil %>%
  mutate(c_load = load - mean(load),
        c_trial = trial - mean(trial))
ls_pupil <- list(
  c_load = df_pupil$c_load,
  c_trial = df_pupil$c_trial,
  p_size = df_pupil$p_size,
  N = nrow(df_pupil)
)
```

We add a generated quantities block to the model as we show below. (It is also possible to run this block in a stand-alone file with the `rstan` function `gqs()`). If we use the variable name `log_lik` in the Stan code, the `loo` package will know where to find the log likelihood of the observations.

We code the effects as `beta1`, `beta2`, `beta3` to more easily compare the model with the one used in the BF chapter, but in this case we could have used a vector or an array instead. This is the model `pupil_null.stan` shown below:

```
data {
  int<lower = 1> N;
  vector[N] c_load;
  vector[N] c_trial;
  vector[N] p_size;
}
parameters {
  real alpha;
  real<lower = 0> beta1;
  real beta2;
  real beta3;
```

```

    real<lower = 0> sigma;
}
model {
    // priors including all constants
    target += normal_lpdf(alpha | 1000, 500);
    target += normal_lpdf(beta1 | 0, 100);
    target += normal_lpdf(beta2 | 0, 100);
    target += normal_lpdf(beta3 | 0, 100);
    target += normal_lpdf(sigma | 0, 1000)
        - normal_lccdf(0 | 0, 1000);
    target += normal_lpdf(p_size | alpha + c_load * beta1 + c_trial * beta2 +
                           c_load .* c_trial * beta3, sigma);
}
generated quantities{
    real log_liik[N];
    for (n in 1:N){
        log_liik[n] = normal_lpdf(p_size[n] | alpha + c_load[n] * beta1
                                   + c_trial[n] * beta2
                                   + c_load[n] * c_trial[n] * beta3,
                                   sigma);
    }
}

```

For the null model, we just omit the term with `beta1` in both the model block and the generated quantities block. This is the model `pupil_model_cv.stan` shown below:

```

data {
    int<lower = 1> N;
    vector[N] c_load;
    vector[N] c_trial;
    vector[N] p_size;
}
parameters {
    real alpha;
    real beta2;
    real beta3;

```

```

    real<lower = 0> sigma;
}
model {
    // priors including all constants
    target += normal_lpdf(alpha | 1000, 500);
    target += normal_lpdf(beta2 | 0, 100);
    target += normal_lpdf(beta3 | 0, 100);
    target += normal_lpdf(sigma | 0, 1000)
        - normal_lccdf(0 | 0, 1000);
    target += normal_lpdf(p_size | alpha + c_trial * beta2 +
                           c_load .* c_trial * beta3, sigma);
}
generated quantities{
    real log_liks[N];
    for (n in 1:N){
        log_liks[n] = normal_lpdf(p_size[n] | alpha + c_trial[n] * beta2 +
                                   c_load[n] * c_trial[n] * beta3, sigma);
    }
}

```

The models can be found in the `bcogsci` package:

```

pupil_model_cv <- system.file("stan_models",
                               "pupil_model_cv.stan",
                               package = "bcogsci")
pupil_null <- system.file("stan_models",
                           "pupil_null.stan",
                           package = "bcogsci")

```

Fit the models:

```

fit_pupil_int_pos_ll <- stan(
    file = pupil_model_cv,
    iter = 3000,
    data = ls_pupil
)

```

```
fit_pupil_int_null_ll <- stan(  
  file = pupil_null,  
  iter = 3000,  
  data = ls_pupil  
)
```

Show summary of predictive accuracy of the models:

```
(loo_pos <- loo(fit_pupil_int_pos_ll))
```

```
##  
## Computed from 6000 by 41 log-likelihood matrix  
##  
##           Estimate    SE  
## elpd_loo   -258.5  5.6  
## p_loo       3.2   1.0  
## looic      517.0 11.1  
## -----  
## Monte Carlo SE of elpd_loo is 0.0.  
##  
## All Pareto k estimates are good (k < 0.5).  
## See help('pareto-k-diagnostic') for details.
```

```
(loo_null <- loo(fit_pupil_int_null_ll))
```

```
##  
## Computed from 6000 by 41 log-likelihood matrix  
##  
##           Estimate    SE  
## elpd_loo   -258.2  5.6  
## p_loo       3.3   1.0  
## looic      516.5 11.2  
## -----  
## Monte Carlo SE of elpd_loo is 0.0.  
##  
## All Pareto k estimates are good (k < 0.5).
```

```
## See help('pareto-k-diagnostic') for details.
```

Unlike with `brms`, the functions from `loo` warn the user when $\hat{k} > 0.5$. In practice, given that PSIS-LOO-CV has good performance for values of \hat{k} up to 0.7, the message can be ignored.

```
loo_compare(loo_pos, loo_null)
```

```
##          elpd_diff se_diff
## model2    0.0      0.0
## model1   -0.3      0.4
```

As it happened with the cloze probability effect in the previous section, we cannot decide which model has better predictive accuracy according to PSIS-LOO.

17.6.1.1 K-fold-CV in Stan

If we want to use K-fold-CV (or LOGO-CV) in Stan (as opposed to PSIS-LOO), we need to be careful to store the log-likelihood of the *held-out data*, since we evaluate our model with only this subset of the data. The following example closely follows the vignette <https://cran.r-project.org/web/packages/loo/vignettes/loo2-elpd.html>.

We follow these steps:

1. Split the data in 10 folds.

Since there are no subjects we don't need to stratify (using `kfold_split_stratified`), and we use `kfold_split_random` from the `loo` package.

```
df_pupil$fold <- kfold_split_random(K = 10, N = nrow(df_pupil))
# Show number of obs for each fold:
df_pupil %>%
  group_by(fold) %>%
  count() %>%
  print(n=10)
```

```
## # A tibble: 10 × 2
## # Groups:   fold [10]
##       fold     n
##   <int> <int>
## 1     1     4
## 2     2     4
## 3     3     4
## 4     4     4
## 5     5     4
## 6     6     4
## 7     7     4
## 8     8     4
## 9     9     4
## 10    10    5
```

2. Fit and extract the log pointwise predictive densities for each fold.

We compile the alternative and the null models first with `stan_model`, and we prepare two matrices to store the predictive densities from the held out data. Each matrix has as many rows as post-warmup iterations we'll produce (3000×2), and as many columns as observations in the data set.

```
pupil_stanmodel <- stan_model(pupil_model_cv)
pupil_null_stanmodel <- stan_model(pupil_null)
log_pd_kfold <- matrix(nrow = 6000, ncol = nrow(df_pupil))
log_pd_null_kfold <- matrix(nrow = 6000, ncol = nrow(df_pupil))
```

We loop over the 10 folds. In each loop we do the following. First, we fit each model (i.e. the alternative and null model) to all the observations except the ones belonging to the held-out fold using `sampling()` with the already compiled models. Second, we compute the log pointwise predictive densities for the held-out fold with `gqs()`. This function produces generated quantities based on samples from a posterior (in the `draw` argument) and ignores all the blocks except the generated quantities one. Last, we store the predictive density for the observations of the held-out fold in a matrix by extracting the log likelihood of the held-out data. The output

of this loop is a matrix of the log pointwise predictive densities of all the observations.

```

        data = ls_pupil_ho)
# Extract log likelihood which represents
# the pointwise predictive density
log_pd_kfold[, df_pupil$fold == k] <-
  extract_log_lik(gq_ho)
log_pd_null_kfold[, df_pupil$fold == k] <-
  extract_log_lik(gq_null_ho)
}

```

3. Compute K-fold elpd

Now we evaluate the predictive performance of the model on the 10 folds using `elpd()`.

```

(elpd_pupil_kfold <- elpd(log_pd_kfold))

## 
##      Computed      from      6000      by      41      log-
likelihood matrix using the generic elpd function
##
##      Estimate    SE
## elpd   -259.7  6.0
## ic     519.5 12.0

(elpd_pupil_null_kfold <- elpd(log_pd_null_kfold))

## 
##      Computed      from      6000      by      41      log-
likelihood matrix using the generic elpd function
##
##      Estimate    SE
## elpd   -259.6  6.0
## ic     519.1 12.1

```

4. Compare the elpd estimates

```
loo_compare(elpd_pupil_kfold, elpd_pupil_null_kfold)

##          elpd_diff se_diff
## model2    0.0      0.0
## model1   -0.2      0.5
```

As with PSIS-LOO, we cannot decide which model has better predictive accuracy according to K-fold-CV.

17.7 Summary

In this chapter, we learned how to use K-fold cross-validation and leave-one-out cross-validation, using both built-in functionality in `brms` as well as Stan, in conjunction with the `loo` package. We saw an example of model comparison where cross-validation helped distinguish between the two models (Log-normal vs Normal likelihood), and another example where no important differences were found between the models being compared (the N400 data with cloze probability as predictor). In general, cross-validation will be helpful when comparing rather different models (for an example from psycholinguistics, see Nicenboim and Vasishth 2018); when the models are highly similar, it will be difficult to distinguish between them. In particular, for typical psychology and linguistics data sets, it will be difficult to get conclusive results from model comparisons using cross-validation that aim to find evidence for the presence of a fixed effect, if the effect is very small and/or the data are relatively sparse (this is often the case, especially in psycholinguistic data). In such cases, if the aim is to find evidence for a theoretical claim, other model comparison methods like Bayes factors might be more meaningful.

17.8 Further reading

A technical discussion about cross-validation methods can be found in Chapter 7 of Gelman et al. (2014). For a discussion about the advan-

tages and disadvantages of (leave-one-out) cross-validation, see Gronau and Wagenmakers (2018), Vehtari, Simpson, et al. (2019) and Gronau and Wagenmakers (2019). Cross-validation is still an active area of research, there are multiple websites and blog posts about it: Aki Vehtari, the creator of the `loo` package has a comprehensive FAQ about cross-validation in <https://avehtari.github.io/modelselection/CV-FAQ.html>; he also discusses in Andrew Gelman's blog when cross-validation can be applied in <https://statmodeling.stat.columbia.edu/2018/08/03/loo-cross-validation-approaches-valid/>.

17.9 Exercises

Exercise 17.1. Predictive accuracy of the linear and the logarithm effect of cloze probability.

- a. Is there a difference in predictive accuracy between the model that incorporates a linear effect of cloze probability and one that incorporates log-transformed cloze probabilities?

Exercise 17.2. Lognormal model in Stan

Use PSIS-LOO to compare a model of Stroop as the one in 11.1 with a model that assumes no population-level effect

- (a) in brms.
- (b) in Stan.

Part VI

Computational cognitive modeling with Stan



18

Introduction to computational cognitive modeling

This part introduces an approach to implementing cognitive models using Stan. In many cases, a great deal of cognitive detail is sacrificed for tractability. The broader lesson to learn here is that it is possible to specify an underlying generative process for the data that reflects theoretical assumptions from a particular research area.

18.1 Further reading

Good textbooks on computational modeling for cognitive science are Lee and Wagenmakers (2014), Busemeyer and Diederich (2010), Farrell and Lewandowsky (2018). Also see the article by Lee (2011), and the special issue (<https://tinyurl.com/zd63arz4>) on hierarchical Bayesian modeling in the Journal of Mathematical Psychology. Wilson and Collins (2019) discusses good practices in the computational modeling of behavioral data using examples from reinforcement learning. Haines et al. (2020) discusses how generative models produce higher test-retest reliability and more theoretically informative parameter estimates than do traditional methods.



19

Multinomial processing trees

In this chapter, we introduce a widely-used cognitive model that can be implemented in Stan, the multinomial processing tree. This model is useful in situations where the behavioral response from the subject is one of several possible categorical outcomes. As an example, we will look into a word production task, where we ask subjects with aphasia (a language impairment mostly due to a cerebrovascular accident or head trauma, Damasio 1992), to name the object shown in a picture, e.g., a picture of a cat. A subject could produce the correct name (“cat”), a semantically and phonologically related but incorrect name (“rat”), a semantically unrelated but phonologically related word (“hat”), or a non-word (“cag”). The researcher may have a theory about how each possible outcome ends up being probabilistically produced. Such a theoretical process model can be expressed as a multinomial processing tree. Before we dive into multinomial processing trees, we discuss the distributions that generalize the binomial and Bernoulli distribution for modeling more than two possible outcomes.

19.1 Modeling multiple categorical responses

One way to model categorical responses is using multinomial or categorical distributions. The categorical responses could be “yes” or “no”; “blue”, “red” or “yellow”; “true”, “false”, or “I don’t know”; or more complicated categories, but crucially the response of each observation can be coded as only belonging to only one category. The multinomial and the categorical distribution represent two ways of characterizing the underlying generative process for such data.

The *multinomial distribution* is the generalization of the binomial distribu-

tion for more than two possible outcomes. Recall that the binomial works like this: in order to randomly generate the number of successes in observation consisting of 10 trials, with the probability of success 0.5, one can type:

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

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

It is possible to repeatedly generate multiple observations as follows. Suppose five simulated observations are needed, each with 10 trials:

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

```
## [1] 6 5 7 7 2
```

Now, suppose that there are N=3 possible answers to a question (yes, no don't know), and suppose that the probabilities of producing each answer are:

- $P(\text{yes}) = 0.1$
- $P(\text{no}) = 0.1$
- $P(\text{don't know}) = 0.8$

The probabilities must sum to 1, because those are the only three possible outcomes. Given such a situation, it is possible to simulate a single experiment with 10 trials, where each of the three possibilities appears a certain number of times. We do this with `rmnom` from the `extraDistr` package (one could have used `rmultinom()` in R equally well, but the output would look a bit different):

```
(random_sample <- rmnom(1, size = 10, prob = c(0.1, 0.1, 0.8)))
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    7
```

The above call returns the result of the random sample: 1 cases of the first answer type, 2 cases of the second; and 7 cases of the third.

Analogously to the binomial function shown above, five observations can be simulated, each having 10 trials:

```
rmnom(5, size = 10, prob = c(0.1, 0.1, 0.8))
```

```
##      [,1] [,2] [,3]
## [1,]     1     1     8
## [2,]     3     1     6
## [3,]     1     1     8
## [4,]     0     2     8
## [5,]     0     0    10
```

The *categorical distribution* is the generalization of the Bernoulli distribution for more than two possible outcomes, and it is the special case of the multinomial distribution when we have only one trial. Recall that the Bernoulli distribution can be used as follows. If we carry out a coin toss (each coin toss counts as a single trial), we will either get a heads or a tails:

```
rbern(5, prob = 0.5)
```

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

```
## equivalent rbinom command:
rbinom(5, size = 1, prob = 0.5)
```

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

Thus, what the Bernoulli is to the Binomial, the Categorical is to the Multinomial. For example, one can simulate five observations, each of which will give one of the three responses with the given probabilities. We do this with `rcat` from the `extraDistr` package.

```
rcat(5, prob = c(0.1, 0.1, 0.8), labels = c("yes", "no", "dontknow"))
```

```
## [1] dontknow dontknow yes      yes      dontknow
## Levels: yes no dontknow
```

The above is analogous to using the multinomial with `size = 1` (a single

trial in each experiment). In the output below, the `rmnom` function shows which of the three categories is produced.

```
rmnom(5, size = 1, prob = c(0.1, 0.1, 0.8))
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    1
## [2,]    0    0    1
## [3,]    0    0    1
## [4,]    0    0    1
## [5,]    0    0    1
```

With these distributions as background, consider now a simulated situation where multiple responses are possible.

19.1.1 A model for multiple responses using the multinomial likelihood

Impaired picture naming (anomia) is common in most cases of aphasia. It is assessed as part of most comprehensive aphasia test batteries, since picture naming accuracy is a relatively easily obtained and is a reliable test score; in addition, the types of errors that are committed can provide useful information for diagnosis.

In this simulated experiment, the responses are categorized as shown in Table 19.1.

TABLE 19.1: Categorization of responses for the simulated experiment.

Category	Description	Example
Correct	The response matches the target.	cat
Neologism	The response is not a word, but it has a phonological relation to the target.	cag
Formal	The response is a word with only a phonological relation to the target.	hat
Mixed	The response is a word with both a semantic and phonological relation the target.	rat
NR	All other responses, including omissions, descriptions, non-nouns, etc.	-

First, generate data assuming a multinomial distribution. The outcomes will be determined by a vector θ (called `true_theta` below in the R code) that indicates the probability of each outcome:

```
(true_theta <- tibble(theta_NR = .2,
                     theta_Neologism = .1,
                     theta_Formal = .2,
                     theta_Mixed = .08,
                     theta_Correct = 1 -
                     (theta_NR + theta_Neologism + theta_Formal + theta_Mixed)))

## # A tibble: 1 × 5
##   theta_NR theta_Neologism theta_Formal theta_Mixed theta_Correct
##       <dbl>          <dbl>        <dbl>        <dbl>        <dbl>
## 1      0.2            0.1         0.2        0.08        0.42

## The probabilities must sum to 1:
sum(true_theta)

## [1] 1
```

Given this vector of probabilities θ , generate values assuming a multinomial distribution of responses in 100 trials:

```
N_trials <- 100
(ans_mn <- rmultinom(1, N_trials, true_theta))

## [,1]
## theta_NR      19
## theta_Neologism    7
## theta_Formal     17
## theta_Mixed      7
## theta_Correct     50
```

Now, we'll try to recover the probability of each answer with a model with the following likelihood:

$$ans \sim \text{Multinomial}(\theta) \quad (19.1)$$

where $\theta = \{\theta_{nr}, \theta_{neol.}, \theta_{formal}, \theta_{mix}, \theta_{corr}\}$.

A common prior for multinomial likelihood is the Dirichlet distribution, which extends the Beta distribution to cases where more than two categories are available.

$$\theta \sim \text{Dirichlet}(\alpha) \quad (19.2)$$

The Dirichlet distribution has a parameter α , called concentration parameter, and it is a vector with the same length as θ . If we set $\alpha = \{2, 2, 2, 2, 2\}$, this is analogous to $\sim \text{Beta}(2, 2)$, that is, the intuition behind this concentration parameter is that the prior probability distribution of the vector θ corresponds to have seen 2 outcomes of each category in the past.

A Stan model assuming a multinomial likelihood and Dirichlet prior is shown below. Since the elements of θ should sum to one we declare this vector, `theta`, as a simplex. A simplex guarantees that its elements sum to one and also constraints them to have non-negative values. In order to generate the vector α that contains five times the value two, we use `rep_vector(2, 5)` (which is similar to `rep(2, 5)` in R).

```

data {
    int<lower = 1> N_trials;
    int<lower = 0,upper = N_trials> ans[5];
}
parameters {
    simplex[5] theta;
}
model {
    target += dirichlet_lpdf(theta | rep_vector(2, 5));
    target += multinomial_lpmf(ans | theta);
}
generated quantities{
    int pred_ans[5] = multinomial_rng(theta, 5);
}

```

Fit the model:

```
# Create a list:
# c(ans_mn) makes a vector out of the matrix ans_mn
data_mn <- list(N_trials = N_trials,
                 ans = c(ans_mn))

str(data_mn)

## List of 2
## $ N_trials: num 100
## $ ans      : int [1:5] 19 7 17 7 50

multinom <- system.file("stan_models",
                        "multinom.stan",
                        package = "bcogsci")
fit_mn <- stan(file = multinom,
                data = data_mn)
```

Print the posteriors:

```
print(fit_mn, pars = c("theta"))

##           mean 2.5% 97.5% n_eff Rhat
## theta[1] 0.19 0.13 0.26 4416    1
## theta[2] 0.08 0.04 0.14 4737    1
## theta[3] 0.17 0.11 0.25 4810    1
## theta[4] 0.08 0.04 0.14 4583    1
## theta[5] 0.47 0.38 0.57 4359    1
```

Next, we use `mcmc_recover_hist` in the code below to confirm that the posterior distributions of the elements of θ are close to the true values that were set up when simulating the data. See Figure 19.1.

```
as.data.frame(fit_mn) %>%
  select(starts_with("theta")) %>%
```

```
mcmc_recover_hist(true = unlist(true_theta)) +
coord_cartesian(xlim = c(0, 1))
```

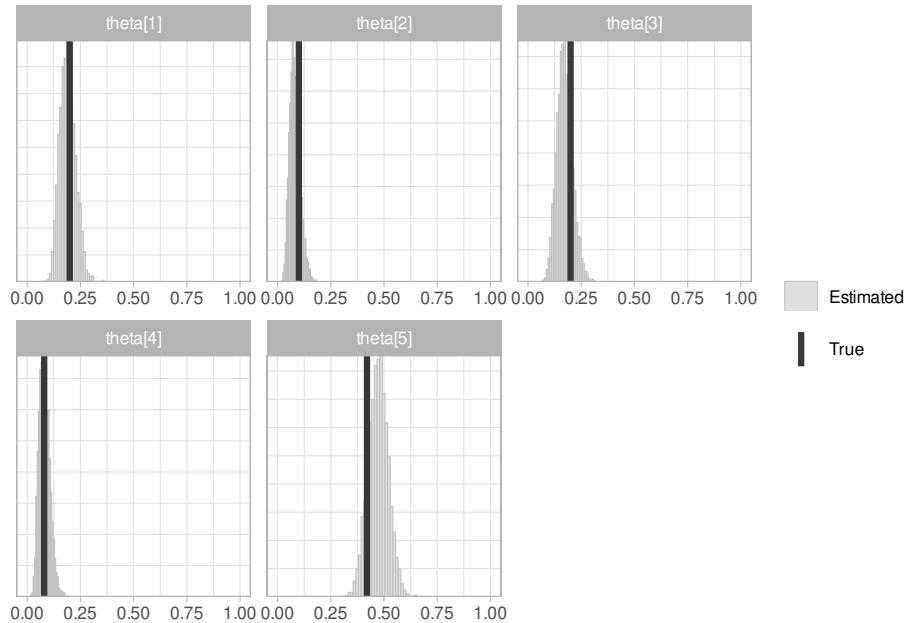


FIGURE 19.1: Posterior distributions and true means of theta for the multinomial model defined in `multinom.stan`.

We evaluate here whether our model is able to “recover” the true value of its parameters. By “recover”, we mean that the true values are somewhere inside the posterior distribution of the model.

The frequentist properties of Bayesian models guarantee that if we simulate data several times, 95% of the true values should be inside of the 95% CrI intervals generated by a “well-calibrated” model. Furthermore, if the true values of some parameters are consistently well above or below their posterior distribution, it may mean that there is some problem with the model specification. We follow Cook, Gelman, and Rubin (2006) here, and for now we are going to verify that our model is roughly correct. A more principled (and computationally demanding) approach uses simulation based calibration (SBC), which is discussed in Talts et al. (2018) and Schad, Betancourt, and Vasishth (2020).

19.1.2 A model for multiple responses using the categorical distribution

Using the same information as above, we can model each response one at a time, instead of aggregating them. Using the categorical distributions gives us more flexibility to define what happens at every trial. However, we are not using the additional flexibility for now, and hence the next model and the previous one are equivalent.

```
data {
    int<lower = 1> N_obs;
    int<lower = 1, upper = 5> w_ans[N_obs];
}
parameters {
    simplex[5] theta;
}
model {
    target += dirichlet_lpdf(theta | rep_vector(2, 5));
    for(n in 1:N_obs)
        target += categorical_lpmf(w_ans[n] | theta);
}
generated quantities{
    int pred_w_ans[N_obs];
    for(n in 1:N_obs)
        pred_w_ans[n] = categorical_rng(theta);
}
```

Given the same set of probabilities θ as above, generate 100 individual observations:

```
N_obs <- 100
ans_cat <- rcat(N_obs, prob = as.matrix(true_theta))
```

The above output is how Stan expects to see the data. The data fed into the Stan model is defined as a list as usual:

```
data_cat <- list(N_obs = N_obs,
                  w_ans = ans_cat)
str(data_cat)
```

```
## List of 2
## $ N_obs: num 100
## $ w_ans: num [1:100] 2 5 3 5 5 5 3 5 5 5 ...
```

Fitting the Stan model (`categorical.stan`) should yield approximately the same θ as with the multinomial likelihood defined in the model `multinom.stan`. See Figure ??.

```
categorical <- system.file("stan_models",
                           "categorical.stan",
                           package = "bcogsci")
fit_cat <- stan(file = categorical,
                 data = data_cat)
```

```
print(fit_cat, pars = c("theta"))
```

```
##           mean 2.5% 97.5% n_eff Rhat
## theta[1] 0.17 0.11 0.25 4410    1
## theta[2] 0.11 0.06 0.17 4528    1
## theta[3] 0.26 0.19 0.35 4913    1
## theta[4] 0.06 0.03 0.11 4662    1
## theta[5] 0.39 0.30 0.48 4315    1
```

The above models estimate the posterior distribution for the probability for each possible response. If we had some experimental manipulation, we could even fit regressions to these parameters. This is called a multinomial logistic regression or categorical regression, see further reading for some examples.

19.2 Modeling picture naming abilities in aphasia with MPT models

Multinomial processing tree (MPT) modeling is a method that estimates latent variables that have a psychological interpretation given categorical data (a review is provided in Batchelder and Riefer 1999). In other words, an MPT model is just one way to model categorical responses following a

multinomial or categorical distribution. MPT models assume that the observed response categories result from a sequences of underlying cognitive events which are represented as a binary branching tree. Each binary branching is associated with a parameter that represents the probability of going down either branch. Every successive node is assumed to be independent of the preceding node, allowing us to use the product rule from probability theory to compute the probability of going down a particular path. The leaves of the binary branching tree the observed response in the data. The goal is to derive posterior distributions of the latent probability parameters specified for the binary branching in the model.

Walker, Hickok, and Fridriksson (2018) created an MPT model that specifies a set of possible internal errors that lead to the various possible response types during a picture naming trial for aphasic patients. Here we'll explore a simplification of the original model.

The model assumes that when an attempt is made to produce a word, errors in production can arise at the whole word level (lexical level) or the segmental level (phonological level). Semantic errors are assumed to arise from the lexical substitutions, and neologism errors from phonological substitutions. Real word responses that are phonologically related to the correct target word can arise from substitutions at the lexical or phonological level.

The task for the subject is to view a picture and name the object represented in the picture. When an attempt is made to retrieve the word from memory, the following possible steps can unfold (this is a simplified version of the original model):

- Either the subject will make some lexical selection, or fail to make a lexical selection, returning a non-response (NR). The probability of making some lexical selection is a , so the probability of a non-response is $1 - a$, as these are only two possibilities at this initial stage of the binary branching tree. Example: the subject sees the picture of a cat, and either produces the response "I don't know", or starts the process of producing a word.
- If a lexical selection is made, the target word is selected with probability t , or some other word is chosen with probability $1 - t$.
- Once a word is selected, either its phonological representation is

selected with probability f , or some other (incorrect) phonological representation is selected with probability $1 - f$.

- Once a word is selected, there can be a phonological change that leads to a real, formally related word with probability c , or a neologism with probability $1 - c$. Example: the subjects produces either a formally related word “hat”, or a neologism like “cag”.

The end-result of walking down this tree is that the subject either produces a non-response (“I don’t know” or silence), a correct response, a related word, a mixed word, or a neologism. There is more than one way to produce a neologism or a related word, and the posterior probabilities of the various paths will determine the probability of each possible path.

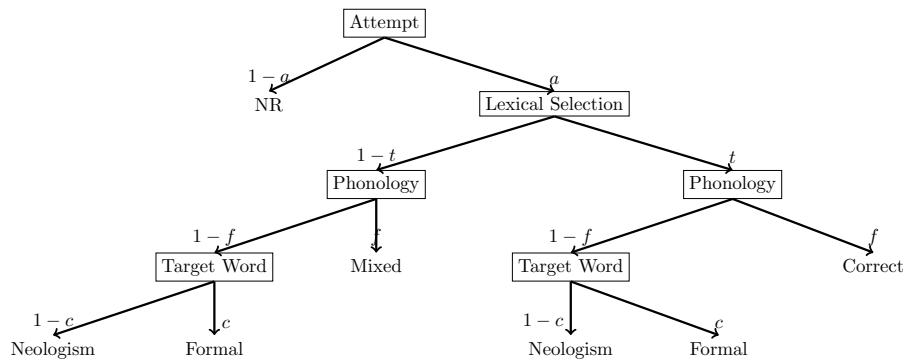


FIGURE 19.2: Representation of a simplification of the MPT used in Walker, Hickok, and Fridriksson (2018).

TABLE 19.2: Psychological interpretation of the parameters of the MPT model.

Param.	Description
a	Probability of initiating an attempt
t	Probability of selecting a target word over competitors
f	Probability of retrieving correct phonemes
c	Probability of a phoneme change in the target word creating a real word

19.2.1 Calculation of the probabilities in the MPT branches

By navigating through the branches of the MPT (Figure 19.2), we can calculate the probabilities of the four responses (the categorical outcomes), based on the underlying parameters assumed in the MPT:

- $P(NR|a, t, f, c) = 1 - a$
- $P(Neologism|a, t, f, c) = a \cdot (1 - t) \cdot (1 - f) \cdot (1 - c) + a \cdot t \cdot (1 - f) \cdot (1 - c)$
- $P(Formal|a, t, f, c) = a \cdot (1 - t) \cdot (1 - f) \cdot c + a \cdot t \cdot (1 - f) \cdot c$
- $P(Mixed|a, t, f, c) = a \cdot (1 - t) \cdot f$
- $P(Correct|a, t, f, c) = a \cdot t \cdot f$

Given that

$$\begin{aligned} P(NR|a, t, f, c) + P(Neologism|a, t, f, c) + P(Formal|a, t, f, c) + \\ P(Mixed|a, t, f, c) + P(Correct|a, t, f, c) = 1 \end{aligned} \quad (19.3)$$

there is no need to characterize every outcome: we can always calculate any one of the remaining responses as one minus the other responses.

19.2.2 A simple MPT model

First, simulate 200 trials assuming no variability between items and subjects. It is convenient to define functions to compute each outcome's probability, based on the previous MPT.

```

# Probabilities of different answers
Pr_NR <- function(a, t, f, c)
  1 - a
Pr_Neologism <- function(a, t, f, c)
  a * (1 - t) * (1 - f) * (1 - c) + a * t * (1 - f) * (1 - c)
Pr_Formal <- function(a, t, f, c)
  a * (1 - t) * (1 - f) * c + a * t * (1 - f) * c
Pr_Mixed <- function(a, t, f, c)
  a * (1 - t) * f
Pr_Correct <- function(a, t, f, c)
  a * t * f
# true underlying values for simulated data
a_true <- .75
t_true <- .9
f_true <- .8
c_true <- .1
# Probability of the different answers:
Theta <- tibble(NR = Pr_NR(a_true, t_true, f_true, c_true),
                 Neologism = Pr_Neologism(a_true, t_true, f_true, c_true),
                 Formal = Pr_Formal(a_true, t_true, f_true, c_true),
                 Mixed = Pr_Mixed(a_true, t_true, f_true, c_true),
                 Correct = Pr_Correct(a_true, t_true, f_true, c_true))
N_trials <- 200
(ans <- rmultinom(1, N_trials, c(Theta)))

```

```

##          [,1]
## NR         51
## Neologism  24
## Formal     3
## Mixed      9
## Correct    113

```

The above data can be modeled in Stan as discussed below (see `mpt_mnm.stan`). The probabilities of the different categories go into the transformed parameters section because they are derived from the probability parameters in the model. The data are modeled as coming from a multinomial likelihood. If priors are not specified, then a Beta

distribution with $a = 1$ and $b = 1$ (a Uniform(0,1) distribution) is assumed for the parameters a , t , f , and c . Unlike θ , the values of these parameters are independent of each other and they do not sum to one. For this reason, we should not use a Dirichlet prior here.

We define the following model:

$$\begin{aligned}
 \theta_{nr} &= 1 - a \\
 \theta_{neol.} &= a \cdot (1 - t) \cdot (1 - f) \cdot (1 - c) + a \cdot t \cdot (1 - f) \cdot (1 - c) \\
 \theta_{formal} &= a \cdot (1 - t) \cdot (1 - f) \cdot c + a \cdot t \cdot (1 - f) \cdot c \\
 \theta_{mix} &= a \cdot (1 - t) \cdot f \\
 \theta_{corr} &= a \cdot t \cdot f \\
 \theta &= \{\theta_{nr}, \theta_{neol.}, \theta_{formal}, \theta_{mix}, \theta_{corr}\} \\
 ans &\sim Multinomial(\theta) \\
 a, t, f, c &\sim Beta(2, 2)
 \end{aligned} \tag{19.4}$$

This translates to the following code:

```

data {
    int<lower = 1> N_trials;
    int<lower = 0, upper = N_trials> ans[5];
}
parameters {
    real<lower = 0, upper = 1> a;
    real<lower = 0, upper = 1> t;
    real<lower = 0, upper = 1> f;
    real<lower = 0, upper = 1> c;
}
transformed parameters {
    simplex[5] theta;
    theta[1] = 1 - a; //Pr_NR
    theta[2] = a * (1 - t) * (1 - f) * (1 - c) + a * t * (1 - f) * (1 -
c); //Pr_Neologism
    theta[3] = a * (1 - t) * (1 - f) * c + a * t * (1 - f) * c; //Pr_Formal
    theta[4] = a * (1 - t) * f; //Pr_Mixed
    theta[5] = a * t * f; //Pr_Correct
}

```

```

model {
    target += beta_lpdf(a | 2, 2);
    target += beta_lpdf(t | 2, 2);
    target += beta_lpdf(f | 2, 2);
    target += beta_lpdf(c | 2, 2);
    target += multinomial_lpmf(ans | theta);
}
generated quantities{
    int pred_ans[5];
    pred_ans = multinomial_rng(theta, 5);
}

```

Fit the model:

```

data_sMPT <- list(N_trials = N_trials,
                   ans = c(ans))
mpt_mnm <- system.file("stan_models",
                        "mpt_mnm.stan",
                        package = "bcogsci")
fit_sMPT <- stan(file = mpt_mnm, data = data_sMPT)

```

Print out a summary of the posterior of the parameter of interest:

```
print(fit_sMPT, pars = c("a", "t", "f", "c"))
```

```

##   mean 2.5% 97.5% n_eff Rhat
## a 0.74 0.68 0.80 4598    1
## t 0.91 0.86 0.96 4872    1
## f 0.81 0.74 0.87 4377    1
## c 0.16 0.06 0.31 4429    1

```

What the model gives us is posterior distributions of each of the parameters a , t , f , c . From these we can derive the probabilities of producing the different observed responses, and the posterior predictive distributions, which could be used for model evaluation.

An important sanity check in modeling is checking whether the model can

in principle recover the true parameters that generated the data; see Figure 19.3.

```
as.data.frame(fit_sMPT) %>%
  select(c("a","t","f","c")) %>%
  mcmc_recover_hist(true = c(a_true, t_true, f_true, c_true)) +
  coord_cartesian(xlim = c(0, 1))
```

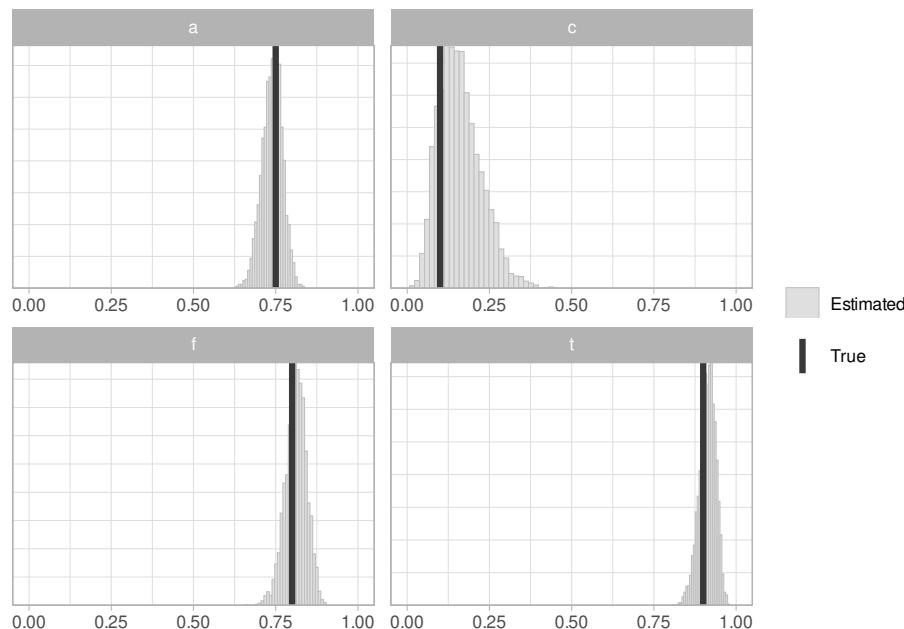


FIGURE 19.3: Posterior distributions and true values of the parameters of the simple MPT model (`mpt_mnm.stan`).

The above figure shows that the model can indeed recover the true parameters fairly accurately.

The posterior distributions of the θ parameters can also be summarized:

```
print(fit_sMPT, pars = c("theta"))

##           mean 2.5% 97.5% n_eff Rhat
## theta[1] 0.26 0.20  0.32   4598    1
```

```
## theta[2] 0.12 0.08 0.17 4524    1
## theta[3] 0.02 0.01 0.05 4381    1
## theta[4] 0.05 0.03 0.09 4791    1
## theta[5] 0.55 0.48 0.62 4549    1
```

These posteriors tell us the probability of producing each of the possible responses. This model might be useful to estimate the latent parameters, a, t, f, c , but without further constraints it is unfalsifiable.

Recall that for the multinomial likelihood in section 19.1.1, we had a simplex of size five, which means that we had four free parameters (since the fifth can be deduced based on the others). With five possible answers we can always estimate a vector of probabilities, θ , that fits the data, in the same way that with two possible answers (e.g., zeros and ones), we can always estimate a probability θ that fits the data (using a Bernoulli or Binomial likelihood). The MPT that we present here is just re-parametrizing the vector θ of the multinomial likelihood (with the same number of free parameters). This means that it will always achieve a perfect fit. Also see exercise 19.2. This doesn't mean that this MPT model is "useless": Under the assumption that the model is meaningful, one can estimate its latent parameters and this estimation can have theoretical implications. If we want to be able to falsify this model, we'll need to constrain it more, as we suggest below.

19.2.3 An MPT assuming by-item variability

The use of aggregated data implies the assumption that the estimated parameters do not vary too much between subjects and items. If this assumption is incorrect, the analysis of aggregated data may lead to erroneous conclusions: reliance on aggregated data in the presence of parameter heterogeneity may lead to biased parameter estimates and the underestimation of credible intervals.

If it is known that f is affected by the phonological complexity of the individual word (e.g., *cat* is easier to produce than *umbrella*), the previous model does not have a way to include that information.

Simulated data can be generated taking into account the complexity of the items. Assume here for simplicity that the complexity of items is scaled and centered; i.e., mean complexity is represented by o , and the standard

deviation is assumed to be 2. We will assume a regression model that determines the parameter, f , as a function of the phonological complexity of each trial.

One important detail is that f is a probability and needs to be bounded between 0 and 1. To make sure that this property is met, the computation of f for each item will be converted to probability space using the logistic function. This is achieved as follows.

Suppose that f' is a linear function of complexity. For example, two parameters α_f and β_f (intercept and slope respectively) could determine how f' is affected by complexity:

$$f'_j = \alpha_f + \text{complexity}_j \cdot \beta_f.$$

The parameters α_f and β_f are defined in an unconstrained log-odds space (they can be any real number). The model that is fit then yields an f'_j value for each item j in log-odds space. The log-odds value f'_j can be converted to a probability value f_{true} by applying the logistic function (or the inverse logit, $logit^{-1}$) to f' . Recall from the generalized linear model discussed earlier that if we have a model in log-odds space:

$$\log\left(\frac{p_j}{1-p_j}\right) = \alpha + \beta \cdot x_j = \mu_j \quad (19.5)$$

Then we can recover the probability p_j by solving for p_j :

$$p_j = \frac{\exp(\mu_j)}{1 + \exp(\mu_j)} = \frac{1}{1 + \exp(-\mu_j)} \quad (19.6)$$

The above is the logistic or inverse logit function: it takes as input μ_j and returns the corresponding probability p_j . The `plogis` function in R carries out the calculation shown above.

```
N_obs <- 50
complexity <- rnorm(N_obs, mean = 0, sd = 2)
## choose some hypothetical values:
alpha_f <- .3
# the negative sign indicates that
```

```
# increased complexity will lead to a reduced value of f
beta_f <- -.3
# f' as a linear function of complexity
f_prime <- alpha_f + complexity * beta_f
head(f_prime)
```

```
## [1] 0.5781 0.1167 0.3504 0.0538 0.1898 -0.7672
```

```
## probabilities f for each item
f_true <- plogis(f_prime)
head(f_true)
```

```
## [1] 0.641 0.529 0.587 0.513 0.547 0.317
```

This change in our assumptions entails that the probability of each response changes depending on the item associated with each observation. The parameters theta now have to be a matrix. This is in R; in Stan, we will code it as an array of simplexes, i.e., an array of non-negative values that sums to 1.

We continue with the functions defined in 19.2.2, and the same values for a_true, t_true, and c_true as defined in that section. Since most of the equations depend on f, and f is a vector now, the outcomes are automatically vectors. But this is not the case for theta_NR_v, and thus we need to repeat the value.

```
theta_NR_v <- rep(Pr_NR(a_true, t_true, f_true, c_true), N_obs)
theta_Neologism_v <- Pr_Neologism(a_true, t_true, f_true, c_true)
theta_Formal_v <- Pr_Formal(a_true, t_true, f_true, c_true)
theta_Mixed_v <- Pr_Mixed(a_true, t_true, f_true, c_true)
theta_Correct_v <- Pr_Correct(a_true, t_true, f_true, c_true)
theta_item <- matrix(
  c(theta_NR_v,
    theta_Neologism_v,
    theta_Formal_v,
    theta_Mixed_v,
```

```

    theta_Correct_v),
  ncol = 5)
dim(theta_item)

## [1] 50 5

head(theta_item, n = 3)

##      [,1]  [,2]  [,3]  [,4]  [,5]
## [1,] 0.25 0.243 0.0270 0.0480 0.432
## [2,] 0.25 0.318 0.0353 0.0397 0.357
## [3,] 0.25 0.279 0.0310 0.0440 0.396

```

Store this in a data frame.

```

sim_data_cx <- tibble(item = 1:N_obs,
                      complexity = complexity,
                      w_ans = c(rcat(N_obs,theta_item)))

sim_data_cx

## # A tibble: 50 x 3
##   item complexity w_ans
##   <int>     <dbl> <dbl>
## 1     1     -0.927     1
## 2     2      0.611     1
## 3     3     -0.168     5
## # ... with 47 more rows

```

The following model (saved in `mpt_cat.stan`) is essentially doing the same as the previous model but instead of fitting a multinomial to the summary of all the trials, it is fitting a categorical distribution to each individual observation. (This is analogous to the difference between the Bernoulli and Binomial distributions).

This is still not the appropriate model for the generative process that we are assuming in this section, because it still ignores the effect of complexity, but it is a good start.

```

data {
    int<lower=1> N_obs;
    int<lower=1,upper=5> w_ans[N_obs];
}
parameters {
    real<lower=0,upper=1> a;
    real<lower=0,upper=1> t;
    real<lower=0,upper=1> f;
    real<lower=0,upper=1> c;
}
transformed parameters {
    simplex[5] theta[N_obs];

    for(n in 1:N_obs){
        //Pr_NR:
        theta[n, 1] = 1 - a;
        //Pr_Neologism:
        theta[n, 2] = a * (1 - t) * (1 - f) * (1 - c) + a * t * (1 -
f) * (1 - c);
        //Pr_Formal:
        theta[n, 3] = a * (1 - t) * (1 - f) * c + a * t * (1 - f) * c;
        //Pr_Mixed:
        theta[n, 4] = a * (1 - t) * f;
        //Pr_Correct:
        theta[n, 5] = a * t * f;
    }
}
model {
    target += beta_lpdf(a | 2, 2);
    target += beta_lpdf(t | 2, 2);
    target += beta_lpdf(f | 2, 2);
    target += beta_lpdf(c | 2, 2);
    for(n in 1:N_obs)
        target += categorical_lpmf(w_ans[n] | theta[n]);
}
generated quantities{
    int pred_w_ans[N_obs];
}

```

```
for(n in 1:N_obs)
  pred_w_ans[n] = categorical_rng(theta[n]);
}
```

An important aspect of the previous model is that `theta` is declared as `simplex[5]` `theta[N_obs]`. This means that `theta` is an array of simplexes and thus has now two dimensions: each element of the array (of length `N_obs`) is a simplex and sums to one. That's why we iterate over the `N_obs`. However, one limitation of the previous model is that the latent parameters `a`, `t`, `f`, `c` are declared as `real` and they do not vary in each iteration of the loop. Before moving to the next section, you might want to do exercise 19.3, where you are asked to edit the previous chunk of code to incorporate the fact that `f` is now a transformed parameter that depends on the trial information and two new parameters.

19.2.4 A hierarchical MPT

The previous model doesn't take into account that subjects might vary (and neither does the modification to this model that is suggested in exercise 19.3). Let's focus on taking into account the differences between subjects.

Different subjects might not be equally motivated to do the task. This can be accounted by adding a hierarchical structure to the parameter `a`, the probability of initiating an attempt. Begin by simulating some data that incorporates by-subject variability.

First, define the number of items and subjects, and the number of observations:

```
# Data:
N_item <- 20
N_subj <- 30
N_obs <- N_item * N_subj
```

Then, generate a vector for subjects and for items. Assume here that each subject sees each item.

```
subj <- rep(1:N_subj, each = N_item)
item <- rep(1:N_item, time = N_subj)
```

A vector representing complexity is created for the number of items we have, and this vector is repeated as many times as there are subjects:

```
complexity <- rep(rnorm(N_item, 0, 2), times = N_subj)
```

Next, create a data frame with all the above information:

```
(exp_sim <- tibble(subj = subj,
                     item = item,
                     complexity = complexity))
```

```
## # A tibble: 600 x 3
##   subj   item complexity
##   <int> <int>     <dbl>
## 1     1     1     -2.36
## 2     1     2     -1.51
## 3     1     3     -0.663
## # ... with 597 more rows
```

To create subject-level variability in the data, a between-subject standard deviation needs to be defined. This standard deviation represents the deviations of subjects about the grand mean. We are defining this adjustment in log-odds space.

```
# New parameters, in log-odds space:
tau_u_a <- 1.1
## generate subject adjustments in log-odds space:
u_a <- rnorm(N_subj, 0, tau_u_a)
str(u_a)
```

```
## num [1:30] 0.78 0.81 1.502 -0.634 -0.885 ...
```

Given the fixed `a_true` probability value of 0.75, the subject-level values for individual `a_true` can be derived by (a) first converting the overall `a_true`

value to log-odds space, (b) adding the by-subject adjustment to this converted overall value, and (c) then converting back to probability space using the logistic or inverse logit (`plogis`) function. Essentially we generate data assuming the following:

$$\begin{aligned} a'_{h,n} &= \alpha_a + u_{a,subj[n]} \\ a_{h,n} &= \text{logit}^{-1}(a'_{h,n}) \end{aligned} \quad (19.7)$$

Where $u_{a,subj[n]}$ is a vector with the same length as the total number of observations. The meaning of this notation was explained in the section 11.1 of chapter 11.

This is done in R as follows:

```
a_true <- .75 # as before
## convert the intercept to log-odds space:
alpha_a <- qlogis(a_true)
## a_h' in log-odds space:
a_h_prime <- alpha_a + u_a[subj]
## convert back to probability space
a_true_h <- plogis(a_h_prime)
str(a_true_h)

## num [1:600] 0.867 0.867 0.867 0.867 0.867 ...
```

What this achieves mathematically is adding varying intercepts by subjects to `alpha_a`, and then the values adjusted by subject are saved in probability space.

As before, `f_true` is computed as a function of complexity:

```
alpha_f <- .3; beta_f <- -.3
f_true <- plogis(alpha_f + complexity * beta_f)
```

We continue with the same probability functions and the rest of the true values remain the same as well.

```
t_true <- .9; c_true <- .1
Pr_NR <- function(a, t, f, c)
  1 - a
Pr_Neologism <- function(a, t, f, c)
  a * (1 - t) * (1 - f) * (1 - c) + a * t * (1 - f) * (1 - c)
Pr_Formal <- function(a, t, f, c)
  a * (1 - t) * (1 - f) * c + a * t * (1 - f) * c
Pr_Mixed <- function(a, t, f, c)
  a * (1 - t) * f
Pr_Correct <- function(a, t, f, c)
  a * t * f
```

Now, we can define the probabilities of different outcomes:

```
# Aux. parameters that define the probabilities:
theta_NR_v_h <- Pr_NR(a_true_h, t_true, f_true, c_true)
theta_Neologism_v_h <- Pr_Neologism(a_true_h, t_true, f_true, c_true)
theta_Formal_v_h <- Pr_Formal(a_true_h, t_true, f_true, c_true)
theta_Mixed_v_h <- Pr_Mixed(a_true_h, t_true, f_true, c_true)
theta_Correct_v_h <- Pr_Correct(a_true_h, t_true, f_true, c_true)
theta_h <- matrix(
  c(theta_NR_v_h,
    theta_Neologism_v_h,
    theta_Formal_v_h,
    theta_Mixed_v_h,
    theta_Correct_v_h),
  ncol = 5)
dim(theta_h)
```

```
## [1] 600   5
```

The probability specifications shown above can now generate the simulated data:

```
# simulated data:
```

```
(sim_data_h <- mutate(exp_sim,
                      w_ans = rcat(N_obs, theta_h)))
```

```
## # A tibble: 600 x 4
##   subj item complexity w_ans
##   <int> <int>      <dbl> <dbl>
## 1     1     1      -2.36     5
## 2     1     2      -1.51     5
## 3     1     3      -0.663    2
## # ... with 597 more rows
```

We define now the following model; we omit the steps with f' and a' and we directly apply the logistic function to a regression. The parameters t , c do not vary by item or subject and therefore do not have the subscript n . We start by defining relatively weak priors for all the parameters in the following model.

$$\begin{aligned}
 \alpha_a, \alpha_f, \beta_f &\sim Normal(0, 2) \\
 t, c &\sim Beta(2, 2) \\
 \tau_u &\sim Normal(0, 1) \\
 u_a &\sim Normal(0, \tau_{u_a}) \\
 a_n &= logit^{-1}(\alpha_a + u_{a,subj[n]}) \\
 f_n &= logit^{-1}(\alpha_f + complexity_n \cdot \beta_f) \\
 \theta_{n,nr} &= 1 - a_n \\
 \theta_{n,neol.} &= a_n \cdot (1 - t) \cdot (1 - f_n) \cdot (1 - c) + a_n \cdot t \cdot (1 - f_n) \cdot (1 - c) \\
 \theta_{n,formal} &= a_n \cdot (1 - t) \cdot (1 - f_n) \cdot c + a_n \cdot t \cdot (1 - f_n) \cdot c \\
 \theta_{n,mix} &= a_n \cdot (1 - t) \cdot f_n \\
 \theta_{n,corr} &= a_n \cdot t \cdot f_n \\
 \theta_n &= \{\theta_{n,nr}, \theta_{n,neol.}, \theta_{n,formal}, \theta_{n,mix}, \theta_{n,corr}\} \\
 ans_n &\sim Categorical(\theta_n)
 \end{aligned} \tag{19.8}$$

The corresponding Stan model `mpt_h.stan` will look like this:

```
data {
```

```
int<lower = 1> N_obs;
int<lower = 1, upper = 5> w_ans[N_obs];
real complexity[N_obs];
int<lower = 1> N_subj;
int<lower = 1, upper = N_subj> subj[N_obs];
}

parameters {
    real<lower = 0, upper = 1> t;
    real<lower = 0, upper = 1> c;
    real alpha_a;
    real<lower = 0> tau_u_a;
    vector[N_subj] u_a;
    real alpha_f;
    real beta_f;
}
transformed parameters {
    simplex[5] theta[N_obs];
    for (n in 1:N_obs){
        real a = inv_logit(alpha_a + u_a[subj[n]]);
        real f = inv_logit(alpha_f + complexity[n] * beta_f);
        //Pr_NR
        theta[n, 1] = 1 - a;
        //Pr_Neologism
        theta[n, 2] = a * (1 - t) * (1 - f) * (1 - c) +
            a * t * (1 - f) * (1 - c);
        //Pr_Formal
        theta[n, 3] = a * (1 - t) * (1 - f) * c
            + a * t * (1 - f) * c;
        //Pr_Mixed
        theta[n, 4] = a * (1 - t) * f;
        //Pr_Correct
        theta[n, 5] = a * t * f;
    }
}
model {
    target += beta_lpdf(t | 2, 2);
    target += beta_lpdf(c | 2, 2);
```

```

target += normal_lpdf(alpha_a | 0, 2);
target += normal_lpdf(alpha_f | 0, 2);
target += normal_lpdf(beta_f | 0, 2);
target += normal_lpdf(u_a | 0, tau_u_a);
target += normal_lpdf(tau_u_a | 0, 1) - normal_lccdf(0 | 0, 1);
for(n in 1:N_obs)
    target += categorical_lpmf(w_ans[n] | theta[n]);
}
generated quantities{
    int<lower = 1, upper = 5> pred_w_ans[N_obs];
    for(n in 1:N_obs)
        pred_w_ans[n] = categorical_rng(theta[n]);
}

```

For ease of exposition, we are not using the non-centered parametrization discussed previously in 11.1.2. We could also apply it here and it will speed up and improve the convergence of the model; see Exercise 19.4.

It would be a good idea to plot prior predictive distributions for this model; we skip this step here. Next, fit the model to the simulated data, by first defining the data as a list:

```

sim_list_h <- list(N_obs = nrow(sim_data_h),
                     w_ans = sim_data_h$w_ans,
                     N_subj = max(sim_data_h$subj),
                     subj = sim_data_h$subj,
                     complexity = sim_data_h$complexity)

```

```

mpt_h <- system.file("stan_models",
                      "mpt_h.stan",
                      package = "bcogsci")
fit_mpt_h <- stan(file = mpt_h,
                   data = sim_list_h,
                   control = list(adapt_delta = .9))

```

Print out a summary of the posterior.

```
print(fit_mpt_h,
      pars = c("t", "c", "tau_u_a", "alpha_a", "alpha_f", "beta_f")
```

	mean	2.5%	97.5%	n_eff	Rhat
## t	0.91	0.87	0.95	8868	1
## c	0.11	0.08	0.16	7953	1
## tau_u_a	0.67	0.40	0.99	1793	1
## alpha_a	0.98	0.67	1.31	2913	1
## alpha_f	0.15	-0.05	0.35	7120	1
## beta_f	-0.29	-0.42	-0.18	7926	1

If we had fit this to real data, we would now conclude that (i) given the value of `beta_f`, complexity has an adverse effect on the probability of retrieving the correct phonemes, and (ii) given the value of `tau_u_a`, there is a great deal of variation in the subjects' probability of initiating an attempt at each trial. Furthermore, if we had some expectation about *t* and *c* based on previous research we could conclude that our results are in line (or not) with previous findings.

One could inspect how one unit of complexity is affecting the probability of retrieving the correct phoneme (*f*). We first derive the value of *f* for an item of zero complexity (that is $\alpha_f + 0 \times \beta_f$) and then the value of *f* for an item with a complexity of one ($\alpha_f + 1 \times \beta_f$). We are interested in summarizing the difference between the two:

```
as.data.frame(fit_mpt_h) %>%
  select(alpha_f, beta_f) %>%
  mutate(f_0 = plogis(alpha_f),
         f_1 = plogis(alpha_f + beta_f),
         diff_f = f_1 - f_0) %>%
  summarize(Estimate = mean(diff_f),
            `2.5%` = quantile(diff_f, 0.025),
            `97.5%` = quantile(diff_f, 0.975))
```

## Estimate	2.5%	97.5%
## 1 -0.0731	-0.103	-0.0441

One further interesting step could be to develop a competing model that

assumes a different latent process, and then comparing the performance of the MPT with this competing model, using Bayes factors or K-fold-CV (or both).

Since we generated the data based on known latent parameters, we also plot the posteriors together with the true values of the parameters in Figure 19.4. This is something that we can only do with simulated data.

```
as.data.frame(fit_mpt_h) %>%
  select(tau_u_a, alpha_a, t, alpha_f, beta_f, c) %>%
  mcmc_recover_hist(true = c(tau_u_a,
    qlogis(a_true),
    t_true, alpha_f,
    beta_f, c_true))
```

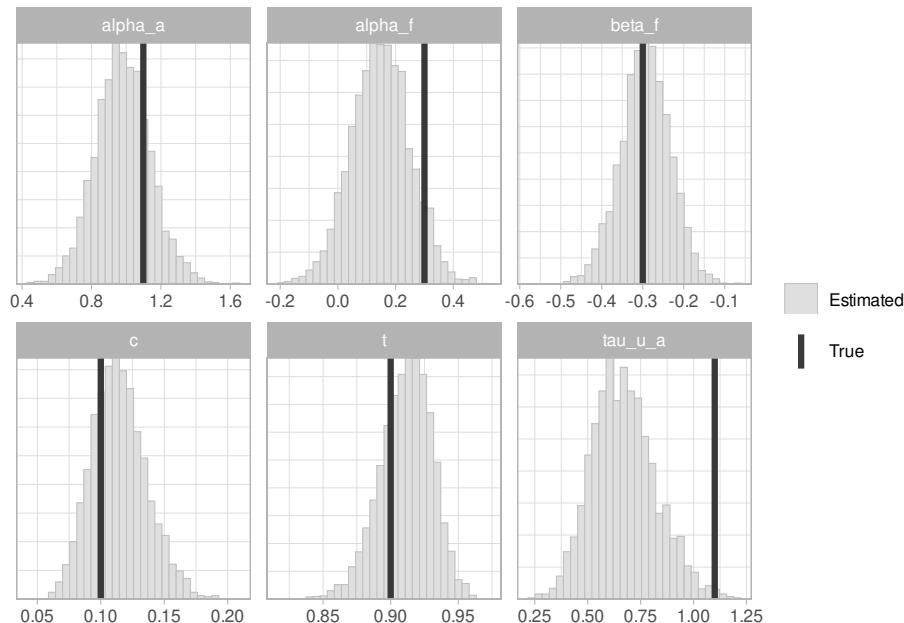


FIGURE 19.4: Posterior of the hierarchical MPT with true values as vertical lines (model `mpt_h.stan`).

If everything is correctly defined in the model, we should be able to generate posterior predictive data based on our estimates that looks quite similar to the simulated data; see Figure 19.5.

```
gen_data <- rstan:::extract(fit_mpt_h)$pred_w_ans
ppc_bars(sim_list_h$w_ans, gen_data) +
  ggtitle ("Hierarchical model")
```

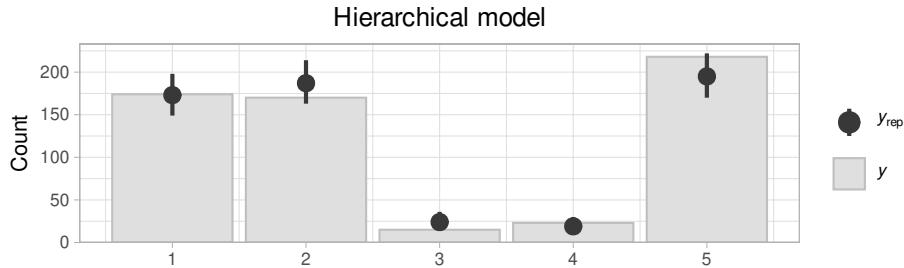


FIGURE 19.5: Posterior predictive check for aggregated data in the hierarchical MPT model

It is also useful to look at the individual subjects' posteriors; these are shown in Figure 19.6.

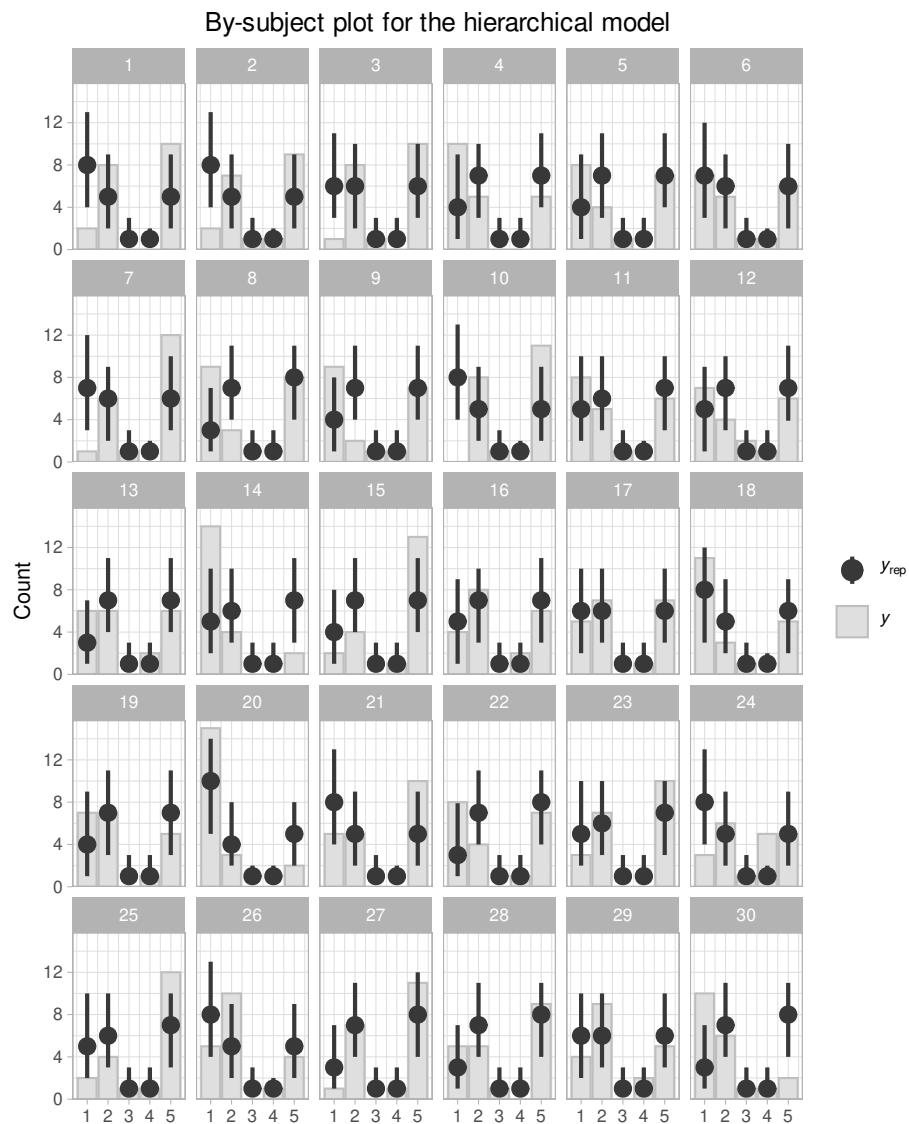
```
ppc_bars_grouped(sim_list_h$w_ans,
                  gen_data, group = subj) +
  ggtitle ("By-subject plot for the hierarchical model")
```

But what about the first *non-hierarchical* MPT model (`mpt_cat.stan`)?:

```
mpt_cat <- system.file("stan_models",
                       "mpt_cat.stan",
                       package = "bcogsci")
fit_sh <- stan(file = mpt_cat, data = sim_list_h)
```

The aggregated data looks great (Figure 19.7).

```
gen_data_sMPT <- rstan:::extract(fit_sh)$pred_w_ans
ppc_bars(sim_list_h$w_ans, gen_data_sMPT) +
  ggtitle ("Non-hierarchical model")
```

**FIGURE 19.6:** Individual subjects in the hierarchical MPT model.

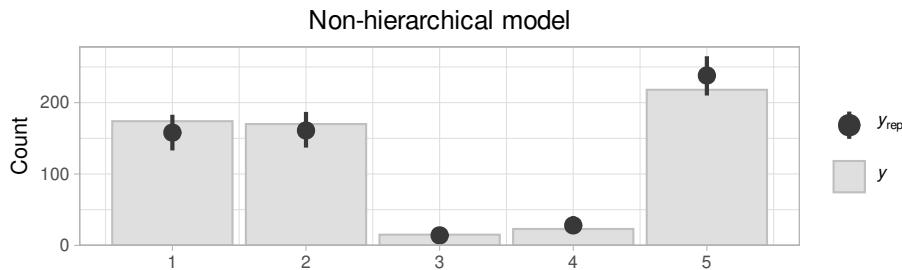


FIGURE 19.7: Posterior predictive check for aggregated data in a non-hierarchical MPT model (mpt_cat.stan).

However, the fit to individual subjects looks less good (Figure 19.8) for the non-hierarchical model.

```
ppc_bars_grouped(sim_list_h$w_ans, gen_data_sMPT, group = subj) +
  ggtitle ("By-subject plot for the non-hierarchical model")
```

The hierarchical model does a better job of modeling individual-level variability.

19.3 Further reading

Koster and McElreath (2017) present a tutorial on Multinomial logistic regression/Categorical regression in the context of behavioral ecology and anthropology. Another tutorial on MPTs is presented by Matzke et al. (2015). For the complete implementation of an MPT relating to aphasia, see Walker, Hickok, and Fridriksson (2018). Some examples of cognitive models using MPTs are Lee et al. (2020) and Smith and Batchelder (2010).

19.4 Exercises

Exercise 19.1. Modeling multiple categorical responses.

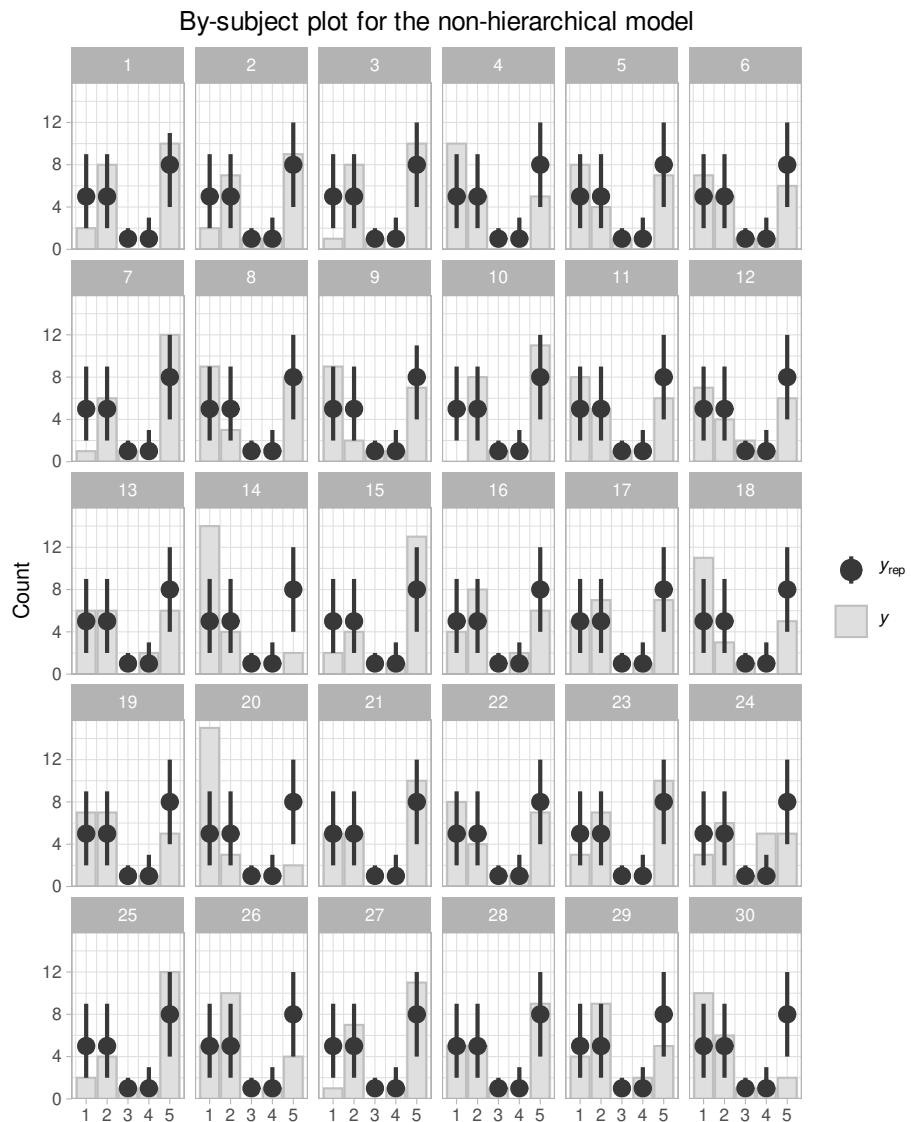


FIGURE 19.8: Individual subjects in the non-hierarchical MPT model (mpt_cat.stan).

- a. Re-fit the model presented in section 19.1.2, adding the assumption that you have more information about the probability of giving a correct response in the task. Assume that you know that subjects' answers have around 60% accuracy. Encode this information in the priors with two different degrees of certainty. (Hint: 1. As with the Beta distribution, you can increase the pseudo-counts to increase the amount of information and reduce the "width" of the distribution; compare $Beta(9, 1)$ with $Beta(900, 100)$. 2. You'll need to use a column vector for the Dirichlet concentration parameters. $[..., ..., ...]$ is a `row_vector` that can be transposed and converted into a column vector by adding the transposition symbol '`'` after the right bracket.)
- b. What is the difference between the multinomial and categorical parametrizations?
- c. What can we learn about impaired picture naming from the models in 19.1.1 and 19.1.2?

Exercise 19.2. An alternative MPT to model the picture recognition task.

Build *any* alternative tree with four parameters w, x, y, z to fit the data generated in 19.2.2. Compare the posterior distribution of the auxiliary vector θ (that goes in the `multinomial_lpmf`) with the one derived in section ??.

Exercise 19.3. A simple MPT model that incorporates phonological complexity in the picture recognition task.

Edit the Stan code `mpt_cat.stan` from `bcogsci` presented in section 19.2.3 to incorporate the fact that f is now a transformed parameter that depends on the trial information and two new parameters, α_f and β_f . The rest of the latent parameters do not need to vary by trial.

$$\begin{aligned} f'_j &= \alpha_f + \text{complexity}_j \cdot \beta_f \\ f_j &= \text{logit}^{-1}(f'_j) \end{aligned} \tag{19.9}$$

The inverse logit or logistic function is called `inv_logit` in Stan. Fit the model to the data of 19.2.3 and report the posterior distributions of the latent parameters.

Exercise 19.4. A more hierarchical MPT.

Modify the hierarchical MPT presented in section 19.2.4 so that all the parameters are affected by individual differences. Simulate data and fit it. How well can you recover the parameters? You should use the non-centered parameterization for the by-subject adjustments. (Hint: Convergence will be reached much faster if you don't assume that the adjustment parameters are correlated as in 11.1.2, but you could also assume a correlation between all (or some of) the adjustments by using the Cholesky factorization discussed in 11.1.3.)

Exercise 19.5. Advanced: Multinomial processing trees.

The data set `df_source_monitoring` in `bcogsci` contains data from the package `psychotools` coming from a source-monitoring experiment (Batchelder and Riefer 1990) performed by Wickelmaier and Zeileis (2018).

In this type of experiment, subjects study items from (at least) two different sources, A and B. After the presentation of the study items, subjects are required to classify each item as coming from source A, B, or as new: N (that is, a distractor). In Wickelmaier & Zeileis' version of the experiment, subjects had to read items either quietly (think) or aloud (say). In the recall task, they wrote them down (write) or read them aloud (say).

- `experiment`: write-say or think-say
- `age`: Age of the respondent in years.
- `gender`: Gender of the respondent.
- `subj`: Subject id.
- `source`: Item source, a, b or b (new)
- `a, b, N`: Number of responses for each type of stimuli

Fit a multinomial processing tree following Figure ?? to investigate whether experiment type, age and/or gender affects the different processes assumed in the model. As in Batchelder and Riefer (1990), assume that $a = g$ (for identifiability) and that discriminability is equal for both sources ($d_1 = d_2$).

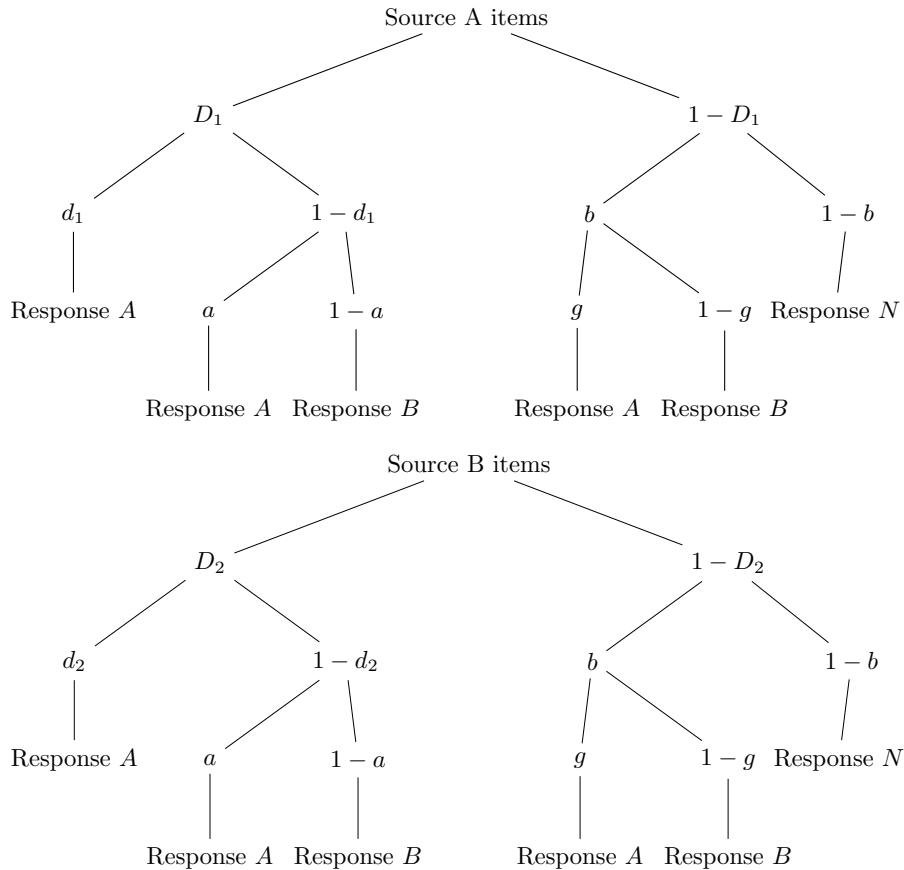


FIGURE 19.9: Multinomial processing tree for the source B items from source monitoring paradigm (Batchelder and Riefer, 1990). D_2 stand for the detectability of source B items, d_2 stands for the source discriminabilities for source B, b stands for the bias for responding “old” to a nondetected item, a stands for guessing that a detected but nondiscriminated item belongs to Source A, and g stands for guessing that the item is a source A item.

Notice the following:

- The data are aggregated at the level of source, so you should use `multinomial_lpmf` for every row of the data set rather than `categorical_lpmf()`.
- In contrast to the previous example, `source` determines three different trees, this means that the parameter `theta` has to be defined in relationship to the item source.

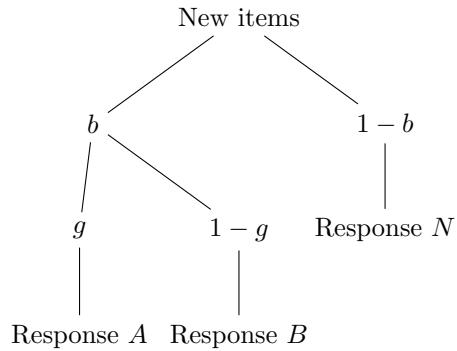


FIGURE 19.10: Multinomial processing tree for the new items in the source monitoring paradigm (Batchelder and Riefer, 1990). b stands for the bias for responding “old” to a nondetected item, a stands for guessing that a detected but nondiscriminated item belongs to source A, and g stands for guessing that the item is a source A item.

- All the predictors are between subject, this means that only a by-intercept adjustment (for every latent process) is possible.

If you want some basis to start with, you can have a look at the incomplete code in `source.stan`, by typing the following in R:

```

cat(readLines(system.file("stan_models",
                         "source.stan",
                         package = "bcogsci")),
    sep = "\n")
  
```



20

Mixture models

Mixture models integrate multiple data generating processes into a single model. This is especially useful in cases where we cannot fully identify from the data alone which observations belong to which process. Mixture models are important in cognitive science because many theories of cognition assume that the behavior of subjects in certain tasks is determined by an interplay of different cognitive processes (e.g., response times in schizophrenia in Levy et al. 1993; retrieval from memory in sentence processing in McElree 2000; Nicenboim and Vasishth 2018; fast choices in Ollman 1966; Dutilh et al. 2011). It is important to stress that a mixture distribution of observations is an *assumption* of the latent process developing trial by trial based on a given theory—it doesn't necessarily represent the true generative process. The role of Bayesian modeling is to help us understand the extent to which this assumption is well-founded, by using posterior predictive checks and comparing different models.

We focus here on the case where we have only two components; each component represents a distinct cognitive process based on the domain knowledge of the researcher. The vector z serves as an *indicator variable* that indicates which of the mixture components an observation y_n belongs to ($n = 1, \dots, N$ is the number of data points). We assume two components, and thus each z_n can be either 0 or 1 (this will allow us to generate z_n with a *Bernoulli* distribution). We also assume two different generative processes, p_1 and p_2 , which generate different distributions of the observations based on a vector of parameters indicated by Θ_1 and Θ_2 , respectively. These two processes occur with probability θ and $1 - \theta$, and each observation is generated as follows:

$$z_n \sim \text{Bernoulli}(\theta)$$

$$y_n \sim \begin{cases} p_1(\Theta_1), & \text{if } z_n = 1 \\ p_2(\Theta_2), & \text{if } z_n = 0 \end{cases} \quad (20.1)$$

We focus on only two components because this type of models is already barely identifiable in most applied situations. However, if the number of components in the mixture is finite and also determined by the researcher, the approach presented here can in principle be extended to any number of mixtures by replacing the Bernoulli distribution by a categorical one.

In order to fit this model, we need to estimate the posterior of each of the parameters contained in the vectors Θ_1 and Θ_2 (intercepts, slopes, group-level effects, etc.), the probability θ , and the indicator variable that corresponds to each observation z_n . One issue that presents here is that z_n must be a discrete parameter, and Stan only allows continuous parameters. This is because Stan's algorithm requires the derivatives of the (log) posterior distribution with respect to all parameters, and discrete parameters are not differentiable (since they have "breaks"). In probabilistic programming languages like WinBUGS and JAGS (Lunn et al. 2012; Plummer 2016), discrete parameters are possible to use; but not in Stan. In Stan, we can circumvent this issue by marginalizing out the indicator variable z .¹ If p_1 appears in the mixture with probability θ , and p_2 with probability $1 - \theta$, then the joint likelihood is defined as a function of Θ (which concatenates both Θ_1 and Θ_2), and importantly z_n "disappears":

$$p(y_n|\Theta) = \theta \cdot p_1(y_n|\Theta_1) + (1 - \theta) \cdot p_2(y_n|\Theta_2) \quad (20.2)$$

The intuition behind this formula is that each likelihood function, p_1 , p_2 is weighted by its probability of being the relevant generative process. For our purposes, it suffices to say that marginalization works; the reader interested in the mathematics behind marginalization is directed to the further reading section at the end of the chapter.²

¹See section 1.6.1.1 in chapter 1 for a review on the concept of marginalization.

²As mentioned above, other probabilistic languages that do not rely on Hamiltonian dynamics (exclusively) are able to deal with this. However, even when sampling discrete parameters is possible, marginalization is more efficient (Yackulic et al. 2020): when z_n is omitted we are fitting a model with n parameters fewer.

Even though Stan cannot fit a model with the discrete indicator of the latent class z that we used in equation (20.1), this equation will prove very useful when we want to generate synthetic data.

In the following sections, we use a well-known phenomenon (i.e., the speed-accuracy trade-off) as an example of a cognitive model that assumes an underlying finite mixture process. We start from the verbal description of the model, and we then implement the model in Stan step by step.

20.1 A mixture model of the speed-accuracy trade-off: The fast-guess model account

When we are faced with multiple choices that require an immediate decision, we can speed up the decision at the expense of accuracy and become more accurate at the expense of speed; this is called the speed-accuracy trade-off (Wickelgren 1977). The most popular class of models that can incorporate both response times and accuracy, and give an account for the speed-accuracy trade-off is the class of sequential sampling models, which include the drift diffusion model (Ratcliff 1978), the linear ballistic accumulator (Brown and Heathcote 2008), and the log-normal race model (Heathcote and Love 2012; Rouder et al. 2015), which we discuss in chapter 21; for a review see Ratcliff et al. (2016).

However, an alternative model that has been proposed in the past is Ollman's simple fast-guess model (Ollman 1966; Yellott 1967, 1971).³ Although it has mostly fallen out of favor (but see Dutilh et al. 2011 for a more modern variant of this model), it presents a very simple framework using finite mixture modeling that can also account for the speed-accuracy trade-off. In the next sections, we'll use this model to exemplify the use of finite mixtures to represent different cognitive processes.

³Ollman original model was meant to be relevant for only means, and Yellott (1967, 1971) generalized it to a distributional form.

20.1.1 The global motion detection task

One way to examine the behavior of human and primate subjects when faced with two-alternative forced choices is the detection of the global motion of a random dot kinematogram (Britten et al. 1993). In this task, a subject sees a number of random dots on the screen from which a proportion of them move in a single direction (e.g., up) and the rest move in random directions. The subject's goal is to estimate the overall direction of the movement. One of the reasons for the popularity of this task is that it permits the fine-tuning of the difficulty of trials (Dutilh et al. 2019): The task is harder when the proportion of dots that move coherently (the level of *coherence*) is lower; see Figure 20.1.

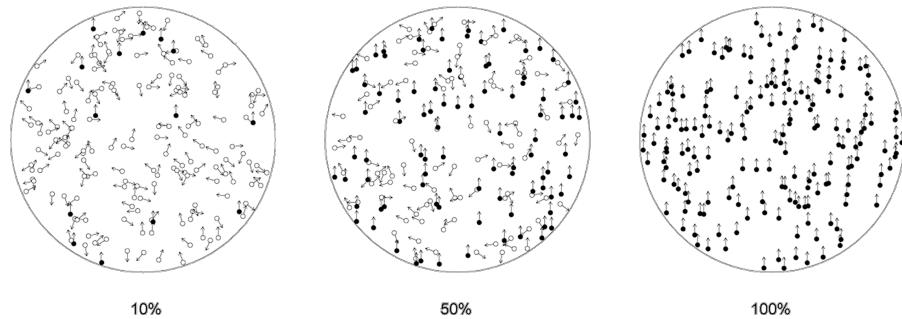


FIGURE 20.1: Three levels of difficulty of the global motion detection task. The figures show a consistent upward movement with three levels of coherence (10%, 50%, and 100%). The subjects see the dots moving in the direction indicated by the arrows. The subjects do not see the arrows and all the dots look identical in the actual task. Adapted from Han et al. (2018); licensed under CC BY 4.0.

Ollman's (1966) fast-guess model assumes that the behavior in this task (and in any other choice task) is governed by two distinct cognitive processes: (i) a guessing mode, and (ii) a task-engaged mode. In the guessing mode, responses are fast and accuracy is at chance level. In the task-engaged mode, responses are slower and accuracy approaches 100%. This means that intermediate values of response times and accuracy can only be achieved by mixing responses from the two modes. Further assumptions of this model are that response times depend on the difficulty of the choice, and that the probability of being on one of the two states depend on the speed incentives during the instructions. (To simplify matters, we

ignore the possibility of the accuracy of the choice being also affected by the difficulty of the choice. Also, we ignore the possibility that subjects might be biased to one specific response in the guessing mode, but see exercise 20.3.)

20.1.1.1 Data set

We implement the assumptions behind Ollman's fast-guess model and examine its fit to data of a global motion detection task from Dutilh et al. (2019).

The data set from Dutilh et al. (2019) contains ~2800 trials of each of the 20 subjects participating in a global motion detection task and can be found as `df_dots` in the `bcogsci` package. There were two levels of coherence yielding hard and easy trials (`diff`), and the trials were done under instructions that emphasized either accuracy or speed (`emphasis`). More information about the data set can be found by accessing the documentation for the data set (by typing `?df_dots` in the R console).

```
data("df_dots")
df_dots

## # A tibble: 56,097 x 12
##   subj diff emphasis    rt    acc fix_dur stim resp trial block
##   <int> <chr> <chr>     <dbl> <int>    <dbl> <chr> <chr> <int> <int>
## 1     1 easy  speed      482     1    0.738  R     R     1     6
## 2     1 hard  speed      602     1    0.784  R     R     2     6
## 3     1 hard  speed      381     1    0.651  R     R     3     6
##   block_trial bias
##           <int> <chr>
## 1             1 no
## 2             2 no
## 3             3 no
## # ... with 56,094 more rows
```

We might think that if the fast-guess model were true, we would see a bimodal distribution, when we plot a histogram of the data. Unfortunately, when two similar distributions are mixed, we won't see any apparent bimodality; see Figure 20.2.

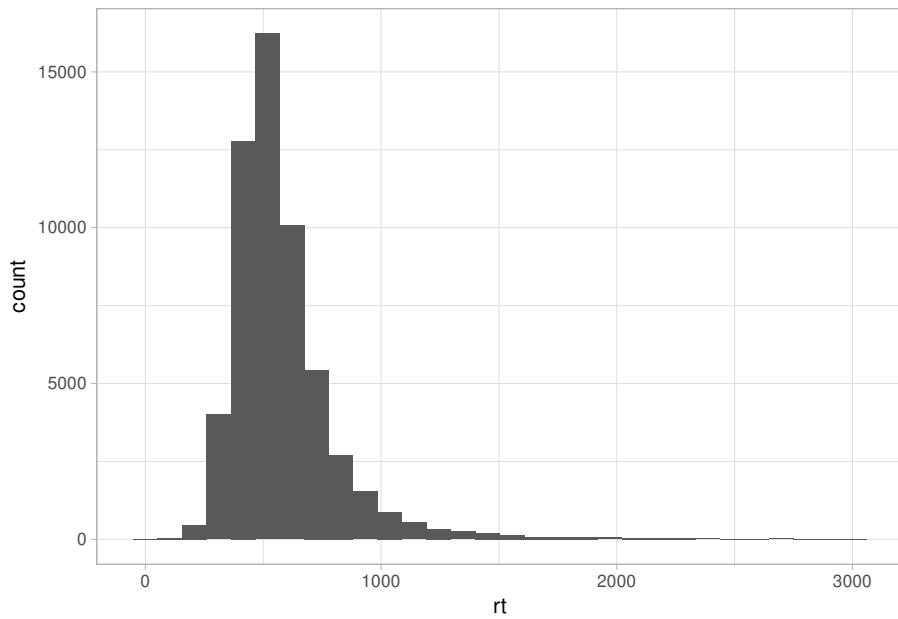


FIGURE 20.2: Distribution of response times in the data of the global motion detection task in Dutilh et al. (2019).

However, Figure 20.3 reveals that incorrect responses are generally faster, and this is especially true when the instructions emphasized accuracy.

20.1.2 A very simple implementation of the fast-guess model

The description of the model makes it clear that an ideal subject who never guesses has a response time that depends only on the difficulty of the trial. As we did in previous chapters, we assume that response times are log-normally distributed, and for simplicity we start by modeling the behavior of a single subject:

$$rt_n \sim LogNormal(\alpha + \beta \cdot x_n, \sigma) \quad (20.3)$$

In the previous equation, x is larger for difficult trials. If we center x , α represents the average logarithmic transformed response times for a subject engaged in the task, and β is the effect of trial difficulty on log-response time. We assume a non-deterministic process, with a noise pa-

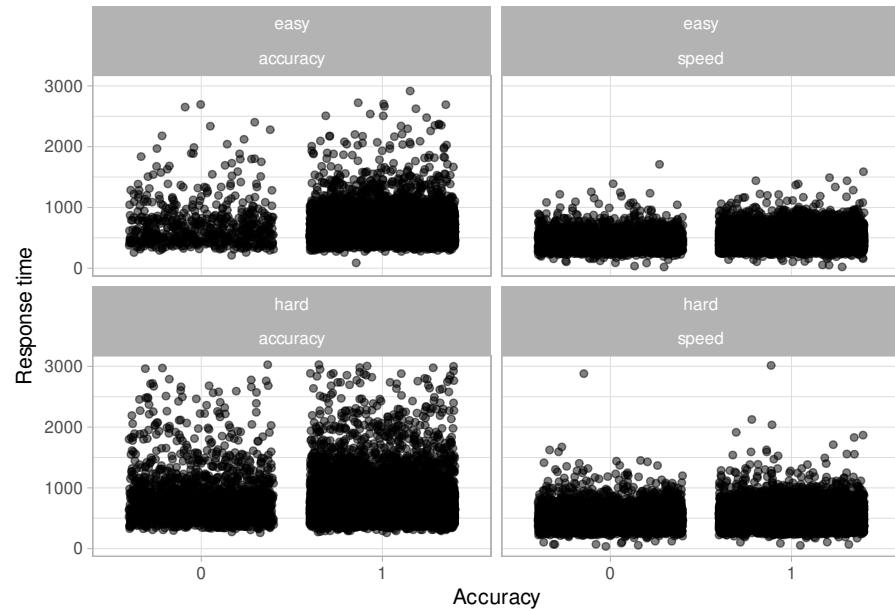


FIGURE 20.3: Distribution of response times by accuracy in the data of the global motion detection task in Dutilh et al. (2019).

parameter σ . See also Box 4.3 for more information about log-normally distributed response times.

Alternatively, a subject that guesses in every trial would show a response time distribution that is independent of the difficulty of the trial:

$$rt_n \sim LogNormal(\gamma, \sigma_2) \quad (20.4)$$

Here γ represents the average logarithmic transformed response time when a subject only guesses. We assume that responses from the guessing mode might have a different noise component than from the task-engaged mode.

The fast-guess model makes the assumption that during a task, a single subject would behave in these two ways: They would be engaged in the task a proportion of the trials and would guess on the rest of the trials. This means that for a single subject, there is an underlying probability of

being engaged in the task, p_{task} , that determines whether they are actually choosing ($z = 1$) or guessing ($z = 0$):

$$z_n \sim Bernoulli(p_{task}) \quad (20.5)$$

The value of the parameter z in every trial determines the behavior of the subject. This means that the distribution that we observe is a mixture of the two distributions presented before:

$$rt_n \sim \begin{cases} LogNormal(\alpha + \beta \cdot x_n, \sigma), & \text{if } z_n = 1 \\ LogNormal(\gamma, \sigma_2), & \text{if } z_n = 0 \end{cases} \quad (20.6)$$

In order to have a Bayesian implementation, we also need to define some priors. We use priors that encode what we know about reaction time experiments. These priors are slightly more informative than the ones that we used in section 4.2, but they still can be considered regularizing priors. One can verify this by performing prior predictive checks. As we increase the complexity of our models, it's worth to spend some time designing more realistic priors. These will speed up computation and in some cases they will be crucial to solve convergence problems.

$$\begin{aligned} \alpha &\sim Normal(6, 1) \\ \beta &\sim Normal(0, .1) \\ \sigma &\sim Normal_+(.5, .2) \end{aligned} \quad (20.7)$$

$$\begin{aligned} \gamma &\sim Normal(6, 1) \\ \sigma_2 &\sim Normal_+(.5, .2) \end{aligned} \quad (20.8)$$

For now, we allow all values for the probability of having an engaged response having equal likelihood; we achieve this by setting the following prior to p_{task} :

$$p_{task} \sim Beta(1, 1) \quad (20.9)$$

This represents a flat, uninformative prior over the probability parameter p_{task} .

Before we fit our model to the real data, we generate synthetic data to make sure that our model is working as expected.

We first define the number of observations, predictors, and fixed point values for each of the parameters. We assume 1000 observations and two levels of difficulty -0.5 and 0.5 . The point values chosen for the parameters are relatively realistic (based on our previous experience on reaction time experiments). Although in the priors we try to encode the range of possible values for the parameters, in this simulation we assume only one instance of this possible range:

```
N <- 1000
x <- c(rep(-.5, N/2), rep(.5, N/2))
# Parameters true values
alpha <- 5.8
beta <- 0.05
sigma <- .4
sigma2 <- .5
gamma <- 5.2
p_task <- .8
# Median time
c("engaged" = exp(alpha), "guessing" = exp(gamma))

## engaged guessing
##      330       181
```

For generate a mixture of response times, we use the indicator of a latent class, z .

```
z <- rbern(n = N, prob = p_task)
rt <- ifelse(z == 1,
             rlnorm(N,
                     meanlog = alpha + beta * x,
                     sdlog = sigma),
             rlnorm(N,
                     meanlog = gamma,
```

```
sdlog = sigma2))
df_dots_simdata1 <- tibble(trial = 1:N, x = x, rt = rt)
```

We verify that our simulated data is realistic, that is, it's on the same range as the original data; see Figure 20.4.

```
ggplot(df_dots_simdata1, aes(rt)) +
  geom_histogram()
```

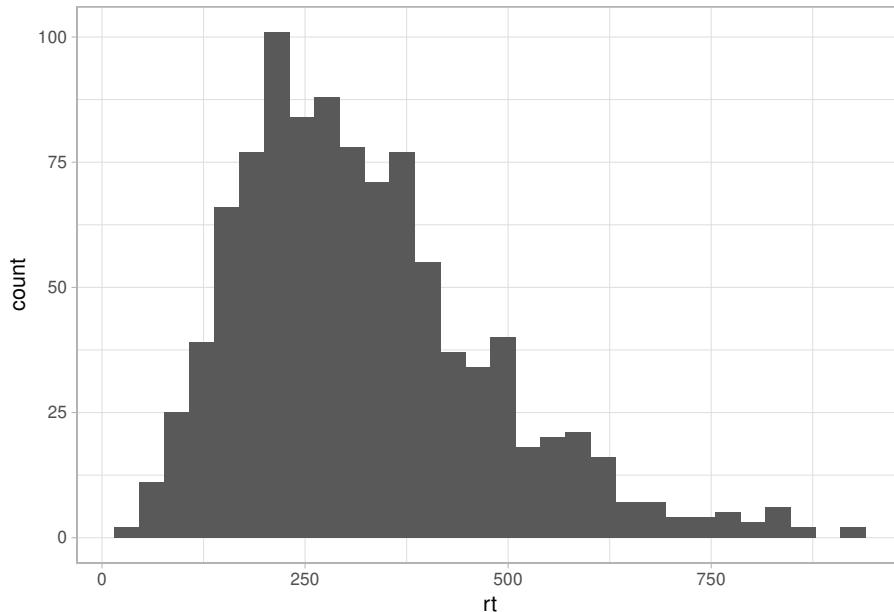


FIGURE 20.4: Response times in the simulated data (`df_dots_simdata1`) that follows the fast-guess model.

To implement the mixture model defined in Equation (3.13) in Stan, the discrete parameter z needs to be marginalized:

$$p(rt_n | \Theta) = p_{task} \cdot \text{LogNormal}(rt_n | \alpha + \beta \cdot x_n, \sigma) + \\ (1 - p_{task}) \cdot \text{LogNormal}(rt_n | \gamma, \sigma_2) \quad (20.10)$$

In addition, Stan requires the likelihood to be defined in log-space:

$$\log(p(rt|\Theta)) = \log(p_{task} \cdot \text{LogNormal}(rt_n|\alpha + \beta \cdot x_n, \sigma) + (1 - p_{task}) \cdot \text{LogNormal}(rt_n|\gamma, \sigma_2)) \quad (20.11)$$

A “naive” implementation in Stan would look like the following (recall that `_lpdf` functions provide log-transformed densities)

```
target += log(p_task + exp(lognormal_lpdf(rt[n] | alpha + x[n] * beta, sigma)),
              (1-p_task) + exp(lognormal_lpdf(rt[n] | gamma, sigma2)));
```

However, we need to take into account that $\log(A \pm B)$ can be numerically unstable (i.e., prone to under/overflow), when A is much larger than B or vice-versa. Stan provides several functions to deal with different special cases of logarithms of sums and differences. Here we need `log_sum_exp(x, y)` that corresponds to `log(exp(x) + exp(y))` and `log1m(x)` that corresponds to `log(1-x)`.

First, we need to take into account that the first summand of the logarithm, `p_task + exp(lognormal_lpdf(rt[n] | alpha + x[n] * beta, sigma))` corresponds to `exp(x)`, and the second one, `(1-p_task) + exp(lognormal_lpdf(rt[n] | gamma, sigma2))` to `exp(y)` in `log_sum_exp(x, y)`. This means that we need to first apply the logarithm to each of them to use them as arguments of `log_sum_exp(x, y)`:

```
target += log_sum_exp(
    log(p_task) + lognormal_lpdf(rt[n] | alpha + x[n] * beta, sigma),
    log(1-p_task) + lognormal_lpdf(rt[n] | gamma, sigma2));
```

Now we can just replace `log(1-p_task)` by the more stable `log1m(p_task)`:

```
target += log_sum_exp(
    log(p_task) + lognormal_lpdf(rt[n] | alpha + x[n] * beta, sigma),
    log1m(p_task) + lognormal_lpdf(rt[n] | gamma, sigma2));
```

The complete model is shown below:

```
data {
    int<lower = 1> N;
    vector[N] x;
    vector[N] rt;
}
```

```

parameters {
    real alpha;
    real beta;
    real<lower = 0> sigma;
    real gamma; //guessing
    real<lower = 0> sigma2;
    real<lower = 0, upper = 1> p_task;
}
model {
    // priors for the task component
    target += normal_lpdf(alpha | 6, 1);
    target += normal_lpdf(beta | 0, .1);
    target += normal_lpdf(sigma | .5, .2)
        - normal_lccdf(0 | .5, .2);
    // priors for the guessing component
    target += normal_lpdf(gamma | 6, 1);
    target += normal_lpdf(sigma2 | .5, .2)
        - normal_lccdf(0 | .5, .2);
    target += beta_lpdf(p_task | 1, 1);
    for(n in 1:N)
        target += log_sum_exp(log(p_task) +
            lognormal_lpdf(rt[n] | alpha + x[n] * beta, sigma),
            log1m(p_task) +
            lognormal_lpdf(rt[n] | gamma, sigma2));
}

```

Call the Stan model `mixture_rt.stan`, and fit it to the simulated data. First, we set up the simulated data as a list structure:

```

ls_dots_simdata <- list(N = N,
                        rt = rt,
                        x = x)

```

Then fit the model:

```

mixture_rt <- system.file("stan_models",
                           "mixture_rt.stan",
                           package = "rstan")

```

```

    package = "bcogsci")
fit_mix_rt <- stan(file = mixture_rt,
                     data = ls_dots_simdata)

## Warning: The largest R-hat is 1.74, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#r-hat

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians ma
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess

## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail c
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess

```

There are a lot of warnings, the Rhats are too large, and number of effective samples is too low:

```

print(fit_mix_rt)

##          mean   2.5%   97.5% n_eff Rhat
## alpha     5.49   4.86   5.84    2 2.49
## beta      0.08  -0.04   0.26    3 1.60
## sigma     0.47   0.38   0.61    3 1.71
## gamma     5.47   4.67   5.86    2 2.27
## sigma2    0.47   0.37   0.64    3 1.63
## p_task    0.52   0.09   0.94    2 2.80
## lp__  -6381.80 -6386.41 -6378.52    6 1.25

```

A traceplot shows clearly that the chains aren't mixing; see Figure 20.5.

```
traceplot(fit_mix_rt)
```

The problem with this model is that the mixture components (i.e., the fast-guesses and the engaged mode) are underlyingly exchangeable and thus not identifiable. Each chain doesn't know how each component was identified by the rest of the chains. However, we do have information that

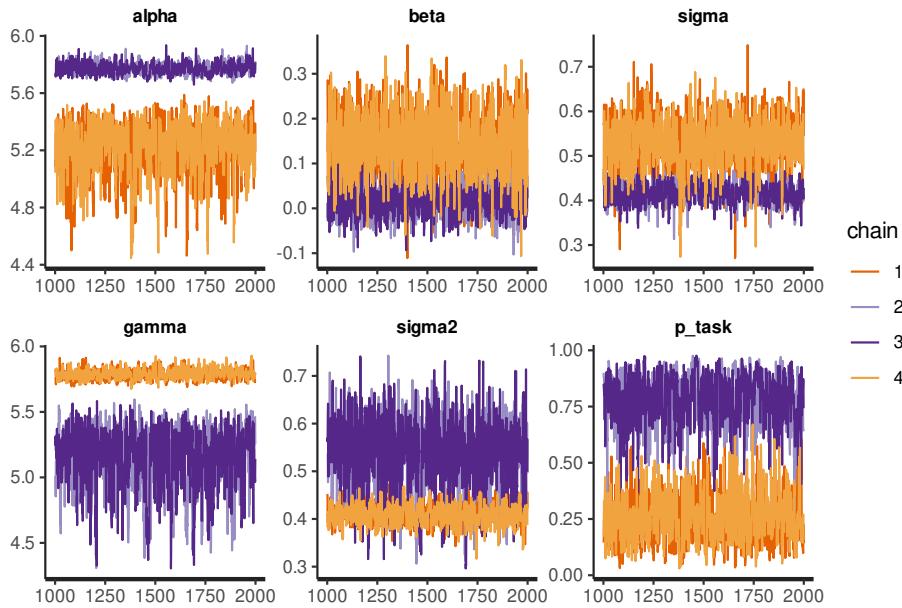


FIGURE 20.5: Traceplots from the model `mixture_rt.stan` fit to simulated data.

can identify the components: According to the theoretical model, we know that the average response in the engaged mode, represented by α , should be slower than the average response in the guessing mode, γ .

Even though the theoretical model assumes that guesses are faster than engaged responses, this is not explicit in our computational model. That is, our model lacks some of the theoretical information that we have, namely that the distribution of engaged response times should be slower than the distribution of guess times. This can be encoded with a strong prior for γ , where we assume that its prior distribution is truncated on an upper bound by the value of α :

$$\gamma \sim \text{Normal}(6, 1), \text{ for } \gamma < \alpha \quad (20.12)$$

This would be enough to make the model identifiable.

Another softer constraint that we could add to our implementation is the assumption that subjects are generally more likely to be trying to do the task than just guessing. If this assumption is correct, we also improve the

accuracy of our estimation of the posterior of the model. (The opposite is also true: If subjects are not trying to do the task, this assumption will be unwarranted and our prior information will lead us farther from the “true” values of the parameters). The following prior has the probability density concentrated nearer to 1.

$$p_{task} \sim Beta(8, 2) \quad (20.13)$$

Plotting this prior confirms where most of the probability mass lies; see Figure 20.6.

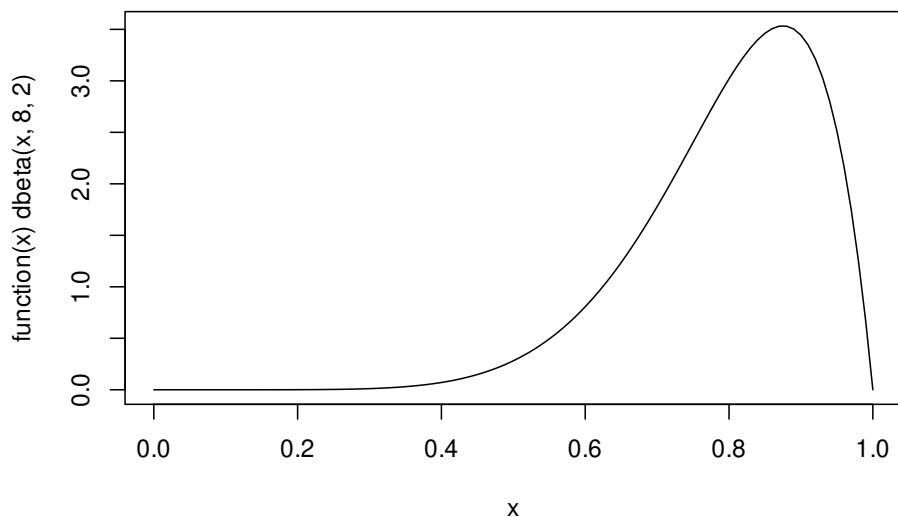


FIGURE 20.6: Density plot for the prior of p_{task} , $Beta(8, 2)$

The Stan code for this model is shown below as `mixture_rt2.stan`.

```
data {
    int<lower = 1> N;
    vector[N] x;
    vector[N] rt;
}
parameters {
    real alpha;
    real beta;
    real<lower = 0> sigma;
```

```

real<upper = alpha> gamma; //guessing
real<lower = 0> sigma2;
real<lower = 0, upper = 1> p_task;
}
model {
    // priors for the task component
    target += normal_lpdf(alpha | 6, 1);
    target += normal_lpdf(beta | 0, .3);
    target += normal_lpdf(sigma | .5, .2)
        - normal_lccdf(0 | .5, .2);
    // priors for the guessing component
    target += normal_lpdf(gamma | 6, 1) -
        normal_lcdf(alpha | 6, 1);
    target += normal_lpdf(sigma2 | .5, .2)
        - normal_lccdf(0 | .5, .2);
    target += beta_lpdf(p_task | 8, 2);
    for(n in 1:N)
        target += log_sum_exp(log(p_task) +
            lognormal_lpdf(rt[n] | alpha + x[n] * beta, sigma),
            log1m(p_task) +
            lognormal_lpdf(rt[n] | gamma, sigma2));
}

```

Once we change the higher boundary of `gamma` in the `parameters` block, we also need to truncate the distribution in the `model` block by correcting the PDF with its CDF. This correction is carried out using the CDF because we are truncating the distribution at the right-hand side; recall that earlier we used the complement of the CDF when we truncate a distribution at the left-hand side); also see Box 4.1.

```

target += normal_lpdf(gamma | 6, 1) -
    normal_lcdf(alpha | 6, 1);

```

Fit this model (call it `mixture_rt2.stan`) to the same simulated data set that we used before:

```

mixture_rt2 <- system.file("stan_models",
    "mixture_rt2.stan",

```

```
package = "bcogsci")
fit_mix_rt2 <- stan(file = mixture_rt2,
                     data = ls_dots_simdata)
```

Now the summaries and traceplots look fine; see Figure 20.7.

```
print(fit_mix_rt2)
```

	mean	2.5%	97.5%	n_eff	Rhat
## alpha	5.78	5.72	5.85	994	1.01
## beta	0.02	-0.04	0.08	2452	1.00
## sigma	0.38	0.34	0.42	1037	1.01
## gamma	5.07	4.61	5.48	729	1.01
## sigma2	0.45	0.25	0.61	790	1.00
## p_task	0.81	0.57	0.95	789	1.01
## lp__	-6331.84	-6335.92	-6329.39	1405	1.00

```
traceplot(fit_mix_rt2)
```

20.1.3 A multivariate implementation of the fast-guess model

A problem with the previous implementation of the fast-guess model is that we ignore the accuracy information in the data. We can implement a closer version of the verbal description of the model: In particular, we also want to model the fact that accuracy is at chance level in the fast-guessing mode and that accuracy approaches 100% during the task-engaged mode.

This means that the mixture affects two pairs of distributions:

$$z_n \sim Bernoulli(p_{task}) \quad (20.14)$$

A response time distribution

$$rt_n \sim \begin{cases} LogNormal(\alpha + \beta \cdot x_n, \sigma), & \text{if } z_n = 1 \\ LogNormal(\gamma, \sigma_2), & \text{if } z_n = 0 \end{cases} \quad (20.15)$$

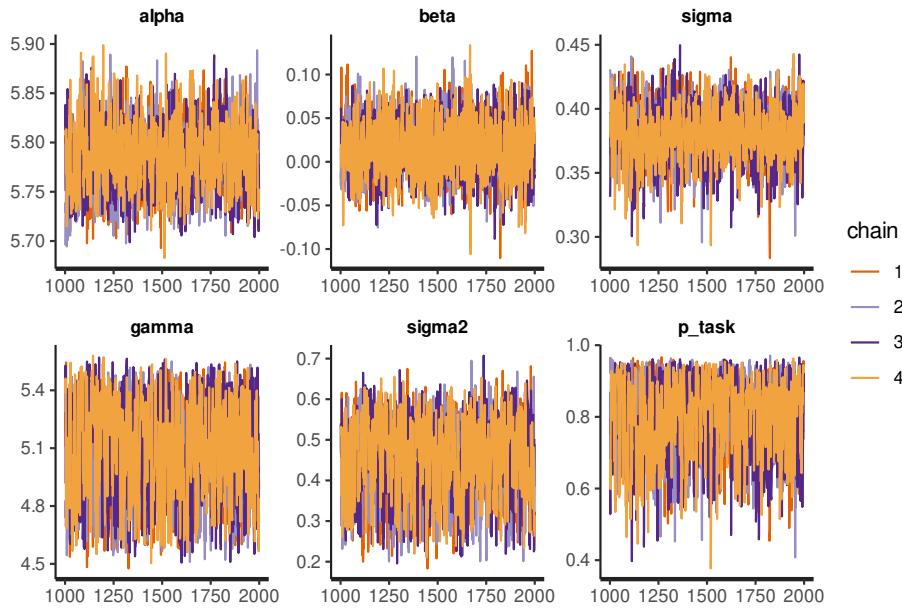


FIGURE 20.7: Traceplots from the model `mixture_rt2.stan` fit to simulated data.

and an accuracy distribution

$$acc_n \sim \begin{cases} Bernoulli(p_{correct}), & \text{if } z_n = 1 \\ Bernoulli(.5), & \text{if } z_n = 0 \end{cases} \quad (20.16)$$

We have a new parameter $p_{correct}$, which represent the probability of making a correct answer in the engaged mode. The verbal description says that it is closer to 100%, and here we have the freedom to choose whatever prior we believe represents for us values that are close to 100% accuracy. We translate this belief into a prior as follows; our prior choice is relatively informative but does not impose a hard constraint; if a subject consistently shows relatively low (or high) accuracy, $p_{correct}$ will change accordingly:

$$p_{correct} \sim Beta(995, 5) \quad (20.17)$$

In our simulated data, we assume that the global motion detection task is done by a very accurate subject, with an accuracy of 99.9%.

First, simulate reaction times as before.

```
N <- 1000
x <- c(rep(-.5, N/2), rep(.5, N/2))
alpha <- 5.8
beta <- 0.05
sigma <- .4
sigma2 <- .5
gamma <- 5.2
p_task <- .8
z <- rbern(n = N, prob = p_task)
rt <- ifelse(z == 1,
             rlnorm(N,
                     meanlog = alpha + beta * x,
                     sdlog = sigma),
             rlnorm(N,
                     meanlog = gamma,
                     sdlog = sigma2))
```

Simulate accuracy and include both reaction times and accuracy in the simulated dataset.

```
p_correct <- .999
acc <- ifelse(z, rbern(n = N, p_correct),
               rbern(n = N, .5))
df_dots_simdata3 <- tibble(trial = 1:N,
                            x = x,
                            rt = rt,
                            acc = acc) %>%
  mutate(diff = ifelse(x == .5, "hard", "easy"))
```

Plot the simulated data in Figure 20.8. This time we can see the effect of task difficulty on the simulated response times and accuracy:

```
ggplot(df_dots_simdata3, aes(x = factor(acc), y = rt)) +
  geom_point(position = position_jitter(width = .4, height = 0),
```

```

alpha = .5) +
facet_wrap(diff ~ .) +
xlab("Accuracy") +
ylab("Response time")

```

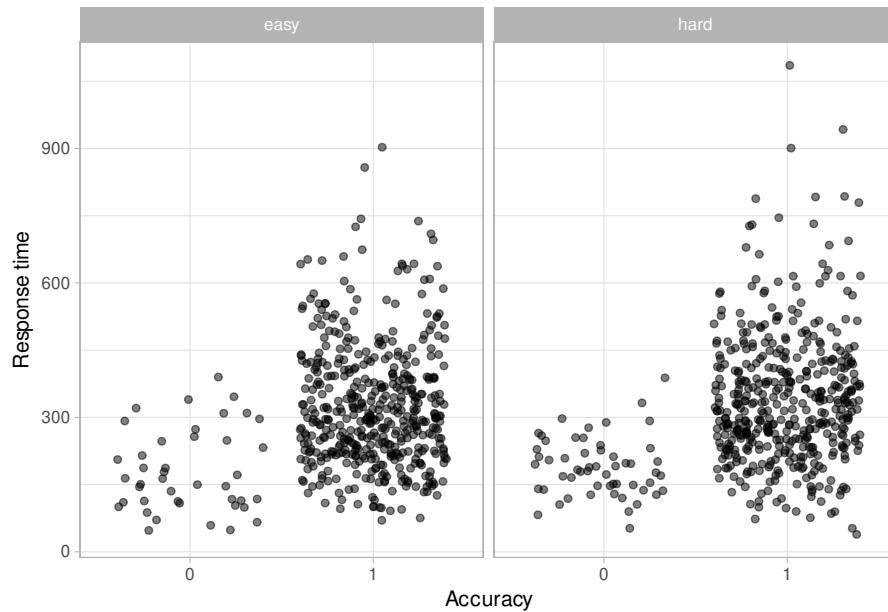


FIGURE 20.8: Response times by accuracy accounting for task difficulty in the simulated data (`df_dots_simdata3`) that follows the fast-guess model.

Next, we need to marginalize out the discrete parameters from both pairs of distributions.

$$\begin{aligned}
p(rt, acc | \Theta) = & p_{task} \cdot \\
& LogNormal(rt_n | \alpha + \beta \cdot x_n, \sigma) \cdot \\
& Bernoulli(acc_n | p_{correct}) \\
& + \\
& (1 - p_{task}) \cdot \\
& LogNormal(rt_n | \gamma, \sigma_2) \cdot \\
& Bernoulli(acc_n | .5)
\end{aligned} \tag{20.18}$$

In log-space:

$$\begin{aligned}
 \log(p(rt, acc | \Theta)) = & \log(\exp(
 \log(p_{task}) + \\
 \log(LogNormal(rt_n | \alpha + \beta * x_n, \sigma)) + \\
 \log(Bernoulli(acc_n | p_{correct}))) \\
 + \\
 \exp(
 \log(1 - p_{task}) + \\
 \log(LogNormal(rt_n | \gamma, \sigma_2)) + \\
 \log(Bernoulli(acc_n | .5)))
)
 \end{aligned} \tag{20.19}$$

Our model translates into the following Stan code (`mixture_rtacc.stan`):

```

data {
    int<lower = 1> N;
    vector[N] x;
    vector[N] rt;
    int acc[N];
}
parameters {
    real alpha;
    real beta;
    real<lower = 0> sigma;
    real<upper = alpha> gamma; //guessing
    real<lower = 0> sigma2;
    real<lower = 0, upper = 1> p_correct;
    real<lower = 0, upper = 1> p_task;
}
model {
    // priors for the task component
    target += normal_lpdf(alpha | 6, 1);
    target += normal_lpdf(beta | 0, .3);
    target += normal_lpdf(sigma | .5, .2)
}

```

```

    - normal_lccdf(0 | .5, .2);
    // priors for the guessing component
    target += normal_lpdf(gamma | 6, 1) -
        normal_lcdf(alpha | 6, 1);
    target += normal_lpdf(sigma2 | .5, .2)
        - normal_lccdf(0 | .5, .2);
    target += beta_lpdf(p_correct | 995, 5);
    target += beta_lpdf(p_task | 8, 2);
    for(n in 1:N)
        target += log_sum_exp(log(p_task) +
            lognormal_lpdf(rt[n] | alpha + x[n] * beta, sigma) +
            bernoulli_lpmf(acc[n] | p_correct),
            log1m(p_task) +
            lognormal_lpdf(rt[n] | gamma, sigma2) +
            bernoulli_lpmf(acc[n] | .5));
}

```

Next, set up the data in list format:

```

ls_dots_simdata <- list(N = N,
                         rt = rt,
                         x = x,
                         acc = acc)

```

Then fit the model:

```

mixture_rtacc <- system.file("stan_models",
                             "mixture_rtacc.stan",
                             package = "bcogsci")
fit_mix_rtacc <- stan(file = mixture_rtacc,
                      data = ls_dots_simdata)

```

We see that our model can be fit to both response times and accuracy, and its parameters estimates have sensible values (given the fixed parameters we used to generate our simulated data).

```
print(fit_mix_rtacc)
```

	mean	2.5%	97.5%	n_eff	Rhat
## alpha	5.79	5.76	5.82	4238	1
## beta	0.02	-0.04	0.08	5445	1
## sigma	0.38	0.36	0.41	5539	1
## gamma	5.17	5.07	5.27	3742	1
## sigma2	0.50	0.43	0.57	3999	1
## p_correct	0.99	0.99	1.00	4608	1
## p_task	0.80	0.76	0.84	5089	1
## lp__	-6607.59	-6612.33	-6604.88	1864	1

We will evaluate the recovery of the parameters more carefully when we deal with the hierarchical version of the fast-guess model in section 20.1.5. Before we extend this model hierarchically, let us also take into account the instructions given to the subjects.

20.1.4 An implementation of the fast-guess model that takes instructions into account

The actual global motion detection experiment that we started with has another manipulation that can help us to evaluate better the fast-guess model. In some trials, the instructions emphasized accuracy (e.g., “Be as accurate as possible.”) and in others speed (e.g., “Be as fast as possible.”). The fast-guess model also assumes that the probability of being in one of the two states depends on the speed incentives given during the instructions. This entails that now p_{task} depends on the instructions x_2 , where we encode a speed incentive with -0.5 and an accuracy incentive with 0.5 . Essentially, we need to fit the following regression:

$$\alpha_{task} + x_2 \cdot \beta_{task} \quad (20.20)$$

As we did with MPT models in the previous chapter (in section 19.2.3), we need to bound the previous regression between 0 and 1; we achieve this using the logistic or inverse logit function:

$$p_{task} = \text{logit}^{-1}(\alpha_{task} + x_2 \cdot \beta_{task}) \quad (20.21)$$

This means that we need to interpret $\alpha_{task} + x_2 \cdot \beta_{task}$ in log-odds, which is bounded by $(-\infty, \infty)$ rather than as a probability; see also section 19.2.3 in the previous chapter.

The likelihood (defined before in section 20.1.3) remains the same:

$$z_n \sim Bernoulli(p_{task}) \quad (20.22)$$

A response time distribution is defined:

$$rt_n \sim \begin{cases} LogNormal(\alpha + \beta \cdot x_n, \sigma), & \text{if } z_n = 1 \\ LogNormal(\gamma, \sigma_2), & \text{if } z_n = 0 \end{cases} \quad (20.23)$$

and an accuracy distribution is defined as well:

$$acc_n \sim \begin{cases} Bernoulli(p_{correct}), & \text{if } z_n = 1 \\ Bernoulli(.5), & \text{if } z_n = 0 \end{cases} \quad (20.24)$$

The only further change in our model is that rather than a prior on p_{task} , we now need priors for α_{task} and β_{task} , which are on the log-odds scale.

For β_{task} , we assume an effect that can rather large and we won't assume a direction a priori (for now):

$$\beta_{task} \sim Normal(0, 1) \quad (20.25)$$

This means that the subject could be affected by the instructions in the expected way with an increased probability to be task-engaged (leading to better accuracy) when the instructions emphasize accuracy ($\beta_{task} > 0$), or the subject might be behaving in an unexpected way with a decreased probability to be task-engaged (leading to worse accuracy) when the instructions emphasize accuracy ($\beta_{task} < 0$). The latter situation, $\beta_{task} < 0$, could represent the instructions being misunderstood. It's certainly possible to include priors that encode the expected direction of the effect instead: $Normal_+(0, 1)$. Unless there is a compelling reason to constrain the prior in this way, in general we will, following Cromwell's rule, leave open the possibility of the β parameter having negative values.

How can we choose a prior for α_{task} that encodes the same information that we had in the previous model in p_{task} ? One possibility is to create an auxiliary parameter p_{btask} , that represents the baseline probability of being engaged in the task, with the same prior that we use in the previous section, and then transform it to an unconstrained space for our regression with the logit function:

$$\begin{aligned} p_{btask} &\sim \text{Beta}(8, 2) \\ \alpha_{task} &= \text{logit}(p_{btask}) \end{aligned} \tag{20.26}$$

To verify that our priors make sense, we plot the difference in prior predicted probability of being engaged in the task under the two emphasis conditions in Figure 20.9.

```
Ns <- 1000 # number of samples for the plot
# Priors
p_btask <- rbeta(n = Ns, shape1 = 8, shape2 = 2)
beta_task <- rnorm(n = Ns, mean = 0, sd = 1)
# Predicted probability of being engaged
p_task_easy <- plogis(qlogis(p_btask) + .5 * beta_task)
p_task_hard <- plogis(qlogis(p_btask) + -.5 * beta_task)
# Predicted difference
diff_p_pred <- tibble(diff = p_task_easy - p_task_hard)

diff_p_pred %>%
  ggplot(aes(diff)) +
  geom_histogram()
```

Figure 20.9 shows that we are predicting a priori that the difference in p_{task} will be mostly smaller than ± 0.3 , which seems to make sense intuitively. If we had more information about the likely range of variation, we could of course have adapted the prior to reflect that belief.

We are ready to generate a new data set, by deciding on some fixed values for β_{task} and p_{btask} .

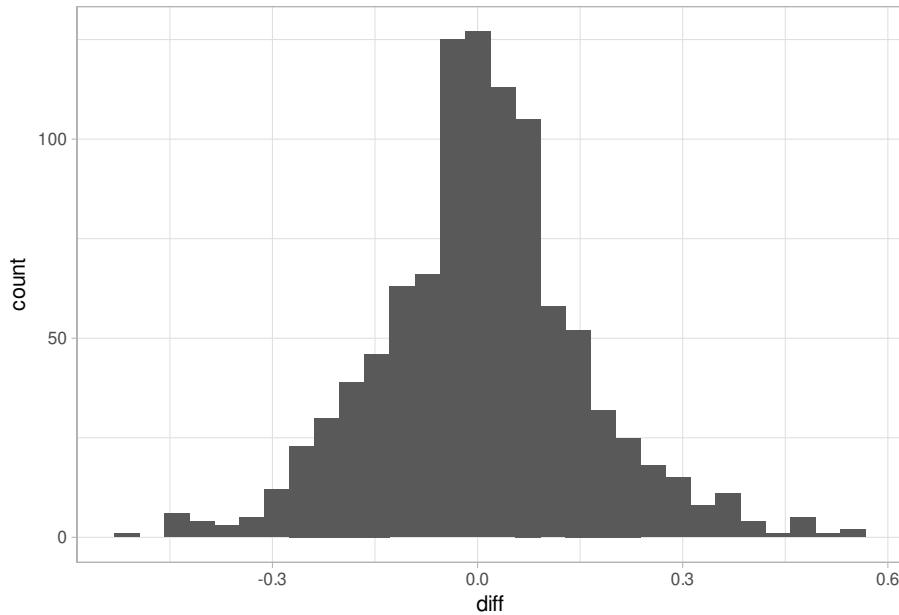


FIGURE 20.9: Difference in prior predicted probability of being engaged in the task under the two emphasis conditions for the simulated data (diff_p_pred) that follows the fast-guess model.

```
# New predictor
x2 <- rep(c(-.5, .5), N/2)
# Verify that the predictors are crossed:
predictors <- tibble(x, x2)
xtabs(~ x + x2, predictors)
```

```
##          x2
## x      -0.5 0.5
##   -0.5 250 250
##   0.5 250 250
```

```
# True values:
N <- 1000
x <- c(rep(-.5, N/2), rep(.5, N/2))
alpha <- 5.8
```

```

beta <- 0.05
sigma <- .4
sigma2 <- .5
gamma <- 5.2
p_correct <- .999
# New true values:
p_btask <- .85
beta_task <- 0.5
# Generate data:
alpha_task <- qlogis(p_btask)
p_task <- plogis(alpha_task + x2 * beta_task)
z <- rbern(N, prob = p_task)
rt <- ifelse(z,
              rlnorm(N, meanlog = alpha + beta * x, sdlog = sigma),
              rlnorm(N, meanlog = gamma, sdlog = sigma2))
acc <- ifelse(z, rbern(N, p_correct),
               rbern(N, .5))
df_dots_simdata4 <- tibble(trial = 1:N,
                             x = x,
                             rt = rt,
                             acc = acc,
                             x2 = x2) %>%
  mutate(diff = ifelse(x == .5, "hard", "easy"),
        emphasis = ifelse(x2 == .5, "accuracy", "speed"))

```

We can generate a plot now where both the difficulty of the task and the instructions are manipulated; see Figure 20.10.

```

ggplot(df_dots_simdata4, aes(x = factor(acc), y = rt)) +
  geom_point(position = position_jitter(width = .4, height = 0),
             alpha = .5) +
  facet_wrap(diff ~ emphasis) +
  xlab("Accuracy") +
  ylab("Response time")

```

In the Stan implementation, `log_inv_logit(x)` is applying the logistic (or

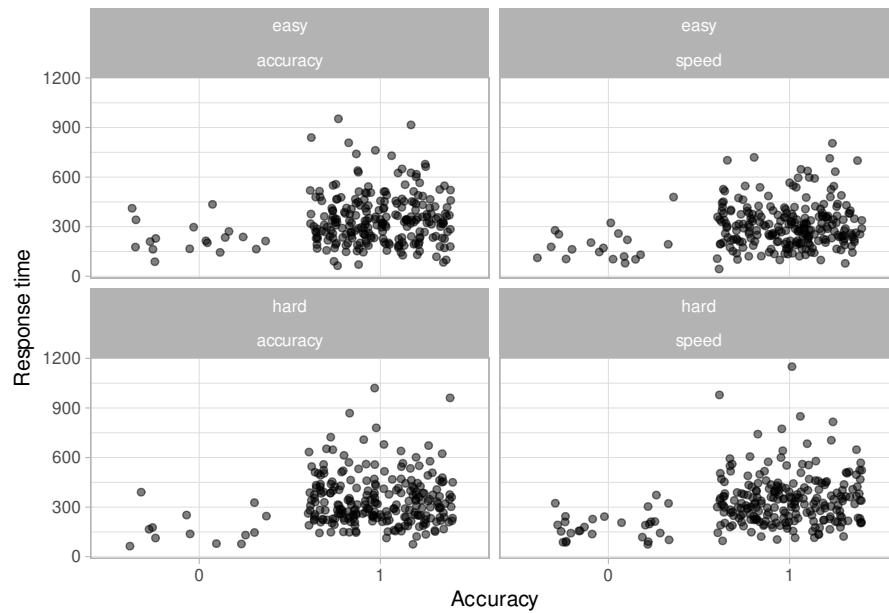


FIGURE 20.10: Response times and accuracy by the difficulty of the task and the instructions type for the simulated data (`df_dots_simdata4`) that follows the fast-guess model.

inverse logit) function to x to transform it into a probability and then applying the logarithm; `log1m_inv_logit(x)` is applying the logistic function to x , and then applying the logarithm to its complement $(1 - p)$. We do this because rather than having `p_task` in probability space, we have `lodds_task` in log-odds space:

```
real lodds_task = logit(p_btask) + x2[n] * beta_task;
```

The parameter `lodds_task` estimates the mixing probabilities in log-odds:

```
target += log_sum_exp(log_inv_logit(lodds_task) + ...,
                      log1m_inv_logit(lodds_task) + ...)
```

We also add a generated quantities block that can be used for further (prior or posterior) predictive checks. In this block we do use z as an indicator of the latent class (task-engaged mode or fast-guessing mode), since we do not estimate z , but rather generate it based on the parameter's posteriors.

We use the dummy variable `onlyprior` to indicate whether we use the data or we only sample from the priors. One can always do the predictive checks in R, transforming the code that we wrote for the simulation into a function, and writing the priors in R. However, it can be simpler to take advantage of Stan output format and rewrite the code in Stan. One downside of this is that the `stanfit` object that stores the model output can become too large for the memory of the computer.

```

data {
    int<lower = 1> N;
    vector[N] x;
    vector[N] rt;
    int acc[N];
    vector[N] x2; //speed or accuracy emphasis
    int<lower = 0, upper = 1> onlyprior;
}
parameters {
    real alpha;
    real beta;
    real<lower = 0> sigma;
    real<upper = alpha> gamma; //guessing
    real<lower = 0> sigma2;
    real<lower = 0, upper = 1> p_correct;
    real<lower = 0, upper = 1> p_btask;
    real beta_task;
}
model {
    // priors for the task component
    target += normal_lpdf(alpha | 6, 1);
    target += normal_lpdf(beta | 0, .1);
    target += normal_lpdf(sigma | .5, .2)
        - normal_lccdf(0 | .5, .2);
    // priors for the guessing component
    target += normal_lpdf(gamma | 6, 1)-
        normal_lcdf(alpha | 6, 1);
    target += normal_lpdf(sigma2 | .5, .2)
        - normal_lccdf(0 | .5, .2);
    target += normal_lpdf(beta_task | 0, 1);
}

```

```

target += beta_lpdf(p_correct | 995, 5);
target += beta_lpdf(p_btask | 8, 2);
if(onlyprior != 1)
  for(n in 1:N){
    real lodds_task = logit(p_btask) + x2[n] * beta_task;
    target += log_sum_exp(log_inv_logit(lodds_task) +
      lognormal_lpdf(rt[n] | alpha + x[n] * beta, sigma) +
      bernoulli_lpmf(acc[n] | p_correct),
      log1m_inv_logit(lodds_task) +
      lognormal_lpdf(rt[n] | gamma, sigma2) +
      bernoulli_lpmf(acc[n] | .5));
  }
}
generated quantities {
  real rt_pred[N];
  real acc_pred[N];
  int z[N];
  for(n in 1:N){
    real lodds_task = logit(p_btask) + x2[n] * beta_task;
    z[n] = bernoulli_rng(inv_logit(lodds_task));
    if(z[n]==1){
      rt_pred[n] = lognormal_rng(alpha + x[n] * beta, sigma);
      acc_pred[n] = bernoulli_rng(p_correct);
    } else{
      rt_pred[n] = lognormal_rng(gamma, sigma2);
      acc_pred[n] = bernoulli_rng(.5);
    }
  }
}

```

We save the code as `mixture_rtacc2.stan`, and before fitting it to the simulated data, we perform prior predictive checks.

20.1.4.1 Prior predictive checks

Generate prior predictive distributions, by setting `onlyprior` to 1.

```
ls_dots_simdata <- list(N = N,
                        rt = rt,
                        x = x,
                        x2 = x2,
                        acc = acc,
                        onlyprior = 1)
mixture_rtacc2 <- system.file("stan_models",
                               "mixture_rtacc2.stan",
                               package = "bcogsci")
fit_mix_rtacc2_priors <- stan(file = mixture_rtacc2,
                                data = ls_dots_simdata,
                                chains = 1, iter = 2000)
```

We plot prior predictive distributions of response times as follows. We are plotting them against our simulated data in Figure 20.11, by setting $y = rt$ in `ppc_dens_overlay`; this distribution can be thought as a hand-picked instance of the prior predictive distribution.

```
rt_pred <- extract(fit_mix_rtacc2_priors)$rt_pred
ppc_dens_overlay(y = rt, yrep = rt_pred[1:100,]) +
  coord_cartesian(xlim = c(0, 5000))
```

Some of the predictive data sets contain responses that are too large, and some of the have too much probability mass close to zero, but there is nothing clearly wrong in the prior predictive distributions (considering that the model hasn't "seen" the data).

If we want to plot the prior predicted distribution of differences in response time conditioning on task difficulty, we need to define a new function. Then we use the `bayesplot` function `ppc_stat()` that takes as an argument of `stat` any summary function; see Figure 20.12.

```
meanrt_diff <- function(rt){
  mean(rt[x == .5]) -
  mean(rt[x == -.5])
}
```

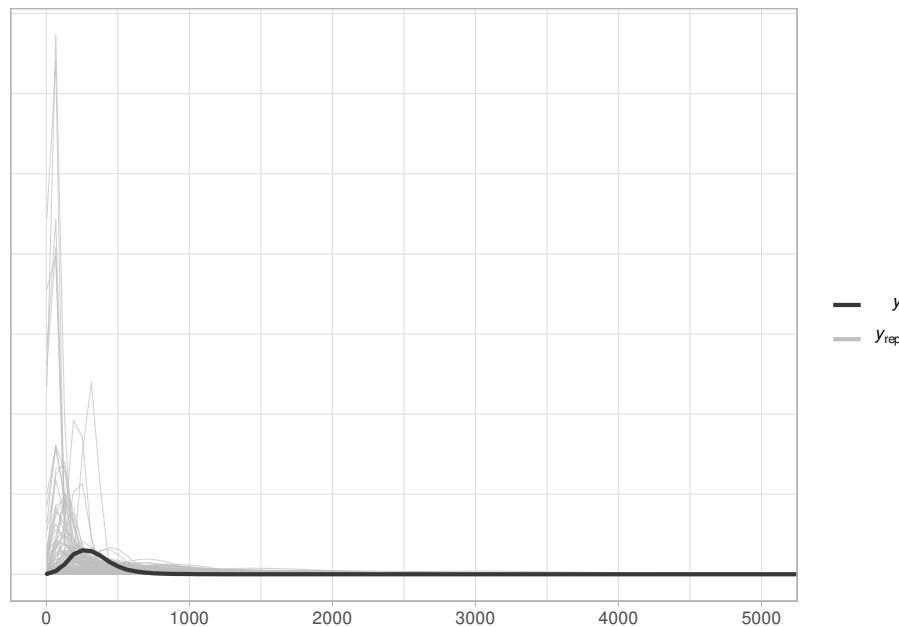


FIGURE 20.11: Prior predictive distributions of response times against the simulated data (using `mixture_rtacc2.stan`) that follows the fast-guess model.

```
ppc_stat(rt,
          yrep = rt_pred,
          stat = meanrt_diff)
```

We find that the range of response times look reasonable. There are, however, always more checks that can be done, for example, plotting other summary statistics, or predictions conditioned on other aspects of the data.

20.1.4.2 Fit to simulated data

Fit it to data, by setting `onlyprior = 0`:

```
ls_dots_simdata <- list(N = N,
                         rt = rt,
```

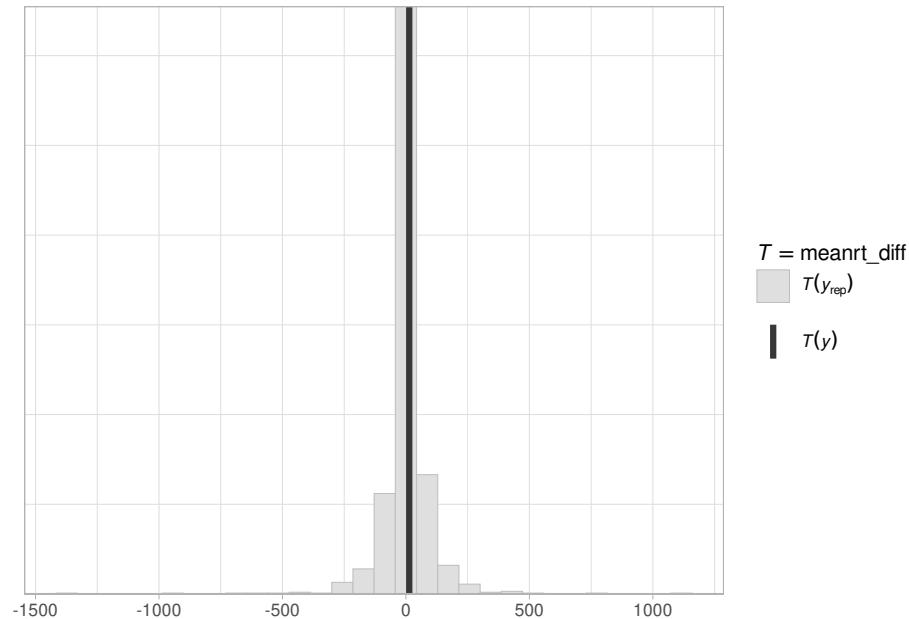


FIGURE 20.12: Prior predicted distribution (using `mixture_rtacc2.stan`) of differences in response time conditioning on task difficulty.

```
x = x,
x2 = x2,
acc = acc,
onlyprior = 0)
```

```
fit_mix_rtacc2 <- stan(file = mixture_rtacc2,
                        data = ls_dots_simdata)
```

```
print(fit_mix_rtacc2,
      pars = c("alpha", "beta", "sigma", "gamma", "sigma2",
              "p_correct", "p_btask", "beta_task"))
```

```
##          mean 2.5% 97.5% n_eff Rhat
## alpha     5.80 5.77  5.83   5889    1
## beta      0.06 0.00  0.12   7200    1
```

```
## sigma     0.39 0.37  0.41  5873    1
## gamma    5.21 5.09  5.32  4366    1
## sigma2   0.53 0.46  0.61  4721    1
## p_correct 0.99 0.99  1.00  4756    1
## p_btask   0.85 0.82  0.88  4521    1
## beta_task 1.12 0.64  1.63  5780    1
```

We see that we fit the model without problems. Before we evaluate the recovery of the parameters more carefully, we implement a hierarchical version of the fast-guess model.

20.1.5 A hierarchical implementation of the fast-guess model

So far we have evaluated the behavior of one simulated subject. We discussed before (in the context of distributional regression models, in section 5.1.6, and in the MPT modeling chapter, 19) that every parameter in a model can, in principle, be made hierarchical. This doesn't guarantee, however, that we'll learn anything from the data for those parameters or that our model will converge! The best advice here is to start simple with simulated data. Despite the fact that convergence with simulated data does not guarantee the convergence of the same model with real data, the reverse is in general true.

For our hierarchical version, we assume that both response times and the effect of task difficulty vary by subject, and that different subjects have different guessing times. This entails the following change to the response time distribution:

$$rt_n \sim \begin{cases} LogNormal(\alpha + u_{subj[n],1} + x_n \cdot (\beta + u_{subj[n],2}), \sigma), & \text{if } z_n = 1 \\ LogNormal(\gamma + u_{subj[n],3}, \sigma_2), & \text{if } z_n = 0 \end{cases} \quad (20.27)$$

We assume that the three vectors of u (adjustment to the intercept and slope of the task-engaged distribution, and the adjustment to the guessing time distribution) follow a multivariate normal distribution centered on zero. For simplicity and lack of any prior knowledge about this experiment design and method, we assume the same (weakly informative) prior distribution for the three variance components and the same reg-

ularizing LKJ prior for the three correlations between the adjustments ($\rho_{u_{1,2}}, \rho_{u_{1,3}}, \rho_{u_{2,3}}$):

$$\begin{aligned} \boldsymbol{u} &\sim \mathcal{N}(0, \Sigma_u) \\ \tau_{u_{1,3}} &\sim \text{Normal}_+(0, .5) \\ \rho_u &\sim \text{LKJcorr}(2) \end{aligned} \tag{20.28}$$

Before we fit the model to the real data set, we simulate data again; this time we simulate 100 trials of each of 20 subjects.

```
# Build the fake stimuli
N_subj <- 20
N_trials <- 100
# Parameters true values
alpha <- 5.8
beta <- 0.05
sigma <- .4
sigma2 <- .5
gamma <- 5.2
beta_task <- 0.1
p_btask <- .85
alpha_task <- qlogis(p_btask)
p_correct <- .999
tau_u <- c(.2, .005, .3)
rho <- .3
```

```
## Build the data set here:
N <- N_subj * N_trials
stimuli <- tibble(x = rep(c(rep(-.5, N_trials / 2),
                           rep(.5, N_trials / 2)), N_subj),
                  x2 = rep(rep(c(-.5, .5), N_trials / 2), N_subj),
                  subj = rep(1:N_subj, each = N_trials),
                  trial = rep(1:N_trials, N_subj))
stimuli
```

```

## # A tibble: 2,000 x 4
##       x     x2   subj trial
##   <dbl> <dbl> <int> <int>
## 1 -0.5  -0.5     1     1
## 2 -0.5    0.5     1     2
## 3 -0.5   -0.5     1     3
## # ... with 1,997 more rows

# Build the correlation matrix for the adjustments u[1...3]:
Cor_u <- matrix(rep(rho, 9), nrow = 3)
diag(Cor_u) <- 1
Cor_u

##      [,1] [,2] [,3]
## [1,]  1.0  0.3  0.3
## [2,]  0.3  1.0  0.3
## [3,]  0.3  0.3  1.0

# Variance covariance matrix for subjects:
Sigma_u <- diag(tau_u, 3, 3) %*% Cor_u %*% diag(tau_u, 3, 3)
# Create the correlated adjustments
u <- mvrnorm(n = N_subj, c(0, 0, 0), Sigma_u)
# Check whether they are correctly correlated
# (There will be some random variation,
# but if you increase the number of observations and
# the value of the correlation, you'll be able to obtain
# a more exact value below)
cor(u)

##      [,1] [,2] [,3]
## [1,] 1.000 0.375 0.495
## [2,] 0.375 1.000 0.500
## [3,] 0.495 0.500 1.000

# Check the SDs
sd(u[, 1]); sd(u[, 2]); sd(u[, 3])

```

```

## [1] 0.227
## [1] 0.0049
## [1] 0.353

# Create the simulated data
df_dots_simdata <- stimuli %>%
  mutate(z = rbern(N, prob = plogis(alpha_task + x2 * beta_task)),
        rt = ifelse(z,
                     rlnorm(N, meanlog = alpha + u[subj, 1] +
                           (beta + u[subj, 2]) * x,
                     sdlog = sigma),
                     rlnorm(N, meanlog = gamma + u[subj, 3],
                           sdlog = sigma2)),
        acc = ifelse(z, rbern(n = N, p_correct),
                     rbern(n = N, .5)),
        diff = ifelse(x == .5, "hard", "easy"),
        emphasis = ifelse(x2 == .5, "accuracy", "speed"))

```

Verify that the distribution of the simulated response times conditional on the simulated accuracy and the experimental manipulations make sense with the following plot shown in Figure 20.13.

```

ggplot(df_dots_simdata, aes(x = factor(acc), y = rt)) +
  geom_point(position = position_jitter(width = .4,
                                         height = 0), alpha = .5) +
  facet_wrap(diff ~ emphasis) +
  xlab("Accuracy") +
  ylab("Response time")

```

We implement the model in Stan as follows in `mixture_h.stan`. The hierarchical extension uses the Cholesky factorization for the group-level effects (as we did in 11.1.3).

```

data {
  int<lower = 1> N;
  vector[N] x;

```

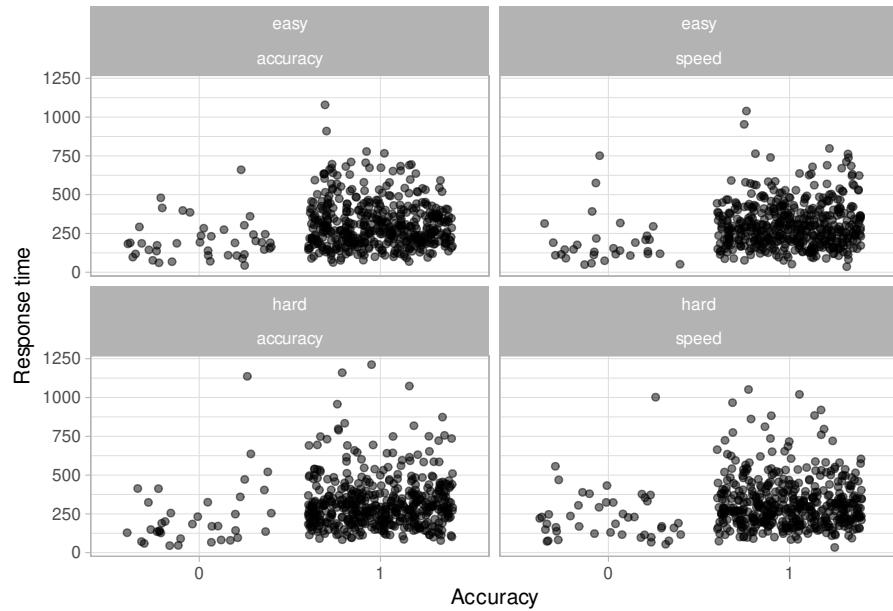


FIGURE 20.13: Distribution of response times conditional on the simulated accuracy and the experimental manipulations for the simulated hierarchical data (df_dots_simdata) that follows the fast-guess model.

```

vector[N] rt;
int acc[N];
vector[N] x2; //speed or accuracy emphasis
int<lower = 1> N_subj;
int<lower = 1, upper = N_subj> subj[N];
}

parameters {
  real alpha;
  real beta;
  real<lower = 0> sigma;
  real<upper = alpha> gamma; //guessing
  real<lower = 0> sigma2;
  real<lower = 0, upper = 1> p_correct;
  real<lower = 0, upper = 1> p_btask;
  real beta_task;
  vector<lower = 0>[3] tau_u;
}

```

```
matrix[3, N_subj] z_u;
cholesky_factor_corr[3] L_u;
}

transformed parameters {
    matrix[N_subj, 3] u;
    u = (diag_pre_multiply(tau_u, L_u) * z_u)';
}

model {
    // priors for the task component
    target += normal_lpdf(alpha | 6, 1);
    target += normal_lpdf(beta | 0, .1);
    target += normal_lpdf(sigma | .5, .2)
        - normal_lccdf(0 | .5, .2);
    // priors for the guessing component
    target += normal_lpdf(gamma | 6, 1) -
        normal_lcdf(alpha | 6, 1);
    target += normal_lpdf(sigma2 | .5, .2)
        - normal_lccdf(0 | .5, .2);
    target += normal_lpdf(tau_u | 0, .5)
        - 3* normal_lccdf(0 | 0, .5);
    target += normal_lpdf(beta_task | 0, 1);
    target += beta_lpdf(p_correct | 995, 5);
    target += beta_lpdf(p_btask | 8, 2);
    target += lkj_corr_cholesky_lpdf(L_u | 2);
    target += std_normal_lpdf(to_vector(z_u));

    for(n in 1:N){
        real lodds_task = logit(p_btask) + x2[n] * beta_task;
        target += log_sum_exp(log_inv_logit(lodds_task) +
            lognormal_lpdf(rt[n] | alpha + u[subj[n], 1] +
                x[n] * (beta + u[subj[n], 2]), sigma) +
            bernoulli_lpmf(acc[n] | p_correct),
            log1m_inv_logit(lodds_task) +
            lognormal_lpdf(rt[n] | gamma + u[subj[n], 3], sigma2) +
            bernoulli_lpmf(acc[n] | .5));
    }
}
```

```
generated quantities {
  corr_matrix[3] rho_u = L_u * L_u';
}
```

Save the model code and fit it to the simulated data:

```
ls_dots_simdata <- list(N = N,
                         rt = df_dots_simdata$rt,
                         x = df_dots_simdata$x,
                         x2 = df_dots_simdata$x2,
                         acc = df_dots_simdata$acc,
                         subj = df_dots_simdata$subj,
                         N_subj = N_subj)
mixture_h <- system.file("stan_models",
                          "mixture_h.stan",
                          package = "bcogsci")
fit_mix_h <- stan(file = mixture_h,
                   data = ls_dots_simdata,
                   iter = 3000,
                   control = list(adapt_delta = .9))
```

Print the posterior summary:

```
print(fit_mix_h, pars = c("alpha", "beta", "sigma", "gamma", "sigma2",
                           "p_correct", "p_btask", "beta_task", "tau_u",
                           "rho_u[1,2]", "rho_u[1,3]", "rho_u[2,3]"))
```

	mean	2.5%	97.5%	n_eff	Rhat
## alpha	5.73	5.65	5.82	821	1.00
## beta	0.05	0.00	0.09	6061	1.00
## sigma	0.40	0.38	0.42	8705	1.00
## gamma	5.16	4.93	5.38	2383	1.00
## sigma2	0.55	0.49	0.61	8497	1.00
## p_correct	1.00	0.99	1.00	8264	1.00
## p_btask	0.85	0.83	0.88	8341	1.00
## beta_task	0.07	-0.24	0.39	9331	1.00
## tau_u[1]	0.19	0.13	0.28	1549	1.00

```
## tau_u[2]    0.06  0.00  0.14  1499 1.00
## tau_u[3]    0.46  0.31  0.67  2519 1.00
## rho_u[1,2]  0.04 -0.60  0.64  7077 1.00
## rho_u[1,3]  0.04 -0.40  0.46  2717 1.00
## rho_u[2,3] -0.08 -0.71  0.58   615 1.01
```

We see that we can fit the hierarchical extension of our model to simulated data. Next we'll evaluate whether we can recover the true values of the parameters.

20.1.5.1 Recovery of the parameters

By “recovering” the true values of the parameters, we mean that the true values are somewhere inside the bulk of the posterior distribution of the model.

We use `mcmc_recover_hist` to compare the posterior distributions of the relevant parameters of the model with their true values in Figure 20.14.

```
df_fit_mix_h <- fit_mix_h %>% as.data.frame() %>%
  select(c("alpha", "beta", "sigma", "gamma", "sigma2",
         "p_correct", "p_btask", "beta_task", "tau_u[1]",
         "tau_u[2]", "tau_u[3]", "rho_u[1,2]", "rho_u[1,3]",
         "rho_u[2,3]"))
true_values <- c(alpha, beta, sigma, gamma, sigma2,
                  p_correct, p_btask, beta_task, tau_u[1],
                  tau_u[2], tau_u[3], rep(rho, 3))
mcmc_recover_hist(df_fit_mix_h, true_values)
```

The model seems to be underestimating the probability of being correct of the subjects (`p_correct`) and overestimating the probability of being engaged in the task (`p_btask`). However, the numerical differences are very small. We can be relatively certain that the model is not seriously misspecified. A more principled (and computationally demanding) approach uses simulation based calibration (SBC), which is discussed in Talts et al. (2018) and Schad, Betancourt, and Vasishth (2020).

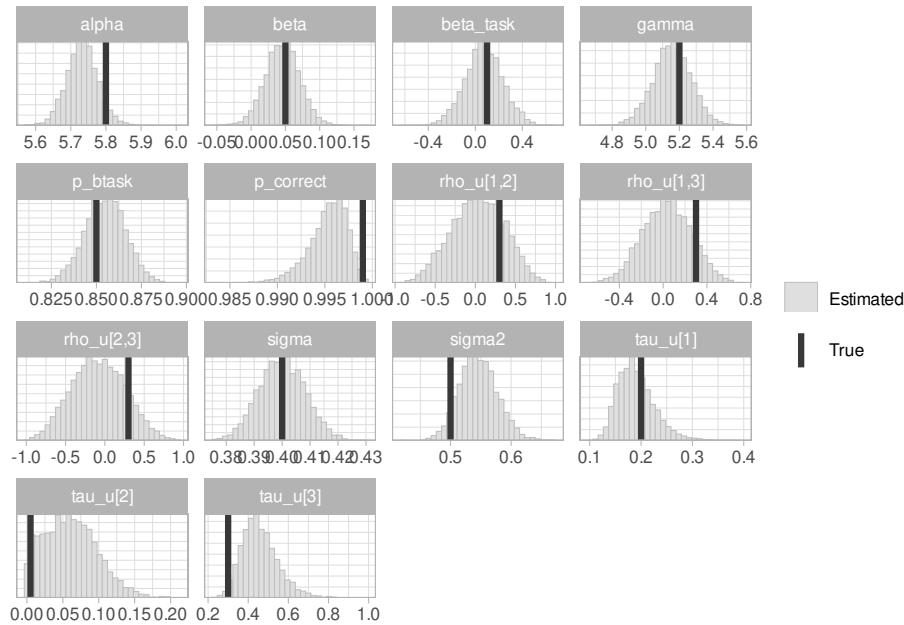


FIGURE 20.14: Posterior distributions of the main parameters of the mixture model `fit_mix_h` together with their true values.

20.1.5.2 Fitting the model to real data

After verifying that our model works as expected, we are ready to fit it to real data. We code the predictors x and x_2 as we did for the simulated data:

```
df_dots <- df_dots %>%
  mutate(x = if_else(diff == "easy", -.5, .5),
        x2 = if_else(emphasis == "accuracy", .5, -.5))
```

The main obstacle now is that fitting the entire data set takes around 12 hours! We'll sample 600 observations of each subject as follows:

```
df_dots_data_short <- df_dots %>%
  group_by(subj) %>%
  sample_n(600)
```

```
ls_dots_data_short <-
  list(N = nrow(df_dots_data_short),
       rt = df_dots_data_short$rt,
       x = df_dots_data_short$x,
       x2 = df_dots_data_short$x2,
       acc = df_dots_data_short$acc,
       subj = as.numeric(df_dots_data_short$subj),
       N_subj = length(unique(df_dots_data_short$subj)))
fit_mix_data <- stan(file = mixture_h,
                      data = ls_dots_data_short,
                      chains = 4,
                      iter = 2000,
                      control = list(adapt_delta = .9,
                                     max_treedepth = 12))
```

The model has not converged at all!

```
print(fit_mix_data,
      pars = c("alpha", "beta", "sigma", "gamma", "sigma2",
              "p_correct", "p_btask", "beta_task", "tau_u",
              "rho_u[1,2]", "rho_u[1,3]", "rho_u[2,3]"))
```

	mean	2.5%	97.5%	n_eff	Rhat
## alpha	6.36	6.28	6.47	3	1.55
## beta	0.11	0.07	0.17	2	2.44
## sigma	0.24	0.22	0.29	2	11.65
## gamma	6.24	6.07	6.35	2	2.48
## sigma2	0.38	0.21	0.44	2	18.19
## p_correct	0.98	0.93	1.00	2	11.74
## p_btask	0.75	0.68	0.92	2	8.77
## beta_task	2.00	0.78	5.71	2	10.99
## tau_u[1]	0.14	0.10	0.20	40	1.06
## tau_u[2]	0.05	0.03	0.08	16	1.08
## tau_u[3]	0.18	0.12	0.27	7	1.18
## rho_u[1,2]	0.46	0.00	0.82	23	1.06
## rho_u[1,3]	0.16	-0.28	0.55	518	1.02

```
## rho_u[2,3] 0.02 -0.46  0.52     13   1.11
```

The traceplots in Figure 20.15 show that the chains are not mixing at all. It seems that the posterior is multimodal, and there are at least two combinations of parameters that would fit the data equally well.

```
traceplot(fit_mix_data)
```

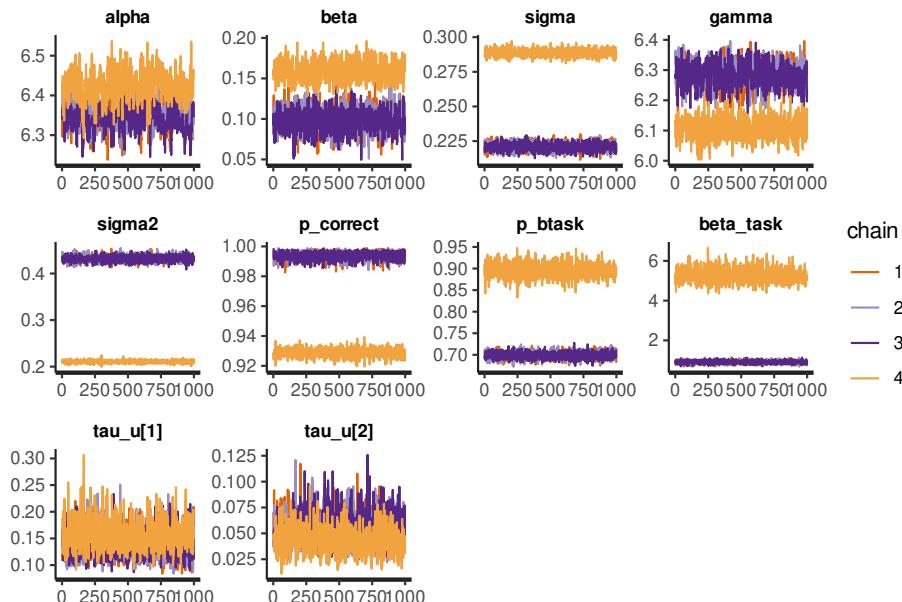


FIGURE 20.15: Traceplots from the hierarchical model (`mixture_h.stan`) fit to (a subset) of the real data. The traceplot shows clearly that the posterior has at least two modes.

What should we do now? It can be a good idea to back off and simplify the model. Once the simplified model converges, we can think about adding further complexity. The verbal description of our model says that the accuracy in the task-engaged mode should be close to 100%. To simplify the model, we'll assume that it's exactly 100%. This entails the following:

$$p_{correct} = 1 \quad (20.29)$$

We adapt our Stan code in `mixture_h2.stan` reflecting that `p_correct` is no

longer a parameter and it's now in a block called `transformed data`. There we assign to `p_correct` the value of 1.

```

data {
    int<lower = 1> N;
    vector[N] x;
    vector[N] rt;
    int acc[N];
    vector[N] x2; //speed or accuracy emphasis
    int<lower = 1> N_subj;
    int<lower = 1, upper = N_subj> subj[N];
}
transformed data {
    real p_correct = 1;
}
parameters {
    real alpha;
    real beta;
    real<lower = 0> sigma;
    real<upper = alpha> gamma; //guessing
    real<lower = 0> sigma2;
    real<lower = 0, upper = 1> p_btask;
    real beta_task;
    vector<lower = 0>[3] tau_u;
    matrix[3, N_subj] z_u;
    cholesky_factor_corr[3] L_u;
}
transformed parameters {
    matrix[N_subj, 3] u;
    u = (diag_pre_multiply(tau_u, L_u) * z_u)';
}
model {
    // priors for the task component
    target += normal_lpdf(alpha | 6, 1);
    target += normal_lpdf(beta | 0, .1);
    target += normal_lpdf(sigma | .5, .2)
        - normal_lccdf(0 | .5, .2);
    // priors for the guessing component
}
```

```

target += normal_lpdf(gamma | 6, 1) -
    normal_lcdf(alpha | 6, 1);
target += normal_lpdf(sigma2 | .5, .2)
    - normal_lccdf(0 | .5, .2);
target += normal_lpdf(tau_u | 0, .5)
    - 3* normal_lccdf(0 | 0, .5);
target += normal_lpdf(beta_task | 0, 1);
target += beta_lpdf(p_btask | 8, 2);
target += lkj_corr_cholesky_lpdf(L_u | 2);
target += std_normal_lpdf(to_vector(z_u));

for(n in 1:N){
    real lodds_task = logit(p_btask) + x2[n] * beta_task;
    target += log_sum_exp(log_inv_logit(lodds_task) +
        lognormal_lpdf(rt[n] | alpha + u[subj[n], 1] +
            x[n] * (beta + u[subj[n], 2]), sigma) +
        bernoulli_lpmf(acc[n] | p_correct),
        log1m_inv_logit(lodds_task) +
        lognormal_lpdf(rt[n] | gamma + u[subj[n], 3], sigma2) +
        bernoulli_lpmf(acc[n] | .5));
}
generated quantities {
    corr_matrix[3] rho_u = L_u * L_u';
}
```

Fit the model again to the same data:

```

fit_mixs_data <- stan(file = mixture_h,
                      data = ls_dots_data_short,
                      chains = 4,
                      iter = 2000,
                      control = list(adapt_delta = .9,
                                     max_treedepth = 12))
```

The model has now converged.

```
print(fit_mixs_data,
      pars = c("alpha", "beta", "sigma", "gamma", "sigma2",
              "p_btask", "beta_task", "tau_u",
              "rho_u[1,2]", "rho_u[1,3]", "rho_u[2,3]"))

##          mean 2.5% 97.5% n_eff Rhat
## alpha     6.34 6.29 6.40   667    1
## beta      0.10 0.07 0.12  1841    1
## sigma     0.22 0.21 0.23  4727    1
## gamma     6.29 6.21 6.35 1254    1
## sigma2    0.43 0.42 0.44 4386    1
## p_btask   0.69 0.68 0.70 4686    1
## beta_task 0.88 0.75 1.02 4608    1
## tau_u[1]  0.14 0.10 0.20  942    1
## tau_u[2]  0.05 0.03 0.08 1737    1
## tau_u[3]  0.19 0.13 0.27 1552    1
## rho_u[1,2] 0.40 -0.07 0.75 2194    1
## rho_u[1,3] 0.15 -0.28 0.53 1390    1
## rho_u[2,3] -0.03 -0.48 0.44 1071    1
```

The traceplots in Figure 20.16 show that this times the chains are mixing well.

What can we say about the fit of the model now?

Under the assumptions that we have made (e.g., there are two processing modes, response times are affected by the difficulty of the task in the task-engaged mode, accuracy is not affected by the difficulty of the task and is perfect at the task-engaged mode, etc.), we can look at the parameters and conclude the following:

- The instructions seemed to have a strong effect on the processing mode of the subjects (`beta_task` is relatively high), and in the expected direction (emphasis in accuracy led to a higher probability to be in the task engaged mode).
- The guessing mode seemed to be much noisier than the task-engaged mode (compare `sigma` with `sigma2`).

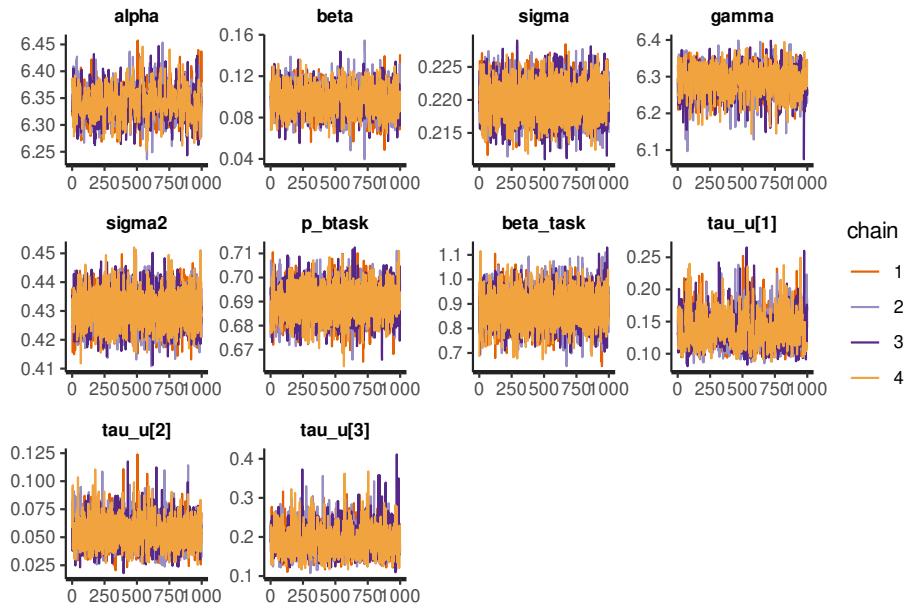


FIGURE 20.16: Traceplots from the simplified hierarchical model (`mixture_h.stan`, assuming that `p_correct = 1`) fit to (a subset) of the real data. The traceplot shows that chains are mixing well.

- Slow subjects seemed to show a stronger effect of the experimental manipulation (`rho_u1[1, 2]` is mostly positive).

If we want to know whether our model achieves descriptive adequacy, we need to look at the posterior predictive distributions of the model. However, by using posterior predictive checks, we won't be able to conclude that our model is not overfitting. Our success in fitting the fast-guess model to real data does not entail that the model is a good account of the data. It just means that it's flexible enough to fit the data. One further step could be to develop a competing model, and then comparing the performances of the models, using Bayes factors or cross validation.

20.1.5.3 Posterior predictive checks

For the posterior predictive checks, we can write the generated quantities block in a new file. The advantage is that we can generate as many observations as needed *after estimating the parameters*. There is no model block in the follow Stan program. We use the `gqs()` function in the `rstan` library,

which allows us to use the posterior draws from a previously fitted model to generate posterior predicted data.

Generate 500 simulated data sets as follows:

```
mixture_h_gen <- system.file("stan_models",
                               "mixture_h_gen.stan",
                               package = "bcogsci")
gen_model <- stan_model(mixture_h_gen)
draws_par <- as.matrix(fit_mixs_data)[1:500, , drop = FALSE]
gen_mix_data <- gqs(gen_model,
                      data = ls_dots_data_short,
                      draws = draws_par)
```

We first take a look at the general distribution of response times generated by the posterior predictive model and by our real data in Figure 20.17.

```
rt_pred <- extract(gen_mix_data)$rt_pred
ppc_dens_overlay(y = ls_dots_data_short$rt, yrep = rt_pred[1:100,]) +
  coord_cartesian(xlim = c(0, 2000))
```

We see that the distribution of the observed response times is narrower than the predictive distribution. We are generating response times that are more spread out than the real data.

Next we examine the effect of the experimental manipulation in Figure 20.18: The posterior predictive check reveals that the model underestimates the observed effect of the experimental manipulation: the observed difference between response times is well outside the bulk of the predictive distribution.

```
meanrt_diff <- function(rt){
  mean(rt[ls_dots_data_short$x == .5]) -
  mean(rt[ls_dots_data_short$x == -.5])
}
ppc_stat(ls_dots_data_short$rt,
```

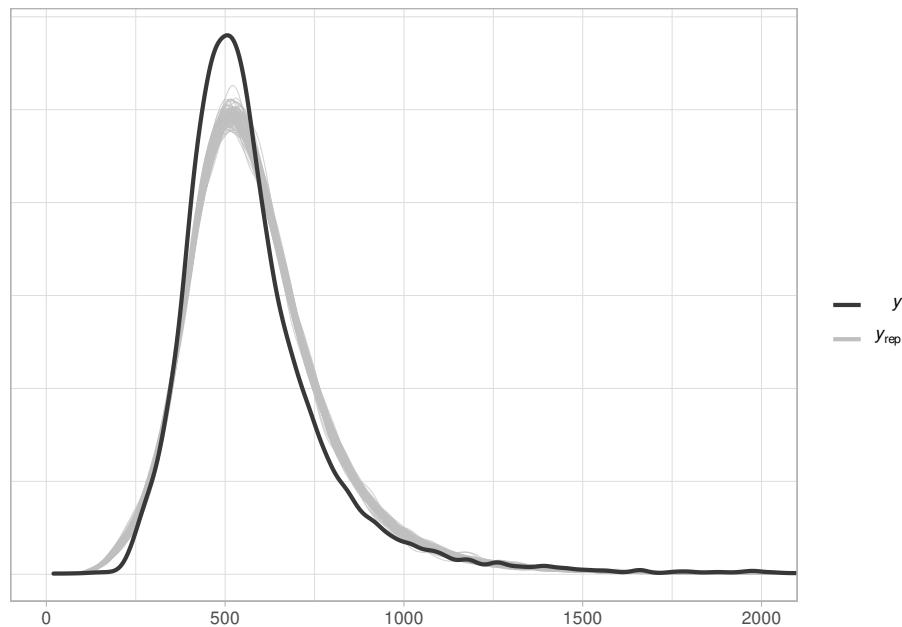


FIGURE 20.17: Posterior predictive distribution of the hierarchical fast-guess model (using `mixture_h_gen.stan`) in comparison with the observed response times.

```
    yrep = rt_pred,
    stat = meanrt_diff)
```

Another important posterior predictive check includes comparing the fit of the model using a quantile probability plot, which is presented in the next chapter.

We also look at some instances of the predictive distribution. Figure 20.19 shows a simulated data set in black overlaid onto the real observations in gray. As we noticed in Figure 20.17, the model is predicting less variability than what we find in the data, especially when the emphasis is on accuracy.

```
acc_pred <- extract(gen_mix_data)$acc_pred
df_dots_pred <-
```

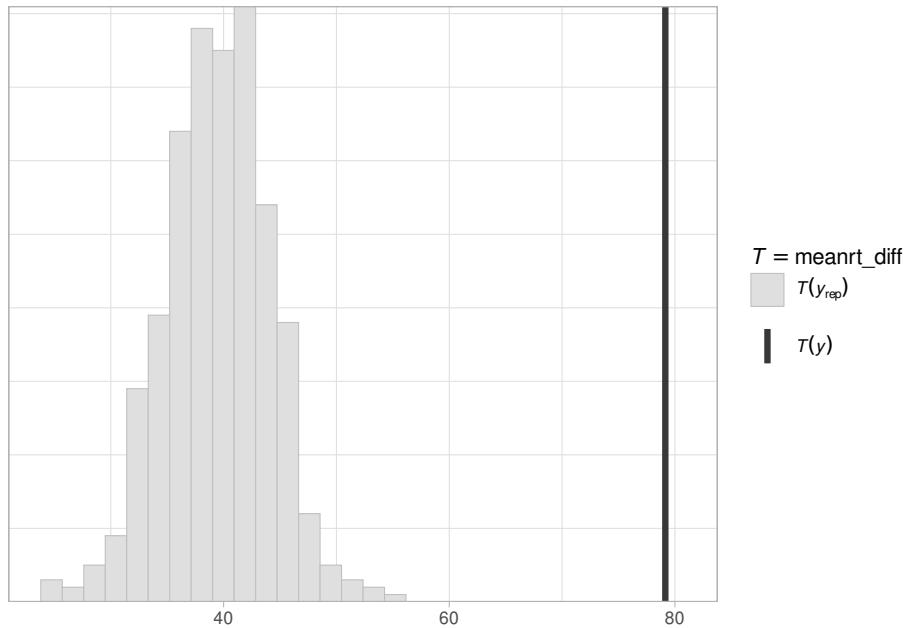


FIGURE 20.18: Posterior predictive distribution (using `mixture_h_gen.stan`) of the difference in response time due to the experimental manipulation. The vertical bar shows the observed difference in the data.

```
tibble(rt = ls_dots_data_short$rt,
       acc = ls_dots_data_short$acc,
       difficulty = ifelse(ls_dots_data_short$x == .5,
                           "hard", "easy"),
       emphasis = ifelse(ls_dots_data_short$x2 == -.5,
                         "speed", "accuracy"),
       acc_pred1 = acc_pred[1,],
       rt_pred1 = rt_pred[1,])
```

To conclude, the fast-guess model shows a relatively decent fit to the data and is able to account for the speed-accuracy trade-off. The model shows some inaccuracies that could lead to its revision and improvement. To what extent the inaccuracies are acceptable or not depends on (i) the empirical finding that we want to account for (for example, we can already

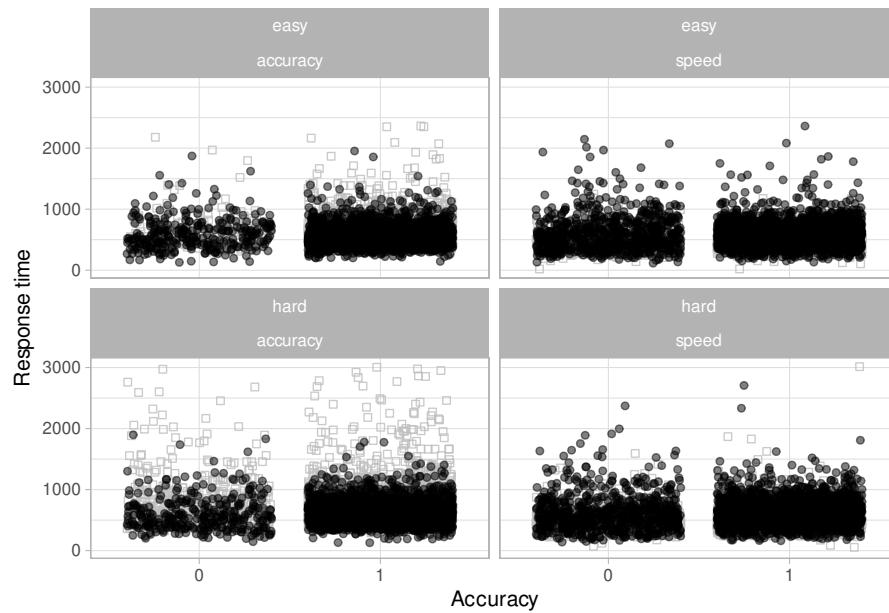


FIGURE 20.19: A simulated (posterior predictive) data set in black overlaid onto the observations in gray (based on `mixture_h_gen.stan`).

assume that the model will struggle to fit data sets that show slow errors); and (ii) its comparison with a competing account.

20.2 Summary

In this chapter, we learned to fit increasingly complex two-component mixture models using Stan, starting with a simple model and ending with a fully hierarchical model. We saw how to evaluate model fit using the usual prior and posterior predictive checks, and to investigate parameter recovery. Such mixture models are notoriously difficult to fit, but they have a lot of potential in cognitive science applications, especially in developing computational models of different kinds of cognitive processes.

20.3 Further reading

The reader interested in a deeper understanding of marginalization is referred to Pullin, Gurrin, and Vukcevic (2021). Betancourt discusses problems of identification in Bayesian mixture models in a case study (https://mc-stan.org/users/documentation/case-studies/identifying_mixture_models.html). An in depth treatment of the fast-guess model and other mixture models of response times is provided in Chapter 7 of Luce (1991).

20.4 Exercises

Exercise 20.1. Changes in the true values.

Change the true value of `p_correct` to .5 and .1, and generate data for the non-hierarchical model. Can you recover the value of this parameter. Perform posterior predictive checks.

Exercise 20.2. RTs in schizophrenic patients and control.

Response times for schizophrenic patients in a simple visual tracking experiment show more variability than for non-schizophrenic controls; see Figure 20.20. It has been argued that at least some of this extra variability arises from an attentional lapse that delays some responses. We'll use the data examined in Belin and Rubin (1990) (`df_schizophrenia` in the `bcogsci` package) analysis to investigate some potential models:

- M_1 . Both schizophrenic and controls show attentional lapses, but the lapses are more common in schizophrenics. Other than that there is no difference in the latent response times and the lapses of attention.
- M_2 . Only schizophrenic patients show attentional lapses. Other than that there is no difference in the latent response times.
- M_3 . There no (meaningful number of) lapses of attention in neither group.

1. Fit the three models.
2. Do posterior predictive checks for each model; can they account for the data?
3. Do model comparison (with Bayes factor and cross-validation)

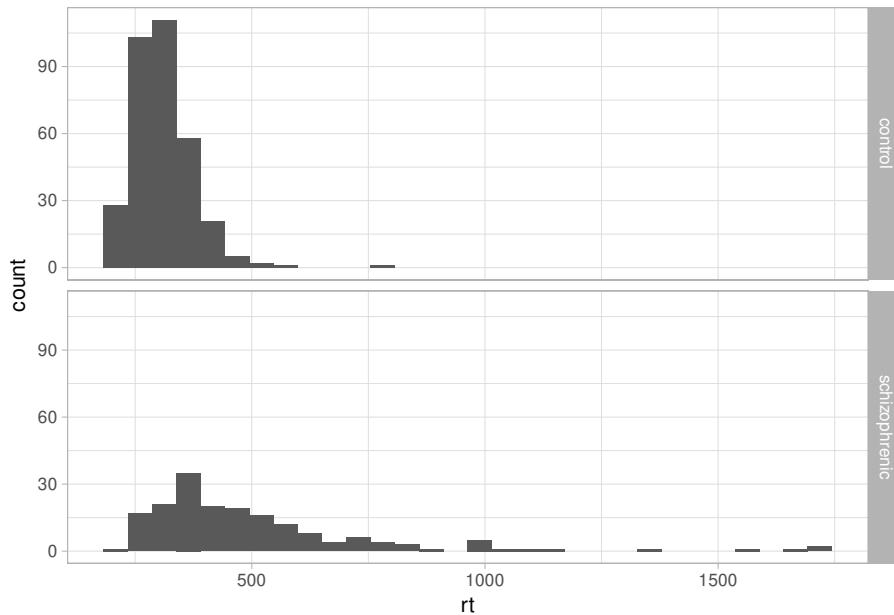


FIGURE 20.20: Distribution of response times for control and schizophrenic patients in `df_schizophrenia`.

Exercise 20.3. Advanced: Guessing bias in the model.

In the original model, it was assumed that subjects might have a bias (a preference) to one of the two answers when they were in the guessing mode. To fit this model we need to change the dependent variable and add more information, now we not only care if the participant answered correctly or not, but also which answer they gave (left or right).

- Implement a unique bias for all the subjects. Fit the new model to (a subset of) the data.
- Implement a hierarchical bias, that is there is a common bias, but every subject has its adjustment. Fit the new model to (a subset of) the data.

21

A simple accumulator model to account for choice response time

CHAPTER IN PREPARATION

As mentioned in chapter 20, the most popular class of cognitive-process models that can incorporate both response times and accuracy are sequential sampling models (for a review see Ratcliff et al. 2016). This class of models includes, among others, the drift diffusion model (Ratcliff 1978), the linear ballistic accumulator (Brown and Heathcote 2008), and the log-normal race model (Heathcote and Love 2012; Rouder et al. 2015), which we discuss in this chapter. Sequential sampling or evidence-accumulation models are based on the idea that decisions are made by gathering evidence from the environment (e.g., the computer screen in many experiments) until sufficient evidence is gathered and a threshold of evidence is reached.

The log-normal race model seems to be the simplest sequential sampling model that can account for the joint distribution of response times and response choice or accuracy (Heathcote and Love 2012; Rouder et al. 2015). This model belongs to the subclass of *race models*, where the evidence for each response grows gradually in time in separate racing accumulators until a threshold. A response is made when one of these accumulators first reaches the threshold, and wins the race against the other accumulators. This model is sometimes referred as deterministic (or non-stochastic, and ballistic), since the noise only affects the rate of accumulation of evidence before each race starts, but once the accumulator is ac-

cumulating evidence the rate is fixed. This means that a given accumulator can be faster or slower in different trials (or between choices) but its rate of accumulation will be fixed during a trial (or within choices). Brown and Heathcote (2005) claim that even though it is clear that a range of factors might cause within-choice noise, the behavioral effects might sometimes be small enough to neglect (this is in contrast to models, such as, the drift diffusion model where both types of noise are present). The two main advantages of the log-normal race model in comparison with other sequential sampling models are that (i) it is very simple, making it easy to extend hierarchically and presenting few convergence issues, and that (ii) it is straightforward to model as many choices as wanted. We choose to work with this specific model for pedagogical purposes: it's relatively easy to derive its likelihood from its basic assumptions. However, even though it's a "legitimate" cognitive model (see its connection with a cognitive architecture in Nicenboim and Vasishth 2018), the majority of the literature fits choice response times with the linear ballistic accumulator and/or the drift diffusion model which provide more flexibility to the modeler.

We turn next to exemplify how to fit the log-normal race model with a lexical decision task.

21.1 Modeling a lexical decision task

In a lexical decision task, a subject is presented with a string of letters in the screen and they need to decide whether the string is a word or a non-word; see Figure 21.1. We'll use a subset of 600 observations of each of 20 subjects from the data of the British Lexicon project (Keuleers et al. 2012) as presented in the object `df_blp` in the package `bcogsci`. In this data set, the lexicality of the string (word or non-word) is indicated in the column `lex`, and we are going to examine how word frequency indicated in the column `freq` (which shows the frequency per million according to the British National Corpus) affects the lexical decision task. For more details about the data set, see `?df_blp`.

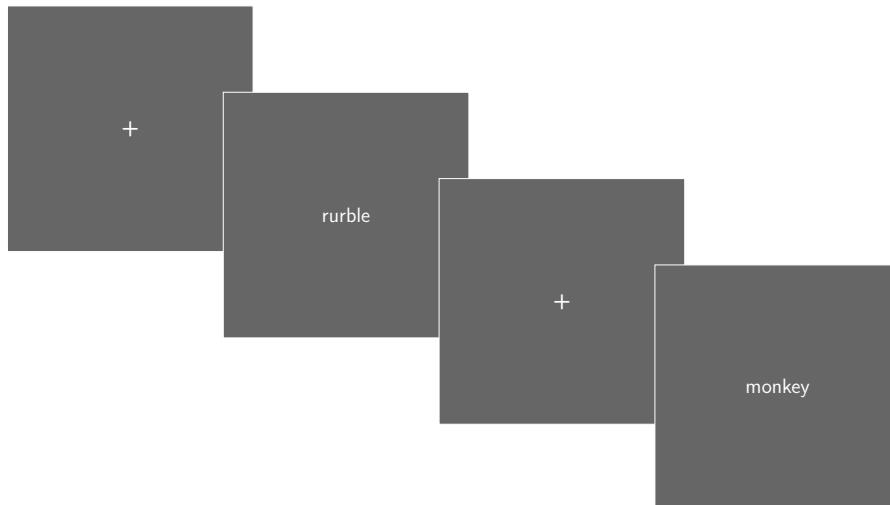


FIGURE 21.1: Two trials of a lexical decision task. For the first trial, `rurble`, the correct answer would be to press the key mapped to “non-word” response, for the second trial, `monkey`, the correct answer would be to press the key mapped to the “word” response.

```
data("df_blp")
df_blp
```

```
## # A tibble: 24,000 x 8
##   subj block lex     trial string    acc     rt freq
##   <dbl> <dbl> <chr>   <dbl> <fct>    <dbl> <dbl> <dbl>
## 1     1     57 non-word 28263 paybods    1     591     0
## 2     1     53 non-word 26414 lunned    1     621     0
## 3     1     49 non-word 24333 pertrax    1     575     0
## # ... with 23,997 more rows
```

The following chunk of code adds 0.01 (which corresponds to a word that appears only once in the corpus) to avoid word frequencies of zero and then log-transform the frequencies to compress their range of values (see Brysbaert, Mandera, and Keuleers 2018 for a more in-depth treatment of word frequencies). It also creates a new variable that sum-codes the lexicality of the each given string (either a word, 0.5, or a non-word, -0.5).

```
df_blp <- df_blp %>% mutate(lfreq = log(freq + 0.01),
                                c_lex = ifelse(lex == "word", 0.5, -0.5))
```

If one wants to study the effect of frequency on words, the “traditional” way to analyze these data would be to fit response times and choice data in two separate models on words, ignoring non-words. One model would be fit on the response times of correct responses, and a second model on the accuracy, as we do below.

To fit the response times model, subset the correct responses given to strings that are words.

```
df_blp_word_c <- df_blp %>%
  filter(acc == 1, lex == "word")
```

Fit a hierarchical model with a log-normal likelihood and log-transformed frequency as a predictor (using `brms` here).

```
fit_rt_word_c <- brm(rt ~ lfreq + (lfreq | subj),
                      data = df_blp_word_c,
                      family = lognormal,
                      prior = c(
                        prior(normal(6, 1.5), class = Intercept),
                        prior(normal(0, 1), class = sigma),
                        prior(normal(0, 1), class = sd),
                        prior(normal(0, 1), class = b),
                        prior(lkj(2), class = cor)
                      ), iter = 3000)
```

Show the estimate of the effect of log-frequency in log-ms scale.

```
posterior_summary(fit_rt_word_c, variable = "b_lfreq")
```

```
##           Estimate Est.Error    Q2.5    Q97.5
## b_lfreq   -0.0381   0.00268 -0.0434 -0.0329
```

To fit the accuracy model, subset the responses given to strings that are words.

```
df_blp_word <- df_blp %>%
  filter(lex == "word")
```

Fit a hierarchical model with a Bernoulli likelihood (and logit link) using log-transformed frequency as a predictor (using `brms`).

```
fit_acc_word <- brm(acc ~ lfreq + (lfreq | subj),
  data = df_blp_word,
  family = bernoulli(link = logit),
  prior = c(
    prior(normal(6, 1.5), class = Intercept),
    prior(normal(0, 1), class = sd),
    prior(normal(0, 1), class = b),
    prior(lkj(2), class = cor)
  ), iter = 3000)
```

Show the estimate of the effect of log-frequency in log-odds scale.

```
posterior_summary(fit_acc_word, variable = "b_lfreq")
```

```
##           Estimate Est.Error Q2.5 Q97.5
## b_lfreq     0.573    0.0257 0.524 0.624
```

For this specific data set, it does not matter whether we choose response times or accuracy as our dependent variable, since both of them yield results with similar interpretation: More frequent words are identified more easily, that is, with shorter reading times (and thus the negative estimate) and with better accuracy. However, it might be the case, that some data set shows opposite directions in response times and accuracy (e.g., a slowdown and increase of accuracy). Furthermore, we fit two models treating response times and accuracy as independent, while there is plenty of evidence that they are related (e.g., speed accuracy trade-off). Figure 21.2 shows clearly, for example, that as frequency increases cor-

rect answers are given faster and that most errors are for low frequency words.

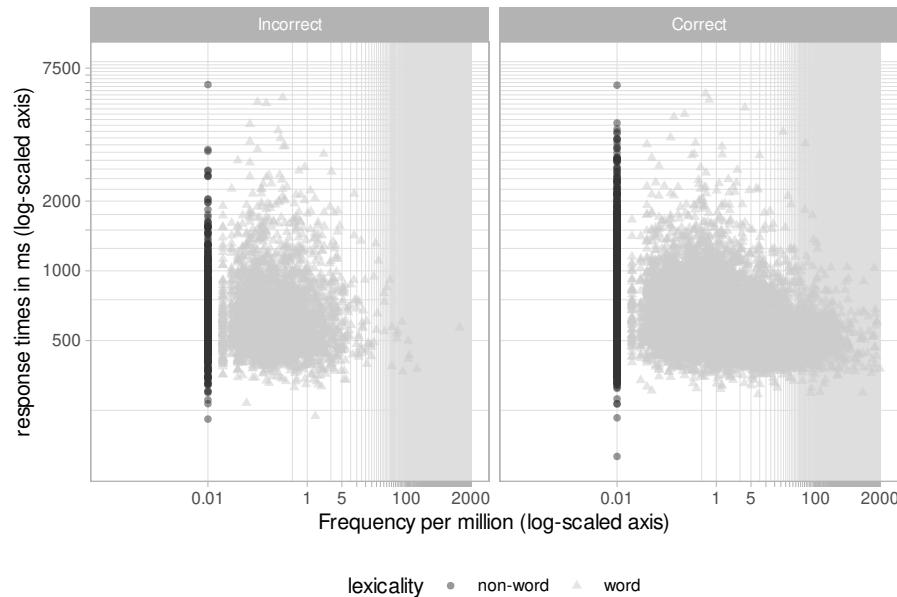


FIGURE 21.2: Distribution of response times for words and non-words, and correct and incorrect answers.

A powerful way to convey the relationship between response times and accuracy are *quantile probability plots* (Ratcliff and Tuerlinckx 2002; and closely related to the latency probability plots of Audley and Pike 1965).

A quantile probability plot shows quantiles of the response times distribution (typically .1, .3, .5, .7, and .9) for correct and incorrect responses on the y-axis against probabilities of correct and incorrect responses for experimental conditions on x-axis. The plot is built by first aggregating the data.

We do this by creating our custom function `qpf()` that takes as arguments a data set grouped by an experimental condition (e.g., words vs non-words, here by `lex`), and the quantiles we want to display (by default, 0.1, 0.3, 0.5, 0.7, 0.9). The function works as follows: First, calculate the desired quantiles of the response times for incorrect and correct responses by condition (here in `rt_q`). Second, calculate the proportion of incorrect and correct responses by condition (here in `p`); since we are going to need

this information for each quantile, we'll need to repeat it as many times as we have quantiles (here, five times). Last, include to which quantile each response time and response probability corresponds (here in `q`), and whether it corresponds to an incorrect or a correct response (here in `response`).

```
qpf <- function(df_grouped, quantiles = c(.1, .3, .5, .7, .9)) {
  df_grouped %>% summarize(
    rt_q = c(quantile(rt[acc == 0], quantiles),
               quantile(rt[acc == 1], quantiles)),
    p = c(rep(mean(acc == 0), length(quantiles)),
            rep(mean(acc == 1), length(quantiles))),
    q = rep(quantiles, 2),
    response = c(rep("incorrect", length(quantiles)),
                  rep("correct", length(quantiles))))
  )
  df_blp_lex_q <- df_blp %>%
    group_by(lex) %>%
    qpf()
```

The aggregated data will look as follows.

```
df_blp_lex_q %>% print(n = 10)
```

```
## # A tibble: 20 x 5
## # Groups:   lex [2]
##   lex      rt_q      p      q response
##   <chr>    <dbl>  <dbl> <dbl> <chr>
## 1 non-word 433.  0.0521  0.1 incorrect
## 2 non-word 521.  0.0521  0.3 incorrect
## 3 non-word 613   0.0521  0.5 incorrect
## 4 non-word 779.  0.0521  0.7 incorrect
## 5 non-word 1110  0.0521  0.9 incorrect
## 6 non-word 448   0.948   0.1 correct
## 7 non-word 513   0.948   0.3 correct
## 8 non-word 575   0.948   0.5 correct
## 9 non-word 666   0.948   0.7 correct
```

```
## 10 non-word 905 0.948 0.9 correct
## # ... with 10 more rows
```

We are ready to plot it, by joining the points that belong to the same quantiles with lines. Given that incorrect responses in most tasks occur in less than 50% of the trials and correct responses occur in a complementary distribution (i.e., in more than 50% of the trials), incorrect responses usually appear in the left half of the plot, and correct ones in the right half. The code that appears below produces this plot, which is shown in Figure 21.3.

```
ggplot(df_blp_lex_q, aes(x = p, y = rt_q)) +
  geom_vline(xintercept = .5, linetype = "dashed") +
  geom_point(aes(shape = lex)) +
  geom_line(aes(group = interaction(q, response))) +
  ylab("RT quantiles (ms)") +
  xlab("Response proportion") +
  scale_shape_discrete("Lexicality") +
  annotate("text", x = .40, y = 500, label = "incorrect") +
  annotate("text", x = .60, y = 500, label = "correct")
```

The vertical spread among the lines shows the shape of response times distribution. The lower quantile lines correspond to the left part of the response times distribution, and the higher quantiles to the right part of the distribution. Since the response times distribution is long tailed and right skewed, the higher quantiles are spread apart more than the lower quantiles.

We can also use a quantile probability plot to corroborate the observation that high frequency words are easier to recognize. To do that we subset the data to only words, and group the strings according to their “frequency group” (that is, to which quantile of frequency they belong). Whereas we previously aggregated over all the observations, ignoring subjects, we can also aggregate by subjects first, and then average the results. This will prevent that some idiosyncratic subjects will dominate in the plot. (We can also plot individual quantile probability plots by subject). Besides that, the code below follows the same steps as before, and we plot this in Figure 21.4.

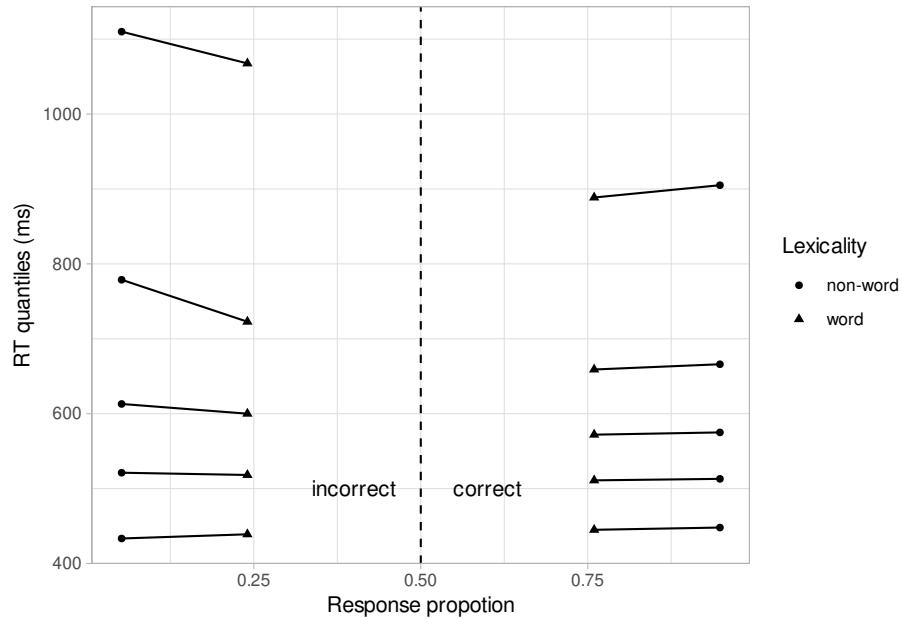


FIGURE 21.3: Quantile probability plot showing 0.1, 0.3, 0.5, 0.7, and 0.9 response times quantiles plotted against proportion of incorrect responses (left) and proportion of correct responses (right) for strings that are words and non-words.

```
df_blp_freq_q <- df_blp %>%
  # Subset only words
  filter(lex == "word") %>%
  # Create 5 word frequencies group
  mutate(freq_group =
    cut(lfreq,
        quantile(lfreq, c(0,.2,.4, .6, .8, 1)),
        include.lowest = TRUE,
        labels =
        c("0-.2", ".2-.4", ".4-.6", ".6-.8", ".8-1")))
  ) %>%
  # Group by condition and subject
  group_by(freq_group, subj) %>%
  # Apply the quantile probability function
```

```

qpf() %>%
  # Group again removing subject
  group_by(freq_group, q, response) %>%
  # Get averages of all the quantities
  summarize(rt_q = mean(rt_q),
            p = mean(p))
# Plot
ggplot(df_blp_freq_q, aes(x = p, y = rt_q)) +
  geom_point(shape = 4) +
  geom_text(
    data = df_blp_freq_q %>%
      filter(q == .1),
    aes(label = freq_group), nudge_y = 12) +
  geom_line(aes(group = interaction(q, response))) +
  ylab("RT quantiles (ms)") +
  xlab("Response proportion") +
  annotate("text", x = .40, y = 900, label = "incorrect") +
  annotate("text", x = .60, y = 900, label = "correct")


```

Warning: Removed 5 rows containing missing values (geom_point).

Warning: Removed 1 rows containing missing values (geom_text).

Warning: Removed 5 row(s) containing missing values (geom_path).

So far, we have seen several ways to describe the data by representing them graphically, next we turn to modeling the data.

21.1.1 Modeling the lexical decision task with the log-normal race model

We'll use the log-normal race model to examine the effect of word frequency in both response times and choice (word vs. non-word) in the lexical decision task presented before. In this example, we limit to two choices (even though this model can in principle fit more than two). When we have a task with two choices, there are two ways to account for the data: We fit the response times and the accuracy (i.e., accuracy coding: correct vs. incorrect), or we fit the response times and actual responses (i.e., stimulus

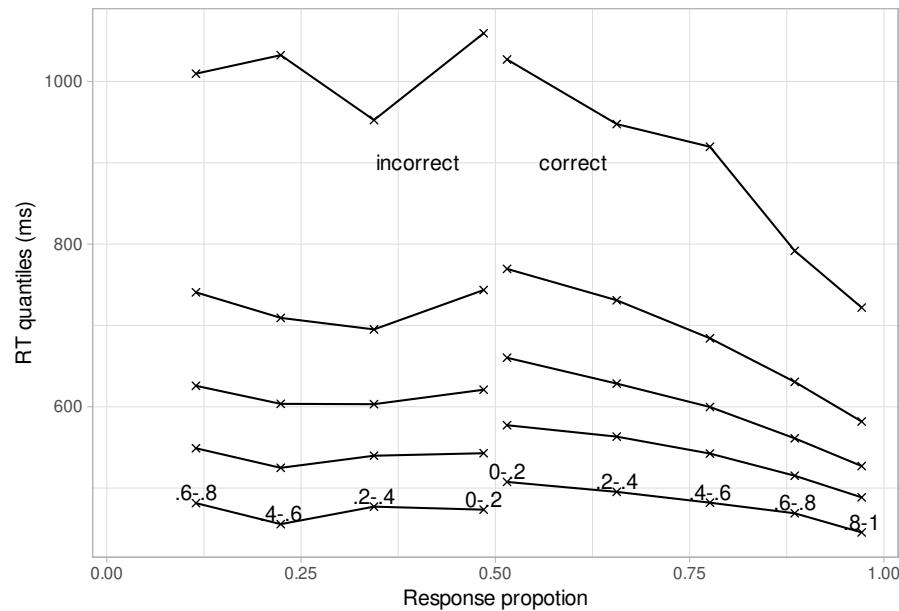


FIGURE 21.4: Quantile probability plot showing 0.1, 0.3, 0.5, 0.7, and 0.9 response times quantiles plotted against proportion of incorrect responses (left) and proportion of correct responses (right) for only words of different frequency. Word frequency are grouped according to quantiles.

coding: in this case word vs. non-word). In this example, we will use the stimulus-coding approach.

We add a new column that incorporates the actual choice made (as `word` vs `non-word` in `choice` and as `1` vs `2` in `nchoice`) with the following chunk of code:

```
df_blp <- df_blp %>%
  mutate(choice = ifelse((lex == "word" &
    acc == 1) | (lex == "non-word" &
    acc == 0), "word", "non-word"),
  nchoice = ifelse(choice == "word", 1, 2))
```

We start modeling the data by thinking about the behavior of one syn-

thetic subject. This subject accumulates evidence for the response, “word” in one accumulator, and for “non-word” in another independent accumulator, simultaneously. Unlike other sequential sampling models, an increase of evidence for one choice doesn’t necessarily reduce the evidence for the other choices. Rouder et al. (2015) points out that it might seem odd to consider that we accumulate evidence for a non-word in the same manner as we accumulate for a word, since non-words may be conceptualized as the absence of a word. However, they stress that this approach is closely related to novelty detection, where the salience of stimuli never experienced seems to indicate that novelty is psychologically represented as more than the absence of familiarity. Still notions of words and non-word evidence accumulation are indeed controversial (see Dufau, Grainger, and Ziegler 2012). The alternative approach of fitting accuracy rather than stimuli discussed before doesn’t really circumvent the problem. This is because when the correct answer is word, we assume that the “correct” accumulator accumulates evidence for word, and the incorrect one for non-word, and the other way around when the correct answer is non-word.

21.1.2 A generative model for a race between accumulators

To build a generative model of the task based on the log-normal race model, we start by spelling out our assumptions. In a race of accumulators model, we assume that the time, T , it takes for each accumulator of evidence to reach D (the distance to the threshold), is simply defined by

$$T = D/V \quad (21.1)$$

where the denominator, V , is the rate (velocity, sometimes also called drift rate) of evidence accumulation.

The log-normal race model assumes that the rate in each trial is sampled from a log-normal distribution:

$$V \sim LogNormal(\mu_v, \sigma_v) \quad (21.2)$$

A log-normal distribution is partly justified by the work from Ulrich and

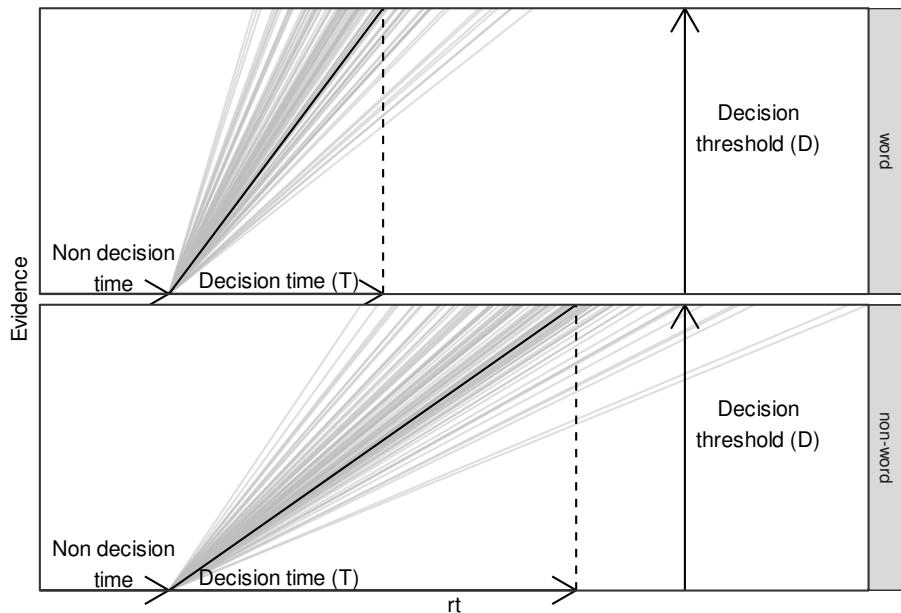


FIGURE 21.5: A schematic illustration of the log-normal race model for the lexical-decision task with a word stimulus. A larger rate of accumulation (V) leads to a larger angle.

Miller (1993) (and see Box 4.3), and we will see later that it is very mathematically convenient.

For simplicity, we assume that the threshold D is kept constant. This might not be a good assumption, when we know that participants change their threshold depending on speed or accuracy incentives (which were absent in this experiment), or during the experiment due to, for example, fatigue, or if we think that there might be random fluctuations in this threshold. We'll discuss what happens if we relax this assumption at a later stage. We assume that the location of the distribution of rates of accumulation of evidence for word, μ_w , for the trial n is a function of the lexicality of the string presented (only a word will increase this rate of accumulation and not a non-word), frequency (i.e., high frequency words might be easier to identify increasing the rate more than low frequency word), and bias (i.e., a subject might have a tendency to answer word rather than non-word or vice-versa regardless of the stimuli). We model this with a

linear regression over μ_w , with parameters that represent the bias, α_w , the effect of lexicality, β_{lex_w} , and effect of frequency β_{lfreq_w} .

$$\mu_{w,n} = \alpha_w + lex_n \cdot \beta_{lex_w} + lfreq_n \cdot \beta_{lfreq_w} \quad (21.3)$$

The location for the rate of accumulation of evidence for the non-word accumulator is defined similarly:

$$\mu_{nw,n} = \alpha_{nw} + lex_n \cdot \beta_{lex_{nw}} + lfreq_n \cdot \beta_{lfreq_{nw}} \quad (21.4)$$

Thus the rates are generated as follows:

$$\begin{aligned} V_{w,n} &\sim LogNormal(\mu_{w,n}, \sigma) \\ V_{nw,n} &\sim LogNormal(\mu_{nw,n}, \sigma) \end{aligned} \quad (21.5)$$

The accumulators reach the threshold in times:

$$\begin{aligned} T_{w,n} &= D/V_{w,n} \\ T_{nw,n} &= D/V_{nw,n} \end{aligned} \quad (21.6)$$

The choice for trial n corresponds to the accumulator with the shortest time for that trial,

$$choice_n = \begin{cases} word, & \text{if } T_{w,n} < T_{nw,n} \\ non-word, & \text{otherwise} \end{cases} \quad (21.7)$$

and the decision for trial n is made in time

$$T_n = \min(T_{w,n}, T_{nw,n}) \quad (21.8)$$

We also need to take into account that not all the time spent in the task involves making the decision: Time is spent fixating the gaze on the screen, pressing a button, etc. We'll add a shift to the distribution, representing the minimum amount of time that a subject needs for all the peripheral processes that happened before and after the decision (see also Rouder 2005). We represent this with T_{nd} . Although some variation in the non-decision time is highly likely, we use a constant as an approximation that

will be reasonable if its variation is small relative to the variation associated with the decision time (Heathcote and Love 2012).

$$rt_n = T_{nd} + T_n \quad (21.9)$$

The following chunk of code sets true values to the parameters and translates the previous equations to R. The true values are relatively arbitrary and were decided by trial and error until a relatively realistic distribution of response times was obtained. Considering that this is only one subject (unlike what was shown in previous figures), Figure ?? looks relatively fine. (One can also inspect the quantile probability plots of individual subjects in the real data).

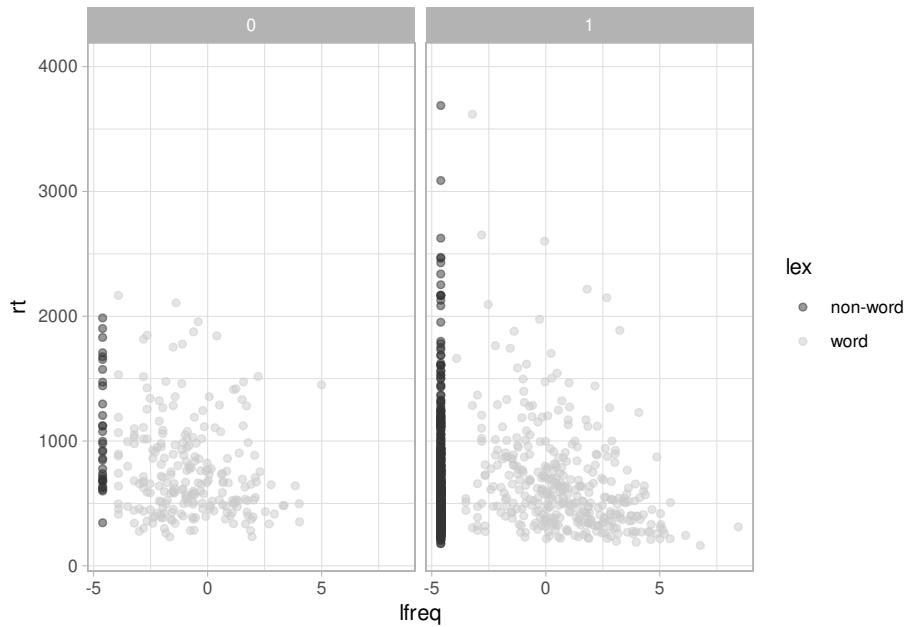
```
# generate always the same data:
set.seed(123)
df_blp_1subj <- df_blp %>%
  filter(subj == 1)
# True values:
D <- 1800
alpha_w <- .8
beta_wlex <- .5
beta_wlfreq <- .2
alpha_nw <- 1
beta_nwlex <- -.5
beta_nwlfreq <- -.05
sigma <- .8
T_nd <- 150
N <- nrow(df_blp_1subj)
mu_w <- alpha_w + df_blp_1subj$lfreq * beta_wlfreq +
  df_blp_1subj$c_lex * beta_wlex
mu_nw <- alpha_nw + df_blp_1subj$lfreq * beta_nwlfreq +
  df_blp_1subj$c_lex * beta_nwlex
# Rate of accumulation
V_w <- rlnorm(N, mu_w, sigma)
V_nw <- rlnorm(N, mu_nw, sigma)
T_w <- D / V_w
T_nw <- D / V_nw
```

```

# pairwise minimum
T_winner <- pmin(T_w, T_nw)
# choice
accumulator_winner <- ifelse(T_w == pmin(T_w, T_nw),
                               "word",
                               "non-word")
df_blp1_sim <- df_blp_1subj %>%
  mutate(rt = T_nd + T_winner,
        choice = accumulator_winner,
        nchoice = ifelse(choice == "word", 1, 2)) %>%
  mutate(acc = ifelse(lex == choice, 1, 0))

```

Coordinate system already present. Adding new coordinate system, which will replace the existing one.



21.1.3 Fitting the log-normal race model

A first issue that we face when we attempt to fit the log-normal race model, is that we need to fit its likelihood to a ratio of the random variables D and V , that is, we need a ratio or quotient distribution function. While for arbitrary distributions this requires solving (sometimes extremely complex)

integrals (see, for example, Nelson 1981), there are two situations that are compatible with our assumptions and are mathematically simple:

- (1) If we assume that D is a constant k , then $T = k/V$, and

$$\log(T) = \log(k/V) = \log(k) - \log(V) \quad (21.10)$$

Since V is log-normally distributed, $\log(V) \sim \text{Normal}(\mu_v, \sigma_v)$, and $\log(k)$ is a constant:

$$\begin{aligned} \log(T) &\sim \text{Normal}(\log(k) - \mu_v, \sigma_v) \\ T &\sim \text{LogNormal}(\log(k) - \mu_v, \sigma_v) \end{aligned} \quad (21.11)$$

- (2) A log-normally distributed time is not uniquely predicted by assuming that distance is a constant. It also follows if distance is also a log-normally distributed variable: If we assume that $D \sim \text{LogNormal}(\mu_d, \sigma_d)$ then T is the ratio of two random variables D/V , and

$$\log(T) = \log(D/V) = \log(D) - \log(V) \quad (21.12)$$

We have a difference of normally distributed random variables, then

$$\begin{aligned} \log(T) &\sim \text{Normal}(\mu_d - \mu_v, \sqrt{\sigma_d^2 + \sigma_v^2}) \\ T &\sim \text{LogNormal}(\mu_d - \mu_v, \sqrt{\sigma_d^2 + \sigma_v^2}) \end{aligned} \quad (21.13)$$

From Equations (21.11) and (21.13), it should be clear that the threshold and accumulation rate cannot be disentangled: a manipulation that affects the rate or the decision threshold will affect the location of the distribution in the same way. That is, the lognormal race model lacks identifiable decision thresholds (also see Rouder et al. 2015). Another important observation is that T won't have a log-normal distribution when D has any other distributional form.

Following Rouder et al. (2015), we assume that the noise parameter is the same for each the accumulator, since this means that contrasts between

finishing time distributions are captured completely by contrasts of the locations of the log-normal distributions.

In each trial n , with an accumulator for words w and one for non-words nw , we can model the time it takes for each accumulator to get to the threshold D in the following way. For the word accumulator,

$$\begin{aligned}\mu'_{w,n} &= \mu_{d_w} - \mu_{w,n} \\ \mu'_{w,n} &= \mu_{d_w} - (\alpha_w + lex_n \cdot \beta_{lex_w} + lfreq_n \cdot \beta_{lfreq_w}) \\ \mu'_{w,n} &= (\mu_{d_w} - \alpha_w) - lex_n \cdot \beta_{lex_w} - lfreq_n \cdot \beta_{lfreq_w} \\ \mu'_{w,n} &= \alpha'_w - lex_n \cdot \beta_{lex_w} - lfreq_n \cdot \beta_{lfreq_w} \\ T_{w,n} &\sim LogNormal(\mu'_{w,n}, \sigma)\end{aligned}\tag{21.14}$$

The parameter α' absorbs the location of the threshold distribution minus the intercept of the rate distribution, and represents a bias. As α'_w gets smaller, the accumulator will be more likely to reach the threshold first all things being equal, biasing the responses to word.

Similarly, for the non-word accumulator,

$$\begin{aligned}\mu'_{nw,n} &= \alpha'_{nw} - lex_n \cdot \beta_{lex_{nw}} - lfreq_n \cdot \beta_{lfreq_{nw}} \\ T_{nw,n} &\sim LogNormal(\mu'_{nw,n}, \sigma)\end{aligned}\tag{21.15}$$

The only observed time is the one associated with the winner accumulator, the response selected s , which corresponds to the fastest accumulator:

$$T_{accum=s,n} \sim LogNormal(\mu_{accum=s,n}, \sigma)\tag{21.16}$$

If we only fit the observed finishing times of the accumulators, we're always ignoring that in a given trial the accumulator that lost was slower than the accumulator for which we have the latency, this means that we underestimate the time it takes to reach the threshold and we overestimate the rate of accumulation of both accumulators. We can treat this as a problem of *censored data*, where for each trial slower observations are censored.

Since the potential decision time for the accumulator that wasn't selected

is for sure longer than the one of the winner accumulator, we obtain the likelihood for each unobserved time integrating all the possible decision times that the accumulator could have, that is from the time it took for the winner accumulator to reach the threshold to infinitely large decision times:

$$P(T_{\text{accum} \neq s,n}) = \int_{T_{\text{accum}=s,n}}^{\infty} \text{LogNormal}(T | \mu_{a=s,n}, \sigma) dT \quad (21.17)$$

This integral is the complement of the CDF of the log-normal distribution evaluated at $T_{\text{accum}=s,n}$.

$$P(T_{\text{accum} \neq s,n}) = 1 - \text{LogNormal_CDF}(T_{\text{accum}=s,n} | \mu_{\text{accum}=s,n}, \sigma) \quad (21.18)$$

As mentioned before, our dependent variable is response times, rt , the sum of the decision times, T , and the non decision times, T_{nd} . This requires a change of variable in our model, $T_n = rt_n - T_{nd}$, since rt but not T is available as data. In this case, however, no Jacobian adjustment is necessary, since adjusting the likelihood is equal to multiplying the likelihood by one (or to sum zero to the log likelihood). This is because $|d \frac{rt_n - T_0}{drt_n}| = 1$; for more details see section 12.1 in chapter 12.

To sum up, our model will look as follows:

$$\begin{aligned} T_n &= rt_n - T_{nd} \\ \mu'_{w,n} &= \alpha'_w - lex_n \cdot \beta_{lex_w} - lfreq_n \cdot \beta_{lfreq_w} \\ \mu'_{nw,n} &= \alpha'_{nw} - lex_n \cdot \beta_{lex_{nw}} - lfreq_n \cdot \beta_{lfreq_{nw}} \\ T_n &\sim \begin{cases} \text{LogNormal}(\mu'_{w,n}, \sigma) & \text{if } choice = \text{word} \\ \text{LogNormal}(\mu'_{nw,n}, \sigma) & \text{otherwise} \end{cases} \\ T_{censored,n} &= rt_{censored,n} - T_{nd} \\ T_{censored,n} &\sim \begin{cases} \text{LogNormal}(\mu'_{nw,n}, \sigma) & \text{if } choice = \text{word} \\ \text{LogNormal}(\mu'_{w,n}, \sigma) & \text{otherwise} \end{cases} \end{aligned} \quad (21.19)$$

Rather than trying to estimate all the censored observations, we integrate them out:

$$P(T_{censored}) = \begin{cases} 1 - \text{LogNormal_CDF}(T_n | \mu'_{w,n}, \sigma) & \text{if } choice = \text{word} \\ 1 - \text{LogNormal_CDF}(T_n | \mu'_{nw,n}, \sigma) & \text{otherwise} \end{cases} \quad (21.20)$$

We need priors for all the parameters. These are in log-scale as the mixture model in chapter 20. An added complication here is the prior for the non-decision time, T_{nd} : we need to make sure that it's not negative and also that it's smaller than the shortest observed response time. This is because the decision time for each observation, T_n should be strictly positive:

$$\begin{aligned} T_n &> 0 \\ T_{nd} &> 0 \\ T_n &= rt_n - T_{nd} \end{aligned} \quad (21.21)$$

We manipulate the last two expressions together:

$$\begin{aligned} rt_n - T_{nd} &> 0 \\ rt_n &> T_{nd} \\ \min(rt_n) &> T_{nd} \end{aligned} \quad (21.22)$$

We thus truncate the prior of T_{nd} in zero and in $\min(rt)$. Given the time it takes to fixate the gaze on the screen and a minimal motor response time, centering the prior in 150 ms seems reasonable. One should use prior predictive checks to verify that the order of magnitude of all the priors is appropriate. We skip this step here and we present the priors below.

$$\begin{aligned} \boldsymbol{\alpha} &\sim \text{Normal}(6, 1) \\ \boldsymbol{\beta} &\sim \text{Normal}(0, .5) \\ \sigma &\sim \text{Normal}_+(.5, .2) \\ T_{nd} &\sim \text{Normal}(150, 100) \text{ with } 0 < T_{nd} < \min(rt_n) \end{aligned} \quad (21.23)$$

where $\boldsymbol{\alpha}$ is a vector of $\{\alpha'_n, \alpha'_{nw}\}$, and $\boldsymbol{\beta}$ is a vector of all the β used in the distributions.

To translate the model into Stan we need a normal distribution truncated at 0 and $\min(rt)$ for the prior of T_{nd} . This means dividing the original distribution by the difference of the CDFs evaluated at these two points; see

Box 4.1. In log-space, this is a difference between the original distribution, but log-transformed and the logarithm of the difference of the CDFs. The function `log_diff_exp` is a more stable version of this last operation.

```
target += normal_lpdf(T_nd | 150, 100)
    - log_diff_exp(normal_lcdf(min(rt) | 150, 100),
                    normal_lcdf(0 | 150, 100));
```

We implement the likelihood of each joint observation of response time and choice with an if-else clause that calls the likelihood of the accumulator that corresponds to the choice selected in the trial n , and the complement CDF for the accumulator that was not selected:

```
if(nchoice[n] == 1)
    target += lognormal_lpdf(T[n] | alpha[1] -
        c_lex[n] * beta[1] -
        lfreq[n] * beta[2], sigma) +
    lognormal_lccdf(T[n] | alpha[2] -
        c_lex[n] * beta[3] -
        lfreq[n] * beta[4], sigma);
else
    target += lognormal_lpdf(T[n] | alpha[2] -
        c_lex[n] * beta[3] -
        lfreq[n] * beta[4], sigma) +
    lognormal_lccdf(T[n] | alpha[1] -
        c_lex[n] * beta[1] -
        lfreq[n] * beta[2], sigma);
}
```

The complete Stan code for this model is shown below as `lnrace.stan`.

```
data {
    int<lower = 1> N;
    vector[N] lfreq;
    vector[N] c_lex;
    vector[N] rt;
    int nchoice[N];
}
parameters {
    real alpha[2];
```

```

real beta[4];
real<lower = 0> sigma;
real<lower = 0, upper = min(rt)> T_nd;
}
model {
vector[N] T = rt - T_nd;
// priors for the task component
target += normal_lpdf(alpha | 6, 1);
target += normal_lpdf(beta | 0, .5);
target += normal_lpdf(sigma | .5, .2)
    - normal_lccdf(0 | .5, .2);
target += normal_lpdf(T_nd | 150, 100)
    - log_diff_exp(normal_lcdf(min(rt) | 150, 100),
                    normal_lcdf(0 | 150, 100));
for(n in 1:N){
    if(nchoice[n] == 1)
        target += lognormal_lpdf(T[n] | alpha[1] -
            c_lex[n] * beta[1] -
            lfreq[n] * beta[2], sigma) +
            lognormal_lccdf(T[n] | alpha[2] -
                c_lex[n] * beta[3] -
                lfreq[n] * beta[4], sigma);
    else
        target += lognormal_lpdf(T[n] | alpha[2] -
            c_lex[n] * beta[3] -
            lfreq[n] * beta[4], sigma) +
            lognormal_lccdf(T[n] | alpha[1] -
                c_lex[n] * beta[1] -
                lfreq[n] * beta[2], sigma);
}
}

```

Store the data in a list and fit the model. Some warnings might appear during the warm-up, but we can ignore them, since they no longer appear afterwards, and all the convergence checks look fine (omitted here).

```

lnrace <- system.file("stan_models",
                         "lnrace.stan",
                         package = "bcogsci")
ls_blp1_sim <- list(N = nrow(df_blp1_sim),
                      rt = df_blp1_sim$rt,
                      nchoice = df_blp1_sim$nchoice,
                      c_lex = df_blp1_sim$c_lex,
                      lfreq = df_blp1_sim$lfreq)
fit_blp1_sim <- stan(lnrace,
                       data = ls_blp1_sim)

```

Print the parameters values.

```
print(fit_blp1_sim, pars = c("alpha", "beta", "T_nd", "sigma"))
```

	mean	2.5%	97.5%	n_eff	Rhat
## alpha[1]	6.66	6.55	6.77	2262	1
## alpha[2]	6.53	6.41	6.65	2118	1
## beta[1]	0.40	0.18	0.64	2325	1
## beta[2]	0.21	0.17	0.24	2216	1
## beta[3]	-0.52	-0.72	-0.33	2109	1
## beta[4]	-0.05	-0.10	-0.01	2028	1
## T_nd	142.41	126.22	155.39	2274	1
## sigma	0.78	0.73	0.83	2442	1

As we did in previous chapters, we use `mcmc_recover_hist` to compare the posterior distributions of the relevant parameters of the model with their true values in Figure 21.6. First, however, we need to re-parametrize the true values, since D cannot be known, and we don't fit V , but rather D/V , with V log-normally distributed. Then we obtain an estimate of α' , rather than α , such that $\alpha' = \log(mu_d) - \alpha$.

```

true_values <- c(log(D) - alpha_w,
                     log(D) - alpha_nw,
                     beta_wlex,
                     beta_wlfreq,

```

```

    beta_nwlex,
    beta_nwlfreq,
    sigma,
    T_nd)

estimates <- as.data.frame(fit_blp1_sim) %>%
  select(-lp__)
mcmc_recover_hist(estimates, true_values)

```

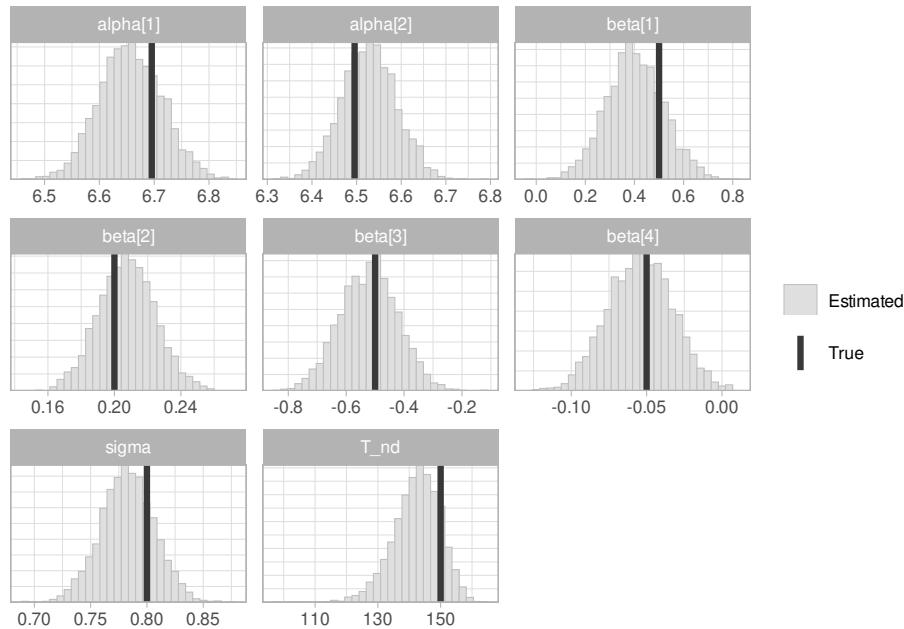


FIGURE 21.6: Posterior distributions of the main parameters of the log-normal race model `fit_blp1_sim` together with their true values.

Before moving on to a more complex version of this model, it's worth to spend some time making the code more modular, we encapsulate the likelihood of the lognormal race model by writting it as a function. The function has four arguments, the decision time, τ , the choice, `nchoice` (this will only work with two choices, 1, and 2), an array of locations `mu` (which again we implicitly assume that has two elements), and a common scale `sigma`.

```
functions {
```

```

real lognormal_race2_lpdf(real T, int nchoice, real[] mu, real sigma){
    real lpdf;
    if(nchoice == 1)
        lpdf = lognormal_lpdf(T | mu[1] , sigma) +
            lognormal_lccdf(T | mu[2], sigma);
    else
        lpdf = lognormal_lpdf(T | mu[2], sigma) +
            lognormal_lccdf(T | mu[1], sigma);
    return lpdf;
}
}

```

Now for each iteration n of the original for loop, we generate an auxiliary variable T which contains the decision time for the current trial, and μ as an array of size two that contains all the parameters that affect the location at each trial. This will allow us to use our new function as follows in the `transformed parameters` block:

```

transformed parameters {
    real log_lik[N];
    for(n in 1:N){
        real T = rt[n] - T_nd;
        real mu[2] = {alpha[1] -
                      c_lex[n] * beta[1] -
                      lfreq[n] * beta[2],
                      alpha[2] -
                      c_lex[n] * beta[3] -
                      lfreq[n] * beta[4]};
        log_lik[n] = lognormal_race2_lpdf(T | nchoice[n], mu, sigma);
    }
}

```

The variable `log_lik` contains the log-likelihood for each trial. We must not forget to add the total log-likelihood to the target variable. This is done simply by `target += sum(log_lik)`. There are two advantage of iterating first and then adding the total log likelihood to `target`: (i) we can use the variable `log_lik` for model comparison with cross validation (see chapter 17) without the need to repeating code in the generated quantities,

and (ii) using `sum` and adding to `target` once is slighter faster than adding to `target` at each iteration.

The complete Stan code for this model is shown below as `lnrace_mod.stan`, it is left for the reader to verify that the results are the same.

```

functions {
    real lognormal_race2_lpdf(real T, int nchoice, real[] mu, real sigma){
        real lpdf;
        if(nchoice == 1)
            lpdf = lognormal_lpdf(T | mu[1] , sigma) +
                lognormal_lccdf(T | mu[2], sigma);
        else
            lpdf = lognormal_lpdf(T | mu[2], sigma) +
                lognormal_lccdf(T | mu[1], sigma);
        return lpdf;
    }
}
data {
    int<lower = 1> N;
    vector[N] lfreq;
    vector[N] c_lex;
    vector[N] rt;
    int nchoice[N];
}
parameters {
    real alpha[2];
    real beta[4];
    real<lower = 0> sigma;
    real<lower = 0, upper = min(rt)> T_nd;
}
transformed parameters {
    real log_lik[N];
    for(n in 1:N){
        real T = rt[n] - T_nd;
        real mu[2] = {alpha[1] -
                      c_lex[n] * beta[1] -
                      lfreq[n] * beta[2],

```

```

        alpha[2] -
        c_lex[n] * beta[3] -
        lfreq[n] * beta[4]};
log_lik[n] = lognormal_race2_lpdf(T | nchoice[n], mu, sigma);
}
}
model {
    // priors for the task component
target += normal_lpdf(alpha | 6, 1);
target += normal_lpdf(beta | 0, .5);
target += normal_lpdf(sigma | .5, .2)
    - normal_lccdf(0 | .5, .2);
target += normal_lpdf(T_nd | 150, 100)
    - log_diff_exp(normal_lcdf(min(rt) | 150, 100),
                    normal_lcdf(0 | 150, 100));
target += sum(log_lik);
}

```

21.1.4 A hierarchical implementation of the log-normal race model

A simple hierarchical version of the previous model assumes that all the parameters α and β have by-subject adjustments.

$$\begin{aligned}\mu'_{w,n} &= \alpha'_w + u_{subj[n],1} - lex_n \cdot (\beta_{lex_w} + u_{subj[n],2}) - lfreq_n \cdot (\beta_{lfreq_w} + u_{subj[n],3}) \\ \mu'_{nw,n} &= \alpha'_{nw} + u_{subj[n],4} - lex_n \cdot (\beta_{lex_{nw}} + u_{subj[n],5}) - lfreq_n \cdot (\beta_{lfreq_{nw}} + u_{subj[n],6})\end{aligned}\quad (21.24)$$

Similarly to what we did in the hierarchical implementation of the fast-guess model in section 20.1.5 of the previous chapter, we assume that u is a matrix with as many rows as subjects and six columns, which follows a multivariate normal distribution centered on zero. For lack of more information, we assume the same (weakly informative) prior distribution for the six variance components $\tau_{u_{1,\dots,6}}$ with a somewhat smaller effect than we assumed for the prior of σ . As we did with previous hierarchical mod-

els, we assign a regularizing LKJ prior for the correlations between the adjustments.¹

$$\begin{aligned} \mathbf{u} &\sim \mathcal{N}(0, \Sigma_u) \\ \tau_{u_{1..6}} &\sim \text{Normal}_+(.1, .1) \\ \rho_u &\sim \text{LKJcorr}(2) \end{aligned} \quad (21.25)$$

Before we fit the model to the real data, we'll verify that it works with simulated data. To create synthetic data of several subjects, we repeat the same generative process we used before and we add the by-subject adjustments u in the same way we did in section 20.1.5 from chapter 20. This version of the log-normal race model assumes that all the parameters α and β have by-subject adjustments, that is 6 adjustments. To simplify the model, we ignore the possibility of an adjustment for the non decision time T_{nd} , but see Nicenboim and Vasishth (2018) for an implementation of the log-normal race model with a hierarchical non-decision time. For simplicity, all the adjustments, u are normally distributed with the same standard deviation of .2, and they have a .3 correlation between pairs of us; see below `tau_u` and `rho`.

```
# generate always the same data:
set.seed(42)
df_blp_sim <- df_blp
df_blp_sim <- df_blp %>%
  group_by(subj) %>%
  slice_sample(n = 100) %>%
  ungroup()

# True values:
D <- 1800
alpha_w <- .8
beta_wlex <- .5
beta_wlfreq <- .2
alpha_nw <- 1
beta_nwlex <- -.5
```

¹There are 15 correlations since there are 15 ways to choose 2 variables out of 6, where order doesn't matter. This is calculated with $\binom{6}{2}$ which is `choose(6, 2)` in R.

```
beta_nwlfreq <- -.05
sigma <- .8
T_nd <- 150
N <- nrow(df_blp_sim)
# Hierarchical structure
N_subj <- max(df_blp_sim$subj)
N_adj <- 6
tau_u <- rep(.2, N_adj)
rho <- .3
Cor_u <- matrix(rep(rho, N_adj * N_adj), nrow = N_adj)
diag(Cor_u) <- 1
Sigma_u <- diag(tau_u, N_adj, N_adj) %*%
  Cor_u %*%
  diag(tau_u, N_adj, N_adj)
u <- mvrnorm(n = N_subj, rep(0, N_adj), Sigma_u)
subj <- df_blp_sim$subj
mu_w <- alpha_w + u[subj, 1] +
  df_blp_sim$lfreq * (beta_wlfreq + u[subj, 2]) +
  df_blp_sim$c_lex * (beta_wlex + u[subj, 3])
mu_nw <- alpha_nw + u[subj, 4] +
  df_blp_sim$lfreq * (beta_nwlfreq + u[subj, 5]) +
  df_blp_sim$c_lex * (beta_nwlex + u[subj, 6])
# Rate of accumulation
V_w <- rlnorm(N, mu_w, sigma)
V_nw <- rlnorm(N, mu_nw, sigma)
T_w <- D / V_w
T_nw <- D / V_nw
# pairwise minimum
T_winner <- pmin(T_w, T_nw)
# choice
accumulator_winner <- ifelse(T_w == pmin(T_w, T_nw),
  "word",
  "non-word")
df_blp_sim <- df_blp_sim %>%
  mutate(rt = T_nd + T_winner,
    choice = accumulator_winner,
```

```
nchoice = ifelse(choice == "word", 1, 2),
acc = ifelse(lex == choice, 1, 0))
```

The Stan code for this model implements the non-centered parametrization for correlated adjustments (see section 11.1.3 for more details). The model is shown below as `lnrace_h.stan`.

```
functions {
    real lognormal_race2_lpdf(real T, int nchoice, real[] mu, real sigma){
        real lpdf;
        if(nchoice == 1)
            lpdf = lognormal_lpdf(T | mu[1] , sigma) +
                lognormal_lccdf(T | mu[2], sigma);
        else
            lpdf = lognormal_lpdf(T | mu[2], sigma) +
                lognormal_lccdf(T | mu[1], sigma);
        return lpdf;
    }
}
data {
    int<lower = 1> N;
    int<lower = 1> N_subj;
    vector[N] lfreq;
    vector[N] c_lex;
    vector[N] rt;
    int nchoice[N];
    int subj[N];
}
transformed data{
    real min_rt = min(rt);
    real max_rt = max(rt);
    int N_re = 6;
}
parameters {
    real alpha[2];
    real beta[4];
    real<lower = 0> sigma;
```

```
real<lower = 0, upper = min(rt)> T_nd;
vector<lower = 0>[N_re] tau_u;
matrix[N_re, N_subj] z_u;
cholesky_factor_corr[N_re] L_u;
}
transformed parameters {
matrix[N_subj, N_re] u;
u = (diag_pre_multiply(tau_u, L_u) * z_u)';
}
model {
real log_lik[N];
target += normal_lpdf(alpha | 6, 1);
target += normal_lpdf(beta | 0, .5);
target += normal_lpdf(sigma | .5, .2)
- normal_lccdf(0 | .5, .2);
target += normal_lpdf(T_nd | 150, 100)
- log_diff_exp(normal_lcdf(min(rt) | 150, 100),
normal_lcdf(0 | 150, 100));
target += normal_lpdf(tau_u | .1, .1)
- N_re * normal_lccdf(0 | .1, .1);
target += lkj_corr_cholesky_lpdf(L_u | 2);
target += std_normal_lpdf(to_vector(z_u));
for(n in 1:N){
real T = rt[n] - T_nd;
real mu[2] = {alpha[1] + u[subj[n], 1] -
c_lex[n] * (beta[1] + u[subj[n], 2]) -
lfreq[n] * (beta[2] + u[subj[n], 3]),
alpha[2] + u[subj[n], 4] -
c_lex[n] * (beta[3] + u[subj[n], 5]) -
lfreq[n] * (beta[4] + u[subj[n], 6])};
log_lik[n] = lognormal_race2_lpdf(T | nchoice[n], mu, sigma);
}
target += sum(log_lik);
}
generated quantities {
corr_matrix[N_re] rho_u = L_u * L_u';
}
```

Store the simulated data in a list and fit it.

```
lnrace_h <- system.file("stan_models",
                         "lnrace_h.stan",
                         package = "bcogsci")
ls_blp_h_sim <- list(N = nrow(df_blp_sim),
                      N_subj = max(df_blp_sim$subj),
                      subj = df_blp_sim$subj,
                      rt = df_blp_sim$rt,
                      nchoice = df_blp_sim$nchoice,
                      c_lex = df_blp_sim$c_lex,
                      lfreq = df_blp_sim$lfreq)
fit_blp_h_sim <- stan(lnrace_h, data = ls_blp_h_sim)
```

Create a vector with the unique correlations. Print the parameters values.

```
rho_us <- c(paste0("rho_u[1,", 2:6, "]"),
             paste0("rho_u[2,", 3:6, "]"),
             paste0("rho_u[3,", 4:6, "]"),
             paste0("rho_u[4,", 5:6, "]"),
             "rho_u[5,6]")
print(fit_blp_h_sim, pars = c("alpha",
                             "beta",
                             "T_nd",
                             "sigma",
                             "tau_u",
                             rho_us))

##               mean   2.5%  97.5% n_eff Rhat
## alpha[1]      6.74   6.63   6.85  2190 1.00
## alpha[2]      6.59   6.48   6.71  2542 1.00
## beta[1]       0.35   0.14   0.54  2296 1.00
## beta[2]       0.18   0.13   0.24  1459 1.01
## beta[3]      -0.54  -0.69  -0.39  4415 1.00
## beta[4]      -0.15  -0.24  -0.06  1774 1.00
## T_nd        149.38 146.04 152.09  4902 1.00
## sigma        0.82   0.79   0.84  5427 1.00
```

```

## tau_u[1]      0.19  0.12  0.27  2106 1.00
## tau_u[2]      0.19  0.03  0.34  1069 1.01
## tau_u[3]      0.15  0.11  0.19  2043 1.00
## tau_u[4]      0.21  0.12  0.29  2245 1.00
## tau_u[5]      0.13  0.02  0.27  1461 1.01
## tau_u[6]      0.23  0.18  0.29  1643 1.00
## rho_u[1,2]    -0.20 -0.69  0.37  3355 1.00
## rho_u[1,3]    -0.32 -0.67  0.10  765  1.01
## rho_u[1,4]    0.20 -0.25  0.63  1515 1.00
## rho_u[1,5]    -0.02 -0.59  0.54  3240 1.00
## rho_u[1,6]    -0.16 -0.53  0.23  852  1.00
## rho_u[2,3]    0.16 -0.38  0.66  447  1.01
## rho_u[2,4]    -0.09 -0.64  0.48  616  1.01
## rho_u[2,5]    0.03 -0.59  0.62  1660 1.00
## rho_u[2,6]    0.28 -0.28  0.74  545  1.00
## rho_u[3,4]    -0.48 -0.78 -0.08  2857 1.00
## rho_u[3,5]    0.39 -0.26  0.82  3620 1.00
## rho_u[3,6]    0.28 -0.05  0.58  2208 1.00
## rho_u[4,5]    -0.20 -0.73  0.40  4414 1.00
## rho_u[4,6]    -0.22 -0.57  0.17  1736 1.00
## rho_u[5,6]    0.03 -0.53  0.59  541  1.00

```

The code below compares the posterior distributions of the relevant parameters of the model with their true values, and plots them in Figure 21.7. The true value for all the correlations was 0.3, but we need to correct the sign depending on whether there was a minus sign in front of the adjustment when we built `mu_w` and `mu_wn` or not: For example, there is no minus before `u[subj, 1]`, but there is one before `u[subj, 2]`, thus the true correlation between `u[subj, 1]` and `u[subj, 2]` we generated should be negative (plus times minus is minus); and there is a minus before `u[subj, 3]` and thus the correlation between `u[subj, 2]` and `u[subj, 3]` should positive (minus times minus is positive).

```

## c(-1, -1, 1, -1, -1, 1, -1, 1, 1, -1, 1, 1, -1, -1, 1)

corrs <- rho * c(-1, -1, 1, -1, -1, 1, -1, 1, 1, -1, 1, 1, -1, -1, 1)
true_values <- c(log(D) - alpha_w,
                  log(D) - alpha_nw,

```

```

    beta_wlex,
    beta_wlfreq,
    beta_nwlex,
    beta_nwlfreq,
    T_nd,
    sigma,
    tau_u,
    corrs)
par_names = c("alpha",
             "beta",
             "T_nd",
             "sigma",
             "tau_u",
             rho_us)
estimates <- as.data.frame(fit_blp_h_sim) %>%
  select(contains(par_names))
mcmc_recover_hist(estimates, true_values)

```

We see that we can recover the true values quite well, even if there is a big deal of uncertainty over the posteriors of the correlations.

We are now ready to fit the model to the observed data.

Create a list with the real data and fit the same Stan model.

```

lnrace_h <- system.file("stan_models",
                        "lnrace_h.stan",
                        package = "bcogsci")
ls_blp_h <- list(N = nrow(df_blp),
                  N_subj = max(df_blp$subj),
                  subj = df_blp$subj,
                  rt = df_blp$rt,
                  nchoice = df_blp$nchoice,
                  c_lex = df_blp$c_lex,
                  lfreq = df_blp$lfreq)
fit_blp_h_sim <- stan(lnrace_h, data = ls_blp_h)

```

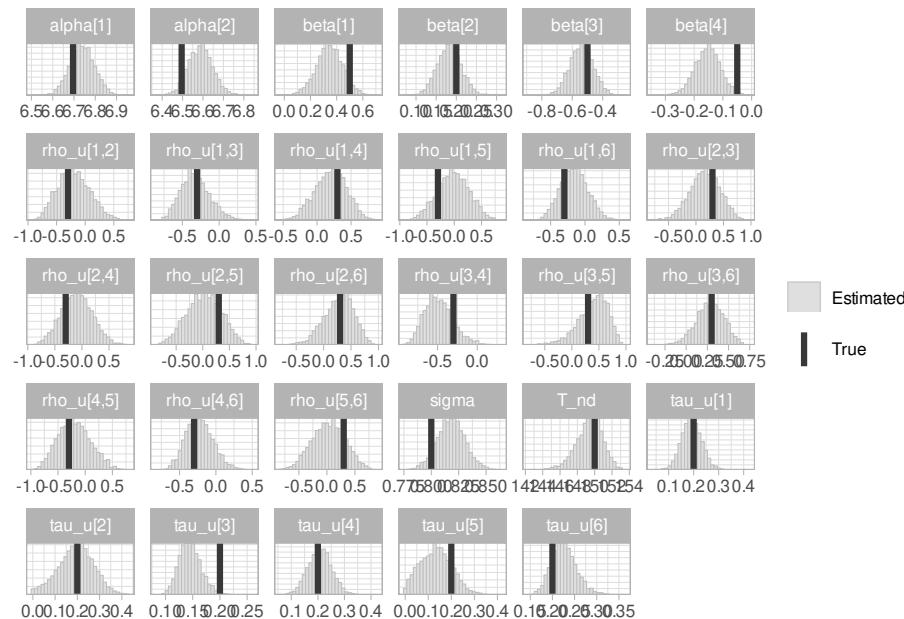


FIGURE 21.7: Posterior distributions of the main parameters of the log-normal race model `fit_blp_h_sim` together with their true values.

Print the summary (omit the correlations for now).

```
print(fit_blp_h, pars = c("alpha",
                           "beta",
                           "T_nd",
                           "sigma",
                           "tau_u"))

##           mean  2.5% 97.5% n_eff Rhat
## alpha[1]  6.67  6.62  6.73   410 1.00
## alpha[2]  6.78  6.73  6.83   631 1.00
## beta[1]   0.35  0.31  0.39  1439 1.00
## beta[2]   0.06  0.06  0.07  1032 1.00
## beta[3]  -0.14 -0.17 -0.10  2066 1.00
## beta[4]  -0.06 -0.07 -0.06  2692 1.00
## T_nd     6.21  5.14  7.17  9163 1.00
## sigma    0.34  0.34  0.34  8822 1.00
```

```
## tau_u[1] 0.14 0.11 0.18 819 1.01
## tau_u[2] 0.09 0.06 0.12 1856 1.00
## tau_u[3] 0.02 0.01 0.02 1413 1.00
## tau_u[4] 0.13 0.11 0.17 1172 1.00
## tau_u[5] 0.08 0.05 0.11 1597 1.00
## tau_u[6] 0.01 0.00 0.02 1256 1.00
```

Even if the model converged, the posterior summary shows a clear problem with the model: The estimate for the non-decision time, T_{nd} is less than 2 milliseconds! This is just not possible, physiological research shows that the eye-to-brain lag, the time it takes for the visual features on the screen until they are propagated from the retina to the brain is approximately 50 milliseconds (Clark, Fan, and Hillyard 1994). Besides identifying the stimuli, the subjects also need to initiate a motor response which takes at least 100 milliseconds. Then how is it possible that we obtained this extremely fast non-decision time? The reason is that the parameter τ_{nd} is constrained to be between zero and the shortest reaction time.

Verify what is the shortest reaction time in the dataset and how many observations are below 150 milliseconds.

```
min(df_blp$rt)
## [1] 16

sum(df_blp$rt < 150)
## [1] 2
```

This shows that some responses must have been initiated even before the stimuli was presented! We turn to deal with the *contaminant* observations next.

21.1.5 Dealing with contaminant responses

So far we have assumed that all the observations were coming from responses done after a decision was made. But what happens if there are anticipations or lapses of attention where the subject responds either before the stimuli is presented or after the the stimuli but without attend-

ing it? We are in a situation analogous to what we described before in chapter 20 with the fast-guess model (Ollman 1966). There, we assumed that the behavior of a subject would be the mixture of two distributions, one corresponding to a guessing mode of responses and another one to a task-engaged mode. There are two major differences with the fast guess model, however. First, we assume here that guesses can be fast (e.g., anticipations) as well as slow. Second, here guesses occur in a minority of the cases and choice and response times are mostly explained by the log-normal race model. The distribution that corresponds to these guesses is sometimes called a *contaminant distribution*. When the contaminant response times are outside the usual range of response times (either shorter or longer), they can cause major problems in data analysis, distorting estimates. As we saw before, extremely short response times caused by anticipating the response can make virtually impossible to estimate the non-decision time.

A recommended approach for dealing with this problem that we follow here is to assume that the responses come from a mixture between the sequential sampling model (in this case the log-normal race model) and a uniform distribution bounded at the minimum and maximum observed response time (e.g., Ratcliff and Tuerlinckx 2002).

The new likelihood function will look as follows:

$$p(rt_n, choice_n) = \theta_c \cdot \text{Uniform}(rt_n | \min(rt), \max(rt)) \cdot \text{Bernoulli}(choice_n | \theta_{bias}) + (1 - \theta_c) \cdot p_{lnrace}(rt_n, choice_n | \mu', \sigma) \quad (21.26)$$

The first term of the sum represents the contaminant component that occurs with probability θ_c and has a likelihood that depends on the response time, represented with the uniform PDF, and on the response given represented with a Bernoulli PDF. When a subject is guessing, the likelihood of each choice depends on θ_{bias} .

The second term of the likelihood represents the log-normal race model that occurs with probability $1 - \theta_c$. We use $p_{lnrace}(rt_n, choice_n)$ as a shorthand for the following function (which we have already used in the models before):

$$p_{lnrace}(rt_n, choice_n) = \begin{cases} LogNormal(\mu'_{w,n}, \sigma) \cdot \\ (1 - LogNormal_CDF(T_n | \mu'_{w,n}, \sigma)) \text{ if } choice = \text{word} \\ \\ LogNormal(\mu'_{nw,n}, \sigma) \cdot \\ (1 - LogNormal_CDF(T_n | \mu'_{nw,n}, \sigma)) \text{ otherwise} \end{cases} \quad (21.27)$$

To simplify the model, we assume that contaminant responses are completely random (i.e., there is no bias to word or non-word) by setting $\theta_{bias} = 0.5$.² This makes $Bernoulli(choice_n | \theta_{bias}) = .5$.

For this model to be identifiable, we need to assume that $\theta_c \ll 1$. This is a sensible assumption since the contaminant distribution is assumed to only happen in a minority of the cases, with the race model contributing to most of the speed-accuracy trade-off in the data. We set the following prior to θ_c .

$$\theta_c \sim Beta(.9, 70) \quad (21.28)$$

By setting the first parameter of the beta distribution to a number smaller than 1 we get a distribution of possible probabilities with a “horn” on the left, see Figure 21.8. Our prior belief for θ_c is on average 0.007, and its 95% CrI is [0, 0.048].³

We also want to “push” the non-decision time further from zero to get more realistic values. For this reason we increase the informativity of the prior of T_{nd} . A log-normal prior discourages values too close to zero, even with a similar location (on log-scale) than the truncated normal prior. We settle with the following prior:

$$T_{nd} \sim LogNormal(log(150), .6) \quad (21.29)$$

²This is, of course, just an assumption that could be verified. But we'll see that the model is already quite complex and achieving convergence is not trivial.

³We calculate the average doing $.9/(.9 + 70)$. The 95% quantile can be calculated in R with `qbeta(c(.025, .975), .9, 70)`.

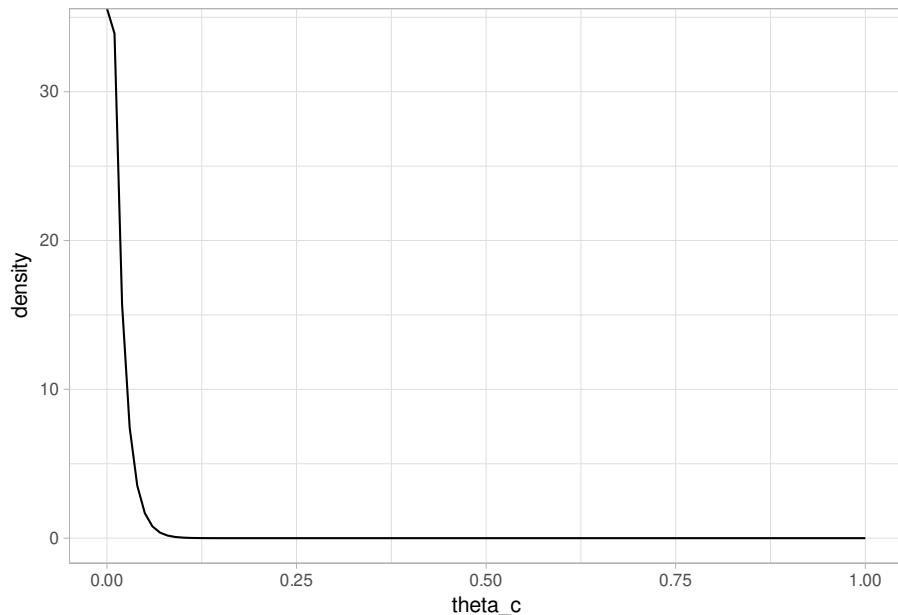


FIGURE 21.8: Prior distribution for θ_c . Most of the probability mass is close to 0.

Finally, we want to be able to account for response times that are actually faster than the non decision time. If the observed response time, rt is smaller than the non decision time, τ_{nd} , we can be sure that the observation belongs to the contaminant distribution, because otherwise the decision time, τ should be negative! This means that in this case, the log-normal race likelihood is 0 (and its logarithm is infinite). When $T < 0$, the log-likelihood of our model is $\log(\theta \times Uniform(rt_n | min, max)) + (1 - \theta) \times p_{lnrace}$ with $p_{lnrace} = 0$. This means that we only use the following code:

```
target += log(theta_c) + uniform_lpdf(rt[n] | min_rt, max_rt) +
    log(0.5);
```

This also means that we need to relax the constraints on τ_{nd} , it doesn't need to be smaller than the smallest observed response time, since some of the observations are responses from the contaminant distribution.

```
real<lower = 0> T_nd;
```

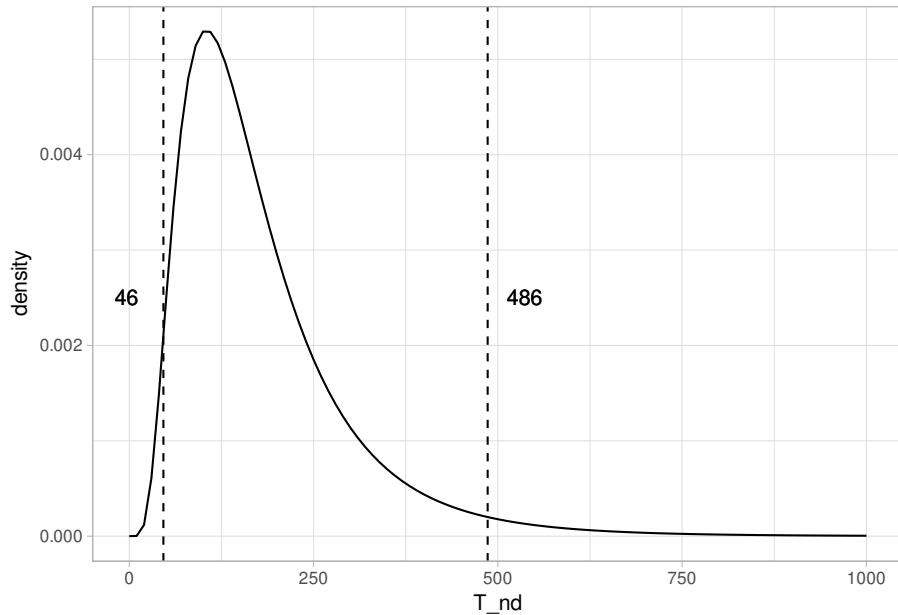


FIGURE 21.9: Prior distribution for T_{nd} . The dashed lines show the 95% credible interval.

When $T > 0$, the likelihood is a mixture of the contaminant distribution and the log-normal race model as defined in (21.26). We use `log_sum_exp` exactly as we did in 20.1.2 and 20.1.3 for the fast We set a mixture distribution between a contaminant distribution and the lognormal distribution as defined in our likelihood in (21.26) by log-transforming it.

```
target += log_sum_exp(
    log(theta_c) + uniform_lpdf(rt[n] | min_rt, max_rt)
    + log(0.5),
    log1m(theta_c) + lognormal_race2_lpdf(T[n] | nchoice[n],
                                             mu, sigma));
```

Finally, the complete Stan code for this model is shown below as `lnrace_cont.stan`.

```
functions {
real lognormal_race2_lpdf(real T, int nchoice, real[] mu, real sigma){
    real lpdf;
    if(nchoice == 1)
```

```
lpdf = lognormal_lpdf(T | mu[1] , sigma) +
       lognormal_lccdf(T | mu[2], sigma);
else
  lpdf = lognormal_lpdf(T | mu[2], sigma) +
       lognormal_lccdf(T | mu[1], sigma);
return lpdf;
}
}
data {
  int<lower = 1> N;
  int<lower = 1> N_subj;
  vector[N] lfreq;
  vector[N] c_lex;
  vector[N] rt;
  int nchoice[N];
  int subj[N];
}
transformed data{
  real min_rt = min(rt);
  real max_rt = max(rt);
  int N_re = 6;
}
parameters {
  real alpha[2];
  real beta[4];
  real<lower = 0> sigma;
  real<lower = 0> T_nd;
  real<lower = 0, upper = 1> theta_c;
  vector<lower = 0>[N_re] tau_u;
  matrix[N_re, N_subj] z_u;
  cholesky_factor_corr[N_re] L_u;
}
transformed parameters {
  matrix[N_subj, N_re] u;
  u = (diag_pre_multiply(tau_u, L_u) * z_u)';
}
model {
```

```

real log_lik[N];
target += normal_lpdf(alpha | 6, 1);
target += normal_lpdf(beta | 0, .5);
target += normal_lpdf(sigma | .5, .2)
    - normal_lccdf(0 | .5, .2);
target += lognormal_lpdf(T_nd | log(150), .6);
target += beta_lpdf(theta_c | .9, 70);
target += normal_lpdf(tau_u | .1, .1)
    - N_re * normal_lccdf(0 | .1, .1);
target += lkj_corr_cholesky_lpdf(L_u | 2);
target += std_normal_lpdf(to_vector(z_u));
for(n in 1:N){
    real T = rt[n] - T_nd;
    if(T > 0){
        real mu[2] = {alpha[1] + u[subj[n], 1] -
                      c_lex[n] * (beta[1] + u[subj[n], 2]) -
                      lfreq[n] * (beta[2] + u[subj[n], 3]),
                      alpha[2] + u[subj[n], 4] -
                      c_lex[n] * (beta[3] + u[subj[n], 5]) -
                      lfreq[n] * (beta[4] + u[subj[n], 6])};
        log_lik[n] = log_sum_exp(
            log(theta_c) + uniform_lpdf(rt[n] | min_rt, max_rt)
            + log(.5), // + bernoulli_lpmf(nchoice[n] -
1 | theta_bias),
            log1m(theta_c) + lognormal_race2_lpdf(T | nchoice[n], mu, sigma));
    } else {
        // Observed time is smaller than the non-decision time
        log_lik[n] = log(theta_c) + uniform_lpdf(rt[n] | min_rt, max_rt)
            + log(.5); // bernoulli_lpmf(nchoice[n] -
1 | theta_bias);
    }
}
target += sum(log_lik);
}

```

Store the real data in a list and fit the model.

```

lnrace_h_cont <- system.file("stan_models",
                             "lnrace_h_cont.stan",
                             package = "bcogsci")
ls_blp_h <- list(N = nrow(df_blp),
                  N_subj = max(df_blp$subj),
                  subj = df_blp$subj,
                  rt = df_blp$rt,
                  nchoice = df_blp$nchoice,
                  c_lex = df_blp$c_lex,
                  lfreq = df_blp$lfreq)
fit_blp_h_cont <- stan(lnrace_h_cont, data = ls_blp_h)

```

This model takes more than a day to finish in a relatively powerful computer and, disappointingly, it doesn't converge as it is clear from the traceplots in Figure 21.10

```

traceplot(fit_blp_h_cont, pars = c("alpha",
                                   "beta",
                                   "T_nd",
                                   "theta_c",
                                   "sigma",
                                   "tau_u"))

```

The traceplot in Figure ?? shows that the chains get stuck and don't mix well. It seems that there is not enough information to constraint the model. If we look at the parameter `theta_c`, we see that its chains are getting stuck at very unlikely values that are over 0.25. While in general is not recommended to cut off values from a prior, just because it's unlikely, in this case restricting the parameter `theta_c` to be smaller than 0.1 solves the convergence problems. To truncate the prior for θ_c in Stan we declare the parameter to have a higher boundart of .1,

```
real<lower = 0, upper = .1> theta_c;
```

and we change its prior distribution in the `model` block to the following.

```

target += beta_lpdf(theta_c | .9, 70) -
  beta_lcdf(.1 | .9, 70);

```

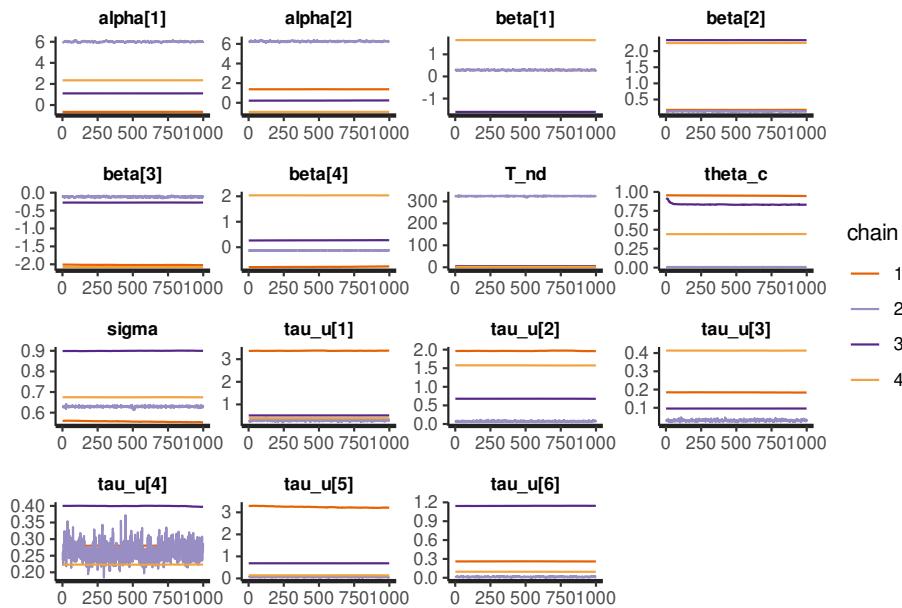


FIGURE 21.10: The traceplots of the `fit_blp_h` shows that the chains are clearly not mixing well.

Fit the modified model to the same data.

```
lnrace_h_contb <- system.file("stan_models",
                               "lnrace_h_contb.stan",
                               package = "bcogsci")
fit_blp_h_contb <- stan(lnrace_h_cont, data = ls_blp_h)
```

This time the model took a day and a half! The results do make sense now!

Print the summary of the main parameters.

```
print(fit_blp_h_contb, pars = c("alpha",
                                "beta",
                                "T_nd",
                                "sigma",
                                "theta_c",
                                "tau_u"))
```

```
##               mean 2.5% 97.5% n_eff Rhat
## alpha[1]      6.01 5.91  6.11   567    1
## alpha[2]      6.26 6.17  6.35   791    1
## beta[1]       0.29 0.26  0.32  2001    1
## beta[2]       0.13 0.12  0.15  1261    1
## beta[3]      -0.12 -0.14 -0.09  2150    1
## beta[4]      -0.13 -0.13 -0.12  2649    1
## T_nd        324.30 322.86 325.63 4121    1
## sigma        0.63 0.62  0.64  5105    1
## theta_c      0.01 0.01  0.01  6243    1
## tau_u[1]     0.30 0.25  0.36  1139    1
## tau_u[2]     0.07 0.05  0.10  1880    1
## tau_u[3]     0.03 0.02  0.04  1559    1
## tau_u[4]     0.26 0.21  0.32  1431    1
## tau_u[5]     0.07 0.05  0.09  2164    1
## tau_u[6]     0.01 0.01  0.02  1752    1
```

Print the summary of the correlations

```
print(fit_blp_h_contb, pars = rho_us)
```

```
##               mean 2.5% 97.5% n_eff Rhat
## rho_u[1,2]   0.04 -0.26  0.34  1987    1
## rho_u[1,3]   0.19 -0.10  0.47  1664    1
## rho_u[1,4]   0.72  0.53  0.85  1639    1
## rho_u[1,5]   0.07 -0.24  0.37  2478    1
## rho_u[1,6]  -0.48 -0.80 -0.06  3895    1
## rho_u[2,3]  -0.39 -0.67 -0.04  1026    1
## rho_u[2,4]   0.08 -0.25  0.39  1739    1
## rho_u[2,5]  -0.60 -0.84 -0.28  2186    1
## rho_u[2,6]  -0.14 -0.60  0.35  3945    1
## rho_u[3,4]   0.05 -0.25  0.34  1803    1
## rho_u[3,5]   0.26 -0.11  0.57  2799    1
## rho_u[3,6]  -0.15 -0.57  0.32  3523    1
## rho_u[4,5]  -0.19 -0.50  0.12  2876    1
## rho_u[4,6]  -0.21 -0.60  0.22  4378    1
## rho_u[5,6]  -0.12 -0.56  0.42  3414    1
```

```
lnrace_h_contb_gen <- "lnrace_h_contb_gen.stan"
gen_model <- stan_model(lnrace_h_contb_gen)
samples <- as.matrix(fit_blp_h_contb,
                      pars = c("alpha", "beta", "sigma", "T_nd", "theta_c", "tau_u", "u"))
```

```
gen_race <- gqs(gen_model,
                  draws = samples[1:500,],
                  data = ls_blp_h)
```

```
df_rt <- rstan::extract(gen_race)$rt_pred %>%
  # Convert a matrix of 500 x 66000 (iter x N obs) into data.frame
  # where each column is V1... V66000
  as.data.frame() %>%
  # Add a column which identifies each iter as a simulation
  mutate(sim = 1:n()) %>%
  # Pivot the data frame so that it is 500 * 66000 long
  # Each row indicates in
  # - sim: from which simulation the observation is coming
  # - obs_id: identifies the 66000 observations
  # - rt_pred: simulated RT
  # Since each observation is in a column starting with V
  # `names_prefix` removes the "V"
  pivot_longer(cols = -sim,
               names_to = "obs_id",
               names_prefix = "V",
               values_to = "rt_pred") %>%
  # Make sure that obs_id is a number (and not a character)
  mutate(obs_id = as.numeric(obs_id))

# Repite with nchoice
df_nchoice <- rstan::extract(gen_race)$nchoice_pred %>%
  as.data.frame() %>%
  mutate(sim = 1:n()) %>%
  pivot_longer(cols = -sim,
```

```
    names_to = "obs_id",
    names_prefix = "V",
    values_to ="nchoice_pred") %>%
  mutate(obs_id = as.numeric(obs_id))

# Main columns from the data set
df_blp_min <- df_blp %>%
  select(subj, lex, lfreq, rt, nchoice) %>%
  mutate(obs_id = 1:n())
# All the information is joined
df_blp_pred <- left_join(df_rt, df_nchoice) %>%
  left_join(select(df_blp_min, -rt, -nchoice)) %>%
  rename(rt = rt_pred, nchoice = nchoice_pred) %>%
  bind_rows(df_blp_min) %>%
  mutate(acc = ifelse((nchoice == 1 & lex == "word") |
    (nchoice == 2 & lex == "non-word"), 1, 0))

df_blp_pred %>% group_by(sim) %>% summarize(mean(acc))

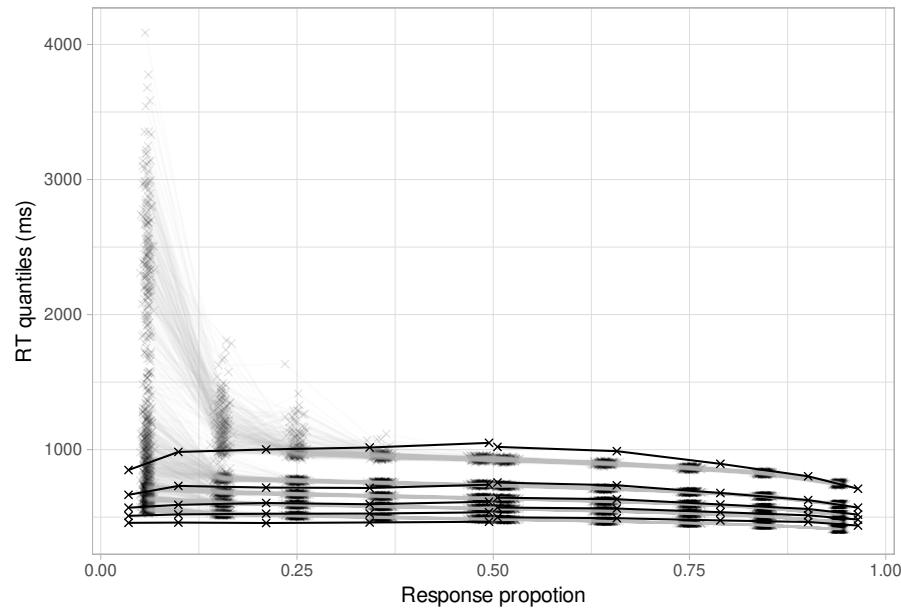
df_blp_pred_qpf <- df_blp_pred %>%
  # Subset only words
  filter(lex == "word") %>%
  # Create 5 word frequencies group
  mutate(freq_group =
    cut(lfreq,
      quantile(lfreq, c(0,.2,.4, .6, .8, 1)),
      include.lowest = TRUE,
      labels =
        c("0-.2", ".2-.4", ".4-.6", ".6-.8", ".8-1")))
  ) %>%
  # Group by condition and subject
  group_by(freq_group, sim, subj) %>%
  # Apply the quantile probability function
  qpf() %>%
##  # Group again removing subj
  group_by(freq_group, sim, q, response) %>%
```

```

## # Get averages of all the quantities
summarize(rt_q = mean(rt_q),
          p = mean(p))

ggplot(df_blp_pred_qpf %>% filter(sim < 200), aes(x = p, y = rt_q)) +
  geom_point(shape = 4, alpha = 0.1) +
  geom_line(aes(group = interaction(q, response, sim)), alpha = 0.05, color = "grey") +
  ylab("RT quantiles (ms)") +
  xlab("Response proportion") +
  geom_point(data = df_blp_pred_qpf %>% filter(is.na(sim)),
             shape = 4) +
  geom_line(data = df_blp_pred_qpf %>% filter(is.na(sim)),
            aes(group = interaction(q, response)))

```



21.2 Further reading

Heathcote et al. (2019) outline how to fit evidence-accumulation models in a Bayesian framework with a custom R package that relies on a Differential-Evolution sampler (DE-MCMC; Turner et al. 2013).



References

- Allaire, JJ, Yihui Xie, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, Hadley Wickham, Joe Cheng, Winston Chang, and Richard Iannone. 2019. *Rmarkdown: Dynamic Documents for R*. <https://CRAN.R-project.org/package=rmarkdown>.
- Anderson, John R., Dan Bothell, Michael D. Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. 2004. “An Integrated Theory of the Mind.” *Psychological Review* 111 (4): 1036–60.
- Ashby, F Gregory. 1982. “Testing the Assumptions of Exponential, Additive Reaction Time Models.” *Memory & Cognition* 10 (2). Springer: 125–34.
- Ashby, F Gregory, and James T Townsend. 1980. “Decomposing the Reaction Time Distribution: Pure Insertion and Selective Influence Revisited.” *Journal of Mathematical Psychology* 21 (2). Elsevier: 93–123.
- Audley, RJ, and AR Pike. 1965. “Some Alternative Stochastic Models of Choice 1.” *British Journal of Mathematical and Statistical Psychology* 18 (2). Wiley Online Library: 207–25.
- Auguie, Baptiste. 2017. *GridExtra: Miscellaneous Functions for "Grid" Graphics*. <https://CRAN.R-project.org/package=gridExtra>.
- Aust, Frederik. 2019. *Citr: RStudio Add-in to Insert Markdown Citations*. <https://CRAN.R-project.org/package=citr>.
- Aust, Frederik, and Marius Barth. 2020. *papaja: Create APA Manuscripts with R Markdown*. <https://github.com/crsh/papaja>.
- Baayen, R Harald, Douglas J Davidson, and Douglas M Bates. 2008. “Mixed-Effects Modeling with Crossed Random Effects for Subjects and Items.” *Journal of Memory and Language* 59 (4). Elsevier: 390–412.

- Baguley, Thomas. 2012. *Serious Stats: A Guide to Advanced Statistics for the Behavioral Sciences*. Macmillan International Higher Education.
- Barr, Dale J, Roger Levy, Christoph Scheepers, and Harry J Tily. 2013. “Random Effects Structure for Confirmatory Hypothesis Testing: Keep It Maximal.” *Journal of Memory and Language* 68 (3). Elsevier: 255–78.
- Batchelder, William H, and David M Riefer. 1990. “Multinomial Processing Models of Source Monitoring.” *Psychological Review* 97 (4). American Psychological Association: 548.
- . 1999. “Theoretical and Empirical Review of Multinomial Process Tree Modeling.” *Psychonomic Bulletin & Review* 6 (1). Springer: 57–86.
- Bates, Douglas M, Reinhold Kliegl, Shravan Vasishth, and Harald Baayen. 2015. “Parsimonious Mixed Models.”
- Bates, Douglas M, and Martin Maechler. 2019. *Matrix: Sparse and Dense Matrix Classes and Methods*. <https://CRAN.R-project.org/package=Matrix>.
- Bates, Douglas M, Martin Mächler, Ben Bolker, and Steve Walker. 2015a. “Fitting Linear Mixed-Effects Models Using lme4.” *Journal of Statistical Software* 67 (1): 1–48. <https://doi.org/10.18637/jss.v067.i01>.
- . 2015b. “Fitting Linear Mixed-Effects Models Using lme4.” *Journal of Statistical Software* 67 (1): 1–48. <https://doi.org/10.18637/jss.v067.i01>.
- Belin, TR, and DB Rubin. 1990. “Analysis of a Finite Mixture Model with Variance Components.” In *Proceedings of the Social Statistics Section*, 211–15.
- Bennett, Charles H. 1976. “Efficient Estimation of Free Energy Differences from Monte Carlo Data.” *Journal of Computational Physics* 22 (2): 245–68. [https://doi.org/10.1016/0021-9991\(76\)90078-4](https://doi.org/10.1016/0021-9991(76)90078-4).
- Bernardo, José M, and Adrian FM Smith. 2009. *Bayesian Theory*. Vol. 405. John Wiley & Sons.
- Betancourt, Michael J. 2016. “Identifying the Optimal Integration Time in Hamiltonian Monte Carlo.”
- . 2017. “A Conceptual Introduction to Hamiltonian Monte Carlo.”
- . 2018. “Towards a Principled Bayesian Workflow.” [https:](https://)

//betanalpha.github.io/assets/case_studies/principled_bayesian_workflow.html.

Betancourt, Michael J., and Mark Girolami. 2013. “Hamiltonian Monte Carlo for Hierarchical Models.”

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning*. Springer.

Blitzstein, Joseph K., and Jessica Hwang. 2014. *Introduction to Probability*. Chapman; Hall/CRC.

Blumberg, Eric J., Matthew S. Peterson, and Raja Parasuraman. 2015. “Enhancing Multiple Object Tracking Performance with Noninvasive Brain Stimulation: A Causal Role for the Anterior Intraparietal Sulcus.” *Frontiers in Systems Neuroscience* 9: 3. <https://doi.org/10.3389/fnsys.2015.00003>.

Bolker, Ben. 2018. “Https://Github.com/Bbolker/Mixedmodels-Misc/Blob/Master/Notes/Contrasts.rmd.”

Box, George EP. 1979. “Robustness in the Strategy of Scientific Model Building.” In *Robustness in Statistics*, 201–36. Elsevier.

Brée, David S. 1975. “The Distribution of Problem-Solving Times: An Examination of the Stages Model.” *British Journal of Mathematical and Statistical Psychology* 28 (2): 177–200. <https://doi.org/10.cnx3q7>.

Britten, Kenneth H., Michael N. Shadlen, William T. Newsome, and J. Anthony Movshon. 1993. “Responses of Neurons in Macaque Mt to Stochastic Motion Signals.” *Visual Neuroscience* 10 (6). Cambridge University Press: 1157–69. <https://doi.org/10.1017/S0952523800010269>.

Broadbent, Donald E., and Margaret H. P. Broadbent. 1987. “From Detection to Identification: Response to Multiple Targets in Rapid Serial Visual Presentation.” *Perception & Psychophysics* 42 (2): 105–13. <https://doi.org/10.3758/BF03210498>.

Brown, Scott D., and Andrew Heathcote. 2008. “The Simplest Complete Model of Choice Response Time: Linear Ballistic Accumulation.” *Cognitive Psychology* 57 (3): 153–78. <https://doi.org/10.1016/j.cogpsych.2007.12.002>.

Brown, Scott, and Andrew Heathcote. 2005. “A Ballistic Model of Choice

- Response Time." *Psychological Review* 112 (1). American Psychological Association: 117.
- Browne, William J, and David Draper. 2006. "A Comparison of Bayesian and Likelihood-Based Methods for Fitting Multilevel Models." *Bayesian Analysis* 1 (3). International Society for Bayesian Analysis: 473–514.
- Brysbaert, Marc, Paweł Mandera, and Emmanuel Keuleers. 2018. "The Word Frequency Effect in Word Processing: An Updated Review." *Current Directions in Psychological Science* 27 (1): 45–50. <https://doi.org/10.1177/0963721417727521>.
- Burger, Edward B, and Michael Starbird. 2012. *The 5 Elements of Effective Thinking*. Princeton University Press.
- Busemeyer, Jerome R, and Adele Diederich. 2010. *Cognitive Modeling*. Sage.
- Buzsáki, György, and Kenji Mizuseki. 2014. "The Log-Dynamic Brain: How Skewed Distributions Affect Network Operations." *Nature Reviews Neuroscience* 15 (4): 264–78. <https://doi.org/10.1038/nrn3687>.
- Bürki, Audrey, Shereen Elbuy, Sylvain Madec, and Shravan Vasishth. 2020. "What Did We Learn from Forty Years of Research on Semantic Interference? A Bayesian Meta-Analysis." *Journal of Memory and Language*. <https://doi.org/10.1016/j.jml.2020.104125>.
- Bürkner, Paul-Christian. 2017. "brms: An R Package for Bayesian Multi-level Models Using Stan." *Journal of Statistical Software* 80 (1): 1–28. <https://doi.org/10.18637/jss.v080.i01>.
- . 2019. *Brms: Bayesian Regression Models Using 'Stan'*. <https://CRAN.R-project.org/package=brms>.
- Bürkner, Paul-Christian, and Emmanuel Charpentier. 2020. "Modelling Monotonic Effects of Ordinal Predictors in Bayesian Regression Models." *British Journal of Mathematical and Statistical Psychology*. Wiley Online Library.
- Bürkner, Paul-Christian, and Matti Vuorre. 2018. "Ordinal Regression Models in Psychological Research: A Tutorial." *PsyArXiv Preprints*.

- Carlin, Bradley P, and Thomas A Louis. 2008. *Bayesian Methods for Data Analysis*. CRC Press.
- Carney, Dana R, Amy JC Cuddy, and Andy J Yap. 2010. "Power Posing: Brief Nonverbal Displays Affect Neuroendocrine Levels and Risk Tolerance." *Psychological Science* 21 (10). Sage Publications Sage CA: Los Angeles, CA: 1363–8.
- Carpenter, Bob, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael J. Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. "Stan: A Probabilistic Programming Language." *Journal of Statistical Software* 76 (1). Columbia Univ., New York, NY (United States); Harvard Univ., Cambridge, MA (United States).
- Chambers, Chris. 2019. *The Seven Deadly Sins of Psychology: A Manifesto for Reforming the Culture of Scientific Practice*. Princeton University Press.
- Chang, Winston. 2018. *Webshot: Take Screenshots of Web Pages*. <https://CRAN.R-project.org/package=webshot>.
- Chen, Stanley F, and Joshua Goodman. 1999. "An Empirical Study of Smoothing Techniques for Language Modeling." *Computer Speech & Language* 13 (4): 359–94. <https://doi.org/https://doi.org/10.1006/csla.1999.0128>.
- Cheng, Joe. 2018. *MiniUI: Shiny UI Widgets for Small Screens*. <https://CRAN.R-project.org/package=miniUI>.
- Christensen, Ronald, Wesley Johnson, Adam Branscum, and Timothy Hanson. 2011. "Bayesian Ideas and Data Analysis." CRC Press.
- Clark, Vincent P, Silu Fan, and Steven A Hillyard. 1994. "Identification of Early Visual Evoked Potential Generators by Retinotopic and Topographic Analyses." *Human Brain Mapping* 2 (3). Wiley Online Library: 170–87.
- Cook, Samantha R, Andrew Gelman, and Donald B Rubin. 2006. "Validation of Software for Bayesian Models Using Posterior Quantiles." *Journal of Computational and Graphical Statistics* 15 (3). Taylor & Francis: 675–92. <https://doi.org/10.1198/106186006X136976>.
- Cumming, Geoff. 2014. "The New Statistics: Why and How." *Psychological Science* 25 (1): 7–29.

- Damasio, Antonio R. 1992. "Aphasia." *New England Journal of Medicine* 326 (8). Mass Medical Soc: 531–39.
- DeLong, Katherine A, Thomas P Urbach, and Marta Kutas. 2005. "Probabilistic Word Pre-Activation During Language Comprehension Inferred from Electrical Brain Activity." *Nature Neuroscience* 8 (8): 1117–21. <https://doi.org/10.1038/nn1504>.
- DerSimonian, Rebecca, and Nan Laird. 1986. "Meta-Analysis in Clinical Trials." *Controlled Clinical Trials* 7 (3). Elsevier: 177–88.
- Dickey, James M, BP Lientz, and others. 1970. "The Weighted Likelihood Ratio, Sharp Hypotheses About Chances, the Order of a Markov Chain." *The Annals of Mathematical Statistics* 41 (1). Institute of Mathematical Statistics: 214–26.
- Dillon, Brian, Alan Mishler, Shayne Sloggett, and Colin Phillips. 2013. "Contrasting Intrusion Profiles for Agreement and Anaphora: Experimental and Modeling Evidence." *Journal of Memory and Language* 69 (2). Elsevier: 85–103.
- Dillon, Brian William. 2011. "Structured Access in Sentence Comprehension." PhD thesis.
- Dobson, Annette J, and Adrian Barnett. 2011. *An Introduction to Generalized Linear Models*. CRC press.
- Drton., Mathias. 2013. *SIN: A Sinful Approach to Selection of Gaussian Graphical Markov Models*. <https://CRAN.R-project.org/package=SIN>.
- Duane, Simon, A. D. Kennedy, Brian J. Pendleton, and Duncan Roweth. 1987. "Hybrid Monte Carlo." *Physics Letters B* 195 (2): 216–22. [https://doi.org/10.1016/0370-2693\(87\)91197-X](https://doi.org/10.1016/0370-2693(87)91197-X).
- Dufau, Stéphane, Jonathan Grainger, and Johannes C Ziegler. 2012. "How to Say 'No' to a Nonword: A Leaky Competing Accumulator Model of Lexical Decision." *Journal of Experimental Psychology: Learning, Memory, and Cognition* 38 (4). American Psychological Association: 1117.
- Dutilh, Gilles, Jeffrey Annis, Scott D Brown, Peter Cassey, Nathan J Evans, Raoul PPP Grasman, Guy E Hawkins, et al. 2019. "The Quality of Response Time Data Inference: A Blinded, Collaborative Assessment of the Validity

- of Cognitive Models." *Psychonomic Bulletin & Review* 26 (4). Springer: 1051–69.
- Dutilh, Gilles, Eric-Jan Wagenmakers, Ingmar Visser, and Han L. J. van der Maas. 2011. "A Phase Transition Model for the Speed-Accuracy Trade-Off in Response Time Experiments." *Cognitive Science* 35 (2): 211–50. <https://doi.org/10.1111/j.1551-6709.2010.01147.x>.
- Ebersole, Charles R., Olivia E. Atherton, Aimee L. Belanger, Hayley M. Skulborstad, Jill M. Allen, Jonathan B. Banks, Erica Baranski, et al. 2016. "Many Labs 3: Evaluating Participant Pool Quality Across the Academic Semester via Replication." *Journal of Experimental Social Psychology* 67: 68–82. <https://doi.org/https://doi.org/10.1016/j.jesp.2015.10.012>.
- Eddelbuettel, Dirk, Romain Francois, JJ Allaire, Kevin Ushey, Qiang Kou, Nathan Russell, Douglas M Bates, and John Chambers. 2019. *Rcpp: Seamless R and C++ Integration*. <https://CRAN.R-project.org/package=Rcpp>.
- Engelmann, Felix, Lena A. Jäger, and Shravan Vasishth. 2020. "The Effect of Prominence and Cue Association in Retrieval Processes: A Computational Account." *Cognitive Science* 43 (12): e12800. <https://doi.org/10.1111/cogs.12800>.
- Faraway, Julian J. 2016. *Extending the Linear Model with R: Generalized Linear, Mixed Effects and Nonparametric Regression Models*. Chapman; Hall/CRC.
- Faraway, Julian James. 2002. *Practical Regression and ANOVA using R*. Vol. 168. Citeseer.
- Farrell, Simon, and Stephan Lewandowsky. 2018. *Computational Modeling of Cognition and Behavior*. Cambridge University Press.
- Fedorenko, Evelina, Edward Gibson, and Douglas Rohde. 2006. "The Nature of Working Memory Capacity in Sentence Comprehension: Evidence Against Domain-Specific Working Memory Resources." *Journal of Memory and Language* 54 (4). Elsevier: 541–53.
- Fieller, Nick. 2016. *Basics of Matrix Algebra for Statistics with R*. Boca Raton, FL: CRC Press.
- Fosse, Nathan E. 2016. "Replication Data for 'Power Posing: Brief Non-verbal Displays Affect Neuroendocrine Levels and Risk Tolerance' by Car-

- ney, Cuddy, Yap (2010)." Harvard Dataverse. <https://doi.org/10.7910/DVN/FMEGS6>.
- Fox, John. 2009. *A Mathematical Primer for Social Statistics*. Vol. 159. Sage.
- . 2015. *Applied Regression Analysis and Generalized Linear Models*. Sage Publications.
- Francois, Romain. 2017. *Bibtex: Bibtex Parser*. <https://CRAN.R-project.org/package=bibtex>.
- Frank, Stefan L., Leun J. Otten, Giulia Galli, and Gabriella Vigliocco. 2015. "The ERP Response to the Amount of Information Conveyed by Words in Sentences." *Brain and Language* 140: 1–11. <https://doi.org/10.1016/j.bandl.2014.10.006>.
- Frank, Stefan L., Thijs Trompenaars, and Shravan Vasishth. 2015. "Cross-Linguistic Differences in Processing Double-Embedded Relative Clauses: Working-Memory Constraints or Language Statistics?" *Cognitive Science* 40: 554–78. <https://doi.org/10.1111/cogs.12247>.
- Friendly, Michael, John Fox, and Phil Chalmers. 2020. *Matlib: Matrix Functions for Teaching and Learning Linear Algebra and Multivariate Statistics*. <https://CRAN.R-project.org/package=matlib>.
- Gabry, Jonah, and Rok Češnovar. 2021. *Cmdstanr: R Interface to 'Cmdstan'*.
- Gabry, Jonah, and Tristan Mahr. 2019. *Bayesplot: Plotting for Bayesian Models*. <https://CRAN.R-project.org/package=bayesplot>.
- Gabry, Jonah, Daniel Simpson, Aki Vehtari, Michael J. Betancourt, and Andrew Gelman. 2017. "Visualization in Bayesian Workflow." *arXiv Preprint arXiv:1709.01449*.
- Gamerman, Dani, and Hedibert F Lopes. 2006. *Markov chain Monte Carlo: Stochastic simulation for Bayesian inference*. CRC Press.
- Ge, Hong, Kai Xu, and Zoubin Ghahramani. 2018. "Turing: A Language for Flexible Probabilistic Inference." In *Proceedings of Machine Learning Research*, edited by Amos Storkey and Fernando Perez-Cruz, 84:1682–90. Playa Blanca, Lanzarote, Canary Islands: PMLR. <http://proceedings.mlr.press/v84/ge18b.html>.

- Geisser, Seymour, and William F Eddy. 1979. "A Predictive Approach to Model Selection." *Journal of the American Statistical Association* 74 (365). Taylor & Francis Group: 153–60.
- Gelman, Andrew. 2006. "Prior Distributions for Variance Parameters in Hierarchical Models (Comment on Article by Browne and Draper)." *Bayesian Analysis* 1 (3). International Society for Bayesian Analysis: 515–34.
- Gelman, Andrew, and John B. Carlin. 2014. "Beyond Power Calculations: Assessing Type S (Sign) and Type M (Magnitude) Errors." *Perspectives on Psychological Science* 9 (6). SAGE Publications: 641–51.
- Gelman, Andrew, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. 2014. *Bayesian Data Analysis*. Third Edition. Boca Raton, FL: Chapman; Hall/CRC Press.
- Gelman, Andrew, and Jennifer Hill. 2007. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press.
- Gelman, Andrew, Daniel Simpson, and Michael J. Betancourt. 2017. "The Prior Can Often Only Be Understood in the Context of the Likelihood." *Entropy* 19 (10): 555. <https://doi.org/10.3390/e19100555>.
- Gelman, Andrew, Aki Vehtari, Daniel Simpson, Charles C Margossian, Bob Carpenter, Yuling Yao, Lauren Kennedy, Jonah Gabry, Paul-Christian Bürkner, and Martin Modrák. 2020. "Bayesian Workflow." *arXiv Preprint arXiv:2011.01808*.
- Gentle, James E. 2007. "Matrix Algebra: Theory, Computations, and Applications in Statistics." *Springer Texts in Statistics* 10. New York, NY: Springer.
- Gibson, Edward, and James Thomas. 1999. "Memory Limitations and Structural Forgetting: The Perception of Complex Ungrammatical Sentences as Grammatical." *Language and Cognitive Processes* 14(3): 225–48.
- Gibson, Edward, and H-H Iris Wu. 2013. "Processing Chinese Relative Clauses in Context." *Language and Cognitive Processes* 28 (1-2). Taylor & Francis: 125–55.
- Gill, Jeff. 2006. *Essential Mathematics for Political and Social Research*. Cambridge University Press Cambridge.
- Gneiting, Tilmann, and Adrian E Raftery. 2007. "Strictly Proper Scoring Rules, Prediction, and Estimation." *Statistica Sinica* 17 (4): 1695–1710.

- ing Rules, Prediction, and Estimation.” *Journal of the American Statistical Association* 102 (477). Taylor & Francis: 359–78. <https://doi.org/10.1198/016214506000001437>.
- Gohel, David, Hadley Wickham, Lionel Henry, and Jeroen Ooms. 2019. *Gdtools: Utilities for Graphical Rendering*. <https://CRAN.R-project.org/package=gdtools>.
- Good, I. J. 1952. “Rational Decisions.” *Journal of the Royal Statistical Society. Series B (Methodological)* 14 (1). [Royal Statistical Society, Wiley]: 107–14. <http://www.jstor.org/stable/2984087>.
- Goodrich, Ben, Jonah Gabry, Imad Ali, and Sam Brilleman. 2018. “Rstanarm: Bayesian Applied Regression Modeling via Stan.” <http://mc-stan.org/>.
- Goodrich, Ben, Andrew Gelman, Bob Carpenter, Matt Hoffman, Daniel Lee, Michael Betancourt, Marcus Brubaker, et al. 2019. *StanHeaders: C++ Header Files for Stan*. <https://CRAN.R-project.org/package=StanHeaders>.
- Grassi, Massimo, Camilla Crotti, David Giofrè, Ingrid Boedker, and Enrico Toffalini. 2021. “Two Replications of Raymond, Shapiro, and Arnell (1992), the Attentional Blink.” *Behavior Research Methods* 53 (2): 656–68. <https://doi.org/10.3758/s13428-020-01457-6>.
- Grodner, Daniel, and Edward Gibson. 2005. “Consequences of the Serial Nature of Linguistic Input.” *Cognitive Science* 29: 261–90.
- Gronau, Quentin F, Alexandra Sarafoglou, Dora Matzke, Alexander Ly, Udo Boehm, Maarten Marsman, David S Leslie, Jonathan J Forster, Eric-Jan Wagenmakers, and Helen Steingroever. 2017a. “A Tutorial on Bridge Sampling.” *Journal of Mathematical Psychology* 81. Elsevier: 80–97.
- . 2017b. “A Tutorial on Bridge Sampling.” *Journal of Mathematical Psychology* 81: 80–97. <https://doi.org/10.1016/j.jmp.2017.09.005>.
- Gronau, Quentin F., Henrik Singmann, and Eric-Jan Wagenmakers. 2020. “bridgesampling: An R Package for Estimating Normalizing Constants.” *Journal of Statistical Software* 92 (10): 1–29. <https://doi.org/10.18637/jss.v092.i10>.
- Gronau, Quentin F, Henrik Singmann, and Eric-Jan Wagenmakers. 2017.

- “Bridgesampling: An R Package for Estimating Normalizing Constants.” *Arxiv*. <http://arxiv.org/abs/1710.08162>.
- Gronau, Quentin F, and Eric-Jan Wagenmakers. 2018. “Limitations of Bayesian Leave-One-Out Cross-Validation for Model Selection.” *Computational Brain & Behavior*. <https://doi.org/10.1007/s42113-018-0011-7>.
- . 2019. “Rejoinder: More Limitations of Bayesian Leave-One-Out Cross-Validation.” *Computational Brain & Behavior* 2 (1). Springer: 35–47.
- Guo, Jiqiang, Jonah Gabry, and Ben Goodrich. 2019. *Rstan: R Interface to Stan*. <https://CRAN.R-project.org/package=rstan>.
- Haines, Nathaniel, Peter D Kvam, Louis H Irving, Colin Smith, Theodore P Beauchaine, Mark A Pitt, Woo-Young Ahn, and Brandon Turner. 2020. “Learning from the Reliability Paradox: How Theoretically Informed Generative Models Can Advance the Social, Behavioral, and Brain Sciences.” *Unpublished*. PsyArXiv.
- Hammerly, Christopher, Adrian Staub, and Brian Dillon. 2019. “The Grammaticality Asymmetry in Agreement Attraction Reflects Response Bias: Experimental and Modeling Evidence.” *Cognitive Psychology* 110: 70–104.
- Han, Ding, Jana Wegrzyn, Hua Bi, Ruihua Wei, Bin Zhang, and Xiaorong Li. 2018. “Practice Makes the Deficiency of Global Motion Detection in People with Pattern-Related Visual Stress More Apparent.” *PLOS ONE* 13 (2). Public Library of Science: 1–13. <https://doi.org/10.1371/journal.pone.0193215>.
- Harrell Jr, Frank E. 2015. *Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis*. Springer.
- Harris, Christopher M., and Jonathan Waddington. 2012. “On the Convergence of Time Interval Moments: Caveat Sciscitator.” *Journal of Neuroscience Methods* 205 (2): 345–56. <https://doi.org/https://doi.org/10.1016/j.jneumeth.2012.01.017>.
- Harris, Christopher M., Jonathan Waddington, Valerio Biscione, and Sean Manzi. 2014. “Manual Choice Reaction Times in the Rate-Domain.” *Frontiers in Human Neuroscience* 8: 418. <https://doi.org/10.3389/fnhum.2014.00418>.

- Hayes, Taylor R., and Alexander A. Petrov. 2016. "Mapping and Correcting the Influence of Gaze Position on Pupil Size Measurements." *Behavior Research Methods* 48 (2): 510–27. <https://doi.org/10.3758/s13428-015-0588-x>.
- Heathcote, Andrew, Yi-Shin Lin, Angus Reynolds, Luke Strickland, Matthew Gretton, and Dora Matzke. 2019. "Dynamic Models of Choice." *Behavior Research Methods* 51 (2). Springer: 961–85.
- Heathcote, Andrew, and Jonathon Love. 2012. "Linear Deterministic Accumulator Models of Simple Choice." *Frontiers in Psychology* 3: 292. <https://doi.org/10.3389/fpsyg.2012.00292>.
- Heister, Julian, Kay-Michael Würzner, and Reinhold Kliegl. 2012. "Analysing Large Datasets of Eye Movements During Reading." *Visual Word Recognition* 2: 102–30.
- Henrich, Joseph, Steven J. Heine, and Ara Norenzayan. 2010. "The Weirdest People in the World?" *Behavioral and Brain Sciences* 33 (2-3). Cambridge University Press: 61–83. <https://doi.org/10.1017/S0140525X0999152X>.
- Henry, Lionel, and Hadley Wickham. 2019. *Purrr: Functional Programming Tools*. <https://CRAN.R-project.org/package=purrr>.
- Hester, Jim, Gábor Csárdi, Hadley Wickham, Winston Chang, Martin Morgan, and Dan Tenenbaum. 2021. *Remotes: R Package Installation from Remote Repositories, Including 'Github'*. <https://CRAN.R-project.org/package=remotes>.
- Higgins, Julian, and Sally Green. 2008. *Cochrane Handbook for Systematic Reviews of Interventions*. New York: Wiley-Blackwell.
- Hoffman, Matthew D., and Andrew Gelman. 2014. "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo." *Journal of Machine Learning Research* 15 (1): 1593–1623. <http://dl.acm.org/citation.cfm?id=2627435.2638586>.
- Hsiao, Fanny Pai-Fang, and Edward Gibson. 2003. "Processing Relative Clauses in Chinese." *Cognition* 90: 3–27.
- Izrailev, Sergei. 2014. *Tictoc: Functions for Timing R Scripts, as Well as*

Implementations of Stack and List Structures. <https://CRAN.R-project.org/package=tictoc>.

Jackman, Simon. 2009. *Bayesian Analysis for the Social Sciences*. Vol. 846. John Wiley & Sons.

JASP Team. 2019. “JASP (Version 0.11.1)[Computer software].” <https://jasp-stats.org/>.

Jaynes, Edwin T. 2003. *Probability Theory: The Logic of Science*. Cambridge university press.

Jäger, Lena A., Felix Engelmann, and Shravan Vasishth. 2017. “Similarity-Based Interference in Sentence Comprehension: Literature review and Bayesian meta-analysis.” *Journal of Memory and Language* 94:316–39. <https://doi.org/https://doi.org/10.1016/j.jml.2017.01.004>.

Jäger, Lena A, Daniela Mertzen, Julie A Van Dyke, and Shravan Vasishth. 2020. “Interference Patterns in Subject-Verb Agreement and Reflexives Revisited: A Large-Sample Study.” *Journal of Memory and Language* 111. Elsevier: 104063.

Jeffreys, Harold. 1939. *Theory of Probability*. Oxford: Clarendon Press.

Just, Marcel A., and Patricia A. Carpenter. 1992. “A Capacity Theory of Comprehension: Individual Differences in Working Memory.” *Pr* 99(1): 122–49.

Kadane, Joseph, and Lara J Wolfson. 1998. “Experiences in Elicitation: [Read Before the Royal Statistical Society at a Meeting on’Elicitation ‘on Wednesday, April 16th, 1997, the President, Professor Afm Smith in the Chair].” *Journal of the Royal Statistical Society: Series D (the Statistician)* 47 (1). Wiley Online Library: 3–19.

Kass, Robert E, and Joel B Greenhouse. 1989. “[Investigating Therapies of Potentially Great Benefit: ECMO]: Comment: A Bayesian Perspective.” *Statistical Science* 4 (4). JSTOR: 310–17.

Kass, Robert E, and Adrian E Raftery. 1995. “Bayes Factors.” *Journal of the American Statistical Association* 90 (430). Taylor & Francis: 773–95.

Kerns, G.J. 2014. *Introduction to Probability and Statistics Using R*. Second Edition.

- Keuleers, Emmanuel, Paula Lacey, Kathleen Rastle, and Marc Brysbaert. 2012. “The British Lexicon Project: Lexical Decision Data for 28,730 Monosyllabic and Disyllabic English Words.” *Behavior Research Methods* 44 (1). Springer: 287–304.
- Kolmogorov, Andrei Nikolaevich. 1933. *Foundations of the Theory of Probability: Second English Edition*. Courier Dover Publications.
- Koster, Jeremy, and Richard McElreath. 2017. “Multinomial Analysis of Behavior: Statistical Methods.” *Behavioral Ecology and Sociobiology* 71 (9): 138. <https://doi.org/10.1007/s00265-017-2363-8>.
- Kruschke, John. 2014. *Doing Bayesian Data Analysis: A tutorial with R, JAGS, and Stan*. Academic Press.
- Kutas, Marta, and Kara D. Federmeier. 2011. “Thirty Years and Counting: Finding Meaning in the N400 Component of the Event-Related Brain Potential (ERP).” *Annual Review of Psychology* 62 (1): 621–47. <https://doi.org/10.1146/annurev.psych.093008.131123>.
- Kutas, Marta, and Steven A Hillyard. 1980. “Reading Senseless Sentences: Brain Potentials Reflect Semantic Incongruity.” *Science* 207 (4427): 203–5. <https://doi.org/10.1126/science.7350657>.
- . 1984. “Brain Potentials During Reading Reflect Word Expectancy and Semantic Association.” *Nature* 307 (5947): 161–63. <https://doi.org/10.1038/307161a0>.
- Lago, Sol, Diego Shalom, Mariano Sigman, Ellen F Lau, and Colin Phillips. 2015. “Agreement Processes in Spanish Comprehension.” *Journal of Memory and Language* 82: 133–49.
- Laird, Nan M, and James H Ware. 1982. “Random-Effects Models for Longitudinal Data.” *Biometrics*. JSTOR, 963–74.
- Lambert, Ben. 2018. *A Student’s Guide to Bayesian Statistics*. Sage.
- Landau, William Michael. 2021. “The Stantargets R Package: A Workflow Framework for Efficient Reproducible Stan-Powered Bayesian Data Analysis Pipelines.” *Journal of Open Source Software* 6 (60): 3193. <https://doi.org/10.21105/joss.03193>.
- Laurinavichyute, Anna. 2020. “Similarity-Based Interference and Faulty

- Encoding Accounts of Sentence Processing." Dissertation, University of Potsdam.
- Lee, Michael D. 2011. "How Cognitive Modeling Can Benefit from Hierarchical Bayesian Models." *Journal of Mathematical Psychology* 55 (1). Elsevier BV: 1–7. <https://doi.org/10.1016/j.jmp.2010.08.013>.
- Lee, Michael D, Jason R Bock, Isaiah Cushman, and William R Shankle. 2020. "An Application of Multinomial Processing Tree Models and Bayesian Methods to Understanding Memory Impairment." *Journal of Mathematical Psychology* 95. Elsevier: 102328.
- Lee, Michael D., and Eric-Jan Wagenmakers. 2014. *Bayesian Cognitive Modeling: A Practical Course*. Cambridge University Press.
- Lee, Peter M. 2012. *Bayesian Statistics: An Introduction*. John Wiley & Sons.
- Levy, Deborah L, Philip S Holzman, Steven Matthysse, and Nancy R Mendell. 1993. "Eye Tracking Dysfunction and Schizophrenia: A Critical Perspective." *Schizophrenia Bulletin* 19 (3). Oxford University Press: 461–536.
- Lewandowski, Daniel, Dorota Kurowicka, and Harry Joe. 2009. "Generating Random Correlation Matrices Based on Vines and Extended Onion Method." *Journal of Multivariate Analysis* 100 (9): 1989–2001.
- Lewis, Richard L., and Shravan Vasishth. 2005. "An Activation-Based Model of Sentence Processing as Skilled Memory Retrieval." *Cognitive Science* 29: 1–45.
- Lidstone, George James. 1920. "Note on the General Case of the Bayes-Laplace Formula for Inductive or a Posteriori Probabilities." *Transactions of the Faculty of Actuaries* 8 (182–192): 13.
- Limpert, Eckhard, Werner A. Stahel, and Markus Abbt. 2001. "Log-Normal Distributions Across the Sciences: Keys and Clues." *BioScience* 51 (5): 341. [https://doi.org/10.1641/0006-3568\(2001\)051%5B0341:LNDATS%5D2.0.CO;2](https://doi.org/10.1641/0006-3568(2001)051[0341:LNDATS]2.0.CO;2) ([https://doi.org/10.1641/0006-3568\(2001\)051%5B0341:LNDATS%5D2.0.CO;2](https://doi.org/10.1641/0006-3568(2001)051%5B0341:LNDATS%5D2.0.CO;2)).
- Lindley, Dennis V. 1991. *Making Decisions*. Second. John Wiley & Sons.
- Lissón, Paula, Dorothea Pregla, Bruno Nicenboim, Dario Paape, Mick van

- het Nederend, Frank Burchert, Nicole Stadie, David Caplan, and Shravan Vasishth. 2021. “A Computational Evaluation of Two Models of Retrieval Processes in Sentence Processing in Aphasia.” *Cognitive Science*. <https://psyarxiv.com/r7dn5>.
- Logacev, Pavel, and Noyan Dokudan. 2021. “A Multinomial Processing Tree Model of RC Attachment.” In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics*, 39–47. Online: Association for Computational Linguistics. <https://www.aclweb.org/anthology/2021.cmcl-1.4>.
- Logačev, Pavel, and Shravan Vasishth. 2016. “A Multiple-Channel Model of Task-Dependent Ambiguity Resolution in Sentence Comprehension.” *Cognitive Science* 40 (2): 266–98. <https://doi.org/10.1111/cogs.12228>.
- Luce, R Duncan. 1991. *Response Times: Their Role in Inferring Elementary Mental Organization*. Oxford University Press.
- Lunn, David, Chris Jackson, David J Spiegelhalter, Nicky Best, and Andrew Thomas. 2012. *The BUGS Book: A Practical Introduction to Bayesian Analysis*. Vol. 98. CRC Press.
- Lunn, D.J., A. Thomas, N. Best, and D. Spiegelhalter. 2000. “WinBUGS-A Bayesian Modelling Framework: Concepts, Structure, and Extensibility.” *Statistics and Computing* 10 (4). Springer: 325–37.
- Lynch, Scott Michael. 2007. *Introduction to Applied Bayesian Statistics and Estimation for Social Scientists*. Springer.
- MacKay, David JC. 2003. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press.
- MacLeod, Colin M. 1991. “Half a Century of Research on the Stroop Effect: An Integrative Review.” *Psychological Bulletin* 109 (2). American Psychological Association: 163.
- Mahajan, Sanjoy. 2010. *Street-Fighting Mathematics: The Art of Educated Guessing and Opportunistic Problem Solving*. The MIT Press.
- . 2014. *The Art of Insight in Science and Engineering: Mastering Complexity*. The MIT Press.
- Mahowald, Kyle, Ariel James, Richard Futrell, and Edward Gibson. 2016.

- "A Meta-Analysis of Syntactic Priming in Language Production." *Journal of Memory and Language* 91. Elsevier: 5–27.
- Mathot, Sebastiaan. 2018. "Pupilometry: Psychology, Physiology, and Function." *Journal of Cognition* 1 (1): 16. <https://doi.org/10.5334/joc.18>.
- Matuschek, Hannes, Reinhold Kliegl, Shravan Vasishth, R. Harald Baayen, and Douglas M Bates. 2017. "Balancing Type I Error and Power in Linear Mixed Models." *Journal of Memory and Language* 94: 305–15. <https://doi.org/10.1016/j.jml.2017.01.001>.
- Matzke, Dora, Conor V. Dolan, William H. Batchelder, and Eric-Jan Wagenaars. 2015. "Bayesian Estimation of Multinomial Processing Tree Models with Heterogeneity in Participants and Items." *Psychometrika* 80 (1): 205–35. <https://doi.org/10.1007/s11336-013-9374-9>.
- Maxwell, Scott E, Harold D Delaney, and Ken Kelley. 2017. *Designing Experiments and Analyzing Data: A Model Comparison Perspective*. Routledge.
- McClelland, James L. 2009. "The Place of Modeling in Cognitive Science." *Topics in Cognitive Science* 1 (1): 11–38. <https://doi.org/10.1111/j.1756-8765.2008.01003.x>.
- McElreath, Richard. 2015. *Statistical Rethinking: A Bayesian Course with R Examples*. Chapman; Hall/CRC.
- McElree, Brian. 2000. "Sentence Comprehension Is Mediated by Content-Addressable Memory Structures." *Journal of Psycholinguistic Research* 29 (2). Springer: 111–23.
- McLean, Mathew William. 2017. "RefManageR: Import and Manage BibTeX and BibLaTeX References in R." *The Journal of Open Source Software*. <https://doi.org/10.21105/joss.00338>.
- Meng, Xiao-li, and Wing Hung Wong. 1996. "Simulating Ratios of Normalizing Constants via a Simple Identity: A Theoretical Exploration." *Statistica Sinica*, 831–60.
- Miller, I., and M. Miller. 2004. *John E. Freund's Mathematical Statistics with Applications*. Prentice Hall.
- Monnahan, Cole C., James T. Thorson, and Trevor A. Branch. 2017. "Faster Estimation of Bayesian Models in Ecology Using Hamiltonian Monte

- Carlo." Edited by Robert B. O'Hara. *Methods in Ecology and Evolution* 8 (3): 339–48. <https://doi.org/10.1111/2041-210X.12681>.
- Montgomery, D. C., E. A. Peck, and G. G. Vining. 2012. *An Introduction to Linear Regression Analysis*. 5th ed. Hoboken, NJ: Wiley.
- Morin, David J. 2016. *Probability: For the Enthusiastic Beginner*. Createspace Independent Publishing Platform.
- Müller, Kirill, and Hadley Wickham. 2020. *Tibble: Simple Data Frames*. <https://CRAN.R-project.org/package=tibble>.
- Navarro, Daniel. 2015. *Learning Statistics with R*. <https://learningstatisticswithr.com>.
- Navarro, Danielle J. 2019. "Between the Devil and the Deep Blue Sea: Tensions Between Scientific Judgement and Statistical Model Selection." *Computational Brain & Behavior* 2 (1): 28–34. <https://doi.org/10.1007/s42113-018-0019-z>.
- Neal, Radford M. 2003. "Slice Sampling." *Ann. Statist.* 31 (3). The Institute of Mathematical Statistics: 705–67. <https://doi.org/10.1214/aos/1056562461>.
- . 2011. "MCMC Using Hamiltonian Dynamics." In *Handbook of Markov Chain Monte Carlo*, edited by Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. Taylor & Francis. <https://doi.org/10.1201/b10905-10>.
- Nelson, Peter R. 1981. "The Algebra of Random Variables." *Technometrics* 23 (2). Taylor & Francis: 197–98. <https://doi.org/10.1080/00401706.1981.10486266>.
- Nicenboim, Bruno. 2018. "The Implementation of a Model of Choice: The (Truncated) Linear Ballistic Accumulator." In *StanCon*. Aalto University, Helsinki, Finland. <https://doi.org/10.5281/zenodo.1465990>.
- Nicenboim, Bruno, Pavel Logačev, Carolina Gattei, and Shravan Vasishth. 2016. "When High-Capacity Readers Slow down and Low-Capacity Readers Speed up: Working Memory and Locality Effects." *Frontiers in Psychology* 7 (280). <https://doi.org/10.3389/fpsyg.2016.00280>.
- Nicenboim, Bruno, Timo B. Roettger, and Shravan Vasishth. 2018. "Us-

- ing Meta-Analysis for Evidence Synthesis: The case of incomplete neutralization in German.” *Journal of Phonetics* 70: 39–55. <https://doi.org/https://doi.org/10.1016/j.wocn.2018.06.001>.
- Nicenboim, Bruno, Daniel Schad, and Shravan Vasishth. 2020. *Bcogsci: Data and Models for the Book “an Introduction to Bayesian Data Analysis for Cognitive Science”*.
- Nicenboim, Bruno, and Shravan Vasishth. 2016. “Statistical methods for linguistic research: Foundational Ideas - Part II.” *Language and Linguistics Compass* 10 (11): 591–613. <https://doi.org/10.1111/lnc3.12207>.
- . 2018. “Models of Retrieval in Sentence Comprehension: A Computational Evaluation Using Bayesian Hierarchical Modeling.” *Journal of Memory and Language* 99: 1–34. <https://doi.org/10.1016/j.jml.2017.08.004>.
- Nicenboim, Bruno, Shravan Vasishth, Felix Engelmann, and Katja Suckow. 2018. “Exploratory and Confirmatory Analyses in Sentence Processing: A case study of number interference in German.” *Cognitive Science* 42 (S4). <https://doi.org/10.1111/cogs.12589>.
- Nicenboim, Bruno, Shravan Vasishth, and Frank Rösler. 2020a. “Are Words Pre-Activated Probabilistically During Sentence Comprehension? Evidence from New Data and a Bayesian Random-Effects Meta-Analysis Using Publicly Available Data.” *Neuropsychologia* 142. <https://doi.org/10.1016/j.neuropsychologia.2020.107427>.
- . 2020b. “Are Words Pre-Activated Probabilistically During Sentence Comprehension? Evidence from New Data and a Bayesian Random-Effects Meta-Analysis Using Publicly Available Data.” *Neuropsychologia*, 107427.
- Nieuwland, Mante S, Stephen Politzer-Ahles, Evelien Heyselaar, Katrien Segaert, Emily Darley, Nina Kazanina, Sarah Von Grebmer Zu Wolfsthurn, et al. 2018. “Large-Scale Replication Study Reveals a Limit on Probabilistic Prediction in Language Comprehension.” *eLife* 7. <https://doi.org/10.7554/eLife.33468>.
- Normand, S.L.T. 1999. “Tutorial in Biostatistics Meta-Analysis: Formulat-

- ing, Evaluating, Combining, and Reporting." *Statistics in Medicine* 18 (3): 321–59.
- Oakley, J. E., and A. O'Hagan. 2010. *SHELF: The Sheffield Elicitation Framework (version 2.0)*. University of Sheffield, UK: School of Mathematics; Statistics, University of Sheffield. <http://tonyohagan.co.uk/shelf>.
- Oberauer, Klaus. 2019. "Working Memory Capacity Limits Memory for Bindings." *Journal of Cognition* 2 (1): 40. <https://doi.org/10.5334/joc.86>.
- Oberauer, Klaus, and Reinhold Kliegl. 2001. "Beyond Resources: Formal Models of Complexity Effects and Age Differences in Working Memory." *European Journal of Cognitive Psychology* 13 (1-2). Routledge: 187–215. <https://doi.org/10.1080/09541440042000278>.
- O'Hagan, Anthony, Caitlin E Buck, Alireza Daneshkhah, J Richard Eiser, Paul H Garthwaite, David J Jenkinson, Jeremy E Oakley, and Tim Rakow. 2006. *Uncertain Judgements: Eliciting Experts' Probabilities*. John Wiley & Sons.
- O'Hagan, Antony, and Jonathan Forster. 2004. "Kendall's Advanced Theory of Statistics, Vol. 2B: Bayesian Inference." Wiley.
- Ollman, Robert. 1966. "Fast Guesses in Choice Reaction Time." *Psychonomic Science* 6 (4). Springer: 155–56.
- Ooms, Jeroen. 2021. *Pdftools: Text Extraction, Rendering and Converting of Pdf Documents*. <https://CRAN.R-project.org/package=pdftools>.
- Paananen, Topi, Juho Piironen, Paul-Christian Bürkner, and Aki Vehtari. 2021. "Implicitly Adaptive Importance Sampling." *Statistics and Computing* 31 (2). Springer Science; Business Media LLC. <https://doi.org/10.1007/s11222-020-09982-2>.
- Paape, Dario, Serine Avetisyan, Sol Lago, and Shravan Vasishth. 2021. "Modeling Misretrieval and Feature Substitution in Agreement Attraction: A Computational Evaluation." *Cognitive Science*. <https://psyarxiv.com/957e3/>.
- Paape, Dario, Bruno Nicenboim, and Shravan Vasishth. 2017. "Does Antecedent Complexity Affect Ellipsis Processing? An Empirical Investigation." *Glossa: A Journal of General Linguistics* 2 (1).

- Paolacci, Gabriele, Jesse Chandler, and Panagiotis G Ipeirotis. 2010. “Running Experiments on Amazon Mechanical Turk.” *Judgment and Decision Making* 5 (5): 411–19.
- Papaspiliopoulos, Omiros, Gareth O. Roberts, and Martin Sköld. 2007. “A General Framework for the Parametrization of Hierarchical Models.” *Statist. Sci.* 22 (1). The Institute of Mathematical Statistics: 59–73. <https://doi.org/10.1214/088342307000000014>.
- Phillips, Colin, Matthew W. Wagers, and Ellen F. Lau. 2011. “Grammatical Illusions and Selective Fallibility in Real-Time Language Comprehension.” In *Experiments at the Interfaces*, 37:147–80. Emerald Bingley, UK.
- Picton, T.W., S. Bentin, P. Berg, E. Donchin, S.A. Hillyard, R. Johnson JR., G.A. Miller, et al. 2000. “Guidelines for Using Human Event-Related Potentials to Study Cognition: Recording Standards and Publication Criteria.” *Psychophysiology* 37 (2): 127–52. <https://doi.org/10.1111/1469-8986.3720127>.
- Piironen, Juho, Markus Paasiniemi, and Aki Vehtari. 2020. “Projective inference in high-dimensional problems: Prediction and feature selection.” *Electronic Journal of Statistics* 14 (1). Institute of Mathematical Statistics; Bernoulli Society: 2155–97. <https://doi.org/10.1214/20-EJS1711>.
- Piironen, Juho, and Aki Vehtari. 2017. “Comparison of Bayesian Predictive Methods for Model Selection.” *Statistics and Computing* 27 (3): 711–35. <https://doi.org/10.1007/s11222-016-9649-y>.
- Pitt, Mark A., and In Jae Myung. 2002. “When a Good Fit Can Be Bad.” *Trends in Cognitive Sciences* 6 (10): 421–25. [https://doi.org/10.1016/S1364-6613\(02\)01964-2](https://doi.org/10.1016/S1364-6613(02)01964-2).
- Plummer, Martin. 2016. “JAGS Version 4.2.0 User Manual.”
- Pullin, Jeffrey, Lyle Gurrin, and Damjan Vukcevic. 2021. “Statistical Models of Repeated Categorical Ratings: The R Package Rater.”
- Pylyshyn, Zenon W., and Ron W. Storm. 1988. “Tracking Multiple Independent Targets: Evidence for a Parallel Tracking Mechanism.” *Spatial Vision* 3 (3): 179–97. <https://doi.org/10.1163/156856888X00122>.
- Rabe, Maximilian M., Shravan Vasishth, Sven Hohenstein, Reinhold

- Kliegl, and Daniel J. Schad. 2020. "Hypr: An R Package for Hypothesis-Driven Contrast Coding." *The Journal of Open Source Software*. <https://doi.org/10.21105/joss.02134>.
- Rabe, Maximilian M, Shravan Vasishth, Sven Hohenstein, Reinhold Kliegl, and Daniel J Schad. 2020. "Hypr: An R Package for Hypothesis-Driven Contrast Coding." *Journal of Open Source Software* 5 (48): 2134.
- Ratcliff, Roger. 1978. "A Theory of Memory Retrieval." *Psychological Review* 85 (2). American Psychological Association: 59.
- Ratcliff, Roger, Philip L. Smith, Scott D. Brown, and Gail McKoon. 2016. "Diffusion Decision Model: Current Issues and History." *Trends in Cognitive Sciences* 20 (4): 260–81. <https://doi.org/https://doi.org/10.1016/j.tics.2016.01.007>.
- Ratcliff, Roger, and Francis Tuerlinckx. 2002. "Estimating Parameters of the Diffusion Model: Approaches to Dealing with Contaminant Reaction Times and Parameter Variability." *Psychonomic Bulletin & Review* 9 (3). Springer: 438–81.
- Raymond, Jane E, Kimron L Shapiro, and Karen M Arnell. 1992. "Temporary Suppression of Visual Processing in an RSVP Task: An Attentional Blink?" *Journal of Experimental Psychology: Human Perception and Performance* 18 (3). American Psychological Association: 849.
- Rayner, K. 1998. "Eye movements in reading and information processing: 20 years of research." *Psychological Bulletin* 124 (3): 372–422.
- R Core Team. 2019. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <https://www.R-project.org/>.
- Reali, Florencia, and Morten H Christiansen. 2007. "Processing of Relative Clauses Is Made Easier by Frequency of Occurrence." *Journal of Memory and Language* 57 (1). Elsevier: 1–23.
- Ripley, Brian. 2019. *MASS: Support Functions and Datasets for Venables and Ripley's Mass*. <https://CRAN.R-project.org/package=MASS>.
- Roberts, Seth, and Harold Pashler. 2000. "How Persuasive Is a Good Fit? A Comment on Theory Testing." *Psychological Review* 107 (2): 358–67.

- Rosenthal, Robert, Ralph L Rosnow, and Donald B Rubin. 2000. *Contrasts and Effect Sizes in Behavioral Research: A Correlational Approach*. Cambridge University Press.
- Ross, Sheldon. 2002. *A First Course in Probability*. Pearson Education.
- Rouder, Jeffrey N. 2005. “Are Unshifted Distributional Models Appropriate for Response Time?” *Psychometrika* 70 (2). Springer Science + Business Media: 377–81. <https://doi.org/10.1007/s11336-005-1297-7>.
- Rouder, Jeffrey N., Julia M Haaf, and Joachim Vandekerckhove. 2018. “Bayesian Inference for Psychology, Part Iv: Parameter Estimation and Bayes Factors.” *Psychonomic Bulletin & Review* 25 (1): 102–13.
- Rouder, Jeffrey N., Jordan M. Province, Richard D. Morey, Pablo Gomez, and Andrew Heathcote. 2015. “The Lognormal Race: A Cognitive-Process Model of Choice and Latency with Desirable Psychometric Properties.” *Psychometrika* 80 (2): 491–513. <https://doi.org/10.1007/s11336-013-9396-3>.
- Rouder, Jeffrey N, Paul L Speckman, Dongchu Sun, Richard D Morey, and Geoffrey Iverson. 2009. “Bayesian T Tests for Accepting and Rejecting the Null Hypothesis.” *Psychonomic Bulletin & Review* 16 (2): 225–37.
- Royall, Richard. 1997. *Statistical Evidence: A Likelihood Paradigm*. New York: Chapman; Hall, CRC Press.
- Safavi, Molood Sadat, Samar Husain, and Shravan Vasishth. 2016. “Dependency Resolution Difficulty Increases with Distance in Persian Separable Complex Predicates: Implications for Expectation and Memory-Based Accounts.” *Frontiers in Psychology* 7 (403).
- Salvatier, John, Thomas V. Wiecki, and Christopher Fonnesbeck. 2016. “Probabilistic Programming in Python Using PyMC3.” *PeerJ Computer Science* 2 (April). PeerJ: e55. <https://doi.org/10.7717/peerj-cs.55>.
- Schad, Daniel J., Michael J. Betancourt, and Shravan Vasishth. 2019. “Toward a Principled Bayesian Workflow in Cognitive Science.” *arXiv Preprint arXiv:1904.12765*.
- . 2020. “Toward a Principled Bayesian Workflow in Cognitive Sci-

- ence." *Psychological Methods* 26 (1). American Psychological Association: 103–26.
- Schad, Daniel J., Bruno Nicenboim, Paul-Christian Bürkner, Michael J. Betancourt, and Shravan Vasishth. 2021. "Workflow Techniques for the Robust Use of Bayes Factors."
- Schad, Daniel J., Shravan Vasishth, Sven Hohenstein, and Reinhold Kliegl. 2019. "How to Capitalize on a Priori Contrasts in Linear (Mixed) Models: A Tutorial." *Journal of Memory and Language* 110. <https://doi.org/10.1016/j.jml.2019.104038>.
- . 2020. "How to Capitalize on a Priori Contrasts in Linear (Mixed) Models: A Tutorial." *Journal of Memory and Language* 110. Elsevier: 104038.
- Schönbrodt, Felix D, and Eric-Jan Wagenmakers. 2018. "Bayes Factor Design Analysis: Planning for Compelling Evidence." *Psychonomic Bulletin & Review* 25 (1): 128–42.
- Shiffrin, Richard, Michael Lee, Woojae Kim, and Eric-Jan Wagenmakers. 2008. "A Survey of Model Evaluation Approaches with a Tutorial on Hierarchical Bayesian Methods." *Cognitive Science: A Multidisciplinary Journal* 32 (8): 1248–84. <https://doi.org/10.1080/03640210802414826>.
- Sidi, Jonathan, and Daniel Polhamus. 2020. *TexPreview: Compile and Preview Snippets of 'Latex'*. <https://CRAN.R-project.org/package=texPreview>.
- Simpson, Daniel, Håvard Rue, Andrea Riebler, Thiago G. Martins, and Sigrunn H. Sørbye. 2017. "Penalising Model Component Complexity: A Principled, Practical Approach to Constructing Priors." *Statistical Science* 32 (1): 1–28. <https://doi.org/10.1214/16-STS576>.
- Singmann, Henrik, Ben Bolker, Jake Westfall, Frederik Aust, and Mattan S. Ben-Shachar. 2020. *Afex: Analysis of Factorial Experiments*. <https://CRAN.R-project.org/package=afex>.
- Sivula, Tuomas, Måns Magnusson, and Aki Vehtari. 2020. "Uncertainty in Bayesian Leave-One-Out Cross-Validation Based Model Comparison."
- Smith, Jared B, and William H Batchelder. 2010. "Beta-MPT: Multinomial Processing Tree Models for Addressing Individual Differences." *Journal of Mathematical Psychology* 54 (1). Elsevier: 167–83.

- Sorensen, Tanner, Sven Hohenstein, and Shravan Vasishth. 2016. “Bayesian Linear Mixed Models Using Stan: A Tutorial for Psychologists, Linguists, and Cognitive Scientists.” *Quantitative Methods for Psychology* 12 (3): 175–200. <http://www.ling.uni-potsdam.de/~vasishth/statistics/BayesLMMs.html>.
- Spector, Robert H. 1990. “The Pupils.” In *Clinical Methods: The History, Physical, and Laboratory Examinations*, edited by H. Kenneth Walker, W. Dallas Hall, and J. Willis Hurst, 3rd ed. Boston: Butterworths.
- Spiegelhalter, David J, Keith R Abrams, and Jonathan P Myles. 2004. *Bayesian Approaches to Clinical Trials and Health-Care Evaluation*. Vol. 13. John Wiley & Sons.
- Spurdle, Abby. 2020a. *Barsurf: Heatmap-Related Plots and Smooth Multiband Color Interpolation*. <https://CRAN.R-project.org/package=barsurf>.
- . 2020b. *Bivariate: Bivariate Probability Distributions*. <https://CRAN.R-project.org/package=bivariate>.
- Spurdle, Abby, and Emil Bode. 2020. *Intoo: Minimal Language-Like Extensions*. <https://CRAN.R-project.org/package=intoo>.
- Stan Development Team. 2021. “Stan Modeling Language Users Guide and Reference Manual, Version 2.27.” <https://mc-stan.org>.
- Stroop, J Ridley. 1935. “Studies of Interference in Serial Verbal Reactions.” *Journal of Experimental Psychology* 18 (6). Psychological Review Company: 643.
- Sutton, Alexander J, Nicky J Welton, Nicola Cooper, Keith R Abrams, and AE Ades. 2012. *Evidence Synthesis for Decision Making in Healthcare*. Vol. 132. John Wiley & Sons.
- Szollosi, Aba, David Kellen, Danielle Navarro, Richard Shiffrin, Iris van Rooij, Trisha Van Zandt, and Chris Donkin. 2019. “Is Preregistration Worthwhile?” PsyArXiv.
- Talts, Sean, Michael J. Betancourt, Daniel Simpson, Aki Vehtari, and Andrew Gelman. 2018. “Validating Bayesian Inference Algorithms with Simulation-Based Calibration.” *arXiv Preprint arXiv:1804.06788*.
- Turner, Brandon M, Per B Sederberg, Scott D Brown, and Mark Steyvers.

2013. "A Method for Efficiently Sampling from Distributions with Correlated Dimensions." *Psychological Methods* 18 (3). American Psychological Association: 368.
- Turner, R.M., D.J. Spiegelhalter, G. Smith, and S.G. Thompson. 2008. "Bias Modelling in Evidence Synthesis." *Journal of the Royal Statistical Society: Series A (Statistics in Society)* 172 (1). Wiley Online Library: 21–47.
- Tversky, Amos, and Daniel Kahneman. 1983. "Extensional Versus Intuitive Reasoning: The Conjunction Fallacy in Probability Judgment." *Psychological Review* 90 (4). American Psychological Association: 293.
- Ulrich, Rolf, and Jeff Miller. 1993. "Information Processing Models Generating Lognormally Distributed Reaction Times." *Journal of Mathematical Psychology* 37 (4): 513–25. <https://doi.org/10.1006/jmps.1993.1032>.
- . 1994. "Effects of Truncation on Reaction Time Analysis." *Journal of Experimental Psychology: General* 123 (1): 34–80. <https://doi.org/10/b8tsnh>.
- Vaidyanathan, Ramnath, Yihui Xie, JJ Allaire, Joe Cheng, and Kenton Russell. 2018. *Htmlwidgets: HTML Widgets for R*. <https://CRAN.R-project.org/package=htmlwidgets>.
- Vasishth, Shravan. 2015. "A Meta-Analysis of Relative Clause Processing in MANDARIN CHINESE Using Bias Modelling." Master's thesis, Sheffield, UK: School of Mathematics; Statistics, University of Sheffield. <http://www.ling.uni-potsdam.de/~vasishth/pdfs/VasishthMScStatistics.pdf>.
- Vasishth, Shravan, Sven Bruessow, Richard L. Lewis, and Heiner Drenhaus. 2008. "Processing Polarity: How the Ungrammatical Intrudes on the Grammatical." *Cognitive Science* 32 (4, 4): 685–712.
- Vasishth, Shravan, Zhong Chen, Qiang Li, and Gueilan Guo. 2013. "Processing Chinese Relative Clauses: Evidence for the Subject-Relative Advantage." *PLoS ONE* 8 (10). Public Library of Science: 1–14.
- Vasishth, Shravan, Nicolas Chopin, Robin Ryder, and Bruno Nicenboim. 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*. London, UK. <https://arxiv.org/abs/1702.00564v2>.
- Vasishth, Shravan, and Felix Engelmann. 2021. *Sentence Comprehension as a Cognitive Process: A Computational Approach*. Cambridge, UK: Cambridge University Press. <https://books.google.de/books?id=6KZKzgFACAAJ>.
- Vasishth, Shravan, Daniela Mertzen, Lena A Jäger, and Andrew Gelman. 2018. “The Statistical Significance Filter Leads to Overoptimistic Expectations of Replicability.” *Journal of Memory and Language* 103: 151–75.
- Vasishth, Shravan, and Bruno Nicenboim. 2016. “Statistical Methods for Linguistic Research: Foundational Ideas – Part I.” *Language and Linguistics Compass* 10 (8): 349–69.
- Vasishth, Shravan, Bruno Nicenboim, Mary E. Beckman, Fangfang Li, and Eun Jong Kong. 2018. “Bayesian Data Analysis in the Phonetic Sciences: A Tutorial Introduction.” *Journal of Phonetics* 71: 141–61. <https://doi.org/10.1016/j.wocn.2018.07.008>.
- Vasishth, Shravan, Daniel Schad, Audrey Bürki-Forschini, and Reinhold Kliegl. 2021a. *Lingpsych: Data and Functions Used in the Book "Linear Mixed Models in Linguistics and Psychology: A Comprehensive Introduction"*.
- Vasishth, Shravan, Daniel J. Schad, Audrey Bürki, and Reinhold Kliegl. 2021b. *Linear Mixed Models for Linguistics and Psychology: A Comprehensive Introduction*. CRC Press. https://vasishth.github.io/Freq_CogSci/.
- Vasishth, Shravan, Katja Suckow, Richard L. Lewis, and Sabine Kern. 2011. “Short-Term Forgetting in Sentence Comprehension: Crosslinguistic Evidence from Head-Final Structures.” *Language and Cognitive Processes* 25: 533–67.
- Vasishth, S., L. A. Jaeger, and B. Nicenboim. 2017. “Feature overwriting as a finite mixture process: Evidence from comprehension data.” In *Proceedings of Mathpsych/Iccm Conference*. Warwick, UK. <https://arxiv.org/abs/1703.04081>.
- Vehtari, Aki, and Andrew Gelman. 2015. “Pareto Smoothed Importance Sampling.” *arXiv Preprint arXiv:1507.02646*.
- Vehtari, Aki, Andrew Gelman, and Jonah Gabry. 2017a. “Prac-

- tical Bayesian Model Evaluation Using Leave-One-Out Cross-Validation and Waic.” *Statistics and Computing* 27 (5): 1413–32. <https://doi.org/10.1007/s11222-016-9696-4>.
- . 2017b. “Practical Bayesian Model Evaluation Using Leave-One-Out Cross-Validation and WAIC.” *Statistics and Computing* 27 (5): 1413–32. <https://doi.org/10.1007/s11222-016-9696-4>.
- Vehtari, Aki, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner. 2019. “Rank-Normalization, Folding, and Localization: An Improved \hat{R} for Assessing Convergence of Mcmc.”
- Vehtari, Aki, and Jouko Lampinen. 2002. “Bayesian Model Assessment and Comparison Using Cross-Validation Predictive Densities.” *Neural Computation* 14 (10): 2439–68. <https://doi.org/10.1162/08997660260293292>.
- Vehtari, Aki, and Janne Ojanen. 2012. “A Survey of Bayesian Predictive Methods for Model Assessment, Selection and Comparison.” *Statist. Surv.* 6 (O). Institute of Mathematical Statistics: 142–228. <https://doi.org/10.1214/12-ss102>.
- Vehtari, Aki, Daniel P. Simpson, Yuling Yao, and Andrew Gelman. 2019. “Limitations of ‘Limitations of Bayesian Leave-One-Out Cross-Validation for Model Selection’.” *Computational Brain & Behavior* 2 (1): 22–27. <https://doi.org/10.1007/s42113-018-0020-6>.
- Venables, William N., and Brian D. Ripley. 2002. *Modern Applied Statistics with S-PLUS*. New York: Springer.
- Verhagen, Josine, and Eric-Jan Wagenmakers. 2014. “Bayesian Tests to Quantify the Result of a Replication Attempt.” *Journal of Experimental Psychology: General* 143 (4): 1457–75. <https://doi.org/10.1037/a0036731>.
- Von Baeyer, Hans Christian. 1988. “How Fermi Would Have Fixed It.” *The Sciences* 28 (5). Blackwell Publishing Ltd Oxford, UK: 2–4.
- Wagenmakers, Eric-Jan, and Scott Brown. 2007. “On the Linear Relation Between the Mean and the Standard Deviation of a Response Time Distribution.” *Psychological Review* 114 (3). American Psychological Association: 830.

- Wagenmakers, Eric-Jan, Raoul P. P. Grasman, and Peter C. M. Molenaar. 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. <https://doi.org/10.1016/j.jmp.2005.02.003>.
- Wagenmakers, Eric-Jan, Michael David Lee, Jeffrey N. Rouder, and Richard Donald Morey. 2019. “The Principle of Predictive Irrelevance, or Why Intervals Should Not Be Used for Model Comparison Featuring a Point Null Hypothesis.” Preprint. PsyArXiv. <https://doi.org/10.31234/osf.io/rqnu5>.
- Wagenmakers, Eric-Jan, Tom Lodewyckx, Himanshu Kuriyal, and Raoul Grasman. 2010. “Bayesian Hypothesis Testing for Psychologists: A Tutorial on the Savage–Dickey Method.” *Cognitive Psychology* 60 (3). Elsevier: 158–89.
- Wahn, Basil, Daniel P. Ferris, W. David Hairston, and Peter König. 2016. “Pupil Sizes Scale with Attentional Load and Task Experience in a Multiple Object Tracking Task.” *PLOS ONE* 11 (12): e0168087. <https://doi.org/10.1371/journal.pone.0168087>.
- Walker, Grant M, Gregory Hickok, and Julius Fridriksson. 2018. “A Cognitive Psychometric Model for Assessment of Picture Naming Abilities in Aphasia.” *Psychological Assessment* 6. American Psychological Association: 809–26. <https://doi.org/10.1037/pas0000529>.
- Wang, Wei, and Andrew Gelman. 2014. “Difficulty of Selecting Among Multilevel Models Using Predictive Accuracy.” *Statistics at Its Interface* 7: 1–8.
- Wickelgren, Wayne A. 1977. “Speed-Accuracy Tradeoff and Information Processing Dynamics.” *Acta Psychologica* 41 (1): 67–85.
- Wickelmaier, Florian, and Achim Zeileis. 2018. “Using Recursive Partitioning to Account for Parameter Heterogeneity in Multinomial Processing Tree Models.” *Behavior Research Methods* 50 (3). Springer: 1217–33.
- Wickham, Hadley. 2019a. *Forcats: Tools for Working with Categorical Variables (Factors)*. <https://CRAN.R-project.org/package=forcats>.
- . 2019b. *Stringr: Simple, Consistent Wrappers for Common String Operations*. <https://CRAN.R-project.org/package=stringr>.

- Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. <https://doi.org/10.21105/joss.01686>.
- Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, and Hiroaki Yutani. 2019. *Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. <https://CRAN.R-project.org/package=ggplot2>.
- Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller. 2019. *Dplyr: A Grammar of Data Manipulation*. <https://CRAN.R-project.org/package=dplyr>.
- Wickham, Hadley, and Lionel Henry. 2019. *Tidyr: Tidy Messy Data*. <https://CRAN.R-project.org/package=tidyr>.
- Wickham, Hadley, Jim Hester, and Romain Francois. 2018. *Readr: Read Rectangular Text Data*. <https://CRAN.R-project.org/package=readr>.
- Wilke, Claus O. 2020. *Cowplot: Streamlined Plot Theme and Plot Annotations for 'Ggplot2'*. <https://CRAN.R-project.org/package=cowplot>.
- Wilson, Greg, Jennifer Bryan, Karen Cranston, Justin Kitzes, Lex Nederbragt, and Tracy K Teal. 2017. "Good Enough Practices in Scientific Computing." *PLoS Computational Biology* 13 (6). Public Library of Science San Francisco, CA USA: e1005510.
- Wilson, Robert C, and Anne GE Collins. 2019. "Ten Simple Rules for the Computational Modeling of Behavioral Data." Edited by Timothy E Behrens. *eLife* 8 (November). eLife Sciences Publications, Ltd: e49547. <https://doi.org/10.7554/eLife.49547>.
- Wolodzko, Tymoteusz. 2019. *ExtraDistr: Additional Univariate and Multivariate Distributions*. <https://CRAN.R-project.org/package=extraDistr>.
- Xie, Yihui. 2019a. *Bookdown: Authoring Books and Technical Documents with R Markdown*. <https://CRAN.R-project.org/package=bookdown>.
- . 2019b. *Knitr: A General-Purpose Package for Dynamic Report Generation in R*. <https://CRAN.R-project.org/package=knitr>.

- . 2019c. *Servr: A Simple Http Server to Serve Static Files or Dynamic Documents*. <https://CRAN.R-project.org/package=servr>.
- Xie, Yihui, Joe Cheng, and Xianying Tan. 2019. *DT: A Wrapper of the Javascript Library 'Datatables'*. <https://CRAN.R-project.org/package=DT>.
- Yackulic, Charles B., Michael Dodrill, Maria Dzul, Jamie S. Sanderlin, and Janice A. Reid. 2020. “A Need for Speed in Bayesian Population Models: A Practical Guide to Marginalizing and Recovering Discrete Latent States.” *Ecological Applications* 30 (5): e02112. <https://doi.org/https://doi.org/10.1002/eap.2112>.
- Yadav, Himanshu, Dario Paape, Garrett Smith, Brian Dillon, and Shravan Vasishth. 2021. “Individual Differences in Cue-Weighting in Sentence Comprehension: An Evaluation Using Approximate Bayesian Computation.”
- Yadav, Himanshu, Garrett Smith, and Shravan Vasishth. 2021a. “Feature Encoding Modulates Cue-Based Retrieval: Modeling Interference Effects in Both Grammatical and Ungrammatical Sentences.” *Proceedings of the Cognitive Science Conference*.
- . 2021b. “Is Similarity-Based Interference Caused by Lossy Compression or Cue-Based Retrieval? A Computational Evaluation.” *Proceedings of the International Conference on Cognitive Modeling*.
- Yao, Yuling, Aki Vehtari, Daniel Simpson, and Andrew Gelman. 2017. “Using Stacking to Average Bayesian Predictive Distributions.” *Bayesian Analysis*. <https://doi.org/10.1214/17-BA1091>.
- Yarkoni, Tal. 2020. “The Generalizability Crisis.” *Behavioral and Brain Sciences*. Cambridge University Press, 1–37. <https://doi.org/10.1017/S0140525X20001685>.
- Yellott, John I. 1967. “Correction for Guessing in Choice Reaction Time.” *Psychonomic Science* 8 (8): 321–22. <https://doi.org/10.3758/BF03331682>.
- . 1971. “Correction for Fast Guessing and the Speed-Accuracy Trade-off in Choice Reaction Time.” *Journal of Mathematical Psychology* 8 (2): 159–99. [https://doi.org/10.1016/0022-2496\(71\)90011-3](https://doi.org/10.1016/0022-2496(71)90011-3).

- Zhu, Hao. 2019. *KableExtra: Construct Complex Table with 'Kable' and Pipe Syntax*. <https://CRAN.R-project.org/package=kableExtra>.

Index

probability, 3

random variable, 4