Kentaro Matsuura

Bayesian Statistical Modeling with Stan, R, and Python

Springer

Bayesian Statistical Modeling with Stan, R, and Python

Kentaro Matsuura

Bayesian Statistical Modeling with Stan, R, and Python

Springer

Kentaro Matsuura
HOXO-M Inc.
Tokyo, Japan

ISBN 978-981-19-4754-4 ISBN 978-981-19-4755-1 (eBook)
https://doi.org/10.1007/978-981-19-4755-1

© Springer Nature Singapore Pte Ltd. 2022
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

About This Book

With the recent rise in popularity of Bayesian statistics, many books on this subject have been published. However, many of these books either only contain basic information, or contain too complicated formulas that are not easy to extend in practice. This book is, therefore, a departure from those books, and is intended to be a very practical book on Bayesian statistical modeling with real-world data analysis.

This book is about *Stan*, a software that conducts statistical modeling in the framework of Bayesian statistics. We also introduce how to use its R package *CmdStanR* and Python package *CmdStanPy*.

Statistical modeling is a process of fitting mathematical models with probabilistic distributions to the observed data, in order to understand phenomena and to further make predictions. In earlier days, such a method was not considered to be a practical method for the analysis of the real-world data, due to several reasons. For instance, it is not easy to collect a large amount of data and the computational time of the method was very long. In addition, fitting a complex model needs high mathematical skills, because the analytical solution is difficult and implementing its solution is even harder.

Yet nowadays, these are not the problems anymore. Today we can obtain a large amount of data relatively easily, and computers have become powerful and efficient compared to the earlier days. Also luckily, several programming languages that are specifically designed for statistical modeling have been developed. Therefore, statistical modeling has become one of the most efficient approaches for data analysis in the recent studies.

In this book, we use a new programing language, Stan, to conduct statistical modeling. Stan incorporates an excellent algorithm, and importantly, it is under active development by an awesome community. Its R and Python packages (CmdStanR and CmdStanPy) are released in parallel, which provide an easy starting point for the beginners. In Stan, many models can be written in about 30 lines of code. This

includes not only the basic models such as multivariate regression and logistic regression, but also more advanced modeling such as hierarchical models, state-space models, Gaussian processes, and many others. Moreover, it enables the problem-specific model extensions based on the problem that every user is trying to solve for. We believe that the skillsets and the way of thinking about the real-world problem using Stan in this book will be helpful for every reader, even if the syntax of the Stan changes in the future, or one day another statistical modeling framework comes out and replaces Stan.

Chapter Structure

This book is structured in a way so that by reading from the beginning to the end, readers can obtain a systematical knowledge and skillsets for statistical modeling using Stan. The chapter structure in this book is organized as follows (see Fig. 1).

Roughly, the chapters can be divided into four large sections. Chapters 1 and 2 are the introduction, and we mainly focus on the theoretical background on statistical modeling and Bayesian inference. Chapters 3–5 are the guide of how to use Stan, specifically for the beginners. We introduce how to use Stan itself, as well as CmdStanR, and CmdStanPy through the example of simple regression analysis such as multiple linear regression and logistic regression. Chapters 6–10 are for more advanced users, and the important components and techniques to master statistical modeling are introduced. Chapters 6, 8, and 10 are the main chapters, which increase your statistical modeling approach repertoire. Chapters 7 and 9 introduce approaches for model improvement. Chapters 11–14 focus on advanced topics, which are applied and discussed quite widely when conducting the real data analysis.

We listed a summary of contents in each chapter: In Chap. 1, we give a brief introduction to statistical modeling and its features, as well as the recommended statistical modeling workflow. In Chap. 2, we concisely summarize the terminologies used in Bayesian inference and MCMC. These terminologies will be necessary to understand the later chapters. In Chap. 3, we introduce how to install Stan and describe its basic syntax. In Chap. 4, we show how to use Stan with its interface CmdStanR or CmdStanPy through an example of simple linear regression. Specifically, we discuss how to generate an MCMC sample, and obtain Bayesian confidence intervals and Bayesian prediction intervals from the MCMC sample. We also show how to explain

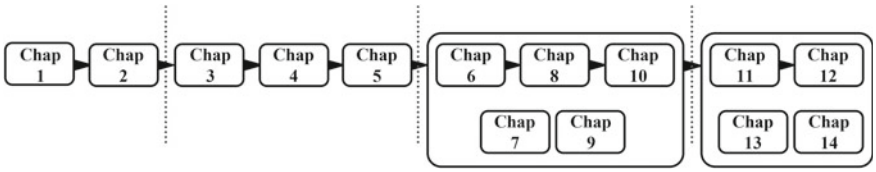


Fig. 1 A flow diagram showing the connections between chapters

the obtained inference results. In Chap. 5, we follow the statistical modeling workflow introduced in Chap. 1. The examples of multiple linear regression, logistic regression, and Poisson regression are also given. We also show examples of how to use plots to check the used models. In Chap. 6, we deviate slightly from the Stan itself and introduce basic probabilistic distributions. We mainly focus on the distributions that are used frequently in analytical practice. In Chap. 7, we summarize some potential issues and troubles when applying regression models to realistic problems. This chapter also includes the introductions on non-linear models, censored data, and data with outliers. In Chap. 8, we introduce hierarchical models (multilevel models). This type of model is used to account for group differences and individual differences. It will be one of the most widely used model types in the future. In Chap. 9, we introduce the troubleshooting approach when the MCMC does not converge, with the focus on setting the weakly informative priors. In Chap. 10, we introduce how to address discrete parameters. The current version of Stan has the limitation that is not able to use the discrete parameters, and we introduce how to solve such issues. In Chap. 11, we use state-space models for time series data. In Chap. 12, we use Markov Random Field models and Gaussian Processes for spatial data. In Chap. 13, we introduce how to efficiently use the MCMC sample from a posterior distribution (or a predictive distribution). In Chap. 14, we discuss some advanced topics, specifically we choose some examples from survival time analysis, matrix factorization, and model selection using information criteria.

Prerequired Background Knowledge

We do not require the readers to have the previous experience on statistical modeling. However, since building models and checking the models are time-consuming and labor-intensive processes, we hope the readers have the mindset and enthusiasm to enjoy such processes.

To conduct statistical modeling, it requires a wide range of knowledge and skills such as the statistics knowledge (especially probability distributions), skill to imagine the mechanisms of the observed phenomenon and to express them using mathematical formulas as well as programming languages. One of the aims of this book is to extend and improve these types of skillsets. That being said, in this book, we assume that the readers have the following basic knowledge and skillsets already as the requirement.

- The fundamental knowledge of probability and statistics: probability, probability distributions, probability density functions, conditional probability, joint probability, marginal probability, correlation coefficients. The knowledge of statistical tests is not required. The next section introduces each of these terminologies briefly, but it would also be helpful to look up some web resources including Wikipedia for those who are less familiar.
- Programming skills: You should be familiar with either R or Python. Specifically, we require the readers to be able to conduct fundamental data processing and data visualizations.

The Terminologies and Symbols Used in This Book

Sum and Products

Σ represents the summation:

$$\sum_{k=1}^{K} a_k = a_1 + a_2 + ... + a_K,$$

and Π represents the product:

$$\prod_{k=1}^{K} a_k = a_1 \times a_2 \times ... \times a_K.$$

Probability Distribution

Probability distribution is the expression of how likely a random variable takes each of the possible values. It also can be simply called as distributions. The probability distribution of random variable a is written as $p(a)$.

Probability Mass Function (PMF)

When the random variable is a discrete variable, the probability distribution is called a *probability mass function.*

Probability Density Function (PDF)

When the random variable is a continuous variable, the probability distribution is called a *probability density function.* It is also called a *density.* Note that if the probability density function is denoted as $p(x)$, the value at $p(x)$ (for instance, $p(x = 0)$) is not a probability. Rather, the integral of $p(x)$ is a probability. For instance, the probability that $0 \leq x \leq 0.2$ can be expressed as follows:

$$\int_{0}^{0.2} p(x)dx$$

Joint Distribution

When there are multiple random variables, a *joint distribution* expresses the probability that how likely each of these random variables takes the possible values for them. It is a type of probability distribution. When there are two random variables a and b, we write their joint distribution as $p(a, b)$, and when we have total of K random variables $\theta_1, \theta_2, \ldots, \theta_K$, we write their joint distribution as $p(\theta_1, \theta_2, \ldots, \theta_K)$.

Marginalization and Marginal Distribution

Marginalization is the elimination of variables by summing up or integrating a joint distribution with respect to random variables in the joint distribution over their possible values. The probability distribution obtained by marginalization is called a *marginal distribution*. For instance, by summing up the joint probability with respect to a discrete random variable a, we can obtain the marginal distribution $p(b)$ from $p(a, b)$:

$$p(b) = \sum_a p(a, b)$$

Similarly, when a is a continuous random variable, by integrating the joint distribution with respect to a, we can obtain the marginal distribution $p(b)$ from $p(a, b)$:

$$p(b) = \int p(a, b)da$$

Figure 2 shows the relationship between joint distribution and marginal distribution. The example where both a and b are discrete variables is shown in the left table of Fig. 2. The table shows the distribution of the species and sexes of animals in a pet store. The region surrounded by the solid gray line is the joint distribution $p(Animal, Sex)$. The region surrounded by the dotted gray line on the right side is the marginal distribution $p(Sex)$, which is obtained by summing up the probability with respect to *Animal*. Similarly, the region surrounded by the dashed gray line on the bottom side is the marginal probability $p(Animal)$, which is obtained by summing up the probability with respect to Sex. It is called a marginal distribution because we commonly show the summed probability on the margins of a table as shown here. The right plot on Fig. 2 shows when both a and b are continuous variables. The contour lines represent the joint distribution $p(a, b)$. The density plot on the right side of the plot is $p(b)$ which is obtained by integrating $p(a, b)$ with respect to a. The density distribution on the top is $p(a)$, which is obtained by integrating $p(a, b)$ with respect to b.

Conditional Probability Distribution

We consider a joint distribution $p(a, b)$. The distribution of the random variable a, when a value b_0 is given to the random variable b, is called the *conditional probability distribution*, and is written as $p(a|b_0)$. The following equation holds true:

$$p(a|b_0) = \frac{p(a, b_0)}{p(b_0)}$$

Figure 3 shows the conditional probability distribution using the data from Fig. 2. The region surrounded by the solid gray line on the left table on Fig. 3 is the conditional probability distribution $p(a|b_0)$ when the sex $b_0 =$ Female is given. The density

b \ a	Cats	Dogs	Birds	Total
Male	0.19	0.23	0.12	0.54
Female	0.22	0.16	0.08	0.46
Total	0.41	0.39	0.20	1.00

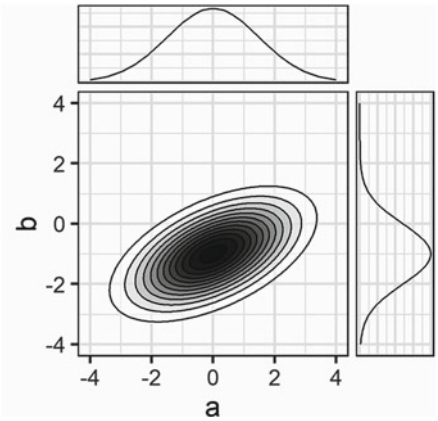


Fig. 2 Examples of joint distribution and marginal distribution

b \ a	Cats	Dogs	Birds	Total
Male				
Female	0.22/0.46	0.16/0.46	0.08/0.46	0.46/0.46

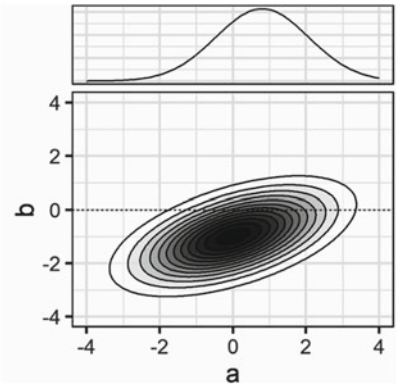


Fig. 3 Examples of conditional probability distribution

function on the top right of Fig. 3 is the conditional probability distribution $p(a|b_0)$ when $b_0 = 0$ is given.

In addition, when the location and the shape of the probability distribution of a random variable y is determined by parameter θ, we write it as $p(y|\theta)$.

Notation of $y \sim p(y)$

The expression of $y \sim p(y)$ represents that the values of a random variable y is generated probabilistically from a probability distribution $p(y)$. In other words, a random variable y follows a probability distribution $p(y)$. For instance, if we write a Poisson distribution with parameter λ as $\text{Poisson}(y|\lambda)$, then $y \sim \text{Poisson}(y|\lambda)$ represents that the values of y is probabilistically generated from $\text{Poisson}(y|\lambda)$, or y follows $\text{Poisson}(y|\lambda)$. We can also write it by omitting y, like $y \sim \text{Poisson}(\lambda)$.

Independence
When we say that two random variables a and b are mutually *independent*, we mean that this equation holds true: $p(a, b) = p(a)p(b)$. Further, from the definition of conditional probability distribution, this is equivalent to $p(a|b) = p(a)$.

Normalization
In this book, we use the term *normalization* to refer multiplying a constant to a function (or dividing by a constant), for the function to satisfy the condition of the probability density function. The constant that is used for normalization is called the normalization constant.

Partial Derivative
Consider a function of two variables $f(\theta_1, \theta_2)$. We consider the derivative by fixing θ_2 as a constant and changing θ_1

$$\lim_{\Delta\theta_1 \to 0} \frac{f(\theta_1 + \Delta\theta_1, \theta_2) - f(\theta_1, \theta_2)}{\Delta\theta_1}$$

and this is called *partial derivative* of $f(\theta_1, \theta_2)$ with respect to θ_1. The same would apply for multivariate function $f(\theta_1, \theta_2, \ldots, \theta_K)$, and when taking its derivative, we only need to consider all variables as constants, except for θ_1.

Uppercase Letters
In this book, all the uppercase letters such as Y and Age refer to the given data, unless otherwise specified.

Vectors and Matrices
In this book, vectors are denoted with arrows on top of each letter ($\overrightarrow{x}$), whereas matrices are denoted with uppercase letters with bold font (**X**). When it is only said to be a vector, its default is a column vector. Sometimes we use transpose sign T with a row vector to express a column vector. The main reason of doing this is simply to save the paper space.

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = (x_1\ x_2\ x_3)^T$$

Subscripts and "[]"
There are two types of subscripts. One is to represent the contents of the variable like σ_y (which represents σ, the standard deviation, of y). The other type is to represent the elements in a vector or a matrix. For instance, when we want to specify the k-th element in a vector $\overrightarrow{\theta}$, we write it as θ_k. In the latter case, sometimes we use "[]" to represent θ_k as of $\theta[k]$. Similarly, we write $\theta[n, k]$ to represent $\theta_{n,k}$ as well. Additinoally, $\overrightarrow{Y[3]}$ represents that the third element of Y is a vector.

Others
In the field of statistical modeling, terms such as mixed models, mixed effect models, fixed effects, and random effects are frequently used. This book also deals with the models that are related to these terms but does not use these. This is because we consider it important to understand model formulas rather than the terms themselves. In addition, since multilevel models and hierarchical models have the same mathematical formula, they are both called hierarchical models in this book.

The Source Code Used in the Book

All the source code and data in each chapter, including the visualization part, are freely available on this book's GitHub repository: https://github.com/MatsuuraKentaro/Bayesian_Statistical_Modeling_with_Stan_R_and_Python.

Also, in some of the chapters, we added some practice examples where the readers can code by themselves and deepen the understanding. The answers to those practical examples are also released on the same GitHub repository. Please use it as a reference. Lastly, in this book, we kept the following formats consistent:

- The set of equations to represent a model (Model Formula X.Y)
- Stan code (`modelX-Y.stan`)
- R and Python code (`run-modelX-Y.R, run-modelX-Y.py`)

The computing environment under which this book was written was Windows 11 (64bit), R 4.1.3, Stan 2.29.2, CmdStanR 0.5.0, CmdStanPy 1.0.1.

Tokyo, Japan Kentaro Matsuura

Acknowledgements First of all, I am particularly grateful to Shuonan Chen for her great contribution on writing this book in English. I would also like to thank Masaki Tsuda for offering to review this book. Next, I wish to express my gratitude to Prof. Motohiro Ishida and Yutaka Hirachi in Springer for giving me the opportunity to write this book. Finally, I would like to thank my family, including my cats.

Contents

Part I Background of Modeling and Bayesian Inference

1 Overview of Statistical Modeling 3
1.1 What is Statistical Modeling? 3
1.2 Purposes of Statistical Modeling 5
1.3 Preparation for Data Analysis 6
1.3.1 Before Data Collection 6
1.3.2 After Collecting Data 6
1.4 Recommended Statistical Modeling Workflow 7
1.5 Role of Domain Knowledge 9
1.6 How to Represent Model 11
1.7 Model Selection Using Information Criteria 12
Reference 12

2 Overview of Bayesian Inference 13
2.1 Problems of Traditional Statistics 13
2.2 Likelihood and Maximum Likelihood Estimation (MLE) 14
2.3 Bayesian Inference and MCMC 17
2.4 Bayesian Confidence Interval, Bayesian Predictive Distribution, and Bayesian Prediction Interval 20
2.5 Relationship Between MLE and Bayesian Inference 26
2.6 Selection of Prior Distributions in This Book 27
References 28

Part II Introduction to Stan

3 Overview of Stan 31
3.1 Probabilistic Programming Language 31
3.2 Why Stan? 32
3.3 Why R and Python? 34

3.4 Preparation of Stan, CmdStanR, and CmdStanPy 35
3.5 Basic Grammar and Syntax of Stan 36
3.5.1 Block Structure 36
3.5.2 Basic Grammar and Syntax 36
3.5.3 Coding Style Guide 38
3.6 lp__ and target in Stan 40
References 42

4 Simple Linear Regression 43
4.1 Statistical Modeling Workflow Before Parameter Inference 43
4.1.1 Set Up Purposes 44
4.1.2 Check Data Distribution 44
4.1.3 Describe Model Formula 45
4.1.4 Maximum Likelihood Estimation Using R 46
4.1.5 Implement the Model with Stan 47
4.2 Bayesian Inference Using NUTS (MCMC) 49
4.2.1 Estimate Parameters from R or Python 49
4.2.2 Summarize the Estimation Result 51
4.2.3 Save the Estimation Result 52
4.2.4 Adjust the Settings of MCMC 54
4.2.5 Draw the MCMC Sample 57
4.2.6 Joint Posterior Distributions and Marginalized Posterior Distributions 59
4.2.7 Bayesian Confidence Intervals and Bayesian Prediction Intervals 60
4.3 `transformed parameters` Block and `generated quantities` Block 61
4.4 Other Inference Methods Besides NUTS 64
4.4.1 Bayesian Inference with ADVI 64
4.4.2 MAP Estimation with L-BFGS 64
4.5 Supplementary Information and Exercises 65
4.5.1 Exercises 66
Reference 66

5 Basic Regressions and Model Checking 67
5.1 Multiple Linear Regression 67
5.1.1 Set Up Purposes 68
5.1.2 Check Data Distribution 68
5.1.3 Imagine Data Generating Mechanisms 69
5.1.4 Describe Model Formula 70
5.1.5 Implement the Model 71
5.1.6 Estimate Parameters 72
5.1.7 Interpret Results 74

5.2 Check Models 75
5.2.1 Posterior Predictive Check (PPC) 76
5.2.2 Posterior Residual Check (PRC) 77
5.2.3 Scatterplot Matrix of MCMC Sample 78
5.3 Binomial Logistic Regression 79
5.3.1 Set Up Purposes 80
5.3.2 Check Data Distribution 80
5.3.3 Imagine Data Generating Mechanisms 80
5.3.4 Describe Model Formula 82
5.3.5 Implement the Model 83
5.3.6 Interpret Results 84
5.4 Logistic Regression 85
5.4.1 Set Up Purposes 86
5.4.2 Check Data Distribution 86
5.4.3 Imagine Data Generating Mechanisms 86
5.4.4 Describe Model Formula 87
5.4.5 Implement Models 87
5.4.6 PPC 88
5.5 Poisson Regression 89
5.5.1 Imagine Data Generating Mechanisms 90
5.5.2 Describe Model Formula 90
5.5.3 Implement the Model 91
5.5.4 Interpret Results 92
5.6 Expression Using Matrix Operation 93
5.7 Supplemental Information and Exercises 94
5.7.1 Exercises 95

Part III Essential Technics for Mastering Statistical Modeling

6 Introduction of Probability Distributions 99
6.1 Notations 100
6.2 Uniform Distribution 101
6.3 Bernoulli Distribution 102
6.4 Binomial Distribution 103
6.5 Beta Distribution 104
6.6 Categorical Distribution 106
6.7 Multinomial Distribution 108
6.8 Dirichlet Distribution 109
6.9 Exponential Distribution 111
6.10 Poisson Distribution 112
6.11 Gamma Distribution 114
6.12 Normal Distribution 116
6.13 Lognormal Distribution 118
6.14 Multivariate Normal Distribution 119
6.15 Cauchy Distribution 123

6.16 Student-t Distribution 125
6.17 Double Exponential Distribution (Laplace Distribution) 126
6.18 Exercise 128
References 128

7 Issues of Regression 129
7.1 Log Transformation 129
7.2 Nonlinear Model 133
7.2.1 Exponential Function 134
7.2.2 Emax Function 137
7.2.3 Sigmoid Emax Function 137
7.2.4 Other Functions 139
7.3 Interaction 141
7.4 Multicollinearity 142
7.5 Model Misspecification 143
7.6 Variable Selection 145
7.7 Censoring 146
7.8 Outlier 148
References 150

8 Hierarchical Model 151
8.1 Introduction of Hierarchical Models 152
8.1.1 Set Up Purposes and Check Data Distribution 152
8.1.2 Without Considering Group Difference 153
8.1.3 Groups Have Varying Intercepts and Slopes 154
8.1.4 Hierarchical Model 156
8.1.5 Model Comparison 161
8.1.6 Equivalent Representation of Hierarchical Models 163
8.2 Hierarchical Model with Multiple Layers 164
8.2.1 Set Up Purposes and Check Data Distribution 165
8.2.2 Imagine Data Generating Mechanisms and Describe Model Formula 167
8.2.3 Implement the Model 168
8.3 Hierarchical Model for Nonlinear Model 169
8.3.1 Set Up Purposes and Check Data Distribution 169
8.3.2 Imagine Data Generating Mechanisms and Describe Model Formula 170
8.3.3 Implement Models 171
8.3.4 Interpret Results 172
8.4 Missing Data 173
8.5 Hierarchical Model for Logistic Regression Model 177
8.5.1 Set Up Purposes 177
8.5.2 Imagine Data Generating Mechanisms 178
8.5.3 Describe Model Formula 178

8.5.4 Implement Models 179
8.5.5 Interpreting Results 180
8.6 Exercises 180
References 181

9 How to Improve MCMC Convergence 183
9.1 Removing Nonidentifiable Parameters 183
9.1.1 Parameter Identifiability 184
9.1.2 Individual Difference 185
9.1.3 Label Switching 185
9.1.4 Multinomial Logistic Regression 187
9.1.5 The Tortoise and the Hare 189
9.2 Use Weakly-Informative Priors to Restrict the Posterior Distributions 192
9.2.1 Weakly Informative Prior for Parameters in $(-\infty, \infty)$ 193
9.2.2 Weakly Informative Prior for Parameters with Positive Values 194
9.2.3 Weakly Informative Prior for Parameters in Range [0, 1] 200
9.2.4 Weakly Informative Prior for Covariance Matrix 200
9.3 Loosen Posterior Distribution by Reparameterization 206
9.3.1 Neal's Funnel 207
9.3.2 Reparameterization of Hierarchical Models 209
9.3.3 Reparameterization of Multivariate Normal Distribution 210
9.4 Other Cases 211
9.5 Supplementary Information 211
References 212

10 Discrete Parameters 213
10.1 Techniques to Handle Discrete Parameters 213
10.1.1 log_sum_exp Function 214
10.1.2 Marginalizing Out Discrete Parameters 214
10.1.3 Using Mathematical Relationships 220
10.2 Mixture of Normal Distributions 221
10.3 Zero-Inflated Distribution 227
10.3.1 Set Up Purposes and Check Data Distribution 228
10.3.2 Imagine Data Generating Mechanisms 228
10.3.3 Describe Model Formula 230
10.3.4 Implement Models 230
10.3.5 Interpret Results 232
10.4 Supplementary Information and Exercises 232
10.4.1 Exercises 233
Reference 233

Part IV Advanced Topics for Real-World Data Analysis

11 Time Series Data Analysis with State Space Model 237
11.1 Introduction to Space State Models 237
11.1.1 Set Up Purposes 239
11.1.2 Check Data Distribution 239
11.1.3 Imagine the Mechanisms of Data Generation Process 240
11.1.4 Describe Model Formula 241
11.1.5 Implement the Model 242
11.1.6 Interpret the Results 243
11.2 Extending System Model 244
11.2.1 Trend Component 244
11.2.2 Regression Component 246
11.2.3 Seasonal Component 247
11.2.4 Switch Component 252
11.2.5 Pulse Component 255
11.2.6 Stationary AR Component 258
11.2.7 Reparameterization of Component 259
11.3 Extending the Observation Model 261
11.3.1 Outliers 261
11.3.2 Binary Values 262
11.3.3 Count Data 262
11.3.4 Vector 263
11.4 State Space Model with Missing Data 263
11.4.1 Observations at Certain Time Points are Missing 263
11.4.2 Time Intervals are not the Same (Unequal Intervals) 263
11.4.3 Vector 264
11.5 (Example 1) Difference Between Two Time Series 264
11.6 (Example 2) Changes in Body Weight and Body Fat 267
11.7 (Example 3) The Transition of Tennis Players' Capabilities 269
11.8 (Example 4) Decomposition of Sales Data 274
11.9 Supplementary Materials and Exercises 283
11.9.1 Exercises 284
References 284

12 Spatial Data Analysis Using Gaussian Markov Random Fields and Gaussian Processes 285
12.1 Equivalence Between State Space Model and One-Dimensional GMRF 286
12.1.1 Posterior Probability of the State Space Model 286
12.1.2 The Equivalence Between the Temporal and Spatial Structures 287

12.2 (Example 1) Data on One-Dimensional Location 289
12.3 (Example 2) Fix the "Age Heaping" 291
12.4 Two-Dimensional GMRF 296
12.5 (Example 3) Geospatial Data on the Map 300
12.6 (Example 4) Data on Two-Dimensional Grid 302
12.7 Introduction to GP 307
12.7.1 Implementation of GP (1) 310
12.7.2 Implementation of GP (2) 313
12.7.3 Prediction with GP 314
12.7.4 Other Kernel Functions 319
12.8 (Example 5) Data on One-Dimensional Location 320
12.9 (Example 6) Data on Two-Dimensional Grid 321
12.10 Inducing Variable Method 324
12.11 Supplementary Information and Exercises 327
12.11.1 Exercises 328
References 328

13 Usages of MCMC Samples from Posterior and Predictive Distributions 331
13.1 Simulation Based Sample Size Calculation 331
13.2 Bayesian Decision Theory 335
13.3 Thompson Sampling and Bayesian Optimization 338
13.3.1 Thompson Sampling 338
13.3.2 Bayesian Optimization 341
References 344

14 Other Advanced Topics 345
14.1 Survival Analysis 345
14.2 Matrix Decomposition and Dimensionality Reduction 352
14.2.1 Matrix Decomposition 352
14.2.2 Dimensionality Reduction 359
14.3 Model Selection Based on Information Criteria 367
14.3.1 Introduction of Generalization Error and WAIC 367
14.3.2 Simulation Study to Evaluate Information Criteria 370
14.3.3 WAIC in a Hierarchical Model 375
14.4 Supplementary Information and Exercises 382
14.4.1 Exercises 383
References 383

Appendix: Differences from BUGS Language 385

Part I
Background of Modeling and Bayesian Inference

Chapter 1
Overview of Statistical Modeling

In this chapter, we will give a general overview on what a model is and how to conduct statistical modeling. Understanding these general concepts is necessary to be able to build your own model. Each section has the following contents.

- Section 1.1 introduces the general concepts of models and what statistical models are.
- Section 1.2 is the discussion on the purpose of statistical modeling. There will be explanation on interpretation, prediction performance, and robustness.
- Section 1.3 introduces the processes to get ready for data analysis. Some of these processes will be the parts of a workflow in the following sections in this book.
- Section 1.4 introduces the recommended statistical modeling workflow adopted in this book. From this section, the readers might find that the statistical modeling is more similar to engineering than arts.
- Section 1.5 is devoted to the discussion about the importance of incorporating domain knowledge into models when working with real problems. We will also discuss about a common misunderstanding about models, such as "correctness" of models.
- Section 1.6 describes how to mathematically represent probabilistic models (statistical models).
- Section 1.7 explains why we do not use information criteria except in the last chapter of this book.

1.1 What is Statistical Modeling?

A *model* is a simplified copy of an original. A model removes the irrelevant elements from the original object and extracts its essential part, in order to reproduce certain properties in the original. For example, a plastic car/airplane model ignores the size, weight, material and function of the original car/airplane, but it keeps the original shape and colors. Similarly, a model that describes the essential parts of phenomenon

© Springer Nature Singapore Pte Ltd. 2022

K. Matsuura, *Bayesian Statistical Modeling with Stan, R, and Python*,
https://doi.org/10.1007/978-981-19-4755-1_1

Table 1.1 Comparison between problem solved by linear equations and statistical modeling. We will explain Maximum Likelihood Estimation and MCMC in Chap. 2

	Simultaneous equations	Statistical modeling
Problem	Speed of a train Concentration of saline solution	Interpretation and prediction of phenomena
Model	Linear equations	Probabilistic model with parameters
Solution	Elimination method Substitution method Cramer's law	Maximum Likelihood Estimation MCMC (Bayesian) Variational Bayes method (Bayesian)
Form of answer	$x = \ldots, y = \ldots$	Parameter values Joint distribution of parameters (Bayesian)

using a mathematical expression is called a *mathematical model*. A mathematical model that uses probability distributions is called a *probabilistic model*, or a *statistical model*. Throughout this book, the term "model" is mostly used to represent a probabilistic model.

Then, *statistical modeling* is an analysis method that promotes understanding and prediction of phenomena by fitting probabilistic models to the data of interest. A probabilistic model consists of probability distributions, parameters, and mathematical expressions representing relations between the parameters. A *parameter* is a variable whose value is unknown before conducting analysis.[1] The values of parameters in a probabilistic model are estimated from data, and these values are further used to interpret and predict the phenomenon of interest.

The advantage of using probabilistic models is that we can incorporate the domain knowledge into the models. However, because all probabilistic models are merely hypotheses that is aimed to obtain a meaningful interpretation, we need to pay attention whether the assumptions of the model are persuasive and whether the ignored elements are indeed negligible for understanding the phenomena of interest.

Let us look at Table 1.1 to familiarize ourselves with these terminologies introduced above. Here, we illustrate the similarities between the mathematics word problems that we solved in junior high school, and the problems that we face in statistical modeling. When you were a junior high school student, you would have mastered simultaneous equations by solving many word problems. Similarly, to master statistical modeling, you need to practice by creating many probabilistic models for various problems. The main difference from what we did in junior high school is that Stan is able to help give the answer almost automatically from the probabilistic model, as we will demonstrate in Chap. 4.

[1] As explained in Sect. 2.3, in the framework of Bayesian statistics, it is assumed that all parameters follow certain probability distributions.

1.2 Purposes of Statistical Modeling

As briefly summarized in the last section, the first purpose of statistical modeling is *interpretation*. It is one of the fundamental human nature that we want to know the reasonings and mechanisms of phenomena, and to acquire new knowledge. Modeling plays a role in facilitating the understanding of how the phenomenon occurred and how the data was generated. From this point of view, models that are easy to interpret are considered as good models. Such models can also be said to be convincing, or easy-to-explain models. Ease of interpretation makes the decision making easier.

The second purpose of statistical modeling is *prediction*. That is, to predict the future output of the phenomenon using the obtained data, or to predict how the system responds to a new input. From this point of view, models with high accuracy of prediction are regarded as good models.

Note that interpretation and prediction are not independent of each other. A convincing model that is consistent with domain knowledge is generally robust. The term *robustness* is used to represent a model property that interpretation and prediction will not change largely when there are slight changes to the model or to the input data. Thus, a model with low robustness often fails to predict against new data sets. This leads to a substantial failure in practice.

Notably, all results, including posterior probabilities, are the results of seeing the worlds through the lenses, which depends on the assumption of our model. We would like to emphasize that, in reality, we never know what the "true" model (or "correct" answer) is. As a result, we cannot argue if the assumptions, estimation methods, or interpretations are "correct" or not. Because we do not know the true model, we cannot conclude that, for instance, "the probability that this hypothesis is true is 0.78" or "our estimation method is able to detect the correlation which is not detected by other methods". Rather, when we are talking about models, it is more important to focus on what the assumptions are, what estimation method is used, and how the result is interpreted. We can also discuss which model is easier to interpret, which is more convincing, and which has the better prediction performance. We do not consider that the statistical modeling as a tool to identify the true mechanism. It is important to realize that we can gradually approach the true mechanism (which we can never attain) and make better predictions by improving models with increasing amount of data. Therefore, we must consider statistical modeling as an improvement process by iteratively modeling rather than just a one-time modeling.

If the high prediction performance is the only purpose of modeling, you may consider adopting various machine learning algorithms, such as support vector machines, gradient boosting decision tree, and deep learning. In these machine learning methods, it is not necessary to convert domain knowledge into mathematical expressions, therefore high prediction performance can be obtained even when the background knowledge of the phenomenon is not given. However, interpreting the result from these methods is generally difficult, and extending these models on our own is usually not straightforward. On the other hand, the results of classical statistical analysis methods, such as ANOVA and simple cross tabulation, are relatively easy to interpret. However, they often do not help on prediction. The problem

with these methods is that they cannot fully take advantage of domain knowledge or past experiences about a phenomenon. Compared to machine learning methods and classical statistical methods, statistical modeling has the advantage of being easy to interpret and be being customized, as well as good predictive performance even for the complex real-world problems. With the increasing amount of data that we can collect nowadays, along with the advancing knowledge and experiences of statistical modeling, we have witnessed useful models having these advantages in many academic papers. We hope this book will help you build such models and apply to in your own field.

1.3 Preparation for Data Analysis

Before we jump into the statistical modeling workflow, let us get ready for practical data analysis by explaining the critical concepts. These concepts are not only necessary for statistical modeling, but also are helpful for the data analysis in general. In particular, "set up purposes" and "check data distribution" are the important processes in statistical modeling and are also used in the examples of this book.

1.3.1 Before Data Collection

- **Examine domain knowledge**: Examine what assumptions, analysis methods, and visualization methods are usually used in your field.
- **Set up purposes**: Summarize what you want to know and conclude from the data. It is a good idea to itemize and prioritize them. Also, imagine that you are to present the analysis result or to submit a report. Consider what stories are easy for the audience and readers to understand, and what graphs would support your argument.
- **Plan analysis**: Describe which methods you plan to use. Consider the best scenario and milestones in the analysis, just as we plan a hiking trip.

1.3.2 After Collecting Data

- **Check data distribution**: This is a visualization process. First visualize the distribution of values of each data attribute (e.g., each column of a data frame if you use R or Python). Draw one-dimensional graphs (histogram) for each attribute and draw two-dimensional graphs (scatter plot or boxplot) for a pair of attributes. For time series data, it is also helpful to draw a line graph with time on the x-axis and values on the y-axis. For a combination of two attributes with categorical values, cross tables might be a good way to visualize their relationships. Next, calculate

summary statistics for each group and individual, and visualize their distributions. It may require some computation by aggregating or by preprocessing the data. These visualizations and aggregated values are very useful in considering the mechanisms of data generation process.

If "check data distribution" is done properly, data cleaning[2] can be done efficiently. We recommend Spotfire[3] and R as the tools for drawing graphs, and R, Python, SQL[4] for data aggregation and preprocessing. OpenRefine[5] is also an option for data cleaning.

1.4 Recommended Statistical Modeling Workflow

Let us return to the statistical modeling workflow. Statistical modeling has a general workflow, which is highly helpful especially for beginners who are about to start the statistical modeling. However, there are a few documents (Gelman et al., 2020) that introduce workflows. In reality, statistical modeling workflows are similar to art works: it is highly changeable, depending on the problems and the data, and there is no established global standard. Even though, we will give a general standard as follows, and we will also proceed along this order in the examples throughout this book (there is no particular order between "interpret results" and "check models").

- **Set up purposes**: See "set up purposes" in the previous section.
- **Check data distribution**: See "check data distribution" in the previous section.
- **Imagine data generating mechanisms**: Suppose that we know exactly what mechanism has caused our data. It is a good idea to draw an illustration that represents the relationships between variables in the model. The illustration typically contains sketches of probability distributions and formulas that generate the data. It is also helpful to confirm which part of the illustration corresponds to the set problem.
- **Describe model formula**: Describe the data generating mechanism using a model formula,[6] i.e., a set of mathematical formulas in this book. The mathematical formulas could help us realize an equivalent representation of the model, which could be a hint for efficient implementation.
- **Simulate models**: Generate data from the model formula by simulation in order to see what properties the model formula has.
- **Implement models**: Implement the model formula in a programming language for estimation.

[2] Data cleaning means to arrange the data so that later analysis will be smooth by correcting or eliminating abnormal values mixed by human error and by unifying notation fluctuations.

[3] http://spotfire.tibco.jp/.

[4] https://en.wikipedia.org/wiki/SQL.

[5] http://openrefine.org/.

[6] The explanation of the model formula will be in Sect. 1.6.

- **Estimate parameters**: Estimate parameters with both simulated and real data.
- **Check models**: Check whether the true values are recovered from simulated data. This is called *parameter recovery*. If the true values cannot be recovered, fitting the model to real data will not provide useful information. Visualization is also very helpful to check the goodness of fit.
- **Interpret results**: Interpret and visualize the estimated results to solve the set problem.

The key to follow this workflow is to start from a simple model. When we try to handle real-world data, we tend to assume a complex mechanism from the very first. However, we may end up being stuck there and not able to move on to the next step. Or even if we succeed implementing the model, parameter estimation may not work. To avoid such issues, it is a common practice to try from a simple model. Here are some suggestions as examples:

- If it is possible to use a simple model that is frequently used in textbooks in the field, start from that simple model.
- If there are many explanatory variables,[7] narrow them down.
- Treat random variables as independent as possible. For instance, use univariate distributions instead of multivariate distributions.
- Ignore group differences or individual differences in the first model.

After starting from a simple model that works fine, repeat all or parts of the above workflow several times in cycles, gradually make it a more complex model. It seems to be a circuitous at first glance, but it enables us to find any improper assumptions. Also, by doing this, there will be less possibility introducing bugs for the model. These will help to take the minimum amount of time to a successful model.

If we have large data that takes a long time to calculate, it may be a good idea to use a part of it in the beginning. However, it is not always recommended to reduce data size because a structure that we want to know may diminish. Examples of reducing data are as follows:

- Randomly draw 1/10 of the original data points and use this small subset of the whole data to build the first model.
- Limit the levels of categories. For example, data that has multiple factors, such as store $\times$ year $\times$ item, can be handled by fixing a store.

Suppose that you have created the best model by practicing above. However, data analysis has not been completed yet. As mentioned in Sect. 1.2, data analysis should be done in cycles as follows:

- If the prediction performance is low, consider what the reason is.
- Collect new data to resolve the cause of low performance.
- With the amount of the data increases, it becomes possible to build and estimate more complex models with more assumptions.
- Then, we can make new interpretation from the model and obtain high prediction performance as well.

[7] See Chap. 4.

Technical Point: Confirmation of Reproducibility

We did not include "confirmation of reproducibility" into the above workflow, but it is also an important task. *Reproducibility* indicates that the same results can be obtained regardless of who and when the analysis is conducted, as long as it follows the same procedure. This concept is particularly emphasized recently in data analysis. Version control of programming code is also mentioned when discussing about reproducibility in data analysis. Here we would like to describe how to check reproducibility of estimation results. Reproducibility is closely related to robustness mentioned in Sect. 1.2. For example, check the following points:

- Whether the estimation result changes largely if a few data points are removed or the model (including prior distributions) is slightly modified.
- Whether the estimation result remains similar when using other data sets.
- Whether the estimation result remains similar even if software and algorithms are changed (for example, whether the estimation result of Stan is similar to that of WinBUGS and/or JAGS).
- If we are using an algorithm that depends on the initial values or random numbers, whether the estimation result remains similar if these are changed.

1.5 Role of Domain Knowledge

For the step "imagine data generating mechanisms" in the workflow mentioned in the previous section, domain knowledge is used. Let us use a simple example to explain why domain knowledge is necessary.

In Fig. 1.1, suppose that rotating the handle (○) on the left side of this figure would cause would cause the horizontal movement of the star (☆) on the right side. Then we ask: what is the mechanism inside of the gray box that converts the rotational motion into the horizontal motion? Here, only inputs and outputs are known, and a mechanism which links them together is not given, and such problem is called an *inverse problem*. Usually there are multiple answers to an inverse problem. A problem like this one, where the answer is not uniquely determined, is called an *ill-posed problem* and cannot be solved uniquely unless other information such as domain knowledge is included to the analysis. Suppose that we add domain knowledge about the mechanisms, such as "the number of parts is small, and all the parts are made of iron, strong and long-lasting, and can be made cheaply". Then, for example, arranging the mechanical parts as shown in Fig. 1.2 can be one answer of this mechanism.

This example resembles how we do statistical modeling. Most of the time, we are given only a piece of information about the inputs and outputs, namely data, to explain phenomena in the real world, but the mechanism that how they are linked together is unknown. Therefore, it cannot be solved without some assumptions about the model. In order to uniquely determine the answer, we often make various assumptions about

Fig. 1.1 An example of inverse problems

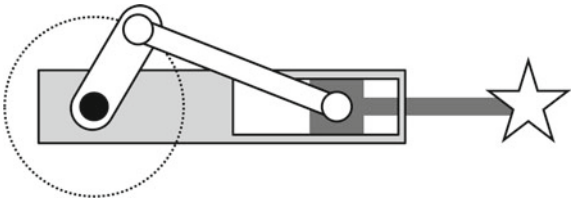


Fig. 1.2 The mechanism underlying the slider crank devised from domain knowledge (the position of the fasteners of ● does not change, and the positions of the two ○ fasteners move as we rotate the handle)

the data and thereby set constraints for the model. This is equivalent to converting the problem with `P>>D` into a problem with `P<<D`, where P is the number of parameters and D is the number of data points.

However, it should also be noted that each assumption should be carefully examined to make sure that it is supported by sufficient reasons and evidences. Assumptions that are merely made to fit the model perfectly to the data are not necessarily the proper assumptions, and they may lead to overfitting,[8] and may yield inaccurate conclusions and predictions subsequently. Rather, assumptions should be made to build a robust model and facilitate the interpretation of the analysis.

We would like to address another topic that is often misunderstood about modeling. That is, a model that incorporates all details in a phenomenon is not necessarily a good model. For example, assume that we want to know how many tomatoes a farmer would harvest when several new fertilizers are used. To answer this question, is it necessary to think about how each fertilizer ingredient is absorbed from the roots, how the ingredients are used for proteins and minerals to synthesize stems and tomato fruits, and include all these processes in our models? Of course, there are some cases where we cannot understand a phenomenon without considering every such process in detail. In most of the practical situations, however, simpler models are preferable, and unnecessarily complex models would lead to difficulty in interpretation. On the other hand, in the tomato farm example, if we want to know how much increase in phosphate concentration will be caused by using each fertilizer, it will be important to consider the process how it is absorbed from the roots. Therefore, even if the data is the same, the model may change depending on the problem set up.

[8] Overfitting is explained in Sect. 2.2.

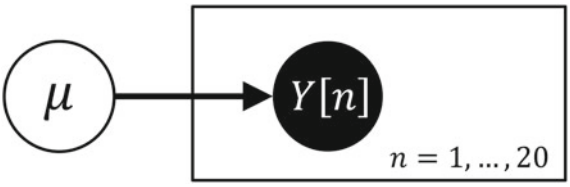


Fig. 1.3 An example of graphical models

1.6 How to Represent Model

There are several ways to represent models. In this book, we use model formulas. A *model formula* is a set of mathematical formulas describing the relationship between random variables in a model. In the model formula, variables generated deterministically are denoted with "=", and variables generated probabilistically are denoted with "~".

As an example, suppose there are 20 data points $Y[1], Y[2], \ldots, Y[20]$ independently generated from a normal distribution with an unknown mean with standard deviation (SD) of 1. The model formula in this case is expressed as follows.

$$Y[n] \sim \mathcal{N}(\mu, 1) \qquad n = 1, 2, \ldots, 20$$

where $\mathcal{N}(\mu, \sigma)$ denotes the normal distribution with parameters μ (mean) and σ (SD). This is a short way of saying

$$Y[1] \sim \mathcal{N}(\mu, 1),\ Y[2] \sim \mathcal{N}(\mu, 1),\ \ldots,\ \text{and}\ Y[20] \sim \mathcal{N}(\mu, 1).$$

and can be phrased as:

"Each of $Y[1], Y[2], \ldots, Y[20]$ is probabilistically generated from $\mathcal{N}(\mu, 1)$"

or

"Each of $Y[1], Y[2], \ldots, Y[20]$ follows $\mathcal{N}(\mu, 1)$".

Model formulas can express models thoroughly and are easy to transition to implementation.

The disadvantage of model formula is that when the number of variables increases, relationships between them can be difficult to understand intuitively. Therefore, in addition to the model formulas, there are several ways to use graphs to express models. One of them is a *graphical model*, which is often included in academic papers dealing with complex models. For instance, the above example can be expressed as a graphical model in Fig. 1.3.

Most of graphical models follow certain rules, such as:

- A white circle (node) with black text represents a random variable to be estimated.[9]
- A black circle (node) with white text represents a random variable whose value is observed.

[9] There is also a way to color random variables of interest.

- An arrow between nodes (directed edge) indicates that we assume the conditional probability distribution of the variable on the tip of the arrow.
- A line without arrow between nodes (undirected edge) indicates that we assume a joint distribution of these nodes.
- A rectangular plate represents a repeating structure.

Although these are some common rules, some graphical models might adopt other versions of rules. As an alternative way to express models using graphs, path diagrams in Structural Equation Modeling (SEM) are also often used. One of the disadvantages of using graphs is that they do not include mathematical formulas, therefore it is hard to directly convert these representations to the implementation. In most of the models in this book, we do not use expressions by graphs because the number of variables in the models is relatively small.

1.7 Model Selection Using Information Criteria

For model selection, various information criteria are often are used in the statistics field. Despite the fact that they are commonly used in model selection, they should be used with extra caution.

First of all, model selection using information criteria could lead to the problem of overfitting because information criteria are calculated from given data. For example, model selection with WAIC is theoretically less likely to overfit the data, because it is asymptotically equal to leave-one-out cross-validation when the number of data points increases. In practice, however, it is not easy to avoid overfitting even if we use cross-validation. Therefore, selecting a model using information criteria does not always give a good model.

Also, if we overly rely on the information criteria, we may fail to fully explore models. This is not a problem when the models were built with all the consideration on the background knowledge about the collected data. If enough models have not been tried yet, imagine more other models before using the information criteria in model selection.

Therefore, we will not expand the discussion about using information criteria other than Chap. 14, which is our own warning against model selection that relies solely on the information criteria. The information criteria are the last "dessert."

Reference

Gelman, A., Vehtari, A., Simpson, D., Margossian, C. C., Carpenter, B., Yao, Y. et al. (2020). Bayesian workflow. arXiv preprint arXiv:2011.01808.

Chapter 2
Overview of Bayesian Inference

In this chapter, we overview the basis of Bayesian inference and MCMC, which are the fundamental steps before appropriately using Stan. The mathematical implementation of MCMC will not be mentioned in this book. We refer those who are interested to (Casella & Robert, 2005) for further details. This chapter is organized as follows:

- Section 2.1 illustrates three problems that exist in the traditional statistics.
- Section 2.2 introduces likelihood functions and maximum likelihood estimation (MLE), which are key concepts to fit a model to given data. Overfitting is mentioned in this section too.
- Section 2.3 introduces the terminology used in Bayesian inference and MCMC, and how to assess convergence of MCMC based on a trace plot.
- Section 2.4 explains the Bayesian confidence interval, which can be derived from the posterior distribution of a parameter. Then we explain a predictive distribution of a new data point and a Bayesian prediction interval.
- Section 2.5 provides the relationship between MLE and Bayesian inference. Also in this section, we will see the consistency of results between the one obtained using Bayesian inference with non-informative prior distributions and using traditional statistics approach with MLE.
- Section 2.6 discusses how we choose prior distributions in this book, following the results we obtain from Sect. 2.5. Usage of weakly informative prior distributions is discussed in Sect. 9.2.

2.1 Problems of Traditional Statistics

In the traditional statistics, we assume that the true value of parameter is an unknown constant. This causes several practical problems, such as:

- **Interpretation of statistical test becomes non-intuitive**: In the traditional statistics, under the hypothesis H and model M, probability of getting data Y or more

© Springer Nature Singapore Pte Ltd. 2022

K. Matsuura, *Bayesian Statistical Modeling with Stan, R, and Python*,
https://doi.org/10.1007/978-981-19-4755-1_2

extreme data (e.g., $p_M(y \geq Y|H)$) is called "p-value". Even though p-value is used widely, its statistical meaning is non-intuitive and is hard to interpret. In contrast, in Bayesian statistics, the probability of hypothesis H being true given data Y and model M (i.e., $p_M(H|Y)$) can be obtained as a posterior distribution of a parameter, and thus has much easier and more straightforward interpretation.

- **Interpretation of confidence intervals becomes non-intuitive**: In the traditional statistics, we treat parameter θ as a constant, rather than a random variable, thus we cannot say that the probability of the value θ being in the interval $[a, b]$ is 95%. Rather, we need to interpret this as follows: when "collecting data and calculating an interval $[a, b]$" is repeated 100 times, there are 95 times the value of θ being within this interval.
- **It is difficult to calculate confidence intervals and prediction intervals of complex models**: It is theoretically and mathematically challenging to derive confidence intervals of parameters and prediction intervals of data in the complex models. However, knowing the intervals (or the distributions) is critical when we want to interpret the results and make reasonable decisions. For example, when we are predicting the profit for the coming year, "10 million dollars" is not a sufficient description to obtain useful information. This is because we know that saying "10 ± 1 million dollars" and "10 ± 100 million dollars" have completely different meaning in practice, even though they have the same mean value.

Also, although main approaches in traditional statistics is using MLE, there are some limitations. We will see more detailed explanation in the next sessions.

2.2 Likelihood and Maximum Likelihood Estimation (MLE)

Let us consider an example where we have 20 data points $Y[1], Y[2], \ldots, Y[20]$ that are independently generated from a normal distribution with standard deviation (SD) of 1 (Fig. 2.1, left). Suppose that you were given a transparent board with a curve of normal distribution with SD of 1 (Fig. 2.1, middle), and you were told that the data was generated from this distribution. Now we ask you to align the bottom edge of the board with the x-axis of Fig. 1.1 (left), and move the board horizontally to the location where the curve fits best to these 20 data points. Where is the best location to stop the board?

In order to answer this question, we have to decide the numerical value that measures the "goodness of fit". Suppose that we focused on the y-axis of the normal distribution at each data point, and defined the product of all y values as L, and decided to stop moving the board when L is maximized. Here L is called a *likelihood* and is defined to be the indicator of goodness of fit.

Let us formulate the above problem mathematically. We consider the mean of the normal distribution to be an unknown parameter μ. Then the model formula can be expressed as follows (see Sect. 1.6).

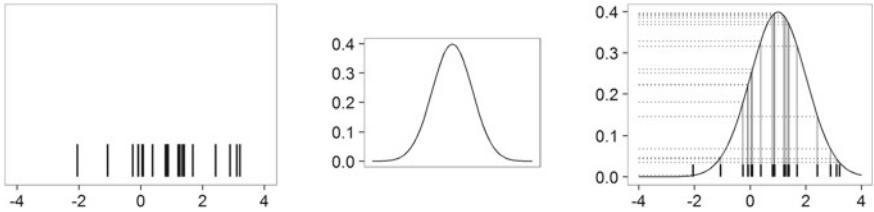


Fig. 2.1 Left: data points. The vertical bars denote the positions of data points. Middle: the transparent board to fit to the data points on the plot on the left. Right: one possible location of the board at $\mu = 1$. The product of the values on the y-axis at all data points is the likelihood $L(\mu = 1)$

$$Y[n] \sim \mathcal{N}(\mu, 1) \qquad n = 1, 2, \ldots, 20, \tag{2.1}$$

where $\mathcal{N}(\mu, \sigma)$ denotes the normal distribution with the parameters μ (mean) and σ (SD). Since the value of the y-axis of the normal distribution at point Y is $\mathcal{N}(y = Y|\mu, 1)$, the likelihood is expressed as a function of μ as follows (Fig. 2.1, right).

$$L(\mu) = \prod_{n=1}^{20} \mathcal{N}(y = Y[n]|\mu, 1) \tag{2.2}$$

This μ is the parameter to be estimated. "moving the board horizontally to the location where the curve fits best" corresponds to "moving μ to the location where the likelihood is largest". Such an estimation method for finding the parameter value with the largest likelihood is called *Maximum Likelihood Estimation* (*MLE*). The value of the determined parameter using this method is called the *maximum likelihood estimate*. Note that the estimated value is a single value, namely point estimation.

To generalize what we have so far, let $p(y|\theta)$ be a probability distribution whose shape is determined by the parameter value θ. Then the likelihood $L(\theta)$ is defined to be $p(y = Y|\theta)$, which is obtained by substituting data Y into the probability distribution. In the case of the above example, if we let y_n be the random variable that corresponds to the data point $Y[n]$, then $p(y|\theta)$ is the joint distribution $p(y_1, y_2, \ldots, y_{20}|\mu)$. Here, since $y_1, y_2, \ldots, y_{20}$ are mutually independent, the joint distribution is equivalent to:

$$\begin{aligned} p(y_1, y_2, \ldots, y_{20}|\mu) &= p(y_1|\mu) \times \cdots \times p(y_{20}|\mu) \\ &= \mathcal{N}(y_1|\mu, 1) \times \cdots \times \mathcal{N}(y_{20}|\mu, 1) \end{aligned}$$

Substituting $y_1 = Y[1], \ldots, y_{20} = Y[20]$ into this distribution yields the likelihood (2.2). Hereafter, we denote $p(y = Y|\theta)$ as $p(Y|\theta)$ for simplicity.

Note that a likelihood $p(Y|\theta) = L(\theta)$ is a function of parameter θ. A likelihood is an indicator of the goodness of fit of a model, but it is not a probability function because the integral with respect to θ is not necessarily 1. When the probability

distribution $p(y|\theta)$ is a probability density function, it is possible that the likelihood is larger than 1. In general, a likelihood has a very small value because it is the product of many values that are smaller than 1. As a result, computing such small number in a computer can be problematic. Therefore, it is common to take logarithm of a likelihood for estimation, i.e., a *log-likelihood.* For example, the log-likelihood obtained by taking the logarithm of (2.2) is as follows:

$$\log L(\mu) = \sum_{n=1}^{20} \log \mathcal{N}(y = Y[n]|\mu, 1) \tag{2.3}$$

Returning to the topic on MLE we have been discussing, if we only have a single parameter as in the previous example, all we have to do is to differentiate the log-likelihood with respect to θ and obtain the value of θ that maximizes that log-likelihood by solving the following equation:

$$\frac{\partial}{\partial \theta}\big[\log L(\theta)\big] = 0.$$

However, when the number of parameters increases, it is difficult to solve the above equation analytically. In such cases, it is necessary to use numerical calculation methods to get the maximum likelihood estimate from a computer. In a typical algorithm, the maximum likelihood estimate is calculated in two steps. First, set initial values of the parameters in a certain way. Next, move the parameters in the direction in which the log-likelihood increases most from the current position. We repeat this second step until the log-likelihood does not increase any more. As an example, `optim` function in R and an optimizer in TensorFlow does MLE by roughly adopting this method.

If the types and amount of available data increase, we can reflect our domain knowledge about data and can build more complex models for better interpretation and prediction. In these cases, however, MLE often does not work well for several reasons such as:

- **Overfit the data easily**: We can consider an extreme case that you roll a fair dice three times and the outcomes happened to be "1" in all three rolls. Based on these outcomes, using MLE would give the conclusion that the probability of getting "1" by rolling this dice is 1, and the other probabilities are 0. However, this is contrary to our experiences. In other words, prediction ability (called generalization ability) is low for new data by specifically fitting only to the observed data. This is known as *overfitting*. Overfitting can occur not only in the extreme cases as in this example, but also in cases where the amount of data is large. For instance, when data are divided based on group information, it might cause the overfitting if some groups have a limited number of data points.
- **Diverge likelihood**: In some cases, the estimation may fail when the likelihood diverges to infinity. For instance, fitting a $\mathcal{N}(\mu, \sigma)$ to data with only one data point $Y = 0$, would give the estimated parameters $\mu = 0$ and $\sigma = 0$, resulting in an

infinitely large likelihood. This could also occur when we estimate the parameters for Beta distribution or for individual groups in hierarchical models.

- **Stuck to a local optimum**: The value where the likelihood is maximized in the whole parameter space is called the *global optimum*, and the value where the likelihood is maximized within a certain limited range is called a *local optimum.* One of the limitations of the algorithms for MLE is that they tend get stuck in a local optimum close to the initial value, rather than the global optimum. Therefore, it is common to start from various initial values and adopt an optimum value from these results. However, as the number of parameters increases, the number of combinations increases as well, making it difficult to experiment with many initial values.[1]

2.3 Bayesian Inference and MCMC

One way to solve the problems listed in Sects. 2.1 and 2.2 is the combination of *Bayesian inference* using Bayes' theorem and Markov Chain Monte Carlo methods (*MCMC*).

The difference between traditional statistics and Bayesian statistics is that Bayesian statistics assume that all parameters are random variables and follow cerain probability distributions. Therefore, in the Bayesian framework, it is possible to make a clear statement such as "the probability that the value of parameter θ is in the interval $[a, b]$ is 95%". Also, it allows us to answer some questions such as "What is the probability that more than ten people will visit the shop tomorrow?".

Recall that the goal of MLE is to find a particular value for the parameter θ. By contrast, in Bayesian inference, we are interested in the distribution of the parameter $p(\theta|Y)$ after the data Y was observed. $p(\theta|Y)$ is called a *posterior distribution,* emphasizing that it is a distribution computed *after* the data is observed. In some contexts, it is also called a *posterior probability.* Correspondingly, a distribution $p(\theta)$ before any data is observed is called a *prior distribution*, or a *prior probability.* If there are more than one parameters, the posterior distribution for these parameters $\theta_1, \ldots, \theta_K$ becomes a joint distribution $p(\theta_1, \ldots, \theta_K|Y)$. In this case, $p(\theta_1|Y)$ is the posterior distribution of a single paramter θ_1, which is obtained by marginalizing the joint distribution.

Using Bayes' theorem, a posterior distribution can be obtained as follows:

$$p(\theta|Y) = \frac{p(Y|\theta)p(\theta)}{p(Y)} \propto p(Y|\theta)p(\theta)$$

[1] In the case where the number of parameters is 100, if 10 values are tried for each parameter, the number of combinations will be 10^{100} ways. Therefore, there is a method to create an initial value using a random value. However, this method does not guarantee to explore the parameter space sufficiently.

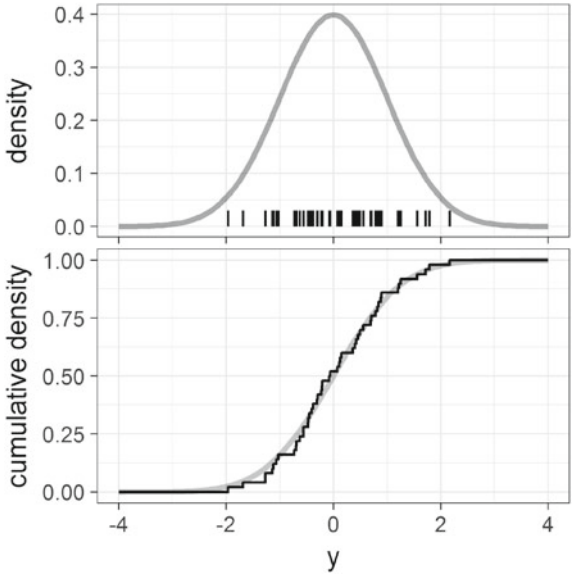


Fig. 2.2 An example of an MCMC sample drawn from a standard normal distribution. Top: the gray curve is the standard normal distribution, and the black vertical lines on the bottom represent 50 draws from the normal distribution. Bottom: the black line is the cumulative distribution function that are obtained from the draws (as shown in the vertical lines on the top), and it approximates the gray curve, which is the cumulative distribution function of the original density function

That is, the posterior distribution $p(\theta|Y)$ is proportional to the product of likelihood $p(Y|\theta)$ and the prior distribution $p(\theta)$. $p(Y)$ is a constant value that depends only on the observed data Y and does not depend on θ. Therefore, $p(Y|\theta)p(\theta)$ determines the posterior distribution of θ, and $p(Y)$ can be considered as a normalization constant. Computing the likelihood and the prior distribution is straightforward, but computing $p(Y)$ is usually not easy. Therefore, analytical solutions for a posterior distribution are usually too complex or intractable.

MCMC enables us to obtain the posterior distributions, by performing numerical approximation.[2] MCMC ignores the normalization constant $p(Y)$ and draws many random values in proportion to $p(Y|\theta)p(\theta)$, and uses the set of these *draws* instead of the posterior distribution itself. In this book, the set of random draws obtained by MCMC are called an *MCMC sample*. In Fig. 2.2, we show the MCMC sample drawn from a standard normal distribution. We usually require the MCMC sample size (the number of draws) to be greater than 1000 to ensure that the approximated posterior distribution is sufficiently close to the posterior distribution. From the Central Limit Theorem, we know that improving the posterior distribution accuracy by one unit requires increasing the number of draws a hundred times. Based on the desired precision, it should be adjusted accordingly.

The MLE method search for the parameters with the largest likelihood based on the log-likelihood, whereas Bayesian inference explores the parameter space based on the log posterior probabilities obtained by log transformation of posterior probabilities. The log posterior probability is expressed by the following equation:

[2] Other methods for obtaining posterior distribution are such as Variational Bayes and Expectation Propagation.

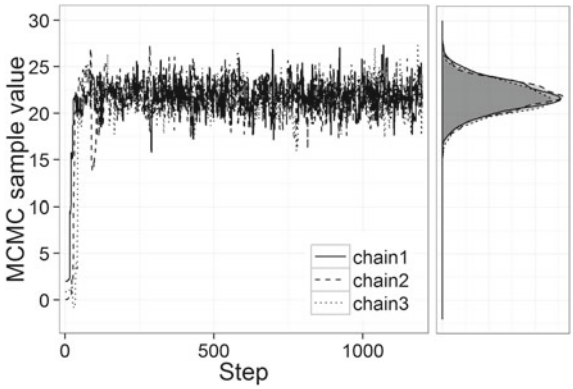


Fig. 2.3 The trace plot on the left shows the chains. The x-axis represents the number of iterations, the y-axis represents the value of the draws, and each line represents a chain. The density plot on the right is probability density calculated from the draws after discarding the first 200 iterations

$$\log p(\theta|Y) = \log p(Y|\theta) + \log p(\theta) + \text{const.},$$

where const. denotes a constant term. In Stan, "$\log p(Y|\theta) + \log p(\theta)$" is denoted as the log posterior probability (`lp__`), which drops the irrelevant constant term for sampling. Note that $\log p(Y|\theta)$ is the log-likelihood.

Two most widely known methods for drawing an MCMC sample are Metropolis–Hastings sampling and Gibbs sampling. As we will mention in Sect. 3.2, Stan uses another method called NUTS. All these methods start from setting initial values of the parameters. Then, they iterate the step where they update the values of all the parameters by drawing random values based on the current values. These parameter values compose an MCMC sample. A sequence of the drawn values along the number of iterations is called a *chain*. Figure 2.3 is an example of typical chains. The left side shows a *trace plot*, with the index of iterations on the x-axis and the value of the draws for the parameter on the y-axis.

Figure 2.3 shows three chains that start from different initial values. For each chain, the initial 200 iterations of MCMC search for the space where the value of $p(Y|\theta)p(\theta)$ (remember this is proportional to the posterior distribution of the parameter) is large. Generally, these earlier 200 iterations are discarded, because they depend heavily on the initial value and cannot be regarded as the sample from the posterior distribution. These discarded initial iterations are called a *warmup* or *burn-in*.[3] In practice, the length of the warmup period depends on the problem of interest. From the discussions earlier, we already know that the draws after the warmup period can be regarded as a sample from the posterior distribution. The probability density on the right side of Fig. 2.3 is the posterior distributions calculated from the draws after the warmup period. Notably in this plot, the shapes of these posterior density plot do not change even when the number of total iterations increases, and the shapes are identical for all chains. Therefore, it can be said that this posterior distribution *converge*s to a steady distribution. A posterior distribution is not always

[3] Stan team sometimes use the term *adaptation* because Stan tunes the algorithm parameters during the warmup period.

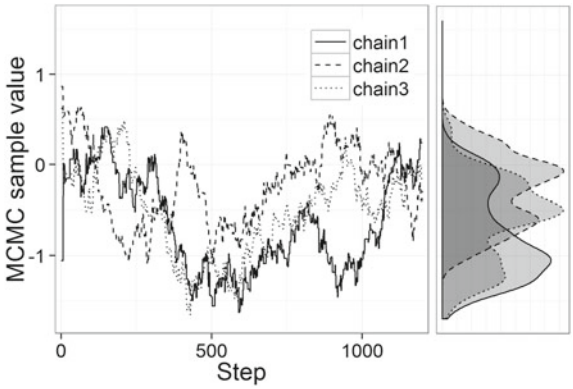


Fig. 2.4 An example that has poor convergence. The legend is the same as in Fig. 2.3

a symmetrical, bell-shaped distribution: it can be a skewed distribution and it can also have several peaks.

An example of a posterior distribution that did not successfully converge is shown in Fig. 2.4. As the number of iterations increases, the shape of the posterior distribution keeps changing, and the distributions calculated are different across each chain as well. Therefore, the posterior distribution has poor convergence. In such a case, the autocorrelation of a sequence is typically large. In other words, the value at each iteration highly depends on the value at the earlier iterations. And hence, the effective MCMC sample size does not increase. If autocorrelation is not very high, we can improve the convergence by keeping only one draw from every several iterations, a process called *thinning*. However, in the case of the chains shown in Fig. 2.4, it will probably not converge even if we increase the number of iterations or thin the sequences, and some modifications are needed for the model. In Chap. 9, we will further discuss the approaches of modifying the model to improve MCMC convergence.

Convergence assessment is critical in the Bayesian modeling process. If the posterior distribution has poor convergence, changing the initial values or the random seed would produce considerably different posterior distributions, which will cause non-reproducible results of the analysis. Therefore, if MCMC convergence is poor, the MCMC sample obtained should not be used for the further analysis.

2.4 Bayesian Confidence Interval, Bayesian Predictive Distribution, and Bayesian Prediction Interval

As mentioned in the previous section, since Bayesian statistics assumes probability distributions for all parameters, it is easy to calculate the "width" of the parameter distributions. We call this width *Bayesian confidence interval* in this book in order to clearly distinguish it from the confidence interval in traditional statistics. It is also

called *credible interval*. The interval between $\alpha/2$ percentile and $(1-\alpha)/2$ percentile of the posterior distribution is called $(1-\alpha)\%$ *Bayesian confidence interval*. When we use MCMC, we can easily obtain Bayesian confidence intervals by calculating the quantiles of the draws.

Next, we give an example to explain the predictive distribution. In order to determine the efficacy of a new drug, we conducted a clinical trial with 20 patients. Assume that the result shows that the new drug successfully cured 14 patients out of 20 (i.e., $Y = 14$, $N = 20$). Suppose that the number of cured people y follows the following Binomial distribution:

$$\text{Binomial}(y|N,\theta) = \frac{N!}{y!(N-y)!}\theta^y(1-\theta)^{N-y}$$

Now we are planning to conduct the second trial using the same drug for 100 patients. Based on the result from the first trial, how many patients will be cured in the second experiment? Or more generally, what is the probability that y patients will be cured by the second trial? To answer this question, we need to consider a *predictive distribution* of the new data, instead of the distribution of the parameters.

In the case of MLE, a predictive distribution is obtained by substituting the maximum likelihood estimate of the parameter $\widehat{\theta}$ into $p(y|\theta)$, which is the probability distribution for generating data. Mathematically, this is:

$$p_{\text{pred}}(y|Y) = p\left(y|\widehat{\theta}\right)$$

Such a predictive distribution is sometimes called plug-in predictive distribution. In this example, since $\widehat{\theta}$ is calculated as $\widehat{\theta} = 14/20 = 0.7$, the predictive distribution is:

$$\text{Binomial}(y|N=100,\theta=0.7) = \frac{100!}{y!(100-y)!}0.7^y(1-0.7)^{100-y}$$

This indicates that the probability of $y = 70$ in the second clinical trial is about 0.087, which is obtained by substituting y with 70.

The limitation of this approach is that it does not take into account the uncertainty of the parameter estimation. For example, if there are 140 cured patients out of 200 in the first experiment, it still gives the same predictive distribution as the above case. In the case of MLE, the uncertainty from the result of the first clinical trial is not being considered, and therefore, the results overfit to this particular data generated from one single trial. To avoid overfitting, a method called scenario analysis is often used. In the scenario analysis, several analyses are performed under different scenario assumptions, and in some cases, the results from different scenarios are integrated to obtain the final result. For instance, in the above example, one would perform scenario analysis by assuming that the weight of $\widehat{\theta} = 0.7$ is 0.5 (scenario of MLE), the weight of $\widehat{\theta} = 0.5$ is 0.4 (pessimistic scenario of small clinical effect), and the weight of $\widehat{\theta} = 0.6$ is 0.1 (optimistic scenario of large clinical effect). The predictive distribution

in this example is considered as a mixture distribution of plug-in distributions in each scenario, as follows:

$$\begin{aligned} p_{\text{pred}}(y|Y) = &\text{Binomial}(y|N, \theta = 0.7) \times 0.5 + \\ &\text{Binomial}(y|N, \theta = 0.5) \times 0.4 + \\ &\text{Binomial}(y|N, \theta = 0.4) \times 0.1 \end{aligned}$$

Expanding this scenario analysis, first we calculate the probabilities of all possible scenarios given the data, $p(\theta|Y)$, which is the posterior distribution of θ. Then we use $p(\theta|Y)$ as the weight for each scenario, and get the weighted sum of the probability distribution for generating data $p(y|\theta)$. The resulting mixture distribution is a *Bayesian predictive distribution.* Writing it in a mathematical formula, we have:

$$p_{\text{pred}}(y|Y) = \int p(y|\theta)p(\theta|Y)d\theta \tag{2.4}$$

In other words, the predictive distribution in Bayesian inference is the average of the probability distributions for generating data over the posterior distribution of parameters. Thus, it can be said that the predictive distribution in Bayesian inference considers the uncertainty of parameter estimates. In this book, when we simply use the term "predictive distribution", it refers to the posterior predictive distribution after the data Y is obtained.

When we use MCMC, we obtain the posterior distribution $p(\theta|Y)$ in the form of MCMC sample. Therefore, instead of using Eq. (2.4), we can use the sum over the draws to replace the integral with respect to θ in this equation, and get the predictive distribution. By approximating $p(y|\theta)$ by random draws, we can also generate the draws of the predictive distribution. Specifically, for each draw $\theta_{(1)}, \ldots, \theta_{(i)} \ldots, \theta_{(N_{\text{ms}})}$ from the posterior distribution $p(\theta|Y)$, drawing a single random value $y_{(i)}$ from the probability distribution $p\big(y|\theta_{(i)}\big)$ generates $y_{(1)}, \ldots, y_{(i)} \ldots, y_{(N_{\text{ms}})}$, where N_{ms} is the MCMC sample size (the number of draws) from the posterior distribution. Thus, the resulting set of $y_{(1)}, \ldots, y_{(i)} \ldots, y_{(N_{\text{ms}})}$ can be regarded as the MCMC sample from the predictive distribution $p_{\text{pred}}(y|Y)$. For instance, $p\big(y|\theta_{(i)}\big)$ has the form of $\text{Binomial}\big(y|N, \theta_{(i)}\big)$ in the above clinical trial example. See the next Technical Point for the mathematical details of the MCMC sample of the predictive distribution.

A prediction interval can be calculated from this MCMC sample in almost the same way as the Bayesian confidence interval. We call such a prediction interval as a *Bayesian prediction interval.* Similarly, the interval between $\alpha/2$ percentile and $(1-\alpha)/2$ percentile of the predictive distribution is called a $(1-\alpha)\%$ *Bayesian prediction interval.*

With a simple model and a large number of draws, the predictive distribution obtained by MLE and the predictive distribution obtained by Bayesian inference are very close. However, when a model is complex, the predictive distribution obtained from these two methods may be different: the predictive distribution obtained by

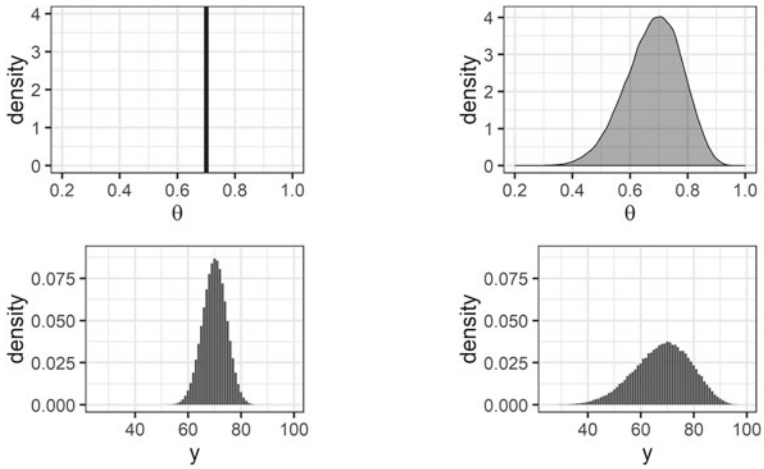


Fig. 2.5 Comparison between MLE and Bayesian inference. Left and right panels show the results from MLE and Bayesian inference, respectively, and top and bottom panels show the posterior distribution of parameter θ and the posterior predictive distribution $p_{\text{pred}}(y|Y)$, respectively

MLE tends to have the issue of overfitting and poorer predictive performance. We show the predictive distributions obtained using MLE and Bayesian inference from the clinical trial example in Fig. 2.5.

Instead of using a posterior distribution, we can also use prior distribution for weighting and obtain a prior predictive distribution:

$$p_{\text{pred}}(y|.) = \int p(y|\theta)p(\theta)d\theta$$

It is denoted as $p_{\text{pred}}(y|.)$ to specify that data is not used to obtain this distribution. If there are candidate models of interest, obtaining predictive distributions by simulation is also useful to know what properties the models have. Further, it is also possible to calculate Bayesian prior prediction intervals from the prior predictive distributions.

You may wonder why we would calculate such intervals, instead of a full probability distribution, which has more information. Of course, when the number of parameters is small, it is more straightforward to visualize the distributions. But when the number of parameters is large, it can be hard to visualize a high-dimensional posterior distribution and understand its characteristics. Using Bayesian confidence intervals computed from the posterior distribution would make it easier to understand the characteristics of the posterior distribution. For example, when analyzing time series data, instead of visualizing the posterior distributions at the time points, it is much easier to visualize their Bayesian confidence intervals to comprehend their temporal transition.

Technical Point: Expression Using the Dirac Delta Function

Mathematically, a distribution approximated by random draws can be expressed as a mixture of the Dirac delta functions.[4] For example, a posterior distribution can be expressed by using Dirac delta functions as follows:

$$p(\theta|Y) = \frac{1}{N_{\text{ms}}} \sum_{i=1}^{N_{\text{ms}}} \delta\big(\theta - \theta_{(i)}\big) \tag{2.5}$$

where N_{ms} is the MCMC sample size (the number of draws), and $\theta_{(i)}$ is the i-th random draw. The Dirac delta function $\delta(x)$ is a normal distribution function whose peak is extremely sharp at $x = 0$. It has the following property:

$$\int_{-\infty}^{\infty} f(x)\delta(x-a)dx = f(a)$$

By using this property, the average of $f(\theta)$ over the posterior distribution can be written as:

$$\begin{aligned}\int f(\theta)p(\theta|Y)d\theta &= \int f(\theta)\frac{1}{N_{\text{ms}}}\sum_{i=1}^{N_{\text{ms}}}\delta\big(\theta-\theta_{(i)}\big)d\theta \\ &= \frac{1}{N_{\text{ms}}}\sum_{i=1}^{N_{\text{ms}}} f\big(\theta_{(i)}\big)\end{aligned}$$

In other words, by expressing the posterior distribution as random draws, the integral with respect to θ can be replaced with computing the sum over the draws.

Now let us look at the case where we express the predictive distribution as random draws. By using the property of the Dirac delta function, the predictive distribution can also be obtained by replacing the integral with the sum over draws:

$$\begin{aligned}p_{\text{pred}}(y|Y) &= \int p(y|\theta)p(\theta|Y)d\theta \\ &= \int p(y|\theta)\frac{1}{N_{\text{ms}}}\sum\nolimits_{i=1}^{N_{\text{ms}}}\delta\big(\theta-\theta_{(i)}\big)d\theta \\ &= \frac{1}{N_{\text{ms}}}\sum_{i=1}^{N_{\text{ms}}} p\big(y|\theta_{(i)}\big)\end{aligned} \tag{2.6}$$

[4] See https://en.wikipedia.org/wiki/Dirac_delta_function for details.

For instance, in the above example of the clinical trials, the distribution for generating data was a Binomial distribution:

$$p\left(y|\theta_{(i)}\right) = \text{Binomial}\left(y|N, \theta = \theta_{(i)}\right)$$

Therefore, the predictive distribution becomes a mixture of Binomial distributions as follows:

$$p_{\text{pred}}(y|Y) = \frac{1}{N_{\text{ms}}} \sum_{i=1}^{N_{\text{ms}}} \text{Binomial}\left(y|N, \theta = \theta_{(i)}\right)$$

However, using $p_{\text{pred}}(y|Y)$ expressed in this format makes the later calculation difficult.[5] Thus, we would like to approximate this distribution by random draws. For this purpose, we draw M random values $y_{(i,1)},\ldots, y_{(i,M)}$ from $p\left(y|\theta_{(i)}\right)$ for each i, and approximate $p\left(y|\theta_{(i)}\right)$ by using these M draws as follows:

$$p\left(y|\theta_{(i)}\right) = \frac{1}{M} \sum_{m=1}^{M} \delta\left(y - y_{(i,m)}\right)$$

By substituting this equation into Eq. (2.6), we can write the predictive distribution as:

$$p_{\text{pred}}(y|Y) = \frac{1}{N_{\text{ms}} M} \sum_{i=1}^{N_{\text{ms}}} \sum_{m=1}^{M} \delta\left(y - y_{(i,m)}\right)$$

Compared to (2.4), this expression formulates that the predictive distribution $p_{\text{pred}}(y|Y)$ can be expressed with the $N_{\text{ms}}M$ random draws $y_{(1,1)},\ldots,y_{(1,M)},y_{(2,1)},\ldots,y_{(2,M)}, \ldots\ldots,y_{(N_{\text{ms}},1)}, \ldots,y_{(N_{\text{ms}},M)}$. Regardless of how small M is, when N_{ms} is large enough, this expression is a good approximation of the predictive distribution. Therefore, in this book we will use $M = 1$, and use $y_{(i)}$ to denote a single random draw from $p\left(y|\theta_{(i)}\right)$.

[5] For example, if you calculate the average of $f(y)$ over $p_{\text{pred}}(y|Y)$, the result will be:

$$\int f(y) p_{\text{pred}}(y|Y) dy = \frac{1}{N_{\text{ms}}} \sum_{i=1}^{N_{\text{ms}}} \left[\int f(y) p\left(y|\theta_{(i)}\right) dy \right]$$

This means that the integration has to be done N_{ms} times.

In short, for each draw $\theta_{(1)}, \ldots, \theta_{(i)} \ldots, \theta_{(N_{\mathrm{ms}})}$ from the posterior distribution $p(\theta|Y)$, drawing a single random value $y_{(i)}$ from the probability distribution $p\big(y|\theta_{(i)}\big)$ generates a set of draws $y_{(1)}, \ldots, y_{(i)} \ldots, y_{(N_{\mathrm{ms}})}$. This set of $y_{(i)}$ can be seen as the MCMC sample from the predictive distribution $p_{\mathrm{pred}}(y|Y)$.

2.5 Relationship Between MLE and Bayesian Inference

A *non-informative prior distribution* includes a sufficiently wide uniform distribution, or a sufficiently flat normal distribution. The value of θ^* at which posterior distribution $p(\theta|Y)$ is maximized is called *maximum a posteriori estimate* (*MAP estimate*). There is a close relationship between the maximum likelihood estimate and the MAP estimate when using a non-informative prior distribution $p(\theta)$. Because $p(\theta)$ can be regarded as constant over a wide range, the following holds for the MAP estimate θ^*:

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, p(\theta|Y) = \underset{\theta}{\operatorname{argmax}}\, [p(Y|\theta)p(\theta)] = \underset{\theta}{\operatorname{argmax}}\, p(Y|\theta),$$

where $\operatorname{argmax}_\theta f(\theta)$ returns θ that maximizes $f(\theta)$. Because $p(Y|\theta)$ in the last equation is the likelihood, θ^* is equal to the maximum likelihood estimate. That is, a result obtained using non-informative prior distributions is consistent with the result from the traditional statistics using MLE.

Let's look at the relationship between MLE, Bayesian inference, and MAP estimate using the problem and data we used in Sect. 2.2. Recall that the model formula we used was:

$$Y[n] \sim \mathcal{N}(\mu, 1) \qquad n = 1, 2, \ldots, 20$$

And the log-likelihood derived from this model formula was:

$$\log p(Y|\theta) = \sum_{n=1}^{20} \log \mathcal{N}(Y[n]|\mu, 1)$$

In Bayesian statistics, all parameters are random variables and they follow certain probability distributions. Here, we assume that the parameter μ follows a non-informative prior distribution (sufficiently flat normal distribution), then the model can be given by:

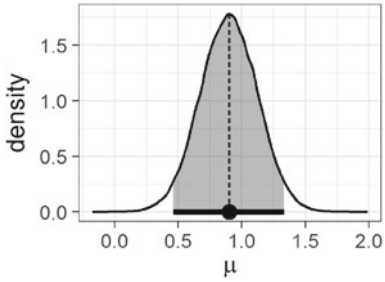


Fig. 2.6 The posterior distribution of μ. The bell-shaped distribution with the solid line is the posterior distribution, the gray area (from 0.46 to 1.34) represents the 95% Bayesian confidence interval of μ, and the position of the dotted line (0.90), which is the mode of the distribution, represents the MAP estimate of μ

Model Formula 2.1

$$\begin{aligned} Y[n] &\sim \mathcal{N}(\mu, 1) \qquad n = 1, 2, \ldots, 20 \\ \mu &\sim \mathcal{N}(0, 100) \end{aligned}$$

The posterior distribution (ignoring the normalization factor) derived from this model formula can be written as follows:

$$\begin{aligned} p(\theta|Y) &\propto p(Y|\theta)p(\theta) \\ &= \left[\prod_{n=1}^{20} \mathcal{N}(Y[n]|\mu, 1)\right] \times \mathcal{N}(\mu|0, 100) \end{aligned}$$

Therefore, the log posterior probability is:

$$\log p(\theta|Y) = \left[\sum_{n=1}^{20} \log\mathcal{N}(Y[n]|\mu, 1)\right] + \log\mathcal{N}(\mu|0, 100) + \text{const.}$$

Again, $\log\mathcal{N}(\mu|0, 100)$ can be considered as constant, and therefore, the same μ value maximizes both log-likelihood and log posterior probability. Figure 2.6 shows the posterior distribution of μ obtained by Stan using the data in Sect. 2.2 and Model Formula 2.1. The Bayesian confidence interval and the MAP estimate are also shown.

2.6 Selection of Prior Distributions in This Book

So far, we have seen that the combination of Bayesian statistics and MCMC enabled the flexible and interpretable statistical modeling, but there are some aspects that we need to pay extra attentions to. One of them is about the convergence of MCMC, which are mentioned in Sect. 2.3. Especially, there is no standard process to improve

a model if MCMC does not converge, and this could be hard to tackle when we have limited experience in modeling. Another is how to select the prior distributions, which is the focus of this section.

There are different criterions for the selection of the prior distributions. In this book, we consider that the non-informative prior distributions described in the previous section as the first option. If there is no clear information about the parameters, we should avoid using prior distributions with strong subjectivity that likely to generate specific values. This is because the choice of strongly subjective prior distributions often varies depending on analysts, which causes bias and reduces reproducibility of the analysis. If you use a strongly subjective prior distribution, it would be helpful to compare that result with the results using non-informative prior distribution, to make sure that we are conducting a robust analysis.

However, this is not the case if information or experience about parameters are available. For example, we can reasonably presume that the height of a person is a positive value and does not exceed 3 m. So if we want to estimate the mean height of a group of people, we could use a uniform distribution likeUniform$(0, 3)$.[6] In this case, it is recommended to use weakly informative priors that partially reflects such information. More about the usage of weakly information prior distributions is discussed in Sect. 9.2.

In the past when the computing speed was low, a conjugate distribution of the likelihood was mainly used as the prior distribution, because the posterior distribution can be solved analytically. Also, the conjugate prior enables us to update the posterior distribution every time when the new data is obtained. This updating method is called Bayesian updating. Recently, with the rapid improvement of the computational power, the benefits of using a non-conjugate prior have surpassed the benefits of using the conjugate prior. The non-conjugate prior enables us more flexible and more interpretable modeling. One example where we may want to use a non-conjugate instead of a conjugate prior is when we model the variance in the hierarchical models (see Chap. 8 and Sect. 9.2.2). A conjugate prior in this case might not be a proper choice because the result may not be robust in many cases. Because the calculation time in Stan would not be faster by using the conjugate prior, in this book, we did not limit our choice to conjugate prior distributions. Note that in Stan, calculation of a posterior distribution should be done using all the available data at the time, instead of using Bayesian updating.

References

Casella, G., & Robert, C. (2005). *Monte Carlo statistical methods* (2nd ed.). Springer.

Gelman, A., Hill, J., & Vehtari, A. (2020). *Regression and other stories.* Cambridge University Press.

McGrayne, S. B. (2011). *The theory that would not die: How Bayes' rule cracked the enigma code, hunted down Russian Submarines, and emerged triumphant from two centuries of controversy.* Yale University Press.

[6] In Sect. 9.2, we will recommend $\mathcal{N}(1.5, 1.5)$ instead of Uniform$(0, 3)$.

Part II
Introduction to Stan

Chapter 3
Overview of Stan

In this chapter, we will introduce probabilistic programming languages and Stan. Each section has the following content:

- Section 3.1 explains the concepts of the probabilistic programming languages, which facilitate model description and help try many different models.
- Section 3.2 briefly introduces Stan and its characteristics.
- Section 3.3 explains why we use R or Python with Stan. The reasons are mostly for efficiency and convenience in visualization, simulation, and data processing.
- Section 3.4 illustrates how to install Stan and set up recommended development environment.
- Section 3.5 explains the basic grammar of Stan, with the focuses on block structures (`data`, `parameters` and `model` block), probabilistic generation, and `for` loop statement. We also give suggestions on the coding styles.
- Section 3.6 introduces `lp__` and `target`, which are the key variables that express posterior probability in Stan internal. We show their usage with a simple example.

3.1 Probabilistic Programming Language

Generally speaking, in order to estimate parameters in a probabilistic model, an analyst must derive complicated math formulas and implement its algorithm in a programming language. Therefore, it tends to induce errors or bugs when the formulas become complicated. And because the derivation and implementation processes depend on the model, we need to modify the formulas and its implementation every time the model changes.

Clearly, the above process is very hard for analysts who use statistical modeling. However, it is the probabilistic programming language that overcomes this issue. A *probabilistic programming language* is a programming language whose purpose is to fit probabilistic models to data. It typically has specialized functions for probability

© Springer Nature Singapore Pte Ltd. 2022

K. Matsuura, *Bayesian Statistical Modeling with Stan, R, and Python*,
https://doi.org/10.1007/978-981-19-4755-1_3

distributions, likelihood computation, and efficient algorithms for parameter estimation. All a user has to do is to write the model as programming code and pass the data to the program, and it will almost automatically estimate parameters. By separating model description from the complicated process of calculating the estimation, it makes a model more readable and produces much fewer errors and bugs. Therefore, we can focus on the model description. The power of the probabilistic programing languages is especially maximized when analysts need to try many different models.

If a model is relatively simple, we can estimate parameters by using the basic functions[1] of R without using probabilistic programming language. The estimation results using these R functions and probabilistic programming languages are the same. Using conventional R packages,[2] we can also deal with complex models that include group differences or individual differences. However, there are some caveats at using these existing packages. For instance, every package and function have particular ways of model description and it is a time-consuming process to master one by one. Also, when analyzing real data, we need to find a suitable model and functions from these packages. If we cannot find a suitable model, we cannot proceed any further analysis, because the conventional R packages have limited model extensibility. For these reasons, it may be more beneficial to learn how to describe and implement models in a consistent way with probabilistic programming languages, rather than learning how to use every package and functions.

3.2 Why Stan?

For a long time, WinBUGS[3] and JAGS[4] have been the mainstream probabilistic programming languages. They estimate parameters using an algorithm called MCMC, and the estimation results are obtained in the form of MCMC sample drawn from the posterior distribution (see Chap. 2). WinBUGS and JAGS have R interfaces.[5] However, the error messages in WinBUGS are not easy to understand, and this makes modifying models difficult. In addition, WinBUGS works only on Windows, and it is unknown whether it will continue to be available since its last update dates back to 2007. JAGS has relatively easy-to-understand error messages. However, similar to WinBUGS, JAGS has not been updated much either, and available references and examples are also limited.

We highly recommend using Stan, which is another probabilistic programming language that has been actively developed on GitHub since about 2012 by Andrew Gelman, Bob Carpenter, Daniel Lee, Ben Goodrich, Michael Betancourt et al. (Stan Development Team, 2019). Like WinBUGS and JAGS, Stan performs sampling

[1] `lm()`, `glm()`, and so on.

[2] `{glmmML}`, `{lme4}`, `{nlme}`, `{MCMCpack}`, `{bayesm}`, and so on.

[3] http://www.mrc-bsu.cam.ac.uk/software/bugs/the-bugs-project-winbugs/.

[4] http://mcmc-jags.sourceforge.net/.

[5] For example, `{R2WinBUGS}` and `{rjags}`.

from the posterior distribution. Different interfaces have been released, including CmdStanR (interface with R), CmdStanPy (interface with Python), as well as interfaces with Matlab, Julia, and Mathematica.

One of the most important features of Stan is that it adopts No-U-Turn Sampler (NUTS) as the default algorithm for computing the estimation. NUTS is an implementation of Hamiltonian Monte Carlo (HMC), which is one type of MCMC. The strength of NUTS is that it can sample efficiently even when the number of parameters is large. Compared to WinBUGS and JAGS, one iteration of sampling using NUTS takes more time due to its elaborate algorithm, but autocorrelation between iterations is lower. Therefore, MCMC calculation that requires 100,000 iterations in WinBUGS or JAGS can often be done in 1000 iterations in Stan. As a whole, the computation time is greatly shortened. Also, thanks to its efficient algorithm, Stan can sample appropriately even for complex models where WinBUGS or JAGS would stop sampling in error. The only disadvantage is that discrete parameters (parameters that take discrete values) cannot be used in Stan, because there is no theoretical solution to deal with such parameters in HMC. Sometimes we can express the equivalent model by marginalizing out such parameters in the model description. This technique will be introduced in Chap. 10.

Besides NUTS, we can use two other parameter estimation methods: Automatic Differentiation Variational Inference (ADVI) and Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS). ADVI is an implementation of variational Bayes methods, and it draws from a distribution that approximates the posterior distribution. Note that we cannot determine whether the approximation is good from results of ADVI only. L-BFGS is an algorithm that performs MAP estimation, which is one type of point estimation (see Sect. 2.5). Compared with NUTS, these estimation methods require shorter calculation time but poorer estimation accuracy. If the amount of data is large and a model is complex, NUTS may not be able to complete the calculation within a reasonable time. In such a case, ADVI or L-BFGS can be an alternative method.

Stan's strengths are not limited to its performance aspects described above. For instance, Stan also has much clearer error messages compared to WinBUGS and JAGS, thus makes the debuging process much easier. Also, Stan is well-documented and well-maintained. It has references[6] with over 600 pages in total, and the Q & A as well as the discussion on Stan's discouse website[7] is highly active. In addition, there are numerous modeling examples on the official GitHub[8] page, which include some examples in Bayesian textbooks, and other more advanced case studies, which is always updated actively.

[6] http://mc-stan.org/documentation/.

[7] https://discourse.mc-stan.org.

[8] https://github.com/stan-dev/example-models.

3.3 Why R and Python?

Both R[9] and Python[10] are popular scripting languages and have many packages. R has particular strengths in data processing and visualization, while Python is an object-oriented language and is suitable for system development. Also, Python has high-quality deep learning and machine learning packages. CmdStanR is an R package to easily run Stan models from R. The sampling result can be used as an R object. CmdStanPy is a Python package for Stan just like CmdStanR.

Using Stan from R or Python is also a very good choice for efficient statistical modeling workflows. This is mainly because they can easily conduct visualizations and simulations. Statistical modeling requires frequent visualizations, and their graphical functions are very helpful. Simulation is also helpful to understand the behavior of complex models. In addition, powerful data processing functions in R and Python are essential to use MCMC sample in the estimation result smoothly.

Technical Point: Comparison of Probabilistic Programming Languages

Software name	Implementation on own	MCMCpack (R)	WinBUGS/OpenBUGS (BUGS)	JAGS (BUGS)	Stan	PyMC (Python)	Infer.NET (C#, F#)	MCMC proc (SAS)
General versatility	C	B	A	A	A	A	A	A
Modeling with less bugs	C	A	A	A	A	A	A	A
Readability of error messages	–	A	C	B	A	A	A	A
High speed MCMC	A+	A	A	A	A+	B	C[1]	B
Variational inference	A+	–	–	–	A	A	A	-
References and examples	–	B	A	B	A+	B	A	A
Activities of development	–	B	C	B	A	A	B	B
Others			2		3			4

1 only conjugate prior distributions are allowed
2 includes a plugin that can handle spatial structure easily
3 discrete parameters cannot be used
4 not free

A+, A, B, and C are assessed based on web information and experiences. We generally recommend Stan to solve practical problems, and JAGS for beginners

[9] https://www.r-project.org/.
[10] https://www.python.org/.

of statistical modeling. If there is a need to work on a particular problem for a long period of time, it would be helpful to implement on own.

3.4 Preparation of Stan, CmdStanR, and CmdStanPy

Installation procedures change time to time, therefore in order to check the latest information, it is a good idea to go to the official web site that Stan development team provided.[11,12]

The development environment is also important for statistical modeling. An editor and version control would make statistical modeling efficient. They will be briefly described here.

Editor

An editor is software for typing source code. You can use any editor you like. However, it is desired that an editor has the following functions: syntax highlighting of a Stan code, a warning for parentheses mismatches, and complementation of function names and variable names. These functions will help reduce the number of typos and other syntax issues dramatically. We recommend RStudio in this book. For descriptions of installing and using RStudio,[13] please refer the RStudio main website.

Version Control

The management of source code change histories is called version control. The recommended websites for version control are GitHub[14] or Bitbucket.[15] Although it is not easy, it is worth spending some time to learn how to use these tools, not only for programmers, but also for researchers and consultants. It is important because you can return to the previous version by typing a single command and you can easily obtain the differences between the current version and any older version. If you need to share source code with others, version control will be even more powerful. For information on how to use them, refer to related websites and books.

[11] https://mc-stan.org/cmdstanr/.
[12] https://cmdstanpy.readthedocs.io/en/latest/index.html.
[13] https://www.rstudio.com/.
[14] https://github.com/.
[15] https://bitbucket.org/.

3.5 Basic Grammar and Syntax of Stan

3.5.1 Block Structure

The most basic structure of a Stan code is composed from three blocks (`data`, `parameters`, `model`) as follows:

```
data {
  declaration of data Y
}
parameters {
  declaration of parameters θ to be sampled.
}
model {
  description of likelihoods p(Y|θ)
  description of prior distributions p(θ)
}
```

The execution of the above code produces the MCMC sample of the parameter θ. The sample of θ is drawn at a frequency proportional to the value of the posterior distribution $p(\theta|Y) \propto p(Y|\theta)p(\theta)$. In Stan, θ includes not only the parameter that determines the position and shape of a probability distribution, but also all variables whose values are unknown and should be estimated.

The block structure that requires such a declaration may seem tedious, but it clarifies which data must be passed from R or Python to Stan and which variables are to be estimated. In addition to these three blocks, Stan has four additional blocks for convenience, which will be explained in later examples.

Here is our regular way to write a Stan code. First, we write the likelihood part (and the prior distribution part) in `model` block. For the variables that appeared in the likelihood part, we write the variables that have the data in data block and the remaining variables in `parameters` block. It works better in this order, rather than starting from `data` and `parameters` blocks and try to write everything down before knowing what variables will be used in the model.

When you write and run a Stan code, it is converted into C++ code, compiled, and run for MCMC sampling. The order of the blocks is fixed because C++ Code requires the sequential order. Variables should be declared and used sequentially from the top line.

3.5.2 Basic Grammar and Syntax

Here we explain the basics of Stan's grammar through the code that solves the problem discussed in Sect. 2.2. The purpose is to fit a normal distribution with fixed

SD of 1 to the data (see Fig. 2.1 , middle). We can express a model as follows (see Sect. 2.5).

Model Formula 3.1

$$Y[n] \sim \mathcal{N}(\mu, 1) \qquad n = 1, 2, \ldots, N \tag{3.1}$$

$$\mu \sim \mathcal{N}(0, 100) \tag{3.2}$$

where Y is data, N is the number of data points (20 in this case), the subscript n is the index of each data point, and $\mathcal{N}$ represents a normal distribution. Equation (3.1) represents that we assume each data point is independently generated from a normal distribution with mean of μ and SD of 1. μ is the only unknown parameter in this example and is to be estimated from the data. As mentioned in the earlier chapter, in Bayesian statistics, all parameters follow certain probability distributions. Here we assume that μ follows a noninformative prior distribution (a sufficiently flat normal distribution with mean of 0 and SD of 100).

The implementation of Model Formula 3.1 in Stan is as follows:

```
 1  data {
 2    int N;
 3    vector[N] Y;
 4  }
 5
 6  parameters {
 7    real mu;
 8  }
 9
10  model {
11     for (n in 1:N) {
12       Y[n] ~ normal(mu, 1);
13     }
14     mu ~ normal(0, 100);
15  }
```

Now we have written the above code starting from `model` block as we mentioned in the previous subsection. Let us look at the explanation of each block in this order.

model block

Lines 11–13 correspond to (3.1) of Model Formula 3.1. "∼" in the code represents probabilistic generation as well as "∼" in model formulas, here is denoting the likelihood of data `Y[n]`. "`;`" is required at the end of each sentence. The line 11 is called a `for` statement that repeats the contents within "{}". It is also called a `for` loop statement. In this case, this `for` loop statement means that the subscript n is repeatedly assigned from 1 to N.

Line 14 corresponds to (3.2) where the prior distribution of `mu` is set to a noninformative prior distribution. Without this line, Stan would still work perfectly, because

having no prior description in the Stan code is equivalent to setting a sufficiently wide uniform distribution. Therefore, we omit the description of noninformative prior distributions in model formulas and Stan codes from now on.

parameters block

We declare parameters to be estimated with their types in this block (`mu` in this example). The `real` of "`real mu`" means `mu` can only store a real value. As mentioned in Sect. 3.2, Stan cannot use discrete parameters, that is, integer parameters.

data block

In this block, we declare the data that is used in `model` block. Specifically in this example, we declare `N` as the number of data, and `Y` as the content of data. The `int` of "`int N`" stands for integer, and `N` can only store an integer value. "`vector[N] Y`" means that `Y` is a vector that can only store `real` values,[16] indicating that there are `N` elements of `Y[1]`, `Y[2]`, ..., `Y[N]`. `real`, `int`, and `vector` are called *types*.

Variables can be declared anywhere immediately after "`{`" in each block. However, for beginners, it is easy to start by declaring the variables at the beginning of each block.

comment

In a Stan code, comments can be written with the "`//`" at the beginning of a comment part, and any texts after this will not be executed. If a comment spans multiple lines, enclose the comment with "`/*`" and "`*/`".

Technical Point: Comparisons between Stan and WinBUGS

Note that Stan and WinBUGS have different notations for a normal distribution: WinBUGS expresses it as `dnorm(mean, reciprocal of variance)`, while Stan expresses it as `normal(mean, SD)`. Other grammar and syntax differences between Stan and WinBUGS are explained in Appendix.

3.5.3 Coding Style Guide

Although personal preferences of coding styles differ largely, there are many benefits of following several common rules, and these rules make it easy for other people to read. Therefore, there is a rule called a coding style guide to unify styles as much as possible. We suggest several rules as follows:

[16] If we want to store multiple integers, we need to use array. See Sect. 5.3.

- Do not forget indentation
- For variable names in data block, use uppercase for the first letter; for variable names in other blocks, use lowercase for the first letter.
- Insert a blank line between each block.
- For the variable names, use "snake_case" (word break: underscore) instead of "camelCase" (word break: upper case).
- Put spaces before and after each "~" (probabilistic generation) and "=" (assignment)

The first two rules are particularly important that everyone should follow. If one failed to follow the rule 1, the Stan code in the previous subsection will look like:

```
data {
int N;
vector[N] Y;
}
parameters {
real mu;
}

model {
for (n in 1:N) {
 Y[n] ~ normal(mu, 1);
 }
 mu ~ normal(0, 100);
}
```

As we can see, it would be very difficult to understand the entire structure of the code, especially for the three blocks and the `for` loop statement. In addition to the readability, this will be likely to induce extra bugs, and debugging becomes even more difficult. Therefore, keep in mind to add indentation immediately after "`{`".[17] If you are using RStudio to write a Stan code, it will automatically detect the structures of blocks and other statements (such as `for` and `if` statement) and inserts indentation correspondingly, which makes this process much easier. The rule 2 makes it easier to identify the data and parameters in `model` block.

Appendix "Stan Program Style Guide" in the Stan reference includes some other rules. For example, the maximum length of a line is 80 characters, how to insert spaces, and so on. If you post a question to Stan's discouse website, those rules should be followed strictly.

[17] Discussion on the indentation width: the Stan development team recommends using two spaces instead of a hard tab (Tab, \t) for the indentation, but the comfortable indentation width varies from person to person. In order to avoid religious war that corrects only the indent width in version control, we agree with a hard tab person's opinion that "Indent with hard tabs. Using your editor, replace them with arbitrary multiple spaces and display them". In this book, indentation is displayed with two spaces to follow the recommendation of Stan development team.

3.6 lp__ and target in Stan

Let us explain `lp__` and `target` together with internal mechanism of Stan.

`lp__`

In order to efficiently search parameter space that has a high posterior probability $p(\theta|Y) \propto p(Y|\theta)p(\theta)$, Stan uses a gradient obtained by computing the partial derivatives of the log posterior probability:

$$\log p(\theta|Y) = \log[p(Y|\theta)p(\theta)] + \text{const.} = \log p(Y|\theta) + \log p(\theta) + \text{const.}$$

with respect to the parameter θ. For the partial differentiation, Stan internally has $\log p(Y|\theta^*) + \log p(\theta^*)$ at θ^* as `lp__` (short for log posterior) in each MCMC step.[18] As mentioned above, Stan samples more often at places with larger log posterior probability by using the partial derivative of `lp__`. Note that only this partial derivative determines the sampling behavior in Stan. Therefore, adding or subtracting a constant term to `lp__` does not affect the sampling behavior. From another viewpoint, we can say that `model` block determines the function form of `lp__` by describing likelihoods and prior distributions.[19]

`target`

In a Stan code, a likelihood $p(Y|\theta)$ is typically expressed as: "$Y \sim$ certain distribution (θ)". This notation has two interpretations:

- We consider that Y is generated from a certain distribution with a parameter θ
- Stan computes $\log p(Y|\theta)$ and adds this term to `lp__`

We usually consider and create models based on the first interpretation in real data analysis. However, it is important to keep the second interpretation in mind. In addition to the "~" notation, we can also use "`target`" (a nickname of `lp__`) notation to emphasize the second interpretation. Using this `target` notation enables us to directly manipulate the `lp__` object, and this is critical for model extension and defining a customized distribution using Stan.

Here we show these two interpretations using the example in Sect. 3.5.2. The posterior probability of Model Formula 3.1 is expressed by a mathematical formula as follows:

$$p(\theta|Y) \propto p(Y|\theta)p(\theta) = \left[\prod_{n=1}^{20} \mathcal{N}(Y[n]|\mu, 1)\right] \times \mathcal{N}(\mu|0, 100)$$

[18] Strictly speaking, it may differ by a constant.

[19] Since the log posterior probability can be freely defined, Stan can be used for any application that samples more at places with larger values of any function $f(\theta)$.

Taking the logarithm of the right-hand side provides the log posterior probability `lp__`:

$$\left[\sum_{n=1}^{20} \log \mathcal{N}(Y[n]|\mu, 1)\right] + \log \mathcal{N}(\mu|0, 100) \tag{3.3}$$

We omitted the constant term that will be 0 by partial differentiation. Let us clarify a relationship between (3.3) and `model` block in the Stan code:

```
10  model {
11    for (n in 1:N) {
12      Y[n] ~ normal(mu, 1);
13    }
14    mu ~ normal(0, 100);
15  }
```

The `for` loop in line 11–13 lines make Stan add the first term of (3.3) to `lp__` internally, and line 14 makes Stan add the second term of (3.3) to `lp__`. Using the `target` notation provides an alternative expression of the above model, and should yield the equivalent results:

```
10  model {
11    for (n in 1:N) {
12      target += normal_lpdf(Y[n] | mu, 1);
13    }
14    target += normal_lpdf(mu | 0, 100);
15  }
```

where `normal_lpdf(Y[n] | mu, 1)` is the convenient function that represents $\log \mathcal{N}(Y[n]|\mu, 1)$.[20] "`target += x`" is an abbreviation for "`target = target + x`", where "=" is assignment as in R and Python. Thus, using `target` notation clarifies the second interpretation, which is to add (3.3) to `lp__`.

Considering that model formulas using " ~ " are often seen in existing books and articles, we usually use the notation of " ~ " in Stan codes in this book. However, the `target` notation is required for more advanced models which will be discussed in Sect. 7.7 and Chap. 10.

It is beyond the scope of this book to describe the details of HMC, NUTS, and ADVI algorithms. Interested readers should read (Betancourt, 2017) for HMC, the original papers for NUTS (Hoffman and Gelman, 2014), and the original paper for ADVI (Kucukelbir et al., 2015).

[20] "`lpdf`" stands for log probability density function.

References

Betancourt, M. (2017). *A conceptual introduction to Hamiltonian Monte Carlo*. arXiv:1701.02434.

Hoffman, M. D., & Gelman, A. (2014). The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research, 15*, 1593–1623.

Kucukelbir, A., Ranganath, R., Gelman, A., & Blei, D. M. (2015). Automatic variational inference in stan. arXiv:1506.03431.

Stan Development Team. (2022). Stan: A C++ library for probability and sampling, Version 2.29. Retrieved from http://mc-stan.org.

Chapter 4
Simple Linear Regression

In this chapter, we will introduce how we typically use Stan with the example of univariate regressions. We will use R or Python to run Stan codes and estimate parameters. As we mentioned in the previous chapter, there are three ways of estimating parameters in Stan: Bayesian inference using NUTS (MCMC), Bayesian inference using ADVI, and point estimation using L-BFGS. Among these, NUTS (MCMC) will be the focus of this book. In Sect. 4.2, we will explain in detail how to do estimation, and how to use the draws generated from MCMC, such as computing Bayesian confidence intervals and Bayesian prediction intervals. In the second half of the chapter, we will introduce how to infer the parameters using ADVI and L-BFGS.

4.1 Statistical Modeling Workflow Before Parameter Inference

In this chapter, we use a dataset from 15 employees in company B (Data file 4.1, which is a comma-separated file, and a row corresponds to an employee). Specifically, we have the years of work experience (X, unit: years) of each employee in a particular business field and his/her annual income (Y, unit: \$1 k). Let us assume that the annual income can be represented as the combination of the baseline income y_{base} and the noise ε. Also in company B, the it is assumed that baseline income for an employee is proportional to the years of work experience. Although ε may include several factors, we assume that ε follows a normal distribution with the mean 0.

Data file 4.1 Structure of `data-salary.csv`

```
X,Y
7,45.7
10,48.2
16,51.8
...
21,51.7
```

© Springer Nature Singapore Pte Ltd. 2022

K. Matsuura, *Bayesian Statistical Modeling with Stan, R, and Python*,
https://doi.org/10.1007/978-981-19-4755-1_4

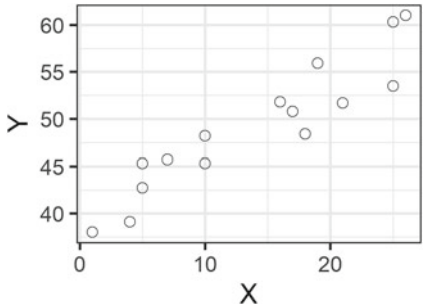


Fig. 4.1 Scatter plot with *X* values on the x-axis and *Y* values on the y-axis

4.1.1 Set Up Purposes

In this analysis, let us consider that we need to answer this question: how much will the annual income be for a person who started working in this company with 10 years of work experience?

This is a prediction problem, i.e., predict the annual income from the years of work experience, by estimating the linear relationship between the two. The input variables used for prediction (that is, the years of work experience in this example) are called the *explanatory variables*. The variables that are being predicted (that is, the annual income in this example) are called the *response variables*. When there is only one explanatory variable, this problem is called *simple linear regression*. The explanatory variable is also known as the *predictor variable*, *covariate*, *feature*, or *independent variable*. The response variable is also known as the *explained variable*, *outcome variable*, *dependent variable*, *objective variable*.

4.1.2 Check Data Distribution

Before conducting the analysis, we would want to check if it is reasonable to fit the data to a linear model. For this purpose, it is common to plot the scatter plot, with the explanatory variable on the x-axis and the response variable on the y-axis. It might help us to realize that it is hard to fit a linear model based on the scatter plot. If this is the case, we can conclude that some other factors rather than the baseline income have larger impacts on the response variable, and it is not practical to predict the annual income just from the years of work experience. It is thus important to visualize the data first, before any of the following steps. The scatter plot of this data is shown in Fig. 4.1. From this plot, we can conclude that the annual income indeed increases with longer work experience, almost in a linear manner.[1]

[1] In our book, we use `ggplot2` package in R. We can also use the standard plotting function in R or use the matplotlib library in Python.

4.1.3 Describe Model Formula

Let us write down the model equation for this example. Expressing what we described in the earlier section, we will get:

Model Formula 4.1

$$\begin{aligned} Y[n] &= y_{\text{base}}[n] + \varepsilon[n] && n = 1, \ldots, N \\ y_{\text{base}}[n] &= a + b\ X[n] && n = 1, \ldots, N \\ \varepsilon[n] &\sim \mathcal{N}(0,\ \sigma) && n = 1, \ldots, N \end{aligned} \tag{4.1}$$

where N represents the total number of the employees, $y_{\text{base}}[n]$ represents the baseline annual income, and $X[n]$ represents the years of work experience. The subscript n represents the index of each employee. a and b are the intercept and the slope of the linear model, respectively. $\varepsilon[n]$ represents the noise term for the other factors besides the baseline annual income. Eq. (4.1) indicates that $\varepsilon[1], \varepsilon[2], \ldots, \varepsilon[N]$ are independently generated from a normal distribution with the mean 0 and the standard distribution σ, where σ represents the scale of noise. If we do not consider the noise term, then the annual income will be determined purely from the linear relationship: $Y = a + bX$.

The parameters to be estimated in this example are a, b, and σ. In the Bayesian framework, not only the data but also the parameters follow certain probability distributions, as we explained in Chap. 2. Therefore, we need to set the prior distributions for these parameters. We will use noninformative priors (uniform distributions on a sufficiently wide range) by following Sect. 2.6. In this book, if a parameter follows a noninformative prior, we will not specifically write down its formula.

Next, we will look into two other expressions that have the equivalent meanings to Model Formula 4.1 because they might be used in other literatures. In addition, in Stan, it might improve the convergence of MCMC and speed up the computation when a model is written in a certain way than others. Therefore, we should understand the fact that these expressions can be used interchangeably.

First, by eliminating $y_{\text{base}}[n]$ in Model Formula 4.1, we can write it as follows:

Model Formula 4.2

$$\begin{aligned} Y[n] &= a + b\ X[n] + \varepsilon[n] && n = 1, \ldots, N \\ \varepsilon[n] &\sim \mathcal{N}(0,\ \sigma) && n = 1, \ldots, N \end{aligned}$$

Similarly, by eliminating $\varepsilon[n]$ in Model Formula 4.1, we can write it as follows:

Model Formula 4.3

$$y_{\text{base}}[n] = a + b\ X[n] \qquad n = 1, \ldots, N$$
$$Y[n] \sim \mathcal{N}(y_{\text{base}}[n],\ \sigma) \qquad n = 1, \ldots, N$$

Further, by eliminating both $y_{\text{base}}[n]$ and $\varepsilon[n]$:

Model Formula 4.4

$$Y[n]\ \sim\ \mathcal{N}(a + b\ X[n],\ \sigma) \qquad n = 1, \ldots, N$$

Model Formula 4.4 indicates that each employee's annual income is independently generated from a normal distribution, with the mean $a + bX[n]$ and the SD σ. a, b, and σ are the parameters to be estimated from the data.

In general, the computation in Stan is faster when we eliminate $\varepsilon[n]$. Therefore, we often use this eliminated expression in this book, although there are some exceptions, such as the hierarchical model mentioned in Chap. 8.

4.1.4 Maximum Likelihood Estimation Using R

Let us estimate the value of a and b by fitting the model to the data. Because we are just using a simple linear model for this example, we will see that there's no big difference in the predictive distributions computed by maximum likelihood estimate and the one computed by Bayesian inference with Stan, as also explained in Sect. 2.4. Therefore, for the explanatory purpose, let us use `lm` function in R and use the maximum likelihood estimate to fit the linear model. We show an R code here because it is more straightforward to use R to fit the linear model. But this can be done using Python library such as `statsmodels`. For a Python code, readers can refer the link in the footnote.[2]

```
> d <- read.csv(file='input/data-salary.csv')
> res_lm <- lm(Y ~ X, data=d)
```

The first line reads the data file, and the second line fits the data to the model and saves the result in the object named `res_lm`. We can obtain the intercept a and the slope b by specifying this object from R console and hitting the Enter key.

[2] https://www.statsmodels.org/stable/regression.html.

```
> res_lm

Call:
lm(formula = Y ~ X, data = d)

Coefficients:
(Intercept)        X
     38.697    0.752
```

From the last line of the output, we can see that the estimated intercept a is `38.697` and the estimated slope b is `0.752`. We can explain this result as: when the work experience increases one year, the annual income increases by about \$752. Using the result from the `lm` function, we can obtain the confidence intervals for the parameters, as well as the prediction intervals for the new data points. The R code for this process is as follows:

```
1 > X_pred <- data.frame(X=0:28)
2 > conf_95 <- predict(res_lm, X_pred, interval='confidence',
level=0.95)
3 > pred_95 <- predict(res_lm, X_pred, interval='prediction',
level=0.95)
```

Line 1: The years of work experience (X_{pred}) under which we want to predict the annual income is created as `data.frame`.

Line 2: The 95% confidence intervals of the baseline income ($a + bX_{\text{pred}}$) are computed for the employees with X_{pred} years of work experience.

Line 3: The 95% prediction intervals of the annual income including the noise term in addition to the baseline income ($a + bX_{\text{pred}} + \varepsilon$) are computed for the employees with X_{pred} years of work experience.

By specifying the value for the function argument `level`, we can also calculate other intervals (e.g., 50%) in a similar way. By plotting these intervals with the previous scatter plot, we can obtain Fig. 4.2. We see that the estimated slope is consistent with our knowledge, and the data points are located within the prediction intervals, which indicates that this prediction is quite reasonable.

4.1.5 Implement the Model with Stan

From now on, we will start to see Stan codes for model implementation and R codes to run these Stan code. All of these scripts are available from the GitHub repository of this book.[3]

Now we will be implementing the model for the example question above. First, we need to transform Model Formula 4.4 into a Stan code. Let us use a text editor to create the following Stan code (*model file*) and save it as a file named **`model4-4.stan`**.

[3] https://github.com/MatsuuraKentaro/Bayesian_Statistical_Modeling_with_Stan_R_and_Python.

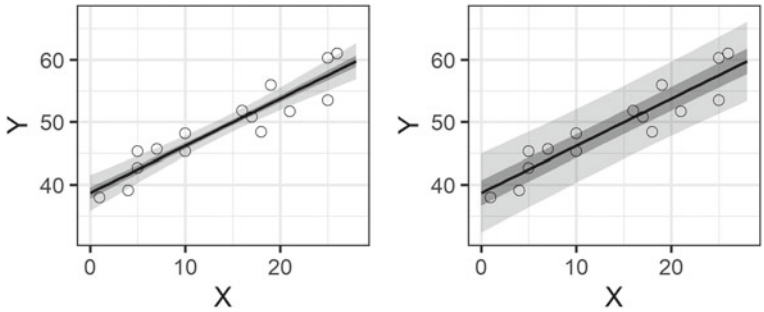


Fig. 4.2 Left: the confidence intervals of the baseline annual income under the work experience between 0 to 28 years. Right: the prediction intervals for the annual income under the same range. These results were obtained from `lm` function. The light gray bands are the 95% intervals, the dark gray bands are the 50% intervals, and the black lines are the MLEs

model4-4.stan

```
 1   data {
 2     int N;
 3     vector[N] X;
 4     vector[N] Y;
 5   }
 6
 7    parameters {
 8     real a;
 9     real b;
10     real<lower=0> sigma;
11   }
12
13   model {
14     for (n in 1:N) {
15       Y[n] ~ normal(a + b*X[n], sigma);
16     }
17   }
```

Let us look into `parameters` and `model` blocks:

parameters block

Line 10: The range of `sigma` is constrained by specifying `< lower = 0 >`, to avoid SD from being a negative value. In Stan, we can easily specify the upper and lower bound of the newly declared variables in this way.[4] This constrain can also be used in `data` block to check whether the input data is in the acceptable range when loading the data.

[4] Internally, when we use `real<lower=a,upper=b> x`, it is implemented by using a variable x^* that is defined as the real number and getting a transformed x by the change of variable formula: $x = a + (b - a) \times 1/(1 + \exp(-x^*))$, which suffices to achieve the desired constrain. In addition, the Jacobian that accompanies with the change of variable process is taken into consideration automatically.

model block

Line 15: This line indicates that `Y[n]` is generated from a normal distribution with the mean `a + b*X[n]` and the SD `sigma`. If we also want to specify the prior distributions for each parameter, we should include it in this block as well. Here, we omit the description of the prior distributions in this code. This is because in Stan, if we do not specify the prior distribution for a parameter, the prior distribution will automatically set to a uniform distribution on a sufficiently wide range (see Sect. 3.5.2). This has the advantage of making the script simpler and cleaner.

In Stan, as in R and numpy,[5] lines 14–16 can be vectorized as follows:

```
Y ~ normal(a + b*X, sigma);
```

To emphasize vectorization, it is desirable to add subscripts to `Y` and/or `X` as follows:

```
Y[1:N] ~ normal(a + b*X[1:N], sigma);
```

Because vectorization not only simplifies notation but also improves computation speed, hereafter, vectorization will be used unless readability is largely reduced.

4.2 Bayesian Inference Using NUTS (MCMC)

In this section, we will explain how to pass the data to the model defined by the Stan code in the previous section, and to implement the parameter estimation. In order to do this, we need to use R or Python to pass the data as a certain type defined in the Stan `data` block, and then write R or Python commands to run the parameter estimation process. We will introduce some of the R packages and Python libraries for using Stan, but in terms of their installation, we won't explain in detail here and the readers are encouraged to go to the website of individual packages/libraries and look into their references. In general, these can be installed by `install.packages` function in R, and using `pip` tool in Python.

4.2.1 Estimate Parameters from R or Python

R

We can implement the estimation process as follows using `cmdstanr` package:

`run-model4-4.R` (Lines 1–6)

```
1    library(cmdstanr)
2
3    d <- read.csv(file='input/data-salary.csv')
```

[5] In numpy, the mechanism is called broadcasting.

```
4    data <- list(N=nrow(d), X=d$X, Y=d$Y)
5    model <- cmdstan_model(stan_file='model/model4-4.stan')
6    fit <- model$sample(data=data, seed=123)
```

Let us take a closer look at each line:

Line 1: The installed package `cmdstanr` is loaded.

Line 3: The Data file 4.1 mentioned in the first section is read.

Line 4: Creating the data as a named list in order to pass it to `data` block. Here `N` represents the total number of data points, `X` represents the years of work experience, and `Y` represents the annual income.

Line 5: The model file that is saved in the previous section is compiled using `cmdstan_model` function. It returns an object of class `CmdStanModel`.

Line 6: The data is passed to the `sample` function, and the sample is drawn from the posterior distribution. We will explain the `seed` argument later. The object `fit` is an object of class `CmdStanMCMC`, and it saves the settings of MCMC, as well as the draws as the estimation result. For instance, `fit$time()` returns computation time. For the other functions of class `CmdStanMCMC`, the readers can type "`help('CmdStanMCMC')`" on the R console to get more information.

Python

We can implement the estimation process as follows using library `cmdstanpy`:

`run-model4-4.py` (Lines 1–8)

```
1    import pandas
2    import cmdstanpy
3
4    d = pandas.read_csv('input/data-salary.csv')
5    data = d.to_dict('list')
6    data.update({'N':len(d)})
7    model = cmdstanpy.CmdStanModel(stan_file='model/model4-
4.stan')
8    fit = model.sample(data=data, seed=123)
```

Line 2: Loading the installed `cmdstanpy` library.

Line 4: The Data file 4.1 is read using `read_csv` function in the `pandas` library.

Lines 5–6: Creating the data as a `dictionary` in order to pass it to `data` block. Here `N` represents the total number of data points.

Line 7: The model file that is saved in the previous section is compiled using `CmdStanModel` function. It returns an object of class `CmdStanModel`.

Line 8: The data is passed to the `sample` function, and the MCMC sample is drawn from the posterior distribution. We will explain the `seed` argument later. The object `fit` is of class `CmdStanMCMC`, and it saves settings of MCMC, as well as the draws as the estimation result. For details, the readers can refer to the `CmdStanPy` reference.

4.2.2 Summarize the Estimation Result

In this part, we will explain how to look at the estimation result. From R console, `fit$cmdstan_summary()` (from Python console, `fit.summary()`) will return something like this:

```
    ...
1           Mean    MCSE      StdDev   5%     50%    95%    N_Eff   N_Eff/s   R_hat
2   lp__    -22     4.2e-02   1.4      -24    -21    -20    1119    7773      1.0
    ...
3   a       39      4.2e-02   1.6      36     39     41     1398    9707      1.0
4   b       0.75    2.6e-03   0.097    0.59   0.75   0.91   1370    9517      1.0
5   sigma   2.9     1.6e-02   0.65     2.1    2.8    4.1    1721    11954     1.0
    ...
```

This gives the summary statistics of the marginalized posterior distribution (we will explain more in Sect. 4.2.6), as well as the summary of the convergence diagnostics of MCMC. For instance, line 3 summarizes the marginalized posterior distribution $p(a|X, Y)$ for parameter a.

Now let us look into each column.

The first column shows the name of the parameter. We introduced `lp__` in Sect. 3.6, and it is in the first row here. We need to ensure that not only the parameters but also `lp__` converges.

`Mean` in the second column is the mean of the marginalized posterior distribution, and it is also called the *posterior mean*. This is computed from the arithmetic mean of all the draws (here, `4000` in total). For instance, the posterior mean of `b` is `0.75`, and this result indicates that with one additional year of work experience, the baseline annual income increases `0.75` (unit: $1 k) on average. This mean can be used as the representative value for the draws.

`MSCE` in the third column is the Monte-Carlo standard errors of the `Mean`. It is computed by dividing the `StdDev` with the square root of `N_Eff` (we will explain `StdDev` and `N_Eff` later).

`StdDev` in the fourth column is the SD of the marginalized posterior distribution. It is computed from the SD of the draws.

From columns 5 to 7, the quantiles of the marginalized posterior distributions are shown. These are computed based on the quantiles of the draws. They correspond to the Bayesian confidence intervals that we introduced in Sect. 2.4. Amongst the quantiles, the median (50%) can be used as the representative value of the draws, as an alternative of the posterior mean. Also, because there's a consistency between the $(1 - \alpha)\%$ intervals and quantiles, the median values are often used when we are plotting the intervals.

`N_eff` in column 8 is the effective number of the draws, and Stan calculates this value based on the autocorrelation of the draws. We consider this value should be at least 100 in order to estimate the distributions and compute other statistics. Also,

when this value is small, it indicates that this parameter has likely not converged yet, and this could be a guidance to the model improvement.

`Rhat` ($\widehat{R}$) in column 10 is an indicator of MCMC convergence, and it is computed for each parameter. Usually, it uses the results from multiple chains and compares the sample variance of each chain. In case when only one chain is used in Stan, it splits the chain into multiple parts and considers each part as one chain, in order to compute `Rhat`. In this book, we followed Sect. 11.4 in (Gelman, 2013), and consider that the MCMC was converged if it satisfies `Rhat < 1.1` for all the parameters. We will not have in-depth discussion about $\widehat{R}$, but readers can refer the book for more details.

Before interpreting results, we must make sure that the MCMC is converged. It is very common for the beginners to ignore this step, and move to the next step of the analysis, such as using the posterior mean, plotting the histogram of draws, and rush into interpreting the results, using the MCMC that has not converged at all. This should definitely be avoided. We need to go through the process of try-and-error multiple times until the MCMC converges.

4.2.3 Save the Estimation Result

Here we describe how to save the Stan's estimation results into a particular file, how to save the summary information from the previous subsection, and how to save the trace plot that we introduced in Sect. 2.3.

R

`run-model4-4.R` (Lines 8–15).

```
 8   fit$save_object(file = 'output/result-model4-4.RDS')
 9   write.table(fit$summary(), file='output/fit-summary.csv',
10           sep=',', quote=TRUE, row.names=FALSE)
11
12   library(coda)
13   pdf(file='output/fit-plot.pdf')
14   plot(as_mcmc.list(fit))
15   dev.off()
```

Line 8: `fit` is saved into a file using `save_object` function. This process makes the use of `fit` in later processes easier.

Lines 9–10: The summary information is saved as a csv file.

Lines 12–15: In order to save the output of the trace plot that we introduced in Sect. 2.3, we use `coda` package because of its fast drawing speed. In line 14, the `fit` object is transformed into an `mcmc.list` object using the `as_mcmc.list` function in `cmdstanr` package. The output plot is shown in Fig. 4.3 (it will be shown in color if we see the real output). The left column shows the trace plots after warmup, and the right column shows the probability density functions of the marginalized

posterior distributions that are computed from all the draws after warmup. From these trace plots, we can see that after warmup, all the chains are going up and down around certain values, which suggests that the MCMC has converged. In addition, we can conclude that the length of warmup (which is 1000 iterations here) was sufficient.

Python

`run-model4-4.py` (Lines 10–16)

```
10    fit.save_csvfiles('output/result-model4-4')
11    fit.summary().to_csv('output/fit-summary.csv')
12
13    import arviz
14    axes = arviz.plot_trace(fit)
15    fig = axes.ravel()[0].figure
16    fig.savefig('output/fit-plot.pdf')
```

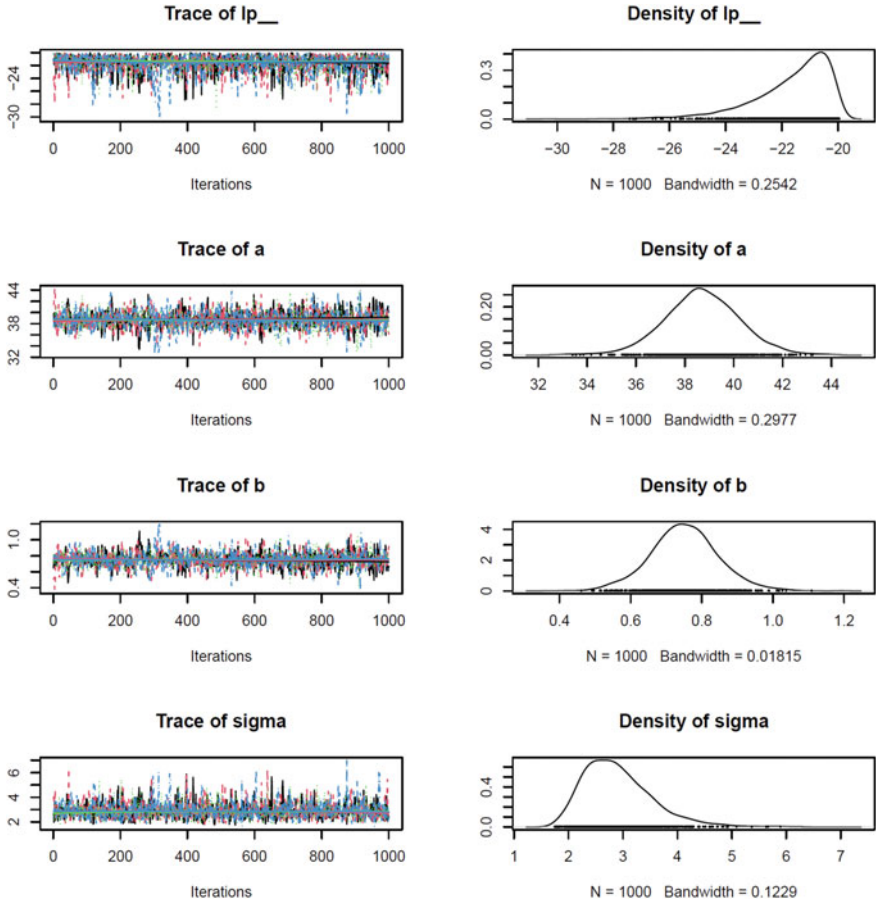


Fig. 4.3 Trace plots and marginalized posterior distributions for each parameter

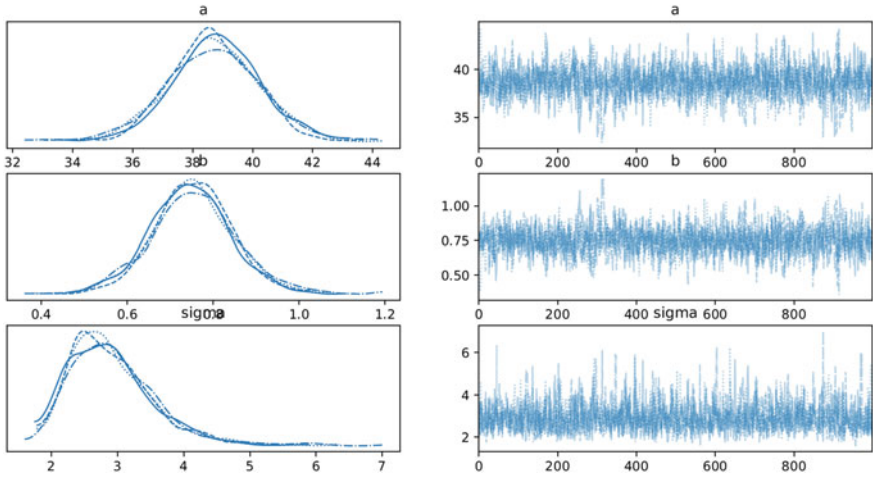


Fig. 4.4 Trace plots and marginalized posterior distributions for each parameter

Line 10: `fit` is saved into a folder using `save_csvfiles` function. This process makes the use of the sampling result in later processes easier by using `from_csv` function.

Line 11: The summary information introduced in the previous subsection is exported into a text file.

Lines 13–16: It is convenient to use the library `arviz`[6] if we want to use Python to export the trace plot into a file. The exported plot is shown in Fig. 4.4 (it will be shown in color if we see the real plot). The right column shows the trace plots after warmup, and the left column side shows the probability density functions of the marginalized posterior distributions which are computed from the draws after warmup in each chain. From these plots, we can see that after the warm-up, all the chains are going up and down around certain values, which suggests that the MCMC has converged. In addition, we can conclude that the length of warmup (which is 1000 iterations here) was sufficient.

4.2.4 Adjust the Settings of MCMC

With the default settings for the `sample` function, the number of chains (`chains`) is `4`, the number of sampling (post-warmup) iterations per chain (`iter_sampling`) is `1000`, the number of warmup iterations per chain (`iter_warmup`) is `1000`. Warmup is used to remove the dependencies on the initial values (see Sect. 2.3). The thinning interval (`thin`) is `1`. The total number of obtained draws are `chains * iter_sampling/thin`, thus here it is `4*1000/1`, which gives `4000`.

[6] https://github.com/arviz-devs/arviz.

It is common that we want to keep the model file but just want to do resampling with different settings of MCMC (such as different values for `iter_sampling`, `iter_warmup` and `thin`), with new initial values for some parameters, or even using new data. These can be achieved easily by changing the arguments in the `sample` function.

Using R:

```
1    init_fun <- function(chain_id) {
2     set.seed(chain_id)
3     list(a=runif(1,30,50), b=1, sigma=5)
4    }
5
6    fit <- model$sample(
7     data=data, seed=123,
8     init=init_fun,
9     chains=3, iter_warmup=500, iter_sampling=500, thin=2,
10    parallel_chains=3,
11    save_warmup=TRUE
12   )
```

Using Python:

```
3    def init_fun():
4        return dict(a=40, b=1, sigma=5)
5
6    fit = model.sample(
7        data=data, seed=123,
8        inits=init_fun(),
9        chains=3, iter_warmup=500, iter_sampling=500, thin=2,
10       parallel_chains=3,
11       save_warmup=True
12   )
```

Let us look into more detail on these lines one by one.

Setting a random seed for Stan

In line 7, we are passing a random seed for Stan. Using the same seed would generate the same MCMC sample. If not specified, a random number is automatically generated as the seed. For reproducibility purpose, it is a good idea to specify a seed every time we run the sampling.

Setting initial values

In line 8, we are setting the initial values for the parameters. The function used here is defined in lines 1–4 in R (lines 3–4 in Python). By passing this function to the `init` argument in R (`inits` argument in Python), we are using different initial values for each chain. We can also choose to set the initial values for only some of the parameters. If initial values are not specified for some parameters, the initial values of these parameters will be generated from a uniform distribution defined

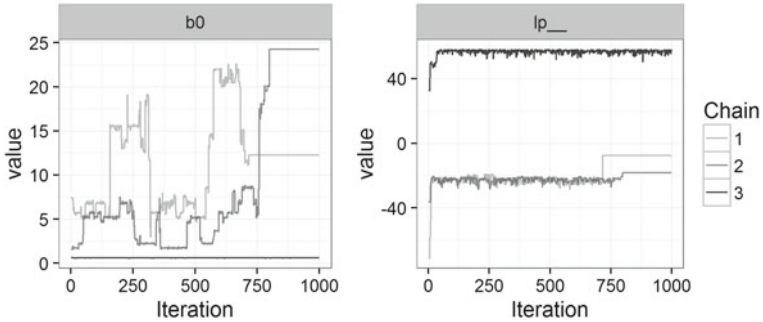


Fig. 4.5 Example trace plots showing the case where several chains are trapped in the local optimum and cannot get out

in the range of $[-2, 2]$.[7] Of note, when the trace plot looks like the one shown in Fig. 4.5, it is possible that these chains with low `lp__` values are trapped in the local optimum, and they are unlikely to get out of these values within a reasonable time. In such cases, setting the initial values might be helpful.

How to decide the values for chains, `iter_sampling`, `iter_warmup`, and, `thin`.

In line 9, we specify the number of total iterations in a chain and other related values. There's no single standard way to decide these values, but we will introduce our recommendation based on our practical experience.

- `chains`: Having at least three chains would be ideal. Stan development team recommends using four chains and we usually use four chains for own analysis as well.
- `iter_sampling`: When we try a variety of different models, we usually use 300–1000 iterations. But it should be increased once the model is finalized. In general, we implement the model with 2000–10,000 at the final stage, although this also depends on the value of `thin`. As a reference, if we use MCMC and want to decrease the standard error of posterior mean by one order, based on the central limit theory we know that we need 100 times more draws. Therefore, we need to increase this `iter_sampling` value when necessary.
- `iter_warmup`: We decide this value based on the trace plot. It varies largely based on models, but having 200–1000 is usually sufficient.
- `thin`: We usually choose 1 for this argument in Stan, because one advantage of Stan is that there is lower autocorrelation in draws, compared to the other MCMC software such as WinBUGS and JAGS. As shown in Fig. 4.6 (left), when some parameter values occasionally have abrupt changes, increasing the value of thin to a larger value (e.g., 5 or 10) can mitigate such changes and reduce autocorrelation,

[7] If the parameter is constrained, such as `real<lower=a,upper=b> x`, the random variable x^* is generated from a uniform distribution in $[-2, 2]$, and then x is generated by the transformation of variables $x = a + (b - a) \times 1/(1 + \exp(-x^*))$, to obtain its initial values.

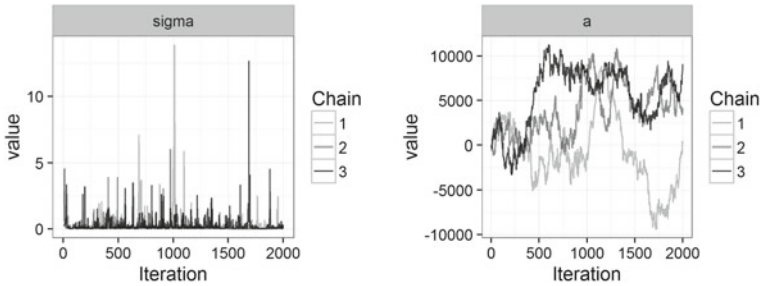


Fig. 4.6 Left: example trace plot where some parameter values occasionally have abrupt changes. Right: example trace plot from where it suggests the model needs to be modified

leading to a better convergence. However, when the trace plot looks like Fig. 4.6 (right), it suggests that the autocorrelation is too high, and the model needs to be modified further.

Running chains in parallel

In line 10, we parallelize Bayesian inference with MCMC. The idea is that because every chain is independent of each other in the MCMC algorithm, we can compute them separately and then put them together at the end of the algorithm. When the sampling takes a long time, by parallelizing the computation of each chain, rather than computing one by one sequentially, we can largely shorten the time usage.

Saving draws of warmup iterations

In line 11, we are setting `save_warmup = True` to save the draws of warmup iterations.

For the other options for MCMC settings, readers can refer the help page of the `sample` function.

4.2.5 Draw the MCMC Sample

Here we introduce how to extract the MCMC sample from the estimation result.

R

The MCMC sample from the posterior distribution is saved in the `fit` object (an object of class `CmdStanMCMC`), and it can be easily drawn by using `draws function:`

```
> d_ms <- fit$draws(format='df')
```

The returned result, d_ms[8] is a data frame with the size of (number of draws) × (number of the variables). By default, draws from the warmup period are not included. As an example, let us check the draws of the slope *b* and compute its 95% Bayesian confidence interval. We can use quantile function to compute the 95% interval because d_ms$b is a numeric vector.

```
> d_ms$b
   [1] 0.644549 0.812056 0.782344 0.676929 …
```

```
> quantile(d_ms$b, probs=c(0.025, 0.975))
  2.5% 97.5%
0.5540 0.9525
```

If we want to obtain the MCMC sample in the form of matrix in R, the draws function can be used with the argument format = 'matrix':

```
> ms <- fit$draws(format='matrix')
```

If the readers are familiar with tidy data (briefly, long data format)[9] and the package dplyr, tidybayes[10] package may be helpful.

Python

The MCMC sample from the posterior distributions is saved in the fit (an object of class CmdStanMCMC), and it can be easily drawn using draws_pd function:

```
> d_ms = fit.draws_pd()
```

The returned result, d_ms[11] is a pandas data frame with the size of (number of draws) × (number of the variables). By default, draws from the warmup period are not included. As an example, let us look at how to extract the draws of the slope parameter *b* and compute its 95% Bayesian confidence interval. We can use percentile function in numpy to compute the 95% intervals because d_ms.b.values is a one-dimensional numpy.ndarray.

```
> d_ms.b.values
array([0.644549, 0.812056, 0.782344, ..., 0.74119 , 0.699928,
0.598053])
> import numpy as np
> np.percentile(d_ms.b, q=[2.5, 97.5])
array([0.55404007, 0.95251197])
```

[8] ms stands for MCMC sample.

[9] http://vita.had.co.nz/papers/tidy-data.html.

[10] http://mjskay.github.io/tidybayes/index.html.

[11] ms stands for MCMC sample.

If we want to obtain the MCMC sample in the form of multidimensional `numpy.ndarray`, the `draws` function can be used:

```
> ms <- fit.draws()
```

4.2.6 Joint Posterior Distributions and Marginalized Posterior Distributions

In this subsection, we will visualize the joint distribution and the marginalized distribution using the generated MCMC sample. This process will be helpful for us to gain a better understanding on the posterior inference results.

The generated MCMC sample is the draws from the posterior distribution $p(a, b, \sigma|X, Y)$. In other words, it is the draws from the joint distribution of a, b and σ. To make it even clearer, let us look into `d_ms` (here, we focus on `a`, `b`, and `sigma` columns) in the previous subsection. The first five rows of `d_ms` will look like:

```
        a       b   sigma
1   39.77  0.6445   2.666
2   38.35  0.8121   1.972
3   37.74  0.7823   2.191
4   40.01  0.6769   3.654
5   38.64  0.7419   2.238
```

Each row of `d_ms` (e.g., the contents of the second row is `(38.35, 0.8121, 1.972)`) represents a draw from the joint distribution $p(a, b, \sigma|X, Y)$. On the other hand, each column of `d_ms` (e.g., column of b) corresponds to the 4000 draws from the marginal distribution (e.g., $p(b|X, Y)$).

In order to gain a better intuition of the joint distribution and the marginal distribution, we can plot `d_ms` into a three-dimensional scatter plot, with each row corresponding to one dot, as shown in Fig. 4.7 (left). The distribution of these dots on the three-dimensional space corresponds to the joint distribution $p(a, b, \sigma|X, Y)$. For each of these dots, we can ignore the value of σ, and visualize only a and b on the two-dimensional space. This is equivalent to projecting each dot vertically on the bottom surface, which yields the white dots on the same plot. The distribution of these dots on the two-dimensional space corresponds to the marginal distribution $p(a, b|X, Y)$. By separating the scatter plot on this two-dimensional space, we can obtain Fig. 4.7 (right). Further, by ignoring the value of b, and only looking at the value of a on this plot, we can obtain the histogram that is shown on the top side of the same plot, and this corresponds to the marginal distribution $p(a|X, Y)$. Similarly, by ignoring the value of a, and only looking at the value of b on this plot, we can obtain the histogram that is shown on the right side of the same plot, and this corresponds to the marginal distribution $p(b|X, Y)$. The posterior mean and the quantile for each

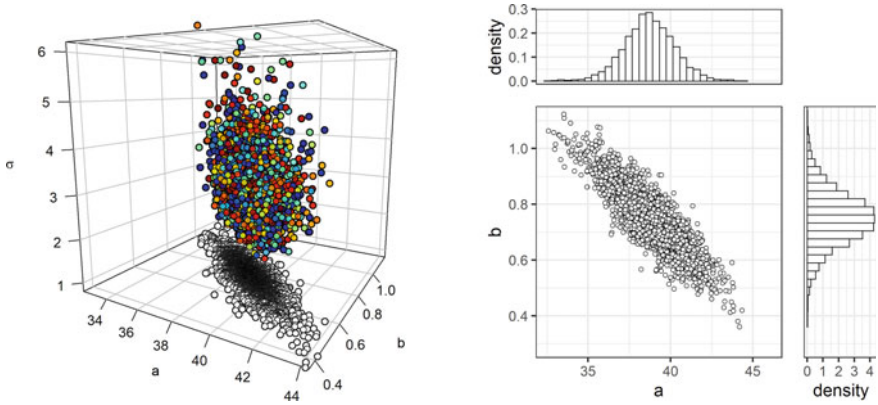


Fig. 4.7 Scatter plot of draws and their marginal distributions. Left: three-dimensional scatter plot with the values of a, b and σ on the three axes. Right: two-dimensional scatter plot, with a on the x-axis and b on the y-axis

parameter in Sect. 4.2.2 is the summary of these marginal distributions which focus only on one parameter, i.e., $p(a|X, Y)$, $p(b|X, Y)$, and $p(\sigma|X, Y)$.

From the two-dimensional scatter plot, we can learn that there is a strong negative correlation between the intercept a and slope b. This negative correlation is a natural consequence, because when we fit a linear model, for a straight line to pass near the data points, the slope needs to be smaller when the intercept is larger and vice versa.

4.2.7 Bayesian Confidence Intervals and Bayesian Prediction Intervals

Let us go back to our original aim of the analysis. Here, we will compute the posterior distribution and the Bayesian confidence intervals of the baseline annual income, as well as the predictive distribution and the Bayesian prediction intervals of the annual income, to answer the question: "What would be the annual income of an employee with 10 years of work experience?". For this purpose, let us first generate the draws from these distributions.

Using R:

```
> N_ms <- nrow(d_ms)
> y10_base = d_ms$a + d_ms$b * 10
> y10_pred <- rnorm(n=N_ms, mean=y10_base, sd=d_ms$sigma)
```

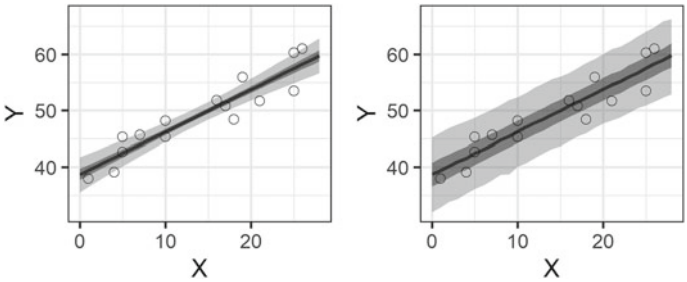


Fig. 4.8 Bayesian confidence intervals for baseline annual income from 0 to 28 years of work experience, and the Bayesian prediction intervals for the annual income, both computed using Stan. The light gray bands are the 95% intervals of the draws, the dark gray bands are the 50% intervals of the draws, and the black lines are the medians of the draws

Using Python:

```
1    > N_ms = len(d_ms.lp__)
2    > y10_base = d_ms.a + d_ms.b * 10
3   > y10_pred = np.random.normal(loc=y10_base, scale=d_ms.sigma,
size=N_ms)
```

In both cases,

Line 1: The MCMC sample size (the number of draws) is saved.

Line 2: The draws are generated from the distribution of baseline annual income for employees with 10 years of work experience.

Line 3: The draws are generated from the predictive distribution of annual income for employees with 10 years of work experience (see Sect. 2.4).

In order to compute the Bayesian confidence intervals and Bayesian prediction intervals, we can use `quantile` function in R or `numpy.percentile` function in Python, with these draws as the input.

In order to visually check the results, we can conduct the same process as the case where the years of work experience of employees ranges from 0 to 28. By plotting the scatter plots and the computed intervals, we can obtain Fig. 4.8. We can see that this is almost consistent with the results obtained from using `lm` function in R (Fig. 4.2). Note that lines indicating the intervals are slightly wiggly, because the draws are generated with randomness.

4.3 `transformed parameters` Block and `generated quantities` Block

In Sect. 4.2.7, we used R and Python to compute various quantities based on the draws generated in Stan. However, there are some limitations of conducting analysis in this way. For instance, when we do the computation from R or Python side, the code could become more complicated and tend to induce bugs in our code. In addition,

the computing speed will be slow too. Therefore, ideally, we want to compute the various quantities in Stan. `transformed parameters` block and `generated quantities` block can be used for this purpose. By using these two blocks, we can rewrite **`model4-4.stan`** as follows:

`model4-4b.stan`

```
 1   data {
 2     int N;
 3     vector[N] X;
 4     vector[N] Y;
 5     int Np;
 6     vector[Np] Xp;
 7   }
 8
 9   parameters {
10     real a;
11     real b;
12     real<lower=0> sigma;
13   }
14
15   transformed parameters {
16     vector[N] y_base = a+b*X[1:N];
17   }
18
19   model {
20     Y[1:N] ~ normal(y_base[1:N], sigma);
21   }
22
23   generated quantities {
24     vector[Np] yp_base=a+b*Xp[1:N];
25     array[Np] real yp = normal_rng(yp_base[1:N], sigma);
26   }
```

The main differences from the previous one are as listed:

Use of transformed parameters block (lines 15–17)

When we want to define variables that follow any of the following conditions, this block can be useful:

- We want to obtain the draws from this variable using `sample` function.
- The variable is not only used in `model` block but also needs to be used in `generated quantities` block.

In Stan, the variables can only be declared after "`{`" (see Sect. 3.5.2), and the declaration and definition can be done in one line like line 16. Note that constraints for variables (i.e., `<lower=0>`) should not be used in `transformed parameters` and `generated quantities` blocks. The "=" means that the value on the right-hand side will be directly assigned to the left-hand side. We can use several types of variables in the right-hand side. For instance, we can use the constants and parameters declared in `data` and `parameters` blocks, or we can also use other functions such as arithmetic operations and log transformation. Note that line 16 is vectorized

(see Sect. 4.1.5), and the line represents `y_base[1] = a + b*X[1]`, `y_base[2] = a + b*X[2]`, ..., `y_base[N] = a + b*X[N]`.

Here the draws of `y_base` will be saved because we have defined `y_base` in `transformed parameters` block. When we do not need to save the draws but only want to use these parameters in `model` block, we can do the following to define the parameters inside of `model` block, instead of using `transformed parameters` block:

```
model {
 vector[N] y_base = a + b*X[1:N];
 Y[1:N] ~ normal(y_base[1:N], sigma);
}
```

Some of the readers might think that, in order to reflect the transformation of random variables in the posterior probability, we need to adjust the absolute values of the Jacobian matrix. But we do not need to worry about it because all of these adjustments are made automatically in Stan.

Use of generated quantities block (lines 23–26)

Using this block is helpful when we want to define a temporary variable that follows this condition:

- We do not need to use a variable in `model` block and want to get the draws of the variable after estimation.

The computation inside of this block is very fast, because it is completely separated from the posterior distribution estimation. In this block, we can use any of the variables and functions that are defined before `model` block. We can also generate random numbers that follow a certain distribution by using a function named as `XX_rng()`, where the `XX` part is the name of this distribution.

In model4-4b.stan, we make predictions with `Xp`,[12] which represents the years of work experience from 0 to 28. `Xp` is declared in line 6. From line 24 to 25, we declare and define the baseline annual income `yp_base` and the annual income `yp` including noise terms, which correspond to the new data points `Xp`. Note that line 25 is vectorized, and the line represents `yp[1] = normal_rng(yp_base[1], sigma)`, ..., `yp[2] = normal_rng(yp_base[2], sigma)`. Note that the `normal_rng` function cannot return a `vector` but `reals`, which means an array of `real`, the declaration must be `array[Np] real`. The main differences between `array` and `vector` are that `array` can store integers, but vectorized operations, notations, and functions for vectors cannot be used.

[12] `p` stands for prediction.

4.4 Other Inference Methods Besides NUTS

Besides NUTS, there are several other attractive inference methods in Stan, and we will introduce these methods in this section. Some of them are used in the latter part of this book as well.

4.4.1 Bayesian Inference with ADVI

Here, we will implement Bayesian inference using ADVI which was mentioned in Sect. 3.2. If we are using R or Python, everything before the sampling process is the same as NUTS. After compiling a model, we will use `variational` function instead of `sample` function as below.

Using R:

```
fit <- model$variational(data=data, seed=123)
```

Using Python:

```
fit = model.variational(data=data, seed=123)
```

In `variational` function, we can specify `data`, `seed`, and `init` in R (`inits` in Python) as the function arguments. The returned object is of class `CmdStanVB`. Of note, however, there's no concept called "chain" in ADVI, and there are no arguments related to this concept. In contrast, ADVI has an argument that specifies the parameters for the ADVI algorithm. More information can be found in the help page of the `variational function.`

Of note, for the data and models mentioned in this chapter, the approximation obtained from ADVI Bayesian inference was not accurate enough to be useful.

4.4.2 MAP Estimation with L-BFGS

We will implement MAP estimation using L-BFGS which we mentioned in Sect. 3.2. As we did for ADVI, we only need to replace `sample` function with `optimize function.`

Using R:

```
fit <- model$optimize(data=data, seed=123)
```

Using Python:

```
fit = model.optimize(data=data, seed=123)
```

In `optimize` function, we can specify `data`, `init` and `seed` as the function arguments. The returned object is of class `CmdStanMLE`. It is possible that the estimated parameters take the local optimum values (see Sect. 2.2), and therefore it is ideal to estimate multiple times using different initial values and random seed settings. Usually, we adopt the estimation result after comparing the posterior probabilities obtained from these multiple trials.

Lastly, because in our model, we specified the prior distributions for both parameters to be the noninformative priors, and therefore the parameter values obtained from the MAP estimation and the Maximum likelihood estimation are theoretically the same (see Sect. 2.5). In fact, the result obtained from the `optimize` function and that of `lm` function from R in Sect. 4.1.4 are identical.

Technical Point: How to implement parallel computing within a chain?

In addition to assigning each chain to a single computing core parallelly, there is a new way of splitting data as to conduct parallel computing even within one chain. And this alternative approach has been starting to be implemented in Stan. We will not get into the detail here, but the chapter on "Multithreading and Map-Reduce" in Stan reference will give more information for those readers who are interested.

4.5 Supplementary Information and Exercises

For Stan, the best source to get more information is always the official Stan reference[13] and Stan discourse.[14] Whenever there are questions, we will most likely get useful information by searching in these resources.

Stan development team has been creating other packages such as `posterior`, `bayesplot`, and `shinystan`. Both of them are developed with the aim to visualize the Bayesian inference results easier. We will not give much information here, but the readers can refer to their website[15] for more detail.

[13] http://mc-stan.org/documentation/.

[14] https://discourse.mc-stan.org.

[15] https://github.com/stan-dev/posterior, https://github.com/stan-dev/bayesplot, https://github.com/stan-dev/shinystan.

4.5.1 Exercises

We will do something equivalent to the t-test using Stan. Using the data generated from the following R code, we want to decide whether the mean parameter μ_1 in group one (`Y1`) differs from μ_2 in group two (`Y2`) by computing Prob[$\mu_1 < \mu_2$], which is the probability of $\mu_1 < \mu_2$.

```
set.seed(123)
N1 <- 30
N2 <- 20
Y1 <- rnorm(n=N1, mean=0, sd=5)
Y2 <- rnorm(n=N2, mean=1, sd=4)
```

(1) Visualize the data from these two groups so that we can intuitively see whether the difference exists between them.
(2) Write a model formula with the assumption that these two groups have the same SD. This corresponds to the Students' t-test.
(3) Create the model file of (2) in Stan and estimate the parameters. Don't use `generated quantities` block here yet, because in the next (4) we will be practicing how to make use of draws from R or Python.
(4) Compute Prob[$\mu_1 < \mu_2$] from the obtained draws using R or Python (hint: we can count how many times the event $\mu_1 < \mu_2$ occurrs in the entire draws, and divide this quantity by the total number of draws).
(5) Write a model formula with the assumption that the two SDs are different. This is equivalent to the Welch's t-test. Similarly, compute Prob[$\mu_1 < \mu_2$].

Reference

Gelman, A. et al. (2013). *Bayesian data analysis* (3rd ed.). Chapman and Hall/CRC.

Chapter 5
Basic Regressions and Model Checking

In this chapter, we introduce regression models that are widely used, including multivariate regression, binomial logistic regression, logistic regression, and Poisson regression. Further, we discuss how to use visualization to check whether a model is proper. Throughout this chapter, our analysis steps follow the workflow we introduced in Sect. 1.4.

5.1 Multiple Linear Regression

In the previous chapter, we used a straight line which fits the income data, and predicted the response variable (income) based on the single explanatory variable (years of work experience). In general, it is more common that we have a larger number of explanatory variables. Such regression with two or more explanatory variables is called *multiple linear regression.*

In this chapter, we will use practice data from an online store that recorded the purchase history of 50 customers (Data file 5.1, a row corresponds to a customer). Each column represents:

PersonID : Customer ID

Sex : A binary value that represents the gender of the customer (0: female, 1: male)

Income : Annual income (Unit: $1000)

Y : The purchase proportion of the year

Data file 5.1 Structure of `data-shopping-1.csv`

```
PersonID,Sex,Income,Y
1,0,85.1,0.56
2,1,64.6,0.59
3,0,97.6,0.56
...
50,1,70.0,0.64
```

© Springer Nature Singapore Pte Ltd. 2022

K. Matsuura, *Bayesian Statistical Modeling with Stan, R, and Python*,
https://doi.org/10.1007/978-981-19-4755-1_5

Here, the purchase proportion is calculated as the ratio of the number of times a customer purchased in the store to the customer's total number of visits this year. Although the total number of visits differs for each customer, we assume that they are in the range between 100 and 200. In this analysis, we consider Y as a response variable. From our experience, we consider that Y can be explained well by Sex and $Income$, and use these two factors as the explanatory variables of multiple linear regression.

5.1.1 Set Up Purposes

In this analysis, we want to know how well we can predict the response variable Y from the two explanatory variables Sex and $Income$. Also, we want to know the contribution of each explanatory variable on the prediction of Y. Assume that you are in the situation to present the results from this analysis in an upcoming meeting, and consider how to visualize the results in order to make the points clear to the audiences throughout the analysis.

5.1.2 Check Data Distribution

The first thing to do when we are given a dataset is to check the data distribution. The easiest place to start with is to visualize the data using *scatterplot matrix* (Fig. 5.1). In R, we can obtain scatter plots using `scatterplotMatrix` function from `car` package, or `pairs` function from the pre-installed `graphics` package. Besides, `ggpairs` function in `GGally` package is also recommended. It provides more flexible customization of the plots. In Python, we can use `PairGrid` function in `Seaborn` library.

From Fig. 5.1, we can infer the following results (throughout this book, we explain the plots on the diagonal first, followed by plots on lower and upper triangle):

- From the histogram in the 2nd row 2nd column, $Income$ seems to follow a bell-shaped distribution.
- From the box plot in the 3rd row 1st column, male ($Sex = 1$) customers have higher purchase proportions than female ($Sex = 0$) customers.
- From the scatterplot in the 3rd row 2nd column, customers who have higher $Income$ are likely to have higher purchase proportions.
- From the correlation coefficients in the 1st row 2nd column, Sex and $Income$ are not correlated.

When the number of explanatory variables is too large to visualize in a single graph, we can print out each plot on diagonal, or on lower triangle, into separate files. For the upper triangle elements, we can export the pairwise correlation coefficients to another Excel file for further assessments.

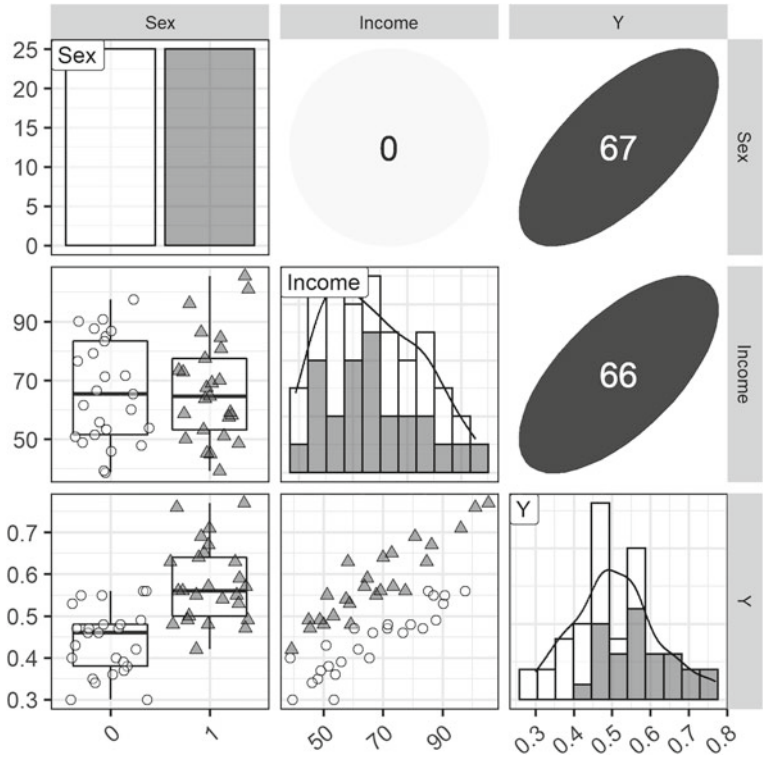


Fig. 5.1 Scatterplot matrix of the data. Here the diagonal plots show the histograms for each column (accumulated bar graph) overlaid with corresponding probability density functions calculated in R. The lower triangle plots show the scatter plots or box plots, and the upper triangle plots show 100× Spearman's rank correlation between two variables. The colors and the shapes on the diagonal plots and lower triangle plots represent the genders of the customers (○: female, ▲: male). For the plots on the upper triangles, pairwise correlations are represented by color and shape of the ellipse: a lower correlation is shown by whiter color and more circular shape; correspondingly, a stronger correlation (absolute value is closer to 100) is shown by darker color and more elliptical shape

5.1.3 Imagine Data Generating Mechanisms

When we look at the $Income$ versus Y scatter plot at the 3rd row 2nd column of Fig. 5.1, and only focus on the customers with $Sex = 0$, we find that there is a trend that when $Income$ is large, Y is also large with a linear relationship. This holds for the customers with $Sex = 1$. Also, from the Sex versus Y box plot at the 3rd row 1st column, we can see that Y is larger for $Sex = 1$ than for $Sex = 0$. Now based on these observations, let us assume that Y is determined by the linear combination of Sex and $Income$. Using mathematical expression, this can be written as: $Y = b_1 + b_2 Sex + b_3 Income$. Since this expression does not contain any probabilistic statement, Y can be uniquely determined and perfectly predicted

from these two variables. Nevertheless, it is reasonable to consider that Y is also affected by other factors (which altogether can be represented as a "noise term" that includes probabilistic fluctuation) in addition to these two explanatory variables. We consider that this "noise term" is different for each customer. We denote this noise term as ε, and assume that ε follows a normal distribution with mean 0.

5.1.4 Describe Model Formula

From the above discussions, we can consider the following model:

Model Formula 5.1

$$\begin{aligned} Y[n] &= b_1 + b_2 Sex[n] + b_3 Income[n] + \varepsilon[n] && n = 1, \ldots, N \\ \varepsilon[n] &\sim \mathcal{N}(0, \sigma) && n = 1, \ldots, N \end{aligned}$$

where N is the total number of the customers and n is the index of each customer. We can interpret that b_1 is the intercept, and the purchase proportion of a customer with $Sex = 1$ is higher by b_2 on average than that of a customer that has the same $Income$ but with $Sex = 0$. Similarly, we can interpret that the purchase proportion of a customer with $Income = any\ value + 1$ is higher by b_3 on average than that of a customer with $Income = any\ value$ and the same Sex. This simplicity in interpretations is the major advantage of multiple linear regression. b_2 and b_3 are called *regression coefficients.* Sometimes regression coefficients include b_1 and a regression coefficient is just called a coefficient.

As we explained in Sect. 4.1.3, by removing $\varepsilon[n]$, we can express Model Formula 5.1 as Model Formula 5.2.

Model Formula 5.2

$$Y[n] \sim \mathcal{N}(b_1 + b_2 Sex[n] + b_3 Income[n], \sigma) \qquad n = 1, \ldots, N$$

We can also consider this model from a different aspect: $Y[n]$ is composed of the baseline variable $\mu[n]$, which is determined by the combination of the two explanatory variables, and a customer-specific noise. This gives an alternative description of the model:

Model Formula 5.3

$$\begin{aligned} \mu[n] &= b_1 + b_2 Sex[n] + b_3 Income[n] && n = 1, \ldots, N \\ Y[n] &\sim \mathcal{N}(\mu[n], \sigma) && n = 1, \ldots, N \end{aligned}$$

In fact, although specified in different manners, three model formulas represent an identical model. In order to keep a consistent expression as the logistic regression and Poisson regression models described below, we will use the Model Formula 5.3 to implement a Stan code. In this model formula, b_1, b_2, b_3, and σ are the parameters with noninformative priors, and they are to be estimated from the data. Because $\mu[n]$ is a parameter that is computed from these parameters, we define it in `transformed parameters` block in the Stan code.

5.1.5 Implement the Model

The implementation of Model Formula 5.3 is as follows:

model5-3.stan

```
1   data {
2     int N;
3     vector<lower=0, upper=1>[N] Sex;
4     vector<lower=0>[N] Income;
5     vector<lower=0, upper=1>[N] Y;
6   }
7
8   parameters {
9     vector[3] b;
10     real<lower=0> sigma;
11   }
12
13   transformed parameters {
14     vector[N] mu = b[1] + b[2]*Sex[1:N] + b[3]*Income[1:N];
15   }
16
17   model {
18     Y[1:N] ~ normal(mu[1:N], sigma);
19   }
20
21   generated quantities {
22     array[N] real yp = normal_rng(mu[1:N], sigma);
23   }
```

This code is similar to the one we used in a univariate regression model (**model4-4b.stan** in Sect. 4.3), thus we will emphasize on the parts that differ largely from the former.

Lines 3–4: The variables `Sex` and `Income`, instead of `X`, are used as data input. In addition, the ranges of the variables are specified to detect an error whenever the inputs include improper values.

Line 9: `vector` type is used for regression coefficients because the number of coefficients increased.

Line 14: `mu` is determined by the linear combination of `Sex` and `Income`.

Lines 21–23: This part is for further visualization of the results that we mentioned in Sect. 5.1.1. One way to plot the results is to check the prediction interval of response

variable for each customer. For instance, we can plot "observed value `Y[n]` for the `n`th customer" on the x-axis, and "median and Bayesian prediction interval of Y, computed from the known explanatory variable for this customer" on the y-axis (see Fig. 5.2 in later Sect. 5.2.1). Therefore, in `generated quantities` block here, in order to visualize the Bayesian prediction interval using MCMC sample from the predictive distribution, we draw a random value from a normal distribution with mean `mu[n]` and SD `sigma`, and assign this value to `yp[n]`.

5.1.6 Estimate Parameters

Let us estimate the parameters using R as in the previous chapter. Before that, let us practice "parameter recovery" that we introduced in Sect. 1.4. Here, we assign tentative values to the parameters, and generate simulation data based on Model Formula 5.3. Then we fit the same model for the generated data and estimate the parameters, and check if we are able to recover the tentative values.

As an example, the following R code shows how we generate the simulation data and estimate the parameters:

run-model5-3.R

```
library(cmdstanr)
set.seed(123)

d <- read.csv(file='input/data-shopping-1.csv')
d$Income <- d$Income/100
N <- nrow(d)
Y_sim_mean <- 0.2 + 0.15*d$Sex + 0.4*d$Income/100
Y_sim <- rnorm(N, mean=Y_sim_mean, sd=0.1)
data_sim <- list(N=N, Sex=d$Sex, Income=d$Income,Y=Y_sim)
data    <- list(N=N, Sex=d$Sex, Income=d$Income,Y=d$Y)

model <- cmdstan_model(stan_file='model/model5-3.stan')
fit_sim <- model$sample(data=data_sim, seed=123,parallel_chains=4)
fit   <- model$sample(data=data,   seed=123,parallel_chains=4)
```

The equivalent Python code is as follows:

run-model5-3.py

```
import pandas
import cmdstanpy
import numpy as np
np.random.seed(123)

d = pandas.read_csv('input/data-shopping-1.csv')
d.Income /= 100
Y_sim_mean = 0.2 + 0.15*d.Sex + 0.4*d.Income
Y_sim = np.random.normal(loc=Y_sim_mean, scale=0.1)
data = d.to_dict('list')
data.update({'N':len(d)})
```

```
12    data_sim = data.copy()
13    data_sim['Y'] = Y_sim
14
15    model =
cmdstanpy.CmdStanModel(stan_file='model/model5-3.stan')
16    fit_sim = model.sample(data=data_sim, seed=123,
parallel_chains=4)
17    fit   = model.sample(data=data,   seed=123,
parallel_chains=4)
```

Lines 4–8 in R (Lines 6–9 in Python): Simulation data is generated. We use the same explanatory variables as the ones used in the example data. For the three regression coefficients and the SD of noise, we assign constant values (0.2, 0.15, 0.4 and 0.1, respectively). We can easily visualize and get intuitive understanding of what type of data this model tends to generate, by changing these constant values or the random seed.

Line 13 in R (Line 16 in Python): The model is fitted to the simulation data. We need to check if the parameters are recovered by comparing the estimated values and the assigned value.

Line 14 in R (Line 17 in Python): The model is fitted to the example data. This result will be used in the next section.

Note that in both cases, we have divided `Income` values by 100 before passing it to the Stan code (Line 5 in R, Line 7 in Python). This is called data *scaling*, a process to transform all the parameter values to a similar scale (about the order of 1 in general). This is especially helpful when the original scales of all parameters vary largely. If the scales across the parameters differ, the sampling would be ineffective, computation cost will be expensive, and the MCMC might not converge easily. On the other hand, if the MCMC converges well, it might not be necessary to conduct scaling process.

In the code above, $Income$ is divided by 100 for scaling, because the maximum value of $Income$ is close to 100. Other common data scaling methods include:

- Divide the data with the unit value.
  For instance, if the dataset uses \$1000 as the price unit, divide the data with \$1000. This scaling will make the interpretation easier.
- Divide the data with the SD that is calculated from the data.
  This is a classic method for data scaling. After this process, the values would be transformed from "$1\times$ SD" to "1".
- Divide with the two times of the SD that is calculated from data.
  The value of "$2\times$ SD" (ranging from "$-1\times$ SD" to "$1\times$ SD") would become "1" after the scaling.
- Divide the data by the maximum of the data.

5.1.7 Interpret Results

The summary of the estimation result will be as follows:

```
1          Mean     MCSE      StdDev    5%      50%     95%     N_Eff     R_hat
2   lp__   136      3.9e-02   1.5e+00   133     136     138     1381524   1.0
3   ...
4   b[1]   0.15     6.3e-04   2.2e-02   0.11    0.15    0.18    1246      1.0
5   b[2]   0.14     2.3e-04   1.1e-02   0.12    0.14    0.16    2377      1.0
6   b[3]   0.44     8.6e-04   3.1e-02   0.39    0.44    0.49    1304      1.0
7   sigma  0.038    9.4e-05   4.2e-03   0.032   0.038   0.046   1974      1.0
8   mu[1]  0.52     1.8e-04   9.6e-03   0.50    0.52    0.54    2923      1.0
9   ...
```

Let us look at the summary statistics of the parameters We found that all the parameters (including `lp__`) have `Rhat` value lower than 1.1, thus concluded that MCMC has successfully converged (as explained in Sect. 2.3), and decided to proceed to the next step of analysis. It is helpful to check the trace plot as well. If we replace all the parameters in Model Formula 5.3 with their estimated posterior mean, we can rewrite the model formula as:

$$\mu[n] = 0.15 - 0.14\ Sex[n] + 0.44\ Income[n]/100 \qquad n = 1, \ldots, N$$
$$Y[n] \sim \mathcal{N}(\mu[n], 0.038) \qquad n = 1, \ldots, N$$

In this formula, there are three key points for the interpretation.

Regression Coefficients

The posterior mean of `b[3]` indicates that a customer with $Income = 90$ has $0.44 \times (90 - 40)/100 = 0.22$ higher purchase proportion on average than a customer with $Income = 40$. Similarly, the posterior mean of `b[2]` indicates that male customers have $0.14 \times (1 - 0) = 0.14$ higher purchase proportions on average than female customers.

As mentioned in Sect. 5.1.4, these regression coefficients only indicate that "a customer with a higher $Income$ by 50 has a higher purchase proportion by 0.22 on average in the given data". If we want to predict how much the purchase proportion will increase when we increase $Income$ by 50, the built model formula should be reflecting the causal relationship between $Income$ and the purchase proportion. Here, we assume that this model reflects the causal relationships between the variables. Here are some examples that do not reflect the causal relationships. In Sect. 4.1, we fit a univariate regression model with annual income as the response variable and the years of work experience as the explanatory variable. Now consider what happens if we switch the response and explanatory variables. Although regression coefficients still can be estimated, deriving such conclusion as "increasing annual income will increase age" is obviously misleading. This is the consequence of confusing the directionality of causality.

Extrapolation

Assume that there is another male customer with $Income = 200$. The posterior mean of his purchase proportion baseline μ can be calculated as $0.15 + 0.14 \times 1 + 0.44 \times 200/100 = 1.17$, whereas any purchase proportion is expected to be in the range of [0, 1]. This resulted from the fact that we predicted the response variable for the new data point with a much larger explanatory variable value ($Income = 200$) than the range of the explanatory variable used in the input data ($Income \in [38.7, 106]$). Predicting a value that is outside of the range of given data is called *extrapolation.* It should be taken extra caution when estimating such values and use the results for the further analysis.

Check Bayesian Confidence Intervals of Parameter Values

So far, we have been using parameter posterior mean only. This is to simplify the explanation of the regression coefficients and the concept of extrapolation. As mentioned in Chap. 2, we are more interested in the joint distribution and confidence intervals of parameters. In this example, the 95% Bayesian confidence interval of `b[3]` is [0.39, 0.49], and since this range is far away from zero, we can conclude that a customer with a higher $Income$ has a higher purchase proportion. In contrast, if the obtained Bayesian confidence interval was [−0.4, 0.8], which included zero, this conclusion cannot be derived. Borrowing the concept in classical view of significance tests for the regression coefficients, this probability of the parameter value being larger than (or smaller than) zero is sometime called *Bayesian p-value.*

5.2 Check Models

In this section, we will introduce how to use visualization to check what kind of properties the assumed model has. The visualization check is a handy and practical method that is frequently used.

From now on in this book, we will not specify Bayesian prediction intervals as "the X% interval of the MCMC sample that is drawn from posterior predictive distribution" every time, because Bayesian prediction intervals are always computed from the draws. Instead, we refer "the X% interval of predictive distribution" or simply "prediction interval" as the equivalents. Same would apply to the median value for the predictive distribution. Besides, so far, we have been using the 95% interval to adjust to what is used in classical statistics. However, analysts are free to adjust this range based on the problem settings. Accordingly, in this chapter, we consider that 80% of prediction accuracy is enough for the purpose, and thus use 80% posterior prediction intervals.

5.2.1 Posterior Predictive Check (PPC)

Posterior predictive check (PPC) is to overlay the posterior predictive distribution and data distribution on a single plot. By visualizing them together and check if there is a large discrepancy, we can assess whether the assumed model is proper for the given data. One practical suggestion when using a large number of explanatory variables is to reduce the number of variables for visualization and summarize the posterior predictive distributions. Specific examples are given below. Of note, these plots are merely examples from many others, and there are far more alternative approaches to do PPC. Based on the purpose set up, a PPC plot that is relevant and interpretable should be chosen.

When There is No Explanatory Variable

The data and the posterior predictive distributions can be visualized together (Fig. 5.2, left).

When There is One Explanatory Variable

This case includes simple linear regression. In this case, the change of the explanatory variable value might cause the corresponding change of shape and the location of the posterior predictive distribution (Fig. 5.2 middle). It should be checked if the data points are within the prediction intervals and whether there is a large discrepancy between the observed data and the predicted value.

When There are Two or More Explanatory Variables

This case includes multiple linear regression. When the number of axes increases, we cannot use the approach we used for visualizing Fig. 5.2 (middle). Instead, we generate a scatter plot with the observed data on the x-axis and the predicted values

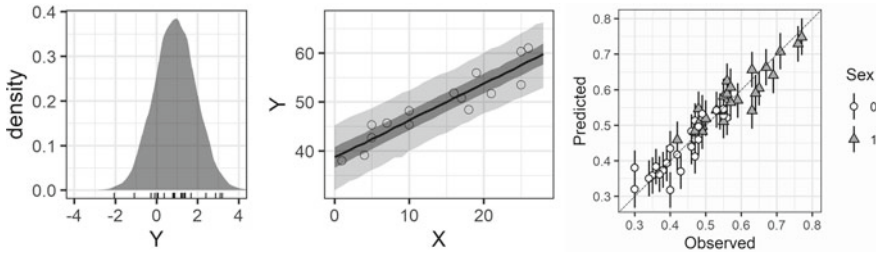


Fig. 5.2 PPC plots. Left: an example from Sect. 2.2. The gray bell-shaped distribution is the posterior predictive distribution, and the black vertical lines on the bottom represent where each data point locates on the x-axis. Middle: an example from Sect. 4.2. Reproduction of Fig. 4.8 (right). The light gray band are the 95% prediction intervals, the dark gray band are the 50% prediction intervals, and the black line are the medians of the predictive distributions. Right: an example from Sect 5.1. The x-axis represents the observed value $Y[n]$, and the y-axis represents the medians (points) and 80% prediction intervals (vertical lines) of $y[n]$ computed from explanatory variables for each customer

and intervals on the y-axis (Fig. 5.2 right). By checking the distance from the points to the diagonal line, which represents "predictive value = observed value", we are able to comprehend if there is systematic bias in the prediction.

We used the data with two explanatory variables in the last section. Therefore, we will use the third method to check the model for this example. We generated a scatter plot with the observed value $Y[n]$ (`Y` in **`model5-3.stan`**) on the x-axis and the predicted Y (calculated from the explanatory variables of the customer) on the y-axis, as shown in Fig. 5.2 (right).[1] From this plot, we can see that most dots are close to the diagonal line which denotes "predicted value = observed value", and this diagonal line is almost in the range of 80% prediction intervals. Therefore, we conclude that the response variable is sufficiently predictive from these two explanatory variables. Furthermore, from the observation that there is no systematic bias in the positions of points in the plot, we can conclude that this model does not have a big problem.

5.2.2 Posterior Residual Check (PRC)

Posterior residual check (PRC) is similar to PPC. But rather than checking the predictive distribution, it is to check whether the noise (i.e., the difference between the observation and the baseline) fits well with the assumed distribution, or whether its distribution has any systematic bias.[2]

In Sects. 5.1.3 and 5.1.4, we assumed that noise term $\varepsilon[n]$ follows $\mathcal{N}(0, \sigma)$. Now we want to check whether the estimated $\varepsilon[n]$, which can be calculated from $Y[n] - \mu[n]$, does indeed follow a normal distribution (Fig. 5.3 left). Because every $\varepsilon[n]$ is estimated for each customer in the Bayesian framework, visualizing all N plots on top of each other will be messy and not very helpful for model checking.

Alternatively, for each $\varepsilon[n]$ we can choose one representative value and use this value to compare it with $\mathcal{N}(0, \sigma)$ (Fig. 5.3, right). Because the purpose is to check the tendency of estimates, it does not matter if we use posterior mean, MAP or median as this representative value. Here, in order to keep integrity with the classical multiple regression analysis, we used MAP estimate (see Sect. 2.5). From Fig. 5.3 (right), we confirmed that the distribution of the representative values has no systematic bias and is similar to $\mathcal{N}(0, \sigma)$. Therefore, we concluded that this analysis is satisfactory.

Although we did not detect an issue in Figs. 5.2 (right) and 5.3, this is not always the case especially when we are working with real data. Some issues can be manifested from these visualizations, such as:

---

[1] Sometimes, it is also plotted with predicted values on the x-axis and observed values on the y-axis. Because here we want to detect which data points are not fitting well, we assigned observed values on the x-axis.

[2] For the similar reason, sometimes a residual plot (with predicted value on the x-axis and "predicted value—observed value" on the y-axis) or a QQ plot are used. Since these methods are not discussed in this book, interested readers can use "residual plot" as a keyword to search for more information.

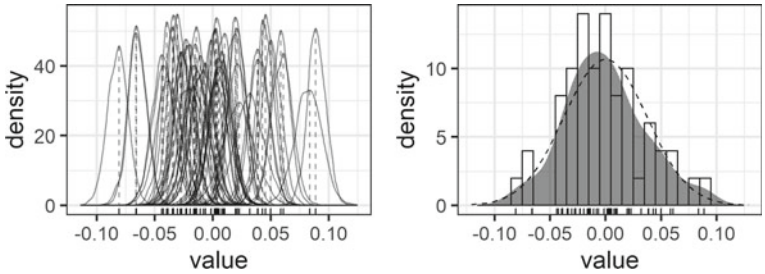


Fig. 5.3 Left: overlaid density plots of $\varepsilon[n]$ for N customers. The dotted vertical lines and the vertical bars below are the location of MAP estimates. Right: a histogram of MAP estimates for N customers (white bars), and the density function that is approximated by a smooth curve (dark mountain). The dotted curve represents the density function of the normal distribution with mean 0 and the SD σ_{MAP}, which is the MAP estimate of sigma from **`model5-3.stan`**

- In Fig. 5.2 (right), only the customers with specific features (e.g., female customers) are out of the straight line, or their prediction interval is wider than the others.
- In Fig. 5.2 (right), only the data points with small values are above the line.
- In Fig. 5.3, the distribution of representative values of $\varepsilon[n]$ is skewed, or is inconsistent from the assumed distribution.

In such cases, the model can be pathological and should be adjusted. Alternatively, the purchase proportion may be affected by other external factors and cannot be explained fully with current variables. If that is the case, other data may need to be collected.

In addition, by using the same approach as we used for PRC, we can check if the estimated value follows the assumed distribution. For instance, in hierarchical model that we will discuss in Chap. 8, the group difference and individual difference are assumed to follow normal distributions. Therefore, we can obtain the representative values of each difference, and then compare the distribution of this values to the normal distribution from the assumption.

5.2.3 Scatterplot Matrix of MCMC Sample

Although this is not directly relevant to model checking, in order to deepen the understanding of the used model, we suggest visualizing the scatterplot matrix of MCMC sample as we did before (see Fig. 4.7). It visualizes a joint posterior distribution. The visualization based on the result of Sect. 5.1 is given in Fig. 5.4. This scatterplot matrix indicates some relevant information. For instance, `b[1]` and `b[3]` have a negative correlation, and `sigma` and `lp__` do not have any strong correlations to other parameters. If there is any unexpected result in the scatterplot matrix, it may indicate how to improve the model.

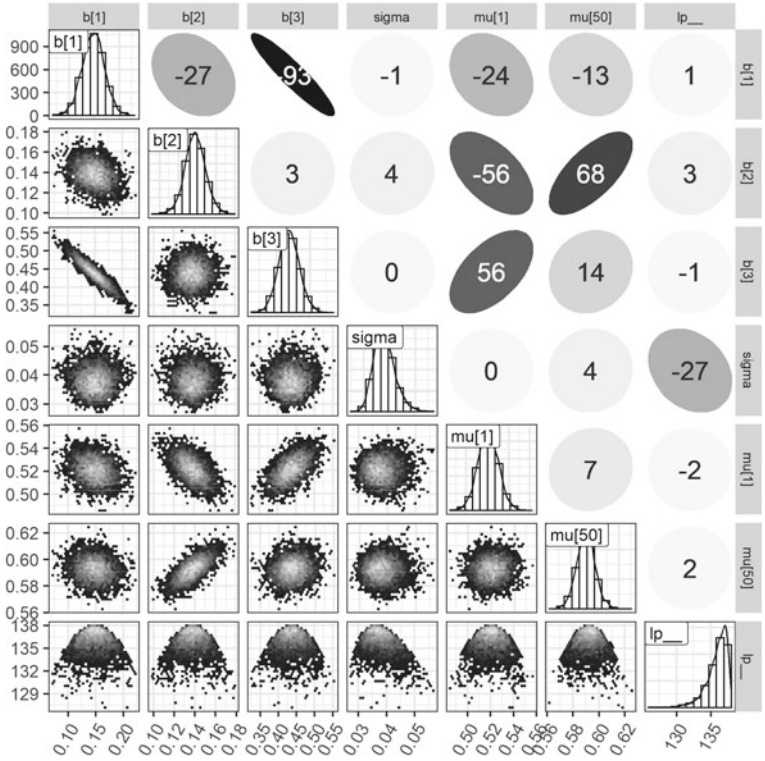


Fig. 5.4 Scatterplot matrix of the MCMC sample. The diagonal plots show the histogram of the draws and their density functions. The plots on the lower left are the scatterplots (lighter color indicates higher density), and the plots on the upper right represents the pairwise Spearman correlation coefficients (multiplied by 100) between two parameters. We have 50 customers, hence 50 marginal posterior distributions for `mu`, but we only show the first and the last customers as examples

5.3 Binomial Logistic Regression

In this section, we use the data similar to the one we worked in Sect. 5.1. Assume that we have a comma-separated file (Data file 5.2, a row corresponds to a customer). Two additional columns represent:

M : How many times the customer visited the website in the past month.

Y : How many times the customer purchased in the past month.

Other columns have same meanings as Data file 5.1.

Data file 5.2 Structure of `data-shopping-2.csv`

```
PersonID,Sex,Income,M, Y
1,0,85.1,9,6
2,1,64.6,12,4
3,0,97.6,6,5
...
50,1,70,12,10
```

5.3.1 Set Up Purposes

The purpose of this analysis is the same as in Sect. 5.1. That is, we want to know how much can be predicted about the response variable from two explanatory variables Sex and $Income$. Also, we want to know how much each of the variable is contributing to the purchase probability. Slight difference from the previous section is that now we consider the response variable is Y under a fixed M. In addition, we want to visualize PPC to report the results.

5.3.2 Check Data Distribution

As in the previous example, we visualized a scatterplot matrix in order to overview data distributions of each column and their pairwise relationships (Fig. 5.5). Here we added an additional column "purchasing proportion $= Y/M$". From this figure, it comes as no surprise that when the purchase count increases, the proportion also goes up, which indicates that these two variables have a strong correlation. Other information, for instance a positive correlation between sex and the count of visiting the website, can also be noted from scatterplot matrix.

5.3.3 Imagine Data Generating Mechanisms

Recall that one of the major purposes of this analysis is to understand the effects of explanatory variables on the purchase probability. Consider what happens if we use the column of purchase proportion $= Y/M$ as the response variable and fit the same multiple regression model as we did in Sect. 5.1.

In this case, the difference from Sect. 5.1 is that there are several observations with zero purchase proportion. Therefore, certain combinations of the explanatory variables may lead predicted purchasing probability to be negative, which is not what we want.[3] Therefore, we can transform the linear combination of the explanatory

[3] When it is a data with large amount of 100% purchase proportion, predicted value can be larger than 1.

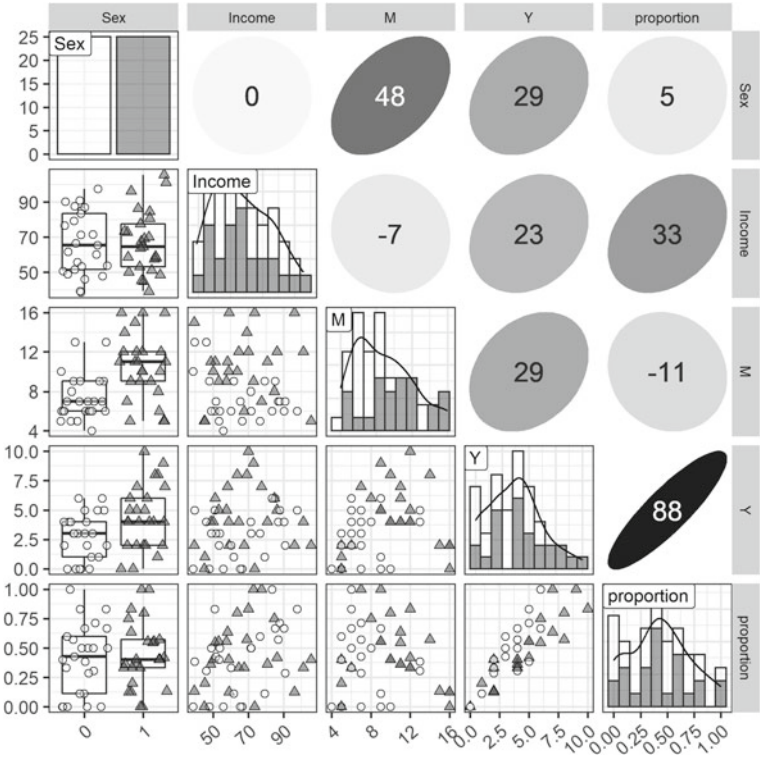


Fig. 5.5 Data scatterplot matrix. The structure of columns and indication colors are the same as Fig. 5.1

variables $b_1 + b_2 Sex + b_3 Income$, which is in the range of $(-\infty, \infty)$, into another value so that the new value is constrained in the range of $(0, 1)$. One of the most common methods for such a transformation is using a logistic function. The logistic function is the inverse function of the logit function, and it is available in Stan as `inv_logit`. Because the linear combination of the explanatory variables is transformed via logistic function, which is a monotonically increasing function, it should be noted that if one of the explanatory variables increases, Y will increase (or decrease) monotonically (Fig. 5.6).

Now let us consider the noise distribution. In the multiple regression, we assumed that the noise follows a normal distribution. In the case of this section, however, if we add noise after data transformation, the new value may lie outside of $(0, 1)$ range. Furthermore, the purchase proportions are rather discrete than continuous, because the total visiting counts, which are the denominators of the purchase proportions, are small (from 4 to 16) in this data. For these reasons, a model using a normal distribution that generates noise with continuous values in the range of $(-\infty, \infty)$ does not seem to fit well as a model for estimating the purchase probability of this data.

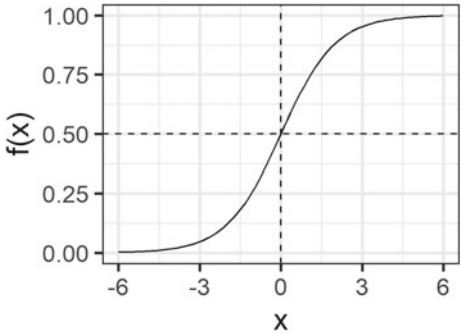


Fig. 5.6 Logistic function (`inv_logit` function)

Here we use a binomial distribution instead of a normal distribution. In fact, it is more natural to use binomial distribution for count data (counts of visiting and counts of purchases). When the purchase proportion data do not include some extreme points such as 0.0 or 1.0, and the values corresponding to the denominators are relatively large, then a binomial distribution can be well-approximated by a normal distribution.[4] When we are not sure whether to use a normal distribution to approximate, it is also a good idea to start with a normal distribution, and change to a binomial distribution if there's any issue with fitting the data. Generally, in probabilistic programming language, using a normal distribution is faster than the combination of logistic function and a binomial distribution. From this perspective, using a normal distribution is recommended when it is appropriate.

To summarize, here we considered the data generation mechanism as follows: first we consider the purchase probability instead of the purchase proportion, and the purchase probability q is determined by the linear combination of Sex and $Income$, which is $b_1 + b_2 Sex + b_3 Income$. This linear combination is transformed to the range of (0, 1) by a logistic function. The count of purchases Y in the M visits is determined by the binomial distribution with the purchase probability q.

5.3.4 Describe Model Formula

The above mechanism can be specified as the following model:

Model Formula 5.4

$$q[n] = \text{inv_logit}(b_1 + b_2 Sex[n] + b_3 Income[n]) \qquad n = 1, \ldots, N$$
$$Y[n] \sim \text{Binomial}(M[n], q[n]) \qquad n = 1, \ldots, N$$

where N represents the total number of customers, and n represents the index of each customer. b_1, b_2, and b_3 are the parameters to be estimated from the data.

[4] For those who are not familiar with a binomial distribution, please see Sect. 6.4.

5.3.5 Implement the Model

The implementation of Model Formula 5.4 is as follows:

model5-4.stan

```
 1    data {
 2     int N;
 3     vector<lower=0, upper=1>[N] Sex;
 4     vector<lower=0>[N] Income;
 5     array[N] int<lower=0> M;
 6     array[N] int<lower=0> Y;
 7    }
 8
 9    parameters {
10     vector[3] b;
11    }
12
13    transformed parameters {
14      vector[N]  q  =  inv_logit(b[1]  +  b[2]*Sex[1:N]  +
         b[3]*Income[1:N]);
15    }
16
17    model {
18     Y[1:N] ~ binomial(M[1:N], q[1:N]);
19    }
20
21    generated quantities {
22     array[N] int yp = binomial_rng(M[1:N], q[1:N]);
23    }
```

We emphasize on those points that differ from the previous multiple regression model code (**model5-3.stan**).

Line 6: Because only integer values are allowed on the left-hand-side of `binomial` function in line 18, the data type of `Y` is changed from `vector` (purchase proportions) to `array` of `int` (purchase counts). Note that `vector` in Stan can only store `real` values. The same reason applies for `M` in line 5.

Line 14: The value is changed to the range (0, 1) using `inv_logit` function. The function can be vectorized.

Lines 18 and 22: A binomial distribution is used to replace the normal distribution. The type of `yp` is changed to `array` of `int`.

5.3.6 Interpret Results

The summary of the estimated parameters will be as follows:

```
1          Mean    MCSE      StdDev   5%      50%    95%     N_Eff   R_hat
2   b[1]   -1.5    1.1e-02   0.40     -2.1    -1.5   -0.82   1260    1.00
3   b[2]   0.097   4.3e-03   0.20     -0.23   0.10   0.41    2084    1.0
4   b[3]   1.6     1.6e-02   0.57     0.68    1.6    2.5     1323    1.0
5   q[1]   0.47    9.3e-04   0.047    0.39    0.47   0.55    2521    1.0
6   ...
```

Substituting the posterior means of the parameters into Model Formula 5.4 gives the following:

$$q[n] = \text{inv_logit}(-1.5 + 0.10\ Sex[n] + 1.6\ Income[n]/100) \quad n = 1, \ldots, N$$
$$Y[n] \sim \text{Binomial}(M[n], q[n]) \quad n = 1, \ldots, N$$

In order to understand how the explanatory variables affect the purchase probability in this model, first we need to understand the property of inv_logit function. In fact, the equation of q is:

$$\begin{aligned} q &= \text{inv_logit(linear combination of explanatory variables)} \\ &= \frac{1}{1 + \exp(-(\text{linear combination of explanatory variables}))} \end{aligned}$$

This can be transformed to:

$$\frac{q}{1-q} = \exp(\text{linear combination of explanatory variables})$$

Here, the left-hand side is called *odds*, defined as the probability of purchasing divided by the probability of not purchasing. When we use odds, it can be said that the probability of purchasing is XX times higher than the probability of not purchasing. For instance, we can say that the odds are one when $q = 0.5$, and the odds are four times higher when $q = 0.8$. One of the advantages of using inv_logit (logistic function) for statistical modeling is that it simplifies the interpretation of the explanatory variables from the perspective of how they affect the odds. On the other hand, because inv_logit is a non-linear function, it is impossible to interpret the regression coefficient for q in the same way as for linear regression.

The result of the example in this section is:

$$\begin{aligned} \frac{q}{1-q} &= \exp\left(-1.5 + 0.10\ Sex + 1.6\frac{Income}{100}\right) \\ &= \exp(-1.5)\exp(0.10\ Sex)\exp\left(1.6\frac{Income}{100}\right) \end{aligned}$$

Next, by comparing the odds of a customer with $Income = 90$ with a customer with $Income = 40$,

$$\begin{aligned}\frac{Odds\ when\ Income = 90}{Odds\ when\ Income = 40} &= \frac{\exp(-1.5)\exp(0.10\ Sex)\exp\left(1.6\frac{90}{100}\right)}{\exp(-1.5)\exp(0.10\ Sex)\exp\left(1.6\frac{40}{100}\right)} \\ &= \frac{\exp\left(1.6\frac{90}{100}\right)}{\exp\left(1.6\frac{40}{100}\right)} = \exp\left(1.6\frac{90-40}{100}\right) \\ &\cong 2.2\end{aligned}$$

We can say that the odds of a customer whose income is \$90(k) are on average 2.2 times higher than the odds of a customer whose income is \$40(k). Similarly, we can say that male customers have on average 1.1 times higher odds than female customers. Using a ratio of odds can simplify the interpretation of the effect of explanatory variables.

We used the posterior mean of each parameter to explain the concept of odds. The three points for interpretation that we introduced in Sect. 5.1.7 apply for logistic regression as well. We skip the discussion on model check in this section.

5.4 Logistic Regression

The previous data was about the counts of visits and the counts of purchases, collected for a month for each customer. Let us consider that after preliminary analysis of this data, we realized that the products discounts have great impact on the purchase probability. Hence in this section, we will use data before aggregating the counts of visits and purchases. And this data is a comma-separated file with records of 444 times of visits (Data file 5.3, a row corresponds to a visit). Each column represents:

Discount : a binary value that indicates if there was a discount (0: no discount, 1: discount)

Y: a binary value that indicates if the customer purchased or not (0: did not purchase, 1: purchased)

Other columns are identical to Data file 5.2. Note that one $PersonID$ can appear in multiple rows, because the same customer can visit and purchase several times.

Data file 5.3 Structure of `data-shopping-3.csv`

```
  1    PersonID,Sex,Income,Discount,Y
  2    1,0,85.1,0,0
  3    1,0,85.1,1,1
  4    1,0,85.1,0,0
       ...
445    50,1,70,0,1
```

5.4.1 Set Up Purposes

For this section, we assume that we want to know if and how the three explanatory variables (Sex, $Income$, $Discount$) can predict the response variable Y, as well as how much each of them contributes to the purchase probability. Because here we are considering probability, we choose to use logistic regression. Again, as in previous part, consider how to make plots for presentation purposes.

5.4.2 Check Data Distribution

It is not easy to make a scatter plot for this data, because the columns of Sex, $Discount$ and Y are all binary variables. Therefore, we can aggregate Y based on whether there was a discount or not. This can be done in R as follows:

```
> d <- read.csv(file='input/data-shopping-3.csv')
> aggregate(Y ~ Discount, data=d, FUN=table)
  Discount Y.0 Y.1
1        0 232 112
2        1  30  70
```

From this result, the purchase probability seems to be higher when there is a discount (`Discount = 1`), compared to no discount (`Discount = 0`). Both R and Python are especially good at data aggregation and processing. We suggest interested readers refer web sites on `{dplyr}` for R or Pandas for Python.

5.4.3 Imagine Data Generating Mechanisms

As we did in Sect. 5.3, we consider that the purchase probability is transformed from the linear combination of explanatory variables, using logistic function. In Sect. 5.3, a binomial distribution was used to represent the counts of purchases in the M visits. In this section, because we consider the purchase of each customer in every single visit, rather than in the M visits, thus we use a Bernoulli distribution instead of a binomial distribution.[5]

[5] For those who are not familiar with a Bernoulli distribution, please see Sect. 6.3.

5.4.4 Describe Model Formula

The above mechanisms can be written as.

Model Formula 5.5

$$q[v] = \text{inv_logit}(b_1 + b_2 Sex[v] + b_3 Income[v] + b_4 Discount[v]) \quad v = 1, \ldots, V$$
$$Y[v] \sim \text{Bernoulli}(q[v]) \quad v = 1, \ldots, V$$

where V represents the total count of visits (= 444 in this example), and v represents the index of each visit. We changed the notation from n to v, because the index represents a visit (instead of a customer). b_1, b_2, b_3, and b_4 are the parameters to be estimated from the data.

Behind this model, there is an assumption that data points from each visit are independently generated. Therefore, the customers who visited 30 times, compared to the customers who visited 10 times, have three times heavier weights for the likelihood. In addition, we did not consider the fact that one customer has multiple visiting records, and the fact that each customer has different purchasing behaviors. The hierarchical model, which we will be discussing in Chap. 8, can account for the individual differences that do not depend on the Sex and $Income$. The rule of thumb is, however, to start the modeling from the simpler assumptions. In this chapter, we will continue the analysis from this simpler model.

5.4.5 Implement Models

The implementation of Model Formula 5.5 is as follows:

model5-5.stan

```
data {
  int V;
  vector<lower=0, upper=1>[V] Sex;
  vector<lower=0>[V] Income;
  vector<lower=0, upper=1>[V] Dis;
  array[V] int <lower=0, upper=1> Y;
}

parameters {
  vector[4] b;
}

transformed parameters {
  vector[V] q = inv_logit(
    b[1] + b[2]*Sex[1:V] + b[3]*Income[1:V] + b[4]*Dis[1:V]);
}

model {
```

```
19      Y[1:V] ~ bernoulli(q[1:V]);
20    }
```

Because it is similar to the code of Binomial logistic regression (**model5-4.stan**), we will focus on describing the major differences. The part that generates predictive distribution is not included in this code.

Line 5: A new variable `Dis` is added to represent if there was a discount or not.

Line 19: A Bernoulli distribution is used instead of a binomial distribution.

In case we need to know the regression coefficients, but do not need to know the purchase probability `q[v]` for each customer, then the internal function `bernoulli_logit` can be used to speed up the computation, as follows:

model5-5b.stan

```
1    data {same as model5-5.stan}
2
3    parameters {same as model5-5.stan}
4
5    model {
6      Y[1:V] ~ bernoulli_logit(
7       b[1] + b[2]*Sex[1:V] + b[3]*Income[1:V] + b[4]*Dis[1:V]);
8    }
```

Likewise, there is another useful internal function `binomial_logit`, which we did not introduce in Sect. 5.3.

The R/Python code to run this Stan code is very similar to the one for multiple regression model, and the interpretation of the estimated results are the same as that in Sect. 5.3, therefore we leave them out for the readers.

5.4.6 PPC

Because the response variable of logistic regression is binary, it is not very straightforward to compare the observed and the predicted value. But in fact, it follows the same idea. We can plot the observed purchase value ($Y[v]$) on the x-axis and the estimated purchase probability for each visit ($q[v]$) on the y-axis. When there are many data points, we can visualize the calculated density function of the medians of the distributions of $q[v]$, or plot box plots of the medians, as shown in Fig. 5.7 (left). If the prediction is good, there should be more points and density distribution on the lower part of the plot when $Y = 0$, and on the upper part of the plot when $Y = 1$. Unfortunately, this is not the case in Fig. 5.7, therefore we conclude that the prediction accuracy is relatively low.

When the response variable is discrete, Receiver Operating Characteristic (ROC) curve is another tool for prediction assessment. For the estimated purchase probability $q[v]$, first we define a threshold that is used for the classification. Let us say we set the threshold as 0.6, and the data points v where $q[v]$ is under the threshold is classified as not purchasing, and above the threshold is classified as purchasing. We then compare this classification result with the observed value, and quantify the accuracy of classification. The same process should be repeated under different

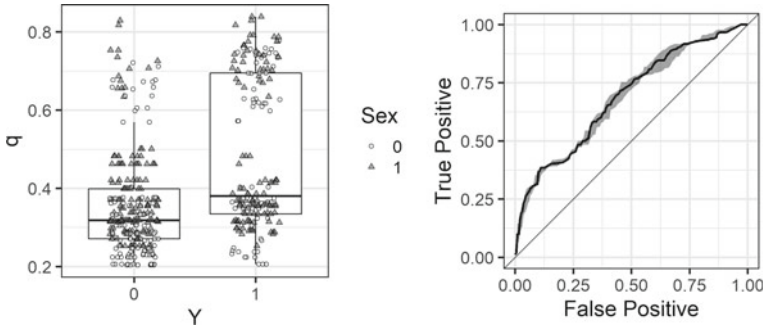


Fig. 5.7 PPC when the response variable is binary. Left: the plot of observed values and the estimated probabilities. Each dot represents a visiting record. Because there are many data points, we only show the posterior median. The colors and shapes of the dots indicate different genders. The value of Y is either 0 or 1, but we added horizontal jitters, for the visualization purpose. Right: ROC curve. The gray band is the 80% interval and the black line is the median

thresholds. Although in this book we will not give further discussion, those interested readers can refer to the corresponding Wikipedia page.[6] The ROC curve obtained from draws is shown in Fig. 5.7 (right). The Area Under the Curve (AUC) can be calculated for classification performance assessment. The closer AUC is to 1, the better the classification is. In general, a good classifier satisfies AUC ≥ 0.8. In this example, because our AUC $= 0.688$ (and the 80% Bayesian confidence interval is [0.676, 0.692]), we conclude that the classification performance is not really good.

5.5 Poisson Regression

Here we will use the same dataset we used in Sect. 5.3. The only difference is that the purpose of this analysis is to understand how much the two explanatory variables of customers (Sex and $Income$) contribute to their counts of visits M. For instance, we are interested to know whether male customers are more likely to visit the website than the female customers. Hence, instead of using Y, which is the counts of purchases, we will use counts of visits M as the response variable. As in this case, when we use data that is non-negative and discrete, and has no upper limit, using Poisson distribution is the first option.

[6] https://en.wikipedia.org/wiki/Receiver_operating_characteristic.

5.5.1 Imagine Data Generating Mechanisms

In above discussion, we mentioned that for this data, the first choice is to use a Poisson distribution for the regression. Let us consider this more into detail. If we follow the same logic as in Sect. 5.1, then we consider that the time of visits M can be modeled as a multiple regression that is represented as the linear combination of two explanatory variables and a noise term that follows a normal distribution. However, this may give negative or fractional values, which would be hard to interpret because the values of the response variable are integers, and it is likely that multiple regression model would not fit the data well.

Therefore, we look at the data generation process from a different perspective. Because the counts of visits are nonnegative, $b_1 + b_2 Sex + b_3 Income$ (the linear combination of Sex and $Income$) is transformed to the nonnegative range to define the mean count of visits λ. The visit counts for each customer, M, are generated probabilistically from a Poisson distribution with parameter λ.[7] The regression using a Poisson distribution is called *Poisson regression.* The exponential function exp is commonly used in order to transform the value range into nonnegative. Because we transform the linear combination of explanatory variables using the exponential function, which is a monotonic function, an increase in one of the explanatory variables would cause monotonic increase (or decrease) in response variable Y.

The fact that the counts of visits follow a Poisson distribution means that we assume that time intervals from a visit to the next visit are independently generated from an exponential distribution (see Sect. 6.9). In general, it is difficult to know whether or not the time intervals are independent, thus for now, it is sufficient to realize that the analysis is proceeding under the strong assumption. Furthermore, a Poisson distribution has the property that the variance is equal to the mean, so it does not work well for data where the variance is clearly larger than the mean. In that case, as in the case of logistic regression, a hierarchical model will be used, assuming that there are large differences derived from individual customers.

When the parameter λ is large enough, a Poisson distribution can be approximated by a normal distribution. Because in this example, counts of visits M is in a relatively small range (4–16), we consider the parameter λ is relatively small as well. But when this value is larger (for instance, over 100), using a normal distribution instead of the Poisson distribution may be a better choice.

5.5.2 Describe Model Formula

This mechanism can be written as the model formula as follows:

Model Formula 5.6

$$\lambda[n] = \exp(b_1 + b_2 Sex[n] + b_3 Income[n]) \qquad n = 1, \ldots, N$$

[7] For those who are not familiar with a Poisson distribution, please refer to Sect. 6.10 in this book.

$$M[n] \sim \text{Poisson}(\lambda[n]) \qquad n = 1, \ldots, N$$

where N represens the number of customers, n represents the index of each customer. b_1, b_2, and b_3 are the parameters to be estimated from the given data.

5.5.3 Implement the Model

The implementation of Model Formula 5.6 is as follows:

model5-6.stan

```
1   data {
2     int N;
3     vector<lower=0, upper=1>[N] Sex;
4     vector<lower=0>[N] Income;
5     array[N] int<lower=0> M;
6   }
7
8   parameters {
9     vector[3] b;
10   }
11
12   transformed parameters {
13   vector[N] lam = exp(b[1] + b[2]*Sex[1:N] + b[3]*Income[1:N]);
14   }
15
16   model {
17     M[1:N] ~ poisson(lam[1:N]);
18   }
19
20   generated quantities {
21     array[N] int mp = poisson_rng(lam[1:N]);
22   }
```

We explain the differences of this code to the binomial logistic regression model (**model5-4.stan**).

Line 5: The response variable is changed to `M`, which represents the counts of visits.

Line 13: `exp` function is used to transform the linear combination of explanatory variables to nonnegative.

Lines 17 and 21: A Poisson distribution is used instead of a normal distribution. `poisson_log(x)` function in Stan is equivalent to `poisson(exp(x))`. Because using former is more stable, thus it is recommended. For the same reason, `poisson_log_rng(x)` can be used to replace `poisson_rng(exp(x))`. By replacing these in the above code, the code can be written as follow (the modified part is highlighted in gray):

model5-6b.stan

```
data {same as model5-6.stan}

parameters {same as model5-6.stan}

transformed parameters {
  vector[N] log_lam = b[1] + b[2]*Sex[1:N] + b[3]*Income[1:N];
}

model {
  M[1:N] ~ poisson_log(log_lam[1:N]);
 }

 generated quantities {
  array[N] int mp = poisson_log_rng(log_lam[1:N]);
 }
```

Because R and Python code to run the Stan code is almost the same as multiple regression model, we leave this part out.

5.5.4 Interpret Results

The summary of the estimated parameters from **model5-6b.stan** will be as follows:

	Mean	MCSE	StdDev	5%	50%	95%	N_Eff	R_hat
b[1]	2.1	5.0e-03	0.19	1.8	2.1	2.4	1499	1.0
b[2]	0.37	2.3e-03	0.098	0.21	0.37	0.54	1826	1.0
b[3]	-0.19	7.2e-03	0.28	-0.64	-0.19	0.26	1467	1.0
...								

Substituting the posterior means of the parameters into Model Formula 5.6 gives the following:

$$\lambda[n] = \exp(2.1 + 0.37\ Sex[n] - 0.19\ Income[n]/100) \qquad n = 1, \ldots, N$$
$$M[n] \sim \text{Poisson}(\lambda[n]) \qquad n = 1, \ldots, N$$

Let us see what we can learn about the effect of explanatory variables on the count of visits. For instance, we can consider a customer whose $Income$ is 90, versus a customer whose $Income$ is 40:

$$\frac{\text{Mean visit count when } Income = 90}{\text{Mean visit count when } Income = 40} = \frac{\exp(2.1)\exp(0.37\ Sex)\exp\left(-0.19\frac{90}{100}\right)}{\exp(2.1)\exp(0.37\ Sex)\exp\left(-0.19\frac{40}{100}\right)}$$
$$= \exp\left(-0.19\frac{90-40}{100}\right) \cong 0.91$$

Therefore, we can say that the mean visit counts for customers whose $Income = 90$ are on average 0.91 times of those whose $Income = 40$. Similarly, on average, visit counts of male customers are $\exp(0.37) \cong 1.4$ times of those of female customers. As we can see from this example, the advantage of regression using an exponential function and a Poisson distribution is that it is easy to interpret the result from the viewpoint of fold change in the count of visits.

5.6 Expression Using Matrix Operation

Here, we will consider the multiple linear regression mentioned in Sect. 5.1 but with the case where the number of explanatory variables increases to four. Consider that we have a comma-separated file for 50 customers (Data File 5.4, a row corresponds to a customer).

Data File 5.4 Structure of `data-shopping-4.csv`

```
PersonID,Sex,Income,X3,X4,Y
1,0,85.1,−0.56,1.2,0.56
2,1,64.6, −0.23,0.67,0.59
3,0,97.6,1.56,0.98,0.56
...
50,1,70, −0.08,1.46,0.64
```

The columns $X3$ and $X4$ are the newly added explanatory variables. When the number of explanatory variables increases, it would be tedious to declare these variables one by one in `data` block. Therefore, we can make the specification simpler by using matrix and put all of them into a single variable. This can be written as the model equation below:

Model Formula 5.7

$$\mu[n] = \left(\mathbf{X}\vec{b}\right)[n] \qquad n = 1, \ldots, N$$

$$Y[n] \sim \mathcal{N}(\mu[n], \sigma) \qquad n = 1, \ldots, N$$

where N indicates the total number of individuals and n represents the index for each. $\mathbf{X}$ is a matrix with N rows and $(1 + 4)$ columns, created by horizontally stacking the column of the intercept term (with all values equal to 1) and columns of four explanatory variables. The coefficients for the explanatory variables, $\vec{b}$, are also a vector of length $(1 + 4) = 5$. $\mathbf{X}\vec{b}$ is a multiplication operation and it returns a vector of length N. $\left(\mathbf{X}\vec{b}\right)[n]$ represents the n-th element in this vector. For instance, if we rewrite this for $n = 3$, it will give $\left(\mathbf{X}\vec{b}\right)[3] = b_1 + \mathrm{X}_{3,2}b_2 + \mathrm{X}_{3,3}b_3 + \mathrm{X}_{3,4}b_4 + \mathrm{X}_{3,5}b_5$ (because $\mathrm{X}_{3,1} = 1$).

The implementation of Model Formula 5.7 is as follows:

model5-7.stan

```
1   data {
2     int N;
3     int D;
4     matrix[N,D] X;
5     vector<lower=0, upper=1>[N] Y;
6   }
7
8   parameters {
9     vector[D] b;
10     real<lower=0> sigma;
11    }
12
13    transformed parameters {
14     vector[N] mu = X*b;
15    }
16
17    model {
18     Y[1:N] ~ normal(mu[1:N], sigma);
19    }
```

Line 3: We declare `D` as the value of "1 (for intercept term) + the number of explanatory variables". This value equals to 5 in this example.

Line 4: $\mathbf{X}$ is declared as `X` of `matrix` type with N rows and D columns. The gender, which are the integers 0 or 1, are converted into real numbers by becoming a part of the matrix type, because `matrix` type can only store `real` values.

Line 9: The coefficient `b` is a vector of length `D`.

Line 14: `X*b` is equivalent to $\mathbf{X}\overrightarrow{b}$ in Model Formula 5.7. This represents a multiplication of a matrix and a vector, and the result is a vector of length `N`. If `b` is an array, we cannot write it in this way, because the multiplication of `matrix` type and a one-dimensional array is not defined.

In the R and Python code to run **model5-7.stan**, we read Data File 5.4, remove the $PersonID$ and Y columns, and pass the remaining data frame to `X` in Stan.

5.7 Supplemental Information and Exercises

We did not mention probit regression in this chapter. It is similar to logistic function, but probit function uses the cumulative distribution function of standard normal distribution (Φ) to transform the range of the linear combination of explanatory variables into the range of (0,1). Despite the fact that it should not lead to a big difference compared to the results from using a logistic regression, sometimes it can facilitate the result interpretations.[8] In this chapter, we focused on the explanation

[8] For instance, consider a case when difficulties levels of questions are generated from a normal distribution, and the subjects' ability follows another normal distribution. Whether the subject success or fail to answer the question is based on the differences between these two variables. The situation leads to probit regression. Another example is winning or losing a game (see Sect. 9.1.5).

of odds using logistic regression, and leave the discussion on probit function for interested readers. "Probit Model" page on Wikipedia is a good place to start if you are interested to learn more.[9]

5.7.1 Exercises

(1) Use the MCMC sample obtained from running **`model5-3.stan`**, and write R or Python codes that computes the MCMC sample of $\varepsilon[n] = Y[n] - \mu[n]$ for each n.
(2) Modify **`model5-3.stan`** to generate $\varepsilon[n]$ (same as above) inside the Stan code.
(3) Plot PPCs for **`model5-4.stan`** and **`model5-6b.stan`**. Recall the discussion in Sect. 5.2.1 and assess if there are any issues with the model.
(4) Modify **`model5-4.stan`** and **`model5-6b.stan`** to use `matrix` type. Refer to Sect. 5.6.

[9] https://en.wikipedia.org/wiki/Probit_model.

Part III
Essential Technics for Mastering Statistical Modeling

Chapter 6
Introduction of Probability Distributions

In Fig. 1.2, we mentioned the slider crank mechanism, which is a basic mechanism in mechanical engineering. Without knowing this mechanism, there is no way that we can come up with Fig. 1.2. The same is true for statistical modeling, where each probability distribution corresponds to a basic mechanism that plays a role as a building block to model the complex data. Without knowing them, it would be difficult to think of how to specify the whole model. Therefore, this chapter introduces several probability distributions that are commonly used in statistical modeling.

In this chapter, we explain the basic properties of these fundamental distributions, and further provide some examples to explain how to use them in practice, and other information that should be taken into account when building a model with them. These distributions are widely used in practice for a long time. Some of them can be derived theoretically from physical laws, but in most cases, the statement that data follows a certain probability distribution is nothing more than an assumption that is based on our observation and domain knowledge on the data.

As the example of slider crank mechanism, designing a machine that can accomplish complex motion requires an appropriate combination of basic mechanisms. Similarly, building a complex model requires multiple probability distributions to be properly combined. The proper combination includes functions that link variables (e.g., a logistic function mentioned in Sect. 5.3.3), nonlinear models mentioned in Sect. 7.2, and hierarchical models mentioned in Chap. 8.

© Springer Nature Singapore Pte Ltd. 2022

K. Matsuura, *Bayesian Statistical Modeling with Stan, R, and Python*,
https://doi.org/10.1007/978-981-19-4755-1_6

6.1 Notations

In this chapter we introduce the probability distribution using the tabular form below:

PMF or PDF[a]	Functional form of $p(y\|\theta)$
Figure	An illustration of the functional form
Support	y: possible values of y
Parameter	θ: possible values of θ
Mean	Theoretical mean of $p(y\|\theta)$
SD (or Variance)	Theoretical SD (or variance) of $p(y\|\theta)$
Example	Sections it appears in this book

[a]PMF stands for probability mass function, and PDF stands for probability density function

where y is a random variable, θ is the parameter that determines the shape and position of a distribution, and $p(y|\theta)$ is the distribution that y follows.

In general, statistics books use variance instead of standard deviation (SD) because it is mathematically easier to handle. In this table, SD is used because mean and SD can be easily compared in the same unit. As we can write them in the form like "mean $\pm$ SD", focusing on SD instead of variance is sometime straightforward for building models. In addition, in Stan, a normal distribution directly takes mean and SD as arguments, as coded as `normal(mu, sigma)`. That being said, for multivariate distributions, in which case it is common to calculate a covariance matrix, it could be simplified using variance rather than SD.

The function $\Gamma(x)$ is often found in the function form of $(y|\theta)$ for normalization. The definition of $\Gamma(x)$ is as follows:

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$$

6.2 Uniform Distribution

PDF	$\text{Uniform}(y\|a, b) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq y \leq b \\ 0 & \text{others} \end{cases}$
Figure	See (Fig. 6.1)
Support	y: a real number in the range of $[a, b]$
Parameter	a, b: real numbers that satisfy $a < b$
Mean	$(a + b)/2$
SD	$(b - a)/\sqrt{12}$
Example	Used in most of the noninformative prior

Usage Example

This distribution is used as a noninformative prior distribution. In Stan, when a prior distribution of a parameter is not set explicitly, a uniform distribution with a given range is automatically set as its prior distribution. For example, when a parameter with the range of [0, 1] is declared without any prior distribution, Uniform(0, 1) is used. When no range is defined, a sufficiently wide uniform distribution with the range of $(-\infty, +\infty)$ is used.

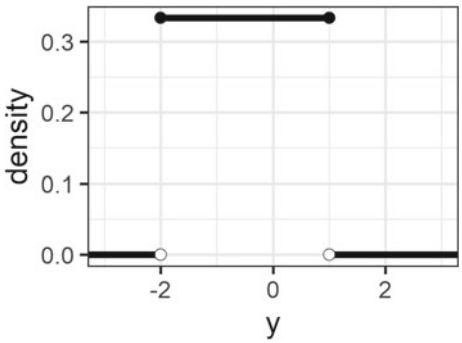


Fig. 6.1 A uniform distribution with $a = -2$, $b = 1$

6.3 Bernoulli Distribution

PMF	$\text{Bernoulli}(y\|\theta) = \begin{cases} \theta & \text{if } y = 1 \\ 1-\theta & \text{if } y = 0 \end{cases}$
Figure	See (Fig. 6.2)
Support	y: an integer of 0 or 1
Parameter	θ: a real number in the range of [0, 1]
Mean	θ
SD	$\sqrt{\theta(1-\theta)}$
Example	Sections 5.4, 8.5, 10.2, 10.3, 11.3.2, and 14.3.2

This distribution can be interpreted as follows: the probability of occurrence of 1 is θ, and that of 0 is $1 - \theta$.

Data Example

This distribution is used for binary data, such as an outcome of a coin flip, customer's decision of whether to buy a product or not, and determining whether a drug has effect or not.

Usage Example

As described in Sect. 5.4, because the parameter θ is a real number in the range [0, 1], this distribution is often used in combination with a logistic function (`inv_logit`).

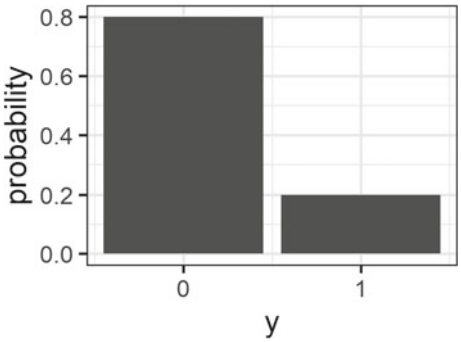


Fig. 6.2 A Bernoulli distribution with $\theta = 0.2$

6.4 Binomial Distribution

PMF	$\text{Binomial}(y\|N,\theta) = \frac{N!}{y!(N-y)!}\theta^y(1-\theta)^{N-y}$
Figure	See (Fig. 6.3)
Support	y: an integer from 0 to N
Parameter	N: a positive integer θ: a real number in the range of [0, 1]
Mean	$N\theta$
SD	$\sqrt{N\theta(1-\theta)}$
Example	Sections 5.3, 10.3, 11.3.2, and 13.1

This distribution can be interpreted as follows: in N times of coin flips, the probability that the number of heads is y out of all trials N. And, an outcome in each trial follows the Bernoulli distribution with the same parameter θ. That is:

if $y_1 \sim \text{Bernoulli}(\theta)$, …, and $y_N \sim \text{Bernoulli}(\theta)$ are independent,
then $y = y_1 + \cdots + y_N \sim \text{Binomial}(N, \theta)$.

A Bernoulli distribution is a special case of a binomial distribution: a binomial distribution with $N = 1$ is equivalent to a Bernoulli distribution.

This distribution has reproductive property. That is, if random variables y_1 and y_2 are independent and each of them follows a binomial distribution with the same parameter θ, their summation follows another binomial distribution. The mathematical expression is as follows:

if $y_1 \sim \text{Binomial}(N_1, \theta)$ and $y_2 \sim \text{Binomial}(N_2, \theta)$ are independent,
then a new random variable $y = y_1 + y_2 \sim \text{Binomial}(N_1 + N_2, \theta)$.

We can also understand this property using the example of coin flips. It should be noted, however, the reproductive property is only applicable when all the independent distributions share identical θ value.

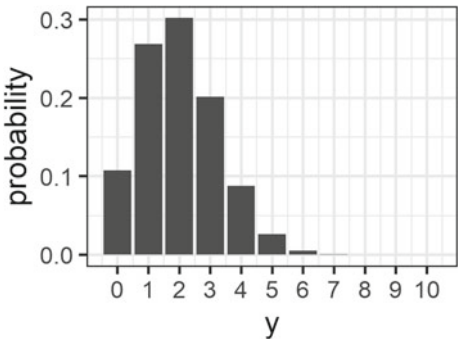


Fig. 6.3 A binomial distribution with $N = 10$, $\theta = 0.2$

If N is sufficiently large and a binomial distribution is nearly bilaterally symmetrical, this distribution can be approximated by a normal distribution with mean of $N\theta$ and SD of $\sqrt{N\theta(1-\theta)}$. As mentioned in Sect. 5.3.3, in probabilistic programming languages, it is generally faster to use a normal distribution than a combination of a logistic function and a binomial distribution. Therefore, using a normal distribution can be a better alternative for a binomial distribution when N and y are sufficiently large.

Data Example

Number of wins out of total 10 games, number of girls among children in a family.

Usage Example

See Sect. 5.3. As in the case of a Bernoulli distribution, θ is often expressed using explanatory variables and a logistic function (`inv_logit`). N is often given or known.

6.5 Beta Distribution

PDF	$\mathrm{Beta}(y\|\alpha, \beta) = \frac{1}{B(\alpha,\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}$
Figure	See (Fig. 6.4)
Support	θ: a real number in the range (0, 1)
Parameter	α, β: positive real numbers
Mean	$\alpha/(\alpha+\beta)$
SD	$\frac{\sqrt{\alpha\beta}}{(\alpha+\beta)\sqrt{\alpha+\beta+1}}$
Example	Section 9.2.3

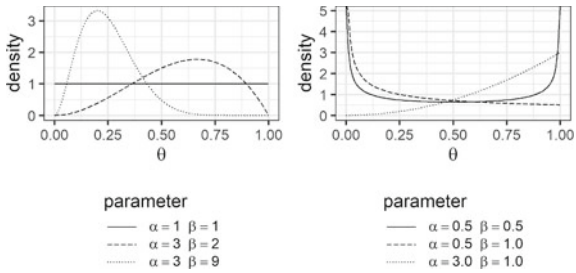


Fig. 6.4 Beta distribution

$B(\alpha, \beta)$ is the normalization constant for setting the integral of this distribution to 1, which is defined as follows:

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha + \beta)}$$

From the expression of mean $\alpha/(\alpha + \beta)$, with a fixed β, the mean of beta distribution approaches to zero with decreasing α, and with a fixed α, the mean approaches to 1 with decreasing β.

This distribution can also be interpreted as a distribution that generates a probability θ because it is a continuous distribution whose support is (0, 1). Of course, it can also be used for other variables in the range of (0, 1). For example, it can be used as a distribution that generates a weight parameter for calculating an interior division point between two points.

Data Example

Suppose that there are multiple products classified in N categories. We can consider a beta distribution for the failure probabilities of the products in each category.

Usage Example

This distribution is often used to generate the parameter θ for a Bernoulli distribution or a binomial distribution.

For example, in the "Data Example" above, suppose that there are $M[n]$ products ($n = 1, \ldots, N$) for each category n, and we detected $Y[n]$ products with failure from the total of $M[n]$ products in the category. Then, we want to estimate the failure probability $\theta[n]$ for each category. Although we can estimate $\theta[n]$ for each category, it will be difficult when the corresponding $M[n]$ is small. Alternatively, we can assume that the failure probabilities are generated from a beta distribution. The model formula is as follows:

Model Formula 6.1

$$\theta[n] \sim \text{Beta}(\alpha, \beta) \qquad n = 1, \ldots, N$$
$$Y[n] \sim \text{Binomial}(M[n], \theta[n]) \qquad n = 1, \ldots, N$$

Besides $\theta[n]$, α and β are also estimated from the data. If the amount of the data is small, or when we have background knowledge, a weakly informative prior distribution may be set as a prior distribution of α and β.

If the failure probabilities of all categories are similar, the beta distribution will be a unimodal distribution, with both α and β larger than 1. On the other hand, if the failure probability of each category is either very low or very high, (i.e., $\theta[n]$ is close to 0 or 1), the beta distribution will be a U-shape, with both α and β smaller than 1. This property allows a beta distribution to represent the distribution of variables in the range of (0, 1) with high flexibility.

In the coin flips example, we can assume that θ is generated from a beta distribution with parameters α and β. When α and β are close and both are greater than 1, this distribution will tend to produce fair coins, whereas when both α and β are less than 1, it will tend to produce a distorted coin. If $\alpha = \beta = 1$, this distribution is equivalent to a uniform distribution on [0, 1], which indicates that there is no information about the fairness of the coin.

6.6 Categorical Distribution

PMF	Categorical$(y = k \mid \boldsymbol{\theta}) = \theta_k$
Figure	See (Fig. 6.5)
Support	y: an integer from 1 to K (it does not include 0)
Parameter	K: a positive integer that satisfies $K \geq 2$ $\vec{\theta}$: K-simplex. That is, a vector of length K where each element is a real number in the range of [0, 1] and the sum of all elements equals to 1
Mean	Mean of $I[y = k]$: θ_k [a]
Variance-covariance	Variance of $I[y = k]$: $\theta_k(1 - \theta_k)$ Covariance between $I[y = k]$ and $I\left[y = k'\right]$ $\left(k \neq k'\right)$: $-\theta_k \theta_{k'}$
Example	Sections 9.1.4, and 11.3.2

[a] $I[y = k]$ is a function that returns 1 if $y = k$, and 0 otherwise

We can interpret that this distribution probabilistically generates category indices. The total number of categories is K, and the occurrence probability of category k is θ_k. In other words, we can also interpret that this distribution is a K -sided dice such that the probability of rolling k is θ_k.

This distribution may look like a multivariate version of a Bernoulli distribution. However, note that a categorical distribution with the number of categories $K = 2$ is not identical to a Bernoulli distribution. This is because y generated from a Bernoulli

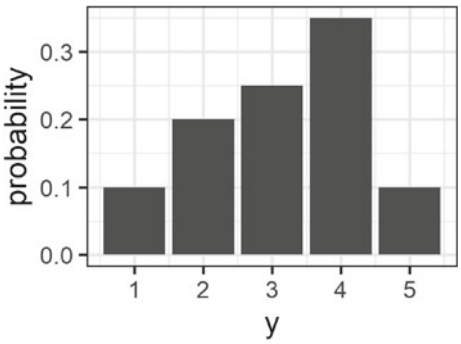


Fig. 6.5 A categorical distribution with $K = 5$, $\vec{\theta} = (0.1 \quad 0.2 \quad 0.25 \quad 0.35 \quad 0.1)^T$

distribution is not an index of categories but a binary (0 or 1) value representing whether an event occurred or not.

Usage Example

Let us consider an example of using this distribution with some explanatory variables to get a better understanding of a categorical distribution. Suppose we have N customers' purchase data $Y[n]$ ($n = 1, \ldots, N$), where each Y is a category index of the purchased product (an integer from 1 to K), and explanatory variables including $Age[n]$, $Sex[n]$, and $Income[n]$ (annual income). The vector $\overrightarrow{\theta[n]}$ represents the probabilities that the customer n chooses a product from each category. We assume that $Y[n]$ is generated from a categorical distribution with parameter $\overrightarrow{\theta[n]}$, where each of $\overrightarrow{\theta[1]}, \ldots, \overrightarrow{\theta[N]}$ is a K-simplex. Assuming that $\overrightarrow{\theta[n]}$ is determined from the linear combination of the above explanatory variables, the model can be specified as:

Model Formula 6.2

$$\overrightarrow{\mu[n]} = \overrightarrow{b_1} + \overrightarrow{b_2}\,Age[n] + \overrightarrow{b_3}\,Sex[n] + \overrightarrow{b_4}\,Income[n] \qquad n = 1, \ldots, N$$

$$\overrightarrow{\theta[n]} = \mathrm{softmax}\left(\overrightarrow{\mu[n]}\right) \qquad n = 1, \ldots, N$$

$$Y[n] \sim \mathrm{Categorical}\left(\overrightarrow{\theta[n]}\right) \qquad n = 1, \ldots, N$$

where $\overrightarrow{\mu[n]}$ represents a linear combination of the explanatory variables. It is a vector of length K and each element expresses the "intensity" for choosing a product in each category. Because the influence of explanatory variables differs from category to category, each of regression coefficients ($\overrightarrow{b_1}$, $\overrightarrow{b_2}$, $\overrightarrow{b_3}$, and $\overrightarrow{b_4}$) is a vector of length K, respectively. These regression coefficients are estimated from the data. Such a model is called *multinomial logistic regression*, which is discussed in Sect. 9.1.4.

The softmax in Model Formula 6.2 is defined as follows:

$$\vec{\theta} = \mathrm{softmax}(\vec{x}) = \left(\frac{\exp(x_1)}{\sum_{k'=1}^{K} \exp(x_{k'})} \cdots \frac{\exp(x_K)}{\sum_{k'=1}^{K} \exp(x_{k'})} \right)^T$$

This function converts all elements of $\overrightarrow{x}$ in the range of $(-\infty, \infty)$ into positive values, and then normalizes them so that the sum of these positive values equals to 1. In other words, this function is one of the functions that converts a vector $\overrightarrow{x}$ of length K into K-simplex $\overrightarrow{\theta}$. Therefore, this distribution is often used in combination with the softmax function.

6.7 Multinomial Distribution

PMF	$\text{Multinomial}\left(\vec{y}\mid N, \vec{\theta}\right) = \frac{N!}{\prod_{k=1}^{K} y_k!} \prod_{k=1}^{K} \theta_k^{y_k}$
Figure	(skip)
Support	$\overrightarrow{y}$: a vector of length K where each element is an integer from 0 to N, and sum of all elements is N
Parameter	K: a positive integer that satisfies $K \geq 2$ N: a positive integer $\overrightarrow{\theta}$: K-simplex
Mean	Mean of y_k:$N\theta_k$
Variance-covariance	Variance of y_k: $N\theta_k(1-\theta_k)$ Covariance between y_k and $y_{k'}$ $(k \neq k')$:$-N\theta_k\theta_{k'}$
Example	Sections 12.3, and 14.2.1

This distribution is a joint distribution of $y_1, \ldots, y_K$ (each element of $\overrightarrow{y}$). We can interpret that this distribution generates the number of occurrences of each side $y_1, \ldots, y_K$ when rolling a K-sided dice N times, when we consider this dice follows a categorical distribution with the parameter $\overrightarrow{\theta}$.

Note that this distribution with the number of trials $N = 1$ is not identical to a categorical distribution. The difference is that y in a categorical distribution is not a vector of the number of counts but is an index of a category.

This distribution seems like a multivariate version of a binomial distribution. However, note that this distribution with the number of categories $K = 2$ is not identical to a binomial distribution. This is because this distribution generates y_1 and y_2, which are the numbers of counts in each category, whereas a binomial distribution generates only y, which is the number of occurrences out of N trials.

Visualizing this distribution is not easy with $K \geq 3$ categories, because the number of dimensions and axis would increase to K.

Data Example

Count data of how many times each category appears in 1,000 lottery tickets.

Usage Example

As in the case of a categorical distribution, $\overrightarrow{\theta}$ is often expressed using explanatory variables and the softmax function. N is often given or known.

6.8 Dirichlet Distribution

PDF	$\text{Dirichlet}\left(\vec{\theta}\middle\vert\vec{\alpha}\right) = \frac{1}{B(\vec{\alpha})}\prod_{k=1}^{K}\theta_k^{\alpha_k-1}$
Figure	See (Fig. 6.6)
Support	$\vec{\theta}$: K-simplex
Parameter	K: a positive integer that satisfies $K \geq 2$ $\vec{\alpha}$: a vector of length K where each element is a positive real number
Mean	Mean of θ_k: α_k/α_{sum} where $\alpha_{sum} = \sum_{k=1}^{K}\alpha_k$
Variance-covariance	Variance of θ_k: $\alpha_k(\alpha_{sum} - \alpha_k)/\left({\alpha_{sum}}^2(\alpha_{sum} + 1)\right)$ Covariance between θ_k and $\theta_{k'}(k \neq k')$:$-\alpha_k\alpha_{k'}/\left({\alpha_{sum}}^2(\alpha_{sum} + 1)\right)$
Example	Sections 9.2.3, and 14.2.1

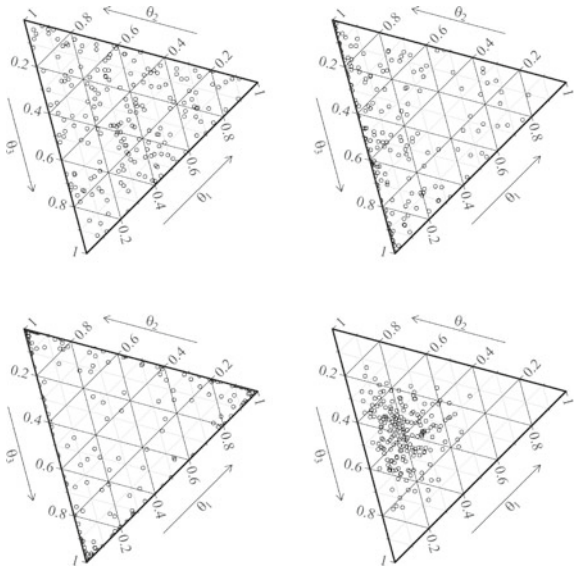


Fig. 6.6 We plotted 200 draws of $\vec{\theta} = (\theta_1 \quad \theta_2 \quad \theta_3)^T$ from a Dirichlet distribution with $K = 3$. The parameter $\vec{\alpha}$ in each plot is: top left: $(1 \quad 1 \quad 1)^T$, top right: $(0.3 \quad 1 \quad 1)^T$, bottom left: $(0.3 \quad 0.3 \quad 0.3)^T$, bottom right: $(3 \quad 6 \quad 6)^T$

This is a joint distribution of $\theta_1, \ldots, \theta_K$ (each element of vector $\vec{\theta}$). $B(\vec{\alpha})$ is the normalization constant that ensures the distribution has the integral equal to 1, which is defined as follows:

$$B(\vec{\alpha}) = \frac{\prod_{k=1}^{K} \Gamma(\alpha_k)}{\Gamma\left(\sum_{k=1}^{K} \alpha_k\right)}$$

This distribution can be interpreted as a distribution that generates K-simplex $\overrightarrow{\theta}$. This distribution may look like a multivariate version of a beta distribution. However, note that a Dirichlet distribution with the number of categories $K = 2$ is not identical to a beta distribution. This is because y in a beta distribution is not a vector of the number of categories but a single value of θ_1. As in the case of a beta distribution, you can use this distribution not only for a probability but also for a weight parameter for calculating an interior division point among 3 or more points.

For the same reason as the multinomial distribution, it is not straightforward to visualize this distribution with $K \geq 3$. But we can drop one dimension because we know that all the elements of $\overrightarrow{\theta}$ sum up to 1. In Fig. 6.6, we drew random numbers from the two-dimensional probability density function, which is expressed by dropping one dimension for the case of $K = 3$, and plotted them in the two-dimensional space.

Usage Example

This distribution is typically used to generate a simplex. For example, it is used for the parameter $\overrightarrow{\theta}$ of a categorical or multinomial distribution:

Model Formula 6.3

$$\vec{\theta} \sim \text{Dirichlet}(\vec{\alpha})$$
$$Y \sim \text{Categorical}\left(\vec{\theta}\right)$$

Here, $\overrightarrow{\theta}$ is usually estimated from data, and $\overrightarrow{\alpha}$ is either given a fixed value or can also be estimated from data.

If we link this distribution to the example of rolling a dice, a Dirichlet distribution determines what shapes of dice tend to be generated. If $\alpha_1, \ldots, \alpha_K$ (elements of $\overrightarrow{\alpha}$) are greater than 1 and are roughly the same value, this distribution tends to generate a fair dice, whereas if $\alpha_1, \ldots, \alpha_K$ are less than 1, it tends to generate a distorted dice, only a few sides of which are likely to appear. If $\alpha_1 = \alpha_2 = \cdots = \alpha_K = 1$, this distribution equals to the uniform distribution in the space of K-simplex, which means we have no information on the shape of a dice.

6.9 Exponential Distribution

PDF	Exponential$(y\|\beta) = \beta\exp(-\beta y)$
Figure	See (Fig. 6.7)
Support	y: a nonnegative real number
Parameter	β: a positive real number, called rate parameter
Mean	$1/\beta$
SD	$1/\beta$
Example	Section 9.2.2

This distribution is the only one continuous probability distribution that has memorylessness property. The memorylessness property states that the following expression is true for any s and t:

$$Pr(y > s + t | y > s) = Pr(y > t)$$

For example, we assume that the time for a wine glass to break after the first usage follows an exponential distribution. Also, suppose that there is a wine glass that did not break after 3 years since the first use. The memorylessness means that the probability that this wine glass will not break in the next year is the same as the probability that another new wine glass will not break in the first year of usage.

The tail of this distribution is heavier than that of a normal distribution because the exponents of the exp function in the PDF is $-y$.

Also, as β decreases, the mean and standard deviation increase at the same speed.

Data Example

Data of time to an event. The event includes death, failure, and observation of a shooting star.

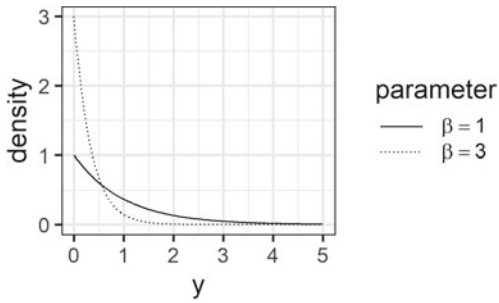


Fig. 6.7 Exponential distribution

Usage Example

In addition to the examples mentioned above, this distribution can be used to generate positive real numbers, such as parameters of a gamma distribution (see Sect. 9.2.2).

6.10 Poisson Distribution

PDF	$\text{Poisson}(y\|\lambda) = \frac{1}{y!}\lambda^y \exp(-\lambda)$
Figure	See (Fig. 6.8)
Support	y: a nonnegative integer
Parameter	λ: a positive real number
Mean	λ
SD	$\sqrt{\lambda}$
Example	Sections 5.5, 10.1, 11.3.3, 12.2, 12.8, and 14.2.1

As the mean of this distribution equals to its parameter λ, "a Poisson distribution with mean λ" is sometimes used to represent "a Poisson distribution with parameter λ".

This distribution has reproductive property:

if $y_1 \sim \text{Poisson}(\lambda_1)$ and $y_2 \sim \text{Poisson}(\lambda_2)$ are independent,
then a new random variable $y = y_1 + y_2 \sim \text{Poisson}(\lambda_1 + \lambda_2)$.

There is a close relationship between a Poisson distribution and an exponential distribution. Suppose several events occur independently with certain time intervals, and these intervals follow an exponential distribution with parameter β (mean $1/\beta$). Then, the number of the events occurring per unit time follows a Poisson distribution with mean β. Also, the number of the events occurring during the period of 5

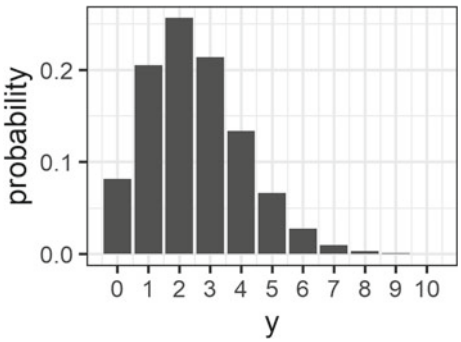


Fig. 6.8 Poisson distribution with $\lambda = 2.5$

time units follows a Poisson distribution with mean 5β because of the reproductive property.

There is also a close relationship between a Poisson distribution and a binomial distribution. If N is sufficiently large and θ is sufficiently small in a Binomial$(y|N, \theta)$, the binomial distribution can be approximated by a Poisson distribution.

For example, the numbers of deaths in traffic accidents in a day can be considered to follow a Poisson distribution because many people have a non-zero probability to die in a traffic accident every day (N is large) and the probability of actually having a traffic accident that causes a death is very low (θ is small).

The tail of this distribution is light because $y!$ grows much faster than λ^y when y is large.

If λ is sufficiently large, the Poisson distribution can be approximated by a normal distribution with mean λ and SD $\sqrt{\lambda}$. As mentioned in Sect. 5.5.1, in probabilistic programming languages, it is generally faster to use a normal distribution than a combination of an exponential function and a Poisson distribution. Therefore, using a normal distribution can be a better alternative for a Poisson distribution when y is sufficiently large.

If $y_1 \sim \text{Poisson}(\lambda_1)$ and $y_2 \sim \text{Poisson}(\lambda_2)$ are independent, and the sum $y_1 + y_2 = N$ is given, then the following is true:

$$y_1 \sim \text{Binomial}\left(N, \frac{\lambda_1}{\lambda_1 + \lambda_2}\right)$$

More generally, if $y_1 \sim \text{Poisson}(\lambda_1)$, …, $y_K \sim \text{Poisson}(\lambda_K)$ are independent, and the sum $y_1 + \cdots + y_K = N$ is given, then the following is true:

$$\vec{y} \sim \text{Multinomial}\left(N, \left(\frac{\lambda_1}{\sum_{k=1}^{K} \lambda_k} \cdots \frac{\lambda_K}{\sum_{k=1}^{K} \lambda_k}\right)^T\right)$$

Data Example

The number of animals observed per day, the number of mails received per hour. It should be noted that time intervals between these observed events follow exponential distributions (as we discussed earlier).

Another example is the number of sunfish that grow to adults from eggs, where we have a binomial distribution with large N and small θ behind this phenomenon, thus Poisson distribution can be used for approximation.

Usage Example

As described in Sect. 5.5, because the parameter λ is a positive real number, this distribution is often used in combination with an exponential function (`exp`).

6.11 Gamma Distribution

PDF	$\text{Gamma}(y\|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} y^{\alpha-1} \exp(-\beta y)$
Figure	See (Fig. 6.9)
Support	y: a positive real number
Parameter	α, β: positive real numbers. α is called a shape parameter and β is called a rate parameter
Mean	α/β
SD	$\sqrt{\alpha}/\beta$
Example	Sections 9.2.2, 10.2, and 11.7

$\beta^\alpha / \Gamma(\alpha)$ in the PDF of this distribution is the normalization constant for ensuring that the integral of the distribution is 1.

This distribution has reproductive property:

if $y_1 \sim \text{Gamma}(\alpha_1, \beta)$ and $y_2 \sim \text{Gamma}(\alpha_2, \beta)$ are independent,
then a new random variable $y = y_1 + y_2 \sim \text{Gamma}(\alpha_1 + \alpha_2, \beta)$.

This property is only applicable in the case when these gamma distributions share the same β (rate parameter). Otherwise, the new variable y does not follow a gamma distribution.

This distribution also has scaling property:

if $y \sim \text{Gamma}(\alpha, \beta)$,
then, for any $c > 0$, a new random variable $cy \sim \text{Gamma}(\alpha, \beta/c)$.

There is a close relationship between a gamma distribution and an exponential distribution:

if a positive integer α is given,
and $t_1 \sim \text{Exponential}(\beta), \ldots, t_\alpha \sim \text{Exponential}(\beta)$ are independent,
then, a new random variable $y = t_1 + \cdots + t_\alpha \sim \text{Gamma}(\alpha, \beta)$.

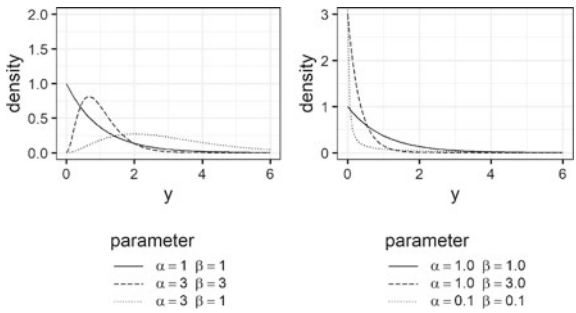


Fig. 6.9 Gamma distribution

Because the mean of each $t_1, \ldots, t_\alpha$ is $1/\beta$, the mean of y is the sum of $1/\beta$, i.e., $1/\beta \times \alpha = \alpha/\beta$. This fact gives us an intuition of why the mean of Gamma(α, β) is α/β. In addition, it is easy to see from the relationship that we can obtain Gamma$(1, \beta) =$ Exponential(β) by setting $\alpha = 1$.

As mentioned above, because this distribution has scaling property and the relationship with an exponential distribution, we often give a fixed value to either α or β, and then estimate the other one.

There is also a close relationship between a gamma distribution and a Poisson distribution. For example, consider the breaking wine glass example we discussed earlier in the previous section. We set the unit time to be a year, and assume that the time for wine glasses to break down after the first usage follows an exponential distribution with parameter $\beta = 0.5$ (with the mean of 2 years). Suppose that we buy 3 of those wine glasses and then use them one by one until all of them breaks. From the above discussion on the relationship of a gamma distribution and an exponential distribution, the time for all 3 wine glasses to break follows Gamma$(3, 0.5)$. Therefore, the probability that the total time for all 3 glasses to break exceeds 5 years is Prob[Gamma$(3, 0.5) > 5$]. We can also express this probability as the probability of breaking two or fewer wine glasses within 5 years. The relationship between a Poisson distribution and an exponential distribution leads to the fact that the number of wine glasses broken per year follows Poisson(0.5). From the reproductive property of the Poisson distribution, the probability that two or fewer wine glasses break within 5 years is expressed as Prob[Poisson$(5 \times 0.5) \leq 2$]. As a result, the following is true:

$$\text{Prob}[\text{Gamma}(3, 0.5) > 5] = \text{Prob}[\text{Poisson}(5 \times 0.5) \leq 2]$$

The tail of this distribution is heavier than that of a normal distribution because the exponent of exp function in the PDF is $-y$.

If α is sufficiently large, the gamma distribution can be approximated by a normal distribution with mean α/β and SD $\sqrt{\alpha}/\beta$. As in the case of a binomial distribution and a Poisson distribution, using a normal distribution can be a better alternative for a gamma distribution when y is sufficiently large.

If $y_1 \sim$ Gamma(α_1, β) and $y_2 \sim$ Gamma(α_2, β) are independent, then the following is true:

$$\frac{y_1}{y_1 + y_2} \sim \text{Beta}(\alpha_1, \alpha_2)$$

More generally, if $y_1 \sim$ Gamma(α_1, β), ..., $y_K \sim$ Gamma(α_K, β) are independent, then the following is true:

$$\left(\frac{y_1}{\sum_{k=1}^{K} y_k} \cdots \frac{y_K}{\sum_{k=1}^{K} y_k} \right)^T \sim \text{Dirichlet}(\vec{\alpha})$$

Data Example

This distribution is often used to model waiting time. For instance, waiting time for observing five shooting stars, and the time before the PC memory disk to break.

Usage Example

In addition to the above data, this distribution may be used to generate positive real numbers such as parameters of a beta distribution or a gamma distribution (see Sect. 9.2.2).

6.12 Normal Distribution

PDF	$\mathcal{N}(y\|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2\right)$
Figure	See (Fig. 6.10)
Support	y: a real number
Parameter	μ: a real number σ: a positive real number
Mean	μ
SD	σ
Example	Many sections

A normal distribution is often described as "a normal distribution with mean μ and SD σ" because "parameter μ = mean and parameter σ = SD". In this book, we write a normal distribution as $\mathcal{N}$(mean, SD) or Normal(mean, SD), in order to be consistent with the `normal` function in Stan.

This distribution has reproductive property:

if $y_1 \sim \mathcal{N}(\mu_1, \sigma_1)$ and $y_2 \sim \mathcal{N}(\mu_2, \sigma_2)$ are independent,

then a new random variable $y = y_1 + y_2 \sim \mathcal{N}\left(\mu_1 + \mu_2, \sqrt{\sigma_1^2 + \sigma_2^2}\right)$.[1]

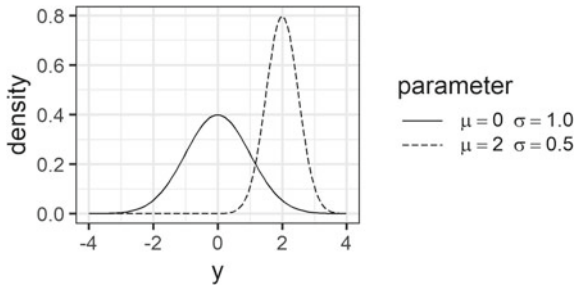


Fig. 6.10 Normal distribution

[1] If we express a normal distribution as $\mathcal{N}$(mean, variance), reproductive property is written as: if $y_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $y_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ are independent, then $y = y_1 + y_2 \sim \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$

For example, suppose that we have a number of children whose heights follow a normal distribution with mean = 100 cm and SD = 12 cm, and a number of bamboo shoots whose heights follow a normal distribution with mean = 30 cm and SD = 5 cm. Next, suppose that we repeat the following measurements: we take out one child and one bamboo shoot at random from each set, and then place the bamboo shoot on the child's head and measure the total height. Reproductive property means that the new distribution of the total heights follows another normal distribution with mean = 130 cm and SD = 13 cm. This reproductive property holds true even if μ_2 is negative.

If $\xi \sim \mathcal{N}(0, 1)$, then $y = \mu + \xi\sigma$ follows a normal distribution with the mean μ and SD σ, that is, $y \sim \mathcal{N}(\mu, \sigma)$. This property is often used in reparameterization (see Sect. 9.3).

A normal distribution is used for many of the data encountered in the real world. Sometimes fitting a normal distribution to such data is supported by physical laws or central limit theorem. But in many other cases, we assume a normal distribution for data generated from uncertain mechanisms, such as human height. Also, it is common to assume a normal distribution to the variables that are not necessarily observed data—such as latent variables, group differences, individual differences, and time dependent changes.

This distribution has the support of $(-\infty, \infty)$ and left–right symmetricity. Also, the tail of this distribution is relatively light because the exponent of exp function in its PDF contains $-y^2$; the value of the PDF sharply decreases as a distance from μ exceeds about $\pm 2\sigma$. Therefore, sometimes an outlier can induce a shift in the parameter estimation result (see Sect. 7.8). To avoid the shift in the result, a Cauchy distribution or a Student-t distribution can be used instead.

We can obtain another distribution by truncating and normalizing the $y \geq 0$ part of $\mathcal{N}(0, \sigma)$. This new distribution is called a half-normal distribution, which can be written as $\mathcal{N}^+(0, \sigma)$.

Data Example

The weights of the bread at a bakery. Written test scores.

Usage Example

See Sect. 5.1. When a normal distribution is assumed for group differences or individual differences, such a model is called a hierarchical model, which is discussed in Chap. 8. A half-normal distribution is often used as a weakly informative prior distribution for parameters that take positive real numbers such as SD (see Sect. 9.2.2).

6.13 Lognormal Distribution

PDF	$\text{LogNormal}(y\|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma}\frac{1}{y}\exp\left(-\frac{1}{2}\left(\frac{\log y-\mu}{\sigma}\right)^2\right)$
Figure	See (Fig. 6.11)
Support	y: a positive real number
Parameter	μ: a real number σ: a positive real number
Mean	$\exp(\mu+\sigma^2/2)$
SD	$\exp(\mu+\sigma^2/2)\sqrt{\exp(\sigma^2)-1}$
Example	Sections 9.2.2, and 14.4.1

The name of the lognormal distribution is derived from the fact that if a random variable y follows $\text{LogNormal}(\mu,\sigma)$, then a new random variable $x = \log y$ follows $\mathcal{N}(\mu,\sigma)$. Conversely, if a random variable x follows a normal distribution, then a new random variable $y = \exp\ x$ follows a lognormal distribution.

Since a normal distribution has reproductive property, the following is true for a lognormal distribution:

if $y_1 \sim \text{LogNormal}(\mu_1,\sigma_1)$ and $y_2 \sim \text{LogNormal}(\mu_2,\sigma_2)$ are independent, then a new random variable generated by the product $y = y_1 y_2 \sim \text{LogNormal}\left(\mu_1+\mu_2, \sqrt{\sigma_1^2+\sigma_2^2}\right)$.

The tail of this distribution is considerably heavy because the exponent of exp function in its PDF contains $-(\log y)^2$. Its tail is heavier than an exponential and a gamma distribution, and is lighter than a Cauchy distribution..[2]

Data Example

Human weight, annual income.

Usage Example

In addition to the above data example, this distribution may be used to generate positive real numbers such as SD (see Sect. 9.2.2).

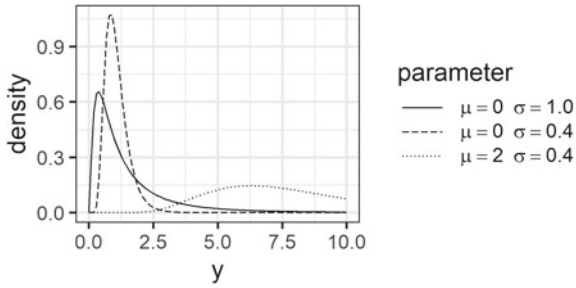


Fig. 6.11 LogNormal distribution

[2] We can compare the heaviness of the tails by taking limit (as $y \to \infty$) of log(PDF of an exponential distribution/ PDF of a lognormal distribution).

6.14 Multivariate Normal Distribution

PDF	$\text{MultiNormal}(\vec{y}\|\vec{\mu}, \mathbf{\Sigma}) = \frac{1}{(2\pi)^{\frac{K}{2}}} \frac{1}{\sqrt{\|\Sigma\|}} \exp\left(-\frac{1}{2}(\vec{y}-\vec{\mu})^T \Sigma^{-1} (\vec{y}-\vec{\mu})\right)$
Figure	See (Fig. 6.12)
Support	$\overrightarrow{y}$: a vector of length K where each element is a real number
Parameter	K: a positive integer $\overrightarrow{\mu}$: a vector of length K where each element is a real number $\mathbf{\Sigma}$: a symmetric positive definite[a] $K \times K$ matrix, which is called variance–covariance matrix, or covariance matrix for simplicity
Mean	Mean of y_k:μ_k
Variance-covariance	Variance of y_k: $\mathbf{\Sigma}_{k,k}$ covariance between y_k and $y_{k'}$ $(k \neq k')$:$\mathbf{\Sigma}_{k,k'}$
Example	Sections 9.2.4, 9.3.3, 11.3.4, 11.6, 12.7, 13.4.2, and 14.2.2

[a]A necessary and sufficient condition of a positive definite matrix is that all eigenvalues are positive real numbers

This distribution is a joint distribution that $y_1, \ldots, y_K$ (each element of $\overrightarrow{y}$) follow. $|\Sigma|$ is the determinant of Σ. It is often called as "a multivariate normal distribution with mean vector $\overrightarrow{\mu}$ and covariance matrix Σ" because "parameter $\overrightarrow{\mu}$ = mean and parameter Σ = covariance matrix".

If the off-diagonal components of Σ are 0, then each of $y_1, \ldots, y_K$ independently follows a one-dimensional normal distribution. The off-diagonal components are associated with the correlations between two elements of $\overrightarrow{y}$. For example, suppose that a vector with two elements $\overrightarrow{y} = \begin{pmatrix} a & b \end{pmatrix}^T$ follows a bivariate normal distribution. Let μ_a and σ_a be the mean and SD of a, μ_b and σ_b be the mean and SD of b, and the correlation between a and b be ρ. Then, $\overrightarrow{\mu}$ and Σ are expressed as follows:

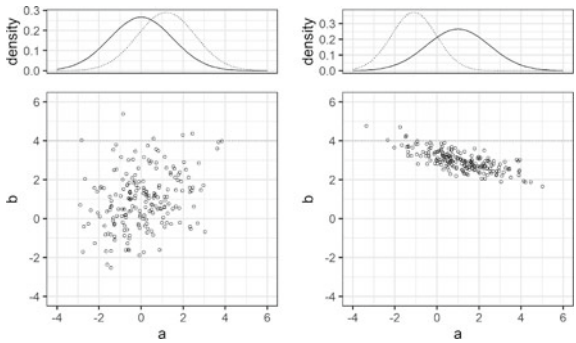


Fig. 6.12 We plotted 200 draws of $\overrightarrow{y} = (a \quad b)^T$ 200 from a MultiNormal distribution with $K = 2$. The parameters in each plot are: left: $\mu_a = 0$, $\mu_b = 1$, $\sigma_a = 1.5$, $\sigma_b = 1.5$, $\rho = 0.4$, right: $\mu_a = 1$, $\mu_b = 3$, $\sigma_a = 1.5$, $\sigma_b = 0.5$, $\rho = -0.7$. See the text for the relationship between σ_a, σ_b, ρ and Σ. The top figures show the distributions obtained from these parameters: the solid line is the marginal distribution $p(a)$, and the dotted line is the conditional distribution $p(a|b = 4)$

$$\vec{\mu} = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix} \qquad \Sigma = \begin{pmatrix} \sigma_a^2 & \sigma_a\sigma_b\rho \\ \sigma_a\sigma_b\rho & \sigma_b^2 \end{pmatrix}$$

We cannot easily visualize this distribution with $K \geq 3$ categories. Here we plotted 200 draws from the two-dimensional density function with $K = 2$ (Fig. 6.12).

This distribution also has reproductive property:

> if $\vec{y} \sim \text{MultiNormal}(\vec{\mu}, \Sigma)$ and $\vec{y}' \sim \text{MultiNormal}\left(\vec{\mu}', \Sigma'\right)$ are independent, then a new random variable $\vec{y}' = \vec{y} + \vec{y}' \sim \text{MultiNormal}\left(\vec{\mu} + \vec{\mu}', \Sigma + \Sigma'\right)$.

If $\vec{\xi} \sim \text{MultiNormal}\left(\vec{0}, \mathbf{I}\right)$ (i.e., each element of $\vec{\xi}$ follows $\mathcal{N}(0, 1)$), then $\vec{y} = \vec{\mu} + \mathbf{L}\vec{\xi}$ follows a multivariate normal distribution with the mean $\vec{\mu}$ and covariance matrix Σ, that is, $y \sim \text{MultiNormal}(\vec{\mu}, \Sigma)$. Here, $\mathbf{L}$ is a Cholesky factor of the covariance matrix, that is, $\Sigma = \mathbf{L}\mathbf{L}^T$. This property is often used in reparameterization (see Sects. 9.3.3 and 12.7).

Note that if each of random variables follows a multivariate normal distribution, their linear combination still follows a multivariate normal distribution. Also, both the marginalized distribution and conditional distribution of a multivariate normal distribution are a multivariate normal distribution. For example, in the case of the bivariate normal distribution that generates $\vec{y} = \begin{pmatrix} a & b \end{pmatrix}^T$, the marginal distribution $p(a)$ obtained by marginalizing $p(a, b)$ with respect to b, is still a normal distribution:

$$p(a) = \int p(a, b)db = \mathcal{N}(\mu_a, \sigma_a).$$

This is also true for $p(b)$. The conditional distribution $p(a|b = b^*)$ is also the normal distribution:

$$p\left(a|b = b^*\right) = \mathcal{N}\left(\mu_a + \rho(\sigma_a/\sigma_b)\left(b^* - \mu_b\right), \sigma_b\sqrt{1 - \rho^2}\right).$$

And again, this is true for $p(b|a = a^*)$.

Data Example

Suppose that we have physical fitness test data from 9-to-12-years-old students in an elementary school. Let us say we have two variables $Y1$ and $Y2$: $Y1$ represents the time record for 50 m run, and $Y2$ is the distance for long jump. Then we can assume that the vector $\vec{Y} = \begin{pmatrix} Y1 & Y2 \end{pmatrix}^T$ follows a bivariate normal distribution.

Usage Example

In the "Data Example" above, consider that we have a comma-separated file with 32 students (Data File 6.1, a row corresponds to a student). Each column represents:

Y1 : time of the 50-m run (unit: seconds)

Y2 : distance of the long jump (unit: meters).

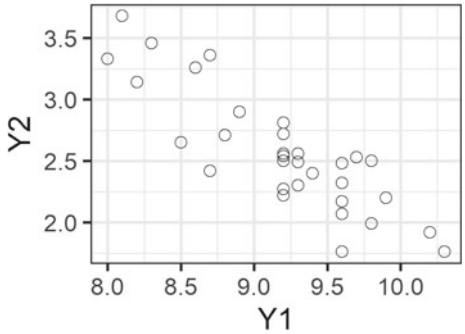


Fig. 6.13 Scatter plot

Data File 6.1 Structure of `data-mvn.csv`

```
 1    Y1,  Y2
 2    9.2,  2.56
 3    9.8,  1.99
 4    9.4,  2.4
      ...
32    8.8,  2.71
```

Fig. 6.13 shows the scatter plot of this data. We can assume that each data $\overrightarrow{Y[n]}$ is the sum of vector $\vec{\mu}$ (population mean) and vector $\overrightarrow{\varepsilon[n]}$ (the other factor). Whatever the cause of this variable $\overrightarrow{\varepsilon[n]}$ is, very likely there is a correlation between the influence on the time of 50 m run and the influence on the distance of long jump. Therefore, considering the distribution of the other factor as a bivariate normal distribution with mean vector $\vec{0}$, we can consider the following model:

Model Formula 6.4

$$\overrightarrow{Y[n]} = \vec{\mu} + \vec{\varepsilon}[n] \qquad n = 1, \ldots, N$$

$$\overrightarrow{\varepsilon[n]} \sim \text{MultiNormal}\left(\vec{0}, \Sigma\right) \qquad n = 1, \ldots, N$$

where N denotes the total number of students, and n represents the index for the students, and $\overrightarrow{Y} = (Y1 \;\; Y2)^T$. By eliminating the term $\overrightarrow{\varepsilon[n]}$, this model can be equivalently represented as below:

Model Formula 6.5

$$\overrightarrow{Y[n]} \sim \text{MultiNormal}(\vec{\mu}, \Sigma) \qquad n = 1, \ldots, N \tag{6.1}$$

The mean vector $\vec{\mu}$ and the covariance matrix Σ are the parameters to be estimated from the data. We used noninformative priors for them. The implementation of Model Formula 6.5 is as follows:

model6-5.stan

```
1   data {
2     int N;
3     int D;
4     array[N] vector[D] Y;
5   }
6
7   parameters {
8     vector[D] mu_vec;
9     cov_matrix[D] cov;
10   }
11
12   model {
13     for (n in 1:N) {
14       Y[n] ~ multi_normal(mu_vec, cov);
15     }
16   }
```

Line 3: The length of the vector generated from a multivariate normal distribution is denoted as `D` (here $D = 2$).

Line 4: The object `Y` is an array that stores `N` vectors, each of length `D`.

Lines 8–9: The parameters for the bivariate normal distribution are declared in these two lines. The mean parameter $\overrightarrow{\mu}$ is declared as a length `D` vector `mu_vec`, and a $D \times D$ covariance matrix $\mathbf{\Sigma}$ is declared as `cov`. A `cov_matrix` type is a special case of a `matrix` type and represents a covariance matrix. If we use `cov_matrix` type, the sampling will be automatically done in the range so that the matrix is symmetric and positive semi-definite.

Lines 13–15: These lines correspond to Eq. (6.1), and when we use the multivariate normal distribution in Stan, we use function `multi_normal`. Note that `Y[n]` on the left-hand-side is a vector of length `D` (see "Multiple indexing and range indexing" section in the Stan reference for details). We can also vectorize these lines into a single-line-code as follows:

```
Y[1:N] ~ multi_normal(mu_vec, cov);
```

The estimated parameters are shown below:

$$\vec{\mu} = \begin{pmatrix} 9.2 \\ 2.6 \end{pmatrix} \quad \Sigma = \begin{bmatrix} 0.42 & -0.30 \\ -0.30 & 0.29 \end{bmatrix}$$

If we use these values, we can learn that in average, the correlation between $Y1$ and $Y2$ is $-\frac{0.30}{(\sqrt{0.42}\sqrt{0.29})} \approx -0.86$.

As such, it is straightforward to implement the model using multivariate normal distributions in Stan. Yet, sometimes it is hard to estimate the mean vector and covariance matrix from noninformative priors. This can be caused by different reasons including having a small amount of data. If this is the case, we can use weakly-informative priors instead (we will further introduce it in Sect. 9.2). In addition, it

should be noted that when the vector generated from the multivariate normal distribution becomes longer, the number of correlations that is estimated will increase quadratically with respect to the length of the vector. We should be aware of this point when we use a large covariance matrix.

This distribution is also widely used in more advanced models such as Gaussian process (see Sect. 12.7).

6.15 Cauchy Distribution

PDF	$\text{Cauchy}(y\|\mu,\sigma) = \frac{1}{\pi\sigma}\frac{1}{1+((y-\mu)/\sigma)^2}$
Figure	See (Fig. 6.14)
Support	y: a real number
Parameter	μ: a real number σ: a positive real number
Mean	Undefined[a]
SD	Undefined
Example	Sections 7.8, 11.2.4, 11.2.5, 11.3.1, and 11.5

[a]Calculating $\int y\text{Cauchy}(y)dy$ according to the definition of the mean will result in undefined value that can take any value. This might seem to be counterintuitive, but the mean of a distribution is not only determined by the shape of a PDF, but also the value that is generated

Here, σ is a parameter that determines the scale, which satisfies the following:

$$\int_{\mu-\sigma}^{\mu+\sigma} \text{Cauchy}(y|\mu,\sigma)dy = 0.5$$

This distribution also has reproductive property:

if $y_1 \sim \text{Cauchy}(\mu_1,\sigma_1)$ and $y_2 \sim \text{Cauchy}(\mu_2,\sigma_2)$ are independent,
then a new random variable $y = y_1 + y_2 \sim \text{Cauchy}(\mu_1+\mu_2, \sigma_1+\sigma_2)$.

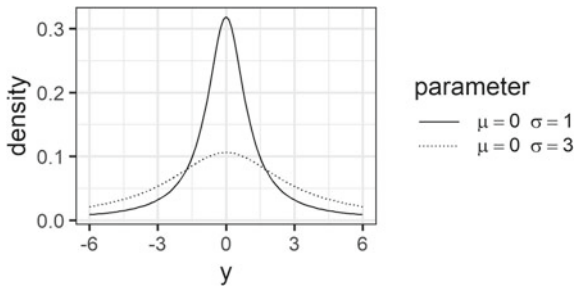


Fig. 6.14 Cauchy distribution

The tail of this distribution is very heavy because the exponent of exp function in its PDF contains $-(log\ y)^2$. In the case of $y \sim \text{Cauchy}(0, 1)$, although the probability that y lies in the range $[-1, 1]$ is around 0.5, there is still about 0.032 probability that y is larger than 10. That is, a Cauchy distribution generates values close to 0 in most cases, but occasionally generates very large values. This property can be very useful in some modeling practice.

If $y_1 \sim \mathcal{N}(0, 1)$ and $y_2 \sim \mathcal{N}(0, 1)$ are independent, then a new random variable $y = y_1/y_2 \sim \text{Cauchy}(0, 1)$.

Usage Example

This distribution is used to fit to the data that contains outliers (see Sect. 7.8) or change points (see Sect. 11.2.4).

Technical Point: Pearson Type VII Distribution

If we want to use a distribution that is symmetric and has a heavier tail than a Cauchy distribution, one option is to use a Pearson type VII distribution. Its PDF is as follows:

$$\text{PearsonVII}(y|b, \mu, \sigma) = C\frac{1}{\left\{1 + ((y - \mu)/\sigma)^2\right\}^b},$$

where C is a normalization constant. Parameter b ($> 1/2$) determines the heaviness of the tail: the smaller b, the heavier the tail. It is equal to a Cauchy distribution if $b = 1$, a Student-t distribution with υ degrees of freedom if $b = (\upsilon + 1)/2$, and a normal distribution if $b = \infty$.

The implementation in Stan uses the following relationship (Devroye, 1986):

if $y_1 \sim \mathcal{N}(0, 1)$ and $y_2 \sim \text{Gamma}(b, 1)$ are independent,
then $y = \mu + \frac{\sigma y_1}{\sqrt{y_2/2}} \sim \text{PearsonVII}(b, \mu, \sigma)$.

6.16 Student-t Distribution

PDF	$\text{Student_t}(y \mid \upsilon, \mu, \sigma) = \frac{\Gamma((\upsilon+1)/2)}{\Gamma(\upsilon/2)\sqrt{\pi\upsilon}\sigma}\left(1 + \frac{1}{\upsilon}\left(\frac{y-\mu}{\sigma}\right)^2\right)^{-(\upsilon+1)/2}$
Figure	See (Fig. 6.15)
Support	y: a real number
Parameter	υ: a positive real number, which is called the degrees of freedom μ: a real number σ: a positive real number
Mean	μ if $\upsilon > 1$, undefined otherwise
SD	$\sigma\sqrt{\upsilon/(\upsilon-2)}$ if $\upsilon > 2$, ∞ if $1 < \upsilon \leq 2$, undefined otherwise
Example	Sections 7.8, 9.2, 11.3.1, and 11.6

This distribution with one degree of freedom is equal to a Cauchy distribution, and this distribution with ∞ degrees of freedom is equal to a normal distribution. Degree of freedom is usually fixed.

The heaviness of the tail of $\text{Student_t}(\upsilon, 0, 1)$ greatly changes with the degrees of freedom υ as shown in Table 6.1. Therefore, if we want to use a heavy-tailed distribution but not as heavy as a Cauchy distribution, a Student-t distribution with 2 to 8 degrees of freedom can be used.

We can obtain another distribution by truncating and normalizing the $y \geq 0$ part of $\text{Student_t}(\upsilon, 0, \sigma)$. This new distribution is called a half-t distribution, which can be written as $\text{Student_t}^{+}(\upsilon, 0, \sigma)$.

Usage Example

This distribution is used for the data that include outliers (see Sect. 7.8). It can also be used for a weakly informative prior of regression coefficients (see Sect. 9.2.1). A half-normal distribution is often used as a weak informative prior distribution of parameters taking positive real numbers such as SD (see Sect. 9.2.2).

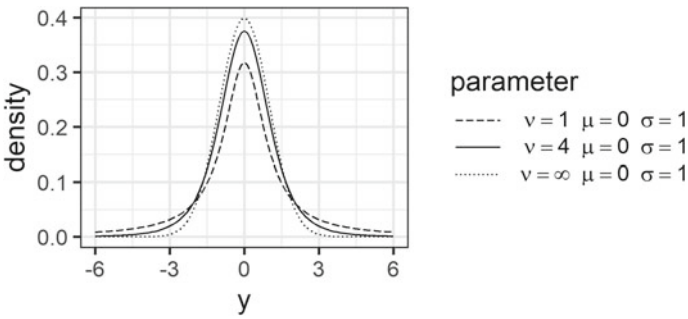


Fig. 6.15 Student-t distribution

Table 6.1 The heaviness of Student_t$(y|\upsilon, 0, 1)$ (probabilities are rounded)

Degrees of freedom	Probability that y is in the range of $(-1, 1)$	Probability that $y > 3$	Probability that $y > 10$
1	0.50	0.10	0.032
2	0.58	0.048	0.0049
3	0.61	0.029	0.0011
4	0.63	0.020	0.00028
6	0.64	0.012	0.000029
8	0.65	0.0085	0.0000042
∞	0.68	0.0014	$<10^{-10}$

6.17 Double Exponential Distribution (Laplace Distribution)

PDF	DoubleExponential$(y|\mu, \sigma) = \frac{1}{2\sigma}\exp\left(-\frac{\lvert y-\mu\rvert}{\sigma}\right)$
Figure	See (Fig. 6.16)
Support	y: a real number
Parameter	μ: a real number σ: a positive real number
Mean	μ
SD	$\sqrt{2}\sigma$
Appearance	Section 7.6

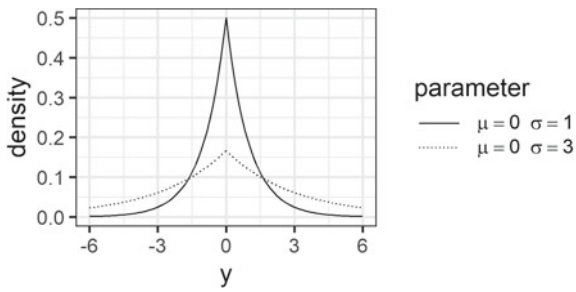


Fig. 6.16 Double exponential distribution (Laplace distribution)

This distribution can be represented by two exponential distributions that are symmetrically located in each side of the y-axis.

The tail of this distribution is heavier than that of a normal distribution because the exponent of exp function in its PDF contains $-y$.

If $y_1 \sim \text{Exponential}(\beta)$ and $y_2 \sim \text{Exponential}(\beta)$ are independent, then a new random variable $= y_1 - y_2 \sim \text{DoubleExponential}(0, 1/\beta)$.

Usage Example.

This distribution can be used for variable selection by using it as the prior distribution of regression coefficients because the PDF has a sharp peak centered around μ (see Sect. 7.6). Prophet[3] uses this distribution to estimate the change points of the linear trend of a time series.

Technical Point: Horseshoe Prior Distribution (Carvalho et al., 2008)

Stan development team recommends using a horseshoe prior for variable selection. The name comes from the fact that if we use this distribution as a prior, the "shrinkage factor" of regression coefficients will have a distribution that looks like a horseshoe-like shape (similar to $\text{Beta}(\alpha, \beta)$, where $\alpha, \beta < 1$). A double exponential distribution tends to shrink all regression coefficients towards 0 (i.e., shrinkage factors $= 1$). In contrast, a horseshoe prior can restrict the shrinkage factors to be either close to 1 (complete shrinkage) or close to 0 (no shrinkage). This property guarantees that only parts of the regression coefficients shrink towards 0, and the rest have the values close to their maximum likelihood estimate. Also, the fact that a horseshoe prior is always differentiable (while a double exponential distribution is not) makes Stan's calculation stable.

The prior distribution for regression coefficients $\beta[d]$ $(d = 1, \ldots, D)$ can be expressed and coded in Stan models as below:

$$\beta[d] \sim \mathcal{N}(0, \tau\lambda[d]) \qquad d = 1, \ldots, D$$
$$\lambda[d] \sim \text{Cauchy}^{+}(0, 1) \qquad d = 1, \ldots, D$$
$$\tau \sim \text{a weakly informative prior}$$

[3] https://github.com/facebook/prophet.

6.18 Exercise

(1) Generate random numbers that follow the probability distribution introduced in this chapter by using R or Python. (a hint for R users: use `sample.int()` for a Bernoulli distribution and a categorical distribution, `gtools` package for a Dirichlet distribution, and `mvtnorm` package for multivariate normal distribution)
(2) Search a distribution that is not introduced in this chapter. Then generate random numbers from that distribution. (a hint for R users: you may need to install and use `gamlss.dist` package or `extraDistr` package).

In this chapter, we explained PMFs and PDFs of several distributions, but did not mention their cumulative distribution functions. Also, deriving the theoretical values of means and SDs of the distributions are not discussed. The relevant information could be found from online resources such as Wikipedia.[4]

Unfortunately, we could not cover all the distributions that can be implemented in Stan. See the Stan language manual for the available distributions.

References

Carvalho, C., Polson, N., & Scott, J. (2008). The horseshoe estimator for sparse signals. Discussion Paper 2008–31. Duke University Department of Statistical Science.
Devroye, L. (1986). *Non-uniform random variate generation.* Springer.

[4] For example,
https://en.wikipedia.org/wiki/Beta_distribution,
https://en.wikipedia.org/wiki/Gamma_distribution, and
https://en.wikipedia.org/wiki/Relationships_among_probability_distributions.

Chapter 7
Issues of Regression

In this chapter, we will discuss several points that can potentially be problematic in extending regression analysis, and how to deal with these issues.

- Section 7.1 discusses the log transformation of data and its interpretability, and further provides some specific examples.
- Section 7.2 introduces nonlinear models and explains how to implement them.
- Section 7.3 introduces a concept called "interaction" and how it could be an issue when an explanatory variable is continuous or is categorical but has many categories.
- Section 7.4 introduces multicollinearity and some strategies that can be used to address it.
- Section 7.5 introduces model misspecification and explains how to improve it.
- Section 7.6 provides some examples to discuss how to deal with a large number of explanatory variables, and a modeling approach called Bayesian Lasso that can be used to select variables through parameter estimation.
- Section 7.7 explains how to incorporate censored data into likelihood such as data that includes inequality.
- Section 7.8 explains how to handle data that includes outliers, and demonstrates an example using a heavy tail distribution.

7.1 Log Transformation

In general, converting the data is not recommended especially when it makes data interpretation harder. One exception is transformation to the log scale. Log transformation is commonly used in data analysis, because of its intuitive mathematical meanings and interpretability.

We will consider an example of the price of rental apartments in Tokyo (`data-rental.csv`, 100 apartments). The response variable is the total cost Y (unit: 10 K yen) for 2 years of renting an apartment, and the explanatory variable is

© Springer Nature Singapore Pte Ltd. 2022

K. Matsuura, *Bayesian Statistical Modeling with Stan, R, and Python*,
https://doi.org/10.1007/978-981-19-4755-1_7

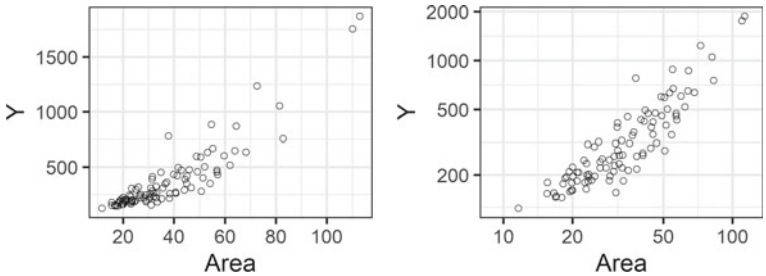


Fig. 7.1 A scatter plot with the explanatory variable on the x-axis and the response variable on the y-axis. Left: Linear scales. Right: Log scales

the $Area$ (m^2) of the apartment. Let us draw a scatter plot to check the data distribution. Plotting the values of the response variable and explanatory variable with linear scales (i.e., without any transformation) produces Fig. 7.1 (left), and plotting on log scales (i.e., taking log for both Y and $Area$) produces Fig. 7.1 (right).

Let us try simple linear regression using both scales.

For the first model, we use the values without any transformations. That is, we assume that $Area$ of the room determines the mean μ, and adding the noise term to μ will produce the total cost Y. This can be expressed as.

Model Formula 7.1

$$\begin{aligned} \mu[n] &= b_1 + b_2 Area[n] && n = 1, \ldots, N \\ Y[n] &\sim \mathcal{N}(\mu[n], \sigma) && n = 1, \ldots, N \end{aligned}$$

where N is the total number of apartments, and n is the index of each apartment. b_1, b_2 and σ are estimated from data. This model formula is equivalent to fitting a straight line in Fig. 7.1 (left).

In the second model, we fit the model after log transformations of each variable. That is, we assume that $\log_{10} Area$ determines the mean μ, and adding the noise term to μ will produce $\log_{10} Y$. This can be expressed as.

Model Formula 7.2

$$\begin{aligned} \mu[n] &= b_1 + b_2 \log_{10} Area[n] && n = 1, \ldots, N \\ \log_{10} Y[n] &\sim \mathcal{N}(\mu[n], \sigma) && n = 1, \ldots, N \end{aligned}$$

This model formula is equivalent to fitting a straight line in Fig. 7.1 (right).

We implemented both models with Stan and estimated the parameters. Substituting the posterior mean of each parameter into the original Model Formula 7.1 produces.

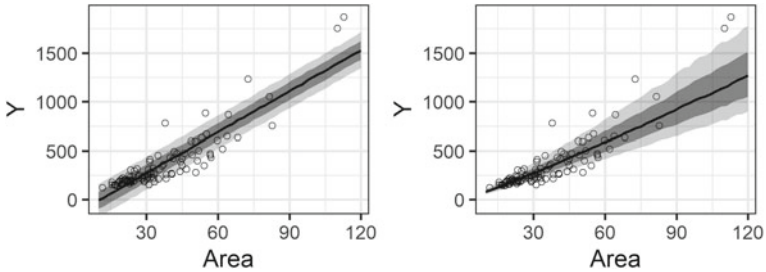


Fig. 7.2 PPC. Left: Model Formula 7.1. Right: Model Formula 7.2 The light gray bands are the 80% intervals, the dark gray bands are the 50% intervals, the black lines are the medians

$$\mu[n] = -147.8 + 13.99 Area[n] \qquad n = 1, \ldots, N$$
$$Y[n] \sim \mathcal{N}(\mu[n], 131.2) \qquad n = 1, \ldots, N$$

From here, we can interpret from this result that as $Area$ increases by 10 m^2, the cost Y increases by $13.99 \times 10 \approx 140$ (1400 K yen) on average. Likewise, the latter Model Formula 7.2 using log transformation becomes to.

$$\mu[n] = 0.80 + 1.11 \log_{10} Area[n] \qquad n = 1, \ldots, N$$
$$\log_{10} Y[n] \sim \mathcal{N}(\mu[n], 0.11) \qquad n = 1, \ldots, N$$

From here, we can interpret that as $Area$ increases 10 m^2, the cost Y increases $10^{1.11 \times 1} \approx 12.9$ times on average. Either model is easy to interpret, and log transformation does not reduce the interpretability of the model compared to the previous one.

Next, we plot two graphs (Figs. 7.2 and 7.3) for PPC (see Sect. 5.2.1) and one graph (Fig. 7.4) for PRC (see Sect. 5.2.2). For comparison, we plot both graphs on the linear scales.

From these graphs, we can see that there are following disadvantages in the regression model without the log transformation.

- The 80% prediction interval contains negative values (Figs. 7.2 and 7.3, left), which is an unrealistic prediction.
- The larger and more expensive apartments whose $Area > 70m^2$ distort the noise distribution from a normal distribution (Fig. 7.4, left). The reason is that the 30 M yen error for a 10 M yen apartment has 10 times more weight than the 3 M yen error for a 10 M yen apartment. Excluding several data points where $Area > 100m^2$ will largely change the estimation result. Thus, we consider that this analysis is not *robust*, and we should avoid such situations as much as we could (see Sect. 1.2).

Performing regression with the data after the log transformation improved these conditions (Figs. 7.2, 7.3 and 7.4, right), although it widens the prediction interval of the apartment whose $Area > 100m^2$ (Fig. 7.3, right).

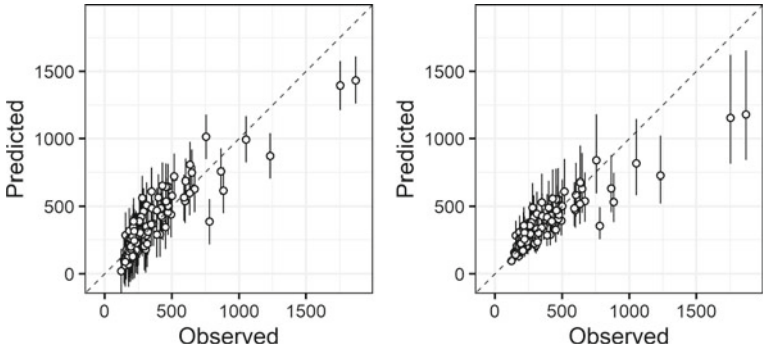


Fig. 7.3 PPC. Plot of observed data versus predicted data. Left: Model Formula 7.1. Right: Model Formula 7.2. The x-axis is the observed data $Y[n]$. The y-axis is the corresponding predicted value, showing the median and the 80% interval of the predictive distribution Y calculated from explanatory variables of the n-th apartment

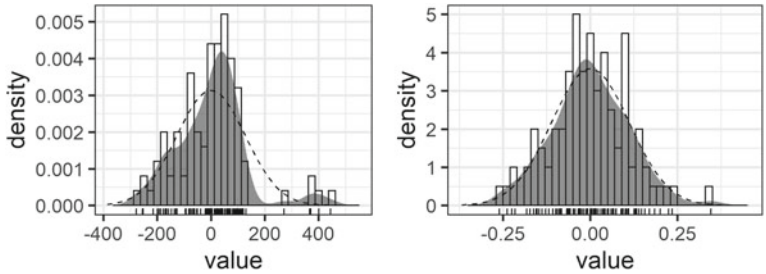


Fig. 7.4 PRC. The histogram and density function were calculated from the MAP estimates of the residuals of each n. The dotted line is a normal distribution whose mean is 0 and SD is MAP estimate of σ. Left: Model Formula 7.1. The x-axis represents the noise $\varepsilon[n] = Y[n] - \mu[n]$. Right: Model Formula 7.2. The x-axis represents the noise $\varepsilon[n] = \log_{10}(Y[n]) - \mu[n]$

However, as explained in Sect. 1.5, we cannot say whether a model with or without a log transformation is "correct". Rather, it is more important to be aware of the fact that we have chosen such an assumption.

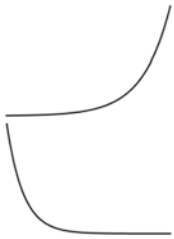
Another criterion for determining whether or not to take log is the background mechanism of the phenomenon. Typically, we take log in cases where we assume that the mechanism involves numbers that increase in double or exponential manner. For example, the number of cancer cells that doubles with cell division under an ideal environment. Also, the assets of compound interest increase exponentially. These are the cases where we might want to perform a log transformation.

7.2 Nonlinear Model

A model that uses a nonlinear curve is called a nonlinear model. In this section, we will introduce some examples of using the nonlinear model. When we know the data generating mechanism from the problem background, we may want to fit a nonlinear curve rather than a simple line, and this is one of the motivations of using the nonlinear model. A nonlinear model is widely used, mostly because a model that successfully incorporates the background knowledge has better generalizability and high prediction performance.

In Table 7.1, we have summarized the nonlinear functions that are commonly used in statistical modelling. In the following sections, we will look into each of these functions more in detail, and we will also introduce the examples.

Table 7.1 Frequently used nonlinear curves

Function name	Form of function $f(x)$	Example plot
Exponential	$a\exp(bx)$	
	$a\{1-\exp(-bx)\}$	
	$a\{1-\exp(-b(x-x_0))\}$	
Emax	$E_0+\frac{E_{\max}x}{ED_{50}+x}$	
Sigmoid Emax	$E_0+\frac{E_{\max}x^h}{ED_{50}^h+x^h}$	

7.2.1 Exponential Function

The first row of Table 7.1 is an exponential function. If the increase of the data has the mechanism of doubling, then this data follows an exponential function of time x.

$$a \exp(bx) \qquad \text{where } x \geq 0, a > 0, b > 0$$

This can be applied to various real-life phenomena. For instance, in the case of the compound interest, the asset increases in an exponential manner.[1] So does the number of cells under the mitosis process. Obviously, the cells will not undergo mitosis in an exponential manner forever, but it is known that before the available resources become limited and induce a constraint for the cell growth, the exponential function fit quite well for this phenomenon.

On the other hand, the pendulum amplitude is known to decrease exponentially, because of the air resistance.[2]

$$a \exp(-bx) \qquad \text{where } x \geq 0, a > 0, b > 0$$

Also, the value of items, second-hand electronics products, decreases exponentially. There is a wide range of examples where the value changes exponentially with time. Of note, we do not have to use the time for the x-axis for this function. For instance, if we use the x-axis to denote the distance from the train station, then we can also consider that the apartment rent decreases exponentially with its distance from the train station.

Next, as in the second row in Table 7.1, if we want to express that the value increases monotonically until it saturates at some point (i.e., reaches the plateau), then this can be expressed in the following function that contains the exponential function.

$$a\{1 - \exp(-bx)\} \qquad \text{where } x \geq 0, a > 0, b > 0 \tag{7.1}$$

where a is the parameter that determines the height of the plateau, and b is the parameter that determines at which point x it reaches the plateau. For instance, it is known that the effect of a new policy is 0 when $x = 0$, and it increases rapidly when the value of x is small, but then it saturates when the value of x is large. Another

[1] The interest is proportional to the current asset, so we can write this into an ordinary differential equation $dy/dx = by$ $(b > 0)$, where x represents the time and y represents the asset. The solution to this equation is $y = a \exp(bx)$ when we take a to be some initial values, and hence it increases exponentially. The same applies to the case of cell mitosis. Oftentimes, a mathematical model that uses the differential equation can provide some hints for us to think about how to build the models in statistical modelling.

[2] Behind this phenomenon, we can consider a relationship expressed as an ordinary differential equation: $dy/dx = -by$ $(b > 0)$. The solution to this equation is $y = a \exp(-bx)$, with a taking an initial value.

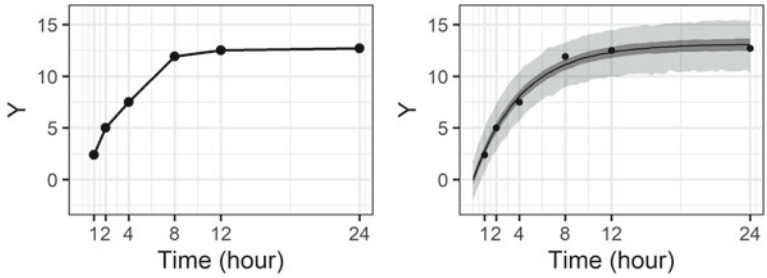


Fig. 7.5 Left: the relationship between the $Time$ after drug administration and the concentration Y. Right: PPC for the model by fitting Eq. (7.1). The light gray band is the 95% intervals, the dark band gray band is the 50% intervals and the black line is the median. The dots are the data

example is the concentration of a drug in the blood certain time after its intravenous administration, which also follows Eq. (7.1).[3] Later, we will use this example with its implementation.

We can also consider the following function as the function similar to the one mentioned above.

$$\begin{cases} a\{1 - \exp(-b(x - x_0))\} & \text{if } x \geq x_0 \\ 0 & \text{if } x < x_0 \end{cases} \quad \text{where } x \geq 0, a > 0, b > 0 \tag{7.2}$$

This function expresses that when the value of x is smaller than x_0, it takes 0. When x is larger than x_0, it increases rapidly until it reaches the plateau. Consider that there is an outdoor event next to a restaurant. We can use x to represent the number of the event attendees, and use y to represent the increase in the restaurant's sales caused by the event. The relationship between x and y follows Eq. (7.2), because we can intuitively think that there will be no effect on the sales until the scale of the event is large enough.

From here, we will introduce the implementation using the example of Eq. (7.1). Here we have a dataset (`data-conc.csv`) that recorded the concentration of a drug in the blood of a patient, Y (unit: ug/mL), and $Time$ (unit: hours) after the administration. We have plotted a line graph to check the data distribution (Fig. 7.5 left).

[3] Here, consider x represents the time, and y represents the concentration of a drug in the blood. When we administrate the drug into the body via intravenous administration, a certain amount of the drug ($A > 0$ in a unit of concentration) flows into the blood per time. Also, we can consider that the degradation of the drug in the body is proportional to its concentration. Therefore, the decrease in the concentration of the drug in the blood in a unit time is $-by$ ($b > 0$). By writing all of these into one ordinary differential equation, we obtain $dy/dx = A - by$. Further, by dividing A into $b \times a$, we can write it into $dy/dx = b(a - y)$. The solution to this equation would be $y = a\left[1 - C_0\exp(-bx)\right]$, with a constant value C_0. Here, we obtain $C_0 = 1$ using the condition $y = 0$ at $x = 0$.

For this data, we will implement regression by fitting Eq. (7.1). By writing it as the model formula, we have:

Model Formula 7.3

$$Y[t] \sim \mathcal{N}(a\{1 - \exp(-bX[t])\}, \sigma) \qquad t = 1, \ldots, T$$

where T represents the total number of time points when the drug concentrations were measured, and t is the index for each time point. $X[t]$ is the elapsed time after administration (i.e., $Time[t]$). σ represents the scale of noise, including the measurement error. We will estimate a, b and σ from this data. The implementation is as follows:

model7-3.stan

```
 1    data {
 2      int T;
 3      vector[T] X;
 4      vector[T] Y;
 5      int Tp;
 6      vector[Tp] Xp;
 7    }
 8
 9    parameters {
10      real<lower=0, upper=100> a;
11      real<lower=0, upper=5> b;
12      real<lower=0> sigma;
13    }
14
15    model {
16      Y[1:T] ~ normal(a*(1 - exp(-b*X[1:T])), sigma);
17    }
18
19    generated quantities {
20      array[Tp] real yp = normal_rng(a*(1 - exp(-b*Xp[1:Tp])),
sigma);
21    }
```

In line 5, the number of time points to be predicted is declared. In line 6, the elapsed times where we want to calculate the predictive distribution is declared. Compared to a simple linear regression, the difference is only in lines 16 and 20, and here we are using the nonlinear curve as the mean of the normal distribution. In lines 10 and 11, the parameters of the nonlinear curve are constrained based on Fig. 7.5 (left) and also background knowledge. The parameter estimation would not work well without these constraints. By plotting the prediction intervals from the estimation result, we can obtain Fig. 7.5 (right).

7.2.2 Emax Function

An Emax function on the fourth row in Table 7.1 also has the similar shape: it reaches the plateau at a certain point. It can be written as follows.

$$E_0 + \frac{E_{\max} x}{ED_{50} + x} \qquad \text{where } x \geq 0,\ E_{\max} > 0,\ ED_{50} > 0$$

An Emax function takes E_0 when $x = 0$. Therefore, E_0 is a parameter that represents the initial value. When x is close to 0, an Emax function can be approximated by $E_0 + (E_{\max}/ED_{50})x$, and this value increases linearly with the value of x. When the value of x is large, the value of an Emax function can be approximated by $E_0 + E_{\max}$, and this is the plateau. $E_{\max}$ is a parameter that represents the maximum value of the total increase. When $x = ED_{50}$, the value of an Emax function is $E_0 + E_{\max}/2$. Therefore, we can consider that ED_{50} is a parameter that represents x value where the value of the function achieves the half (50%) of its maximum $E_{\max}$.

The form of an Emax function is very similar to the Michaelis–Menten equation, which is used in the field of biochemistry. A part of the derivation of the Michaelis–Menten equation is introduced in Wikipedia, where interested readers can find as a reference.[4]

This is also very similar to Eq. (7.1), except that here we use ED_{50} instead of b to represent how fast we reach the plateau.

One example where we can use an Emax function is the administration dosage of drugs and their effects.

7.2.3 Sigmoid Emax Function

A sigmoid Emax function on the fifth row of Table 7.1 is an S-shaped function and can be expressed as follows.

$$E_0 + \frac{E_{\max} x^h}{ED_{50}^h + x^h} \qquad \text{where } x \geq 0,\ E_{\max} > 0,\ ED_{50} > 0,\ h > 0 \tag{7.3}$$

where h is a parameter called Hill coefficient. When $h = 1$, a sigmoid Emax function becomes an Emax function and when $h = \infty$, it is the same as a step function. Sometimes we estimate the value of h but sometimes we also fix it to a constant such as $h = 3$. When x is smaller than ED_{50}, a sigmoid Emax function takes a small value close to E_0. It increases rapidly in the region close to $x = ED_{50}$, and when x is larger than ED_{50}, it gradually reaches the plateau at $E_0 + E_{\max}$.

[4] https://en.wikipedia.org/wiki/Michaelis%E2%80%93Menten_kinetics.

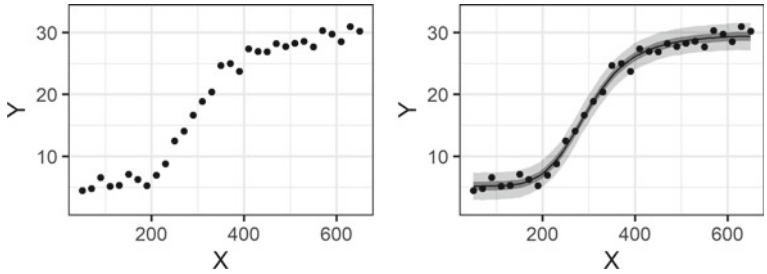


Fig. 7.6 Left: the relationship between the weight of the steak piece X and the satisfaction level Y. Right: PPC for the model by fitting Eq. (7.3). The light gray band represents 95% intervals, the dark gray band represents 50% intervals, and the black line represents the medians. The dots are the data points

The form of a sigmoid Emax function is almost the same as the Hill equation used in the field of biochemistry. A part of derivation of the Hill equation is introduced in Wikipedia page, where interested readers can find more information.[5]

Consider an example where a person eats a piece of steak every Sunday night. The weight of the steak (X, unit: g) is an unknown random number. Assume that we quantified and measured this person's satisfaction level (Y) after eating the steak on each Sunday night. And then we can consider that the relationship between X and Y follows a sigmoid Emax function. From now on, we will discuss how we can implement it. We have plotted a scatter plot from the data obtained during 31 weeks (`data-sigEmax.csv`) in Fig. 7.6 (left).

We can implement the regression by fitting Eq. (7.3) to this data. This can be written into the model formula as follows:

Model Formula 7.4

$$Y[n] \sim \mathcal{N}\left(E_0 + \frac{E_{\max} X[n]^h}{ED_{50}^h + X[n]^h}, \sigma\right) \qquad n = 1, \ldots, N$$

where N represents the number of measured data points (a pair of X and Y is considered to be one data point), and n is the index of each data point. We estimate the parameters E_0, $E_{\max}$, ED_{50}, h and σ from the data. The implementation is shown below:

model7-4.stan

```
data {
  int N;
  vector[N] X;
  vector[N] Y;
  int Np;
  vector[Np] Xp;
}
```

[5] https://en.wikipedia.org/wiki/Hill_equation_(biochemistry).

```
 8
 9    parameters {
10      real e0;
11      real<lower=0> emax;
12      real<lower=0, upper=max(X)> ed50;
13      real<lower=0, upper=10> h;
14      real<lower=0> sigma;
15    }
16
17    model {
18      Y[1:N] ~ normal(
19        e0 + emax * X[1:N] .^ h ./ (ed50 ^ h + X[1:N] .^ h), sigma);
20    }
21
22    generated quantities {
23      array[Np] real yp = normal_rng(
24          e0 + emax * Xp[1:Np] .^ h ./ (ed50 ^ h + Xp[1:Np] .^ h),
sigma);
25    }
```

We used the sigmoid Emax function in lines 19 and 24. Note that "`.^`" and "`./`" are the element-wise operators of "^" and "/" for `vector` type, respectively. From lines 10–13, we added constraints to the parameters of the curve based on Fig. 7.6 and background knowledge. Besides these constraints, we can also use weakly informative priors. We can plot the prediction intervals obtained from the estimation result, as shown in Fig. 7.6 (right).

7.2.4 Other Functions

Other well-known nonlinear functions include the following:

- A curve that increases in S-shape.[6]

$$C_0/\{1 + a\exp(-bx)\} \qquad \text{where } x \geq 0, a > 0, b > 0, C_0 > 0$$

- A curve that increases in the beginning and then decreases later.[7]

[6] This is the solution to the ordinary differential equation $dy/dx = -py^2 + qy$ (where $p, q > 0$ and they link a, b, C_0).

[7] This is the solution to the following simultaneous ordinary differential equation (representing how y_2 changes over time).

$$\begin{cases} dy_1/dx = -b_1 y_1 \\ dy_2/dx = b_1 y_1 - b_2 y_2 \end{cases}$$

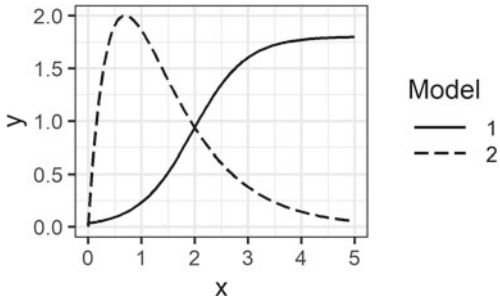


Fig. 7.7 Example of other nonlinear functions. The equation for Model 1 is $y = 1.8/\{1 + 50\exp(-2x)\}$ and the one for Model 2 is $y = 8\{\exp(-x) - \exp(-2x)\}$

$$C_0\{\exp(-b_2 x) - \exp(-b_1 x)\} \qquad \text{where } x \geq 0, b_1 > b_2 > 0, C_0 > 0$$

- Polynomial function (e.g., quadratic functions)
- Trigonometric functions

We have shown the first two in Fig. 7.7.

Besides ordinary differential equations, there are many other mechanisms that generate the nonlinear functions. For instance, the golf putting example (Gelman et al., 2020) is very instructive, and is highly encouraged for those who haven't read it yet.

Technical Point: Incorporating a complex differential equation into the model

Most of the nonlinear functions used in this section are from the differential equations, and they are the results of explicitly expressing the solution as the function of time t. However, when the differential equations become a little more complex, we cannot explicitly write down the solution, and cannot use it as the function of t. When we want to incorporate such complex differential equations using Stan, we need to use other special functions, which allow us to solve the differential equations and estimate the parameters at the same time. One function developed for this purpose is `ode_rk45` etc. We will not explain it in detail in this book, but readers that are interested in this topic can refer to the section of "Ordinary Differential Equations" in the Stan reference.

7.3 Interaction

Considering *interaction* in modeling means adding additional terms for the products of explanatory variables in regression analysis. It is aimed to account for the relationships between variables, which cannot be sufficiently represented using simple linear relationships.

For instance, in the example of multiple linear regression in Sect. 5.1, we used Model Formula 5.3

Model Formula 5.3

$$\begin{aligned} &\mu[n] = b_1 + b_2 Sex[n] + b_3 Income[n] && n = 1, \ldots, N \\ &Y[n] \sim \mathcal{N}(\mu[n], \sigma) && n = 1, \ldots, N \end{aligned}$$

Adding an interaction term yields the following model formula.

$$\begin{aligned} &\mu[n] = b_1 + b_2 Sex[n] + b_3 Income[n] + b_4 Sex[n] Income[n] && n = 1, \ldots, N \\ &Y[n] \sim \mathcal{N}(\mu[n], \sigma) && n = 1, \ldots, N \end{aligned} \tag{7.4}$$

There are several ways to interpret this model, but here we transform Eq. (7.4) as follows:

$$\begin{aligned} &\mu[n] = b_1 + b_2 Sex[n] + (b_3 + b_4 Sex[n]) Income[n] && n = 1, \ldots, N \\ &Y[n] \sim \mathcal{N}(\mu[n], \sigma) && n = 1, \ldots, N \end{aligned} \tag{7.5}$$

That is, the slope of $Income$ is not a constant b_3 but depends on another explanatory variable Sex.

This case gave us the simple example of interaction, where the binary variable Sex is involved in the interaction term. A more realistic case can be given, for instance, by considering a term including other continuous variable such as $Weight$ instead of Sex. If we include the interaction term between $Weight[n]$ and $Income[n]$ by replacing the one in Eq. (7.5), it means that the slope of $Income$ is not b_3 but $b_3 + b_4 Weight[n]$. Interpreting this term is more challenging, compared to the case using only Sex. If it is assumed that there is no interaction between $Income$ and $Weight$, the interaction term can be removed and we can easily understand their effects on the regression. Therefore, in multiple linear regression problem, it is often assumed that interactions do not exist, to simplify the modeling process. However, if we clearly observe the interaction during the "check data distribution" process, or when estimating the strength of interaction between the variables is itself the aim of the study, we should include interaction terms.

Now suppose we use the information of ethnicity (4 different ethnicities in total) instead of $Sex[n]$. To expand Eq. (7.5), we will make dummy variables $C_j[n]$ which

are 1 when a person belongs to the j-th ethnicity and 0 otherwise. Then, we will need to add the following term:

$$(b_3 + b_4 C_1[n] + b_5 C_2[n] + b_6 C_3[n] + b_7 C_4[n]) Income[n]$$

In this case, if the number of people with ethnicity 4 is small (i.e., the counts of $C_4[n] = 1$ is small), then the estimation of b_7 fails. One way to solve this problem is to set a loose constraint to regulate these coefficients parameters (b_4 to b_7) so that they behave similarly, which is discussed in hierarchical model in Chap. 8.

7.4 Multicollinearity

In multiple regression analysis, sometimes correlations between some of explanatory variables can be high, and hence regression coefficients cannot be estimated properly. This problem is known as *multicollinearity*.

Suppose that there are highly correlated explanatory variables $A[n]$ and $B[n]$. If $B[n] \cong A[n]$ ($B[n]$ is almost equal to $A[n]$), then a model formula

$$\begin{aligned} \mu[n] &= b_1 + b_2 A[n] + b_3 B[n] && n = 1, \ldots, N \\ Y[n] &\sim \mathcal{N}(\mu[n], \sigma) && n = 1, \ldots, N \end{aligned}$$

can be approximated by

$$\begin{aligned} \mu[n] &= b_1 + (b_2 + b_3) A[n] && n = 1, \ldots, N \\ Y[n] &\sim \mathcal{N}(\mu[n], \sigma) && n = 1, \ldots, N \end{aligned}$$

Let us estimate the parameters b_2 and b_3. Here, assume that $b_2 + b_3 = 3$ is satisfied no matter what the combination for (b_2, b_3) is. It could be (0, 3), (3, 0), or (1.3, 1.7), as long as they satisfy $b_2 + b_3 = 3$. In other words, their values are not uniquely determined, which makes it hard for MCMC to converge when we implement the model.

A simple way to solve this situation is to discard one of these explanatory variables based on background knowledge. For example, in the apartment rent data in Sect. 7.1, suppose that apartments have two highly correlated explanatory variables "Area" and "Number of rooms". In this case we can exclude the variable "Number of rooms" in the modeling processes. When the number of explanatory variables is large, we can calculate the pairwise correlations for variables, to obtain a correlation coefficient matrix. Then, if we find a pair of explanatory variables whose correlation coefficient exceeds the threshold, we can exclude one of them. Usually, a threshold is chosen from 0.8 to 0.95.

7.5 Model Misspecification

As we have briefly mentioned in Sect. 5.1.7, when we expect that the coefficients in the regression model represent the effect by changing an explanatory variable, we need to make sure that the model reflects the causal relationship correctly. When the specified model is wrong, it is called *model misspecification.* In this subsection, we will use the example where, due to the model misspecification, the interpretation of the coefficients in a regression model does not match with the background knowledge, and we will try to improve the situation.

We will use a 50-meter run record from the students in an elementary school (`data-50m.csv`, a total of 66 students). The response variable is the average speed Y (unit: meter per second), which is computed from the running time. The explanatory variable is the $Weight$ (unit: kg) of each student. To check the data distribution, we plotted a scatter plot in Fig. 7.8 (left). From here, we can observe that when the weight is larger, the speed is faster. However, this does not make sense if we think about the people around us. In fact, considering that this is data from the students in elementary school, there is another important Age factor, which should be another explanatory variable that we did not include here. By including the age data for all the students, we can plot a similar scatter plot, with the different marker shapes representing the student's age, as shown in Fig. 7.8 (right). From this plot and our experience, we can interpret this observation as: the student's weights would increase with the increasing ages, and the students would develop more muscle when they get older, which results in faster speed; however, if a student is overweight, the speed would decrease. This relationship is illustrated in Fig. 7.9.

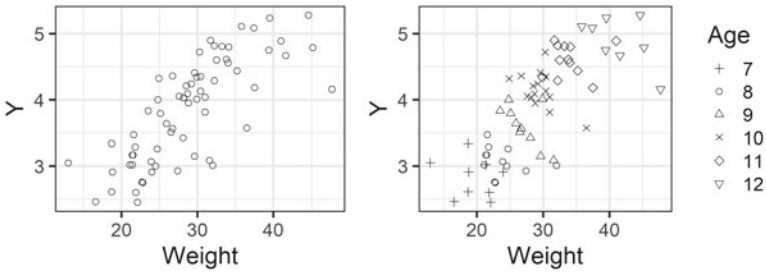


Fig. 7.8 Scatter plots with the weight on the x-axis and the average speed (meter per second) on the y-axis. Left: a plot without consideration on the students ages. Right: a plot with marker shapes representing students' age

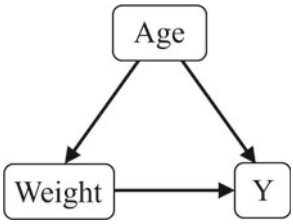


Fig. 7.9 The relationship between age, weight and speed

Based on the above discussion, we can consider the following possible model:

Model Formula 7.5

$$\begin{aligned}
\mu_w[n] &= c_1 + c_2 Age[n] && n = 1, \ldots, N \\
Weight[n] &\sim \mathcal{N}(\mu_w[n], \sigma_w) && n = 1, \ldots, N \\
\mu_y[n] &= b_1 + b_2 Age[n] + b_3 Weight[n] && n = 1, \ldots, N \\
Y[n] &\sim \mathcal{N}\big(\mu_y[n], \sigma_y\big) && n = 1, \ldots, N
\end{aligned}$$

where N is the total number of the students, and n is the index for each of them. We will estimate c_1, c_2, σ_w, b_1, b_2, b_3 and σ_y from the data. We implemented this model, and the estimated median and the 95% Bayesian confidence intervals of the parameters are as follows.

$$c_2 : 3.95(3.55, 4.38); b_2 : 0.59(0.51, 0.66); b_3 : -0.03(-0.05, -0.02)$$

From this result, we can confirm that the parameters c_2 (the age's effect on the weight) and b_2 (the age's effect on the speed) are both positive, and b_3 (the weight's effect on the speed) is negative. Although the result seems reasonable, note this does not mean that we can justify that this model formula is the "correct" one.

If we use a model that does not consider the age, we can still compute the predictive distribution of the running speed. And this prediction is likely to be quite accurate as long as there is no data of a young student who is overweight, or an old student who is underweight. However, when such data is observed and the model gives a bad performance, it is important that we can explore the cause. We need to execute the trial-and-error process and improve the model, considering the possibility of using other variables as well as other data. Usually, it is not easy to collect all the related explanatory variables and build the model that can explain the causal relationship from the very beginning. Instead, we should avoid model misspecification by using domain knowledge with the experts in the field and by repeating the data analysis workflow in cycles to improve the models (refer Sect. 1.4).

The more involved discussion on the causal inference would be beyond the scope of this book. But for the interested readers, we suggest having a read on the causal inference book (Hernán & Robins, 2020).

7.6 Variable Selection

Regression analysis with too many explanatory variables makes it hard to conduct parameter estimation and interpretation. This can be solved by including relationships between explanatory variables in the model. For example, suppose we have a dataset of expression of many genes before and after a drug treatment and we want to estimate effectiveness of this drug from this data. In this case, it would be preferred if we can include the biological information about the regulation relationships among the genes (such as suppression and activation) in the models.

However, information between explanatory variables is not always available to us. The number of explanatory variables should also be reduced to get a better prediction. To address these challenges, for example, we can use the following approaches:

- Suppose we have an explanatory variable A that takes binary values "0" or "1". If only a few data points out of total N data points have value "1", we do not have enough data with "1" and thus we may fail to estimate the coefficient for that explanatory variable. In such case, it is preferred to exclude variable A for modeling.
- We can apply a clustering method, such as hierarchical clustering,[8] to the explanatory variables. Using this result, we can merge similar explanatory variables or choose those main explanatory variables for modeling.
- Alternatively, we can create new explanatory variables by decreasing the dimension of data using methods like principal component analysis. One caveat is that this may make it harder for interpretation.

We can also reduce the number of explanatory variables through estimation. For instance, for a prior distribution of the coefficients β_k of the explanatory variables, we can set a double exponential distribution with $\mu = 0$ and a fixed σ (see Sect. 6.17). This is equivalent to adding

$$\log\left[\prod_k \frac{1}{2\sigma}\exp\left(-\frac{|\beta_k|}{\sigma}\right)\right] = -\sum_k \frac{|\beta_k|}{\sigma} + \text{const.}$$

to the original log-likelihood. Therefore, using this prior distribution means that we add a penalty to the large absolute value of β_k.[9] σ is a hyper parameter that controls the scale of the penalty. Usually, it is necessary to try different values of σ for adjustment. After parameter estimation, if the k-th explanatory variable has coefficients β_k that is close to 1, then we can consider this explanatory variable only has minor effect on the prediction. This method is called *Bayesian Lasso*. Besides double exponential distribution, Stan development team recommends using a horseshoe prior distribution for the same purpose (see Technical Point in Sect. 6.17).

[8] `hclust` function in R and `scipy.cluster.hierarchy.linkage` function in Python.

[9] This is called L1 regularization in the framework of machine learning.

7.7 Censoring

Suppose that in health examinations, we measure the concentration of protein Y from a person's blood 6 times. Because of the detection limit of the devices, a measurement that is lower than a certain threshold will change the data with an inequality sign. Such data is called *censored* data or data with *censoring*. An example is shown in Data File 7.1 (a row corresponds to a measurement).

Data File 7.1 Structure of `data-protein.csv`

```
1    Y
2    <25
3    32.3
4    <25
5    28.3
6    30.8
7    35.2
```

In this data, inequality sign represents that the detection lower limit is 25. The purpose of the analysis is to estimate the mean and SD of numerical values of protein Y. However, it becomes more challenging because of the inequality sign. Discarding these measurements that have inequalities means ignoring low values, therefore the estimated mean will be higher than the unbiased value. On the other hand, if we replace " `<25`" with "`25`", the estimated SD will be smaller than the actual value because the measurements with inequalities will be the same value. It is difficult to find a solution in traditional statistics textbooks, but Stan can deal with such a situation efficiently.

First, let us imagine the data generating mechanism. We assume that the true measurement value y is generated by adding noise (i.e., measurement error) to the true mean μ. Let the threshold of the detection limit be L, and if y is less than L, replace original y with "$< L$". Then, the model formula is as follows.

Model Formula 7.6

In the case of measurements without censoring:

$$Y[n] \sim \mathcal{N}(\mu, \sigma) \qquad n = 1, \cdots, N_{\text{obs}}$$

In the case of measurements with censoring:

$$y[n] \sim \mathcal{N}(\mu, \sigma) \text{ and } y[n] < L \qquad n = 1, \cdots, N_{\text{cens}}$$

where N_{obs} is the total number of measurement values that are above threshold (therefore not censored), and N_{cens} is the number of measurement values that are below the threshold and include inequality. n is the index of each measurement. We will estimate μ and σ from the data.

Next, let us consider the likelihood for each measurement. In the case of measurement without censoring, the likelihood is simply $\mathcal{N}(\text{Y}|\mu, \sigma)$. In the case of measurement with censoring, the likelihood is the probability that $y < L$ (i.e., Prob$[y < L]$). Prob$[y < L]$ can be transformed as follows.

$$
\begin{aligned}
\text{Prob}[y < L] &= \int_{-\infty}^{L} \mathcal{N}(\text{y}|\mu, \sigma)\text{dy} \\
&= \int_{-\infty}^{L} \frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2\right] dy \\
&= \int_{-\infty}^{\frac{L-\mu}{\sigma}} \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{1}{2}z^2\right] dz = \int_{-\infty}^{\frac{L-\mu}{\sigma}} \varphi(z)dz \\
&= \Phi\left(\frac{L-\mu}{\sigma}\right)
\end{aligned}
$$

where φ is the probability density function of the standard normal distribution[10] and Φ is its cumulative distribution function (CDF). Therefore, we can add $\log \Phi((L-\mu)/\sigma)$, which is the likelihood of these data with inequality, to the log posterior probability (`lp__` or `target` in Stan, see Sect. 3.6), to account for the censored measurement.

The implementation of Model Formula 7.6 is as follows

model7-6.stan

```
1    data {
2      int N_obs;
3      int N_cens;
4      vector[N_obs] Y_obs;
5      real L;
6    }
7
8    parameters {
9      real mu;
10      real<lower=0> sigma;
11    }
12
13    model {
14      Y_obs[1:N_obs] ~ normal(mu, sigma);
15      target += N_cens * normal_lcdf(L | mu, sigma);
16    }
```

Line 4: `Y_obs` are measurements that are higher than the censoring threshold.

Lines 14–15: We describe the log-likelihood separately for these measurements without censoring and these with censoring.

Line 14: The description is simple for data that are higher than the threshold.

Line 15: `normal_lcdf(L | mu, sigma)` is a useful Stan function and represents $\log \Phi((L-\mu)/\sigma)$. This line is equivalent to the following code

[10] The normal distribution with mean 0 and SD 1.

```
for (n in 1:N_cens) {
   target += normal_lcdf(L | mu, sigma);
}
```

Because the contents inside of `for` statement do not depend on the index `n`, we put these three lines together (in line 15) for simplicity. The notation of line 15 is more efficient because the number of operations of `target` is small.

7.8 Outlier

Suppose that most data points are in a certain range. A data point largely out of the range is called an *outlier*, although there is no strict objective definition. Assessment of outliers is generally a difficult task, and how to handle outliers in data analysis is also an unsolved problem. We will consider a countermeasure for outliers in statistical modeling in this section.

If it is obvious that an outlier was caused by some known artifacts, that data point could be excluded or appropriately adjusted before further analysis. But these obvious artifacts are not the only cause of outliers. For instance, suppose that a distribution of test scores can be approximated by a normal distribution, and most of the test taker's scores are between 50 and 80. In addition, suppose that there are two people scored 10 and another one scored 100. This is a possible situation, where most people have similar scores, while only a few individuals have extremely low or high scores. These extreme scores are also outliers but are different by nature from those caused by known artifacts. One way to deal with such a case is to exclude them for further analysis, recognizing them as exceptions. An alternative approach that is discussed here is to assume a mechanism that generates such rare values for the modeling. Here we show two examples for the latter.

Use Cauchy distribution or Student's t distribution

Consider an example of simple linear regression where the response variable is Y and the explanatory variable is X. From background knowledge, X and Y in this data are known to have a proportional relationship (`data-outlier.csv`). The scatter plot is shown in Fig. 7.10. The analysis team is discussing whether the point of $X = 3$ is an outlier or not.

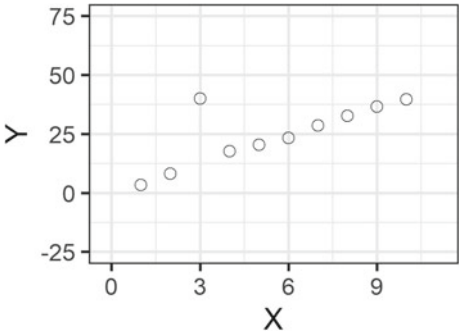
Let us use a normal distribution for the noise term as in the simple linear regression in Sect. 4.1. The model formula is as follows

Model Formula 7.7 (identical to Model Formula 4.4)

$$Y[n] \sim \mathcal{N}(a + bX[n], \sigma) \qquad n = 1, \ldots, N$$

The predictive distribution after estimation is visualized in Fig. 7.11 (left). It can be seen that in the range of $X = 1$ to 2, the line is pulled upwards by the outlier, thereby the slope is reduced by the outlier.

Fig. 7.10 Scatter plot



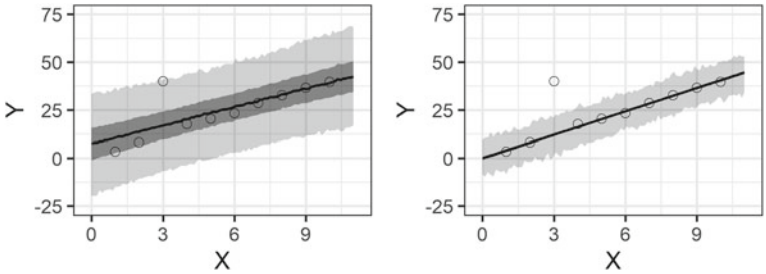


Fig. 7.11 PPC for each model. Left: model using a normal distribution for the noise term. Right: model using a Cauchy distribution for the noise term. The light gray bands are the 95% intervals, the dark gray bands are the 50% intervals, and the black lines are the medians

Next, we use a Cauchy distribution instead of a normal distribution, assuming that there is a mechanism that generates values close to the mean in most cases, but it also occasionally generates large values (see Sect. 6.15). The model formula is as follows:

Model Formula 7.8

$$Y[n] \sim \text{Cauchy}(a + bX[n], \sigma) \qquad n = 1, \cdots, N$$

The implementation of Model Formula 7.8 is as follows.

model7-8.stan

```
data {
  int N;
  vector[N] X;
  vector[N] Y;
  int Np;
  vector[Np] Xp;
}

```

```
 9    parameters {
10      vector[2] b;
11      real<lower=0> sigma;
12    }
13
14    model {
15      Y[1:N] ~ cauchy(b[1]+b[2]*X[1:N], sigma);
16    }
17
18    generated quantities {
19    array[Np] real yp=cauchy_rng(b[1]+b[2]*Xp[1:Np], sigma);
20    }
```

This code does not differ largely from the code of simple linear regression (**`model4-4b.stan`** in Sect. 4.3). The only difference is that a Cauchy distribution is used instead of a normal distribution in lines 15 and 19. The predictive distribution after estimation is visualized in Fig. 7.11 (right).

Because the outlier is generated from the heavy tail of the Cauchy distribution, the slope is not pulled by the outlier. The width of the 95 and 50% intervals also became smaller. However, these results do not suggest that either using a normal distribution or a Cauchy distribution in this case is "correct" (see Sect. 1.5). It is more important that we are aware that we have chosen such assumptions.

Use a mixture of normal distributions or Zero-Inflated Poisson distribution

Suppose that there are more than a few extreme values in the data (e.g., about 5% of the total data points). Also, the histogram of data shows two peaks, which resemble two normal distributions. In such a case, there is also an option of using a mixture of normal distributions, in order to fit one normal distribution to the data points that are closer to the smaller peak (for example, see Sect. 11.2.5).

For instance, the counts of visiting a store for people in the city can be expressed as a mixture of distributions: people who never visits and those who visits frequently. A Zero-Inflated Poisson (ZIP) distribution can be used in this case. This distribution is a mixture of a Bernoulli distribution and a Poisson distribution, accounting for these two patterns of people. Although Stan can handle a mixture of normal distributions and a ZIP distribution efficiently, we need the knowledge of Chap. 10. We will explain them there in detail (see Sect. 10.3).

References

Gelman, A., Hill, J., & Vehtari, A. (2020). *Regression and other stories*. Cambridge University Press.

Hernán, M. A., & Robins, J. M. (2020). *Causal inference: What If*. Chapman & Hall/CRC.

Chapter 8
Hierarchical Model

For all the models we have introduced so far, the differences in response variables are mainly represented by the differences in explanatory variables. Those differences that cannot be fully explained by the explanatory variables are treated as noise. The noise follows a distribution and the distribution is shared by all the data points. However, in many situations, it is possible that individual- or group-specific characteristics, which have not been observed as explanatory variables, can affect the response variables. In this book, we use terms "group difference" and "individual difference", to denote these effects. Using hierarchical models is one way to manage group and individual differences in the data. Specifically, we assume that parameters from each group are similar, and thus are generated from the same distribution. This assumption is a variation of parameter constraints and can provide a stable estimation and easier interpretation of results.

As we advance in the age of big data, information is more accessible than ever before. Nonetheless, particular information about certain subjects, such as groups or individuals, might be inadequate under some circumstances. Oftentimes, therefore, estimation will be easily conducted only when we use hierarchical models.

The contents of this chapter are as follows:

- In Sect. 8.1, we introduce the motivation of building a hierarchical model. For a dataset example, we implement different models, from a simple model to a hierarchical model, and discuss several features of hierarchical models.
- In Sect. 8.2, we expand the example in Sect. 8.1, and introduce how to build a hierarchical model that has multiple levels.
- In Sects. 8.3 and 8.5, we learn how to apply hierarchical models to other datasets. In particular, we build a hierarchical model based on the nonlinear model mentioned in Sect. 7.2 and a logistic regression model mentioned in Sect. 5.4.
- In Sect. 8.4, we introduce how to handle data that include missing values. Data transformation from "wide-format" to "long-format" is key.

© Springer Nature Singapore Pte Ltd. 2022

K. Matsuura, *Bayesian Statistical Modeling with Stan, R, and Python*,
https://doi.org/10.1007/978-981-19-4755-1_8

8.1 Introduction of Hierarchical Models

In this section, we will use a dataset that is similar to the one in Chap. 4. We have the records from a total of 40 employees in four big companies in a particular business field, and the data is saved as a comma-separated file (Data File 8.1, a row corresponds to an employee). Each column represents:

X: length of work experience of each employee (X, unit: years).
Y: employee's annual income (Y, unit: \$1 k).
CID: Company ID, that is, index of the company that the employee belongs to.

Data File 8.1 Structure of `data-salary-2.csv`

```
X,Y,CID
7,45.7,1
10,48.2,1
16,51.8,1
...
22,72.8,4
```

Here, the annual income Y is assumed to be the sum of the baseline income and the other factors (noise). In these four big companies, we believe that the baseline income is proportional to the employee's work experience. We also believe that "the baseline income for a zero-experienced employee" and "the amount of increase in the baseline income accompanied with one additional year of work experience" vary considerably from company to company. This motivates us to think about group differences (here, we also call them "company differences").

8.1.1 Set Up Purposes and Check Data Distribution

The purpose of this analysis is to understand the relationships between the baseline income and the work experience. Also, we want to include the marginal posterior distributions of "the baseline income for a zero-experienced employee" and "the increase in the baseline income accompanied with one additional year of work experience" for each company.

First of all, in order to check the possible relationships between the annual income and the work experience, we plot a scatter plot with work experience on the x-axis and annual income on the y-axis (Fig. 8.1). It shows that the annual income has nearly linear increase with work experience for each company. Note, however, for $CID = 4$, despite the fact that the baseline income is proportional to work experience in this company, the simple linear fit has a negative slope. This is because only three employees' ($X = 22, 24, 28$) records are available for this company. This is also an example in the discussion on overfitting in Sect. 2.2.

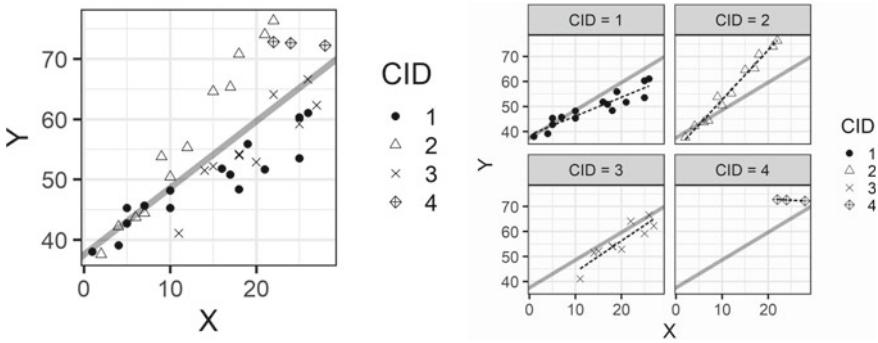


Fig. 8.1 Scatterplot showing association between the work experience and the annual income. The shapes of points indicate four different companies. Left: scatterplot of all the data points from four companies. The gray solid line is obtained by simple linear regression using all data points. Right: the same scatterplots for each company. The gray solid line is the same line as the one on the left plot, and the black dotted lines are separately obtained by simple linear regression using the data points from each company

8.1.2 Without Considering Group Difference

In this subsection, we simply fit regression model $a + bX$ for all four companies, without considering the possible difference among them. Here, the intercept parameter a can be interpreted as the annual income for a zero-experienced employee, and the slope parameter b can be interpreted as the increase in the income accompanied with one additional year of work experience.

Describe model formula

Recall the model we mentioned in Sect. 4.1.3:

Model Formula 4.1

$$
\begin{aligned}
Y[n] &= y_{\text{base}}[n] + \varepsilon[n] && n = 1, \ldots, N \\
y_{\text{base}}[n] &= a + bX[n] && n = 1, \ldots, N \\
\varepsilon[n] &\sim \mathcal{N}\big(0, \sigma_y\big) && n = 1, \ldots, N
\end{aligned}
$$

For notations, see Sect. 4.1.3. This model assumes that 40 employees in the four companies share the same underlying linear model $a + bX$, and each Y is generated by adding an individual noise term $\varepsilon[n]$ to $a + bX$. Also, we can rewrite the model by eliminating $y_{\text{base}}[n]$ and $\varepsilon[n]$, as:

Model Formula 8.1

$$
Y[n] \sim \mathcal{N}\big(a + b\ X[n], \sigma_y\big) \qquad n = 1, \ldots, N
$$

Implement models

The implementation of Model Formula 8.1 is as follows:

model8-1.stan

```
 1  data {
 2   int N;
 3   vector[N] X;
 4   vector[N] Y;
 5  }
 6
 7  parameters {
 8   real a;
 9   real b;
10   real<lower=0> s_y;
11  }
12
13  model {
14   Y[1:N] ~ normal(a + b*X[1:N], s_y);
15  }
```

Lines 8–9: The parameters `a` and `b` are declared. The straight line determined by these two parameters are fitted to all the data points.

Interpret results

The median and 95% Bayesian confidence intervals of the estimated parameters are:

$$a : 37.6(32.7 \text{ to } 42.8) \quad b : 1.10(0.81 \text{ to } 1.40) \quad \sigma_y : 6.80(5.45 \text{ to } 8.79).$$

Because the assumption is that a single linear model $a + bX$ is shared by all 40 employees, most of the observed values of annual income are far apart from the line $a + bX$, which also results in a large value for estimated σ_y.

Although this is one way of conducting the analysis, if we can separate the noise into group difference (company difference) and smaller noise (other factors), it may give practical advantages for interpretation. So now let us consider another way to build the model, by incorporating the company difference.

8.1.3 Groups Have Varying Intercepts and Slopes

We consider that baseline income for a zero-experienced employee and the increase in the baseline income accompanied with one additional year of work experience are specific features of the individual companies. With this assumption, we will estimate two parameters, intercept a and slope b, for each company separately. On the other hand, the parameter for the SD σ_y, which denotes the scale of noise, is assumed to be shared by all companies. For estimation of a and b, $a[1] + b[1]X$ is determined from the employees who belong to company $ID = 1$, $a[2] + b[2]X$ is determined from

the employees who belong to company $ID = 2, \ldots$, and thus using this approach is almost equivalent to fitting the data independently to four companies.

Describe model formula

This model can be written as:

Model Formula 8.2

$$Y[n] \sim \mathcal{N}\big(a[n2c[n]] + b[n2c[n]]X[n], \sigma_y\big) \qquad n = 1, \ldots, N$$

where $n2c[n]$ returns the company index c that the n-th employee belongs to. In other words, $n2c$ maps the person index n to the company index c, and corresponds to CID column in Data File 8.1. $a[1]$ to $a[C]$, $b[1]$ to $b[C]$, and σ_y are the parameters to be estimated from the data.

Implement models

The implementation of Model Formula 8.2 is as follows:

model8-2.stan

```
 1  data {
 2   int N;
 3   int C;
 4   vector[N] X;
 5   vector[N] Y;
 6   array[N] int<lower=1, upper=C> n2c;
 7  }
 8
 9  parameters {
10   vector[C] a;
11   vector[C] b;
12   real<lower=0> s_y;
13  }
14
15  model {
16   Y[1:N] ~ normal(a[n2c] + b[n2c] .* X[1:N], s_y);
17  }
```

Explanations of major changes from **model8-1.stan** are:

Line 3: The total number of the companies is declared as `C`.

Line 6: The $n2c$ is declared as an array of `int`.

Lines 10–11: The parameters `a`, `b` are replaced by `vector` types. Of note, noninformative priors are used for `a[1]` to `a[C]` and `b[1]` to `b[C]` because there is no prior description for them.

Line 16: This line corresponds to Model Formula 8.2. The behavior when using an array of integers (`n2c`) as subscripts is also the same as in R and numpy. That is, this line is a vectorized version of the following code:

```
for (n in 1:N) {
   Y[n] ~ normal(a[n2c[n]] + b[n2c[n]]*X[n], s_y);
}
```

We added `[1:N]` to `Y` and `X` to emphasize vectorization. Note that "`.*`" is the element-wise operator of "`*`".

Interpret results

The median and 95% Bayesian confidence intervals of the estimated parameters are:

$$a[1] : 38.7(36.0 \text{ to } 41.6) \quad b[1] : 0.75\ (0.57 \text{ to } 0.92) \qquad \sigma_y : 2.7(2.1 \text{ to } 3.5)$$
$$a[4] : 74.4(43.3 \text{ to } 105.4)\ b[4] : -0.08(-1.32 \text{ to } 1.16)$$

The intercept and the slope parameters of the linear models fitted to each company are estimated. However, because we did not make any assumption about relationships of $a[c]$ and $b[c]$ among four companies, variability of a and b cannot be estimated. Therefore, for those other companies that were not included in this dataset, there is no way to infer their intercepts and slopes from this model.

In addition, when the amount of the data in a group is not large enough, estimating the parameters a and b would be hard when we fit models for each company independently Notice that when $CID = 4$, $b[4]$ is estimated to be a negative value (Fig. 8.1, right), which is a manifestation of overfitting that mentioned in Sect. 2.2. In fact, such a case with limited number of data points which is caused by, for instance, having missing data, is common in real-life problem. We might have to remove the data from these particular groups, or make other compromise, in order to apply Model Formula 8.2. This is hardly an efficient way of making use of all the available data. These limitations motivate us to reconsider Model Formula 8.2. As suggested above, we can start considering incorporating group differences.

8.1.4 Hierarchical Model

Imagine data generating mechanisms

In the previous subsection, the slope parameter for $CID = 4$ was estimated to be an extreme value because of overfitting. In fact, it is reasonable to consider that $b[c]$ are relatively similar among these companies. For modeling, this assumption can be converted to mathematical expression by specifying that $b[c]$ are generated from the same distribution.

Specifically, we assume that each $b[c]$ is the sum of two parts: the field mean b_{field} (mean of all the companies in this business field), and the company difference $b_{\text{diff}}[c]$, which indicates deviation from the field mean. The latter term can be considered to follow a normal distribution with the mean 0 and SD σ_b. The same process can be applied to the intercept parameter $a[c]$. Adding this weak constrains to $a[c]$ and $b[c]$ will provide another level of interpretation for the estimated results. For instance, after

obtaining the results, we can state that "among these companies, baseline incomes for a zero-experienced employee have the SD σ_a". Also, even for those companies with only a few data points, we still can make full use of the available data to estimate their a and b, as well as σ_a and σ_b. For the prior choice of σ_a and σ_b, let us use noninformative priors and estimate them from the data. Because there are multiple levels for prior specifications, such a model is called a *hierarchical model*. As in this example, in a hierarchical model, a normal distribution with the mean 0 and SD σ is frequently used for the prior distribution. For σ, noninformative prior (uniform distribution) is used and is estimated from data.

Describe model formula

This model can be written as:

Model Formula 8.3

$$Y[n] \sim \mathcal{N}\big(a[n2c[n]] + b[n2c[n]]X[n], \sigma_y\big) \qquad n = 1, \ldots, N \tag{8.1}$$

$$a[c] = a_{\text{field}} + a_{\text{diff}}[c] \qquad c = 1, \ldots, C \tag{8.2}$$

$$a_{\text{diff}}[c] \sim \mathcal{N}(0, \sigma_a) \qquad c = 1, \ldots, C \tag{8.3}$$

$$b[c] = b_{\text{field}} + b_{\text{diff}}[c] \qquad c = 1, \ldots, C \tag{8.4}$$

$$b_{\text{diff}}[k] \sim \mathcal{N}(0, \sigma_b) \qquad c = 1, \ldots, C \tag{8.5}$$

We will estimate $a_{\text{field}}, a_{\text{diff}}[c], \sigma_a, b_{\text{field}}, b_{\text{diff}}[k], \sigma_b$, and σ_y from data. Noninformative priors are set for a_{field} and b_{field}, as well as σ_a and σ_b. If we remove the company difference terms $a_{\text{diff}}[c]$ and $b_{\text{diff}}[c]$ from Eqs. (8.2) and (8.4), Model Formula 8.3 would be equivalent to Model Formula 8.1. If we remove Eqs. (8.3) and (8.5), on the other hand, it would be equivalent to Model Formula 8.2. The point of Model Formula 8.3 is that we add weak constraints to $a_{\text{diff}}[c]$ and $b_{\text{diff}}[c]$ by specifying normal distributions in Eqs. (8.3) and (8.5).

Simulate models

Now let us generate data via simulation based on Model Formula 8.3. If we randomly sample σ_a and σ_b from noninformative prior, generated $Y_{\text{sim}}[n]$ corresponds to the random draws from the prior predictive distribution (see Sect. 2.4). But then there is a high possibility that we generate too many $b[c]$ with extremely large absolute values, and thus $Y_{\text{sim}}[n]$ are unlikely to be realistic values for annual incomes. Therefore, let us assign constant values for each parameter and then check the simulation results.

The R code to simulate Model Formula 8.3 is as follows:

sim-model8-3.R

```
1  set.seed(123)
2  N <- 40
3  C <- 4
4  N_c <- c(15, 12, 10, 3)
5  a_field <- 350
6  b_field <- 12
7  s_a <- 60
8  s_b <- 4
9  s_y <- 25
10  X <- sample(x=0:35, size=N, replace=TRUE)
11  n2c <- rep(1:4, times=N_c)
12
13  a <- rnorm(C, mean=0, sd=s_a) + a_field
14  b <- rnorm(C, mean=0, sd=s_b) + b_field
15  d <- data.frame(X=X, CID=n2c, a=a[n2c], b=b[n2c])
16  d$Y_sim <- rnorm(N, mean=d$a + d$b*X, sd=s_y)
```

The equivalent Python code is as follows:

sim-model8-3.py

```
1  import numpy as np
2  import pandas as pd
3
4  np.random.seed(123)
5  N = 40
6  C = 4
7  N_c = np.array([15, 12, 10, 3])
8  a_field = 350
9  b_field = 12
10  s_a = 60
11  s_b = 4
12  s_y = 25
13  X = np.random.randint(low=0, high=36, size=N)
14  n2c = np.repeat(np.arange(1,5), N_c)
15
16  a = np.random.normal(loc=0, scale=s_a, size=C) + a_field
17  b = np.random.normal(loc=0, scale=s_b, size=C) + b_field
18  d = pd.DataFrame(data={'X':X, 'CID':n2c, 'a':a[n2c-1],
       'b':b[n2c-1]})
19  d['Y_sim'] = np.random.normal(loc=d.a + d.b*X, scale=s_y,
       size=N)
```

Lines 2–4 in R (Lines 5–7 in Python): The total number of employees, companies, and the number of employees in each company are declared.

Lines 5–9 in R (Lines 8–12 in Python): The constants are assigned to parameters. Here `a_field`, `b_field`, `s_a`, `s_b` and `s_y` correspond to a_{field}, b_{field}, σ_a, σ_b and σ_y in Model Formula 8.3, respectively.

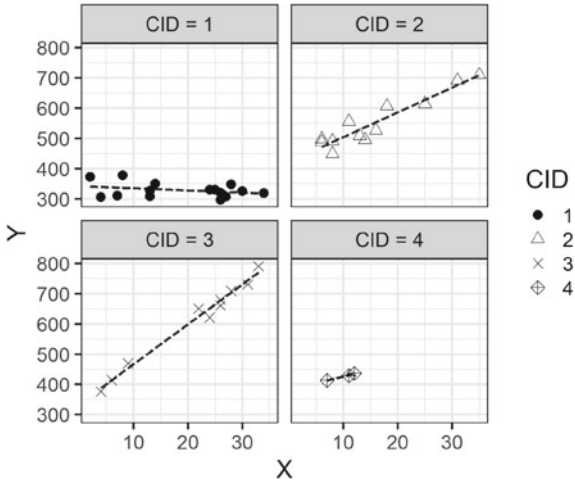


Fig. 8.2 An example of data generated from the simulation. Legends are the same as Fig. 8.1 (right)

Lines 13–14 in R (Lines 16–17 in Python): The parameters $a[c]$ and $b[c]$ are generated probabilistically. Note that the statement `rnorm(C, mean = 0, sd = s_a)` in R and `np.random.normal(loc = 0, scale = s_a, size = C)` in Python generates $a_{\text{diff}}[c]$.

Line 16 in R (Line 19 in Python): From these parameters, y_{base} is computed, and $Y_{\text{sim}}[n]$ is generated by adding noises from $\mathcal{N}(0, \sigma_y)$ to y_{base}. In order to check the distribution of $Y_{\text{sim}}[n]$, we use scatterplots using R for visualization (Fig. 8.2).

Recall that in lines 5–9 in R and lines 8–12 in Python, we assigned some constants to `a_field`, `b_field`, `s_a`, `s_b` and `s_y`. By changing these constants and random seeds, and visualizing the corresponding scatterplots generated as in Fig. 8.2., we can check what kind of data can be generated from Model Formula 8.3. When the model is not built appropriately, the simulation would produce unexpected results. Also, for the data generated from simulation, it is also helpful to run **`model8-3.stan`** (will be mentioned later in this subsection), to check whether the assigned values can be recovered correctly (this is a process called "parameter recovery", see Sect. 1.4). When the model becomes complex, simulation is a critical step for statistical modeling, to check what data is likely to be generated from the model, and what features the model has.

Implement models

The implementation of Model Formula 8.3 is as follows:

model8-3.stan

```
data {
  int N;
  int C;
  vector[N] X;
  vector[N] Y;
  array[N] int<lower=1, upper=C> n2c;
```

```
 7  }
 8
 9  parameters {
10    real a_field;
11    real b_field;
12    vector[C] a_diff;
13    vector[C] b_diff;
14    real<lower=0> s_a;
15    real<lower=0> s_b;
16    real<lower=0> s_y;
17  }
18
19  transformed parameters {
20    vector[C] a = a_field + a_diff[1:C];
21    vector[C] b = b_field + b_diff[1:C];
22  }
23
24  model {
25    a_diff[1:C] ~ normal(0, s_a);
26    b_diff[1:C] ~ normal(0, s_b);
27    Y[1:N] ~ normal(a[n2c] + b[n2c] .* X[1:N], s_y);
28  }
```

Compared to what we had for **model8-2.stan**, this script looks more complicated because `a[c]` is generated from `a_field` and `a_diff[c]`.

Lines 10–16: a_{field}, b_{field}, $a_{\text{diff}}[c]$, $b_{\text{diff}}[c]$, σ_a, and σ_b in Model Formula 8.3 are declared as `a_field`, `b_field`, `a_diff`, `b_diff`, `s_a` and `s_b`, respectively.

Lines 20–21: Parameters $a[c]$ and $b[c]$ in Model Formula 8.3 are declared and defined as `a` and `b`.

Lines 25–26: Normal distributions are assumed for $a_{\text{diff}}[c]$ and $b_{\text{diff}}[c]$ to add the weak constraints.

Interpret results

The median and the 95% Bayes confidence intervals of a part of the estimated parameters are as follows:

$$a_{\text{field}} : 36.6(17.6 \text{ to } 70.2) \quad b_{\text{field}} : 1.24(-0.36 \text{ to } 2.56)$$
$$\sigma_a : 9.3\ (1.1 \text{ to } 78.6) \quad \sigma_b : 0.81(0.32 \text{ to } 4.1)\ \sigma_y : 2.8(2.2 \text{ to } 3.8)$$

We can interpret this result as: among these companies, difference in the baseline incomes for a zero-experienced employee is approximately \$9.3 K; difference in the increase in the income accompanied with one additional year of work experience is approximately \$0.81 K, and the scale of noise is approximately \$2.8 K. Similarly, we can also compute the mean and the intervals for the parameters for individual companies, such as $a[c]$ and $b[c]$.

In this example, because the data is available only from four companies, the estimated 95% intervals for each parameter have large ranges. Two examples are the estimates for b_{field} and σ_a. Specifically, 95% intervals of b_{field} includes negative

values, and that of σ_a includes extreme values (for instance, \$60 K is within the intervals), which is not very realistic in practical. One way to solve this is using a weakly informative prior. We will discuss the weakly informative prior further in Sect. 9.2.

8.1.5 Model Comparison

To deepen our understanding of hierarchical model, let us compare Model Formula 8.1, 8.2, and 8.3 from different aspects.

Graphical model

First, we can look at these three models using the graphical models. In Fig. 8.3, Y are individual data points, a and b are the parameters, and σ_a and σ_b are the parameters that determine a and b. These parameters that determine other parameters are called *hyperparameters*. Checking the model using graphical model can help organize the model structure, and help illustrate the reason why Model Formula 8.3 is called a hierarchical model.

Hierarchical model induces shrinkage

To check what properties were introduced by building model with hierarchsical structure, the estimation result of nonhierarchical model (Model Formula 8.2) and hierarchical model (Model Formula 8.3) were compared (Fig. 8.4). The plot in Fig. 8.4 (left) shows that the values of $a[c]$ estimated from the hierarchical model (●) are closer to the value of a estimated from Model Formula 8.1 using all the data points (the gray solid line) than those estimated from the nonhierarchical model (◯). The further away the estimated $a[c]$ are from the gray line, the larger difference we can observe between the hierarchical and the nonhierarchical models, which is especially large when $CID = 4$. This is because in the hierarchical model, there is an assumption that each of $a[c]$ is similar to each other, and they are all generated from the same distribution. In contrast, there is no specific constraint for the parameter $a[c]$ in the nonhierarchical model. The difference among the estimated parameters

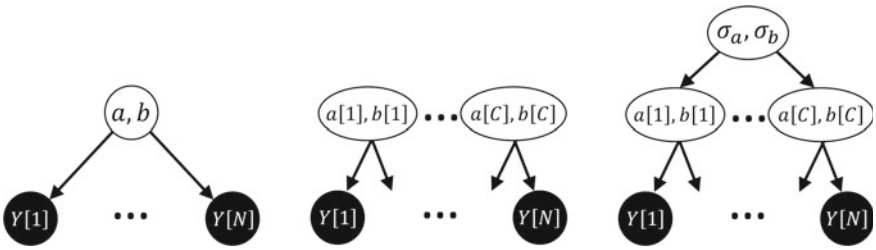


Fig. 8.3 Graphical model for Model Formula 8.1, 8.2, and 8.3 (left to right). For simplicity, a and b are represented as one single node, and the node of σ_y is omitted

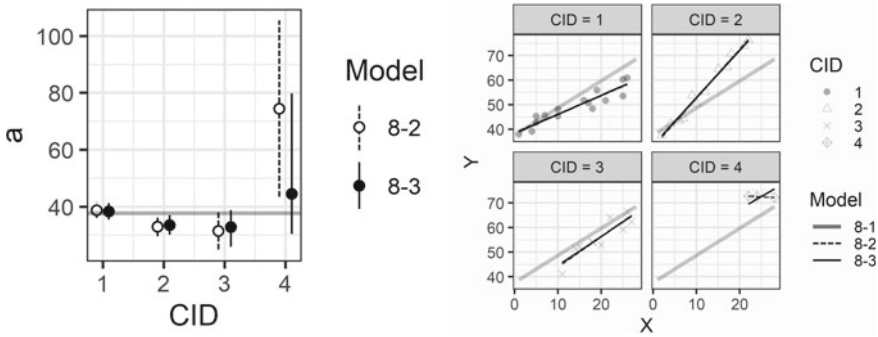


Fig. 8.4 Comparison of the results of the nonhierarchical model (Model Formula 8.2) and the hierarchical model (Model Formula 8.3). Left: showing the value of estimated $a[c]$, with KID on x-axis and the median and 95% Bayesian confidence intervals on the y-axis. The solid gray line is the median value of a that are estimated from Model Formula 8.1. Right: differences in the estimated baseline income. The solid gray line, dotted black line, and solid black line, are the medians of baseline income estimated from Model Formula 8.1, 8.2, and 8.3, respectively

in each group will be smaller when using a hierarchical model, and this property of the hierarchical model is called *shrinkage*. Shrinkage is an important property of the hierarchical model that we should always be aware of.

Although we have only showed an example using $a[c]$, $b[c]$ would have shrinkage as well. As a result, as shown in Fig. 8.4 (right), for $CID = 1, 2, 3$, the differences between the nonhierarchical model and the hierarchical model are small and the lines almost overlap. Only when $CID = 4$, the line fitted from hierarchical model, is closer to that of line fitted from all data points, thereby avoiding overfitting.

Predictive distributions in hierarchical model

In the case of a hierarchical model, multiple predictive distributions can be considered depending on the purpose. For example, in the example in this section, the following two types of predictive distributions can be considered:

- Case 1: One new person will be added to the existing company c
- Case 2: Another new company is added, and one new person is added to the new company

Model Formula 8.2 can predict Case 1, but not Case 2. In contrast, the hierarchical model (Model Formula 8.3) can predict both cases. The mathematical definition and evaluation of these predictive distributions will be discussed in Sect. 14.3.

Hierarchical model is a type of ridge regression

Let us look at the log posterior probability of Model Formula 8.2 and Model Formula 8.3. The one for Model Formula 8.2 can be written as:

$$\sum_{n=1}^{N} \log \mathcal{N}\big(Y[n]|a[n2c[n]] + b[n2c[n]]X[n], \sigma_y\big)$$

And the one for Model Formula 8.3 can be written similarly, with an additional term:

$$\sum_{c=1}^{C} \log \mathcal{N}(a_{\text{diff}}[c]|0, \sigma_a) = -\log \sigma_a - \frac{1}{2}\sum_{c=1}^{C}\left(\frac{a_{\text{diff}}[c]}{\sigma_a}\right)^2 + \text{const.}$$

In other words, the term corresponding to the hierarchical prior distribution causes regularization, thus preventing the absolute value of $a_{\text{diff}}[c]$ from being too large. This can be interpreted as a type of ridge regression. Note that it is not exactly the same as ridge regression, as parameter σ_a still needs to be estimated from the data.

8.1.6 Equivalent Representation of Hierarchical Models

In Sect. 4.1.3, we introduced that a certain model had several equivalent representations. By applying what we discussed, Model Formula 8.3 can have an alternative expression by removing $a_{\text{diff}}[c]$ and $b_{\text{diff}}[c]$:

Model Formula 8.4

$$\begin{aligned} Y[n] &\sim \mathcal{N}\big(a[n2c[n]] + b[n2c[n]]X[n], \sigma_y\big) && n = 1, \ldots, N \\ a[c] &\sim \mathcal{N}(a_{\text{field}}, \sigma_a) && c = 1, \ldots, C \\ b[c] &\sim \mathcal{N}(b_{\text{field}}, \sigma_b) && c = 1, \ldots, C \end{aligned}$$

This expression indicates that we believe that all the companies share the same mean a_{field}, and $a[c]$ of each company is generated from a normal distribution with this mean. The same would apply to b_{field} and $b[c]$. The Stan implementation of this model could result in different computing time and the convergence, even though these two are mathematically equivalent. The implementation of Model Formula 8.4 is as follows:

model8-4.stan

```
data {
  int N;
  int C;
  vector[N] X;
  vector[N] Y;
  array[N] int<lower=1, upper=C> n2c;
}

parameters {
  real a_field;
```

```
11    real b_field;
12    vector[C] a;
13    vector[C] b;
14    real<lower=0> s_a;
15    real<lower=0> s_b;
16    real<lower=0> s_y;
17   }
18
19   model {
20    a[1:C] ~ normal(a_field, s_a);
21    b[1:C] ~ normal(b_field, s_b);
22    Y[1:N] ~ normal(a[n2c] + b[n2c] .* X[1:N], s_y);
23   }
```

The main difference compared to **model8-3.stan** is, a_diff and b_diff are removed from parameters block, and a and b are declared instead. In addition, from lines 20-21, weak constraints are added for a and b. Because of these modifications, execution time for **model8-3.stan** was about 4 seconds per chain, while for **model8-4.stan**, it only took 1 second per chain (on my laptop). In a hierarchical model with multiple layers, we may see even a larger difference. We should be aware that this improvement depends on the data, the model and other factors.

8.2 Hierarchical Model with Multiple Layers

Let us say that the four big companies in Data File 8.1 used in Sect. 8.1 come from the same business field. In this section, we consider a situation where we add data from two additional business fields. Consider that we have collected records from 300 employees, saved in a comma-separated file (Data File 8.2, a row corresponds to an employee).

Data File 8.2 Structure of data-salary-3.csv

```
  1   X,Y,CID,FID
  2   7,45.7,1,1
  3   10,48.2,1,1
  4   16,51.8,1,1
      ...
301   20,72.7,30,3
```

The newly-added column FID represents the index of business filed that each company belongs to. These 300 employees are working in one of the 30 companies, and each of these 30 companies belongs to one of the three business fields. Note that the 40 employees in Data File 8.1 are included in the first field ($FID = 1$).

8.2.1 Set Up Purposes and Check Data Distribution

The new purpose of this analysis is to understand the relationships between the baseline income and work experience for each business field, in addition to that for each company as the previous section. We also want to know how this relationship vary among different business fields, which we call "field difference" hereafter.

If we ignore the information about different business fields, we can apply Model Formula 8.4 (**`model8-4.stan`**). Nonetheless, from background knowledge and real-life experience, it is reasonable to consider in another way. That is, baseline income for a zero-experienced employee, as well as the amount of increase in baseline income accompanied with work experience, would be field-specific. Here we call the mean of these two values "field mean". In addition, the variability of these two values among the companies can also be field-specific.

In order to test this assumption, we can check the data distribution by plotting the same scatterplot as in Fig. 8.1. Here, from Fig. 8.5, a positive relationship can be observed between the work experience and the annual income, and we can also see that the slopes and the intercepts are different from fields to fields.

Because the number of the companies has now increased to 30, plotting all the scatterplots for each company is not practical. Instead, we fitted 30 linear models independently, calculated their intercepts a and slopes b, and then plotted their distributions (Fig. 8.6). From this plot, except for $FID = 1$ where we do not have enough data to tell the shape of distribution, intercepts a and slopes b of other two fields ($FID = 2$ and $FID = 3$) seem to follow bell-shaped distributions, although it is still inadequate to tell if they follow normal distributions. Also, the plots indicate some characteristics of this data. For instance, we can compare the group $FID = 3$ with $FID = 2$ and infer the followings:

From the intercept a of $FID = 3$,

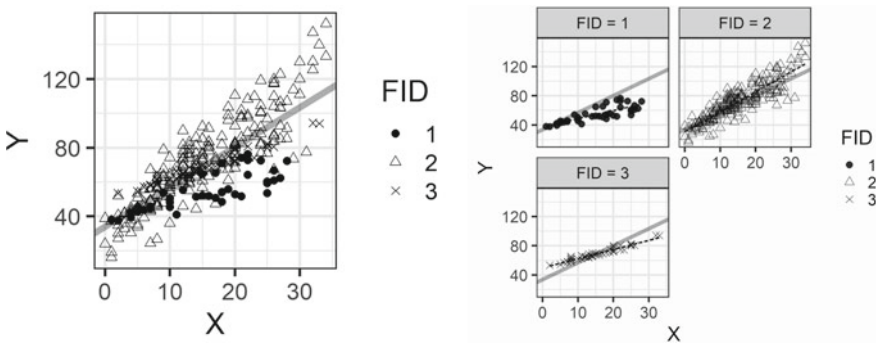


Fig. 8.5 Scatterplots of association between the work experience (years in the field) and the annual income. The only differences from Fig. 8.1 is that the numbers on the legend represent "each field" instead of "each company"

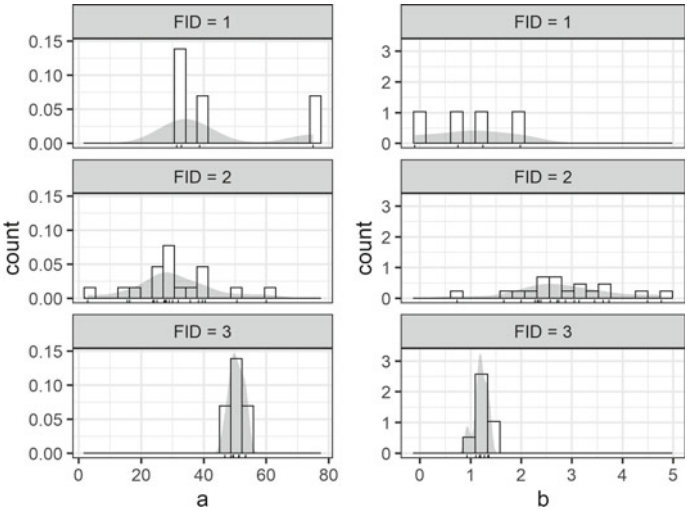


Fig. 8.6 For each company, histograms and the density plots of intercepts a (left) and slopes b (right) are shown. The three FID are represented in each row

1. Because the median of the distribution is more on the right side, this field has a higher field mean.
2. Because the width of the distribution is narrower, the variability of company differences in this field is smaller.

   From the slope b of $FID = 3$,
3. Because the median of the distribution is more on the left side, this field has a lower field mean.
4. Because the width of the distribution is narrower, the variability of company differences in this field is smaller.

The points 1 and 3 indicate that the mean of "the baseline income for a zero-experienced employee" and the mean of "the amount of increase in the baseline income accompanied with one additional year of work experience" are different among companies for each business field. Furthermore, the points 2 and 4 indicate that the variability of both these two quantities are different for each business field. Even if we simply repeat the linear regression in this way, aggregating the results and then checking their distributions will provide important information for modeling. One of the advantages of R and Python is that they are both good at such kinds of exploratory analysis, and we should make full use of this feature.

8.2.2 Imagine Data Generating Mechanisms and Describe Model Formula

We assume that the mean of "the baseline income of a zero-experienced employee", and the mean of "the amount of increase in the baseline income accompanied with one additional year of work experience", are different among the fields.

Let us consider intercept a as an example. As we did in Sect. 8.1.4, each of the $a_{\text{field}}[f]$ for a particular business field is considered to be the sum of two parts: the overall mean a_{all} for all the business fields, and the field difference $a_{\text{diff.field}}[f]$, which indicates deviation from the overall mean. That is, $a_{\text{field}}[f] = a_{\text{all}} + a_{\text{diff.field}}[f]$. An additional constraint was applied to the parameters that accounts for the difference among the fields, by considering that $a_{\text{diff.field}}[f]$ is generated from a normal distribution with mean 0 and SD σ_{af}. For σ_{af}, we set a noninformative prior and it will be estimated from the data. Alternatively, as discussed in Sect. 8.1.6, we can also consider that $a_{\text{field}}[f]$ is generated from a normal distribution with mean a_{all}, by eliminating $a_{\text{diff.field}}[f]$. Further, we consider that $a[c]$ for company c is generated from a normal distribution with mean $a_{\text{field}}[f]$, which is the parameter for the business field that the company c belongs to. The same idea would apply to the slope parameter b.

In addition, as suggested in the previous subsection, we assume that the SDs of $a[c]$ are different among the fields. The same assumption applies to the SDs of $b[c]$. We will use the same representation as the one introduced in Sect. 8.1.6, describing the model as:

Model Formula 8.5

$$Y[n] \sim \mathcal{N}\big(a[n2c[n]] + b[n2c[n]]X[n], \sigma_y\big) \qquad n = 1, \ldots, N \tag{8.6}$$

$$a_{\text{field}}[f] \sim \mathcal{N}\big(a_{\text{all}}, \sigma_{af}\big) \qquad f = 1, \ldots, F \tag{8.7}$$

$$b_{\text{field}}[f] \sim \mathcal{N}\big(b_{\text{all}}, \sigma_{bf}\big) \qquad f = 1, \ldots, F \tag{8.8}$$

$$a[c] \sim \mathcal{N}(a_{\text{field}}[c2f[c]], \sigma_a[c2f[c]]) \qquad c = 1, \ldots, C \tag{8.9}$$

$$b[c] \sim \mathcal{N}(b_{\text{field}}[c2f[c]], \sigma_b[c2f[c]]) \qquad c = 1, \ldots, C \tag{8.10}$$

where N represents the total number of employees, and C represents the number of the companies. The indices n, c and f correspond to employee, company and field, respectively. $c2f[c]$ returns the field index f that the c-th company belongs to. In other words, $c2f$ maps the company index c to the field index f. From the data, we will estimate $a_{\text{field}}[f]$, a_{all}, σ_{af}, $b_{\text{field}}[f]$, b_{all}, σ_{bf}, $a[c]$, $\sigma_a[f]$, $b[c]$, $\sigma_b[f]$, and σ_y.

The noise term, which is the effects from other factors except for work experience, might vary among the fields as well. In that case, we can consider the following equation instead of Eq. (8.6):

$$Y[n] \sim \mathcal{N}\big(a[n2c[n]] + b[n2c[n]]X[n], \sigma_y[n2f[n]]\big) \qquad n = 1, \ldots, N$$

As mentioned in Sect. 1.4, we recommend starting from a simple model, so we will use Eq. (8.6) here. In the process of constructing more complex models, the MCMC may not converge. This could be due to the lack of enough data, compared to the complexity of the model. In that case, we need to go back to the simpler model or add extra assumptions using such as weakly informative prior distributions. The latter will be explained in Sect. 9.2.

8.2.3 Implement the Model

The implementation of Model Formula 8.5 is as follows:

model8-5.stan

```
data {
  int N;
  int F;
  int C;
  vector[N] X;
  vector[N] Y;
  int<lower=1, upper=C> n2c[N];
  int<lower=1, upper=F> c2f[C];
}

parameters {
  real a_all;
  real b_all;
  vector[F] a_field;
  vector[F] b_field;
  vector[C] a;
  vector[C] b;
  real<lower=0> s_af;
  real<lower=0> s_bf;
  vector<lower=0>[F] s_a;
  vector<lower=0>[F] s_b;
  real<lower=0> s_y;
}

model {
  a_field[1:F] ~ normal(a_all, s_af);
  b_field[1:F] ~ normal(b_all, s_bf);
  a[1:C] ~ normal(a_field[c2f], s_a[c2f]);
  b[1:C] ~ normal(b_field[c2f], s_b[c2f]);
  Y[1:N] ~ normal(a[n2c]+b[n2c] .* X[1:N], s_y);
}
```

Line 3: The total number of business fields is declared as `F`.

Line 8: $c2f[c]$ in Model Formula 8.5 is declared as `c2f`.

Lines 14–15, 18–21: $a_{\text{field}}[f]$, $b_{\text{field}}[f]$, σ_{af}, σ_{bf}, σ_a, and σ_b in Model Formula 8.5 are declared as `a_field`, `b_field`, `s_af`, `s_bf`, `s_a` and `s_b`, respectively.

Lines 26–27: Corresponding to Eq. (8.7) and (8.8) in Model Formula 8.5.

Lines 28–29: Corresponding to Eq. (8.9) and (8.10) in Model Formula 8.5.

Implementing this code is straightforward, except that generating `c2f`. It might be a little bit troublesome. For this process, please refer **`run-model8-5.R`** and **`run-model8-5.py`** available on the supporting page of this book.

So far, we have discussed the hierarchical models, using the example from data containing information about annual income and employees' experience in companies. For the rest of this chapter, we will extend our discussion on hierarchical models to the non-linear model and logistic regression, and learn how to apply hierarchical models in different contexts.

8.3 Hierarchical Model for Nonlinear Model

Recall the time series data used in Sect. 7.2.1. We fit a nonlinear model to the data to analyze the relationship between the concentration of a drug in the blood of a patient, Y (unit: ug/mL), and $Time$ (unit: hour) after the administration. In this section, we consider a similar dataset but with additional 15 patients. Assume that there is a comma-separated data file from 16 patients' record (Data File 8.3, a row corresponds to a record from one patient).

Data File 8.3 Structure of `data-conc-2.csv`

```
PersonID,Time1,Time2,Time4,Time8,Time12,Time24
1,2.4,5.0,7.5,11.9,12.5,12.7
2,1.4,3.9,4.4,7.7,6.4,8.3
3,5.2,9.4,19.4,20.2,22.7,24.9
...
16,14.3,18.7,29.3,31.9,31.9,30.6
```

The first column $PersonID$ represents the patient's index. Columns $Time1$, $Time2$,…represent the time after administration (1 h, 2 h…, respectively). The patient whose data was used in Sect. 7.2.1 was assigned as $PersonID = 1$.

8.3.1 Set Up Purposes and Check Data Distribution

Based on the discussion in Sect. 7.2.1, let's say that for each patient, we want to know the maximum value of the drug concentration Y, as well as the value of $Time$ to

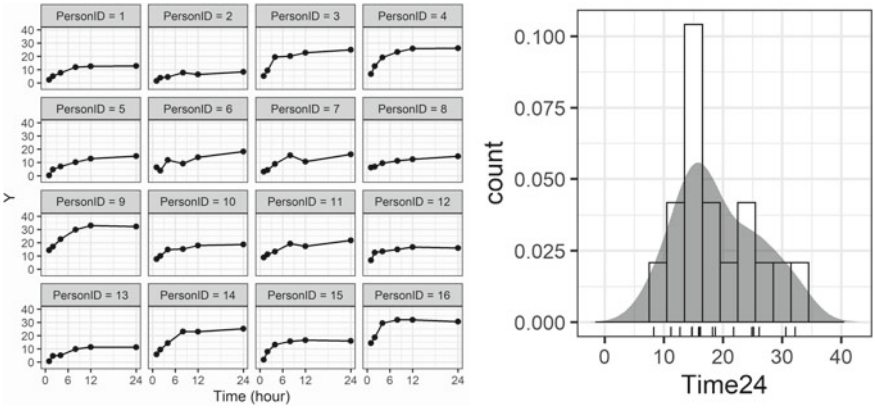


Fig. 8.7 Data distribution check. Left: relationship between the concentration of the drug Y and the $Time$ after administration. The same measurement time points were applied to all 16 patients. Right: histogram and density distribution of Y for each patient at the last time point of measurement

achieve that maximum value. Also, we want to know the scale of differences among these patients, and further visualize the predictive distributions of Y for each patient to order to make the reports.

To check the data distribution, we plotted the line graphs for each patient as shown in Fig. 8.7 (left). The trend for most of the patients is that Y increases initially, and then saturates at later time points. Next, we want to check the distribution of the maximum Y. By plotting the histogram of Y at the last time point of measurement for all the patients, we can obtain Fig. 8.7 (right). Although here we used the last time point, we can also use the maximum value for each patient across different time points, or use the 95th percentiles. This plot shows that the maximum Y seems to follow a unimodal bell-shaped distribution, although it is hard to determine if it follows a normal distribution.

8.3.2 Imagine Data Generating Mechanisms and Describe Model Formula

In Sect. 7.2.1, we considered the mechanism behind this dataset, and fitted the curve $a\{1 - \exp(-bt)\}$, where t represents the time after administration, a represents the maximum value of Y, and b represents the time needed to achieve the maximum value of Y. Now because the number of patients increased to 16 patients, we can fit the same curve $a\{1 - \exp(-bt)\}$ to each individual, and estimate $a[n]$ and $b[n]$ for all of them. This corresponds to the model discussed in Sect. 8.1.3. But as mentioned in Sect. 8.1.3, the disadvantage of this approach is that as the variability of these parameters among patients cannot be estimated, and thus no way to predict Y of the new patients who were not included in the original data. Alternatively, we can

build a hierarchical model, assuming that $a[n]$ and $b[n]$ follow certain distributions. Considering that both of them are nonnegative values, we specify that their log-transformed values $\log(a[n])$ and $\log(b[n])$ follow normal distributions.

Based on the above discussion, we can consider the model as:

Model Formula 8.6

$$
\begin{aligned}
Y[n,t] &\sim \mathcal{N}\big(\mu[n,t], \sigma_y\big) && n = 1,\ldots,N \quad t = 1,\ldots,T \\
\mu[n,t] &= a[n]\{1 - \exp(-b[n]X[t])\} && n = 1,\ldots,N \quad t = 1,\ldots,T \\
\log(a[n]) &\sim \mathcal{N}(a_{\text{all}}, \sigma_a) && n = 1,\ldots,N \\
\log(b[n]) &\sim \mathcal{N}(b_{\text{all}}, \sigma_b) && n = 1,\ldots,N
\end{aligned}
$$

where N represents the number of patients, T represents the number of measurement time points, n is the index for the patients, and t is the index of the time points. $\mu[n,t]$ represents a potential value before adding noise. $X[t]$ is the elapsed time (after drug administration) at the measurement time point t (i.e., $X[1] = 1$, $X[2] = 2$, …, $X[6] = 24$). We will estimate $a[n]$, a_{all}, σ_a, $b[n]$, b_{all}, σ_b, and σ_y from the data.

8.3.3 Implement Models

The implementation of Model Formula 8.6 is as follows:

model8-6.stan

```
data {
  int N;
  int T;
  vector[T] X;
  array[N] vector[T] Y;
  int Tp;
  vector[Tp] Xp;
}

parameters {
  real a_all;
  real b_all;
  vector[N] log_a;
  vector[N] log_b;
  real<lower=0> s_a;
  real<lower=0> s_b;
  real<lower=0> s_y;
}

transformed parameters {
  vector[N] a = exp(log_a[1:N]);
  vector[N] b = exp(log_b[1:N]);
  array[N] vector[T] mu;
  for (n in 1:N) {
```

```
25        mu[n,1:T] = a[n]*(1 - exp(-b[n]*X[1:T]));
26      }
27    }
28
29    model {
30      log_a[1:N] ~ normal(a_all, s_a);
31      log_b[1:N] ~ normal(b_all, s_b);
32      for (n in 1:N) {
33        Y[n,1:T] ~ normal(mu[n,1:T], s_y);
34      }
35    }
36
37    generated quantities {
38      array[N,Tp] real yp;
39      for (n in 1:N) {
40         yp[n,1:Tp] = normal_rng(a[n]*(1 - exp(-b[n]*Xp[1:Tp])),
s_y);
41      }
42    }
```

Line 5: Because the number of patients increases from one to N, `Y` is declared as an array that stores `N` vectors of length `T`.

Lines 13–14: $\log(a[n])$ and $\log(b[n])$ are declared as `log_a` and `log_b`, respectively, and weak constraints are given to them further in lines 30–31.

Lines 21–22: Curve parameters `a` and `b` are defined.

Lines 24–26, 32–34, 38–41: We have added `for` loops for patients because now there are the same number of curves as the number of patients.

In the R/Python code to run **`model8-6.stan`**, we read Data File 8.3, remove the $PersonID$ column, and pass the remaining data frame to `Y` in Stan.

8.3.4 Interpret Results

The median and 95% Bayesian confidence intervals of a part of the parameters are as follows:

$$a_{\text{all}} : 2.86(2.64 \text{ to } 3.08) \quad b_{\text{all}} : -1.17(-1.43 \text{ to } -0.95)$$
$$\sigma_a : 0.40(0.28 \text{ to } 0.64) \quad \sigma_b : 0.37(0.19 \text{ to } 0.65) \quad \sigma_y : 1.70(1.44 \text{ to } 2.04)$$

Note that except for σ_y, others are on logarithm scale, which can easily be reversed to the linear scale by using exp function on the MCMC sample. Similar to these, we can also calculate the medians and intervals of $a[n]$ and $b[n]$ for each patient. By doing the analysis in this way, we can estimate the variability among the patients. Figure 8.8 shows the visualization of the patient-wise predictive distributions, estimated from this analysis.

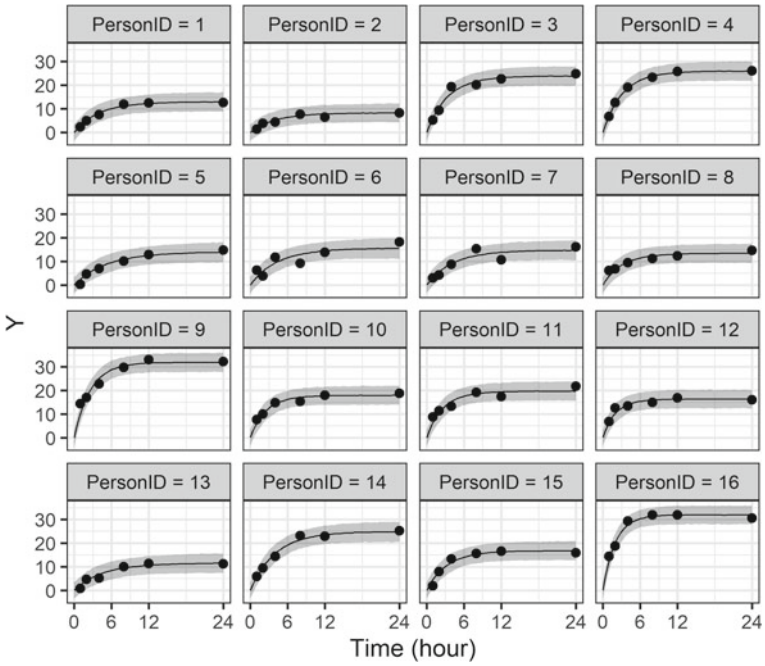


Fig. 8.8 The predictive distributions for each patient. The gray bands are the 95% intervals and the black lines are the estimated medians. The black dots are the observed values

8.4 Missing Data

Let us consider a situation in which the data in the previous section contains missing values, i.e., due to the failure of observations, some of the time points do not contain measurements. Although missing values is common in real data, Stan does not have missing value type. Therefore, we need to address this from the data processing.

Suppose that some of the observations in Data File 8.3 in the previous section have become missing values (NAs) (Data File 8.4). If we declare data `Y` as a two-dimensional array in the Stan code and pass a `data.frame` of Data File 8.4 to run it as we did in **`model8-6.stan`**, we will get an error at the missing value, because Stan does not handle missing values.[1] The Stan code can by simplified by transforming such data from a so-called "wide-format" (Data File 8.4) to a "long-format" (Data File 8.5) where a measurement corresponds to a line.

Data File 8.4 Structure of `data-conc-2-NA-wide.csv` (wide-format)

```
PersonID,Time1,Time2,Time4,Time8,Time12,Time24
1,2.4,NA,7.5,11.9,12.5,NA
2,NA,3.9,4.4,7.7,6.4,8.3
```

[1] In contrast, JAGS and WinBUGS can automatically handle missing values of outcomes.

```
4     3,5.2,9.4,19.4,NA,NA,NA
      ...
17    16,14.3,NA,NA,31.9,31.9,30.6
```

Data File 8.5 Structure of `data-conc-2-NA-long.csv` (long-format)

```
1     PersonID,TimeID,Y
2     1,1,2.4
3     3,1,5.2
4     4,1,6.7
      ...
88    16,6,30.6
```

The values in $TimeID$ columns are the indices of time points rather than the elapsed time. The rows with NA in column Y have been deleted (e.g., the row with $TimeID = 1$ for the patient with $PersonID = 2$ has been deleted). In total, there are 87 observed measurements.

By slightly changing Model Formula 8.6 in the previous section, the model formula for long-format data is as follows:

Model Formula 8.7

$$
\begin{aligned}
Y[i] &\sim \mathcal{N}\big(\mu[i2n[i], i2t[i]], \sigma_y\big) && i = 1, \ldots, I \\
\mu[n, t] &= a[n]\{1 - \exp(-b[n]X[t])\} && n = 1, \ldots, N \quad t = 1, \ldots, T \\
\log(a[n]) &\sim \mathcal{N}(a_{\text{all}}, \sigma_a) && n = 1, \ldots, N \\
\log(b[n]) &\sim \mathcal{N}(b_{\text{all}}, \sigma_b) && n = 1, \ldots, N
\end{aligned}
$$

where I represents the number of measurements observed (87 here) and i represents the index of each measurement. Note that $\mu[n, t]$ still has values at all time points for all patients, regardless of whether the measurement is missing or not. $i2n[i]$ returns the person index n that the i-th measurement belongs to. In other words, $i2n$ maps the measurement index i to the person index n, and corresponds to $PersonID$ column in Data File 8.5. Similarly, $i2t[i]$ returns the time point index t that the i-th measurement belongs to. $i2t$ corresponds to $TimeID$ column and $Y[i]$ corresponds to Y column in Data File 8.5. The rest of the variables are the same as Model Formula 8.6.

The implementation of Model Formula 8.7 is as follows:

model8-7.stan

```
1   data {
2     int I;
3     int N;
4     int T;
5     vector[T] X;
6     array[I] int<lower=1, upper=N> i2n;
7     array[I] int<lower=1, upper=T> i2t;
8     vector[I] Y;
9     int Tp;
```

```
10      vector[Tp] Xp;
11    }
12
13    parameters { the same as model8-6.stan }
14
15    transformed parameters { the same as model8-6.stan }
16
17    model {
18      log_a[1:N] ~ normal(a_all, s_a);
19      log_b[1:N] ~ normal(b_all, s_b);
20      for (i in 1:I) {
21        Y[i] ~ normal(mu[i2n[i], i2t[i]], s_y);
22      }
23    }
24    generated quantities { the same as model8-6.stan }
```

Line 2: The number of observed measurements is declared as `I`.

Lines 6–8: The columns $i2n$, $i2t$, and Y are declared as `i2n`, `i2t`, and `Y`, respectively.

Lines 20–22: These variables allow us to use only the observed measurements for the `for` loop in this way.

The R code to transform the data from wide-format to long-format and to run **`model8-7.stan`** is as follows:

`run-model8-7.R` (Lines 1-17)

```
 1  library(dplyr)
 2  library(tidyr)
 3  library(cmdstanr)
 4
 5  d_wide <- read.csv('input/data-conc-2-NA-wide.csv')
 6  N <- nrow(d_wide)
 7  X <- c(1, 2, 4, 8, 12, 24)
 8  T <- length(X)
 9  Tp <- 60
10  Xp <- seq(from=0, to=24, length=Tp)
11  colnames(d_wide) <- c('PersonID', paste0('TimeID', 1:T))
12  d <- d_wide %>%
13    pivot_longer(-PersonID, names_to='TimeID', values_to='Y') %>%
14    mutate(TimeID=readr::parse_number(TimeID)) %>%
15    na.omit()
16  data <- list(I=nrow(d), N=N, T=T, X=X, Tp=Tp, Xp=Xp,
17               i2n=d$PersonID, i2t=d$TimeID, Y=d$Y)
```

Line 13: `pivot_longer` function in `tidyr` package coverts the wide-formatted data into the long-formatted one.

Line 15: `d` is created by deleting the lines that include missing values. Saving this `d` as a csv file yields the aforementioned Data File 8.5.

Similar, the Python code to transform the data from wide-format to long-format and to run Stan can be as follows:

run-model8-7.py (Lines 1-17)

```
1   import numpy as np
2   import pandas
3   import cmdstanpy
4
5   d_wide = pandas.read_csv('input/data-conc-2-NA-wide.csv')
6   N = len(d_wide)
7   X = [1, 2, 4, 8, 12, 24]
8   T = len(X)
9   Tp = 60
10  Xp = np.linspace(0.0, 24.0, num=Tp)
11  colnames = ['PersonID'] + ['Y' + str(x) for x in range(1,T+1)]
12  d_wide = d_wide.set_axis(colnames, axis='columns')
13  d = pandas.wide_to_long(d_wide, stubnames='Y', i='PersonID',
        j='TimeID')
14  d = d.sort_values(by=['PersonID', 'TimeID']).reset_index()
15  d = d.dropna()
16  data = d.rename(columns={'PersonID':'i2n', 'TimeID':
        'i2t'}).to_dict('list')
17  data.update({'I':len(d), 'N':N, 'T':T, 'X':X, 'Tp':Tp,
        'Xp':Xp})
```

Line 13: `wide_to_long` function in Pandas library coverts the wide-formatted data into the long-formatted one.

Line 15: `d` is created by deleting the lines that include missing values. Saving this `d` as a csv file yields the aforementioned Data File 8.5.

Figure 8.9 shows the transitions of the prediction intervals of the estimated μ for the patients who have missing values. We can see that even if there are missing values, the data at the remaining time points can be used effectively to estimate the parameters.

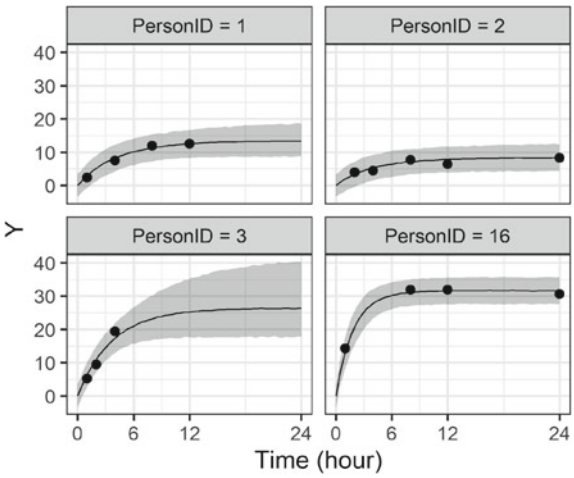


Fig. 8.9 The predictive distributions of Y for four patients. The gray bands are the 95% prediction interval, the black lines are the medians, and the black dots are the observed data

8.5 Hierarchical Model for Logistic Regression Model

In Sect. 5.4, we used logistic regression for the customers' purchase data (Data File 8.6). This is a comma-separated data file containing records from 444 visits, where a row corresponds to a visit. Each column represents:

$PersonID$: Customer ID.

Sex: A binary value representing the sex of the customer (0: female, 1: male).

$Income$: Last year's annual income (Unit: $1 K).

$Discount$: A binary value representing if there was discount (0: no discount, 1: discount).

Y: A binary value representing if the customer purchased or not (0: did not purchase, 1: purchased).

Data File 8.6 Structure of `data-shopping-3.csv`

```
  1    PersonID,Sex,Income,Discount,Y
  2    1,0,85.1,0,1
  3    1,0,85.1,1,1
  4    1,0,85.1,0,0
       ...
445    50,1,70,0,1
```

Note that because the same customers can visit and purchase multiple times, we see that there are several rows with the same $PersonID$. Recall Sect. 5.4 where we used the logistic regression model for the response variable Y as follows:

Model Formula 5.5

$$q[v] = \text{inv_logit}(b_1 + b_2 Sex[v] + b_3 Income[v] + b_4 Discount[v]) \qquad v = 1, \ldots, V$$
$$Y[v] \sim \text{Bernoulli}(q[v]) \qquad v = 1, \ldots, V$$

When we introduced this model, we did not consider that persons have their own purchase patterns. But from real-life experience, it is likely that the purchase probability q vary largely from persons to persons. Therefore, in this section, we will build a model that accounts for this personal difference.

8.5.1 Set Up Purposes

As in Sect. 5.4, we assume that we want to know if and how the three explanatory variables (Sex, $Income$, $Discount$) can predict the response variable Y, as well as how much each of them contributes to the purchase probability. In addition, we want to estimate the variability among the persons. We will leave the data distribution check step for the readers as an exercise.

8.5.2 Imagine Data Generating Mechanisms

As in this case, when we have multiple factors to consider, it is a standard approach to consider their effects separately. For now, we separate the factors that determine the purchase probabilities into two parts: those person-specific factors and those visit-specific factors. Then Sex, $Income$ and person difference are person-specific, and $Discount$ is visit-specific. We fit a regression model for each of these factors with explanatory variables. Finally, the purchase probability is computed from the sum of all the factors, with the logistic transformation into the range of (0, 1).

8.5.3 Describe Model Formula

From these discussions, we can describe the model as follows:

Model Formula 8.8

$$x[v] = b_1 + x_{\text{person}}[v2n[v]] + x_{\text{visit}}[v] \qquad v = 1, \ldots, V \tag{8.11}$$

$$q[v] = \text{inv_logit}(x[v]) \qquad v = 1, \ldots, V \tag{8.12}$$

$$Y[v] \sim \text{Bernoulli}(q[v]) \qquad v = 1, \ldots, \text{V}$$

$$x_{\text{person}}[n] = b_2 Sex[n] + b_3 Income[n] + b_{\text{person.diff}}[n] \qquad n = 1, \ldots, N \tag{8.13}$$

$$b_{\text{person.diff}}[n] \sim \mathcal{N}\big(0, \sigma_{\text{person}}\big) \qquad n = 1, \ldots, N$$

$$x_{\text{visit}}[v] = b_4 Discount[v] \qquad v = 1, \ldots, V \tag{8.14}$$

where N represents the total number of the customers, and n represents the index of each customer. V represents the total visits (= 444 in this example), and v represents the index of each visit. $v2n[v]$ returns the person index n that the v-th visit belongs to. $x_{\text{person}}[n]$ in Eq. (8.13) is the person-specific term determined by the combination of Sex and $Income$, plus the person difference. Similarly, $x_{\text{visit}}[v]$ in Eq. (8.14) is the visit-specific term, and is proportional to the explanatory variable $Discount$. The factors are added up and transformed to the purchase probability $q[v]$ in Eqs. (8.11) and (8.12).

Note that we did not include the intercept term for either Eqs. (8.13) and (8.14) but only for Eq. (8.11). This is because if we include the intercept term in either Eqs. (8.13) or (8.14), when you substitute Eq. (8.13) and (8.14) into Eq. (8.11), a term

like $\text{intercept}_1 + \text{intercept}_2$ will appear. As mentioned in Sect. 7.4 of multicollinearity, MCMC would not converge because of this term. We will further discuss this situation in Sect. 9.1.

8.5.4 Implement Models

The implementation of Model Formula 8.8 is as follows.

model8-8.stan

```
 1  data {
 2    int N;
 3    int V;
 4    vector[N] Sex;
 5    vector[N] Income;
 6    array[V] int<lower=1, upper=N> v2n;
 7    vector[V] Dis;
 8    array[V] int<lower=0, upper=1> Y;
 9  }
10
11  parameters {
12    vector[4] b;
13    vector[N] b_person;
14    real<lower=0> s_person;
15  }
16
17  transformed parameters {
18    vector[N] x_person = b[2]*Sex[1:N] + b[3]*Income[1:N]+
      b_person[1:N];
19    vector[V] x_visit = b[4]*Dis[1:V];
20    vector[V] x = b[1] + x_person[v2n] + x_visit[1:V];
21    vector[V] q = inv_logit(x[1:V]);
22  }
23
24  model {
25    b_person[1:N] ~ normal(0, s_person);
26    Y[1:V] ~ bernoulli(q[1:V]);
27  }
```

Line 6: $v2n$ is declared as `v2n`.

Line13: $b_{\text{person.diff}}[n]$ is declared as `b_person`.

Line14: σ_{person} is declares as `s_person`.

In addition, lines 18, 19, and 20 correspond to Eqs. (8.13), (8.14) and (8.11), respectively.

Here we omit R and Python code that run Stan. Please refer **`run-model8-8.R`** and **`run-model8-8.py`** available on the supporting page of this book.

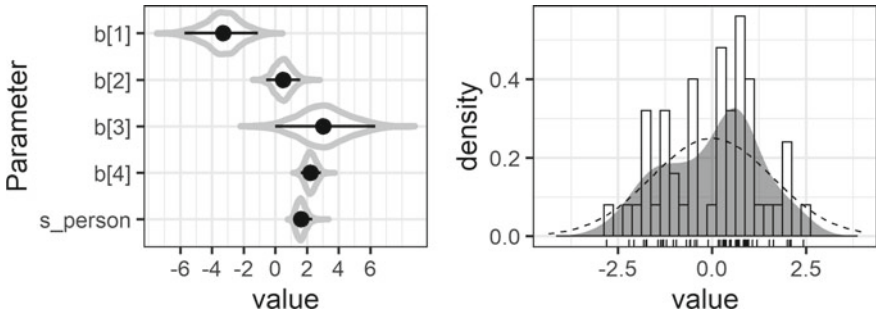


Fig. 8.10 Left: the violin plot of the posterior distribution for each of `b[1]`,…,`b[4]` and `s_person` in **`model8-8.stan`**. The black solid lines are the 95% Bayesian confidence intervals and the dots are the medians. Right: PRC. the histogram of MAP estimates for `b_person[1]`,…,`b_person[50]` and the density function was fitted. The dotted line is a normal distribution with mean 0 and SD equals to the MAP estimate of `s_person`. The y-axis is adjusted to the scale of the histogram

8.5.5 Interpreting Results

From the MCMC sample, we visualized the posterior distribution for each parameter and the posterior residual (Fig. 8.10).

From the distribution of b_4 in Fig. 8.10 (left), we see that the *Discount* has a positive effect for the purchase probability. The distribution of b_3 is relatively wide. The fact that there is a high possibility that σ_{person} is larger than 1[2] suggests relatively large person difference. From Fig. 8.10 (right), we can see that the highest MAP estimate of the purchase probability of the person is more than 4 larger than the lowest one. From the perspective of odds, this is about $\exp(4) \approx 55$ folds of difference between them.

Using ROC curve, we can compute the AUC score 0.841 (with the 80% Bayesian confidence intervals [0.828, 0.852]). We can say that this is an improvement compared to 0.688, which is the result of the model that did not account for the differences in persons.

As shown in this section, hierarchical models have this flexibility to be combined with other models such as logistic regression.

8.6 Exercises

(1) Check the data distribution in Sect. 8.5. In particular, check the person differences by computing the purchase proportion for each person and visualize the results using histogram.

[2] If we compute from the MCMC sample, $\text{Prob}\left[\sigma_{\text{person}} > 1\right]$ was 0.995 in this model.

(2) Implement the model in Sect. 8.5 without using the hierarchical model. In other word, use a noninformative prior instead of a normal distribution for the prior of $b_{\text{person.diff}}[n]$.
(3) Suppose we have data that evaluated mathematical scores of students in three countries ("`data-ex4.csv`" in our GitHub repository). Each country includes 6 schools and each school includes 20 students. Implement a hierarchical model. We assume that there are school differences, and also assume that the SD of measurement noise and SD of school differences are shared across countries. Then estimate the mean scores of three countries. Here, we do not consider country differences because we want to estimate them without shrinkage.

One of the greatest books that provides in-depth discussion on hierarchical model is *Data Analysis Using Regression and Multilevel/Hierarchical Models,* 2007. There is also a plan for releasing a second version of this book with Stan implementation.[3] *Bayesian Population Analysis Using WinBUGS: A Hierarchical Perspective*, 2011, also has detailed explanations for state-space models and hierarchical models with the applications focused on ecology. The book also provides a large amount of simulation examples using R. The original BUGS codes in these two books were also transformed to Stan codes and are available from the official GitHub repository.[4]

References

Gelman, A., & Hill, J. (2007). *Data analysis using regression and Multilevel/Hierarchical models.* Cambridge University Press.

Kery, M., & Schaub, M. (2011). *Bayesian population analysis using WinBUGS: A hierarchical perspective.* Academic Press.

[3] http://andrewgelman.com/2015/06/11/applied-regression-and-multilevel-modeling-books-using-stan/.

[4] https://github.com/stan-dev/example-models.

Chapter 9
How to Improve MCMC Convergence

When modeling real-world data, MCMC may have poor convergence, which will make the calculation speed of sampling very slow. Poor convergence is usually due to the model itself, rather than software problems. In this chapter, we discuss four situations as potential causes of poor convergence, and provide the solutions to improve the model under each situation.

- Section 9.1: Some parameters are nonidentifiable. In such a case, we suggest finding out the problematic parameters by carefully checking and rearranging the model formula. Usually, poor convergence can be improved by eliminating some parameters or fixing some of them to be constants.
- Section 9.2: The model is too complicated whereas the amount of data is relatively small, or the parameters do not have constraints. This usually can be improved by specifying weakly informative priors for these parameters.
- Section 9.3: Strong dependencies between parameters cause their posterior distributions to be distorted. Reparameterization would be an option to solve this issue.
- Section 9.4: Other cases. For instance, the model incorporates a function that has bumpy derivatives, or obtaining posterior distributions is difficult due to having too many local optima. Sometimes, rearranging the model formula or using other advanced techniques can help solve these issues.

9.1 Removing Nonidentifiable Parameters

In Sect. 9.1.1, we first explain what identifiability of parameter is. In the following Sects. 9.1.2–9.1.5, we provide some example models with nonidentifiable parameters, and illustrate how to build a model ensuring its parameter identifiability.

© Springer Nature Singapore Pte Ltd. 2022

K. Matsuura, *Bayesian Statistical Modeling with Stan, R, and Python*,
https://doi.org/10.1007/978-981-19-4755-1_9

9.1.1 Parameter Identifiability

In a statistical model, we say a parameter $\overrightarrow{\theta}$ is *identifiable*, if it satisfies:

$$p\left(y|\overrightarrow{\theta_1}\right)p\left(\overrightarrow{\theta_1}\right) = p\left(y|\overrightarrow{\theta_2}\right)p\left(\overrightarrow{\theta_2}\right) \text{ if and only if } \overrightarrow{\theta_1} = \overrightarrow{\theta_2}$$

for any y. In other words, if there is any combination of $\left(\overrightarrow{\theta_1}, \overrightarrow{\theta_2}\right)$ and $\overrightarrow{\theta_1} \neq \overrightarrow{\theta_2}$ such that $p\left(y|\overrightarrow{\theta_1}\right)p\left(\overrightarrow{\theta_1}\right) = p\left(y|\overrightarrow{\theta_2}\right)p\left(\overrightarrow{\theta_2}\right)$, then we say parameter $\vec{\theta}$ is *nonidentifiable*.

Let us look at an example of nonidentifiable parameters. We start with a simple univariate regression model:

Model Formula A

$$\begin{aligned} Y[n] &= a + bX[n] + \varepsilon[n] && n = 1, \ldots, N \\ \varepsilon[n] &\sim \mathcal{N}(0, \sigma) && n = 1, \ldots, N \end{aligned}$$

For this model, let's modify it a bit by considering it in another way: instead of assuming $\varepsilon[n]$ is generated from a normal distribution with mean zero and standard deviation σ, assuming $\varepsilon[n]$ is generated from a normal distribution with mean $cX[n]$ and standard deviation σ by adding another parameter c. We can then write Model Formula A in a mathematically equivalent way:

Model Formula B

$$\begin{aligned} Y[n] &= a + bX[n] + cX[n] + \varepsilon'[n] \\ &= a + (b + c)X[n] + \varepsilon'[n] && n = 1, \ldots, N \\ \varepsilon'[n] &\sim \mathcal{N}(0, \sigma) && n = 1, \ldots, N \end{aligned}$$

We specify the same noninformative priors for a, b., and c as before, and estimate their posterior distributions from data. As an example, let us assume that we estimated the slope parameter $b = 3.0$ from Model Formula A. Consequently, fitting the same data using Model Formula B would only require satisfying the slope $(b + c) = 3.0$, for which the number of solutions is infinite. Therefore, parameters b and c cannot be fixed to be certain values, namely b and c are nonidentifiable, therefore MCMC for Model Formula B will not converge. Removing either b or c would help solve this problem. Or, we can specify different priors for b and c, inrder to assure their identifiability.

As shown here, when many effects are incorporated into the model, we may accidently introduce nonidentifiable parameters without even realizing it. In order to

avoid this, it is always a good idea to write down each model formula carefully and rearrange them wherever possible, and check if there's any nonidentifiable parameters introduced.

9.1.2 Individual Difference

In this subsection, we discuss cases where parameters are nonidentifiable because they cannot be discriminated from noise terms.

Recall the following multiple regression model discussed in Sect. 5.1:

$$\begin{aligned} Y[n] &= b_1 + b_2 Sex[n] + b_3 Income[n] + \varepsilon[n] && n = 1, \ldots, N \\ \varepsilon[n] &\sim \mathcal{N}(0, \sigma) && n = 1, \ldots, N \end{aligned}$$

Based on this model, here we further consider that the purchase proportion includes the customer difference. By incorporating this term, the model can be written as:

$$\begin{aligned} Y[n] &= b_1 + b_2 Sex[n] + b_3 Income[n] + b_{\text{diff}}[n] + \varepsilon[n] && n = 1, \ldots, N \\ b_{\text{diff}}[n] &\sim \mathcal{N}(0, \sigma_b) && n = 1, \ldots, N \\ \varepsilon[n] &\sim \mathcal{N}(0, \sigma) && n = 1, \ldots, N \end{aligned}$$

If σ_b and σ follow the same noninformative prior distribution, then $b_{\text{diff}}[n]$ and $\varepsilon[n]$, which have the common index, will follow the same distribution, namely they are nonidentifiable parameters. In this example, there's only one data point for each individual. Hence for the effect that cannot be explained by covariates, it is not feasible to split it into the effects by customer difference and by noise. As discussed in the previous example, it is not very difficult to detect these problems if we write down the model formula and rearrange it carefully.

9.1.3 Label Switching

Another other case that causes parameter nonidentifiability is when parameters have interchangeable indices.

Let's consider that we are trying to fit a mixture of normal distributions to data that exhibits a bimodal distribution (Fig. 9.1). A mixture of normal distributions is a distribution that includes multiple components of normal distributions. For this data, we mix two normal distributions, $\mathcal{N}(\mu_1, \sigma_1)$ and $\mathcal{N}(\mu_2, \sigma_2)$, with mixing proportions a and $(1 - a)$. This can be written as:

$$\text{Normal_Mixture}(y|a, \mu_1, \sigma_1, \mu_2, \sigma_2) = a \times \mathcal{N}(y|\mu_1, \sigma_1) + (1 - a) \times \mathcal{N}(y|\mu_2, \sigma_2)$$

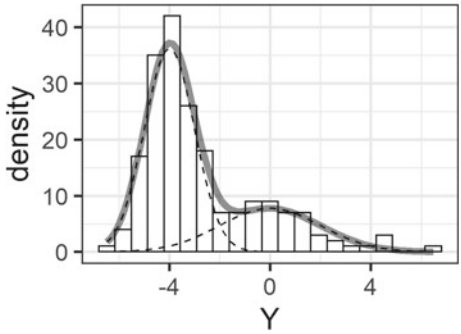


Fig. 9.1 An example of a mixture of two normal distributions. The gray solid line is the probability density function of Normal_Mixture$(y|0.3, 0, 2, -4, 1)$. The two black dotted lines represent $0.7\mathcal{N}(-4, 1)$ and $0.3\mathcal{N}(0, 2)$. The histogram represents the random draws drawn from the density function of this mixture model

where the prior distributions of μ_1 and μ_2, as well as σ_1 and σ_2, are specified to be the same noninformative priors. The problem here is that the models with $(a, \mu_1, \sigma_1, \mu_2, \sigma_2) = (0.3, 0, 2, -4, 1)$ and $(a, \mu_1, \sigma_1, \mu_2, \sigma_2) = (0.7, -4, 1, 0, 2)$ represent two equivalent mixture models, and give identical posterior distributions. Consequently, this again causes parameter nonidentifiability. In other words, either $\mathcal{N}(\mu_1, \sigma_1)$ or $\mathcal{N}(\mu_2, \sigma_2)$ can be the component on the left-hand side of the mixture distribution.

This problem is called *label switching*. MCMC for this model may cause one or more of the following issues:

- A single chain of MCMC sequence keeps shifting between the parameter space near $(0.3, 0, 2, -4, 1)$ and near $(0.7, -4, 1, 0, 2)$. The sampling would not go well due to this inconstancy.
- Estimated values of (μ_1, σ_1) and (μ_2, σ_2) would be very close, and two components of normal distributions would merge to one.
- Because each chain will converge to different parameter space, $\widehat{R}$ cannot be calculated properly.

In a Stan code, a solution for label switching is to use `ordered` type for parameter specification. An `ordered` type is a special case of a `vector` type. The elements of an `ordered` type of length N satisfy $x_1 < x_2 < \cdots < x_N$. For this mixture model specifically, by declaring the two parameters μ_1 and μ_2 as "`ordered[2] mu`", their relationships can be controlled as $\mu_1 < \mu_2$. Therefore, $\mathcal{N}(\mu_1, \sigma_1)$ is forced to represent the component on the left-hand side of the mixture distribution, and only the parameter $(0.7, -4, 1, 0, 2)$ would be estimated in the end. Nonetheless, solving a label switching problem is always challenging in general.

9.1.4 Multinomial Logistic Regression

There is another common situation that leads to parameter nonidentifiability. That is, the parameter values are determined by their relative relationship to the other parameters.

Recall the example used in Sect. 6.6: 300 customers' purchase data (Data File 9.1, a row corresponds to a customer). Each column represents:

Age: Age (Unit: years)

Sex: A binary value that represents gender of the customer (0: female, 1: male)

Income: Annual income (Unit: $1,000)

Y: Category index of the purchased product (an integer between 1 and K, here $K = 6$)

Data File 9.1 Structure of `data-category.csv`

```
  1  Age,Sex,Income,Y
  2  18,1,47.2,2
  3  18,0,46.8,5
  4  18,1,45.1,6
     ...
301  55,0,79.0,4
```

The aim of this analysis is to understand which features of customers (age, sex, and annual income) would affect purchase choices for each product category. For this purpose, we consider a multinomial logistic regression model. That is, we specify the probability of choosing each product category $\vec{\theta}$ to be determined by a linear combination of the age, sex, and annual income:

Model Formula 9.1

$$\overrightarrow{\mu[n]} = \overrightarrow{b_1} + \overrightarrow{b_2} Age[n] + \overrightarrow{b_3} Sex[n] + \overrightarrow{b_4} Income[n] \qquad n = 1, \ldots, N$$

$$\overrightarrow{\theta[n]} = \text{softmax}\left(\overrightarrow{\mu[n]}\right) \qquad n = 1, \ldots, N$$

$$Y[n] \sim \text{Categorical}\left(\overrightarrow{\theta[n]}\right) \qquad n = 1, \ldots, N$$

where $\overrightarrow{\mu[n]}$, $\overrightarrow{\theta[n]}$, $\overrightarrow{b_1}$, $\overrightarrow{b_2}$, $\overrightarrow{b_3}$ and $\overrightarrow{b_4}$ are vectors of length K $\overrightarrow{b_1}$, $\overrightarrow{b_2}$, $\overrightarrow{b_3}$ and $\overrightarrow{b_4}$ are the parameters to be estimated from the data. As mentioned in Sect. 6.6, a handy way to construct a simplex $\vec{\theta}$ from a linear combination of the covariates in the range of $(-\infty, \infty)$ is using the *softmax* function to transform the variables:

$$\vec{\theta} = \text{softmax}(\vec{\mu}) = \left(\frac{\exp(\mu_1)}{\sum_{k'=1}^{K} \exp(\mu_{k'})} \cdots \frac{\exp(\mu_K)}{\sum_{k'=1}^{K} \exp(\mu_{k'})} \right)^T$$

where each element in $\vec{\mu}$ representshe "intensity" that a customer will purchase the product in a certain category. Yet, is not hard to find that these are nonidentifiable

parameters. For an illustration, let us see what happens if we replace μ_k for all the categories with $\mu_k + 1$:

$$\theta_k = \frac{\exp(\mu_k + 1)}{\sum_{k'=1}^{K} \exp(\mu_{k'} + 1)} = \frac{\exp(\mu_k)\exp(1)}{\sum_{k'=1}^{K} \left[\exp(\mu_{k'})\exp(1)\right]} = \frac{\exp(\mu_k)}{\sum_{k'=1}^{K} \exp(\mu_{k'})}$$

Note that this is the same equation as the original one. This suggests that shifting all μ_k by the same amount does not change the value of $\overrightarrow{\theta}$, and hence each μ_k is nonidentifiable.

There are several ways to modify these parameters so that they can be identifiable. Here, we fix the intensity of choosing the first category to be 0. The intensity of choosing the rest of the categories, hence, can be determined by comparing them to the intensity of choosing the first category. For implementation, we only need to define $\mu_1 = 0$. This can be implemented by fixing $b_{1,1} = b_{2,1} = b_{3,1} = b_{4,1} = 0$, where $b_{1,1}$ denotes the first element of the vector $\overrightarrow{b_1}$ (the same applies for $b_{2,1}$, $b_{3,1}$ and $b_{4,1}$).

The implementation of Model Formula 9.1 is as follows:

model9-1.stan

```
1  data {
2    int N;
3    int D;
4    int K;
5    matrix[N,D] X;
6    array[N] int<lower=1, upper=K> Y;
7  }
8
9  transformed data {
10   vector[D] Zeros = rep_vector(0,D);
11 }
12
13 parameters {
14   matrix[D,K-1] b_raw;
15 }
16
17 transformed parameters {
18   matrix[D,K] b = append_col(Zeros, b_raw);
19   matrix[N,K] mu = X*b;
20 }
21
22 model {
23   for (n in 1:N) {
24     Y[n] ~ categorical(softmax(mu[n,]'));
25   }
26 }
```

Lines 3, 5, and 19 are similar to **model5-7.stan** in Sect. 5.6. That is, D in line 3 represents "1+ the number of the explanatory variables", X in line 5 represents the

matrix of the explanatory variables including the intercept term. And in line 19, $\overrightarrow{\mu[n]}$, which is the product of the explanatory variables and the coefficients, is calculated by the matrix operation. Let us take a closer look at the rest of the code.

Line 6: `Y` is an array, which stores the category index of product each customer chose.

Lines 9–11: The `transformed data` block is used to generate data within a Stan code. Sometimes it is not straightforward to transform data using R, or it is necessary to further process the data usi certain Stan functions. Using this block can be very helpful in these cases. In this code, a zero vector is generated in order to obtain $b_{1,1} = b_{2,1} = b_{3,1} = b_{4,1} = 0$. The function `rep_vector(x, m)` generates a (column) vector that contains `m` replicas of a real number `x`.

Line 14: `b_raw` contains coefficients of $\overrightarrow{b_1}$ to $\overrightarrow{b_4}$ for all the categories except $k = 1$. It is declared to be a matrix type, and will be estimated from the model.

Line 18: A `D` × `K` matrix `b` is created by horizontally stacking the column of 0 and a `D` ×`(K-1)` matrix `b_raw`, using a Stan function `append_col`. When we want to obtain parameters that include certain constant values, using `transformed parameters` block would be helpful.

Line 24: The variable is transformed to a simplex by `softmax` function. Note that only a `vector` type can be used as the argument of `softmax` function. Because `mu[n,]` is the `n`-th row of the matrix, namely it is a `row_vector`, here we transpose it to be a (column) `vector` type using a single quotation operation "`'`". In Stan, another convenient function with a stable, fast performance, is called `categorical_logit`. It implements `softmax` function internally. Using this function, line 24 can be simplified as:

```
Y[n] ~ categorical_logit(mu[n,]');
```

9.1.5 The Tortoise and the Hare

Similar to the previous section, we will discuss another example of parameter nonidentifiability, in which the parameter values are determined by their relative relationship to other parameters.

Let us look at the results from games where a tortoise and a hare competed in races. Consider we are to estimate their running capabilities, and assume there is a comma-separated file that records 30 games they competed (Data File 9.2, a row corresponds to a game). Each column represents:

Loser: Index of the player who lost the game.
Winner: Index of the player who won the game.

For the player index, we use 1 to denote the tortoise and 2 to denote the hare. Aggregating the results, the tortoise has 4 wins and 26 losses for games. We can simply pass this aggregated result to the model. But instead, here we use the original

data without aggregating, for the sake of model extensions that will be discussed in the later sections.

Data File 9.2 Structure of `data-tortoise-hare.csv`

```
Loser,Winner
1,2
1,2
2,1
...
1,2
```

Now let us consider possible mechanisms that generate this data set. We denote the running capabilities of the tortoise and the hare as $\mu[1]$ and $\mu[2]$, respectively. For these two players, we use "performances" to represent their capabilities exhibited in each game, and the comparison of their performances in each game determines its result. In other words, we assume that the likelihood of a game is "the probability that the winner has better performance than the loser".

Assume that the performance of the tortoise is generated from a normal distribution with mean $\mu[1]$ and SD $\sigma[1]$. Because $\sigma[1]$ is the variability of his performances exhibited across games, this can be considered as the tortoise's performance fluctuation. The same applies to that of the hare's. Because the amount of data is relatively small, we assume the fluctuations of the tortoise and the hare is equal, i.e., $\sigma[1] = \sigma[2] = \sigma$.

When we consider the likelihood for this model, an important point to be aware of is that the difference of the players' performances is the only factor that determines a game result. In other words, even if $\mu[1]$ and $\mu[2]$ change simultaneously, as long as their difference (i.e., $\mu[1] - \mu[2]$) remains the same, the likelihood for the game results would also remain the same. Therefore, in order to identify the parameters and to understand the players' capabilities, we fix one of these parameters to be zero, by letting $\mu[1] = 0$. Yet, without determining the scale of the fluctuations of the performances, the parameters in this model are still nonidentifiable. For instance, $(\mu[2], \sigma) = (3, 2)$ and $(\mu[2], \sigma) = (6, 4)$ would yield equivalent likelihood for the game result. Therefore, we further fix $\sigma = 1$. These can be expressed in the model formula as:

Model Formula 9.2

$$
\begin{aligned}
l_{\text{game}} &= \prod_{g=1}^{G} \text{Prob}\big[\mathit{performance}_L[g] < \mathit{performance}_W[g]\big] \\
\mathit{performance}_L[g] &\sim \mathcal{N}(\mu[g2L[g]], 1) \qquad g = 1, \ldots, G \\
\mathit{performance}_W[g] &\sim \mathcal{N}(\mu[g2W[g]], 1) \qquad g = 1, \ldots, G \\
\mu[1] &= 0
\end{aligned}
$$

where G denotes the total number of the games, and g denotes the index of each game. The variable l_{game} represents the likelihood of the results of the games, and $performance_L[g]$ and $performance_W[g]$ represent the performance of the loser and the winner for game g. $g2L[g]$ returns the player index of the loser for the g-th game. Inther words, $g2L$ maps the game index g to the player index of the loser, and equals to the *Loser* column in Data file 9.2. Similarly, $g2W[g]$ returns the player index of the winner for the g-th game, and $g2W$ equals to the *Winner* column Data file 9.2. Using this model, we can estimate $\mu[2]$ from the data.

Because we do not need to know the players' performance for each game, we can eliminate both $performance_L[g]$ and $performance_W[g]$. For game g, we use y to denote the difference in performances between two players, $performance_W[g] - performance_L[g]$. Then, from the reproductive property of a normal distribution (Sect. 6.12), the distribution of y can be written as:

$$y \sim \mathcal{N}\Big(\mu[g2W[g]] - \mu[g2L[g]], \sqrt{2}\Big)$$

Therefore, we can express the likihood derived from the result of game g. In other words, we can express the probability of the winner winning the game, $\text{Prob}[y > 0]$, as follows:

$$\text{Prob}[y > 0] = 1 - \Phi\left(\frac{0 - (\mu[g2W[g]] - \mu[g2L[g]])}{\sqrt{2}}\right).$$

where Φ is the cumulative distribution function of the standard normal distribution. If we calculate this probability for each game and log-transform it, and sum up these log probabilities over all the games, we can obtain the log-likelihood for all game results. In Stan, this can be done by adding the likelihood for each game to `target` (also see Sects. 3.6 and 7.7).

The implementation of Model Formula 9.2 is as follows:

model9-2.stan

```
data {
   int N; // num of players
   int G; // num of games
   array[G] int<lower=1, upper=N> g2L; // loser of each game
   array[G] int<lower=1, upper=N> g2W; // winner of each game
}

parameters {
  real b;
}

transformed parameters {
  vector[N] mu = [0, b]';
}

```

```
16 model {
17   for (g in 1:G) {
18    target += normal_lccdf(0 | mu[g2W[g]] - mu[g2L[g]], sqrt(2));
19   }
20 }
```

Lines 4–5: $g2L$ is declared as `g2L`, which is an array of length `G`. Similarly, $g2W$ is declared as `g2W`.

Line 13: `mu` represents the capability of the players (`mu[1]` and `mu[2]` correspond to the capability of the tortoise and the hare, respectively) "`[]`" can make a row vector by stacking basic types. This shows an example where we use `transformed parameters` block to assign some of the elements in a `vector` as constants, and the rest as parameters. By declaring `mu` here, we can make the expression in line 18 simpler.

Line 18: `normal_lccdf(y|mu,sigma)` is a convenient Stan function representing $\log\left[1 - \Phi((y-\mu)/\sigma)\right]$.[1]

From the data, we can estimate `b`, which represents relative capability of the hare when capability of the tortoise is set to be zero. The estimated median and 95% Bayesian confidence interval are 1.60 (0.82, 2.44).

9.2 Use Weakly-Informative Priors to Restrict the Posterior Distributions

Oftentimes, when the amount of data is relatively small compared to model complexity, we might confront poor convergence issues. One way to avoid this is starting from a simpler model and making a small adjustment at a time. This would help find out which added assumptions are problematic and causing the poor convergence. Yet, in order to achieve the purpose of the analysis, sometimes it is necessary to use a model with high complexity in which MCMC does not converge well easily. In other cases, in addition to the poor convergence, estimated parameter values are far away from what we would expect based on our knowledge. This is largely because no prior constraints are added to the parameters. To solve these issues, instead of using a noninformative prior, we can alternatively use a *weakly informative prior* that induces a slight constraint on the parameter. The intuition behind this is to keep the parameter distribution stay in a certain range, rather than sampling posterior from a very broad range of parameter space.

When using a weakly informative prior, we suggest making slight adjustments to distribution functions and shape parameters each time. In this way, we can check the robustness of the model after its implementation, and it also supports the objectivity of the analysis.

[1] lccdf stands for Log Complementary Cumulative Distribution Function.

In the next sessions (from Sects. 9.2.1–9.2.4), we will illustrate some examples of building models using weakly informative priors.

9.2.1 Weakly Informative Prior for Parameters in $(-\infty, \infty)$

For a parameter in the range $(-\infty, \infty)$, it is common to assign a noninformative prior that is extremely flat, such as Normal(0, 100). In some cases, however, we may have background knowledge about the parameters and thus need to assign weakly informative priors to them. Here we show two such examples.

Parameters are likely to be in a certain range $[a, b]$

Suppose that we have high confidence saying that a parameter is in the interval $[a, b]$. With this information, probably the first thing that comes to our mind is using a uniform distribution Uniform(a, b) as a weakly informative prior. One caveat of using a uniform distribution is that estimated results could be sensitive to the range of the distribution we initially define. That is, depending on where we set the border of the distribution (i.e., defining a and b), MCMC sample might be lingering around at the borders. It is therefore crucial to check the posterior density shape and make sure that the parameter distributions do not have abrupt cutoffs at the interval borders.

Because of this limitation of using Uniform(a, b), Stan development team recommends using a distribution such as a normal distribution with mean $(a + b)/2$ and SD $(b - a)/2$, which does not have an upper or lower bound. Taking the integral of this normal distribution over the interval $[a, b]$ (from $-$SD to $+$SD) yields probability 0.68, which means that there is 0.32 possibility left outside of the interval $[a, b]$.

Regression coefficients

When fitting regression models, sometimes the results can be unrealistic or overfitting when the absolute values of coefficients are extremely large. In these cases, we can specify weakly informative priors for these coefficients. By adding penalty terms to those unrealistic values, we can restrain their estimates. This is a common strategy for logistic regression model. For weakly informative priors, Stan development team suggests using Student's t distribution (Student_t$(\upsilon = \upsilon_0, \mu = 0, \sigma = \sigma_0)$), where υ is the degree of freedom that is fixed to be a constant (υ_0) in the range [3, 7]. We can set the distribution mean $\mu = 0$. If we have the background knowledge which indicates that the absolute values of the coefficients are unlikely to be larger than σ_0, scale σ can be determined to be a constant σ_0. Subsequently, when the degree of freedom is assigned to be 4, then integrating Student_t$(4, 0, \sigma_0)$ over range $[-\sigma_0, \sigma_0]$ gives probability 0.63, meaning that there is still 0.37 probability left outside of the range $[-\sigma_0, \sigma_0]$.

When it is necessary to further strengthen the constraint, we can use a normal distribution with mean zero and standard deviation σ_0. From machine learning perspective, using this normal distribution counterparts to applying ridge regression, which uses the sum of squares of the coefficients as a penalty. For instance, if

we specify $\mathcal{N}(0, 10)$ as a weakly informative prior for the coefficient parameter b, this is equivalent to adding $-0.5 \times \sum(b/10)^2$ to the log posterior probability (see Sect. 3.6).

9.2.2 Weakly Informative Prior for Parameters with Positive Values

The first choice of the prior for positive parameters is a distribution that is flat and wide enough (a noninformative prior), such as Uniform(0, 1000). However, depending on the situation (for example, the amount of data is very small), the posterior distribution may have extremely long tail, in which case we may prefer using a weakly informative prior. For illustration, we show examples with SD and other positive parameters.

SD

Previously, conjugate priors were mainly used for SD because computation methods were limited to analytical calculation. Therefore, an inverse-gamma distribution was frequently used as a noninformative or a weakly informative prior for a variance parameter (instead of SD). However, using an inverse-gamma distribution for variance can potentially cause problems. For instance, we might want to build a hierarchical model to estimate the individual differences. If the data contains only the small individual differences and thus the variance is close to 0, then using an inverse-gamma distribution as a prior may result in the lack of the robustness in the analysis (Gelman, 2006).

This is a consequence of the substantial change in the shape of the posterior distribution near zero on the x-axis, under different hyperparameter settings. To explain this point, two inverse-gamma distributions with different hyperparameters (invGamma(0.1, 0.1) and invGamma(0.001, 0.001)) are shown (Fig. 9.2). As an inverse-gamma function inevitably passes through the origin, the probability densities of two distributions differ largely around zero. Although both of these two inverse-gamma distributions have been widely used, these two plots suggest that depending on the choice of hyperparameters, there is a high risk that they would lead to entirely different estimation results. This also applies to a lognormal distribution when it is used as a prior distribution.

Here we introduce a way to avoid this problem. We specify a half-Student's t distribution (or half-t distribution, Student_t$^+(\upsilon = \upsilon_0, \mu = 0, \sigma = \sigma_0)$, Fig. 9.3) as a prior for SD rather than for variance. Because a half-t distribution contains enough probability density around zero, the aforementioned issue of an inverse-gamma distribution can be avoided. Degree of freedom for the half-t distribution is usually set to be 3 or 4. For scale parameter σ, we set a constant σ_0 with the background knowledge that SD is highly likely to be less than σ_0. Alternatively, a half-normal distribution ($\mathcal{N}^+(0, \sigma_0)$)) can be used when stronger prior information

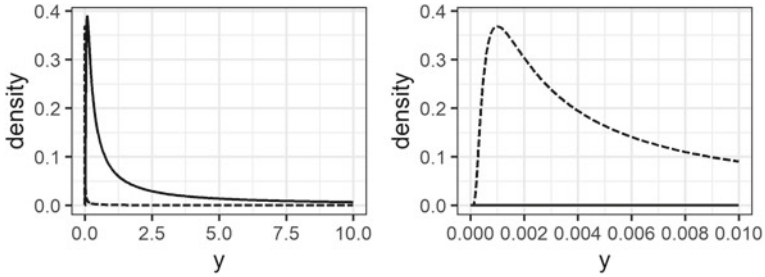


Fig. 9.2 Probability density function of two inverse-gamma distributions. invGamma(0.1, 0.1) and invGamma(0.001, 0.001) are shown as the solid and dotted curves, respectively. Left: probability density in the range [0, 10]. Right: the zoomed-in plot of the left plot in the range of [0, 0.01]

is needed. In a Stan code, we can declare parameters in the `parameters` block by specifying `real <lower = 0>` to representistbutions that are bounded in the positive range.

Let's look at an example where we apply this weakly informative prior on a hierarchical model. Recall **`model8-4.stan`** in Sect. 8.1 where we used a noninformative prior for parameter σ_a (`s_a`) (σ_a represents the SD of the company differences in baseline income for zero-experienced employees). As the estimation result, the median and 95% Bayesian confidence interval were 9.9 [1.6, 64.1]. Note the long tail on the right side on the histogram of MCMC sample (Fig. 9.4 left). Now suppose that we have a piece of background information, which is, in this field of business, the SD σ_a is \$10 K at most. With this piece of information, to specify a weakly informative prior for the SD, we add the following statement into the model:

$$\sigma_a \sim \text{Student_t}^+(4, 0, 10)$$

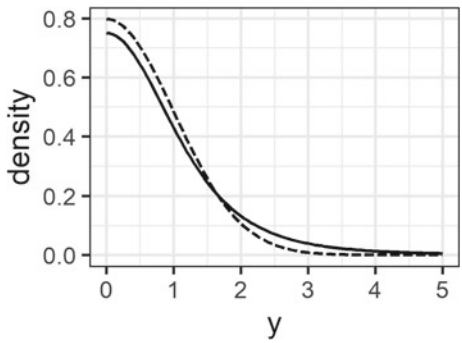


Fig. 9.3 Half-t distribution and half-normal distribution. Probability density function of Student_t$^+$(4, 0, 1), and $\mathcal{N}^+(0, 1)$ are shown as the solid and dotted curves, respectively. Integral of y from 4 to infinity yields 0.016 for Student_t$^+$(4, 0, 1), whereas that of $\mathcal{N}^+(0, 1)$ is 6.3×10^{-5}

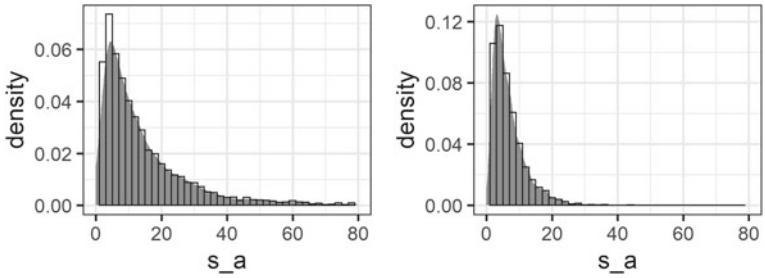


Fig. 9.4 Histogram of MCMC sample and density function for σ_a (`s_a`), with a uniform distribution as a noninformative prior (left), and a half-t distribution as a weakly informative prior (right)

With this change, we implemented the model, and obtained the histogram of MCMC sample of σ_a (`s_a`) as shown in Fig. 9.4 (right). Even though a half-t distribution adds only a weak constraint for the prior distribution, the pathological tail was compressed. Besides, the median and 95% interval have become 5.4 [1.1, 20.5]. For this estimation, we suggest checking model robustness by fitting models under different prior settings of half-t distributions, and see if we get consistent results.

Other cases

In addition to SD, there are other parameters that are bounded to be positive. For instance, a parameter that determines the shape of a Beta distribution takes a positive value. For such a parameter, in addition to a half-t distribution and a half-normal distribution, alternative choices would be an exponential distribution or a gamma distribution. No matter which distributions are being used, visualizing the specified prior distribution and checking if it agrees with what is expected, is always an important step in the modeling process.

As an example, we use an exponential distribution for parameter θ to express its prior distribution as $\theta \sim$ Exponential(β), where parameter β determines the shape of the exponential distribution. We can either fix β to be a constant, or specify it with another probability distribution. Although the former is likely to make the MCMC easier to converge, it is iltpon a stronger assumption and thus is more subjective. For instance, if we believthat the expected value of the parameter is 10, then we can use Exponential(0.1) as its prior.

Similarly, we can use a gamma distribution to specify the parameter distribution as $\theta \sim$ Gamma(α, β). There are multiple ways to specify shape parameter α and rate parameter β. We can fix both of them to be constant values, or alternatively, we can specify probability distribution(s) to either or both of them. For instance, if a positive parameter θ is considered to follow a distribution that can be approximated by a normal distribution with mean $= 1$, and variance $= 0.1$ (SD $= 0.32$), Gamma(10, 10) can be used as a prior. Compared to an exponential distribution, a gamma distribution is more flexible because its distribution shapes have large plasticity. On the other

hand, it involves more complex parameter specifications and sometimes interpreting the prior information could be tricky as well.

As an example of using a gamma distribution as a weakly informative prior, we will use game records of tennis players. We will estimate the capabilities and performance fluctuations for these players. In ball games, "rating systems" are widely used to quantify players' capabilities from multiple games. In a rating system, players transfer their points to their opponent based on the result of each game.[2] Players' ratings reflect their capabilities. Yet there are several limitations for using such systems. One of them is that an accurate evaluation is hardly yielded until the ratings are stabilized from their initial values. Besides, this method is sensitive to the time of the evaluation, and is inclined to the rating inflation too. We will show that these limitations can be overcome by extending Model Formula 9.2 used in Sect. 9.1.5.

We use real game records of professional men's tennis from 2017[3] (175 players[4] and 2380 games), which is saved in a comma-separated file (Data File 9.3, a row corresponds to a game).

Data File 9.3 Structure of `data-tennis-2017.csv`

```
    1   Loser,Winner
    2   169,158
    3   136,101
    4   161,55
    5   ...
 2381   64,37
```

The model we will be building is similar to Model Formula 9.2. That is, the capability of each player is represented by $\mu[n]$ and his performance fluctuations for the games are represented by $\sigma[n]$. We assume that the exhibited performance in each game is generated from $\mathcal{N}(\mu[n], \sigma[n])$. The game results are determined by difference in two players' exhibited performances. In Model Formula 9.2, players' capabilities were the only parameters that were estimated, because the amount of data was not enough for estimating others such as performance fluctuations, which we fixed to be 1. As this is a larger dataset, we will add the players' performance fluctuations $\sigma[n]$ into the estimated parameters, by using a weakly informative prior.

Recall that in Model Formula 9.2, we set the tortoise's capability $\mu[1]$ as the baseline zero. In this tennis players' example, however, the number of players has increased to 175 (whereas it was 2 in Model Formula 9.2), and none of these 175 players has the record of playing with all the other players. Therefore, in this scenario, it is hard to pick one single player as a baseline and set his capability to be zero. Therefore, following the logic in a hierarchical model, we will add a weak constraint to the players' capability parameters $\mu[n]$, by assuming that each of them follows a certain distribution, $\mathcal{N}\left(0, \sigma_\mu\right)$. Also, for the players' performance fluctuations $\sigma[n]$,

[2] One of the most known method is Elo rating system. For detail, see http://en.wikipedia.org/wiki/Elo_rating_system.

[3] http://www.tennis-data.co.uk.

[4] Only the players who had more than five games in the year of 2017 were included.

we use a weakly informative prior Gamma(10, 10), which has mean 1 and standard deviation approximately 0.32. By specifying the model in this way, the performance fluctuations are constrained around 1, thereby the scale of the performance can be determined. With these specifications, even if we do not provide a fixed value for σ_μ, it can still be estimated from the data.

These can be expressed as following equations:

Model Formula 9.3

$$
\begin{aligned}
l_{\text{game}} &= \prod_{g=1}^{G} \text{Prob}\big[performance_L\big[g\big] < performance_W\big[g\big]\big] \\
performance_L\big[g\big] &\sim \mathcal{N}(\mu[g2L[g]], \sigma[g2L[g]]) && g = 1, \ldots, G \\
performance_W[g] &\sim \mathcal{N}(\mu[g2W[g]], \sigma[g2W[g]]) && g = 1, \ldots, G \\
\mu[n] &\sim \mathcal{N}\big(0, \sigma_\mu\big) && n = 1, \ldots, N \\
\sigma[n] &\sim \text{Gamma}(10, 10) && n = 1, \ldots, N
\end{aligned}
$$

where N denotes the number of players and n denotes the index of each player $\mu[n], \sigma[n]$, and σ_μ are the parameters to be estimated from the data. As we did for Model Formula 9.2, $performance_L\big[g\big]$ and $performance_W\big[g\big]$ can be eliminated. Subsequently, the likelihood of single game result can be written as:

$$
1 - \Phi\left(\frac{0 - \big(\mu\big[g2W\big[g\big]\big] - \mu\big[g2L\big[g\big]\big]\big)}{\sqrt{\sigma^2\big[g2L\big[g\big]\big] + \sigma^2\big[g2W\big[g\big]\big]}}\right)
$$

By log-transforming it and adding them to `target`, we can add log-likelihood for a single game. To obtain log-likelihood for the model, we repeat this process for all the games.

The implementation of Model Formula 9.3 is as follows:

model9-3.stan

```
data {
   int N; // num of players
   int G; // num of games
   array[G] int<lower=1, upper=N> g2L; // loser of each game
   array[G] int<lower=1, upper=N> g2W; // winner of each game
}

parameters {
   vector[N] mu;
   real<lower=0> s_mu;
   vector<lower=0>[N] s_pf;
}

model {
    for (g in 1:G) {
```

```
16        target += normal_lccdf(0 | mu[g2W[g]] - mu[g2L[g]],
17          hypot(s_pf[g2L[g]], s_pf[g2W[g]])
18        );
19    }
20    mu[1:N] ~ normal(0, s_mu);
21    s_pf[1:N] ~ gamma(10, 10);
22  }
```

We focus on its difference from **`model9-2.stan`**

Line 9: As the number of players has increased, `mu` ($\mu[n]$) is declared to be a `vector` type. In line 20, we can use a normal function that is compatible with a `vector` type.

Line 11: For the same reason as line 9, `s_pf` ($\sigma[n]$) is declared to be a `vector` type. In line 21, we can use a gamma function that is compatible with a `vector` type.

Line 17: `hypot(a, b)` is a convenient Stan function that represents $\sqrt{a^2 + b^2}$ ($a, b \geq 0$).

From the estimated median of posterior distributions, we picked players that belong to the top five in terms of their capabilities ($\mu[n]$), and players that belong to the top three and the bottom three for the performance fluctuations ($\sigma[n]$) (Table 9.1). As those who are into tennis might find, this result is consistent with our background knowledge about these players. To summarize, our model resolved the shortcomings of the rating systems, and further succeeded in estimating the performance fluctuations.

Table 9.1 A part of the estimation results of the capabilities $\mu[n]$ and performance fluctuations $\sigma[n]$ are listed. The numbers represent the median of the draws for each parameter. Five players with highest capabilities are listed on the left, and three players with highest and lowest fluctuations are listed on the right

Player name	Capability $\mu[n]$	Player name	Fluctuation $\sigma[n]$
Federer R.	2.14	Kyrgios N	1.25
Nadal R.	1.89	Shapovalov D	1.23
Djokovic N.	1.36	Karlovic I	1.19
Zverev A.	1.26		
Del Potro J.M.	1.24	Bautista Agut R	0.70
		Nadal R	0.72
		Chardy J	0.79

9.2.3 Weakly Informative Prior for Parameters in Range [0, 1]

Weakly informative prior options for parameters in the range [0, 1] include a Beta distribution, a Dirichlet distribution and others. In this section, we first show an example using a Dirichlet distribution.

A Dirichlet distribution is often used as a prior when we estimate a K-simplex (i.e., a vector of length K where each element is a real number in the range of [0, 1] and the sum of all elements equals to 1). One simple example is rolling a dice. We can declare `simplex[K] theta` to denote the probability of observing an outcome from a K-sided dice. A `simplex` type is a special case of a `vector` type in Stan. If we do not explicitly assign a prior distribution for `theta`, it will follow a noninformative prior as a default. Specifically, `theta` will follow Dirichlet$\left(\overrightarrow{a_o}\right)$, where $\overrightarrow{a_o}$ represents a K-length vector with all its elements equal to 1. This distribution is equivalent to a uniform distribution, and it indicates that there's no prior information about the shape of the dice. Now, let us suppose that the analysist is aware of how the dice was made, and thus he knows that some sides of the die are more likely to be observed than others. In this scenario, we can use another Dirichlet distribution, Dirichlet$\left(\overrightarrow{a_1}\right)$, as the prior of theta, where $\overrightarrow{a_1}$ represents a K-length vector, with each of its element smaller than 1 (see Sect. 6.8).

In contrast to rolling a dice, tossing a coin only produces two possible outcomes. If the possibility of one outcome is p, then that of alternative outcome is $1 - p$. To assign a weakly informative prior for p, we can use a Beta distribution. Beta(1, 1) and Uniform(0, 1) have identical mathematical expression, whereas Beta(0.5, 0.5) gives another shape of distribution, with higher possibility of p being an extreme value (close to or 1). Using thiprior means that the coin is likely to be distorted (see Sect. 6.5).

9.2.4 Weakly Informative Prior for Covariance Matrix

When we consider that parameters are correlated, we may want to use a multivariate normal distribution and specify a prior for the covariance matrix.

Here, we illustrate it using the hierarchical model with a bivariate normal distribution. Recall the model used in Sect. 8.1:

Model Formula 9.4

$$
\begin{aligned}
Y[n] &\sim \mathcal{N}\big(a[n2c[n]] + b[n2c[n]]\,X[n], \sigma_y\big) && n = 1, \ldots, N \\
a[c] &\sim \mathcal{N}(a_{\text{field}}, \sigma_a) && c = 1, \ldots, C \\
b[c] &\sim \mathcal{N}(b_{\text{field}}, \sigma_b) && c = 1, \ldots, C
\end{aligned}
$$

In this model, $a[c]$ (the baseline income for a zero-experienced employee) and $b[c]$ (the amount of increase in the baseline income accompanied with one additional year of work experience) are specified to be generated from two independent normal distributions. Now suppose that from background knowledge, we assume that these two parameters are not independent, and that a company with a large $a[c]$ has a small $b[c]$. With this new assumption, the model can be written as:

Model Formula 9.5

$$Y[n] \sim \mathcal{N}\big(a[n2c[n]] + b[n2c[n]]\,X[n], \sigma_y\big) \qquad n = 1, \ldots, N \tag{9.1}$$

$$\begin{pmatrix} a[c] \\ b[c] \end{pmatrix} \sim \text{MultiNormal}\left(\begin{pmatrix} a_{\text{field}} \\ b_{\text{field}} \end{pmatrix}, \Sigma\right) \qquad c = 1, \ldots, C \tag{9.2}$$

The parameter Σ represents a covariance matrix, and its value is estimated from the data, with a noninformative prior assigned. The implementation of Model Formula 9.5 is as follows:

model9-5.stan

```
 1  data {
 2     int N;
 3     int C;
 4     vector[N] X;
 5     vector[N] Y;
 6     array[N] int<lower=1, upper=C> n2c;
 7  }
 8
 9  parameters {
10     array[C] vector[2] ab;
11     vector[2] ab_field;
12     cov_matrix[2] cov;
13     real<lower=0> s_y;
14   }
15
16   transformed parameters {
17     vector[C] a = to_vector(ab[1:C,1]);
18     vector[C] b = to_vector(ab[1:C,2]);
19   }
20
21   model {
22     ab[1:C] ~ multi_normal(ab_field, cov);
23     Y[1:N] ~ normal(a[n2c] + b[n2c] .* X[1:N], s_y);
24   }
```

Line 10: The left-hand side of Eq. (9.2) is declared as `ab`.

Lines 11–12: The mean vector of the bivariate normal distribution on the right-hand side of Eq. (9.2) is declared as `ab_field` and the covariance matrix Σ is

declared as `cov` (see Sect. 6.14). Because we do not assign a prior for `ab_field` or `cov`, a noninformative prior will be assigned to them as defaults.

Lines 17–18: `a` and `b` are extracted from `ab` in `transformed parameters` block. Note that `ab[1:C,1]` is an array of `real` type. `to_vector` function can convert a one-dimensional array to a column vector.

Line 22: This line corresponds to Eq. (9.1), using vectorization (see Sect. 6.14).

In fact, the result of **`model9-5.stan`** will still not converge. It is because the amount of data is insufficient, whereas the model is too complicated and there are any restrictions for the parameters. To solve this convergence issue, let's specify a weakly informative prior for the covariance matrix. Previously, an inverse-Wishart distribution was frequently used (Gelman & Jennifer, 2007). An inverse-Wishart distribution is in the form of a multivariate version of an inverse-gamma distribution, and it serves as the conjugate prior for a covariance matrix. However, it has the same limitation as an inverse-gamma distribution, which we discussed in Sect. 9.2.2. Besides, setting parameters for an inverse-Wishart distribution is oftentimes not easy. For these reasons, we will explain two other ways to specify priors for a covariance matrix. First approach is to construct a covariance matrix from the combination of SD and correlation, and the second approach is to use an LKJ correlation distribution, which is recommended by Stan development team.

Constructing a covariance matrix from SD and correlation

We will use the following equation:

$$\Sigma = \begin{pmatrix} \sigma_a^2 & \sigma_a\sigma_b\rho \\ \sigma_a\sigma_b\rho & \sigma_b^2 \end{pmatrix} \tag{9.3}$$

where parameter ρ denotes the correlation between a and$b$. When$\rho = 0$, this model is identical to Model Formula 9.4. By writing the covariance matrix in this way, we can simplify the process of setting weakly informative priors for σ_α and σ_β. Then from σ_α, σ_β and ρ, the covariance matrix Σ is constructed. This approach is also used as an example of reparameterization in the next section.

By specifying weakly informative priors for a_{field} and b_{field}, in addition to σ_α and σ_β, a new model can be written as:

Model Formula 9.6

$$Y[n] \sim \mathcal{N}\big(a[n2c[n]] + b[n2c[n]]X[n], \sigma_y\big) \qquad n = 1, \ldots, N$$

$$\begin{pmatrix} a[c] \\ b[c] \end{pmatrix} \sim \text{MultiNormal}\left(\begin{pmatrix} a_{\text{field}} \\ b_{\text{field}} \end{pmatrix}, \begin{pmatrix} \sigma_a^2 & \sigma_a\sigma_b\rho \\ \sigma_a\sigma_b\rho & \sigma_b^2 \end{pmatrix}\right) \qquad c = 1, \ldots, C$$

$$a_{\text{field}} \sim \mathcal{N}(40, 20) \tag{9.4}$$

$$b_{\text{field}} \sim \mathcal{N}(1.5, 1.5) \tag{9.5}$$

$$\sigma_a \sim \text{ Student_t}^{+}(4, 0, 20) \tag{9.6}$$

$$\sigma_b \sim \text{ Student_t}^{+}(4, 0, 2) \tag{9.7}$$

The explanations for weakly informative priors for Eqs. (9.4)–(9.7) are as follows (see Sects. 9.2.1 and 9.2.2):

- In Eq. (1.4), we added an assumption that a_{field} (mean of a) is likely to be in the range from \$20 to \$60 K.
- In Eq. (1.5), we added an assumption that b_{field} (mean of b) is likely to be in the range from \$0 to \$3 K.
- In Eq. (1.6), we added an assumption that the maximum value of σ_a (SD of a) is likely to be \$20 K.
- In Eq. (1.7), we added an assumption that the maximum value of σ_b (SD of b) is likely to be \$2 K.

The implementation of Model Formula 9.6 is as follows:

model9-6.stan

```
data { same as model9-5.stan }

parameters {
  array[C] vector[2] ab;
  vector[2] ab_field;
  real<lower=0> s_a;
  real<lower=0> s_b;
  real<lower=-1, upper=1> rho;
  real<lower=0> s_y;
}

 transformed parameters {
   vector[C] a = to_vector(ab[1:C,1]);
   vector[C] b = to_vector(ab[1:C,2]);
   matrix[2,2] cov;
   cov[1,1] = square(s_a); cov[1,2] = s_a*s_b*rho;
   cov[2,1] = s_a*s_b*rho; cov[2,2] = square(s_b);
}

 model {
   ab_field[1] ~ normal(40, 20);
   ab_field[2] ~ normal(1.5, 1.5);
   s_a ~ student_t(4, 0, 20);
   s_b ~ student_t(4, 0, 2);
   ab[1:C] ~ multi_normal(ab_field, cov);
   Y[1:N] ~ normal(a[n2c] + b[n2c] .* X[1:N], s_y);
}
```

Lines 6–8: σ_a, σ_b, and correlation ρ are declared as `s_a`, `s_b`, and `rho`, respectively.

Lines 15–17: In `transformed parameters` block, $\mathbf{\Sigma}$ is constructed from σ_a, σ_b, and ρ, corresponding to Eq. (9.3). Note that `cov` is declared as a `matrix` type because constraints for variables (i.e., usage of a `cov_matrix` type) should not be used in this block (see Sect. 4.3).

Lines 21–24: These lines correspond to Eqs. (9.4)–(9.7).

In this model, as there is no prior specified for `rho`, it will be generated from a noninformative prior (i.e., Uniform$(-1, 1)$).

One limitation of this approach is that when the size of the covariance matrix increases (e.g., a 5×5 matrix), it will be hard to specify weakly informative priors for the correlations. Therefore, we provide another approach.

Using an LKJ correlation distribution

For this approach, Student's t distribution is used as a prior for SD, as in **`model9-6.stan`**. In contrast, for a correlation matrix, an *LKJ correlation distribution* is used as a prior. A covariance matrix will be calculated by matrix operation of SD and the correlation matrix. This is a fast and highly practical method. This method can be written as follows:

Model Formula 9.7

$$Y[n] \sim \mathcal{N}\big(a[n2c[n]] + b[n2c[n]]\,X[n], \sigma_y\big) \qquad n = 1, \ldots, N$$

$$\begin{pmatrix} a[c] \\ b[c] \end{pmatrix} \sim \text{MultiNormal_Cholesky}\left(\begin{pmatrix} a_{\text{field}} \\ b_{\text{field}} \end{pmatrix}, \Sigma_{\text{chol}}\right) \qquad c = 1, \ldots, C \tag{9.8}$$

$$\Sigma_{\text{chol}} = \begin{pmatrix} \sigma_a & 0 \\ 0 & \sigma_b \end{pmatrix} \Omega_{\text{chol}} \tag{9.9}$$

In addition to these three equations and Eqs. (9.4)–(9.7) in Model Formula 9.6, we add one another statement:

$$\Omega_{\text{chol}} \sim \text{LKJcorr_Cholesky}(\upsilon) \tag{9.10}$$

where Σ_{chol} and Ω_{chol} represent the Cholesky factors[5] of a covariance matrix and a correlation matrix, respectively. MultiNormal_Cholesky denotes a multivariate normal distribution, where the covariance matrix is represented using its Cholesky factor. An LKJcorr_Cholesky distribution, a prior fo Ω_{chol} r, has a shape parameter $\upsilon \geq 1$ and it generates a Cholesky factor for a correlation matrix. When $\upsilon = 1$,

[5] A Cholesky factor is the lower triangular matrix obtained by implementing Cholesky factorization ($\Sigma = \mathbf{L}\mathbf{L}^T$) on a covariance matrix Σ. Cholesky factorization is a matrix operation method, and it is used here for a faster computation.

the distribution is noninformative, corresponding to a uniform distribution for the correlation matrix. As υ increases, the generated correlation matrix will be closer to an identity matrix, and only some row-column combinations are correlated. This shape parameter is most frequently specified by setting $\upsilon = 2$ or $\upsilon = 4$, giving a slightly larger constraint than setting $\upsilon = 1$.

The implementation of Model Formula 9.7 is as follows:

model9-7.stan

```
 1   data {
 2      (in addition to those variables in model9-5.stan)
 3      real Nu;
 4   }
 5
 6   parameters {
 7      array[C] vector[2] ab;
 8      vector[2] ab_field;
 9      cholesky_factor_corr[2] corr_chol;
10      vector<lower=0>[2] s_vec;
11      real<lower=0> s_y;
12   }
13
14   transformed parameters {
15      vector[C] a = to_vector(ab[1:C,1]);
16      vector[C] b = to_vector(ab[1:C,2]);
17      matrix[2,2] cov_chol = diag_pre_multiply(s_vec, corr_chol);
18   }
19
20   model {
21      ab_field[1] ~ normal(40, 20);
22      ab_field[2] ~ normal(1.5, 1.5);
23      s_vec[1] ~ student_t(4, 0, 20);
24      s_vec[2] ~ student_t(4, 0, 2);
25      corr_chol ~ lkj_corr_cholesky(Nu);
26      ab[1:C] ~ multi_normal_cholesky(ab_field, cov_chol);
27      Y[1:N] ~ normal(a[n2c] + b[n2c] .* X[1:N], s_y);
28   }
29
30   generated quantities {
31     matrix[2,2] corr = multiply_lower_tri_self_transpose(corr_chol);
32     matrix[2,2] cov = multiply_lower_tri_self_transpose(cov_chol);
33   }
```

Adding `generated quantities` block is optional. We did it here so that we can track the correlation and covariance matrix after estimation.

Line 9: Ω_{chol} is declared as `corr_chol`. A `cholesky_factor_corr` type is a special case of a `matrix` type and represents a Cholesky factor of a correlation matrix (i.e., lower triangle matrix, diagonal elements are positive, and squared sum of all the elements in each column is 1).

Line 10: A vector with two elements $(\sigma_a \sigma_b)^T$ is declared as `s_vec`.

Table 9.2 Functions for matrix operations in **`model9-7.stan`**

Functions	Description
`diag_pre_multiply(vector v, matrix m)`	It returns the product of the diagonal matrix whose diagonal elements are `v` and `m` as `matrix` type
`multiply_lower_tri_self_transpose(matrix m)`	It returns $\mathbf{LL}^T$ as `matrix` type, where $\mathbf{L}$ is the part of lower triangle and the diagonal elements of `m`

Line 17: Σ_{chol} is declared as `cov_chol` and it is further defined. It is a Cholesky factor of a covariance matrix (i.e., lower triangle matrix, and the diagonal elements are all positive). This line corresponds to Eq. (9.9).

Line 25: This line corresponds to Eq. (9.10). Shape parameter `Nu` has been declared in `data` block in line 3.

Line 26: This line corresponds to Eq. (9.8).

Line 31: The correlation matrix `corr` is calculated from its Cholesky factor `corr_chol`. The element in `corr[2,1]` corresponds to ρ, representing the correlation between a and b.

Line 32: The covariance matrix `cov` is calculated from its Cholesky factor `cov_chol`.

Several functions for matrix operations were used here, and we summarized them in Table 9.2.

9.3 Loosen Posterior Distribution by Reparameterization

For some models, some parameters are highly dependent on others. In such cases, their posterior distribution can be distorted and directly sampling from them is not efficient. In order to increase the sampling efficiency, a trick is to "loosen" the distribution by rearranging these parameters, namely *reparameterization*. The data scaling mentioned in Sect. 5.1.6 can also be considered as a type of reparameterization. In this section, we will show a specific example named *Neal's funnel*, to illustrate reparameterization (Neal, 2003). We will further apply reparameterization for a hierarchical model as well as another example.

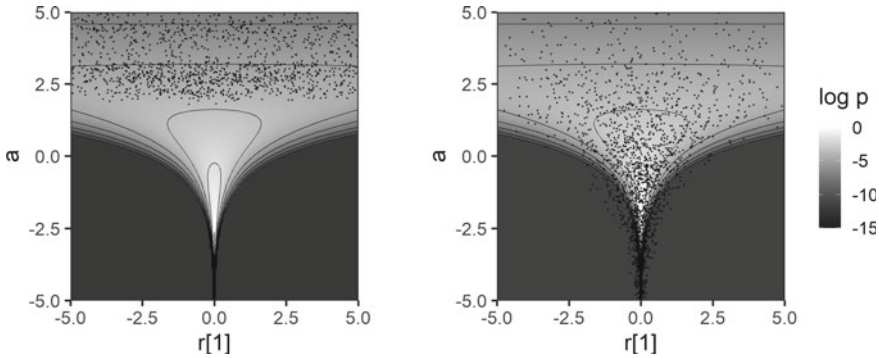


Fig. 9.5 The log marginal posterior distribution of the Neal's funnel, $\log p(r[1], a)$, is shown as the contour plot and is overlaid with dots that represent its draws.[6] The estimated results of Model Formula 9.8 (left) and Model Formula 9.9 (right) are shown

9.3.1 Neal's Funnel

In the Neal's funnel model, the prior is directly transformed as the posterior because there is no data input. The model can be written as:

Model Formula 9.8

$$a \sim \mathcal{N}(0, 3) \tag{9.11}$$

$$r[n] \sim \mathcal{N}(0, \exp(a/2)) \qquad n = 1, \ldots, 1000 \tag{9.12}$$

The parameters a and $r[n]$ will be estimated, and the log posterior distribution of the model can be obtained analytically. This model is named for the shape of a funnel that this log posterior resembles. Let's take a closer look at the shape of the distribution by projecting it onto a two-dimensional space, with $r[1]$ on the x-axis and a on the y-axis. After marginalizing out other parameters and a log transformation, a contour plot of its log posterior density, $\log p(r[1], a)$, can be obtained, as shown in Fig. 9.5. The white area represents the space with high log posterior density, and the region where $r[1]$ is close to zero and a is smaller than zero, is the tip of this funnel.

We omit the code for Model Formula 9.8 `data` block can be omitted for implementation because no data is used for this model. Figure 9.5 (left) shows that the draws from the estimated results, overlaid with log posterior distribution contour plot. MCMC is designed to sample more frequently in the area with higher posterior density. However, as we see from Fig. 9.5 (left), the funnel tip, where the log posterior density is highest, is not well-sampled. This is because of the highly distorted

[6] `chains = 1` and `iter_sampling = 2000` are used as the arguments, in order to avoid the multiple chains being mixed in the results.

distribution of the log posterior: the funnel tip is so narrow that the parameter space surrounding the tip cannot be sufficiently explored. This extreme shape of the log posterior distribution is due to the dependency between $r[n]$ and a. That is, the value of $r[n]$ is highly dependent on a, and its distribution shape changes largely when value of a changes.

In such a case, we can reduce parameter dependency by reparameterizing the model and separating out the scales of the parameters from the distribution that the parameters follow. More specifically, a raw parameter is assumed to follow a certain distribution with a fixed scale, and the original parameter is generated based on the combination of the raw parameter and the scale parameter such as SD. For distributions with a fixed scale, $\mathcal{N}(0, 1)$ or Uniform$(0, 1)$ is usually used.

In particular, Model Formula 9.8 can be represented as:

Model Formula 9.9

$$a_{\text{raw}} \sim \mathcal{N}(0, 1) \tag{9.13}$$

$$a = 3.0 \times a_{\text{raw}} \tag{9.14}$$

$$r_{\text{raw}}[n] \sim \mathcal{N}(0, 1) \qquad n = 1, \ldots, 1000 \tag{9.15}$$

$$r[n] = \exp(a/2) \times r_{\text{raw}}[n] \qquad n = 1, \ldots, 1000 \tag{9.16}$$

Equations. (9.13) and (9.14) together represent Eq. (9.11), and Eqs. (9.15) and (9.16) together represent Eq. (9.12). Here a_{raw}, which follows $\mathcal{N}(0, 1)$., represents the raw parameter[7] of a. Therefore, by making use of the property of a normal distribution, and multiplying scale 3 by the raw parameter, we can represent that a follows $\mathcal{N}(0, 3)$. For $r_{\text{raw}}[n]$ and $r[n]$, we applied the similar way of expression (see Sect. 6.12).

In the parameter space of a_{raw} and $r_{\text{raw}}[n]$, we reparameterized the parameters and thus their scales have been separated. By doing this, the strong funnel-shaped dependencies that we saw previously are gone, and sampling efficiency is largely improved as a result. We will leave the implementation of Model Formula 9.9 to readers as an exercise. The hint tohis exercise is to declare `a_raw` and `r_raw` in `parameters` block, and transform them into `a` and `r` in `transformed parameters` block. Figure 9.5 (right) shows the results of the draws, which are plotted in the same parameter space as in Fig. 9.5 (left).

[7] We use a *raw parameter* to indicate that it is a parameter before the processing and transformation steps.

9.3.2 Reparameterization of Hierarchical Models

Reparameterization can also be useful in hierarchical models. Here we reparameterize **model8-4.stan** in Sect. 8.1.6 as an example. The implementation is as follows:

model9-10.stan

```
 1  data {
 2     int N;
 3     int C;
 4     vector[N] X;
 5     vector[N] Y;
 6     array[N] int<lower=1, upper=C> n2c;
 7  }
 8
 9  parameters {
10     real a_field;
11     real b_field;
12     vector[C] a_raw;
13     vector[C] b_raw;
14     real<lower=0> s_a;
15     real<lower=0> s_b;
16     real<lower=0> s_y;
17  }
18
19  transformed parameters {
20     vector[C] a = a_field + s_a*a_raw[1:C];
21     vector[C] b = b_field + s_b*b_raw[1:C];
22  }
23
24  model {
25     a_raw[1:C] ~ normal(0, 1);
26     b_raw[1:C] ~ normal(0, 1);
27     Y[1:N] ~ normal(a[n2c] + b[n2c] .* X[1:N], s_y);
28  }
```

We explain the lines that are different from **model8-4.stan**.

Lines 12–13: Raw parameters are declared here.

Lines 20–21: Parameters `a` and `b` are obtained by multiplying scales and adding means to their raw parameters.

Lines 25–26: The raw parameters are specified to follow $\mathcal{N}(0, 1)$.

This way of reparameterization may improve the MCMC convergence for a hierarchical model. However, unlike the vectorization, which certainly speeds up the sampling, reparameterization could also slow down the computation. Based on (Betancourt & Girolami, 2013), reparameterization is worth trying when the amount of data is relatively small. In some cases, only a part of the hierarchical layers or the groups have small numbers of data points. We can then consider applying reparameterization for that part of the data.

9.3.3 Reparameterization of Multivariate Normal Distribution

Reparameterization can be useful in the models that use a multivariate normal distribution. Here we reparameterize **model9-7.stan** in Sect. 9.2.4 as an example. The implementation is as follows:

model9-11.stan

```
 1  data { same as model9-7.stan }
 2
 3  parameters {
 4     matrix[2,C] ab_raw;
 5     vector[2] ab_field;
 6     cholesky_factor_corr[2] corr_chol;
 7     vector<lower=0>[2] s_vec;
 8     real<lower=0> s_y;
 9  }
10
11  transformed parameters {
12     matrix[2,C] ab;
13     vector[C] a;
14     vector[C] b;
15     matrix[2,2] cov_chol = diag_pre_multiply(s_vec, corr_chol);
16     for (c in 1:C) {
17       ab[1:2,c] = ab_field[1:2] + cov_chol*ab_raw[1:2,c];
18      }
19     a[1:C] = ab[1,1:C]';
20     b[1:C] = ab[2,1:C]';
21  }
22
23  model {
24     ab_field[1] ~ normal(40, 20);
25     ab_field[2] ~ normal(1.5, 1.5);
26     s_vec[1] ~ student_t(4, 0, 20);
27     s_vec[2] ~ student_t(4, 0, 2);
28     corr_chol ~ lkj_corr_cholesky(Nu);
29     to_vector(ab_raw) ~ normal(0, 1);
30     Y[1:N] ~ normal(a[n2c] + b[n2c] .* X[1:N], s_y);
31  }
```

We explain the lines that are different from **model9-7.stan**.

Line 4: A raw parameter is declared here.

Lines 16–18: The parameter of a `vector` type `ab[1:2,c]` is obtained by multiplying a Cholesky factor of the covariance matrix and the raw parameter of a `vector` type `ab_raw[1:2,c]`, and adding the mean vector (`ab_field`) to this product (see Sect. 6.14).

Lines 19–20: To describe lines 16–18 and 29 smoothly, `ab` is defined as a 2 × C matrix (by stacking `C` column vectors of length `2` horizontally). Therefore, `ab[1,1:C]` is a row vector, and transposition is need to assign it to `a[1:C]`.

Line 29: The raw parameters are specified to follow $\mathcal{N}(0, 1)$ `to_vector` function can convert a `matrix` type into a `vector` type with column-major order.[8] This function is needed because the `normal` is not vectorized for a `matrix` type.

The reparameterization of a multivariate normal distribution will be used in Sect. 13.7.

9.4 Other Cases

With solutions and tricks introduced so far (Sects. 9.1–9.3), MCMC convergence issues may still be unresolved. This could happen when, for instance, functions with discontinuous derivatives are used, or many local optima prevent the sampling from posterior distributions.

Functions with discontinuous derivatives

Some of the Stan functions, such as `step`, `abs`, `fmax`, or `ceil`,[9] have discontinuous derivatives. When a model uses such functions, or when it contains `if` statements that are based on parameter values, computing the partial derivatives can be inefficient. For these models, it should be checked whether these functions can be eliminated by model rearrangement. An alternative option would be using other functions to replace these functions which have discontinuous derivatives.

Models with multiple local optima

Depends on the model formula, we may find that the estimated parameters have multiple local optima. The tricks such as using weakly informative priors, reparameterization, or changing the initial values of MCMC may not be enough to fully solve this issue. More advanced methods, such as *replica exchange method* (also called *parallel tempering*), can be a solution in some cases. Replica exchange method is not implemented in Stan yet, but it can be done via applying some tricks in R or Python. These are over the scope of this book, but interested readers can refer online codes that explain how to implement these advanced techniques.[10]

9.5 Supplementary Information

More examples on parameter identifiability can be found in the chapter "Problematic Posteriors" in the Stan reference.

---

[8] For instance, for a 2×2 matrix variable a, `to_vector` will order elements as `a[1,1], a[2,1], a[1,2], a[2,2]`, and convert to a `vector` type.

[9] For more details on these functions, see the Stan reference, the section "Step-like Functions".

[10] https://gist.github.com/MatsuuraKentaro/bccf13af3ba52c9d6c379c0032725b91.

In Model Formula 9.2, we obtained the log-likelihood by assuming that performance difference between players follows a normal distribution. In fact, this is equivalent to using the probit regression. On the other hand, if the likelihood of the game result is determined from inv_logit$(ay + b)$ $(a, b$ are regression coefficients), then it is equivalent to using logistic regression. From our experience, probit regression yields more intuitive explanations when modeling events that involve competitions, for instance observing game results, or answering questions.

For weakly informative prior distributions used in this chapter, we referred "Prior Choice Recommendations" in the Stan wiki page,[11] we recommend this page for all the Stan users to get useful information. In addition, in the Stan GitHub repository,[12] all the examples in WinBUGS are implemented using Stan. After reading WinBUGS documentation to understand problem background (e.g., the pumps example in vol1), readers can further read these Stan code that uses weakly informative priors. For an LKJ correlation distribution, the Stan reference sections such as "Multilvariate priors for hierarchical models" and "Correlation Matrix Distributions", would provide more details for interested readers.

For the reparameterization, again, the Stan reference is probably the best place for finding more information. In addition to the section "Reparameterization Change of Variable", the section "Optimizing Stan Code" gives examples of reparameterizing parameters that follow multivariate normal distribution.

To understand more on replica exchange, the review paper (Iba, 2001) would be a good place to start.

References

Betancourt, M. & Girolami, M. (2013). Hamiltonian Monte Carlo for hierarchical models. arXiv: 1312.0906.

Gelman, A. (2006). Prior distributions for variance parameters in hierarchical models. *Bayesian Analysis, 1*(3), 515–533.

Gelman, A., & Jennifer, H. (2007). *Data analysis using regression and multilevel/hierarchical models*. Cambridge University Press.

Iba, Y. (2001). Extended ensemble Monte Carlo. *International Journal of Modern Physics C, 12*, 623–656.

Neal, R. (2003). Slice sampling. *Annals of Statistics, 31*(3), 705–767.

[11] https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations.

[12] https://github.com/stan-dev/example-models.

Chapter 10
Discrete Parameters

Hereafter we will call parameters that have discrete integer values as *discrete parameters*. The fact that Stan cannot sample from discrete parameters is a major limitation of Stan. In this section, we will provide several solutions for this problem. The basic approach is to eliminate discrete parameters from the log-likelihood by counting all the possible cases (i.e., marginalizing out the parameters).

The content of each section in this chapter is as follows:

- Section 10.1 explains marginalization and functions necessary to handle discrete parameters.
- Sections 10.2 to 10.3 give examples for discrete parameter applications. Specifically, Sect. 10.2 discusses an example of mixture of normal distributions. Section 10.3 illustrates zero-inflated distributions.

10.1 Techniques to Handle Discrete Parameters

If we want to express log-likelihood of a model involving discrete parameters without directly specifying discrete parameters, it is necessary to use `target` notation (see Sect. 3.6) and `log_sum_exp` function in Stan. In this section, we will first explain `log_sum_exp` function which efficiently calculates logarithm of summation of exponential of variables. And then we will explain the method of eliminating discrete parameters from log-likelihood by counting all possible values that discrete parameters can take, i.e., marginalizing out the parameters.

© Springer Nature Singapore Pte Ltd. 2022

K. Matsuura, *Bayesian Statistical Modeling with Stan, R, and Python*,
https://doi.org/10.1007/978-981-19-4755-1_10

10.1.1 log_sum_exp Function

`log_sum_exp` function takes multiple real number arguments. For example, `log_sum_exp(real x, real y)` defines a function with two `real` type arguments as follows:

$$\text{log_sum_exp}(x, y) = \log(\exp(x) + \exp(y))$$

Alternatively, `log_sum_exp` function can also take single argument of a `vector` type, a `row_vector` type, a `matrix` type or an array. For example, the definition of `log_sum_exp(vector x)` is as follows:

$$\text{log_sum_exp}\left(\begin{pmatrix} x_1 \\ \vdots \\ x_K \end{pmatrix}\right) = \log\left(\sum_{k=1}^{K} \exp(x_k)\right)$$

Because the right-hand side of this equation is being "log sum of exponentials", this function is called `log_sum_exp` function. When we define $\exp(x_k) = u_k$, the function is also expressed as follows:

$$\text{log_sum_exp}\left(\begin{pmatrix} \log u_1 \\ \vdots \\ \log u_K \end{pmatrix}\right) = \log\left(\sum_{k=1}^{K} u_k\right) \tag{10.1}$$

That is, logarithm of the sum of values (right-hand side) is equivalent to passing the vector of logarithm of values to `log_sum_exp` function (left-hand side). Computing the right-hand side of the equation is required for the models in this chapter, but the equivalent computation is done using the formula on the left-hand side because the numerical calculations are more stable. This is because if any elements in vector u_k is a product of small values, the expression on the right-hand side involves computing "product of small values" which is numerically unstable. Instead, using `log_sum_exp` function, "sum of logarithms of small numbers" is computed and this is more stable. Therefore, it is very common to use `log_sum_exp` function when the argument of a log function is a summation. Note, however, that `log_sum_exp` function can only be used when all u_k are positive.

10.1.2 Marginalizing Out Discrete Parameters

As mentioned in Sect. 3.2, it is not allowed to declare discrete (`int` type) parameters in Stan. Thus, we need to modify the model and avoid directly working with discrete

parameters. One way to do this is to list all possible cases of discrete parameter values and sum up the probabilities of each case to eliminate the discrete parameters. This method is called "summing out" or "marginalizing out". In order to marginalize out discrete parameters, we will apply product rule and sum rule, which are basic rules of the probability theory. Product rule is to multiply the possibilities of independent and simultaneous events, and sum rule is to add the probability of events that occur exclusively.

Here we introduce two examples of marginalizing out discrete parameters. The first example is a simple model with two cases, and the second example uses a more complex model.

Discrete parameter that follows Bernoulli distribution (coin toss)

Consider a questionnaire targeting high school students, asking if they have ever smoked. Because this is a sensitive question, if we ask them directly, some students might be hesitant to admit they have smoked before. So instead, we will ask them to toss a coin secretly so that we wouldn't know the outcome of coin toss, and if the outcome is heads, the truth must be told. Otherwise, the student is asked to answer "Yes" regardless of their smoking experiences (Warner, 1965). Suppose we obtained the results from 100 students. (Data File 10.1, a row corresponds to a student). Here, $Y = 1$ represents "Yes" and $Y = 0$ represents "No". We want to estimate the probability of smoking (q), presuming the probability of getting heads from a coin toss is 0.5.

Data File 10.1 Structure of `data-coins.csv`

```
  1   Y
  2   1
  3   0
  4   1
      ...
101   1
```

The model for the data generating mechanism can be expressed as follows:

Model Formula 10.1

$$
\begin{aligned}
coin[n] &\sim \text{Bernoulli}(0.5) && n = 1, \ldots, N \\
\theta[1] = q, &\quad \theta[2] = 1.0 \\
Y[n] &\sim \text{Bernoulli}(\theta[coin[n] + 1]) && n = 1, \ldots, N
\end{aligned}
$$

where N is the number of students who answered the questionnaire, n is the index of each student. $coin[n]$ is a random variable, and it takes 1 if the outcome of the coin toss by student n is heads and it takes 0 otherwise. Because $coin[n]$ cannot be observed by analyst, it is an unknown parameter. θ is the probability of answering "Yes", where $\theta[1]$ is the probability when the outcome of a coin toss is heads (i.e., answering honestly), a $\theta[2]$ is the probability when the outcome of the coin toss is tails (i.e., always answering "Yes").

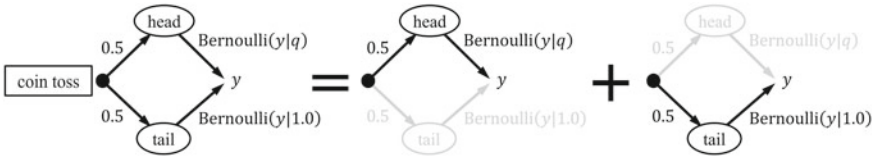


Fig. 10.1 The mechanism of a student answering to the question

Model Formula 10.1 cannot be implemented directly in Stan, because it cannot declare an `int` type parameter for the discrete parameter *coin*[*n*]. So, we eliminate the `int` type parameter from the model by summing up the probabilities of each case of coin toss (i.e., heads or tails). Let us describe the underlying mechanism of a student answering to the smoking question, as illustrated in Fig. 10.1.

Figure 10.1 shows that there are two paths (either upper path or lower path) to answer "Yes". The upper path indicates the case when the outcomes is heads and the student answers honestly (probability $= 0.5 \times \text{Bernoulli}(y|q)$), whereas the lower path indicates the case when the outcome is tails and the student always answers "Yes" (probability $= 0.5 \times \text{Bernoulli}(y|1.0)$). These are exclusive events that do not occur simultaneously, therefore, therobability distribution that generates the data would be the sum of these two probabilities, as described below:

$$p(y|q) = 0.5 \times \text{Bernoulli}(y|q) + 0.5 \times \text{Bernoulli}(y|1.0) \tag{10.2}$$

When Eq. (10.2) is integrated with respect to y, it becomes 1, thus, Eq. (10.2) is a probability distribution. From this distribution, we can calculate the probability that the answer is "Yes" or "No", even if we do not know the result of coin toss. According to the above argument, using Eq. (10.2), the likelihood of q for one student's answer is as follows:

$$p(Y|q) = 0.5 \times \text{Bernoulli}(Y|q) + 0.5 \times \text{Bernoulli}(Y|1.0) \tag{10.3}$$

In order to implement this model in Stan, we need to take logarithm of Eq. (10.3), then sum up the log-likelihood into target as explained in Sect. 3.6. By using `log_sum_exp` function explained in Sect. 10.1.1, the log-likelihood can be expressed as follows:

$$\begin{aligned} \log\ p(Y|q) = \log_\text{sum_exp}(&\log\ 0.5 + \log\ \text{Bernoulli}(Y|q) \\ &\log\ 0.5 + \log\ \text{Bernoulli}(Y|1.0)) \end{aligned} \tag{10.4}$$

In order to extend this log-likelihood for N students, we simply repeat it N times using `for` loop. The implementation of Model Formula 10.1 is as follows:

model10-1.stan

```
1   data {
2     int N;
3     array[N] int<lower=0, upper=1> Y;
4   }
5
6   parameters {
7     real<lower=0, upper=1> q;
8   }
9
10   model {
11     for (n in 1:N) {
12       target += log_sum_exp(
13         log(0.5) + bernoulli_lpmf(Y[n] | q),
14         log(0.5) + bernoulli_lpmf(Y[n] | 1)
15       );
16     }
17   }
```

Lines 12–15: the part of `log_sum_exp` function corresponds to Eq. (10.4).

Lines 13 and 14: the `bernoulli_lpmf(Y | q)` is a convenient function in Stan, and it is equivalent tologBernoulli($Y|q$).[1]

The estimated median and 95% Bayesian confidence interval of parameter q using Data File 10.1 is 0.20 and (0.02 to 0.37), respectively.

Discrete parameters that follow Poisson distribution

Here let us consider another problem of coin toss. Suppose that a random number m is generated from a Poison distribution with an unknown parameter λ. Then we toss m coins and count the number of heads, and record this count as Y. Suppose we obtained 100 records of Y generated by repeating this procedure 100 times (Data File 10.2, a row corresponds to a record).

Data File 10.2 Structure of `data-poisson-binomial.csv`

```
  1  Y
  2  6
  3  5
  4  8
     ...
101  9
```

Here, the maximum value of Y was 9. Let us estimate λ from this data, assuming the probability of coin toss resulting in heads is always 0.5. Then the model formula can be expressed as follows:

[1] In the "`lpmf`", "`l`" stands for "log", and "`pmf`" stands for "probability mass function".

Model Formula 10.2

$$
\begin{aligned}
m[n] &\sim \text{Poisson}(\lambda) && n = 1, \ldots, N \\
Y[n] &\sim \text{Binomial}(m[n], 0.5) && n = 1, \ldots, N
\end{aligned}
$$

where N is the number of records, and n is the index of each record. And $m[n]$ is a discrete parameter that cannot be observed. We need to eliminate m by summing up the probability under all possible values of m. For that, we first consider the likelihood for a single record. Because m could take all nonnegative integers, the likelihood would be as follows:

$$
\begin{aligned}
p(Y|\lambda) = &\text{Poisson}(0|\lambda) \times \text{Binomial}(Y|0, 0.5) + \\
&\text{Poisson}(1|\lambda) \times \text{Binomial}(Y|1, 0.5) + \\
&\text{Poisson}(2|\lambda) \times \text{Binomial}(Y|2, 0.5) + \\
&\cdots
\end{aligned} \tag{10.5}
$$

where the first term of right side corresponds to the case $m = 0$, and the second term corresponds to the case $m = 1$, and so on. Apparently, we cannot sum up all the cases of m. But considering $m \leq 40$ is sufficient in this example because the maximum of Y observed is 9. The probability of Binomial$(9|m, 0.5)$. is much smaller when $m = 40$, compared to that when m. is a smaller value ($m = 12, \ldots, 25$ for instance[2]). Thus, even if we ignore the case when m. exceeds 40, it is still a sufficiently good approximation. In addition, when $m < Y$, the probability of Binomial$(9|m, 0.5)$ is 0, so we omit the term corresponding to these cases from Eq. (10.5). So the likelood couldarranged as follows:

$$
p(Y|\lambda) = \sum_{m=Y}^{40} [\text{Poisson}(m|\lambda) \times \text{Binomial}(Y|m, 0.5)] \tag{10.6}
$$

We need the log-likelihood to implement the model in Stan, so we can use `log_sum_exp` function for Eq. (10.6), as follows:

$$
\log p(Y|\lambda) = \log\left(\sum_{m=Y}^{40} [\text{Poisson}(m|\lambda) \times \text{Binomial}(Y|m, 0.5)]\right)
$$

[2] The value of Binomial$(9|40, 0.5)$ is about 2.5×10^{-4}, which can be obtained by using `dbinom(9, size = 40, prob = 0.5)` in R.

$$= \log_\text{sum_exp}\left(\begin{pmatrix} \log \text{Poisson}(Y|\lambda) & + \log \text{Binomial}(Y|Y, 0.5) \\ \log \text{Poisson}(Y+1|\lambda) & + \log \text{Binomial}(Y|Y+1, 0.5) \\ & \vdots \\ \log \text{Poisson}(40|\lambda) & + \log \text{Binomial}(Y|40, 0.5) \end{pmatrix}\right) \tag{10.7}$$

To extend this log-likelihood to all N records, we just need to repeat Eq. (10.7) N times. The implementation of Model Formula 10.2 is as follows:

model10-2.stan

```
1   data {
2     int N;
3     int M_max;
4     array[N] int<lower=0> Y;
5   }
6
7   parameters {
8     real<lower=0> lambda;
9   }
10
11  model {
12     for (n in 1:N) {
13       vector[M_max-Y[n]+1] lp;
14       for (m in Y[n]:M_max) {
15         lp[m-Y[n]+1] = poisson_lpmf(m | lambda) +
16                        binomial_lpmf(Y[n] | m, 0.5);
17       }
18       target += log_sum_exp(lp);
19     }
20  }
```

Line 3: Declare `M_max` as the maximum value of m that we want to consider (here we set `M_max` = 40).

Line 13: Variables that do not require sampling can be declared just after any "`{`" in `model` block in Stan. So here we declare a local variable `lp` that is a `vector` type of length `M_max-Y[n] + 1`.[3] It corresponds to the argument of `log_sum_exp` function in Eq. (10.7).

Lines 14–16: Assigning each element of the `vector lp` with the corresponding value in Eq. (10.7), and then this `vector` is passed to `log_sum_exp` function. `poisson_lpmf(m | lambda)` is equivalent to $\text{logPoisson}(m|\lambda)$ and `binomial_lpmf(Y | m, 0.5)` is equivalent to $\text{logBinomial}(Y|m, 0.5)$.

The estimated median and 95% Bayesian confidence interval of parameter λ using Data File 10.2 is 9.57 and (8.75 to 10.5), respectively.

[3] The variable name lp is an abbreviation for log probability.

10.1.3 Using Mathematical Relationships

In the previous subsection, we explained the method of eliminating discrete parameters by marginalizing them out. But this is not always applicable approach, because Stan cannot handle the problem where the number of cases is very large. For instance, in the example of "Discrete parameters that follow Poisson distribution" in the previous subsection, if the maximum value of data is not 9 but is much larger, it is necessary to consider a very large value for m. In that case, the estimation would take extremely long time. However, discrete parameters can sometimes be eliminated by using mathematical relationships. The above example is one of such examples. Here we look at the likelihood of a single record Y for explanation. The model is described as follows:

Model Formula 10.3

$$m \sim \text{Poisson}(\lambda)$$
$$Y \sim \text{Binomial}(m, p)$$

The probability of the coin toss resulting in heads is represented as p instead of 0.5 for further explanation. This expression can be rearranged to eliminate m as follows:

Model Formula 10.4

$$Y \sim \text{Poisson}(\lambda\ p)$$

Here is how we derived Model Formula 10.4 from Model Formula 10.3:

$$\begin{aligned}
&\sum_{m=0}^{\infty}[\text{Poisson}(m|\lambda) \times \text{Binomial}(Y|m, p)] \\
&= \sum_{m=Y}^{\infty}\left[\frac{\lambda^m e^{-\lambda}}{m!} \times \sum_{m=Y}^{\infty}\left[\frac{m!}{Y!(m-Y)!}p^Y(1-p)^{m-Y}\right]\right] \\
&= \frac{p^Y e^{-\lambda}}{Y!}\sum_{m=Y}^{\infty}\left[\frac{\lambda^m}{(m-Y)!}(1-p)^{m-Y}\right] \\
&= \frac{p^Y e^{-\lambda}}{Y!}\sum_{\xi=0}^{\infty}\left[\frac{\lambda^{\xi+Y}}{\xi!}(1-p)^{\xi}\right] = \frac{(\lambda p)^Y e^{-\lambda}}{Y!}\sum_{\xi=0}^{\infty}\left[\frac{\lambda^{\xi}}{\xi!}(1-p)^{\xi}\right] \\
&= \frac{(\lambda p)^Y e^{-\lambda}}{Y!}e^{\lambda(1-p)} = \frac{(\lambda p)^Y e^{-(\lambda p)}}{Y!}
\end{aligned}$$

The implementation of Model Formula 10.4 is straightforward. By using the mathematical relationship of the two distributions, in addition to getting a more precise

result by accounting for the case that $m > 40$, we can largely improve the computing speed.

This relationship between two distributions is not unique to Poisson and Binomial distributions, and there are other combinations like this one. But rather than memorizing all of them, we recommend searching the internet or calculating them manually when you encounter specific problems. You can also use symbolic formula manipulation software such as Mathematica and Maxima to facilitate the manipulation. On the other hand, when dealing with large discrete values such as population counts, it is practically impossible to calculate it in Stan. This is because there are too many cases, even if it is finite. If this is a case, there is an alternative option to approximate a discrete distribution such as a Binomial distribution and a Poisson distribution with a continuous distribution such as a normal distribution.

Up to this point we have discussed techniques to deal with discrete parameters in Stan. In subsequent sections, we will introduce some examples that use these techniques for discrete parameters.

10.2 Mixture of Normal Distributions

In this section, we deal with *Score* data, which is recorded as the results of measuring the employees' abilities (continuous values) in a company. The data contains records of 100 employees (Data File 10.3, a row corresponds to a single record).

Data File 10.3 Structure of `data-mix1.csv`

```
  1  Y
  2  -0.84
  3  -0.35
  4  2.34
     ...
101  3.74
```

Figure 10.2 shows the histogram of Y and the estimated density function using R. It is assumed that an analyst knows that employees are composed of two groups: one is a group of hard workers who have high *Score* and the other is of lazy workers that have low *Score*. Therefore, we consider a mixture of two normal distributions for this problem. A mixture of two normal distributions is a distribution that represents two data generation mechanisms: with probability a, it generates a data point from one normal distribution, and with probability $1 - a$, it generates a data point from the other normal distribution. The mathematical expression is as follows:

$$\text{Normal_Mixture}(y|a, \mu_1, \mu_2, \sigma_1, \sigma_2) = a \times \mathcal{N}(y|\mu_1, \sigma_1) + (1 - a) \times \mathcal{N}(y|\mu_2, \sigma_2) \tag{10.8}$$

Equation (10.8) is a probability distribution because integrating the equation with respect to y yields 1. Comparing Eq. (10.8) with Eq. (10.2) in Sect. 10.1.2, you would

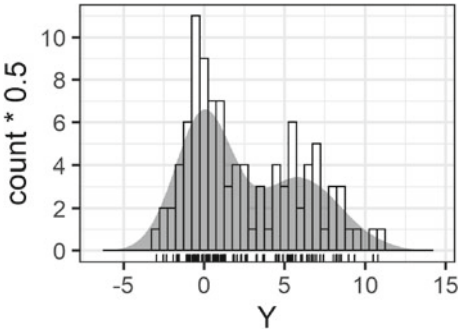


Fig. 10.2 The histogram and estimated density function

find the only difference is that the Bernoulli disions of Eq. (10.2) are replaced with the normal distributions, and 0.5 is substituted with a. Using a mixture of normal distributions, we can describe the model formula as follows:

Model Formula 10.5

$$Y[n] \sim \text{Normal_Mixture}(a, \mu_1, \mu_2, \sigma_1, \sigma_2) \qquad n = 1, \ldots, N$$

where N. is the total number of employees and n is the index of each employee. a, μ_1, μ_2, σ_1, and σ_2 are the parameters to be estimated from the data. The implementation of Model Formula 10.5 is as follows:

model10-5.stan

```
data {
   int N;
   vector[N] Y;
}

parameters {
   real<lower=0, upper=1> a;
   ordered[2] mu;
   vector<lower=0>[2] sigma;
}

model {
    for (n in 1:N) {
      target += log_sum_exp(
        log(a) + normal_lpdf(Y[n] | mu[1], sigma[1]),
        log(1-a) + normal_lpdf(Y[n] | mu[2], sigma[2])
      );
    }
}
```

Lines 14–17: This part is almost the same as the case of Bernoulli distributions (**model10-1.stan**).

Lines 15 and 16: `normal_lpdf(Y | mu, sigma)` represents $\log\mathcal{N}(Y|\mu,\sigma)$.

Line 8: `mu` is declared as `ordered` type to distinguish two normal distributions. If a `vector` type is used instead, you need to check if there is a label switching (see Sect. 9.1.3).

If we use a convenient and stable function `log_mix` in Stan, we can also write lines 14–17 as follows:

```
14  target += log_mix(a,
15    normal_lpdf(Y[n] igma[1]),
16    normal_lpdf(Y[n] | mu[2], sigma[2])
17  );
```

The estimated median and 95% Bayesian confidence interval of parameters are shown below:

a: 0.55 (0.37, 0.68) μ_1: −0.05 (−0.49, 0.43) μ_2: 5.85 (4.23, 6.82).
σ_1.: 1.29 (0.96, 1.72) σ_2: 2.39 (1.69, 3.55).

Note that **model10-5.stan** sometimes does not converge easily. This is because the prior distributions of `mu` and `sigma` are almost noninformative, and the model have many local optima.

Next, let us consider the case of fitting the data to a mixture of K normal distributions. For this purpose, we use *Score* data collected from another 200 employees. (Data File 10.4, a row corresponds to a record).

Data File 10.4 Structure of `data-mix2.csv`

```
  1  Y
  2  9.44
  3  9.77
  4  11.56
     ...
201  29.04
```

As in the previous example, Fig. 10.3 shows the histogram of Y and the estimated density function using R. The definition of a mixture of K. normal distributions are as follows:

$$\text{Normal_Mixture}(y|\vec{a},\vec{\mu},\vec{\sigma}) = \sum_{k=1}^{K} a_k\,\mathcal{N}(y|\mu_k,\sigma_k) \tag{10.9}$$

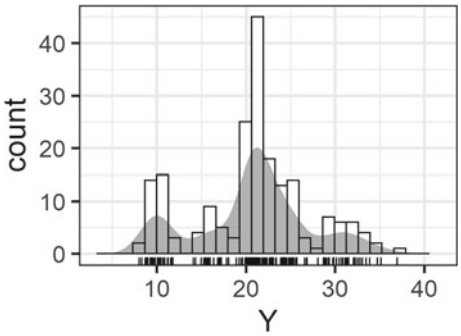


Fig. 10.3 The histogram and estimated density distribution

where each element of $\vec{a}$ is in the range of [0, 1] and the sum of these elements is 1. In other words, $\vec{a}$ is a simplex. Using this definition, we can write the model as follows:

Model Formula 10.6

$$Y[n] \sim \text{ Normal_Mixture}(\vec{a}, \vec{\mu}, \vec{\sigma}) \qquad n = 1, \ldots, N$$

By taking logarithm of Eq. (10.9) and using `log_sum_exp` function, the log-likelihood of this model formula can be written as follows:

$$\begin{aligned}
\log \text{Normal_Mixture}(y|\vec{a}, \vec{\mu}, \vec{\sigma}) &= \log \left[\sum_{k=1}^{K} a_k \mathcal{N}(y|\mu_k, \sigma_k) \right] \\
&= \text{log_sum_exp} \left[\begin{pmatrix} \log\{a_1 \mathcal{N}(y|\mu_1, \sigma_1)\} \\ \vdots \\ \log\{a_K \mathcal{N}(y|\mu_K, \sigma_K)\} \end{pmatrix} \right] \\
&= \text{log_sum_exp} \left[\begin{pmatrix} \log a_1 + \log \mathcal{N}(y|\mu_1, \sigma_1) \\ \vdots \\ \log a_K + \log \mathcal{N}(y|\mu_K, \sigma_K) \end{pmatrix} \right]
\end{aligned} \tag{10.10}$$

Now the implementation of Model Formula 10.6 is as follows:

model10-6.stan

```
data {
   int N;
   int K;
   vector[N] Y;
}

parameters {
```

```
 8     simplex[K] a;
 9     ordered[K] mu;
10     vector<lower=0>[K] sigma;
11     real<lower=0> s_mu;
12   }
13
14   model {
15     mu[1:K] ~ normal(mean(Y), s_mu);
16     sigma[1:K] ~ gamma(1.5, 1.0);
17     for (n in 1:N) {
18       vector[K] lp;
19       for (k in 1:K) {
20         lp[k] = log(a[k]) + normal_lpdf(Y[n]|mu[k], sigma[k]);
21       }
22       target += log_sum_exp(lp);
23     }
24   }
```

Here, we focus on the parts that differ from **model10-6.stan**.

Line 3: The number of normal distributions is declared as K.

Line 8: `a` is declared as a `simplex` type (see Sect. 9.2.3).

Lines 9–10: The number of means and SDs is also K.

Lines 18–22: The `vector` type `lp` is passed to `log_sum_exp` function in Eq. (10.2).

Line 15: A weak constraint was added to `mu`. We added this constraint because the parameters might be not identifiable if only using a subset of K normal distributions is sufficient for the log-likelihood to be high. For instance, without any constraint, `mu` in these remaining components can be very large values. We recommend trying a simple model without such constraints, and then add the constraint if we detect these problems.

Line 16: Similarly, a weak constraint was added to `sigma`. Here we specify `gamma(1.5, 1.0)`. When setting such a weakly informative prior distribution, we often decide them with the trial-and-error process, where we draw histogram of data and observe shapes of gamma distributions with various parameters. As another option, it is a good idea to use a truncated normal distribution instead of a gamma distribution or set all `sigma` to a fixed value such as 1.

In this case, we used $K = 5$. Note that the more components of normal distributions we have for a mixture model, the harder it is for MCMC to converge.

functions block

In this part, we explain how to use `functions` block that allows us to define our own functions in a Stan code. Using `functions` block, **model10-6.stan** can be rewritten as follows:

model10-6b.stan

```
 1    functions {
 2      real normal_mixture_lpdf(real Y, int K, vector a,
 3                               vector mu, vector sigma) {
 4        vector[K] lp;
 5        for (k in 1:K) {
 6          lp[k] = log(a[k]) + normal_lpdf(Y | mu[k], sigma[k]);
 7        }
 8        return log_sum_exp(lp);
 9      }
10    }
11
12   data { same as model10-6.stan }
13
14   parameters { same as model10-6.stan }
15
16   model {
17      mu[1:K] ~ normal(mean(Y), s_mu);
18      sigma[1:K] ~ gamma(1.5, 1.0);
19      for (n in 1:N) {
20        Y[n] ~ normal_mixture(K, a, mu, sigma);
21      }
22   }
```

Lines 2–3: Defining the function name, the arguments, and the data type of a return value.

Line 8: Returning the result of this function.

Line 20: If we define a function name as "`XXX_lpdf`" (for a probability density function) or "`XXX_lpmf`" (for a probability mass function) in `functions` block, we can use the probability distribution "`XXX`" in `model` block as those other internal functions defined in Stan. Internally, line 20 is equivalent to the following statement:

```
target += normal_mixture_lpdf(Y[n], K, a, mu, sigma);
```

In `functions` block, we are not allowed to use variables defined in `data` block (K in this example), because it is specified before `data` block. Thus, in order to use the length of a `vector` type, we need to pass it to the function as an argument. Alternatively, we can obtain it in our own function by using other functions such as `num_elements`, `rows` and `cols`.

`functions` block makes modeling highly flexible, but excessive use of it might reduce readability of code. In general, as in this example, when we aim to define a new probability distribution, using `functions` block can simplify the code. Besides, when the same code appears repeatedly in `model` block or `generated quantities` block, defining the code as a function is also recommended. See the Stan reference chapter "User-Defined Functions" for more examples.

10.3 Zero-Inflated Distribution

In this section, we deal with data obtained from the questionnaires of a restaurant on the number of visits for dinner. Here we have a comma-separated file for 200 respondents (Data File 10.5, a row corresponds to a respondent). Each column represents:

Sex: Gender of the respondent (0: female, 1: male).

Sake: Whether the respondent drinks sake (Japanese alcohol) or not (0: does not drink, 1: drinks).

Age: Age.

Y: The number of visits to this restaurant in the last 60 days by this respondent (response variable).

Data File 10.5 Structure of `data-ZIB.csv`

```
  1  Sex,Sake,Age,Y
  2  1,1,18,5
  3  0,0,18,2
  4  0,1,18,1
     ...
201  0,0,55,6
```

To begin with, we can try to fit multiple linear regression. The summary result of `lm` function in R is as follows:

```
Call:
lm(formula = Y ~ ., data = d)

Residuals:
Min     1Q      Median   3Q     Max
-6.85   -3.38   -0.38    2.65   10.44

Coefficients:
              Estimate   Std. Error   t value   Pr(>|t|)
(Intercept)   0.881      1.038        0.8       0.40
Sex           1.323      0.551        2.4       0.02
Sake          2.657      0.585        4.5       1e-05
Age           0.046      0.026        1.8       0.08

Residual standard error: 3.8 on 196 degrees of freedom
Multiple R-squared: 0.14, Adjusted R-squared: 0.13
F-statistic: 11 on 3 and 196 DF, p-value: 1.2e-06
```

The result indicates that a respondent who is male, drinks sake, and is relatively old, has high frequency of visiting the restaurant. However, the coefficient of determination is low as indicated by Multiple R-squared: 0.14. Furthermore, if we plot predicted distribution for each respondent by using predict function in R, we will

find that some predicted values are negative. This is because multiple regression forcefully fit a normal distribution to the number of visits. This low coefficient of determination and the negative predicted values make it hard to interpret the estimation results. Therefore, to obtain more interpretable result, we will follow the statistical modeling workflow that was introduced in Sect. 1.4.

10.3.1 Set Up Purposes and Check Data Distribution

Here let us assume that the purpose of the analysis is to identify the people who are likely to be the regular customer at this restaurant. We also want to know how much we can predict if they are likely to become regular from these explanatory variables, as well as how much each explanatory variable contributes to whether they will become regular.

Next, we plot a scatterplot matrix to check the data distribution as shown in Fig. 10.4. From this figure, the following indications could be derived:

- 1st row 1st column: the number of female respondents is less than that of men.
- 2nd row 2nd column: the number of respondents who drinks is less than those who does not drink.
- 3rd row 3rd column: the age is uniformly distributed across all the respondents.
- 4th row 4th column: regarding the number of visits, 0 is prominent and the other number is distributed as a mountain-shape. This is an important information.
- 4th row 1st column: the number of visits is small for the female respondents.
- 4th row 3rd column: older people slightly visit more than younger people. However, its rank correlation is weak (as indicated from the 3rd row 4th column), presumably because of the prominence of $Y = 0$.
- 2nd row 4th column: the rank correlation is relatively large (0.34), this might be due to the fact that many people who drink have visited the restaurant at least once.

10.3.2 Imagine Data Generating Mechanisms

Because the distribution of the number of visits Y is distinctive, it is important to consider under what mechanism the data was generated. To begin with, let us consider the mechanism of visits of a person. Here we consider two distinct reasons for a customer to visit the restaurant. The first possible reason is that a customer visited the restaurant for the first time as he/she was exploring a good restaurant in the city. The second is that the place became his/her favorite place after the first visit, and thus he/she visited again as a regular customer. And we further assume the former visit follows a Bernoulli distribution with the probability of a visit for the first time q_f (we call it first visit probability), the latter visits follow a Binomial distribution with the number of trials equals 60 and the regular visit probability q_r. We then apply logistic regression for q_f and a binomial logistic regression for q_r to estimate the effect of each explanatory variables on these two probabilities separately.

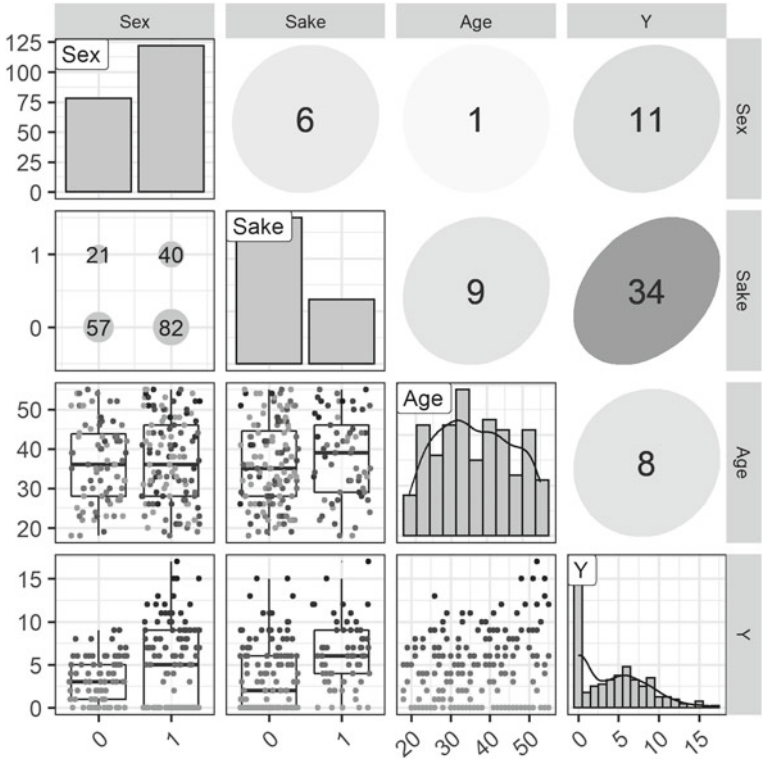


Fig. 10.4 Scatterplot matrix. The legend is almost the same as Fig. 5.1. The darkness of points indicates the number of visits to the restaurant for each subject (darker points indicate more frequent visits). The 2nd row 1st column displays cross table, as the value of both explanatory variables are binary (0 or 1)

Visiting the restaurant for the first time or not (generating 0 or 1 from a Bernoulli distribution) would be a discrete parameter. To implement this model in Stan, we need to marginalize out this parameter. With marginalization, for a single customer, the model for the number of visits y can be written as follows:

$$\mathrm{ZIB}\big(y|60, q_f, q_r\big) = \begin{cases} \mathrm{Bernoulli}(0|q_c) + \mathrm{Bernoulli}\big(1|q_f\big) \times \mathrm{Binomial}(y=0|60, q_r) & \text{if } y = 0 \\ \mathrm{Bernoulli}\big(1|q_f\big) \times \mathrm{Binomial}(y|60, q_r) & \text{if } y \geq 1 \end{cases} \tag{10.11}$$

The probability of the case $y = 0$ is the probability of generating 0 from the Bernoulli distribution, plus the probability of generating 1 from the Bernoulli distribution followed by generating 0 from the binomial distribution. Integration of $\mathrm{ZIB}\big(y|60, q_f, q_r\big)$ with respect to y equals to 1. Therefore, this is a probability distribution which we call a Zero-Inflated Binomial (ZIB) distribution. A ZIB distribution is a mixture of a Bernoulli distribution and a binomial distribution. Here we would

proceed the analysis assuming the number of visits to the restaurant follows this ZIB distribution.

Generally, the distribution that describes data in which 0 is prominent, is called a Zero-Inflated distribution. Other than a ZIB distribution, a mixture of a Bernoulli distribution and a Poisson distribution is called Zero-Inflated Poisson (ZIP) distribution and is also widely used.

10.3.3 Describe Model Formula

The above mechanism can be written as the following model:

Model Formula 10.7

$$q_f[n] = \text{inv_logit}\left(\left(\mathbf{X}\overrightarrow{b_f}\right)[n]\right) \qquad n = 1, \ldots, N$$

$$q_r[n] = \text{inv_logit}\left(\left(\mathbf{X}\overrightarrow{b_r}\right)[n]\right) \qquad n = 1, \ldots, N$$

$$Y[n] \sim \text{ZIB}\left(60, q_f[n], q_r[n]\right) \qquad n = 1, \ldots, N$$

where N is the number of respondents who answered the questionnaire, and n is the index of each respondent. $\mathbf{X}$ is a matrix with N rows and $(1 + 3)$ columns, created by horizontally stackinghe column of intercept term (with all values equal to 1) and columns of three explanatory variables for each respondent. $\overrightarrow{b_f}$ is a vector of regression coefficients with $(1 + 3)$ elements. Notation using such matrix operation $(\mathbf{X}\overrightarrow{b_f})$ is the same as Model Formula 5.7 in Sect. 5.6. $\overrightarrow{b_r}$ has the same shape as $\overrightarrow{b_f}$. Then we use logistic function (inv_logit) to express the first visit probability (q_f) and the regular visit probability (q_r). By using this model, we can separately estimate $\overrightarrow{b_f}$ and $\overrightarrow{b_r}$ for these two probabilities from the data.

10.3.4 Implement Models

The implementation of Model Formula 10.7 is as follows:

model10-7.stan

```
functions {
  real ZIB_lpmf(int Y, int N, real q_f, real q_r) {
    if (Y == 0) {
      return log_sum_exp(
        bernoulli_lpmf(0 | q_f),
        bernoulli_lpmf(1 | q_f) + binomial_lpmf(0 | N, q_r)
      );
    } else {
      return bernoulli_lpmf(1|q_f) + binomial_lpmf(Y|N, q_r);
```

```
10        }
11      }
12    }
13
14    data {
15      int N;
16      int D;
17      matrix[N,D] X;
18      array[N] int<lower=0> Y;
19    }
20
21    parameters {
22      vector[D] b_f;
23      vector[D] b_r;
24    }
25
26    transformed parameters {
27      vector[N] q_f = inv_logit(X*b_f);
28      vector[N] q_r = inv_logit(X*b_r);
29    }
30
31    model {
32      for (n in 1:N) {
33        Y[n] ~ ZIB(60, q_f[n], q_r[n]);
34      }
35    }
```

Lines 1–12: Defining a ZIB distribution in `functions` block as explained in Sect. 10.2.

Lines 3 and 8: Stan can deal with a conditional statement by using the form:

```
if (CONDITION) { EXPRESSION_1} else { EXPRESSION_2}
```

Here, if `CONDITION` is true then `EXPRESSION_1` is executed, and if `CONDITION` is false then `EXPRESSION_2` is executed. In addition, in Stan, there is another notation:

```
CONDITION ? EXPRESSION_1: EXPRESSION_2
```

and it works in the same way. This notation is called conditional operator or ternary operator.

Line 16: `D` stores the sum of the number of columns for intercept, and the number of explanatory variables, thus $\texttt{D} = 1 + 3 = 4$ in this case.

Line 17: Declaring $\mathbf{X}$ as `matrix` type with N rows and D columns.

Line 33: Using the ZIB distribution defined in `functions` block.

Here we omit R/Python code that run Stan. Please refer **`run-model10-7.R`** and **`run-model10-7.py`** available on the supporting page of this book. In the R/Python code, we divided Age with 10 for scaling.

10.3.5 Interpret Results

The summary of the estimated parameters are as follows:

	Mean	MCSE	StdDev	5%	50%	95%	N_Eff	R_hat
b_f[1]	2.6	1.5e-02	7.9e-01	1.3	2.5	3.9e+00	2588	1.0
b_f[2]	-1.6	8.2e-03	4.3e-01	-2.3	-1.6	-9.1e-01	2754	1.0
b_f[3]	3.3	1.5e-02	8.2e-01	2.2	3.2	4.8e+00	3055	1.0
b_f[4]	-0.37	3.3e-03	1.8e-01	-0.67	-0.37	-8.6e-02	2850	1.00
b_r[1]	-3.4	2.8e-03	1.5e-01	-3.7	-3.4	-3.2e+00	2862	1.0
b_r[2]	0.83	1.4e-03	8.7e-02	0.69	0.83	9.8e-01	3697	1.0
b_r[3]	-0.18	1.2e-03	7.8e-02	-0.31	-0.18	-5.3e-02	4292	1.0
b_r[4]	0.22	6.5e-04	3.5e-02	0.17	0.22	2.8e-01	2960	1.0

`b_f[1]` to `b_f[4]` represent the regression coefficients for intercept, *Sex*, *Sake* and *Age*, respectively. In addition, because we scaled *Age* before fitting the model, we need to multiply *Age* coefficients (i.e., `b_f[4]` and `b_r[4]`) by 10, in order to interpret the *Age* coefficient at its original scale.

The result indicates that "female, drink-sake and younger" people have the higher first visit probability, and "male, non-drink-sake and older" people have the higher regular visit probability. Additionally, the median and 95% Bayesian confidence interval of rank correlation coefficient between `q_f` and `q_r` calculated in **`run-model10-7.R`** and **`run-model10-7.py`** is -0.64 $[-0.81, -0.41]$. Thus, there is a relatively strong negative correlation between the first visit probability and the regular visit probability. This result can be interpreted as, the people who are likely to visit for the first time have lower tendencies to visit again. From the above explanation, we can find that this approach could draw more useful implications as compared with simple multiple regression.

10.4 Supplementary Information and Exercises

In this chapter, we introduced Zero-Inflated distribution. However, it is not applicable to data where the frequency of 0 is prominently low. In such cases, it would be better to use a hurdle model. For instance, a hurdle model with a Poisson distribution uses a mixture of a Bernoulli distribution and a truncated Poisson distribution. Although the hurdle model is not mentioned in this book, interested readers may refer to the "Finite Mixtures" chapter in the Stan reference.

10.4.1 Exercises

(1) In the example of the Bernoulli distribution (coin toss) in Sect. 10.1.2, modify **`model10-1.stan`** to express a case where students were told to roll a dice instead of tossing a coin and tell the truth only when the outcome is "1". Here we assume that the probability of observing each number as the outcome equals 1/6. What is the median and 95% Bayesian confidence interval of the estimated smoking probability q?

(2) In the example of the Bernoulli distribution (coin toss) in Sect. 10.1.2, modify **`model10-1.stan`** to express a case where students were told to toss two coins instead one coin, and tell the truth only when the outcome of both coins are heads. What is the median and 95% Bayesian confidence interval of the estimated smoking probability q, under the following assumptions?

(A) The probability that the outcome is heads is 0.5 for one coin, and 0.4 for the other.

(B) The probability that the outcome is heads is 0.5 for one coin, and p_{coin} (unknown) for the other, where prior distribution of p_{coin} is noninformative.

(3) The following Model Formula 10.8 and Model Formula 10.9 are equivalent:

Model Formula 10.8

$$m \sim \text{Poisson}(\lambda)$$
$$\overrightarrow{Y} \sim \text{Multinomial}(m, \vec{p})$$

Model Formula 10.9

$$Y_k \sim \text{Poisson}(\lambda p_k) \qquad k = 1, \ldots, K-1$$

where Multinomial is a multinomial distribution, $\overrightarrow{p}$ is a K-simplex, and $\overrightarrow{Y}$ is a vector of length K, whose elements are nonnegative integers and the sum of these elements is m. Prove that these two models are equivalent when $K = 3$. (Hint: use the fact that $Y_3 = 1 - Y_1 - Y_2$, $p_3 = 1 - p_1 - p_2$).

(4) Rewrite **`model10-1.stan`** in Sect. 10.1.2 using functions block and estimate the model parameters.

Reference

Warner, S. L. (1965). Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association, 60*(309), 63–69.

Part IV
Advanced Topics for Real-World Data Analysis

Chapter 11
Time Series Data Analysis with State Space Model

A challenge in time series data analysis is that it is an extrapolation problem: we are interested in predicting events that are only observable in the future. For an extrapolation problem, capturing mechanisms using a model which gives persuasive interpretations, usually yields a better prediction performance than using a black box method. In this chapter, we will use state space models for time series data. State space models are known for its high interpretability, and because it can be extended easily, they have a wide range of applications. They are especially suitable for data like daily sales data, of which discrete data points are sampled at the time with equivalent intervals.

The contents of this chapter are as follows:

- In Sect. 11.1, we will use body weight data as an example to introduce the concept of state space models.
- In Sects. 11.2 and 11.3, we will further introduce key components of state space models. Building state space models using their full advantages is possible only after we understand these components.
- In Sect. 11.4, we will change our topic to explaining how to handle missing data, which exists commonly in real-world practice.
- Sections 11.5 to 11.8 provides examples of space state model applications. We will combine what we learn from the earlier part of the chapter, and build relatively complex models.

11.1 Introduction to Space State Models

Let us look at some examples of time series data. Here we have three datasets and plotted their line plots as shown in Fig. 11.1.

For Fig. 11.1 (left), the physical law provides the proportional relationship between time and added heat, and between added heat and the temperature. Therefore, it seems reasonable to fit a linear model for this data. The same applies to Fig. 11.1

© Springer Nature Singapore Pte Ltd. 2022

K. Matsuura, *Bayesian Statistical Modeling with Stan, R, and Python*,
https://doi.org/10.1007/978-981-19-4755-1_11

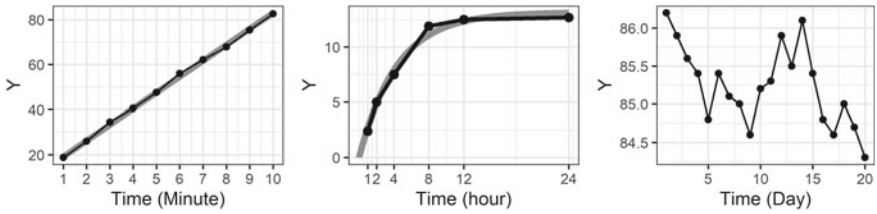


Fig. 11.1 Left: a relationship between the stove's heating time and the water temperature (unit: Celsius). Middle: a relationship between the time elapsed after drug administration and the blood concentration of the drug. Right: daily change of a researcher's body weight (unit: kg) in the most recent 20 days. The thick gray lines on the first two graphs show the theoretically derived curves. The dots represent the data points for all three graphs

(middle), where we know that this data follows certain physical laws. Using differential equations underlying this data, we can derive a theoretical curve (the thick gray line on the plot), which can be used to fit the data (see Sect. 7.2). In addition, the lines can be clearly seen in these plots because the noise is small—another fact that gives us supporting evidence to use a linear model or a theoretical curve to fit the data. In these data, the correlation between the time and the latent state is given clearly. One way to think about these two datasets is that the latent state changes with time and measurements are obtained by adding observation noise to the latent state.

On the other hand, considering mechanisms for Fig. 11.1 (right) is not that straightforward, and we are unaware of the mathematical expressions that connect the time and the latent state. That is, we do not know what curves can fit this data. It does not mean that we have totally no clues about their relationships. At least we know that the latent state is unlikely to have large leaps over time. Rather, the latent state of body weight is more likely to gradually change over time. Therefore, we can add probabilistic fluctuations into the equations between the time and the latent state. In other words, by assuming equations that represent that the latent state changes with time probabilistically, we can consider the state changes and the observations separately, as we did for the previous two datasets (Fig. 11.2).

As this one, when we separately consider that the latent states and the observations, such a model is called a *state space model*. The equation that represents the changes between latent states, which we do not have direct access to, is called a *system model* or a *state equation*. Correspondingly, the equation that represents the observation is called the *observation model* or *observation equation*. From Fig. 11.2, the right arrows represent system model and down arrows represent the observation model.

For the rest of this section, we will apply the state space model to the data shown in Fig. 11.1 (right) (`data-weight.csv`).

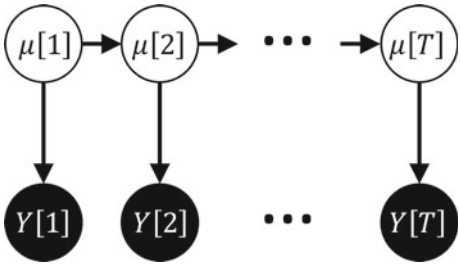


Fig. 11.2 The graphical model for a state space model. Here T is the number of time points, and $[t]$ indicates the time index. A state space model considers the changes between latent states $\mu[t]$ and observations of $Y[t]$ separately

11.1.1 Set Up Purposes

Suppose that the aim of this analysis is to predict the body weight until a certain date (e.g., three days after the last day of observations). We also want to estimate the scale of the fluctuation of the body weight as well as of the observation noise. For the report, we want to show an overlaid plot of the latent state and the 80% Bayesian confidence intervals of the first 20 days of observations, as well as the prediction in the three days following the last day of the observation. Here the reader may think that the observed weight is the "true" value of the weight. However, we assumed this is not the case, and rather, the observed value contains noise such as water and food in the body in addition to the latent weight.

11.1.2 Check Data Distribution

In order to check the data distribution of time series data, we usually check a line plot with time on the x-axis and the observed values on the y-axis as shown in Fig. 11.1 (right).

Alternatively, we can use a *correlogram*, a bar graph with time lags on the x-axis and the sample autocorrelation function (ACF) on the y-axis. The sample ACF $R[k]$ is defined using a lag k as:

$$R[k] = \frac{C[k]}{C[0]}, C[k] = \frac{1}{T}\sum_{t=k+1}^{T}\left(Y[t] - \overline{Y}\right)\left(Y[t-k] - \overline{Y}\right)$$

where $\overline{Y}$ indicates the mean, and is calculated as:

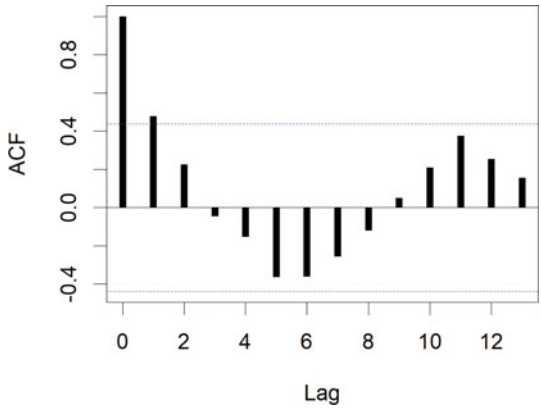


Fig. 11.3 Correlogram of the body weight data

$$\overline{Y} = \frac{1}{T}\sum_{t=1}^{T} Y[t]$$

The sample ACF represents the strength of the association with past values. It is often used to aid in understanding the features of time series data or considering time series models. The sample ACF can be easily computed in R using function `acf`. Figure 11.3 is an example correlogram for body weight data in the right plot of Fig. 11.3.

11.1.3 Imagine the Mechanisms of Data Generation Process

Let us first think about the system model. As mentioned previously, we can assume that the latent states change over time smoothly. There are multiple ways to specifically express this assumption. Here, we will consider that the value $\mu[t]$ at the time point t is very close to the value at the previous time point $\mu[t-1]$. In this case, $\mu[t] - \mu[t-1]$ should be a small value $\varepsilon_\mu[t]$. When representing a small value using a mathematical expression like this, it is common to use a normal distribution. Hence, we consider that $\mu[t] - \mu[t-1]$ follows $\mathcal{N}(0, \sigma_\mu)$, which can be expressed in a mathematical formula as follows:

$$\begin{aligned} \mu[t] - \mu[t-1] &= \varepsilon_\mu[t] \\ \varepsilon_\mu[t] &\sim \mathcal{N}(0, \sigma_\mu) \end{aligned}$$

This can also be written as:

$$\begin{aligned} \mu[t] &= \mu[t-1] + \varepsilon_\mu[t] \\ \varepsilon_\mu[t] &\sim \mathcal{N}(0, \sigma_\mu) \end{aligned}$$

This is equivalent to stating that there is a small probabilistic fluctuation that follows a normal distribution with mean 0 and SD σ_μ, and $\mu[t]$ is generated by adding this fluctuation to its value at the previous time point $\mu[t-1]$.

For the observation model, we consider that $Y[t]$ is generated by adding observation noise to $\mu[t]$. And the observation noise is assumed to follow $\mathcal{N}\big(0, \sigma_y\big)$.

Note that although this body weight data should always be positive, because it can be treated as continuous values, a normal distribution can be used for modeling.

11.1.4 Describe Model Formula

Model Formula 11.1
Based on the discussion above, the model formula can be written as:

$$\begin{aligned}
\mu[t] &= \mu[t-1] + \varepsilon_\mu[t-1] && t = 2, \ldots, T \\
Y[t] &= \mu[t] + \varepsilon_y[t] && t = 1, \ldots, T \\
\varepsilon_\mu[t] &\sim \mathcal{N}\big(0, \sigma_\mu\big) && t = 1, \ldots, T-1 \\
\varepsilon_y[t] &\sim \mathcal{N}\big(0, \sigma_y\big) && t = 1, \ldots, T
\end{aligned}$$

where T represents the number of time points, and t is the index of each time point. For the first time point, as we have no particular information to specify prior for $\mu[1]$, we assign a noninformative prior. By eliminating $\varepsilon_\mu[t]$ and $\varepsilon_y[t]$ from this model formula, Model Formula 11.1 can also be written as:

Model Formula 11.2

$$\mu[t] \sim \mathcal{N}\big(\mu[t-1], \sigma_\mu\big) \qquad t = 2, \ldots, T \tag{11.1}$$

$$Y[t] \sim \mathcal{N}\big(\mu[t], \sigma_y\big) \quad t = 1, \ldots, T \tag{11.2}$$

We will mostly use the form of Model Formula 11.2 in this chapter as the Stan implementation.

11.1.5 Implement the Model

The implementation of Model Formula 11.2 can be as follows:

model11-2.stan

```
1   data {
2     int T;
3     int Tp;
4     vector[T] Y;
5   }
6
7   parameters {
8     vector[T] mu;
9     real<lower=0> s_mu;
10    real<lower=0> s_y;
11  }
12
13  model {
14    mu[2:T] ~ normal(mu[1:(T-1)], s_mu);
15    Y[1:T]  ~ normal(mu[1:T], s_y);
16  }
17
18  generated quantities {
19    vector[T+Tp] mu_all;
20    vector[Tp] yp;
21    mu_all[1:T] = mu[1:T];
22    for (t in 1:Tp) {
23      mu_all[T+t] = normal_rng(mu_all[T+t-1], s_mu);
24      yp[t] = normal_rng(mu_all[T+t], s_y);
25    }
26  }
```

Lines 14–15: These lines correspond to Model Formula 11.2.

Line 14: This line is vectorized and is equivalent to the following statement:

```
for (t in 2:T) {
 mu[t] ~ normal(mu[t-1], s_mu);
}
```

Because we do not specify a prior for `mu[1]`, a noninformative prior will be used, which is the default setting in Stan.

Lines 18–26: `generated quantities` block is used to generate the data points for the following three days.

Line 19: To combine $\mu[t]$ in the entire past days and the next three days, we generate `mu_all`. By doing this, preprocessing the draws and visualizing them would be easier in R or Python.

Line 21: `mu` is assigned for the values at time points from `1` to `T`.

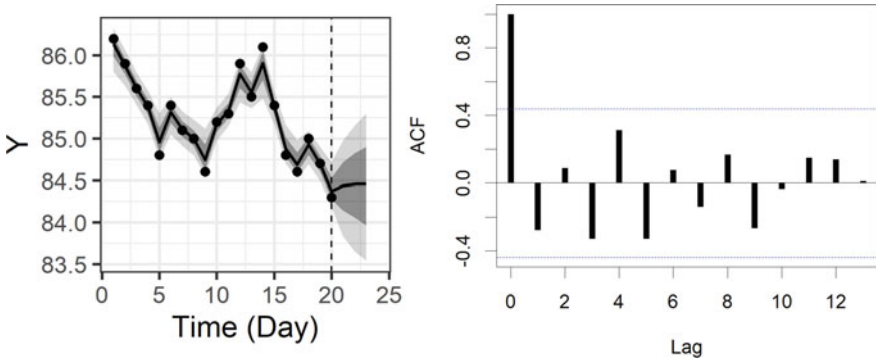


Fig. 11.4 Left: Bayesian confidence intervals for $\mu[t]$. The light gray band is the 80% Bayesian confidence intervals, the dark gray band is the 50% Bayesian confidence intervals, and the thin black line is the medians. The observed data are shown as dots. The points at Time = 21 to 23 is the prediction for the upcoming days. Right: correlogram for the median of the residuals (only the values at Time = 1–20 are used)

Line 23: The values after time points `T + 1` are generated based on the system model.

Line 24: The predicted values `yp` are also generated for the following successive three days.

11.1.6 Interpret the Results

The median and 80% Bayesian confidence intervals for some of the estimated parameters are as follows:

$$\sigma_\mu : 0.38(0.24, 0.52) \quad \sigma_y : 0.20(0.07, 0.37).$$

The plot we wanted in Sect. 11.1.1 is shown in Fig. 11.4 (left). This plot also serves as the PPC. On the other hand, plotting the histogram and correlogram for residuals can help the PPC. We show the correlogram in Fig. 11.4 (right). From this plot, with the consideration that there are not many time points, we concluded that there is no other special component besides the observation noise.

For the next sectionss (Sects. 11.2 and 11.3), we will introduce how to extend system models and observation models for capturing various mechanisms.

11.2 Extending System Model

In the previous section, we assumed that the latent states change smoothly in a system model. That is, $\mu[t]$ is a trend component. The formula for this can be written as:

$$\mu[t] = trend[t] \tag{11.3}$$

Nonetheless, in reality, we often think that there are other effects, besides the trend components. For instance, we might think that the weather effect and some periodic effects such as days of the week are also included in $\mu[t]$. In such a case, we usually decompose $\mu[t]$ into those components with distinct changing patterns, as:

$$\mu[t] = trend[t] + season[t] + weather[t]$$

We then build models for each component based on background knowledge. By doing so, we can interpret the result much easier, and also can comprehend which components have larger impacts on the future predictions.

In this section, we will introduce some components that are frequently used for system model extensions. The illustrations for each component are shown in Table 11.1. As we can see from the plots, some components such as trend components, switch components and pulse components make correlograms less useful, because the sample ACF does not get close to 0 even with a large lag. For instance, for seasonal components, the lags are highly correlated only at certain intervals. For stationary AR components, the autocorrelation changes periodically and becomes closer to 0. Because the observation noises at different time points are independent, there is a strong autocorrelation only at a time point where the lag is 0. Practically, after parameter estimation, we can subtract these components from $Y[t]$ before checking correlograms.

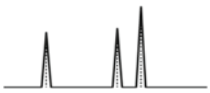
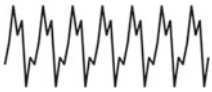
In terms of how these components can be used, we will give examples in Sects. 11.5 to 11.8. When we build models in practice, we suggest starting with a simple model with only a few components. We can then extend this model to a more complex one by adding more components, if it satisfies some of the followings:

- A particular pattern in Table 11.1 is observed in the residual correlogram.
- The change of trend component is not smooth (and the medium- to long-term prediction performance is bad).
- The prediction performance is bad for any other reasons.

11.2.1 Trend Component

There are multiple ways to use model equations to represent the assumption that $trend[t]$ changes smoothly. Here we provide two types of trend components for this assumption.

Table 11.1 Commonly-used components to extend system models, and their correlograms. We show the observation noise as a reference, although it is not a part of a system model

Name of component	Illustration	Correlogram
Trend component		
Regression component		
Seasonal component		
Switch component		
Pulse component		
Stationary AR component		
Observation noise		

The first one is the following formula which we used in the last section:

$$
\begin{aligned}
&trend[t] = trend[t-1] + \varepsilon_{trend}[t] \\
&\varepsilon_{trend}[t] \sim \mathcal{N}(0, \sigma_{trend})
\end{aligned}
$$

eliminating $\varepsilon_{trend}[t]$ from the above formula yields:

$$
trend[t] \sim \mathcal{N}(trend[t-1], \sigma_{trend})
$$

The trend component that follows this equation is called a *first-order difference trend component*. The result in the last section was obtained by using this component on the system model. We may recognize the following limitations from the plot in Fig. 11.4 (left):

- The estimated state changes are bulky, whereas we would expect that the latent state changes are smoother.
- The body weight has the trend to decrease from the day 14 to day 20. However, counterintuitively, the predictions for day 21 to day 23 extend horizontally.

Both of them are caused by the fact that a first-order difference trend component assumes that the value at a time point t is generated from a normal distribution that depends only on the value at $t-1$.

Therefore, we consider representing the assumption on the trend component using the following formula:

$$trend[t] - trend[t-1] = trend[t-1] - trend[t-2] + \varepsilon_{trend}[t] \tag{11.4}$$

$$\varepsilon_{trend}[t] \sim \mathcal{N}(0, \sigma_{trend}) \tag{11.5}$$

This model formula is equivalent to stating that the change of the latent state is smooth. That is, $(trend[t] - trend[t-1]) - (trend[t-1] - trend[t-2])$ is supposed to be a small value $\varepsilon_{trend}[t]$. A trend component that follows this formula is called a *second-order difference trend component.* Equation (11.4) can also be written as:

$$trend[t] = 2\,trend[t-1] - trend[t-2] + \varepsilon_{trend}[t],$$

so we can eliminate $\varepsilon_{trend}[t]$ and write Eqs. (11.4) and (11.5) as follows:

$$trend[t] \sim \mathcal{N}(2\,trend[t-1] - trend[t-2], \sigma_{trend}) \tag{11.6}$$

We define the combination of the decomposition Eq. (11.3), the trend component Eq. (11.6) and the observation model (11.2) as Model Formula 11.3. Because we do not have information for the initial states $trend[t](t = 1, 2)$, we set noninformative priors for them. By fitting this model formula to the data in the last section, and plotting the figures in the same way as Fig. 11.4, we can obtain Fig. 11.5. We see that we successfully overcome the limitations seen in Fig. 11.4

11.2.2 Regression Component

In this book, those components using regressions with explanatory variables are called *regression component*. For example, the effects of weather, event, or advertisement.

As an example, assume that there is daily sales data from an online store. We also have information about whether there is a discount or not for each day (0: no discount, 1: discount). So we will consider a system model including the regression

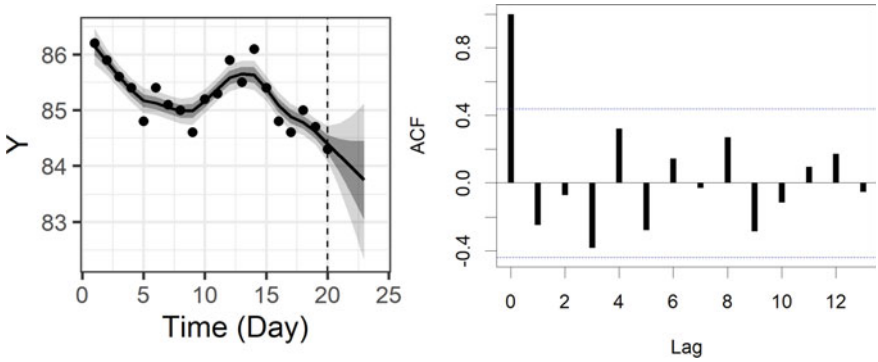


Fig. 11.5 The result of a model using a second-order difference trend component. The legends are the same as Fig. 11.4

component in addition to the trend component. That is, the discount is considered as another explanatory variable. Hence, $\mu[t]$ in this system model can be written as follows:

$$\mu[t] = trend[t] + b\,Discount[t] \qquad t = 1, \ldots, T$$

where b is a parameter representing the regression coefficient.

Let us add more assumptions to this example. Although the above system model assumes that the effect of discount does not change with time, here, let us suppose that the effect of discount changes every day. Then the system model can be written with the time-dependent regression coefficients:

$$\begin{aligned} \mu[t] &= trend[t] + b[t]Discount[t] && t = 1, \ldots, T \\ b[t] &\sim \mathcal{N}(b[t-1], \sigma_b) && t = 2, \ldots, T \end{aligned}$$

$b[t]$ and σ_b will be estimated from the data. Such a coefficient that changes with time is called a *time-varying coefficient*. Including these coefficients makes models be more expressive. Nonetheless, now because the number of parameters increases to the same amount as the number of time points, it may take more time for estimation and may cause MCMC convergence issues.

11.2.3 Seasonal Component

The periodic fluctuations accompanying the change of seasons or time, is called a *seasonal component*. Examples of a seasonal component include the effects of days of the week, the effects of seasons and months. There are multiple ways to express

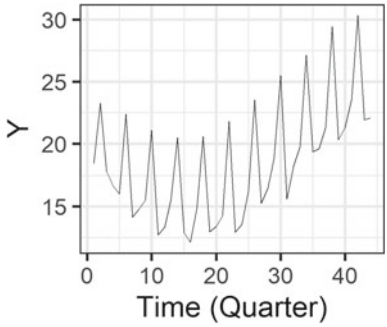


Fig. 11.6 Line plot of quarterly sales data of a seasonal product

the seasonal component, but they can roughly be divided into two groups: zero-sum type and cosine type. We will introduce each of them in the following.

Zero-sum type

In zero-sum type, we declare almost the same number of parameters as the number of periods of seasons. The seasonal component is expressed with the constraint that these parameters add up to zero. One feature of zero-sum type is that it has a high interpretability. It is also called dummy variables type.

As an example, consider time series data of quarterly sales records, which is the record of the purchase numbers for a product (unit: thousand, `data-season.csv`). The data can be plotted as a line plot shown in Fig. 11.6, and we can indeed see that there are periodic fluctuations that change seasonally.

For this data, let us consider a system model by incorporating the seasonal component $season[t]$ in addition to the trend component. Now $\mu[t]$ can be written as:

$$\mu[t] = trend[t] + season[t] \tag{11.7}$$

The seasonal component has periodicity, and the sum of any L successive variables (i.e., $season[t]$, $season[t-1]$, …, $season[t-(L-1)]$) is 0. That is,

$$\sum_{l=0}^{L-1} season[t-l] = 0 \tag{11.8}$$

For this data, the period length L is 4. Equation (11.8) indicates that although the product's sale can be better or worse depending on the seasons, the sum of the seasonal effects at the different time within an entire year can be canceled out to be 0. This makes it easy to interpret how much increase or decrease the sales tend to have for each season. We can implement Eq. (11.8) by declaring the number of parameters minus 1 as the number of periods such that the values of parameters add up to zero.

Here we use the second-order difference trend component, and name the combination of the decomposition Eq. (11.7), the seasonal component Eq. (11.8), the trend component Eq. (11.6), and the observation model Eq. (11.8) as Model Formula 11.4. Its implementation can be as follows:

model11-4.stan

```
1   data {
2     int T;
3     vector[T] Y;
4     int L;
5   }
6
7   parameters {
8     vector[T] trend;
9     vector[L-1] season_raw;
10    real<lower=0> s_trend;
11    real<lower=0> s_y;
12  }
13
14  transformed parameters {
15    vector[T] mu;
16    vector[L] season;
17    season[1:(L-1)] = season_raw[1:(L-1)];
18    season[L] = -sum(season_raw[1:(L-1)]);
19    for (t in 1:T) {
20      mu[t] = trend[t] + season[(t-1)%L + 1];
21    }
22  }
23
24  model {
25    trend[3:T] ~ normal(2*trend[2:(T-1)] - trend[1:(T-2)], s_trend);
26    Y[1:T] ~ normal(mu[1:T], s_y);
27  }
```

Line 16: `season`, which is a `vector` type of length `L`, is declared as the seasonal component.

Lines 9, 17–18: Sum of each element in `season` is assigned to be zero.

Line 20: `(t-1)%L` yields the residuals obtained from dividing `(t-1)` by `L`. By writing it in this way, we can access to the `season` (`season[1]`, `season[2]`, …, `season[L]`, `season[1]`, …) with increasing `t`.

The estimated result of *season*[*t*] is shown in Fig. 11.7 (left). We can see that we successfully separated out the trend component and the seasonal component.

Zero-sum type and fluctuation

Let us see how we can extend the zero-sum type. Here, we relax the constraint of the sum of the seasonal components being exactly 0. Because this relaxation allows the seasonal component to change slightly over time, we can predict the future trend that

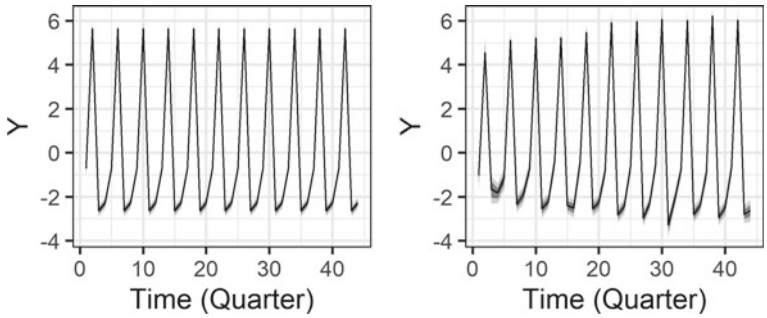


Fig. 11.7 Estimation result of a zero-sum type seasonal component. The legends are the same as those for Fig. 11.4. Unit for the y-axis is thousand. Left: when the sum of periods is exactly zero. Right: when the sum of periods probabilistically fluctuates from 0

reflects the recent information. As a trade-off, the number of parameters will increase and it might result in longer time for estimation, or poor MCMC convergence.

Specifically, we can assume that the seasonal component has period L and the sum of successive L variables ($season[t]$, $season[t-1]$, …, $season[t-(L-1)]$) will be a small value $\varepsilon_{season}[t]$. There are multiple ways to mathematically express the assumption. Among them, we will use a normal distribution and consider that $\sum_{l=0}^{L-1} season[t-l]$ follows $\mathcal{N}(0, \sigma_{season})$. This assumption can be written as:

$$\sum_{l=0}^{L-1} season[t-l] = \varepsilon_{season}[t]$$
$$\varepsilon_{season}[t] \sim \mathcal{N}(0, \sigma_{season}) \tag{11.9}$$

Compare to Eq. (11.8), the right-hand-side has changed from 0 to $\varepsilon_{season}[t]$. Equation (11.9) can also be transformed as:

$$season[t] = -\sum_{l=1}^{L-1} season[t-l] + \varepsilon_{season}[t]$$

It is equivalent to consider that $season[t]$ is generated by adding a probabilistic fluctuation to $-\sum_{l=1}^{L-1} season[t-l]$, and this fluctuation is assumed to follow $\mathcal{N}(0, \sigma_{season})$. Further by eliminating $\varepsilon_{season}[t]$, the seasonal component can be expressed as the following equation:

$$season[t] \sim \mathcal{N}\left(-\sum_{l=1}^{L-1} season[t-l], \sigma_{season}\right) \tag{11.10}$$

Here, we use a second-order difference trend component, and name the combination of the decomposition Eq. (11.7), the seasonal component Eq. (11.10), the trend component Eq. (11.6) and the observation model Eq. (11.2) as Model Formula 11.5. Because no specific knowledge about $trend[t](t = 1, 2)$ or $season[t](t = 1, 2, 3)$ was given, we assign noninformative priors for them. Its implementation can be as follows:

model11-5.stan

```
1   data {
2    int T;
3    vector[T] Y;
4    int L;
5   }
6
7   parameters {
8    vector[T] trend;
9    vector[T] season;
10   real<lower=0> s_trend;
11   real<lower=0> s_season;
12   real<lower=0> s_y;
13  }
14
15  transformed parameters {
16   vector[T] mu = trend[1:T] + season[1:T];
17  }
18
19  model {
20   trend[3:T] ~ normal(2*trend[2:(T-1)] - trend[1:(T-2)], s_trend);
21   for (t in L:T) {
22    season[t] ~ normal(-sum(season[(t-L+1):(t-1)]), s_season);
23   }
24   Y[1:T] ~ normal(mu[1:T], s_y);
25  }
```

Line 9: `season`, which is a `vector` type of length `T`, is declared as the seasonal component.

Lines 21–23: These lines correspond to Eq. (11.10).

The estimated result of $season[t]$ is shown in Fig. 11.7 (right). Compared to Fig. 11.7 (left), we can see that the model can incorporate the fluctuations occurring more recently.

Cosine type

In cosine type, it uses a cosine function and their superposition to express a seasonal component. When the period is not an integer, or when the change can be interpreted easier using a combination of period and amplitude, a cosine type seasonal component can be used.

Using only one cosine function, we can write a seasonal component as:

$$season[t] = a\cos(2\pi f\, t - \theta)$$

where a represents amplitude, f represents frequency (which is the number of waves generated when the time t changes one unit, and $1/f$ equals a period), and θ represents a phase parameter. Sometimes instead of estimating f, we fix it to be a constant value. For some datasets, we can extend a or f to be time-varying coefficients.

When we want to represent a seasonal component with the superposition of multiple cosine functions, f is usually assigned to be a constant value that represents a low frequency wave. By avoiding high-frequency waves and only fitting low-frequency waves, we can avoid overfitting issues.

Examples of a seasonal component with noninteger periods include daily average temperatures and daylight hours, both of which depend on the earth's revolution period (365.25 days). If we use multiple cosine functions, with each has ≤ 6 waves per period, to represent the seasonal component for day t, this can be written as:

$$season[t] = \sum_{i=1}^{6} a[i]\cos\left(2\pi i\frac{t}{365.25} - \theta[i]\right)$$

Parameters $a[i]$ and $\theta[i]$ will be estimated from data. In implementation, it would be necessary to specify a range such that $a > 0$ and $-\pi < \theta < \pi$ for identifiability.

11.2.4 Switch Component

In this subsection, as shown in the 4th row of Table 11.1, we introduce the components that do not change for a while, but sometimes change their values largely, similar to an on/off switch. In this book, we call such a component a *switch component*. We express a switch component using a certain distribution that generates 0 for most of the time, but occasionally generates large numbers. Such a distribution could be Cauchy distribution, Pearson type VII distribution, the mixture of a uniform and a normal distribution, generalized extreme value distribution, etc.

In this subsection, we will use the following system model for the time series dataset (`data-switch.csv`) shown in Fig. 11.8:

$$\mu[t] = switch[t] \tag{11.11}$$

Hereafter, we will represent this $switch[t]$ using a Cauchy distribution or a Pearson type VII distribution.

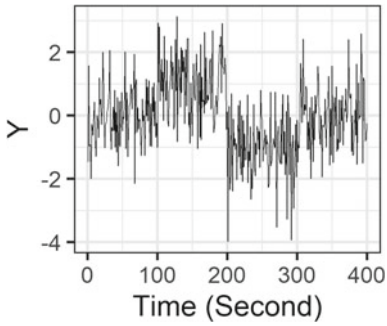


Fig. 11.8 Line plot of a time series dataset that has switching-like changes. We can see abrupt changes at Time = 100, 200, and 300

Cauchy distribution

Using a Cauchy distribution, a switch component can be written as follows:

$$switch[t] \sim \text{Cauchy}(switch[t-1], \sigma_{switch}) \quad t = 2, \ldots, T \tag{11.12}$$

In fact, this equation is obtained by merely replacing the normal distribution with a Cauchy distribution in the first-order difference trend component. Nonetheless, implementing this model will not succeed in estimation—generating values from a Cauchy distribution and using it as a mean of another Cauchy distribution will not yield an efficient estimate of $switch[t]$. This is because the updates of $switch[t]$ in MCMC are too small compared to the length of the tail of the probability distribution. Therefore, the sampling efficiency from the heavy tail part of a Cauchy distribution is very low.

Thus, we will use the reparameterization introduced in Sect. 9.3 and separate the scale from the Cauchy distribution. Because for a Cauchy distribution, its inverse function of cumulative distribution function can be obtained easily, we can apply the inverse transformation method.[1] The cumulative distribution function of a Cauchy distribution is

$$\text{F}(y) = \frac{1}{\pi}\arctan\left(\frac{y-\mu}{\sigma}\right) + 0.5$$

and its inverse function is

$$\text{F}^{-1}(x) = \mu + \sigma\tan(\pi(x - 0.5)) \tag{11.13}$$

Therefore, when x follows Uniform(0, 1), $\text{F}^{-1}(x)$ follows Cauchy(μ, σ).

[1] We omit introduction on how to generate random numbers using inverse functions. Those interested readers can read other references such as Wikipedia page: https://en.wikipedia.org/wiki/Inverse_transform_sampling.

We define the combination of the decomposition Eq. (11.11), the switch component Eq. (11.12), and the observation model Eq. (11.12) as Model Formula 11.6. Its implementation can be as follows:

model11-6.stan

```
1   data {
2    int T;
3    vector[T] Y;
4   }
5
6   parameters {
7    real sw_ini;
8    vector<lower=-pi()/2, upper=pi()/2>[T-1] sw_raw;
9    real<lower=0> s_sw;
10    real<lower=0> s_y;
11   }
12
13   transformed parameters {
14    vector[T] sw;
15    sw[1] = sw_ini;
16    for (t in 2:T) {
17     sw[t] = sw[t-1] + s_sw*tan(sw_raw[t-1]);
18    }
19   }
20
21   model {
22    Y[1:T] ~ normal(sw[1:T], s_y);
23   }
```

Parameter `sw` in line 14 represents the switch component. Equation (11.13) is reflected in lines 8–9 and lines 16–18. Line 8 expresses that $\pi(x - 0.5)$ follows Uniform$(-\pi/2, \pi/2)$ when x follows Uniform$(0, 1)$. The estimated result is shown in Fig. 11.9 (left), from which we can successfully identify when the switching effect occurred.

Pearson type VII distribution

Using a Pearson type VII distribution (see Technical Point in Sect. 6.15), a switch component can be written as the following:

$$switch[t] \sim \text{PearsonVII}(b, switch[t-1], \sigma_{switch}) \quad t = 2, \ldots, T \tag{11.14}$$

Although Pearson type VII distribution is not included as a function in Stan, we can express it by using one of the reparameterization (Devroye, 1985). In brief, when random variables y_1 and y_2 are independent, and $y_1 \sim \mathcal{N}(0, 1)$ and $y_2 \sim$ invGamma$(b, 1)$, then $y = \mu + \sigma y_1 \sqrt{2y_2}$ follows PearsonVII(b, μ, σ).

We define the combination of the decomposition Eq. (11.11), the switch component Eq. (11.14), and the observation model Eq. (11.2) as Model Formula 11.7. The implementation can be as follows:

model11-7.stan

```
1   data {
2     int T;
3     vector[T] Y;
4     real<lower=0> B;
5   }
6
7   parameters {
8     real sw_ini;
9     vector[T-1] sw_raw1;
10    vector<lower=0>[T-1] sw_raw2;
11    real<lower=0> s_sw;
12    real<lower=0> s_y;
13   }
14
15   transformed parameters {
16    vector[T] sw;
17    sw[1] = sw_ini;
18    for (t in 2:T) {
19      sw[t] = sw[t-1] + 1e-5*s_sw*sw_raw1[t-1]*sqrt(sw_raw2[t-1]);
20    }
21   }
22
23   model {
24    sw_raw1[1:(T-1)] ~ normal(0, 1);
25    sw_raw2[1:(T-1)] ~ inv_gamma(B, 1);
26    Y[1:T] ~ normal(sw[1:T], s_y);
27   }
```

Equation (11.14) is reflected in lines 9–11, lines 18–20, and lines 24–25 (`sw_raw1` corresponds to y_1 and `sw_raw2` corresponds to y_2). 1e-5 in line 19 is the scaling constant that constrains `s_sw` to be an order close to 1, so to make MCMC convergence better (this was determined after multiple trials in practice). Parameter `B` in line 4 is a positive parameter for Pearson type VII distribution. A smaller `B` gives a longer tail, and when it is equal to 0.5, this distribution is equivalent to a Cauchy distribution. Here we used `B = 0.2` and estimated the parameters. The result is shown in Fig. 11.9 (right). Although it is hard to visually detect the differences from these given plots, compared to the one using a Cauchy distribution, the estimated *switch*[*t*] except the switching time points have become flatter.

11.2.5 Pulse Component

In this subsection, we introduce a type of component that has a rapid increase, followed by a gradual return to its baseline levels. In this book, a component that changes like a pulse is called a *pulse component*. We will introduce two methods that represent the pulse component, among multiple approaches that can be used.

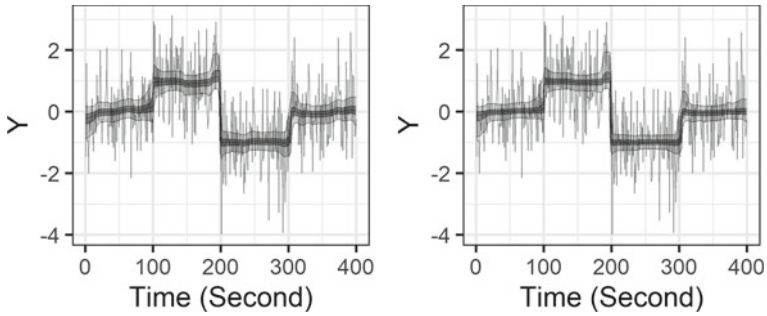


Fig. 11.9 Estimated *switch*[*t*]. The light gray bands are the 95% Bayesian confidence intervals, the dark gray bands are the 50% Bayesian confidence intervals, the thin black lines are the medians, and the thin gray lines are the observed data. Left: the result when using a Cauchy distribution. Right: the result when using a Pearson type VII distribution

The first method uses a Cauchy distribution, as we did for a switch component in the previous subsection. The second one uses a mixture of normal distributions.

In this subsection, we will use the following system model for a time series dataset (`data-pulse.csv`) shown in Fig. 11.10.

$$\mu[t] = pulse[t] \tag{11.15}$$

Hereafter, we will represent this *pulse*[*t*] using a Cauchy distribution or a mixture of normal distributions.

Cauchy distribution

Using a Cauchy distribution, the pulse component can be written as:

$$pulse[t] \sim \text{Cauchy}\big(\rho\, pulse[t-1], \sigma_{pulse}\big) \quad t = 2, \ldots, T \tag{11.16}$$

The only difference from the switch component in Eq. (11.12) is that parameter $(0 < \rho < 1)$, which represents the decreasing speed, is multiplied by $pulse[t-1]$.

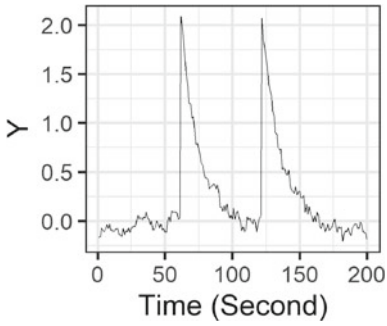


Fig. 11.10 Line plot of this time series dataset that has pulsing-like changes

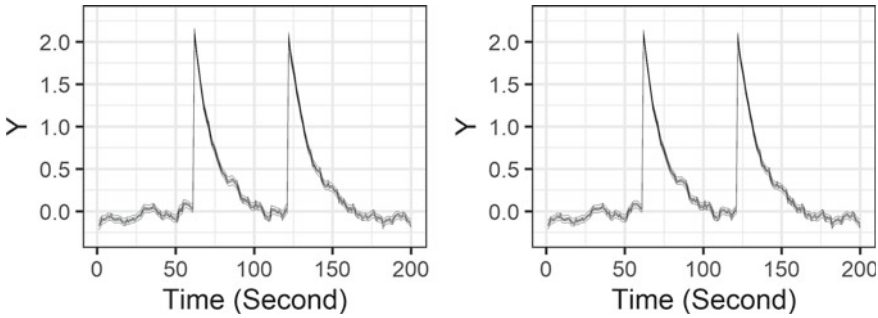


Fig. 11.11 Estimation of $pulse[t]$. The legends are the same as Fig. 11.9. Left: the result when using a Cauchy distribution. Right: the result when using a mixture of normal distributions

We define the combination of the decomposition Eq. (11.15), the pulse component Eq. (11.16), and the observation model Eq. (11.2) as Model Formula 11.8. We omit its implementation here, and show only the estimated result in Fig. 11.11 (left).

By using a Cauchy distribution, we can estimate when a pulsing event happens. However, this is not suitable in a case when the amplitude of the pulse needs to be estimated, in which case we can use a mixture of normal distributions.

A mixture of normal distributions

To use a mixture of normal distributions for the pulse component, we have:

$$pulse[t] \sim q\,\mathcal{N}\big(\rho\, pulse[t-1] + \beta, \sigma_{pulse}\big) + (1-q)\mathcal{N}\big(\rho\, pulse[t-1], \sigma_{pulse}\big) \quad t = 2, \ldots, T \tag{11.17}$$

This indicates that a pulse with height β is generated with a probability q. Here it is assumed that SD σ_{pulse} is a constant regardless of the occurrence of a pulsing event.

We define the combination of the decomposition Eq. (11.15), the pulse component Eq. (11.17), and the observation model Eq. (11.2) as Model Formula 11.9. Its implementation can be as follows:

model11-9.stan

```
data {
  int T;
  vector[T] Y;
}

parameters {
  vector[T] pulse;
  real<lower=0, upper=1> q;
  real<lower=0> beta;
  real<lower=0, upper=1> rho;
  real<lower=0> s_pulse;
  real<lower=0> s_y;
```

```
13   }
14
15   model {
16     for (t in 2:T) {
17       target += log_mix(q,
18         normal_lpdf(pulse[t] | rho*pulse[t-1] + beta, s_pulse),
19         normal_lpdf(pulse[t] | rho*pulse[t-1],     s_pulse)
20       );
21     }
22     beta ~ normal(2, 1);
23     q ~ normal(0, 0.1);
24     Y[1:T] ~ normal(pulse[1:T], s_y);
25   }
```

The pulse component is declared as `pulse` in line 7. Equation (11.17) is reflected in lines 17–20 (see this chapter for the details on a mixture distribution). Weakly informative priors are specified in lines 22–23. The estimation result is shown in Fig. 11.11 (right), and the estimated medians and 95% Bayesian confidence intervals for q and β are:

$$q : 0.013(0.003 - 0.035) \qquad \beta : 2.05(1.99 - 2.12)$$

11.2.6 Stationary AR Component

In this subsection, we introduce the *stationary AR (autoregressive) component*. Sometimes we may observe that with an assumption $\mu[t] = trend[t] + season[t]$, the estimated trend component moves rapidly over time. This estimation is not suitable for long-term predictions, as we would expect a trend component to change slowly and smoothly. When such a result is observed, a stationary AR component may be introduced to slow down the change in the trend component. This is helpful because a stationary AR component can express those periodic changes that have short time scales, which cannot be fully described by trend component or seasonal component. As its name suggests, $ar[t]$, a stationary AR component, follows this S-order AR model:

$$ar[t] \sim \mathcal{N}\left(\sum_{s=1}^{S} \eta[s] ar[t-s], \sigma_{ar}\right)$$

When $S \geq 2$, we can incorporate a periodic autocorrelation function into $ar[t]$. Partly because of this, a commonly used AR component is the one that follows a second-order AR model:

$$ar[t] \sim \mathcal{N}(\eta[1] ar[t-1] + \eta[2] ar[t-2], \sigma_{ar}) \tag{11.18}$$

We would expect this component to be both stationary[2] and periodic. For the AR component to have a stationary property, it should satisfy:

$$-1 < \eta[2] < 1, \quad \text{and} \quad \eta[2] - 1 < \eta[1] < -\eta[2] + 1.$$

For the AR component to have a periodic autocorrelation, it should satisfy:

$$\eta[2] < -\eta[1]^2/4$$

Combining these two conditions, we get:

$$-2 < \eta[1] < 2, \quad \text{and} \quad -1 < \eta[2] < -\eta[1]^2/4 \tag{11.19}$$

Therefore, to define a stationary AR component that describes a periodic change using Stan, we can specify its range in Eq. (11.19).

Additionally, when $\eta[1] = 2$ and $\eta[2] = -1$ in the second-order AR model, it becomes a nonstationary model and the equation becomes identical to the one for the second-order difference trend component.

11.2.7 Reparameterization of Component

Because the formulas of the components generally have the form

$$x[t] \sim \mathcal{N}\left(\sum_{l=1}^{L} a[l]x[t-l], \sigma_x\right),$$

it is often possible to improve convergence by separating out the scales of the parameters from the distribution (see Sect. 9.3), as in the following form:

$$x[t] = \left(\sum_{l=1}^{L} a[l]x[t-l]\right) + \sigma_x \times x_{\text{raw}}[t]$$
$$x_{\text{raw}}[t] \sim \mathcal{N}(0, 1)$$

In this section, we show two examples of reparametrizing the trend component and the seasonal component.

A part of the code that reparameterizes **`model11-2.stan`** (the first-order difference trend component) is shown as follows:

[2] The stationary property, in brief, indicates a property of time series data of which the mean and SD do not change over time.

```
1   parameters {
2     real mu_ini;
3     vector[T-1] mu_raw;
4     real<lower=0> s_mu;
5     real<lower=0> s_y;
6   }
7
8   transformed parameters {
9     vector[T] mu;
10    mu[1] = mu_ini;
11    for (t in 2:T) {
12      mu[t] = mu[t-1] + s_mu*mu_raw[t-1];
13    }
14  }
15
16  model {
17    mu_raw[1:(T-1)] ~ normal(0, 1);
18    Y[1:T] ~ normal(mu[1:T], s_y);
19  }
```

Line 2: To define `mu` smoothly in `transformed parameters` block (lines 9–13), `mu_ini` (that means initial `mu`) is declared.

Line 3: the raw parameter is declared.

Line 17: the raw parameter is specified to follow $\mathcal{N}(0, 1)$. Hereafter, we will omit the index of a raw parameter (i.e., `[1:(T-1)]` in this line) for simplicity.

In this case, however, convergence is worse. It is not easy to determine whether reparameterization is effective for the model before the sampling. We think that it is better to try reparameterization only when the convergence was poor.

The code that reparameterizes **`model11-5.stan`** (the second-order difference trend component and the seasonal component) is shown as follows:

```
data {
  int T;
  vector[T] Y;
  int L;
}

parameters {
  vector[2]  trend_ini;
  vector[T-2] trend_raw;
  vector[L-1]  season_ini;
  vector[T-L+1] season_raw;
  real<lower=0> s_trend;
  real<lower=0> s_season;
  real<lower=0> s_y;
}

transformed parameters {
  vector[T] trend;
  vector[T] season;
```

```
20     vector[T] mu;
21     trend[1:2] = trend_ini[1:2];
22     for (t in 3:T) {
23      trend[t] = 2*trend[t-1] - trend[t-2] +
24        s_trend*trend_raw[t-2];
25     }
26     season[1:(L-1)] = season_ini[1:(L-1)];
27     for (t in L:T) {
28      season[t] = -sum(season[(t-L+1):(t-1)]) +
29        s_season*season_raw[t-L+1];
30     }
31     mu[1:T] = trend[1:T] + season[1:T];
32    }
33
34    model {
35     trend_raw ~ normal(0,1);
36     season_raw ~ normal(0, 1);
37     Y[1:T] ~ normal(mu[1:T], s_y);
38    }
```

Lines 8–9, 18, 21–25: These are the reparameterization of the second-order difference trend component.

Lines 10–11, 19, 26–30: These are the reparameterization of the seasonal component. We will use this reparameterization in Sects. 11.7 and 11.8.

11.3 Extending the Observation Model

So far, we have introduced how to extend the system model, and for the observation model, we used this one for all the examples:

$$Y[t] \sim \mathcal{N}\big(\mu[t], \sigma_y\big)$$

Now in this section, we will introduce some extensions for this observation model. Specifically, we will be using other distributions rather than a normal distribution. These are often used when $Y[t]$ includes some outliers, or when the data points are discrete values or vectors. Let us look at these cases one by one.

11.3.1 Outliers

As we addressed in Sect. 7.8, when we want to avoid the posterior means being pulled by the outliers, we can use a distribution with a long tail, such as a t-distribution or a Cauchy distribution, for instance:

$$Y[t] \sim \text{Student_t}\big(6, \mu[t], \sigma_y\big)$$

11.3.2 Binary Values

When the data is binary, we can use either a Bernoulli distribution or a Binomial distribution. The range of $\mu[t]$ must be constrained so that it can be assigned to be a parameter of these distributions. The most common way to do this is by applying what we did for logistic regression—use the `inv_logit` function to transform $\mu[t]$ into the range (0, 1), as:

$$q[t] = \text{inv_logit}(\mu[t])$$
$$Y[t] \sim \text{Bernoulli}(q[t])$$

One caveat for the implementation is that without parameter constraints such as $-5 \leq \mu[t] \leq 5$ or a weakly informative prior, this model may have poor MCMC convergence.

Similarly, for data with multiple values, we can use `softmax` function, as

$$\overrightarrow{q[t]} = \text{softmax}\Big(\overrightarrow{\mu[t]}\Big)$$
$$Y[t] \sim \text{Categorical}\Big(\overrightarrow{q[t]}\Big)$$

11.3.3 Count Data

Let us consider a case where the observation is count data. If there is an upper limit for the values, we can use a Binomial distribution. If there are large enough data, a normal distribution may be a better alternative. If neither of these is the case, usually we apply what we did for the case with Poisson regression. That is, use `exp` function and transform $\mu[t]$ into a positive value, and then use a Poisson distribution.

$$\lambda[t] = \exp(\mu[t])$$
$$Y[t] \sim \text{Poisson}(\lambda[t])$$

11.3.4 Vector

If the observation is a vector with multiple elements, we use a multivariate normal distribution instead of a normal distribution. Additionally, trend component is usually specified with a multivariate normal distribution as well, with a state represented as a vector. Model formula for these can be written as:

$$\begin{aligned}
\overrightarrow{\mu[t]} &= \overrightarrow{trend[t]} \\
\overrightarrow{trend[t]} &\sim \text{MultiNormal}\left(\overrightarrow{trend[t-1]}, \Sigma_\mu\right) \\
\overrightarrow{Y[t]} &\sim \text{MultiNormal}\left(\overrightarrow{\mu[t]}, \Sigma_y\right)
\end{aligned}$$

We can consider using a second-order difference trend component for $\overrightarrow{trend[t]}$, as in Sect. 11.2.1.

11.4 State Space Model with Missing Data

In this section, we consider the case where there are missing values in the observed time series data. Such data can also be handled easily using state space model.

11.4.1 Observations at Certain Time Points are Missing

In such a case, we consider the system model for all time points, but for the observation model, we consider only those time points at which data points are available. This is equivalent to ignoring the likelihood of the observation model at the other time points.

11.4.2 Time Intervals are not the Same (Unequal Intervals)

There are multiple ways for such a case, but a simple way is to approximate those time points where data points are observed, and force them to be equivalent to the case in the previous subsection. Specifically, we first discretize the time space into small and equally-spaced intervals. Subsequently, each time point where observed data is available is approximated by the closest point on the grid. We consider all points for the system model, and use the limited points where the data are available for the observation model, and then treat the rest of the points as missing data.

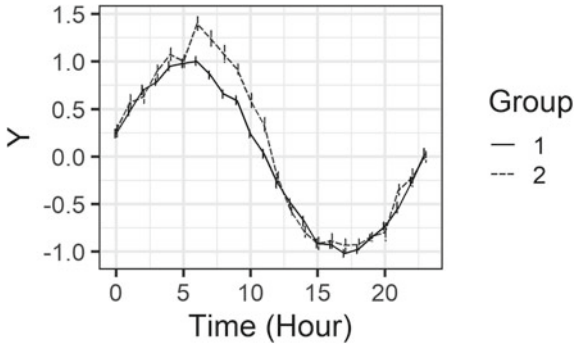


Fig. 11.12 Time series data of the protein blood concentration Y for two groups

11.4.3 Vector

Again, we consider the system model using all time points. When all elements of a vector are missing at certain time points, we can consider the observation model only at points where data points are observed, as we did in Sect. 11.4.1. Otherwise, for observation model at points where vectors with only some elements are missing, we can use a (multivariate) normal distribution, obtained by marginalizing the distribution over missing elements (see Sect. 6.14).

11.5 (Example 1) Difference Between Two Time Series

From Sects. 11.5 to 11.8, we will put what we have learned so far together and apply them to some examples. In this section, we give a modeling example to consider whether there is a difference between two time series.

Suppose that there are 50 subjects that are divided into two groups. The subjects in group 2 were given a chemical therapy and the subjects in group 1 were not. The blood concentrations of a certain protein were measured hourly, with 24 total time points: from the time when a therapy was given (referred as 0 h), to the time 23 h after the therapy. The data is saved as `data-eg1.csv`. In Fig. 11.12, we visualized the line plot which shows the means and standard error (SE) (the bars indicate mean $\pm$ SE) at each time point for each group. At the time interval between 6 and 12 h after the therapy, we can see that the therapy induced a change that resembles a switching effect. The purpose of the analysis is to check when there is a difference and how much the difference is between these two groups.[3]

[3] A frequently used method is to use a t-test for each time point, followed by multiple comparison adjustment. The limitation of this method is that it ignores the correlation between the measurements at nearby time points. Also, even if it can determine whether there is a difference, it is not able to tell when the difference occurs, neither it can quantify the difference.

Now let us consider a possible mechanism that generated this data. Here, we use only a trend component for the states of group 1. For the states of group 2, we use the sum of two parts: the trend component that is shared with group 1, and a switch component that represents the difference from group 1. The time series data for each subject is then considered to be generated by adding observation noise (shared between two groups) to these states. These can be written as a model formula as:

Model Formula 11.10

$$
\begin{aligned}
\mu_1[t] &= trend[t] & & t = 1, \ldots, T \\
\mu_2[t] &= trend[t] + switch[t] & & t = 1, \ldots, T \\
Y_1[n, t] &\sim \mathcal{N}\big(\mu_1[t], \sigma_y\big) & n = 1, \ldots, N_1 \quad & t = 1, \ldots, T \\
Y_2[n, t] &\sim \mathcal{N}\big(\mu_2[t], \sigma_y\big) & n = 1, \ldots, N_2 \quad & t = 1, \ldots, T
\end{aligned}
$$

(Equations for $trend[t]$ and $switch[t]$ are omitted)

where T represents the number of time points, N_1 represents the number of subjects in group 1, μ_1 represents the states of group 1, and Y_1 represents the observed data for group 1. Similarly, N_2, μ_2 and Y_2 represents those of group 2. By expressing the model in this way, the differences between two time series data are incorporated into the switch component, from which we can estimate when there is a difference and how much it is, by checking the posterior probability.

Its implementation is as follows. We used a second-order difference trend component, and a Cauchy distribution to represent the switch component:

model11-10.stan

```
data {
  int T;
  int N1;
  int N2;
  array[N1] vector[T] Y1;
  array[N2] vector[T] Y2;
}

parameters {
  vector[T] trend;
  real sw_ini;
  vector<lower=-pi()/2, upper=pi()/2>[T-1] sw_unif;
  real<lower=0> s_trend;
  real<lower=0> s_sw;
  real<lower=0> s_y;
 }

 transformed parameters {
  vector[T] sw;
  sw[1] = sw_ini;
  for (t in 2:T) {
    sw[t] = sw[t-1] + s_sw*tan(sw_unif[t-1]);
```

```
23      }
24    }
25
26    model {
27      trend[3:T] ~ normal(2*trend[2:(T-1)] - trend[1:(T-2)], s_trend);
28      for (n in 1:N1) {
29        Y1[n] ~ normal(trend[1:T], s_y);
30      }
31      for (n in 1:N2) {
32        Y2[n] ~ normal(trend[1:T] + sw[1:T], s_y);
33      }
34    }
```

Lines 28–33 represent that the time series data for each subject follows normal distributions. In lines 5–6, Y1 and Y2 are passed as wide-format data from R or Python. Implementation for the switch component is the same as Sect. 11.2.4.

The estimated trend component and switch component are visualized in Fig. 11.13. There are time points where the 95% Bayesian confidence intervals of the switch component does not include zero. We can define these points as the time where the difference occurs. From the estimated result, we can say that there is a difference between 6 and 11 h after the therapy. Also, by looking at the probability, we can check how much difference exists between them.

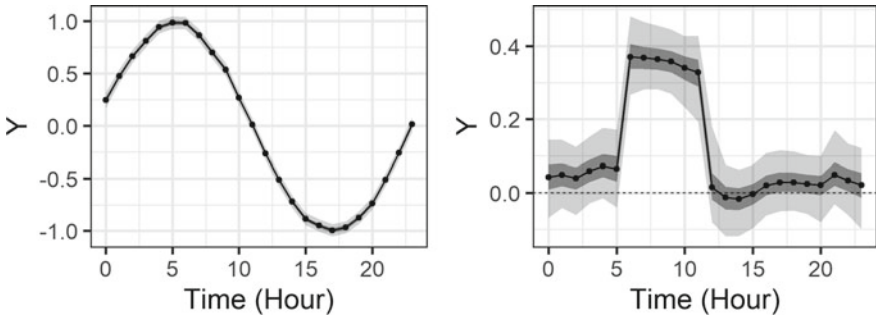


Fig. 11.13 Left: the Bayesian confidence intervals of *trend*[*t*]. Right: the Bayesian confidence intervals of *switch*[*t*]. For both plots, the light gray bands are the 95% Bayesian confidence intervals, dark gray bands are the 50% Bayesian confidence intervals, and the thin black lines, as well as the black dots are the medians

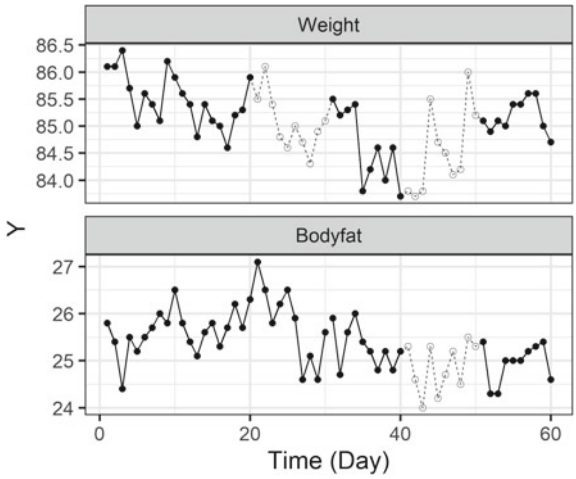


Fig. 11.14 Transitions of body weight and body fat over time. Data that are removed on purpose for the later analysis are shown as gray dotted lines

11.6 (Example 2) Changes in Body Weight and Body Fat

In this section, we give a modeling example where we have two correlated time series with missing observations. This example includes time series data from a researcher's body weight (unit: kg) and body fat (unit: %) records (Nomura, 2016).[4] Daily body weight and body fat were recorded at the same time every day for 60 consecutive days (`data-eg2.csv`). The body weight data from day 21 to 30 were removed on purpose. Likewise, both body weight and body fat data from day 41 to 50 were removed. Let us try to estimate the states. Figure 11.14 shows the line plot of the body weight and body fat data.

Because there is likely to be a correlation between body weight and body fat, as we mentioned in Sect. 11.3.4, we will use a bivariate normal distribution for the system model and the observation model. The model formula is shown as follows. Note that we used a first-order difference trend component and denoted it as $\vec{\mu}$.

Model Formula 11.11

$$\overrightarrow{\mu[t]} \sim \text{MultiNormal}\left(\overrightarrow{\mu[t-1]}, \Sigma_{\mu}\right) \qquad t = 1, \ldots, T \tag{11.20}$$

When there are both body weight and body fat data are observed at a time point:

$$\overrightarrow{Y[t_{obs}2t[t_{obs}]]} \sim \text{MultiNormal}\left(\overrightarrow{\mu[t_{obs}2t[t_{obs}]]}, \Sigma_{y}\right) \qquad t_{obs} = 1, \ldots, T_{obs} \tag{11.21}$$

When only body fat data is observed at a time point:

$$Y_{bf}\left[t_{bf}2t\left[t_{bf}\right]\right] \sim \mathcal{N}\left(\mu_{bf}\left[t_{bf}2t\left[t_{bf}\right]\right], \sigma_{bf}\right) \qquad t_{bf} = 1, \ldots, T_{bf} \tag{11.22}$$

[4] The weight data used in Sect. 11.1 is obtained by extracting the day 9 to day 28 from the data used in this section.

where T represents the number of total time points, T_{obs} represents the number of time points where both body weight and body fat data are observed, and t_{obs} represents their index. $t_{obs}2t$ (an integer from 1 to T) maps the index t_{obs} to the index t. T_{bf} represents the number of time points where only body fat data are observed,[5] and t_{bf} represents the index. $t_{bf}2t$ maps the index t_{bf} to the index t as well. In addition, we have $\vec{\mu} = \left(\mu_{weight}, \mu_{bf}\right)$ and $\sigma_{bf} = \sqrt{\Sigma_y[2,2]}$. As explained in Sect. 11.4.3, Eq. (11.2) uses the marginalized distribution of the multivariate normal distribution for μ_{bf}. Note that the observation model is not used for those time points where neither body weight nor body fat data are available.

The implementation can be as follows:

model11-11.stan

```
1   data {
2    int T;
3    int T_obs;
4    int T_bf;
5    array[T_obs] int t_obs2t;
6    array[T_bf] int t_bf2t;
7    array[T_obs] vector[2] Y_obs;
8    vector[T_bf] Y_bf;
9    real Nu;
10   }
11
12   parameters {
13    array[T] vector[2] mu;
14    cholesky_factor_corr[2] corr_chol_mu;
15    cholesky_factor_corr[2] corr_chol_y;
16    vector<lower=0>[2] s_mu_vec;
17    vector<lower=0>[2] s_y_vec;
18   }
19
20   transformed parameters {
21   matrix[2,2] cov_chol_mu = diag_pre_multiply(s_mu_vec, corr_chol_mu);
22   matrix[2,2] cov_chol_y = diag_pre_multiply(s_y_vec, corr_chol_y);
23   }
24
25   model {
26    mu[2:T,] ~ multi_normal_cholesky(mu[1:(T-1),], cov_chol_mu);
27    Y_obs[1:T_obs,] ~ multi_normal_cholesky(mu[t_obs2t,], cov_chol_y);
28    Y_bf[1:T_bf] ~ normal(mu[t_bf2t, 2], s_y_vec[2]);
29    corr_chol_mu ~ lkj_corr_cholesky(Nu);
30    corr_chol_y ~ lkj_corr_cholesky(Nu);
31    s_mu_vec ~ student_t(6, 0, 0.05);
32    s_y_vec ~ student_t(6, 0, 0.1);
33   }
```

Lines 26, 27, and 28 correspond to Eqs. (11.20), (11.21) and (11.22), respectively. Recall that `multi_normal_cholesky` is a way to represent a multivariate normal distribution, which helps to speed up the computation. For further explanation, see Sect. 9.2.4, where we had a more detailed explanation of relevant functions

[5] "`bf`" stands for Body Fat.

such as `lkj_corr_cholesky`. Line 31 is an added assumption that the changes of the latent weight and fat are likely to be at most 0.5 kg and 0.5%, respectively. Similarly, line 32 indicates that the observation noises for the weight and fat are likely to be at most 1 kg and 1%, respectively. They are on the scale of 1/10, because when passing `Y_obs` and `Y_bf` data to Stan, we divide them by 10 for scaling purpose.

A part of the R/Python code to run this Stan code can be:

A part of code **`run-model11-11.R`**

```
1   d <- read.csv('input/data-eg2.csv')
2   T <- 60
3   t_obs2t <- c(1:20, 31:40, 51:60)
4   T_obs <- length(t_obs2t)
5   t_bf2t <- 21:30
6   T_bf <- length(t_bf2t)
7   Y_obs <- d[t_obs2t, c('Weight', 'Bodyfat')]/10
8   Y_bf <- d[t_bf2t, 'Bodyfat']/10
```

A part of code **`run-model11-11.py`**

```
1   d = pandas.read_csv('input/data-eg2.csv')
2   T = 60
3   t_obs2t = list([*range(1,21), *range(31,41), *range(51,61)])
4   T_obs = len(t_obs2t)
5   t_bf2t = list(range(21,30))
6   T_bf = len(t_bf2t)
7   Y_obs = d.loc[np.array(t_obs2t) - 1, ['Weight','Bodyfat']]/10
8   Y_bf = d.loc[np.array(t_bf2t) - 1].Bodyfat/10
```

`t_obs2t` in line 3 denotes $t_{obs}2t$. Similarly, `t_bf2t` in line 5 denotes $t_{bf}2t$. In lines 7–8, data are divided by 10 for scaling purpose (see Sect. 5.1.6).

Figure 11.15 shows the estimation result of $\overrightarrow{\mu[t]}$. Note that for those points where both body weight and body fat data are missing (Time = 41–50), the estimated line is almost straight. In contrast, for those points where only body weight data are missing, the estimated body weight incorporates information about body fat, as illustrated from the plots.

11.7 (Example 3) The Transition of Tennis Players' Capabilities

In Sect. 9.2.2, we estimated capabilities of tennis players from actual data of professional men's tennis in 2017. In this section, we will extend this example and use the data recorded through 22 years (2000–2021)[6] (in total 442 players[7] and 48,988 games), and estimate players' capabilities using this data.

[6] http://www.tennis-data.co.uk.

[7] Only those players who played more than 30 games for at least 5 years were included in our data.

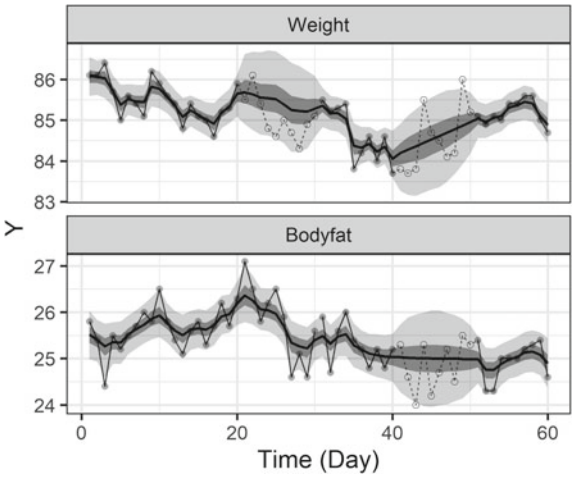


Fig. 11.15 Estimated $\overrightarrow{\mu[t]}$. The light gray bands are the 95% Bayesian confidence intervals, the dark gray bands are the 50% Bayesian confidence intervals, and the thin black lines are the medians. The dots and lines that connect two dots are the observed data

Recall that we used the following model in Sect. 9.2.2:

Model Formula 9.3

$$
\begin{aligned}
l_{\text{game}} &= \prod_{g=1}^{G} \text{Prob}\big[performance_L[g] < performance_W[g]\big] \\
performance_L[g] &\sim \mathcal{N}\big(\mu[g2L[g]], \sigma[g2L[g]]\big) && g = 1, \ldots, G \\
performance_W[g] &\sim \mathcal{N}\big(\mu[g2W[g]], \sigma[g2W[g]]\big) && g = 1, \ldots, G \\
\mu[n] &\sim \mathcal{N}\big(0, \sigma_\mu\big) && n = 1, \ldots, N \\
\sigma[n] &\sim \text{Gamma}(10, 10) && n = 1, \ldots, N
\end{aligned}
$$

where G represents the total number of games, g represents index for each game, l_{game} represents the likelihood of all game results. Parameters $performance_L[g]$ and $performance_W[g]$ represent the performance of loser and winner for the game g, respectively. Parameters $g2L[g]$ and $g2W[g]$ represent the indices of loser and winner for the game g, respectively. N represents the total number of players and n represents index for each player. In Sect. 9.2.2, we used this model to compute game result likelihood for each game, by eliminating $performance_L[g]$ and $performance_W[g]$, and then estimated parameters $\mu[n]$, $\sigma[n]$, and σ_μ from the data.

In this section, in order to estimate the transition of the players' capabilities, we add the dependency of $\mu[n]$ on time. Here, we assume that each of the players' capabilities $\mu[n]$ is a first-order difference trend component, and each of the players' performance fluctuation $\sigma[n]$ does not depend on time. Nevertheless, because different players have played for different numbers of years, a two-dimensional matrix such

$$\vec{\mu} = \{\underbrace{\mu[1], \mu[2], \ldots, \mu[11]}_{\substack{\text{capability of} \\ \text{Acasuso J.} \\ \text{in 2001-2011}}}, \underbrace{\mu[12], \ldots, \mu[18]}_{\substack{\text{capability of} \\ \text{Agassi A.} \\ \text{in 2000-2006}}}, \mu[19], \ldots, \mu[4691]\}$$

$\bullet\bullet\bullet$

Fig. 11.16 Illustration of $\mu[i]$

as $\mu[n, t]$ would be difficult to converge. Therefore, we consider $\mu[i]$, which is a one-dimensional vector of the players' transitions (Fig. 11.16).

The structure of Data File 9.3 was also changed to include the indices in $\vec{\mu}$ (Data File 11.1 and 11.2). Data File 11.1 shows the active years and their corresponding indices in $\vec{\mu}$ of each player. For example, Agassi A. played from 2000 to 2006, and his capabilities in 2000, 2001, and 2006 are stored in $\mu[12]$, $\mu[13]$, and $\mu[18]$, respectively. Data File 11.2 shows the player IDs and indices in $\overrightarrow{\mu}$ for the loser and winner in each game.

Data File 11.1 Structure of `data-eg3-players.csv` (a row corresponds to a player)

```
  1  PID,Player,Start_Year,End_Year,Start_Index,End_Index
  2  1,Acasuso J.,2001,2011,1,11
  3  2,Agassi A.,2000,2006,12,18
  4  3,Albot R.,2013,2020,19,26
     ...
423  422,Zverev M.,2006,2020,4393,4407
```

Data File 11.2 Structure of `data-eg3.csv` (a row corresponds to a game)

```
    1  Year,Loser_PID,Winner_PID,Loser_Index,Winner_Index
    2  2000,69,113,694,1123
    3  2000,14,132,138,1328
    4  2000,364,251,3809,2612
       ...
46696  2020,377,259,3948,2716
```

Based on the above discussion and indices, the model formula can be written as:

Model Formula 11.12

$$
\begin{aligned}
& l_{\text{game}} = \prod_{g=1}^{G} \text{Prob}\big[performance_L[g] < performance_W[g]\big] \\
& performance_W[g] \sim \mathcal{N}\big(\mu[g2Wi[g]], \sigma[g2Wn[g]]\big) \quad g = 1, \ldots, G \\
& performance_W[g] \sim \mathcal{N}\big(\mu[g2Wi[g]], \sigma[g2Wn[g]]\big) \quad g = 1, \ldots, G \\
& \text{for } i \in \text{index of the first year for each player:} \\
& \mu[i] \sim \text{Student_t}^{+}(4, 0, 1)
\end{aligned}
\tag{11.23}
$$

$$
\begin{aligned}
& \text{for } i \in \text{index of the second to last year for each player:} \\
& \mu[i] \sim \mathcal{N}\big(\mu[i-1], \sigma_\mu\big)
\end{aligned}
\tag{11.24}
$$

$$
\sigma[n] \sim \text{Gamma}(10, 10) \qquad n = 1, \ldots, N
$$

Here, $g2Li$ maps the game index g to the index in $\vec{\mu}$ of the loser, and equals to the *Loser_Index* column in Data File 11.2. $g2Ln$ maps the game index g to the player index of the loser, and equals to the *Loser_PID* column in Data File 11.2. The same goes for $g2Wi$ and $g2Wn$. The major differences from Model Formula 9.3 are Eqs. (11.23) and (11.24). As we explained in Sect. 9.2.2, a game result is determined by the difference in two players' exhibited performances. Therefore, the likelihood does not change when the values of all players' capabilities shift simultaneously. Clearly, some constraints for the capabilities are needed in order to ensure the parameter identifiability. Here, we specify each player's capability in the first year to follow a Student-t distribution with mean 0 and scale 1.

Its implementation can be as follows:

model11-12.stan

```
data {
  int N;  // num of players
  int G;  // num of games
  int I;  // num of indices of (player, year)
  array[G] int<lower=1, upper=N> g2Ln; // loser playerID of each game
  array[G] int<lower=1, upper=N> g2Wn; // winner playerID of each game
  array[G] int<lower=1, upper=I> g2Li; // loser index in mu of each game
  array[G] int<lower=1, upper=I> g2Wi; // winner index in mu of each game
  array[N] int<lower=1, upper=I> n2Si;
  array[N] int<lower=1, upper=I> n2Ei;
 }

 parameters {
  vector[N] mu_ini;
  vector[I-N] mu_raw;
  real<lower=0> s_mu;
  vector<lower=0>[N] s_pf;
 }

 transformed parameters {
  vector[I] mu;
  {
```

```
23      int idx = 1;
24      for (n in 1:N) {
25       mu[n2Si[n]] = mu_ini[n];
26       for (t in (n2Si[n]+1):n2Ei[n]) {
27        mu[t] = mu[t-1] + s_mu*mu_raw[idx];
28        idx + = 1;
29       }
30      }
31     }
32    }
33
34    model {
35     for (g in 1:G) {
36      target + = normal_lccdf(0 | mu[g2Wi[g]] - mu[g2Li[g]],
37       hypot(s_pf[g2Ln[g]], s_pf[g2Wn[g]])
38      );
39     }
40     mu_ini ~ student_t(4, 0, 1);
41     mu_raw ~ normal(0, 1);
42     s_pf[1:N] ~ gamma(10, 10);
43    }
```

Line 4: The length of $\overrightarrow{\mu}$ is declared as `I`.

Lines 9–10: `n2Si` equals to the *Start_Index* column in Data File 11.1 and `n2Ei` equals to the *End_Index* column in Data File 11.1 These variables facilitate the implementation of Eq. (11.24).

Lines 14 and 40: These lines correspond to Eq. (11.23).

Lines 15, 26 to 29, and 41: These lines correspond to Eq. (11.24). The trend component of each player is reparametrized as in Sect. 11.2.7. Counting the index of `mu_raw` is troublesome, so we use the `idx` variable. To use such a temporary variable with a fixed value in `transformed parameters` block, we need to create a block with "`{}`" to declare the `idx` variable right after "`{`" (lines 22, 23, and 31).

If we run this model without specifying the initial values for the parameters, two players' `mu` will be too far away from each other, despite the fact that `s_pf` is close to 1. This may cause divergence of the likelihood, which depends on the game result (lines 36–38), and therefore the sampling might not be done properly. Thus, for these parameters, we can set initial values in R/Python code as follows (see Sect. 4.2.4):

A part of code **`run-model11-12.R`**

```
init_fun <- function(chain_id) {
  set.seed(chain_id)
  list(mu_ini = rnorm(N, 0, 0.01),
     mu_raw = rnorm(I-N, 0, 0.01),
     s_pf = rep(5.0, N))
}
```

A part of code **`run-model11-12.py`**

```
def init_fun():
   return dict(
```

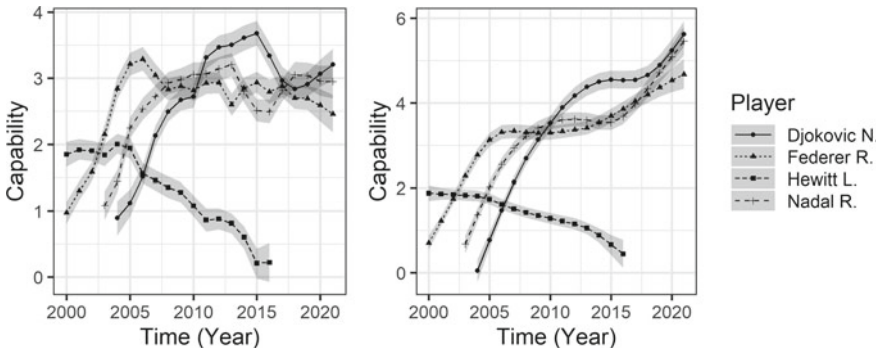


Fig. 11.17 The transitions of capabilities of the four players. The light gray bands are the 50% Bayesian confidence intervals and the black lines are the estimated medians. Left: when using first-order difference trend components. Right: when using second-order difference trend components

```
mu_ini=np.random.uniform(0, 0.01, N),
 mu_raw=np.random.uniform(0, 0.01, I-N),
s_pf=np.repeat(5.0, N))
```

By specifying `mu_ini` and `mu_raw` to be smaller values and `s_pf` to be a larger value, we can avoid the likelihood divergence.

We plotted the transitions of capabilities of the top players from 2000 to 2021, as shown in Fig. 11.17. We have also added the results for the case with second-order difference trend components. However, as explained in Sect. 1.5, we cannot say which trend component is "correct". Rather, it is more important to be aware of the fact that we have chosen an assumption. As shown in this example, the state space model can be easily combined with other models.

11.8 (Example 4) Decomposition of Sales Data

In this section, we use artificial sales data from a restaurant located inside a train station in a business district in Japan (Yamaguchi et al., 2004; Higuchi, 2022). In Japan, restaurants are usually open on both weekdays and weekends. The data contains the records of daily sales for two years (`data-eg4.csv`). Here we conduct analysis with the assumption that the records from August in both first and second years are missing. This sales data is plotted as a line plot in Fig. 11.18.

As background knowledge, in addition to the effects of trends and days of the week, there are other factors that might affect the sales, such as weathers and nearby events. In such a case where multiple factors are expected to be relevant, the thumbs of rules are to decompose these factors. Here, we decomposed $\mu[t]$ into trend component, calendar component, weather component, event component, and stationary AR component. We can express this as the following equation:

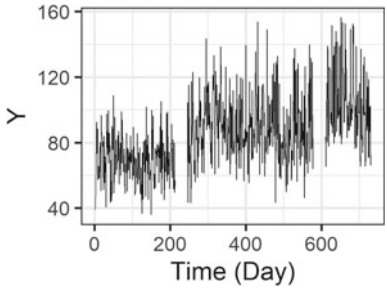


Fig. 11.18 Line plot of daily sales

$$\mu[t] = trend[t] + calendar[t] + weather[t] + event[t] + ar[t]$$

Let's look into each component individually.

For $trend[t]$, we use a second-order difference trend component.

$calendar[t]$ refers to the effects of days in the calendar, which is the sum of the seasonal the-days-of-the-week component (defined as $wday$ component), holiday effects, and before-holiday effects. In other words,

$$\begin{aligned} calendar[t] = {} & wday[t2wd[t]] + \\ & Ho[t]b_{Ho}(wday[7] - wday[t2wd[t]]) + \\ & BH[t]b_{BH}(wday[5] - wday[t2wd[t]]). \end{aligned} \tag{11.25}$$

The first, second, and third terms in the right-hand side of the equation correspond to the wday component, holiday effects, and before-holiday effects, respectively. $t2wd[t]$ maps the day index t to the-day-of-the-week index, and $t2wd[t]$ is an integer from 1 (Monday) to 7 (Sunday). $wday[1]$ to $wday[7]$ correspond to the seasonal wday component, and represent the effects of Monday, Tuesday, …, and Sunday. We ensure that they sum up to 0 (see Sect. 11.2.3). Now let us take a closer look at the holiday effects. $Ho[t]$ equals to 1 when t is a weekday holiday (that is, it is between Monday and Friday but also is a holiday), and equals to 0 otherwise. b_{Ho} is a parameter whose value is in the range of [0, 1], and represents the distance between the weekday holiday and the Sunday. For the sake of illustration, let us consider only the first two terms on the right-hand side of Eq. (11.25). As an example, $b_{Ho} = 0$ will make $calendar[t] = wday[t2wd[t]]$, which means that $calendar[t]$ is equivalent to the value of the original day of the week. In contrast, $b_{Ho} = 1$ will make $calendar[t] = wday[7]$, which means that $calendar[t]$ is equivalent to the value of Sunday. The same rule applies to the "before-holiday" effects. $BH[t]$ equals to 1 when t is a day between Monday and Thursday and it is also a before-holiday, and equals to 0 otherwise. b_{BH} is a parameter whose value is in the range of [0, 1], and represents the distance between the before-holiday and the Friday.

$weather[t]$ represents the effects of weather, and is assumed to be:

$$weather[t] = b_{weather} Weather_val[t]$$

where $Weather_val[t]$ is transformed from the weather of the day t, based on our experience. Specifically, it is a value in the range [0, 1], obtained by transforming (sunny, cloudy, rain, snow) into (0, 0.1, 0.5, 1, 1). The parameter $b_{weather}$ is its regression coefficient.

The event effect $event[t]$ is assumed to be:

$$event[t] = b_{event}[t]Event_val[t]$$

where $Event_val[t]$ is transformed from the total number of participants in the event, and is in the range [0, 1] as well. Specifically, we applied the following equations:

$$Event_val[t] = \begin{cases} 0, & \text{if } Event_n[t] < 0 \\ 1 - \exp[-0.00012(Event_n[t] - 8000)], & \text{if } Event_n[t] \geq 0 \end{cases}$$

where $Event_n[t]$ is the actual number of participants for the event. $Event_val[t]$ equals to 0 when $Event_n[t]$ is a small number, and when $Event_n[t]$ is larger than a certain value (we used 8000 here), it increases rapidly (Fig. 11.19). The parameter $b_{event}[t]$ is its regression coefficient. In contrast to the coefficient for the weather effect, here we consider that this coefficient changes over time t:

$$b_{event}[t] \sim \mathcal{N}\left(b_{event}[t-1], \sigma_{b_{event}}\right) \quad t = 2, \ldots, T$$

$ar[t]$ is the stationary AR component. Here we used a second-order AR model, and added constraints so that it has a periodic autocorrelation. In fact, we analyzed the data initially without using a stationary AR component, and obtained the residual plot as shown in Fig. 11.20. Based on this result, we added $ar[t]$ to the system model, because we considered that a part of this residual can be described well using a stationary AR component.

We define the combination of all equations given in this section, the combination of the trend component Eq. (11.6), stationary AR component Eq. (11.8), and observation model Eq. (11.2) as Model Formula 11.13. Its implementation can be as follows:

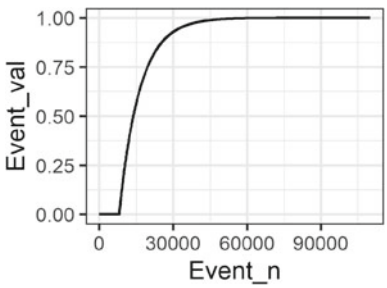


Fig. 11.19 A curve that transforms the number of event participants into the range [0, 1]

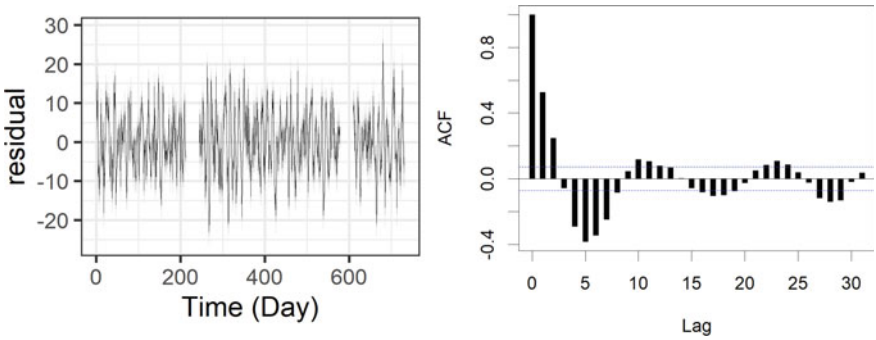


Fig. 11.20 The residual plot and its correlogram from estimated results using a model without including stationary AR component. On the left side, the light gray band is the 95% Bayesian confidence intervals, the dark gray band is the 50% Bayesian confidence intervals, and the thin black line is the medians

model11-13.stan

```
data {
  int T;
  array[T] int<lower=1, upper=7> t2wd;
  vector[T] Ho;
  vector[T] BH;
  vector[T] Weather_val;
  vector[T] Event_val;
  int T_obs;
  array[T_obs] int<lower=1, upper=T> t_obs2t;
  vector[T_obs] Y;
 }

 parameters {
  vector[T-2] trend_raw;
  vector<lower=0, upper=100>[2] trend_ini;
  vector[6] wday_raw;
  real<lower=0, upper=1> b_Ho;
  real<lower=0, upper=1> b_BH;
  real b_weather;
  vector[T-1] b_event_raw;
  real b_event_ini;
  real<lower=-2, upper=2> b_ar1;
  real<lower=-1, upper=-square(b_ar1)/4> b_ar2;
  vector[2] ar_ini;
  vector[T-2] ar_raw;
  real<lower=0> s_trend;
  real<lower=0> s_b_event;
  real<lower=0> s_ar;
  real<lower=0> s_y;
 }

 transformed parameters {
  vector[T] trend;
```

```
34     vector[7] wday;
35     vector[T] calendar;
36     vector[T] weather;
37     vector[T] b_event;
38     vector[T] event;
39     vector[T] ar;
40     vector[T] mu;
41     trend[1:2] = trend_ini[1:2];
42     for (t in 3:T) {
43      trend[t] = 2*trend[t-1] - trend[t-2] + s_trend*trend_raw[t-2];
44     }
45     wday[1:6] = wday_raw[1:6];
46     wday[7] = -sum(wday_raw);
47     calendar[1:T] = wday[t2wd] +
48      Ho[1:T] .* (b_Ho*(wday[7]-wday[t2wd])) +
49      BH[1:T] .* (b_BH*(wday[5]-wday[t2wd]));
50     weather[1:T] = b_weather * Weather_val[1:T];
51     b_event[1] = b_event_ini;
52     for (t in 2:T) {
53      b_event[t] = b_event[t-1] + s_b_event*b_event_raw[t-1];
54     }
55     event[1:T] = b_event .* Event_val[1:T];
56     ar[1:2] = ar_ini[1:2];
57     for (t in 3:T) {
58      ar[t] = b_ar1*ar[t-1] + b_ar2*ar[t-2] + s_ar*ar_raw[t-2];
59     }
60     mu = trend + calendar + weather + event + ar;
61   }
62
63   model {
64     s_trend ~ normal(0, 1);
65     s_b_event ~ normal(0, 1);
66     b_event_raw ~ normal(0, 1);
67     trend_raw ~ normal(0, 1);
68     ar_raw ~ normal(0, 1);
69     ar_ini ~ normal(0, 10);
70     s_ar ~ normal(0, 5);
71     s_y ~ normal(0, 5);
72     Y[1:T_obs] ~ normal(mu[t_obs2t], s_y);
73   }
```

It contains many decomposed components and is not as complicated as it might look like. Lines 47–49 correspond to Eq. (11.25) and constraints are added in lines 17–18. In lines 22–23, constraints are added in order to force stationary AR component to be periodic (see Eq. (11.19) in Sect. 11.2.6). In lines 41–44, 51–54, and 56–59, we applied the reparameterization we introduced in Sect. 11.2.7.

A part of R/Python code to run **model11-13.stan** can be as follows:
A part of code **run-model11-13.R**

```
1   d <- read.csv('input/data-eg4.csv')
2   Weather_val <- c(0, 0.1, 0.5, 1, 1)[d$Weather]
3   Event_val <- ifelse(
4    d$Event_n < 8000 0, 1 - exp(-0.00012*(d$Event_n - 8000)))
5   d[d$Month == 8, ]$Y <- NA
6   t_obs2t <- which(!is.na(d$Y))
7   Y <- d$Y[t_obs2t]
```

A part of code **run-model11-13.py**

```
1   d = pandas.read_csv('input/data-eg4.csv')
2   Weather_val = np.array([0, 0.1, 0.5, 1, 1])[d.Weather - 1]
3   Event_val = np.where(d.Event_n < 8000, \
4      0, 1 - np.exp(-0.00012*(d.Event_n - 8000)))
5   d.loc[d.Month == 8, ['Y']] = np.nan
6   t_obs2t = d.index[d.Y.notnull()] + 1
7   Y = d.loc[t_obs2t - 1].Y
```

Line 2: Weather records are transformed into the range [0, 1].

Lines 3–4: The numbers of event participants are transformed into the range [0, 1].

Lines 5–7: Data in August are treated as missing data.

After fitting this model using `sample` function with argument `max_treedepth = 10` as default, it outputs a warning message, asking to increase `max_treedepth`. Therefore, following the suggestion from "Brief Guide to Stan's Warnings",[8] we fit the model using `max_treedepth = 15`.

Estimation results for each component and residuals are shown in Fig. 11.21 and Fig. 11.22 respectively. From the figures, we can see that the trend component changes smoothly, and residual correlogram is similar to that of observation noise. Therefore, we conclude that the model has a good fit and the derived result is informative.

We will discuss the estimation results of some of the parameters. In Fig. 11.23 we show the change of $b_{event}[t]$ and $wday[t]$ after the estimation. From Fig. 11.23 (left), we can see that the coefficients for event effect increases after a certain time point. From Fig. 11.23 (right), we see that the values during weekdays are higher than those of weekends. The estimated median and 95% Bayesian confidence interval of $b_{weather}$ is 11.9 [10.3, 13.5], which suggests that the bad weather causes more sales. The estimated results of weather effects and the effects of a day of the weeks agree with what we would expect, based on the fact that this restaurant is located in a business district and is also inside a train station building.

As in this sales data example, when there are several factors involved in the observation, decomposition makes the interpretation easier, and also tells us more information such as which component has a larger impact on future values predictions.

[8] http://mc-stan.org/misc/warnings.html.

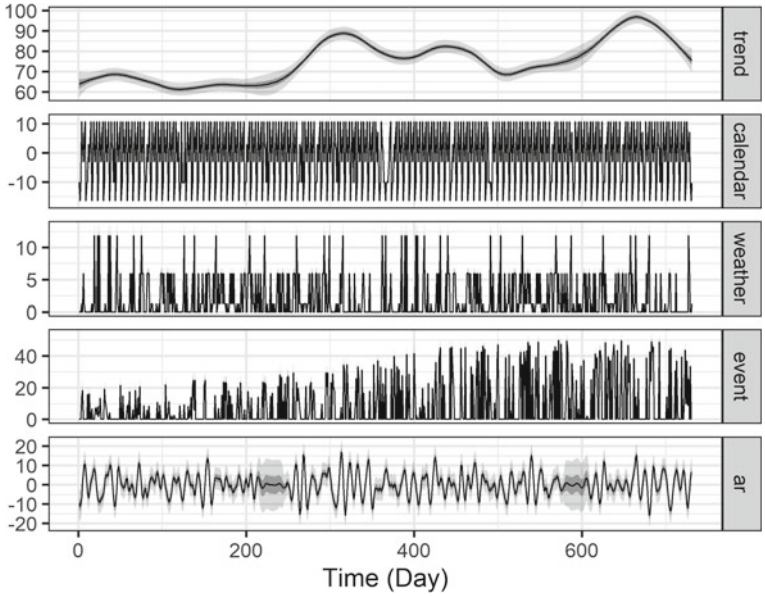


Fig. 11.21 Estimated values of each component. The light gray bands are the 95% Bayesian confidence intervals, the dark gray bands are the 50% Bayesian confidence intervals and the thin black lines are the medians

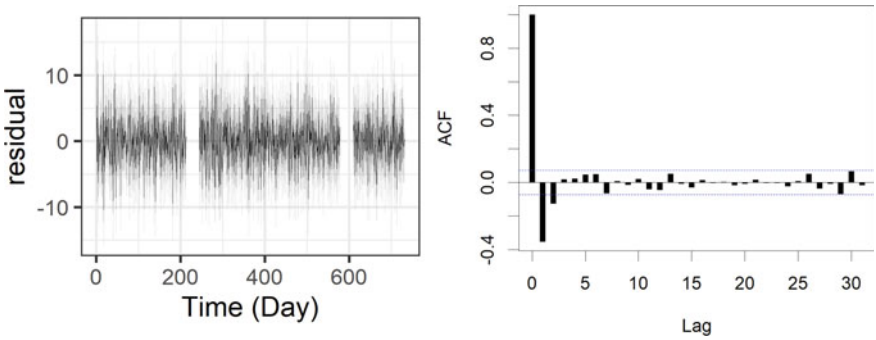


Fig. 11.22 Left: the plot of residuals. Legend is the same as the one in Fig. 11.21. Right: its correlogram

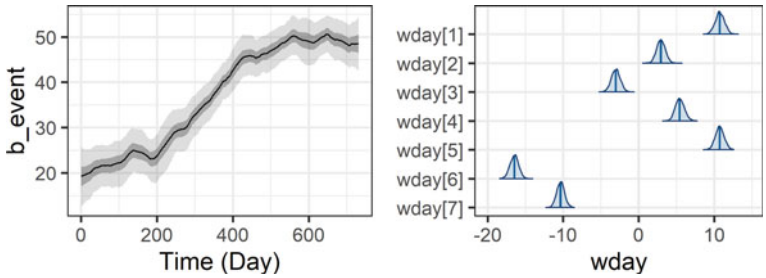


Fig. 11.23 Left: estimated $b_{event}[t]$. Legend is the same as the one in Fig. 11.21 Right: estimated result for $wday[1]$ to $wday[7]$

Technical Point: Prediction, filtering, and smoothing

In order to increase the efficiency of computation and to facilitate understanding, the method used for traditional time series analysis usually divides the problem into the following three subproblems, and then discusses each of them.

Prediction: $p(\mu[\tau]|Y[1:t])$ where $\tau > t$.
Filtering: $p(\mu[\tau]|Y[1:t])$ where $\tau = t$.
Smoothing: $p(\mu[\tau]|Y[1:t])$ where $\tau < t$.

Smoothing usually has more accurate estimation compared to prediction or filtering, because smoothing uses information not only from the time points before τ, but also from the time points after τ. The drawback is that the computation takes more time. In fact, smoothing corresponds to what we do with MCMC software such as Stan. The distribution obtained by the smoothing is called a smoothed distribution. In case of using Stan, the joint posterior distribution that are marginalized on all other parameters except τ, which we focus on, is the smoothed distribution.

In the context of time series analysis, *prediction* means computing distributions of parameters and data at future time points, rather than obtaining the posterior predictive distribution of data. The distribution at the time point $t+1$ is called *one-step-ahead prediction.* When using Stan for such prediction, we can compute the draws at future time points in `generated quantities` block, as we showed in Sect. 11.1.5.

Suppose that we have the one-step-ahead predictive distribution $p(\mu[t]|Y[1:t-1])$, which is calculated using data obtained before t. When the data at time t (i.e., $Y[t]$) is obtained, *filtering* represents the process of updating the distribution of the state at the current time point (at time point t) using that one-step-ahead predictive distribution. The obtained distribution is called *filtered distribution.* Filtering is a helpful step because re-estimating all the parameters with all the data in Stan may take too much time. Specifically,

we express the distribution of $\mu[t]$ as follows[9]:

$$p(\mu[t]|Y[1:t]) \propto p(Y[t]|\mu[t])p(\mu[t]|Y[1:t-1])$$

Its right-hand side can be explained as follows: if we get the draws from one-step-ahead predictive distribution at the time point $t-1$, $p(\mu[t]|Y[1:t-1])$, and weight each of them with $w = p(Y[t]|\mu[t])$, we can consider them as the draws from the filtered distribution. The filtered distribution can be calculated quickly, because both the one-step-ahead predictive distribution and the weight w can be calculated by using $Y[t]$ and the result obtained from the data before t.

For instance, in Sect. 11.1, suppose we have the one-step-ahead predictive distribution obtained using data up to Time $= 20$. And suppose the data at Time $= 21$, $Y[21] = 11.1$, is now available. The filtering that estimates the state of Time $= 21$ using the one-step-ahead predictive distribution can be implemented as follows:

R

```
(...Estimation by Stan...)
mu_all_ms <- fit$draws('mu_all', format='matrix')
s_y_ms <- fit$draws('s_y', format='df')$s_y

t_now <- 21
dens_pred <- density(mu_all_ms[,t_now])
w <- dnorm(x=85.2, mean=mu_all_ms[,t_now], sd=s_y_ms)
dens_flt <- density(mu_all_ms[,t_now], weight=w/sum(w))
post_mean <- sum(mu_all_ms[,t_now] * w/sum(w))
```

Python

```
(...Estimation by Stan...)
from scipy stats import norm, gaussian_kde
mu_all_ms = fit.draws_pd(vars=['mu_all'])
s_y_ms = fit.draws_pd(vars=['s_y']).s_y.values

t_now = 21 - 1
dens_pred = gaussian_kde(mu_all_ms.iloc[:,t_now])
w = norm.pdf(x=85.2, loc=mu_all_ms.iloc[:,t_now], scale=s_y_ms)
dens_flt = gaussian_kde(mu_all_ms.iloc[:,t_now], weights=w/sum(w))
post_mean = np.sum(mu_all_ms.iloc[:,t_now] * w/sum(w))
```

[9] To prove it, we use the definition of condition probability from the first line to the second line:

$$\begin{aligned} p(\mu[t]|Y[1:t]) &= p(\mu[t]|Y[t], Y[1:t-1]) \\ &= \frac{p(Y[t], \mu[t]|Y[1:t-1])}{p(Y[t]|Y[1:t-1])} \propto p(Y[t], \mu[t]|Y[1:t-1]) \\ &= p(Y[t]|\mu[t])p(\mu[t]|Y[1:t-1]) \end{aligned}$$

Table 11.2 Estimation methods and their properties

	MCMC	SMC	Kalman filter
Suitable for	Smoothing	Filtering	Filtering
Unsuitable for	Filtering	Smoothing	None
Requirements for models	None	None	Only applicable for linear model with noise that follows a normal distribution
Computation speed	Slow	Slow	Fast
Estimation stability	Unstable	Unstable	Stable
Theoretical condition for convergence	Number of iterations is ∞	Number of particles is ∞	NA
Example of R packages	`cmdstanr`	`RcppSMC`	`dlm`, `KFAS`

`dens_pred` in line 5 is the density of the one-step-ahead predictive distribution computed from the draws. Line 6 reflects the weight for each draw as explained earlier. `dens_flt` in line 7 is the density function of filtered distribution, computed using the weights. Line 8 gives an example of computing posterior mean of the filtered distribution. For time series problem, sometimes it is more important to have faster computations for prediction and filtering, whereas fast computing for smoothing is less important. In such a case, instead of using MCMC with Stan or other software, it is common to use sequential Monte Carlo (SMC) and Kalman filters. We will omit the detailed explanations of these methods because they are over the scope of this book. Interested readers can refer to (Naesseth et al., 2019) for more information. Lastly, we summarize these methods and other information including their suitable application fields, as shown in Table 11.2.

11.9 Supplementary Materials and Exercises

For time series data, there are other preprocessing steps, such as taking the differences and log-transformation. When we use state space model, however, it is not necessary to take the difference of data, and in fact, it is not a good idea as it will make interpretation harder. Log-transformation is only recommended when it gives more straightforward interpretations, and should be avoided if it is the other way around (see Sect. 7.1).

In this chapter, we briefly introduced the state space model for time series analysis. Readers can refer to (Harvey, 1990; West and Harrison 2006) and the chapter *Time-Series Models* in the Stan reference, for more information on other methods such as ARIMA, stochastic volatility model and Hidden Markov Model.

11.9.1 Exercises

(1) Simulate Model Formula 11.2. Especially, examine what kind of time series data would be generated under the conditions (σ_μ, σ_y) is (large, small), and (small, large), respectively.
(2) Extend **`model11-5.stan`** in Sect. 11.2.3, and compute the prediction intervals for from $y[t+1]$ to $y[t+8]$.
(3) Suppose we buy 10 eggs of a certain insect every month, and record the number of them that are successfully born ("`data-ex3.csv`" in our GitHub repository). For this data, estimate the probability of successful birth in each year using a first-order difference trend component as the system model and a Poisson distribution as the observation model (see Sects. 11.2.1 and 11.3.2). Then estimate again by using the second-order difference trend component instead of the first-order difference trend component.
(4) For "`data-ex4.csv`" in our GitHub repository, estimate the trend of this time series using the system model consisting of the sum of the first-order difference trend component and pulse component (use Eq. (11.16)).

References

Harvey, A. C. (1990). *Forecasting, structural time series models and the Kalman filter*. Cambridge University Press.
Higuchi, T. (2022). *The foundation of statistical modeling for practical prediction* (2nd ed.). Kodansha Ltd. (in Japanese).
Naesseth, C. A., Lindsten, F., & Schön, T. B. (2019). Elements of Sequential Monte Carlo. arXiv preprint arXiv:1903.04797.
Nomura, S. (2016). *Kalman filter—time series Prediction and state space model using R—*. Kyoritsu Shuppan Co. (in Japanese).
West, M., & Harrison, J. (2006). *Bayesian forecasting and dynamic models*. Springer Science & Business Media.
Yamaguchi, R., Tsuchiya, E., & Higuchi, T. (2004). State space modeling approach to decompose daily sales of a restaurant into time-dependent multi-factors (in Japanese). *Transactions of the Operations Research Society of Japan, 49*, 52–60.

Chapter 12
Spatial Data Analysis Using Gaussian Markov Random Fields and Gaussian Processes

In this chapter, we will learn about a Gaussian Markov random filed (GMRF), which can be considered as an extension of the state space model, and how to use it to analyze spatial data. A GMRF is relatively straightforward to understand. It has a wide range of application and can be applied to one-dimensional data, two-dimensional grid data type, and geospatial map data. Later, we will see how a Gaussian process (GP) can be considered as a generalization of a GMRF. A GP can represent smooth functions, and usually gives high prediction performance. The downside is that when the data size is large, it could be computationally expensive and estimation will take a long time.

The contents of this chapter are outlined below.

- In Sect. 12.1, we will explain how a one-dimensional GMRF is equivalent to the state space model discussed in the last chapter.
- In Sects. 12.2 and 12.3, we will see some applications of one-dimensional GMRFs.
- In Sect. 12.4, we will explain a two-dimensional GMRF.
- In Sects. 12.5 and 12.6, we will see examples where we apply two-dimensional GMRFs.
- In Sect. 12.7, we will introduce a GP, with focus on the details of how to do the implementation.
- In Sects. 12.8 and 12.9, we will give examples of GP applications. The examples in Sects. 12.2 and 12.6 will be used.
- In Sect. 12.10, we will address inducing variable method, which is one of the methods that are used to speed up the estimation of a GP.

© Springer Nature Singapore Pte Ltd. 2022

K. Matsuura, *Bayesian Statistical Modeling with Stan, R, and Python*,
https://doi.org/10.1007/978-981-19-4755-1_12

12.1 Equivalence Between State Space Model and One-Dimensional GMRF

In the previous chapter, we introduced the general idea of the state space model and its applications. Here, in order to get a better sense of the state space model, we will derive the posterior log probability of Model Formula 11.2. Later, we will view from the aspect of derived posterior log probability and explain how the temporal structure (the state space model) and one-dimensional spatial structure are equivalent. Let us recall what we had for Model Formula 11.2:

Model Formula 11.2

$$\mu[t] \sim \mathcal{N}\big(\mu[t-1], \sigma_\mu\big) \quad t = 2, \ldots, T \tag{12.1}$$

$$Y[t] \sim \mathcal{N}\big(\mu[t], \sigma_y\big) \quad t = 1, \ldots, T \tag{12.2}$$

12.1.1 Posterior Probability of the State Space Model

First, by writing down and transforming the posterior probability that corresponds to the system model (12.1), we can get:

$$\begin{aligned}
\text{posetrior probability} &= p\big(\sigma_\mu\big) \cdot p(\mu[1]) \cdot \mathcal{N}\big(\mu[2]|\mu[1], \sigma_\mu\big) \cdot \cdots \cdot \mathcal{N}\big(\mu[T]|\mu[T-1], \sigma_\mu\big) \\
&= (\text{const.}) \cdot \frac{1}{\sigma_\mu} \exp\left\{-\frac{(\mu[2]-\mu[1])^2}{2\sigma_\mu^2}\right\} \cdot \cdots \cdot \frac{1}{\sigma_\mu} \exp\left\{-\frac{(\mu[T]-\mu[T-1])^2}{2\sigma_\mu^2}\right\} \\
&= (\text{const.}) \cdot \frac{1}{\sigma_\mu^{T-1}} \exp\left\{-\frac{1}{2\sigma_\mu^2} \sum_{t=2}^{T} (\mu[t]-\mu[t-1])^2\right\}
\end{aligned}$$

On the above transformation, we used constants as the noninformative priors for $p\big(\sigma_\mu\big)$ and $p(\mu[1])$. Taking logarithm of this result, we get:

$$\text{log posterior probability} = -(T-1)\log\sigma_\mu - \frac{1}{2\sigma_\mu^2} \sum_{t=2}^{T} (\mu[t]-\mu[t-1])^2 \tag{12.3}$$

The constant term is ignored in the above equation. This is because in Stan, only the partial derivatives of log posterior probability are used and thus a constant term that give zero after taking derivatives can be ignored (refer Sect. 3.6).

Next, we can write the posterior probability derived from the observation model (12.2) as below:

$$\begin{aligned}\text{posterior probability} &= p(\sigma_y) \cdot \mathcal{N}(Y[1]|\mu[1], \sigma_y) \cdot \cdots \cdot \mathcal{N}(Y[T]|\mu[T], \sigma_y) \\ &= (\text{const.}) \cdot \frac{1}{\sigma_y} \exp\left\{-\frac{(Y[1]-\mu[1])^2}{2\sigma_y^2}\right\} \cdot \cdots \cdot \frac{1}{\sigma_y} \exp\left\{-\frac{(Y[T]-\mu[T])^2}{2\sigma_y^2}\right\} \\ &= (\text{const.}) \cdot \frac{1}{\sigma_y^T} \exp\left\{-\frac{1}{2\sigma_y^2} \sum_{t=1}^{T} (Y[t]-\mu[t])^2\right\}\end{aligned}$$

Again, a constant is used as the prior for $p(\sigma_y)$. By log transformation, we can write it as:

$$\text{log posterior probability} = -T \log \sigma_y - \frac{1}{2\sigma_y^2} \sum_{t=1}^{T} (Y[t]-\mu[t])^2 \tag{12.4}$$

In the above equation, we ignored the constant term as we did for the systems model.

An intuitive explanation of how the value of $\mu[t]$ is determined in the state space model is as follows. To obtain the value of $\mu[t]$ that maximizes the log probability of the systems model, we want $\mu[t]$ and $\mu[t-1]$ to be similar in Eq. (12.3). The extreme case is when $\mu[1] = \mu[2] = \ldots = \mu[T]$, where $\mu[t]$ is a straight horizontal line. On the other hand, to obtain the value of $\mu[t]$ that maximizes the log probability of the observation model, we want the values of $Y[t]$ and $\mu[t]$ to be similar in Eq. (12.4). When $Y[t] = \mu[t] (t = 1, \ldots, T)$, this gives the line of $\mu[t]$ that overlays with the zigzag line of observed value. Therefore, the value of $\mu[t]$ that maximizes the log posterior probability of the overall state space model is determined by finding a balance between these two somehow conflicting conditions. We can also use weakly informative priors to further control this balance.

12.1.2 The Equivalence Between the Temporal and Spatial Structures

Let's look at Eq. (12.3) again. We will notice that for the term $(\mu[t] - \mu[t-1])^2$, swapping $\mu[t-1]$ and $\mu[t]$ will give the same log posterior probability. This indicates that the direction of the time flow (which is, from $t-1$ to t) itself does not have specific meaning, and it only represents that $\mu[t-1]$ and $\mu[t]$ are connected. Especially for Eq. (12.3), we can consider this connection as a spring (Fig. 12.1).

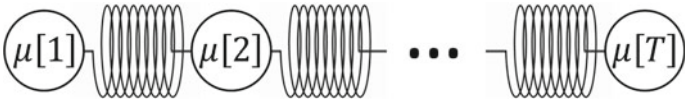


Fig. 12.1 The spring system

Let us convince ourselves with this idea. If we denote the spring extension distance as x, then the potential energy of the spring is ax^2 (where a is a coefficient that determines the stiffness of the spring).[1] Any state is more stable when the energy of the system is lower. To maintain a lower energy ax^2, the spring will try to strain when it is extended (i.e., it wants x to be closer to 0). This effect of reducing the stored energy ax^2 is equivalent to minimizing the difference between $\mu[t-1]$ and $\mu[t]$, thereby to minimize $1/\left(2\sigma_\mu^2\right)(\mu[t]-\mu[t-1])^2$. This will give the same equation form as the one that tries to increase the log posterior probability. In addition, in Eq. (12.3), the parameter σ_μ determines the stiffness of the springs. The larger the σ_μ, the more flexible the springs are. The first term in Eq. (12.3) can be seen as adding a penalty term when the spring is too flexible. The difference between $\mu[t-1]$ and $\mu[t]$ corresponds to the spring extension. Those are the main reasons why we considered this connection as a spring. In the spring system, we assume that the two connected elements should be close to each other by pulling over each other, and we assume that these elements are connected by an undirected edge.

The state space model focuses on the nodes (elements), and there is a directionality from $t-1$ to t. If we see the state space model as the system of springs, however, it focuses on the edges (connections) and there should be no directionality. When the log posterior probability is determined by the connections between the neighbors, the model is called a Markov Random Field (MRF).[2] In particular, when the log probability of the connection is derived from a normal distribution, as in Eq. (12.3), we call it a Gaussian Markov Random Field (GMRF). The way we represent the log posterior probability of a state space model is equivalent to the way we do for a GMRF, and thus we can use the same equation for the directed time structure and the undirected space structure.

In this book, when we emphasize that there's no directionality between neighboring nodes, we will use the joint distribution of $\vec{\mu} = (\mu[1], \ldots, \mu[T])$ as follows, rather than using Model Formula 12.1:

Model Formula 12.1

$$\vec{\mu} \sim \frac{1}{C\sigma_\mu^{T-1}} \exp\left[-\frac{1}{2\sigma_\mu^2}\sum_{t=2}^{T}(\mu[t]-\mu[t-1])^2\right] \tag{12.5}$$

$$Y[t] \sim \mathcal{N}\left(\mu[t], \sigma_y\right) \quad t = 1, \ldots, T$$

[1] For those readers who are not familiar with the energy of springs, we suggest looking up for "spring energy" on the internet. Here we ignored the constant factor for the energy.

[2] Conditional Autoregressive model (CAR) refers to the same type of model as well.

The Eq. (12.5) represents that $\vec{\mu}$ follows the joint distribution on the right-hand side. The constant C is to normalize the right-hand side of the equation. Because it would not affect the derivative of the log posterior probability, we will ignore it in the equations in the next steps.ss.

In the following Sects. 12.2 and 12.3, we will see examples of representing one-dimensional spatial data using GMRFs.

12.2 (Example 1) Data on One-Dimensional Location

Here, we will work with 50 data points, where each data point represents a one-dimensional location with regular intervals between each other, and the values of the data points represent the observed numbers of plants at each location (Data File 12.1, a row represents a location).[3]

Data File 12.1 Structure of `data-1D.csv`

```
  1     Y
  2     0
  3     3
  4     2
        ...
 51     12
```

If we plot this data on a scatter plot, with the location index on the x-axis, and the column Y on the y-axis, we can obtain Fig. 12.2. From here, we represent the number of locations as I and the number of observed plants at location i as $Y[i]$.

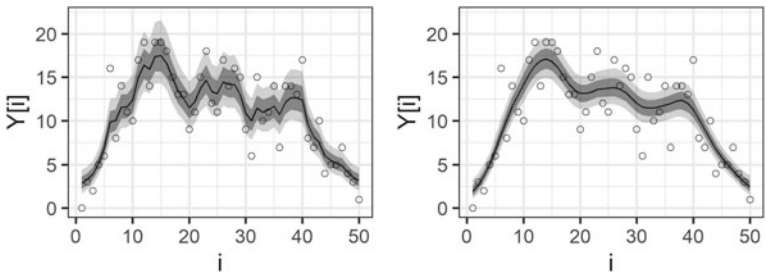


Fig. 12.2 Estimation results by taking the spatial structure into consideration. The x-axis represents the observation locations ($i = 1, \ldots, 50$) and the y-axis represents the plant counts. The light gray bands are the 80% Bayesian confidence intervals, the dark gray bands are the 50% Bayesian confidence intervals and the thin black lines are the medians for the mean plant counts ($\exp(f[i])$). Left: the result using the first-order difference trend component. Right: the result using the second-order difference trend component

[3] Data resource (Japanese): https://kuboweb.github.io/-kubo/ce/IwanamiBook.html.

To analyze this data, we assume that the observation points adjacent to each other should have similar means of Y. There are multiple ways to statistically express such an assumption. Here we will express the assumption using a GMRF. That is, we consider $f[i]$ as the latent value of the log-transformed mean of the plant counts at the location i, and further consider that $f[1], f[2], \ldots, f[I]$ are connected with springs. We consider that $Y[i]$ is generated from a Poisson distribution with the mean at location i (which is $\exp(f[i])$).

This statistical model can be written as:

Model Formula 12.2

$$\vec{f} \sim \frac{1}{\sigma_f^{I-1}} \exp\left[-\frac{1}{2\sigma_f^2} \sum_{i=2}^{I} (f[i] - f[i-1])^2 \right] \tag{12.6}$$

$$Y[i] \sim \text{Poisson}(\exp(f[i])) \quad i = 1, \ldots, I \tag{12.7}$$

where σ_f is the parameter that determines the stiffness of the springs. From the data, the parameters $f[i]$ and σ_f will be estimated. The format of this model is equivalent to Model Formula 12.1, except that we used a Poisson distribution for the observation model. Its implementation can be as follows:

model12-2.stan

```
1   data {
2    int I;
3    array[I] int Y;
4   }
5
6   parameters {
7     vector[I] f;
8     real<lower=0> s_f;
9   }
10
11   model {
12    f[2:I] ~ normal(f[1:(I-1)], s_f);
13    Y[1:I] ~ poisson_log(f[1:I]);
14   }
15
16   generated quantities {
17     vector[I] y_mean = exp(f[1:I]);
18   }
```

Line 12: Corresponds to Eq. (12.6)

Line 13: Corresponds to Eq. (12.7). Note that we can use vectorization even though `Y` is not a vector type (for more detail, refer `poisson_log` function in Stan reference).

The estimated result is shown in Fig. 12.2 (left).

From Fig. 12.2 (left), we realize that the estimated mean value of the plant counts varies based on the location. However, we may expect that the mean of the plant counts should change more smoothly. Let us consider how we can modify the model to account for this. Recall what we talked about the state space model in Sect. 11.2.1: when we wanted to express the assumption that the change of the states should be smooth, we used a second-order difference trend component that follows the formula:

$$trend[t] \sim \text{Normal}(2\,trend[t-1] - trend[t-2], \sigma_{trend}) \tag{12.8}$$

We can write down the log posterior probability for Eq. (12.8). This will help us realize that instead of using Eq. (12.6), in order tos assume a smoother spatial structure, we can write the following equation including the observation model:

Model Formula 12.3

$$\vec{f} \sim \frac{1}{\sigma_f^{I-2}} \exp\left[-\frac{1}{2\sigma_f^2}\sum_{i=3}^{I}(f[i] - 2f[i-1] + f[i-2])^2\right] \tag{12.9}$$

$$Y[i] \sim \text{Poisson}(\exp(f[i])) \quad i = 1, \ldots, I$$

For Eq. (12.9), the log posterior probability will be maximized when the three values $f[i-2]$, $f[i-1]$ and $f[i]$ are the same. As an example, this is equivalent to the case where there is a flexible bar that connects these three elements $f[i-2]$, $f[i-1]$ and $f[i]$, and when the bar is strictly stiff and straight, it gives the minimum energy.

We can implement this model by simply changing the line 12 of the code **`model12-1.stan`** as follows:

```
f[3:I] ~ normal(2*f[2:(I-1)] - f[1:(I-2)], s_f);
```

The estimated result is shown in Fig. 12.2 (right) – we obtained a result that is similar to the one where we assumed a smooth change on the temporal structure.

12.3 (Example 2) Fix the "Age Heaping"

Here, let us see how we can use a GMRF to denoise the census data and estimate the population age structures. We use the 1990 census of population data of Indonesia for this example (Data File 12.2). This data is retrieved from the United Nation

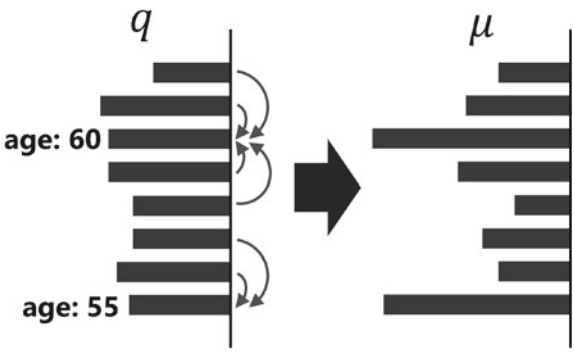


Fig. 12.3 The Indonesia population pyramid in 1990. Left half and right half show the age structure of male and female populations. The y-axis represents the age and the x-axis represents the proportion of population at this age to the total population

database.[4] We have already extracted a part of the data and preprocessed it to make it easier to work with on R and Python. Each column represents:

Sex: The gender category (M: male, F: female).

Age: The age (age of 75 includes those who answered older than 75).

Y: The number of respondents whose gender and age match in each category.

Data File 12.2 Structure of `data-ageheap.csv`

```
  1    Sex,Age,Y
  2    M,0,1923050
  3    M,1,2047035
  4    M,2,2163294
       ...
153    F,75,1104720
```

From this data, we can plot a population pyramid as shown in Fig. 12.3. We can see that from 20 years-old age, there are peaks in every five age years. This phenomenon is called *age heaping*. The reason of observing this odd phenomenon is that many people do not know their exact ages. When they are asked, they tend to answer with rounded numbers that they think is close to their actual age.

By building the statistical model, we want to fix this age heaping and estimate the latent age structure proportion $\vec{q}$. Here, we will divide the data into two categories based on the gender. First, let's consider only the male case. We assume that when the age is similar, the age structure proportion should be similar as well, and we will use a one-dimensional GMRF (second-order difference) for this purpose. Subsequently, we incorporate the mechanism of how the respondents answered with rounded numbers. We also assume that two years-difference would cause people to round their age and those population would "flow" from one age category to another (see Fig. 12.4). The flows make the structure proportion change from $\vec{q}$ to $\vec{\mu}$. Further, we assume that the data is generated from a multinomial distribution with the total population and $\vec{\mu}$ as the parameters. Although here we consider a simple flow, we can also extend it to a more complicated flow where the flow decreases exponentially when the actual ages are farther away from the rounded ages.

[4] http://data.un.org/Default.aspx.

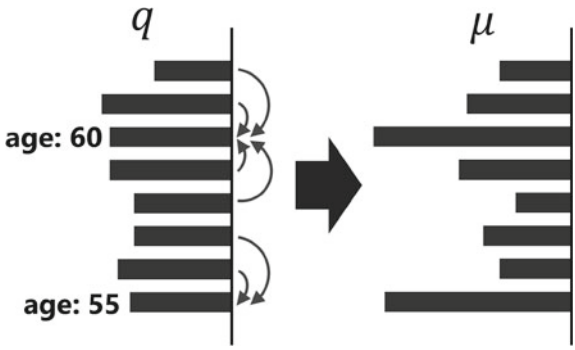


Fig. 12.4 An example of the potential mechanism of how the respondents answered their ages with rounded numbers

By expressing these mechanisms with a model formula, we can obtain:

Model Formula 12.4

$$\overrightarrow{q[1:(A-1)]} \sim \frac{1}{\sigma_q^{A-3}} \exp\left[-\frac{1}{2\sigma_q^2}\sum_{a=3}^{A-1}(q[a]-2q[a-1]+q[a-2])^2\right] \quad (12.10)$$

$\vec{\mu}$ is considered to be generated from the following steps:

1. Initialize $\vec{\mu}$ as $\vec{q}$.
2. for j in $1, \ldots, J$:

$$\begin{aligned} \mu[From[j]] &= \mu[From[j]] &&-r[j]q[From[j]] \\ \mu[To[j]] &= \mu[To[j]] &&+r[j]q[From[j]] \end{aligned} \quad (12.11)$$

$$\vec{Y} \sim \text{Multinomial}(\text{Total Population}, \vec{\mu}) \quad (12.12)$$

where A represents the discrete counts of ages (from 0 to 74 and 75+, in total 76 numbers). $q[1]$ is the original age structure of 0 years-old population, and $q[A]$ is the age structure of the population above 75 years-old. The reason that the left-hand side of Eq. (12.10) is not expressed as $\overrightarrow{q[1:A]}$ is that $q[A]$ also includes those above 75 years, and it is not necessarily similar to the structure of the 74 years-old population. The variable J is the total number of respondents with potential age "flow", and j is the index for each. One flow is composed from the age index $From[j]$, which represents where the flow comes from, and the age index $To[j]$, which represents where the flow goes to (and this is the age categories where the ages are the rounded numbers). In the j-th flow, Eq. (12.11) represents that the age proportion flows from $From[j]$ to $To[j]$, with amount $r[j]q[From[j]]$. That is, the parameter $r[j]$, which takes a value in the range of [0, 1], represents how much of the population in $q[From[j]]$ answered the rounded numbers that are different from their actual age. Lastly, Eq. (12.12) will associate $\vec{\mu}$ and $\vec{Y}$ by a multinomial distribution.

Its implementation is as follows:

model12-4.stan

```
1  data {
2    int A;
3    array[A] int Y;
4    int J;
5    array[J] int From;
6    array[J] int To;
7  }
8
9  parameters {
10   simplex[A] q;
11   real<lower=0> s_q;
12   vector<lower=0, upper=1>[J] r;
13  }
14
15  transformed parameters {
16   vector[A] mu = q[1:A];
17   for (j in 1:J) {
18    mu[To[j]] = mu[To[j]] + r[j]*q[From[j]];
19    mu[From[j]] = mu[From[j]] - r[j]*q[From[j]];
20   }
21  }
22
23  model {
24   q[3:(A-1)] ~ normal(2*q[2:(A-2)] - q[1:(A-3)], s_q);
25   Y ~ multinomial(mu);
26  }
```

Lines 17 to 20 correspond to Eq. (12.11), line 24 corresponds to Eq. (12.10) and line 25 corresponds to Eq. (12.12). Because Stan can implement the multinomial distribution by automatically obtaining the sum from `Y`, there is no need to pass the total population.

A part of the R/Python code that runs the above Stan code is as follows:

A part of code **run-model12-4.R**

```
1    Age_idx <- seq(from=0, to=75, by=5) + 1
2    Flow <- lapply(1:length(Age_idx), function(i) {
3       data.frame(from = Age_idx[i] + c(-2,-1,1,2),
4                          to = Age_idx[i])
5    }) %>%
6    bind_rows() %>%
7    filter(from > 1 & from <= 76)
```

A part of code **run-model12-4.py**

```
1    Age_idx = np.arange(0, 76, 5) + 1
2    Flow = []
3    for a in Age_idx:
4      for f in [-2,-1,1,2]:
5        Flow.append([a + f, a])
6    Flow = pandas.DataFrame(Flow, columns=['from', 'to'])
7    Flow = Flow[(Flow['from'] > 1) & (Flow['from'] <= 76)]
```

In line 1, we create the indices for those ages that caused the flow. Because index 1 is used for the 0-years old in Stan, we added $+1$ to them. From lines 2–7, we create indexes for each of those ages that these population flow into. With `(-2,-1,1,2)`, we represent the minus/plus 2 years of the age that potentially caused the age flow. With this, the flows from -1 years to 0 years-old or from 76 to 75-years old would be included, therefore we removed these unrealistic cases in line 7.

Here, we divided the data by Sex, and $\vec{q}$ was estimated for each gender. By interpolating the estimated result of $\vec{q}$ on top of the structure proportion of the original data, we can obtain Fig. 12.5 (left). Although there is no big difference between the observed and estimated values under the age of 18, results of all the other ages suggest that there are some discrepancies between them. The estimated result of $\vec{r}$ is shown in Fig. 12.5 (right). This plot tells us that there is no large difference between male and female population. Also, older population are more likely to answer rounded numbers, even though the error bars are relatively higher.

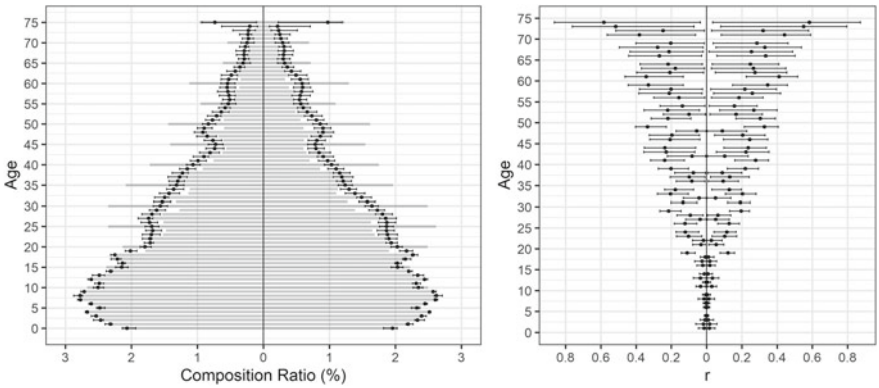


Fig. 12.5 Left: the estimated result of $\vec{q}$. The gray bars are the proportion structure for each gender, aggregated from the data (that is, the proportion between population in each age and the total population in each gender). Right: the estimated values of $\vec{r}$. In both plots, the error bars are the 95% Bayesian confidence intervals, and the black dots are the medians

12.4 Two-Dimensional GMRF

In this section, we will introduce a two-dimensional GMRF. Similar to the one-dimensional case, we can still view it as a spring system. To derive the model formula correctly, we need to start from understanding how to represent a GMRF with a multivariate normal distribution. However, we will leave this part to the next column "Using a multivariate normal distribution to represent a GMRF" for those readers who are interested. In this section, we will not get into the detail on its derivation and only introduce the model as the result of the derivation.

If we consider the connections between neighboring elements only, then this will give us the first-order difference one-dimensional GMRF. Ignoring the normalization terms, this model can be expressed as below:

$$\vec{f} \sim \frac{1}{\sigma_f^{N-1}} \exp\left[-\frac{1}{2\sigma_f^2} \sum_{(i,j) \text{ and } (i',j') \text{ are neighbors}} \left(f[i,j] - f\left[i',j'\right] \right)^2 \right] \tag{12.13}$$

where N is the total number of the nodes, and $f[i, j]$ is an array of nodes that align on the two-dimensional space. We sum across all the nodes that are next to each other. $\vec{f}$ on the left-hand side is the variable that follows a GMRF and it is a vector obtained by rearranging all nodes in one dimension. This formula states that the joint distribution of $\vec{f}$ is determined by the right-hand side of the equation. It looks different but essentially is the same as the one-dimensional case.

When the nodes $f[i, j]$ are aligned on the regular grid, we can also consider a second-order difference that takes into account the connections between the nodes that are located in its front, back, left side, and right side. Such a model formula can be expressed as follows (the normalization constant is ignored):

$$\vec{f} \sim \frac{1}{\sigma_f^{N-3}} \exp\left[-\frac{1}{2\sigma_f^2} \sum_{i=2}^{I-1} \sum_{j=2}^{J-1} (f[i-1,j] + f[i+1,j] + f[i,j-1] + f[i,j+1] - 4f[i,j])^2 \right] \tag{12.14}$$

where I represents the number of vertically-aligned nodes on the grid, and J represents the number of horizontally-aligned nodes on the grid. The other variables have the same meaning as in the first-order difference case. The inside of the summation on the right-hand side is the second derivatives on the vertical and horizontal direction (which is called Laplacian), and is expressed by the difference approximation. It is also possible to consider another approximation, such as by looking at a node and eight nearest neighboring nodes. In this book, we will only use Eq. (12.14). Note that there is a term σ_f^{N-3} outside of exp, which is slightly different from the first-order difference case. This term comes from the rank of a precision matrix (which is the inverse of the covariance matrix) when we express a GMRF using a multivariate normal distribution.

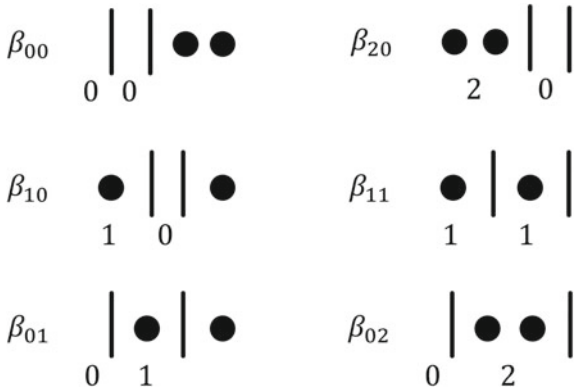


Fig. 12.6 Representing the arrangements of walls and balls in the case of $d = 2, k = 3$. Each arrangement represents the subscripts of β

From Sects. 12.5 to 12.6, we will introduce some examples that use the two-dimensional GMRF for the two-dimensional spatial data.

Technical Point: Using a multivariate normal distribution to represent a GMRF

For simplicity and the computing speed on Stan, in Sect. 12.4, we avoided using a multivariate normal distribution for a GMRF. In fact, a GMRF can be expressed as a multivariate normal distribution and this is mathematically more general way to write a GMRF. Here, we will introduce such an expression and explain the rank of its precision matrix.

Generally, a GMRF can be expressed as the likelihood with a multivariate normal distribution:

$$\vec{f} \sim \frac{1}{\sigma_f^{N-\Delta}} \exp\left[-\frac{1}{2\sigma_f^2} \vec{f}^T \mathbf{Q} \vec{f}\right] \tag{12.15}$$

where N represents the total number of the nodes and $\mathbf{Q}$ represents the precision matrix (inverse of the covariance matrix) with $N \times N$, and it represents the connection between the node pairs. In particular, when the two nodes $f[i]$ and $f\left[i'\right]$ in $\vec{f}$ have a connection, then the value $\mathbf{Q}\left[i, i'\right]$ will be nonzero. Δ represents the rank decrease for $\mathbf{Q}$ (we will explain this later). The GMRF has a property that only some elements in $\mathbf{Q}$ will be nonzero (i.e., $\mathbf{Q}$ is a sparse matrix).

In a one-dimensional first-order difference GMRF (Eq. (12.6)), $\mathbf{Q}$ is:

$$\mathbf{Q} = \begin{pmatrix} 1 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -1 & 2 & -1 & \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 1 \end{pmatrix}$$

Here, note that a linear constraint $\mathbf{Q}\vec{1} = \vec{0}$ holds. This constraint is from the fact that if we add a constant (intercept) to $\vec{f}$ and shift it, it will not change the overall likelihood. This is because the likelihood only depends on the differences between each element in $\vec{f}$. Mathematically, for any constant β_0, $\left(\vec{f} + \vec{1}\beta_0\right)^T \mathbf{Q}\left(\vec{f} + \vec{1}\beta_0\right) = \vec{f}^T \mathbf{Q} \vec{f}$ satisfies. Therefore, the likelihood remains the same. In other words, we can say that using i as an index of locations, if we change $f[i]$ to $f[i] + \beta_0$, the likelihood remains the same. Because of this constraint, the rank of $\mathbf{Q}$ only decreases by 1, which gives $\Delta = 1$.

In a one-dimensional second-order difference GMRF (Eq. (12.9)), $\mathbf{Q}$ is:

$$\mathbf{Q} = \begin{pmatrix} 1 & -2 & 1 & & & & & & \\ -2 & 5 & -4 & 1 & & & & & \\ 1 & -4 & 6 & -4 & 1 & & & & \\ & 1 & -4 & 6 & -4 & 1 & & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & & \\ & & & 1 & -4 & 6 & -4 & 1 & \\ & & & & 1 & -4 & 6 & -4 & 1 \\ & & & & & 1 & -4 & 5 & -2 \\ & & & & & & 1 & -2 & 1 \end{pmatrix}$$

Here, note that a constraint $\mathbf{Q}\mathbf{S}_1 = \vec{0}$ holds, and $\mathbf{S}_1$ is a matrix as below:

$$\mathbf{S}_1 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ \vdots & \vdots \\ 1 & i \\ \vdots & \vdots \\ 1 & I \end{pmatrix}$$

where I is the total number of the nodes. Note that the first column of $\mathbf{S}_1$ is $\vec{1}$, and the second column is the index of each node i. This constraint is from the fact that if we add a linear line (intercept + slope × index) to $\vec{f}$, the overall likelihood remains the same. This is because the likelihood is directly determined by the difference of the elements in $\vec{f}$ (e.g., $(f[3] - f[2]) - (f[2] - f[1])$), and the influence by these lines are canceled out. Mathematically, for any $\vec{\beta} = (\beta_0 \beta_1)^T$, $\left(\vec{f} + \mathbf{S}_1\vec{\beta}\right)^T \mathbf{Q}\left(\vec{f} + \mathbf{S}_1\vec{\beta}\right) = \vec{f}^T\mathbf{Q}\vec{f}$ satisfies. Therefore, the likelihood stays the same. We can also say that using the location index i, changing from $f[i]$ to $f[i] + \beta_0 + \beta_1$ does not change the likelihood. Because of the constraint $\mathbf{QS}_1 = \vec{0}$, the rank of $\mathbf{Q}$ will decrease only by two, and therefore $\Delta = 2$.

In general, for a d-dimensional k-order difference GMRF, the rank decrease is given by:

$$\Delta = \binom{d + k - 1}{k - 1} \tag{12.16}$$

Note that the bracket here represents binomial coefficient rather than a vector. We will explain how to derive Eq. (12.16). Here, $\vec{f}$ is a vector obtained by rearranging all nodes in one dimension, and we assume that the likelihood does not change if we change $\vec{f}$ to $\vec{f} + \mathbf{S}\vec{\beta}$. Then the number of columns of $\mathbf{S}$ (this is equal to the number of elements in $\vec{\beta}$) corresponds to the decrease in the rank. Similar to the examples so far, if we calculate a term that does not change the likelihood when it is added to a node $f[i_1, i_2, \ldots, i_d]$, this is:

$$\sum_{0 \le k_1 + k_2 + \ldots + k_d \le k-1} \frac{1}{k_1! k_2! \ldots k_d!} \beta_{k_1 k_2 \ldots k_d} i_1^{k_1} i_2^{k_2} \ldots i_d^{k_d}$$

where the subscript of β is represented as the integers with the total number of d $(k_1, \ldots, k_d)$. And the number of the combinations of those $k_1, \ldots, k_d$ determines the number of elements in $\vec{\beta}$. Each $k_1, \ldots, k_d$ is an integer within the range of $[0, k - 1]$, and there is an additional constraint—the sum of $k_1, \ldots, k_d$ will be smaller than $k-1$. The number of combinations of $k_1, \ldots, k_d$ that satisfy such constraints is the same number of the unique arrangements of $d + k - 1$ objects, which consist of the indistinguishable d walls and indistinguishable $k-1$ balls. If we order the walls from left to right and label them as $1, 2, \ldots, d$, then:

- The number of the balls on the left side of the first wall is k_1
- The number of the balls between the first and second walls is k_2
- …
- The number of the balls between the $(d - 1)$-th and d-th wall is k_d

– The balls on the right side of the d-th wall will be discarded.

With this setting, the previous conditions are satisfied. The number of possible arrangements of $d + k - 1$ objects is $(d + k - 1)!$. By dividing this number with $d!$ (the number of possible arrangements of d walls) and $(k - 1)!$ (the number of arrangements of $k - 1$ balls), we can obtain Eq. (12.16).

As an example, we explain the case of two-dimensional third-order difference GMRF ($d = 2, k = 3$). By writing (i_1, i_2) as (i, j), then the following term does not change the likelihood when it is added to $f[i, j]$:

$$\beta_{00} + \beta_{10} i + \beta_{01} j + \frac{1}{2}\beta_{20} i^2 + \beta_{11} ij + \frac{1}{2}\beta_{02} j^2$$

Because there are six elements in $\vec{\beta}$, the rank decrease will be six. We illustrate the corresponding walls and the ball arrangement status in Fig. 12.6. For those who are interested to know more about GMRFs, please refer to (Hávard & Leonhard, 2005).

12.5 (Example 3) Geospatial Data on the Map

A GMRF has a wide range of applications on data which can represent the neighboring connections using nodes and edges. In this section, we introduce an example where we use a GMRF to analyze the geospatial data on the map. For this purpose, let us look into the yearly average temperatures of different prefectures in Japan in 2013. The prefectures can be considered as the states in the United States. For this analysis, we prepared two comma-separated files (Data File 12.3 and 12.4). Data File 12.3 includes the yearly average temperature of each prefecture, and $prefID$ column is the index for each prefecture, and Y column represents the temperature (unit in Celsius). The prefectures are ordered based on the JIS code, from Hokkaido with $prefID = 1$ and Okinawa with $prefID = 47$. Data File 12.4 includes the neighboring connections. It indicates that `prefID` on the $From$ column and `prefID` on the To column are spatially adjacent prefectures. Only the case $From < To$ is kept.

Data File 12.3 Structure of `data-map-temperature.csv`

```
prefID,Y
1,9.2
2,10.5
3,10.6
...
47,23.3
```

Data File 12.4 Structure of `data-map-neighbor.csv`

```
  1    From,To
  2    1,2
  3    2,5
  4    2,3
       ...
101    45,46
```

Figure 12.7 (left) is obtained by mapping the yearly average temperature onto the map of Japan. Here, we assume that the yearly temperatures between neighboring prefectures are similar, and thus we use a GMRF. Assuming that the observed yearly temperature $Y[i]$ is generated from $f[i]$, which is determined by the neighboring connections, and other factors. We consider that the aim of this analysis is to detect those prefectures that are affected largely from other factors besides $f[i]$.

The model formula can be written as follows:

Model Formula 12.5

$$\vec{f} \propto \frac{1}{\sigma_f^{I-1}} \exp\left[-\frac{1}{2\sigma_f^2} \sum_{k=1}^{K} (f[To[k]] - f[From[k]])^2 \right] \tag{12.17}$$

$$Y[i] \sim \mathcal{N}\big(f[i], \sigma_y\big) \quad i = 1, \ldots, I$$

where I denotes the total number of prefectures and index i denotes each prefecture. K denotes the total number of neighboring connections between prefectures and the

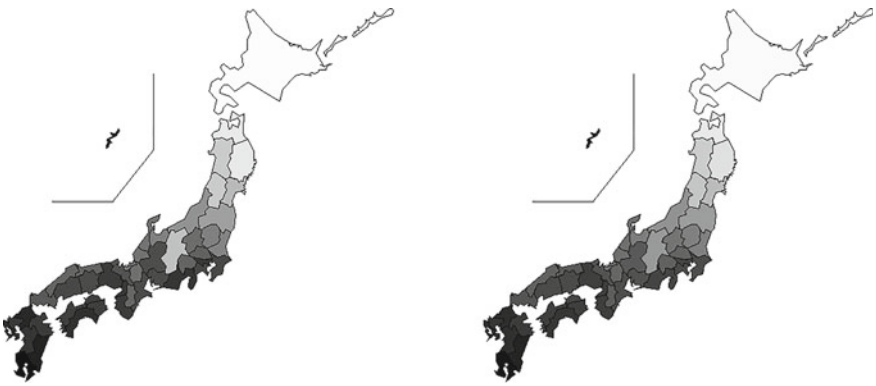


Fig. 12.7 Left: measured yearly average temperature. Right: medians of the spatial structure $f[i]$ that are estimated from the GMRF. In both plots, lighter and darker colors indicate lower and higher temperatures, respectively

index k denotes each connection. Equation (12.17) is an example of Eq. (12.13). The parameters $f[i]$, σ_f and σ_y will be estimated from the data.

Its implementation can be as follows:

model12-5.stan

```
1   data {
2    int I;
3    vector[I] Y;
4    int K;
5    array[K] int<lower=1, upper=I> From;
6    array[K] int<lower=1, upper=I> To;
7   }
8
9   parameters {
10    vector[I] f;
11    real<lower=0> s_f;
12    real<lower=0> s_y;
13   }
14
15   model {
16    vector[K] diff = f[To] - f[From];
17    target += - (I-1) * log(s_f)
18       - 0.5 * inv_square(s_f) * dot_self(diff);
19    Y[1:I] ~ normal(f[1:I], s_y);
20   }
```

Line 16: `diff` is vectorized by using one-dimensional arrays that store indices.

Lines 17–18: These lines correspond to the log-transformed Eq. (12.17).

By overlaying the estimated spatial structure $f[i]$ onto the map of Japan, we obtain Fig. 12.7 (right). We note that the temperature differences are slightly smoothed and it gives more gradual changes. Figure 12.8 shows the posterior predictive check and the posterior residual check (see Sect. 5.2). Here, the residual corresponds to $Y[i] - f[i]$. The absolute median of the other factors' effect ($Y[i] - f[i]$) is largest in Nagano prefecture with $-1.50\,°\mathrm{C}$ (Fig. 12.8 (left)). From Fig. 12.8 (right), we can see that the effect is approximately distributed as a normal distribution, except that Nagano has an extremely small value. For this reason, it would also be reasonable to replace a normal distribution with some other distributions that have longer tails, such as a Students' t distribution.

12.6 (Example 4) Data on Two-Dimensional Grid

In this section, we will introduce the application of a GMRF on two-dimensional grid data, which is one of the most frequently used type of data. For this purpose, we use a dataset from a rectangular plastic plate which is widely used in biological experiments. It is a plate with multiple holes, and is used when we want to treat a large number of compounds at the same time. Here, we consider a case where we want to

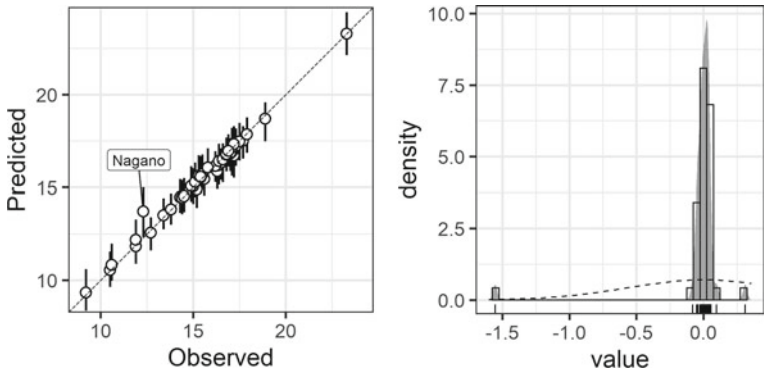


Fig. 12.8 Left: PPC. The x-axis represents $Y[i]$ and the y-axis represents the 95% Bayesian confidence intervals and the median of $f[i]$. Right: PRC. The x-axis represents the MAP estimate of $Y[i] - f[i]$ and it is further visualized as a histogram and a density function. The dotted line is a density function of a normal distribution with mean 0 and the SD equal to MAP estimate of `s_y` in **`model12-5.stan`**

know if any of 96 different treatments has efficacy (that is, to know if Y is high), and for this experiment, we use a 384-hole plate[5]: 16 holes on the vertical dimension and 24 holes on the horizontal dimension. In this experiment, the 96 different treatments are measured and each treatment has 4 replicates.

Here, we assume that we have background knowledge on the spatial information on the plate. That is, the holes on the center of the plate are more likely to yield higher values. In such a case, it is important that we shuffle these 96 treatments and randomly assign them to the plate holes. Otherwise, if 4 replicates of the same treatment are all assigned on the plate center and they all yielded higher values, it would be hard to distinguish whether it was the effect by the treatment or the effect by their locations on the plate. Here we assume that the treatments and locations were randomized already.

Two comma-separated files are obtained (Data File 12.5 and 12.6). Data File 12.5 is the dataset with 16 rows and 24 columns. The i-th row, j-th column in this dataset represents the value obtained from the i-th row, j-th column on the plate ($Y[i, j]$). Data File 12.6 is also a dataset with 16 rows and 24 columns. The value of the i-th row, j-th column represents the index for the treatment that are given for the hole located at the i-th row, j-th column of the plate.

Data File 12.5 Structure of `data-2Dmesh.csv`

```
  1     1.09,1.16,...,1.83,2.06
  2     1.39,1.85,...,1.54,0.01
  3     -0.81,2.24,...,2.48,2.86
        ...
 16     1.69,1.64,...,-1.13,-0.33
```

[5] For those who are not familiar, please search the images of 384-hole plates to get a better intuition on how they look like.

Data File 12.6 Structure of `data-2Dmesh-design.csv`

```
 1      1.69,1.64,...,-1.13,-0.33
 2      81,96,...,17,26
 3      51,61,...,88,56
        ...
16      63,33,...,26,9
```

Now we want to separate the effects by the different treatments from the effects by the plate locations, and understand the effects by these treatments. In addition, we want to estimate the effects by the location on the plate as well.

First, to check the data distribution, we can visualize $Y[i, j]$ on a heatmap, as shown in Fig. 12.9. We notice that the hole on the inner side of the plate yielded higher value and the outer side yielded lower values.

Next let us imagine the data generating mechanisms. First, we decompose the observed value $Y[i, j]$ into multiple components. That is, we assume that each value of $Y[i, j]$ is the summation of the effects by the location on the plates (spatial structure) $f[i, j]$, the effect by the treatment $\beta[ij2t[i, j]]$, and the observation noise $\varepsilon[i, j]$. For the spatial structure, we use second-order difference GMRF. Because we consider that the effects by different treatments $\beta[t]$ are similar, we can add a loose constraint by assuming that they follow a specific distribution, and we use a normal distribution as an example here. From the above discussions, we can write a model formula as follows:

Model Formula 12.6

$$Y[i, j] = f[i, j] + \beta[ij2t[i, j]] + \varepsilon[i, j] \quad i = 1, \ldots, I \quad j = 1, \ldots, J \tag{12.18}$$

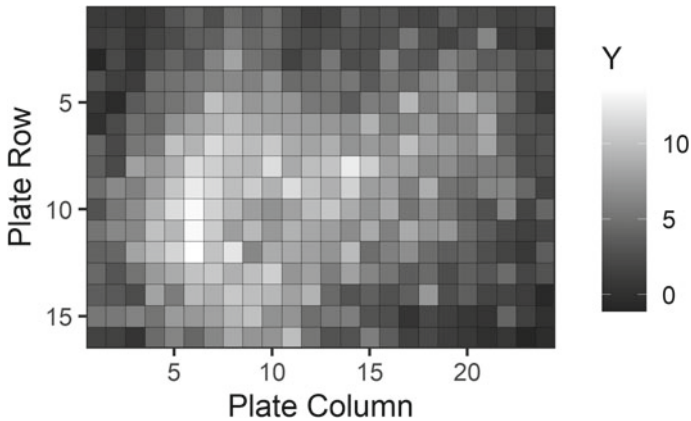


Fig. 12.9 The location on the plate and the observed values $Y[i, j]$

$$\vec{f} \sim \frac{1}{\sigma_f^{IJ-3}} \exp\left[-\frac{1}{2\sigma_f^2} \sum_{i=2}^{I-1} \sum_{j=2}^{J-1} (f[i-1,j] + f[i+1,j] + f[i,j-1] + f[i,j+1] - 4f[i,j])^2 \right] \tag{12.19}$$

$$\beta[t] \sim \mathcal{N}\big(0, \sigma_\beta\big) \quad t = 1, \ldots, T \tag{12.20}$$

$$\varepsilon[i,j] \sim \mathcal{N}\big(0, \sigma_y\big) \quad i = 1, \ldots, I \quad j = 1, \ldots, J \tag{12.21}$$

where I and J indicate the row and column indices, respectively. T represents the total number of treatments, and t is the index for each treatment. Equation (12.19) is an example of Eq. (12.14). The parameters $f[i,j], \sigma_f, \beta[t], \sigma_\beta$ and σ_y will be estimated from the model.

The implementation of Model Formula 12.6 is shown as below:

model12-6.stan

```
1   data {
2       int I;
3       int J;
4       matrix[I,J] Y;
5       int T;
6       array[J,I] int<lower=1, upper=T> ji2t;
7   }
8
9   parameters {
10      matrix[I,J] f;
11      real<lower=0> s_f;
12      vector[T] beta;
13      real<lower=0> s_beta;
14      real<lower=0> s_y;
15   }
16
17   model {
18     vector[(I-2)*(J-2)]diff  = to_vector(
19        f[1:(I-2), 2:(J-1)] + f[3:I, 2:(J-1)] +
20        f[2:(I-1), 1:(J-2)] + f[2:(I-1), 3:J] - 4*f[2:(I-1), 2:(J-1)]
21      );
22      target += - (I*J-3) * log(s_f)
23          - 0.5 * inv_square(s_f) * dot_self(diff);
24
25      beta[1:T] ~ normal(0, s_beta);
26      to_vector(Y) ~ normal(to_vector(f) + beta[to_array_1d(ji2t)],s_y);
27      s_y ~ normal(0, 0.2);
28   }
```

Line 6: The data is passed to `ji2t`, to inform which treatment ($t = 1, \ldots, T$) was given at which location. Here, we assume that the passed data is already transposed, for the processing convenience on line 26.

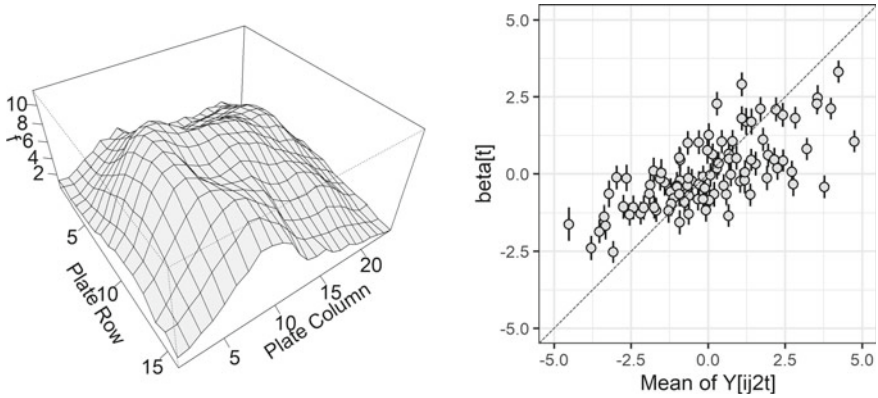


Fig. 12.10 Left: 3d plot of the plate hole coordinates (i, j) and the median of $f[i, j]$. Right: the x-axis is the arithmetic mean value of Y for each treatment group (with the total mean of Y subtracted from each). The y-axis is the estimated effect of each treatment $\beta[t]$ (the 95% Bayesian confidence intervals and the medians are shown). Each dot on the plot represents one type of treatment, and thus there are 96 dots in total

Lines 18–23: These correspond to Eq. (12.19). From lines 19 to 20, the computation is done with the matrix and then the function `to_vector` is used on line 18 to transform the matrix into a one-dimensional `vector`.

Line 26: `to_array_1d` is a function that transforms an array with multiple dimensions to a one-dimension array, in row-major order.[6] In contrast, `to_vector` function converts the input into `vector` type in column-major order. Therefore, in order to make each element consistent, we have transposed `ji2t` on line 6.

Here, we omit the R and Python code to run the above Stan code. The estimated results are shown in Fig. 12.10. From Fig. 12.10 (left), we notice that the estimated result of $f[i, j]$ is consistent with our background knowledge. From Fig. 12.10 (right), we compared the analysis result with the arithmetic means of the observed values for each treatment group. We can notice that using different methods to estimate the treatment effects will have a large impact on the results. In other words, if we simply take the mean for each treatment, the values on the edge or center of the plate will drag the overall value, despite the randomization in the experiment. Here, we can conclude that by building a model that incorporates the spatial structure and by removing the effect by the locations, the estimation was done appropriately.

[6] For example, if we have a two-dimensional array with size 2×2, it will be ordered into a one-dimensional array as `a[1,1], a[1,2], a[2,1], a[2,2]`.

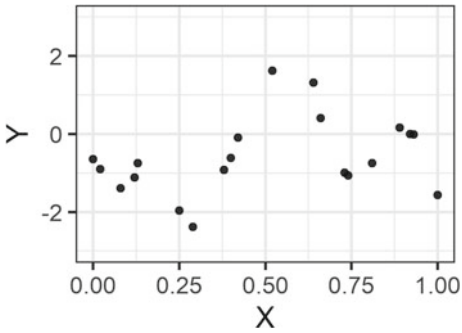


Fig. 12.11 Scatter plot

12.7 Introduction to GP

As we have seen in the grid data, for the data where the locations (coordinates) are the discrete values, we used a GMRF. From this section, we introduce how to analyze the data where the locations are the continuous values. There are two main different approaches for the continuous locations. The first approach is equivalent to the one introduced in Sect. 11.4.2. That is, we first discretize the spatial space into small and equally-spaced intervals. Subsequently, each location is approximated by the closest point on the grid. The second approach, a *Gaussian Process (GP)*, is what we will introduce here. Similar to a GMRF, the GP assumes that spatially nearby points have similar observation values. The GP can be considered as a model that generalizes the GMRF, and it has more capacity to represent smooth functions, and thus is likely to improve the prediction performance. A caveat is that when the data is large, the computation would be very expensive for a GP and the estimation would take a long time. Thus, we need to consider which method to use for each problem.

As an example for this section, we use a dataset where the location is given as a one-dimensional array. It contains the location X and the corresponding observation value Y at that location (Data File 12.7, a row represents a data point). Let the total number of the data points be N ($N = 20$ here). Also, we assume that Y is already centralized.[7] The scatterplot is shown on Fig. 12.11.

Data File 12.7 Structure of `data-gp.csv`

```
X,Y
0.00,-0.65
0.02,-0.9
0.08,-1.39
...
1.00,-1.56
```

Here, as the data generation mechanism, we assume that Y is generated by adding noise to a smooth function $f(x)$ which is determined by x.

[7] It is a preprocessing step where the mean of Y is subtracted from each $Y[n]$. It is called centerization, as this step forces the mean of the data to be zero.

$$Y[n] \sim \mathcal{N}\big(f[n], \sigma_y\big) \quad n = 1, \ldots, N \tag{12.22}$$

Next, we assume that $f(x)$ *follows a GP*. That is, we assume that the value of $f(x)$ follows a multivariate normal distribution that is determined by the location X as follows:

$$\begin{pmatrix} f[1] \\ f[2] \\ \vdots \\ f[N] \end{pmatrix} \sim \text{MultiNormal}\left(\begin{pmatrix} \mu(X[1]) \\ \mu(X[2]) \\ \vdots \\ \mu(X[N]) \end{pmatrix}, \begin{pmatrix} k(X[1], X[1]) & k(X[1], X[2]) & \cdots & k(X[1], X[N]) \\ k(X[2], X[1]) & k(X[2], X[2]) & \cdots & k(X[2], X[N]) \\ \vdots & \vdots & \ddots & \vdots \\ k(X[N], X[1]) & k(X[1], X[1]) & \cdots & k(X[N], X[N]) \end{pmatrix} \right) \tag{12.23}$$

where $f[1]$, …, $f[N]$ on the left-hand side are abbreviations for $f(X[1])$, …, $f(X[N])$. $\mu(x)$ is called a mean function as it represents the mean value at each location x. $k\big(x, x'\big)$ is called a *kernel function*, and it represents the covariance (including correlation) between x and x'. Both these two functions return scalar values. By writing the left-hand side as $\vec{f}$ and writing the mean vector and the covariance matrix as $\vec{\mu}$ and $\mathbf{K}$, we often express them as follows:

$$\vec{f} \sim \text{MultiNormal}(\vec{\mu}, \mathbf{K}) \tag{12.24}$$

$\mathbf{K}$ is called a *kernel matrix* because it is computed from a kernel function. An element in the kernel matrix, $K\big[n, n'\big]$, refers to $k\big(X[n], X\big[n'\big]\big)$. When we want to emphasize that $\vec{f}$ follows a GP, we can also write it as:

$$\vec{f} \sim \text{GP}(\mu, k) \tag{12.25}$$

There are multiple ways to set the functions μ and k. Based on which kernel function we choose, a GP can be equivalent to some well-known models such as linear model, spline model, neural network, and support vector machine. Also, we can set it by combining several kernel functions. More detailed explanation will be over the scope of this book, but for those who are interested, it might be helpful to refer to some specialized books (for instance, Rasmussen & Williams, 2006).[8]

Next, we will introduce some frequently used ways to set the functions μ and k. For the mean function, we usually set $\vec{\mu} = \vec{0}$ after centering Y. Alternatively, we can fix $\vec{\mu}$ to be a vector of length N, with each element representing the mean of $\vec{Y}$. Hereafter, we use $\vec{\mu} = \vec{0}$. The Gaussian kernel is one of the most frequently used one for the kernel function:

$$k\big(X[n], X\big[n'\big]\big) = a^2 \exp\left(-\frac{1}{2\rho^2} \big(X[n] - X\big[n'\big]\big)^2 \right) \tag{12.26}$$

[8] This book is available online for free: http://www.gaussianprocess.org/gpml/.

where the parameter a controls the strength of the correlation between the two outputs $f[n]$ and $f\left[n'\right]$, and the parameter ρ determines the scale of the distance between the two locations $X[n]$ and $X\left[n'\right]$. When the Euclidean distance between two locations $X[n]$ and $X\left[n'\right]$ is smaller, this kernel function will be very close to a^2, and when they are further away, it will reach to zero. From the covariance matrix aspect, this means that with a smaller distance, the correlation will be higher, and when the distance becomes larger, their correlation will be lower. As a result, it can express our original assumption: when the spatial locations of two values are close, their values should be similar. Equation (12.26) is applicable to the case where the location is one dimension, and it can also be generalized to the D-dimensional location:

$$k\left(\overrightarrow{X[n]}, \overrightarrow{X\left[n'\right]}\right) = a^2 \exp\left(-\frac{1}{2\rho^2}\sum_{d=1}^{D}\left(X[n,d] - X\left[n',d\right]\right)^2\right) \tag{12.27}$$

A GP generates a function $\vec{f}$ that can be expressed as a curve (note that this becomes a curved surface in a D-dimensional case). In other words, the draws generated from a GP are curves. To facilitate the understanding, in Fig. 12.12, we generated five curves after assigning constant values for a and ρ in the Gaussian kernel given in Eq. (12.26). In order to make it clearer, we provided the R and Python code that generated these five curves under the setting $a = 0.5, 2$ and $\rho = 0.05, 0.2$ (**simGP.R**, **simGP.py**). This simulation also provides good insight into how to set the prior distribution for a and ρ.

simGP.R

```
library(mvtnorm)
set.seed(123)

S <- 5 # number of draws
N <- 101 # length of f
X <- seq(0, 1, len=N)
a <- 2
rho <- 0.05

K <- matrix(rep(0, N*N), nrow=N)
for (i in 1:N) {
  for (j in 1:N) {
     K[i,j] <- a^2 * exp(-0.5/rho^2*(X[i]-X[j])^2)
  }
}

f_draws <- rmvnorm(S, mean=rep(0 , N), sigma=K)
```

simGP.py

```
import numpy as np
np.random.seed(123)

S = 5 # number of draws
N = 101 # length of f
X = np.linspace(0, 1, num=N)
```

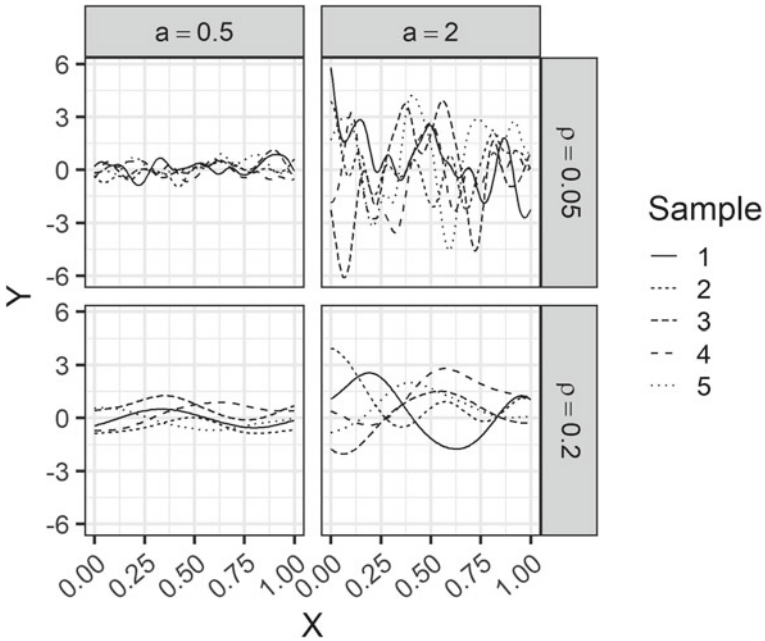


Fig. 12.12 Five curves generated from GP using a Gaussian kernel

```
7   a = 2
8   rho = 0.05
9
10   K = np.zeros((N, N))
11   for i in range(0, N):
12         for j in range(0, N):
13             K[i,j] = a**2 * np.exp(-0.5/rho**2 * (X[i]-X[j])**2)
14
15  f_draws = np.random.multivariate_normal(np.repeat(0, N), K, S)
```

The kernel matrix is generated on lines 10–15 in R (lines 10–13 in Python). Line 13 corresponds to Eq. (12.26), and line 17 in R (line 15 in Python) generates `S` curves by following Eq. (12.24). One curve corresponds to one draw of $\vec{f}$. The generated `f_draws` is a matrix type, with `S` rows and `N` columns (here it is 5 rows with 101 columns).

12.7.1 Implementation of GP (1)

We define the combination of GP Eq. (12.24), Gaussian kernel Eq. (12.26), and observation model Eq. (12.22) as Model Formula 12.7. In the following part, we will

implement Model Formula 12.7 to deepen our understanding on the GP. First of all, we can naively implement the model without doing any variable transformation.

model12-7a.stan

```
1   functions {
2     real gp_lpdf(vector out, array[] real x, vector mu,
3                  real a, real rho, real s2) {
4       int N = size(x);
5       matrix[N,N] k_addI = add_diag(gp_exp_quad_cov(x, a, rho), s2);
6       return multi_normal_cholesky_lpdf(out|mu, cholesky_decompose
       (k_addI));
7     }
8   }
9
10   data {
11     int<lower=1> N;
12     array[N] real X;
13     vector[N] Mu;
14     vector[N] Y;
15   }
16
17   parameters {
18     vector[N] f;
19     real<lower=0> a;
20     real<lower=0> rho;
21     real<lower=0> s_y;
22   }
23
24   model {
25     a ~ normal(0, 1);
26     rho ~ normal(0.1, 0.1);
27     s_y ~ normal(0, 1);
28     f ~ gp(X, Mu, a, rho, 1e-8);
29     Y[1:N] ~ normal(f[1:N], s_y);
30   }
```

Lines 2–7: Defining the log-likelihood of GP that corresponds to Eq. (12.24).

Line 4: The function `size` is used to get the length of array `x`.

Line 5: Building the kernel matrix. `gp_exp_quad_cov` function is a handy function that returns a kernel matrix by following the Gaussian kernel given in Eq. (12.26). Because the function takes an array of `real` type or an array of `vector` type, `X` is declared as such in line 12. `add_diag` function is a function that adds a real number to the diagonal elements of a matrix. Here, we pass a small constant `1e-8` and add it to the diagonal elements, and hereby forcing the kernel matrix to be positive definite. This step ensures that the Cholesky decomposition is stable in line 6.

Line 6: As we addressed in Sect. 9.2.4, `multi_normal_cholesky_lpdf` is a fast and stable version of `multi_normal_lpdf`, and it returns a log-likelihood of a multivariate normal distribution. It takes its own Cholesky factors, instead of the covariance matrix of the multivariate normal distribution.

Lines 25–26: Assigning the priors for the two parameters (a and ρ) that are involved in the Gaussian kernel. This part needs to be modified based on the particular questions and background knowledge.

Line 28: This line represents that `f` follows a GP, and it corresponds to Eq. (12.24). The expression using "∼" is made possible here in Stan because we define a log-likelihood function as `gp_lpdf()`, as we mentioned in Sect. 10.2.

Line 29: This line represents the observation model, corresponding to Eq. (12.22).

However, sampling `f` is likely to fail if we directly implement **`model12-7a.stan`**. Here, we will do reparameterization for the multivariate normal distribution, as we introduced in Sect. 9.3.3.

`model12-7b.stan`

```
functions {
  vector gp(array[] real x, vector mu, vector eta,
            real a, real rho, real s2) {
    int N = size(x);
    matrix[N,N] k_addI = add_diag(gp_exp_quad_cov(x, a, rho), s2);
    vector[N] out = mu + cholesky_decompose(k_addI)*eta;
    return out;
  }
}

data {
  int<lower=1> N;
  array[N] real X;
  vector[N] Mu;
  vector[N] Y;
}

parameters {
  vector[N] eta;
  real<lower=0> a;
  real<lower=0> rho;
  real<lower=0> s_y;
}

transformed parameters {
  vector[N] f = gp(X, Mu, eta, a, rho, 1e-8);
}

model {
  eta ~ normal(0, 1);
  a ~ normal(0, 1);
  rho ~ normal(0, 0.1);
  s_y ~ normal(0, 1);
  Y[1:N] ~ normal(f[1:N], s_y);
}
```

Line 2: We changed the function `gp` so that it takes `eta` as the input and then converts it to `out`.

Line 26: The values of `f` is obtained deterministically by the defined `gp` function.

Line 30: See the reparameterization for a multivariate normal distribution in Sect. 9.3.3.

Here we omitted the R and Python code that runs this Stan code. Because Data File 12.7 is already centralized, we only need to pass zero vectors to the variable `Mu` in `data` block.

In fact, by transforming `f` in this implementation, we can further use other distributions for the observation model, such as a Poisson distribution or a binomial distribution. We will give one of those examples in Sect. 12.4.

12.7.2 Implementation of GP (2)

In this subsection, we implement the GP using a different method from the previous section. This method uses the fact that when the observation model follows a normal distribution, $\vec{f}$ can be marginalized out by using the reproductive property of a multivariate normal distribution. In particular, we add the observation noise σ_y^2 on the diagonal elements of the covariance matrix of the GP that generates $\vec{f}$. By doing this, we can directly represent the GP that generates $\vec{Y}$, and therefore the process of generating $\vec{f}$ can be skipped. With this marginalization, the computation becomes faster and more stable. Therefore, when we do not need to estimate the value of $\vec{f}$ itself, we recommend using this implementation. Equations (12.22) and (12.24) can be written into the mathematical expression as follows:

$$\vec{Y} \sim \text{MultiNormal}\left(\vec{\mu}, \mathbf{K} + \sigma_y^2 \mathbf{I}\right) \tag{12.28}$$

where $\mathbf{I}$ is an identity matrix with size $N \times N$.

We illustrate this implementation below. This code is based on **`model12-7a.stan`** which we failed to implement earlier.

`model12-7c.stan`

```
1  (functions block and data block are the same as model12-7a.stan)
2
3  parameters {
4    real<lower=0> a;
5    real<lower=0> rho;
6    real<lower=0> s_y;
7  }
8
9  model {
```

```
10     a ~ normal(0, 1);
11     rho ~ normal(0, 0.1);
12     s_y ~ normal(0, 1);
13     Y ~ gp(X, Mu, a, rho, square(s_y));
14   }
```

Note that this time, we did not declare `f` in `parameters` block. Additionally, the original value `1e-8` which was added to the diagonal elements on the covariance matrix on line 13 is replaced to be `square(s_y)`, which is the variance of the observation noise.

12.7.3 Prediction with GP

In this subsection, we introduce how to do prediction using a GP. In particular, we want to predict the values $\overrightarrow{f_p} = \left(f_p[1], \ldots, f_p\left[N_p\right]\right)^T$ and $\overrightarrow{y_p} = \left(y_p[1], \ldots, y_p\left[N_p\right]\right)^T$ at the N_p locations $X_p[1], \ldots, X_p\left[N_p\right]$ where observations are not provided.

For this purpose, we create another vector $\overrightarrow{y'}$ by combining $\vec{Y}$ and $\overrightarrow{f_p}$, that is $\overrightarrow{y'} = \left(Y[1], \ldots, Y[N], f_p[1], \ldots, f_p\left[N_p\right]\right)^T$, and we consider a conditional distribution of $\overrightarrow{f_p}$ given $\vec{Y}$. Note that this new vector $\overrightarrow{y'}$ also follows a GP with a kernel matrix with size $\left(N + N_p\right) \times \left(N + N_p\right)$ as its covariance matrix. In other words,

$$\begin{pmatrix} \overrightarrow{Y} \\ \overrightarrow{f_p} \end{pmatrix} \sim \text{MultiNormal}\left(\begin{pmatrix} \overrightarrow{\mu} \\ \overrightarrow{\mu_p} \end{pmatrix}, \begin{pmatrix} \mathbf{K} + \sigma_y^2 \mathbf{I} & \mathbf{k}_p \\ \mathbf{k}_p^T & \mathbf{k}_{pp} \end{pmatrix}\right) \tag{12.29}$$

where $\mathbf{I}$ is an identity matrix with size $N \times N$, and $\mathbf{k}_p$ is a matrix with size $N \times N_p$ whose elements are:

$$\mathbf{k}_p = \begin{pmatrix} k\left(X[1], X_p[1]\right) & k\left(X[1], X_p[2]\right) & \cdots & k\left(X[1], X_p\left[N_p\right]\right) \\ k\left(X[2], X_p[1]\right) & k\left(X[2], X_p[2]\right) & \cdots & k\left(X[2], X_p\left[N_p\right]\right) \\ \vdots & \vdots & \ddots & \vdots \\ k\left(X[N], X_p[1]\right) & k\left(X[N], X_p[2]\right) & \cdots & k\left(X[N], X_p\left[N_p\right]\right) \end{pmatrix} \tag{12.30}$$

$\mathbf{k}_{pp}$ is a matrix with size $N_p \times N_p$, and its elements are:

$$\mathbf{k}_{pp} = \begin{pmatrix} k\left(X_p[1], X_p[1]\right) & \cdots & k\left(X_p[1], X_p\left[N_p\right]\right) \\ \vdots & \ddots & \vdots \\ k\left(X_p\left[N_p\right], X_p[1]\right) & \cdots & k\left(X_p\left[N_p\right], X_p\left[N_p\right]\right) \end{pmatrix} \tag{12.31}$$

Based on these, we consider the conditional distribution of $\overrightarrow{f_p}$ when $\vec{Y}$ is given. In order to do that, let's first check the conditional distribution of a multivariate normal distribution. If we assume that a vector is the combination of $\overrightarrow{y_a}$ and $\overrightarrow{y_b}$, and it follows a multivariate normal distribution:

$$\begin{pmatrix} \overrightarrow{y_a} \\ \overrightarrow{y_b} \end{pmatrix} \sim \text{MultiNormal}\left(\begin{pmatrix} \overrightarrow{\mu_a} \\ \overrightarrow{\mu_b} \end{pmatrix}, \begin{pmatrix} \mathbf{S}_{aa} & \mathbf{S}_{ab} \\ \mathbf{S}_{ab}^T & \mathbf{S}_{bb} \end{pmatrix} \right) \tag{12.32}$$

The conditional distribution of $\overrightarrow{y_b}$ given the value of $\overrightarrow{y_a}$ is:

$$\overrightarrow{y_b} \,|\, \overrightarrow{y_a} \sim \text{MultiNormal} \quad \left(\overrightarrow{\mu_b} + \mathbf{S}_{ab}^T \mathbf{S}_{aa}^{-1} \left(\overrightarrow{y_a} - \overrightarrow{\mu_a} \right), \mathbf{S}_{bb} - \mathbf{S}_{ab}^T \mathbf{S}_{aa}^{-1} \mathbf{S}_{ab} \right) \tag{12.33}$$

We do not give detailed derivation for Eq. (12.33), but Sect. 2.3 in Bishop (2006)[9] would be helpful for those who are interested in this derivation. Here we only give an intuition of Eq. (12.33): the msssean vector on the right-hand side of the equation is determined by correcting the value of $\overrightarrow{\mu_b}$ using the given value of $\overrightarrow{y_a}$. This correction level is proportional to the strengh of relationship between a and b (that is, $\mathbf{S}_{ab}^T$), as well as the certainty about the value of $\overrightarrow{y_a}$ (that is, $\mathbf{S}_{aa}^{-1}$). Simialrly, the covariance matrix is determined by correcting $\boldsymbol{S}_{bb}$. We can say that the variance is decreased by knowing the value of $\overrightarrow{y_a}$. Similar to the mean vector, this correction level is proportional to $\mathbf{S}_{ab}^T$ and $\mathbf{S}_{aa}^{-1}$. Now let us come back to the GP. If we compare Eqs. (12.29) and (12.32), and use Eq. (12.33), the conditional distribution of $\overrightarrow{f_p}$ given $\vec{Y}$ can be written as:

$$\overrightarrow{f_p} \sim \text{MultiNormal}\left(\overrightarrow{\mu_p^{(\text{cond})}}, \mathbf{K}_p^{(\text{cond})} \right) \tag{12.34}$$

$$\overrightarrow{\mu_p^{(\text{cond})}} = \overrightarrow{\mu_p} + \mathbf{k}_p^T \left(\mathbf{K} + \sigma_y^2 \mathbf{I} \right)^{-1} \left(\vec{Y} - \vec{\mu} \right) \tag{12.35}$$

$$\mathbf{K}_p^{(\text{cond})} = \mathbf{k}_{pp} - \mathbf{k}_p^T \left(\mathbf{K} + \sigma_y^2 \mathbf{I} \right)^{-1} \mathbf{k}_p \tag{12.36}$$

where (cond) represents the parameters of the conditional distribution. For $\vec{\mu}$ and $\overrightarrow{\mu_p}$, we usually set $\vec{\mu} = \vec{0}$ and $\overrightarrow{\mu_p} = \vec{0}$ after centering Y. Then we can obtain the predictive distribution $\overrightarrow{y_p}$ by adding the observation noise to $\overrightarrow{f_p}$:

$$y_p[n_p] \sim \mathcal{N}\left(f_p[n_p], \sigma_y \right) \tag{12.37}$$

We can draw the graphical model as Fig. 12.13 to help us better understand the distribution of $\overrightarrow{f_p}$ and $\overrightarrow{y_p}$.

Note that the marginal distribution of f_p at the n_p-th location, or $f_p[n_p]$ is:

[9] This textbook is available online for free: https://www.microsoft.com/en-us/research/people/cmbishop/#!prml-book.

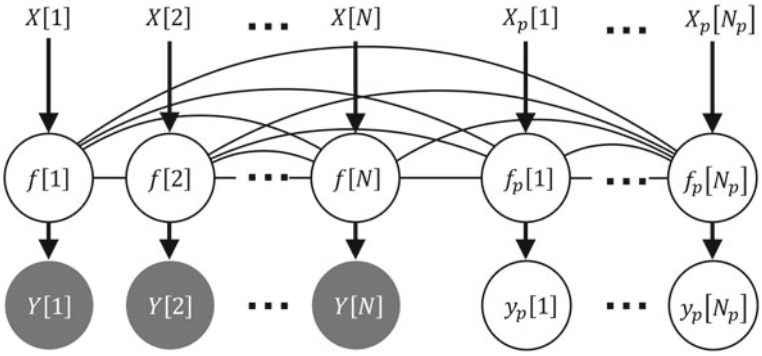


Fig. 12.13 Graphical model of the GP with prediction. The joint distribution is represented as the edges without directions

$$f_p[n_p] \sim \mathcal{N}\left(\mu_p^{(\mathrm{cond})}[n_p], \sigma_p^{(\mathrm{cond})}[n_p]\right) \tag{12.38}$$

$$\sigma_p^{(\mathrm{cond})}[n_p] = \sqrt{K_p^{(\mathrm{cond})}[n_p, n_p]}$$

$$\sigma_p^{(\mathrm{cond})}[n_p] = \sqrt{k_{pp}[n_p, n_p] - \left(\mathbf{k}_p^T\left(\mathbf{K} + \sigma_y^2\mathbf{I}\right)^{-1}\mathbf{k}_p\right)[n_p, n_p]} \tag{12.39}$$

This is because the marginal distribution of a multivariate normal distribution is a normal distribution (see Sect. 6.14). When $\mathbf{k}_{pp}$ in Eq. (12.36) is very large, we can use Eq. (12.38) instead of Eq. (12.34) to simplify the prediction. This simplification corresponds to approximating $\mathbf{K}_p^{(\mathrm{cond})}$ in Eq. (12.36) with a diagonal matrix by replacing the non-diagonal elements with zeros. By doing so, the multivariate normal distribution in Eq. (12.34) can be represented as the multiplication of N_p independent normal distributions. This simplification will be used in Sect. 12.10.

From here, we will implement the prediction of $\overrightarrow{f_p}$ and $\overrightarrow{y_p}$ using Eqs. (12.34) to (12.37). Here, we will use the Gaussian kernel Eq. (12.26) and assume that the observation model follows a normal distribution. The implementation code is based on the previous code **`model12-7c.stan`**:

`model12-7d.stan`

```
functions {
  real gp_lpdf(vector out, array[] real x, vector mu,
               real a, real rho, real s2) {
    int N = size(x);
    matrix[N,N] k_addI = add_diag(gp_exp_quad_cov(x, a, rho), s2);
    return multi_normal_cholesky_lpdf(out|mu, cholesky_decompose
   (k_addI));
  }

  vector gp_pred_rng(array[] real x, array[] real xp,
                     vector mu, vector mup, vector out,
                     real a, real rho, real s2, real sp2) {
```

```
12        int N = size(x);
13        int Np = size(xp);
14        matrix[N, N ] k_addI = add_diag(gp_exp_quad_cov(x, a, rho), s2);
15        matrix[Np,Np] kpp_addI = add_diag(gp_exp_quad_cov(xp, a, rho),sp2);
16        matrix[N, Np] kp = gp_exp_quad_cov(x, xp, a, rho);
17        matrix[Np,N ] coef = kp' / k_addI;
18        vector[Np] mup_cond = mup + coef * (out - mu);
19        matrix[Np,Np] covp_cond = kpp_addI - coef * kp;
20        return multi_normal_rng(mup_cond, covp_cond);
21      }
22    }
23
24    data {
25      int<lower=1> N;
26      int<lower=1> Np;
27      array[N] real X;
28      array[Np] real Xp;
29      vector[N] Mu;
30      vector[Np] Mup;
31      vector[N] Y;
32    }
33
34    parameters {
35      real<lower=0> a;
36      real<lower=0> rho;
37      real<lower=0> s_y;
38    }
39
40    model {
41      a ~ normal(0, 1);
42      rho ~ normal(0.1, 0.1);
43      s_y ~ normal(0, 1);
44      Y ~ gp(X, Mu, a, rho, square(s_y));
45    }
46
47    generated quantities {
48      vector[Np] fp = gp_pred_rng(X, Xp, Mu, Mup, Y, a, rho, square(s_y),1e-8);
49      array[Np] real yp = normal_rng(fp[1:Np], s_y);
50    }
```

Lines 26, 28, and 30: We declare the number of the locations that we want to predict as `Np`, their locations as `Xp`, and the mean vector $\overrightarrow{\mu_p}$ as `Mup`.

Lines 9–21: Using Eqs. (12.34) to (12.36), we define the function that probabilistically generates a curve that follows the GP with $\overrightarrow{\mu_p^{(\mathrm{cond})}}$ and $\mathbf{K}_p^{(\mathrm{cond})}$. This function is used in line 48.

Line 15: `kpp_addI` corresponds to $\mathbf{k}_{pp}$ in Eq. (12.31). In line 48, we pass a small constant `1e-8` and add it to the diagonal elements for stabilization as in **`model12-7a.stan`**. If we do not need to obtain $\overrightarrow{f_p}$, we can generate $\overrightarrow{y_p}$ directly by passing `square(s_y)` instead of `1e-8`.

Line 16: `kp` corresponds to $\mathbf{k}_p$ in Eq. (12.30).

Line 17: `coef` corresponds to $\mathbf{k}_p^T\left(\mathbf{K}+\sigma_y^2\mathbf{I}\right)^{-1}$ in Eqs. (12.35) and (12.36). We store them temporarily to make the computation easier.

Line 18: $\overrightarrow{\mu_p^{(\mathrm{cond})}}$ is computed based on Eq. (12.35) and is assigned to `mup_cond`.

Line 19: $\mathbf{K}_p^{(\mathrm{cond})}$ is computed based on Eq. (12.36) and is assigned to `covp_cond`.

Line 20: This line corresponds to Eq. (12.34), and probabilistically generates a curve that follows the GP with $\overrightarrow{\mu_p^{(\mathrm{cond})}}$ and $\mathbf{K}_p^{(\mathrm{cond})}$.

Line 49: This line corresponds to Eq. (12.37).

Using this model, we obtained the predicted result as shown in Fig. 12.14. Here, the N_p locations X_p were determined by dividing the range [0, 1] at regular intervals ($N_p = 61$).

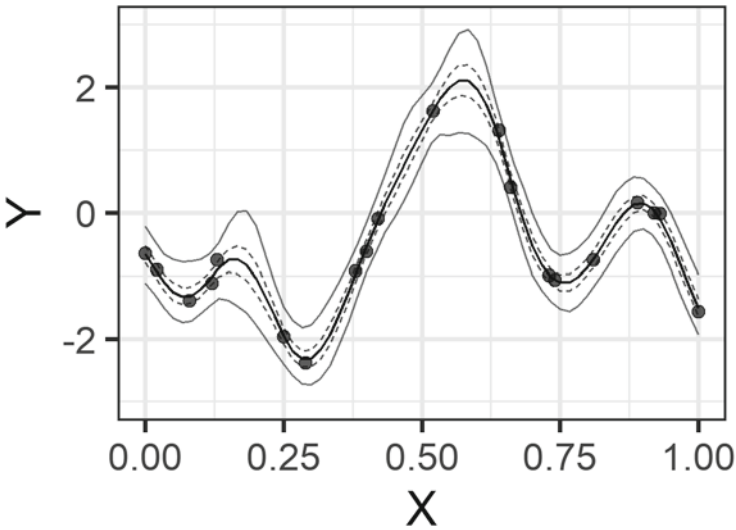


Fig. 12.14 PPC. The solid gray lines are 95% prediction intervals computed from the sampling results of `yp` (which are 4000 curves) for each location. The dotted gray lines are 50% prediction intervals and the black line is the medians. The dots are the observed data

12.7.4 Other Kernel Functions

Besides a Gaussian kernel, there are many other kernel functions. We introduce them briefly here.

- Linear kernel

$$k\left(\vec{x}, \vec{x}'\right) = a\vec{x}^T\vec{x}' \tag{12.40}$$

- Exponential kernel

$$k\left(\vec{x}, \vec{x}'\right) = a\exp\left(-\frac{\left|\vec{x} - \vec{x}'\right|}{\rho}\right) \tag{12.41}$$

- Periodic kernel

$$k\left(\vec{x}, \vec{x}'\right) = a\exp\left(b\cos\left(\frac{\left|\vec{x} - \vec{x}'\right|}{\rho}\right)\right) \tag{12.42}$$

Other than these, there is a Matérn kernel, which has both properties of an exponential kernel and a Gaussian kernel.

Kernel functions can be combined together. For instance, we can construct a new kernel with a linear kernel and a Gaussian kernel as follow:

$$k\left(\vec{x}, \vec{x}'\right) = a_1\vec{x}^T\vec{x}' + a_2\exp\left(-\frac{1}{2\rho^2}\left(\vec{x} - \vec{x}'\right)^2\right) \tag{12.43}$$

Using this kernel, we can do a regression with a curve that is similar to a Gaussian kernel but adding linearity.

We can explain kernel functions as a type of similarity measure. Therefore, if we can define the similarity between character strings in a certain way, then we will be able to construct a GP that takes string $\vec{x}$ as the input. More detail about kernel functions can be found in the presentation materials by Nicolas Durrande.[10]

From next section, we will give examples of modeling using a GP. In order to make it easier to compare with a GMRF, we will reuse the examples we mentioned in Sects. 12.2 and 12.6.

[10] http://gpss.cc/gpss15/talks/KernelDesign.pdf.

12.8 (Example 5) Data on One-Dimensional Location

In this section, we apply a GP to the example in Sect. 12.2. Specifically, we express the spatial structure $\vec{f}$ in Eq. (12.6) by combining the GP Eq. (12.24) and Gaussian kernel Eq. (12.26). Further, we use a Poisson distribution as the observation model Eq. (12.7). We define these equations as Model Formula 12.8.

The implementation of Model Formula 12.8 is as follows. This is based on the code **model12-7b.stan**.

A part of code **model12-8.stan**

```
1    ...
2
3    model {
4       ...
5       Y[1:N] ~ poisson_log(f[1:N]);
6    }
7
8    generated quantities {
9       vector[N] y_mean = exp(f[1:N]);
10    }
```

Line 5: The observation model has changed from a normal distribution to a Poisson distribution. Correspondingly, `Y` is declared as `array[N] int Y` in `data` block.

Line 9: In this problem, we wanted to know the mean number of plants at each location. Therefore, we use `exp(f)` here.

The implementation to create data to pass to `data` block can be as follows:

A part of code **run-model12-8.R**

```
1   Y <- read.csv('input/data-1D.csv')$Y
2   I <- length(Y)
3   Mu <- rep(mean(log(ifelse(Y == 0, 1, Y))), I)
4   X <- (1:I - 1) / (I-1)
5   data <- list(N=I, X=X, mu=Mu, Y=Y)
```

A part of code **run-model12-8.py**

```
1   Y = pandas.read_csv('input/data-1D.csv').Y.values
2   I = len(Y)
3   Mu = np.repeat(np.mean(np.log(Y.clip(1))), I)
4   X = np.arange(I) / (I-1)
5   data = {'N':I, 'X':X, 'Mu':Mu, 'Y':Y}
```

Line 3: Instead of centering `Y`, we took the log of `Y`, calculated the mean of them, replicated the mean `I` times, and assigned it to `Mu`. Here, for a few of `Y` which have value 0, we changed their values to 1 in order to do the log transformation.

Line 4: The values of location `X` are normalized so that they will be in the range [0, 1]. This is an optional step. But we recommend doing so because the priors for

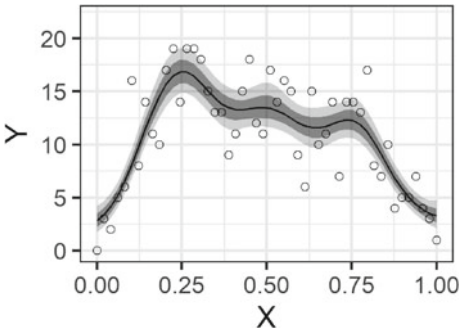


Fig. 12.15 The estimated result using the GP. The legend is the same as Fig. 12.2

the parameters of a GP, a and ρ, will be on the similar scale as in other examples by this normalization.

The estimation result is shown in Fig. 12.15. This result does not look very different from the estimation results from the right plot on Fig. 12.2. By using the GP, we showed that we can estimate the smooth spatial structure with the continuous locations.

12.9 (Example 6) Data on Two-Dimensional Grid

In this section, we look at the example from Sect. 12.6. We represent the spatial structure of $\vec{f}$ by combining the GP Eq. (12.24) and Gaussian kernel Eq. (12.27), where each location corresponds to a D-dimensional vector. As for the rest of the model formula, we use Eqs. (12.18), (12.20), and (12.21). We define them together as Model Formula 12.9.

We show the implementation of Model Eq. 12.9 below. This implementation is based on the code **`model12-7c.stan`**. In Sect. 12.6, we used data from 384-hole plates (24 columns and 16 rows) and there were 96 treatments that are assigned randomly. For that data, we used `matrix` types, such as the observation values `Y` and the treatment index `ji2t`. In contrast, when we use a GP, it is convenient to rearrange such `matrix` types into `vector` types of length N (= 384) in column-major order.

`model12-9.stan`

```
functions {
  real gp_lpdf(vector out, array[] vector x, vector mu,
                 real a, real rho, real s2) {
    (same as gp_lpdf in model12-7d.stan)
  }

  vector gp_pred_rng(array[] vector x, array[] vector xp,
                     vector mu, vector mup, vector out,
                     real a, real rho, real s2, real sp2) {
    (same as gp_pred_pars in model12-7d.stan)
  }
```

```
12   }
13
14   data {
15     int N;
16     array[N] vector[2] X;
17     vector[N] Mu;
18     vector[N] Y;
19     int T;
20     array[N] int<lower=1, upper=T> n2t;
21   }
22
23   parameters {
24     vector[T] beta;
25     real<lower=0> s_beta;
26     real<lower=0> a;
27     real<lower=0> rho;
28     real<lower=0> s_y;
29   }
30
31   model {
32     beta ~ normal(0, s_beta);
33     a ~ normal(0, 1);
34     rho ~ normal(0.1, 0.1);
35     s_y ~ normal(0, 1);
36     Y ~ gp(X, Mu + beta[n2t], a, rho, square(s_y));
37   }
38
39   generated quantities {
40     vector[N] fp = gp_pred_rng(X, X, Mu + beta[n2t], Mu,
41                                Y, a, rho, square(s_y), 1e-8);
42   }
```

Line 16: `X` stores the two-dimensional coordinates of each location on the N-hole plate.

Lines 18 and 20: Each element in `Y` and `n2t` corresponds to an element in `X`.

Lines 2 and 7: In line 2, the type of `x` is converted from `array[] real` type (an array of `real` type) to `array[] vector` type (an array of `vector` type). In line 7, the type of `xp` and `x` is converted in the same way. This is because the Gaussian kernel changes from Eqs. (12.26) to (12.27) when the dimension of the location (coordinates) changes from one to two. If we consider the plate as a two-dimensional hyperplane, using this kernel is equivalent to assume that when the Euclidian distance between two holes is closer, these two outputs would be similar.

Line 36: `Y` follows a GP. Note that the mean vector of the GP is equal to `Mu` plus the treatment effect `beta[n2t]`.

Lines 40–41: In this section, we want to subtract all the treatment effects and know each location effect $\vec{f}$. Therefore, we use function `gp_pred_rng` to predict the value at each location on the plate.

The R and Python code to run **`model12-9.stan`** can be as follows:

run-model12-9.R

```
1   library(cmdstanr)
2
3   T <- 96
4   d <- read.csv('input/data-2Dmesh.csv', header=FALSE)
5   I <- nrow(d)
6   J <- ncol(d)
7   N <- I*J
8
9   X <- expand.grid(Row=(1:I - 1)/(I-1), Column=(1:J - 1)/(J-1))
10   Y_ori <- unlist(d)
11   Y <- Y_ori - mean(Y_ori)
12   d_trt <- read.csv('input/data-2Dmesh-design.csv', header=FALSE)
13   n2t <- unlist(d_trt)
14   data <- list(N=N, X=X, Mu=rep(0, N), Y=Y, T=T, n2t=n2t)
15
16   model <- cmdstan_model('model/model12-9.stan')
17   fit_MAP <- model$optimize(data=data, seed=123)
```

run-model12-9.py

```
1   import pandas
2   import numpy as np
3   import cmdstanpy
4
5   T = 96
6   d = pandas.read_csv('input/data-2Dmesh.csv', header=None).values
7   I, J = np.shape(d)
8   N = I*J
9   X = np.array([(i/(I-1), j/(J-1)) for j in range(J) for i in range(I)])
10   Y_ori = d.T.reshape(N)
11   Y = Y_ori - np.mean(Y_ori)
12   d_trt = pandas.read_csv('input/data-2Dmesh-design.csv', header = None)
13   n2t = d_trt.values.T.reshape(N)
14   data = {'N':N, 'X':X, 'Mu': np.zeros(N), 'Y':Y, 'T':T, 'n2t':n2t}.
15
16   model = cmdstanpy.CmdStanModel(stan_file='model/model12-9.stan')
17   fit_MAP = model.optimize(data=data, seed=123, output_dir='output')
```

Line 9: `X` stores the normalized two-dimensional coordinates for each hole on the N-hole plate. By passing the grids of the columns and rows to the function `expand.grid` in R (list comprehension in Python), the two-dimensional coordinates of each hole will be ordered in column-major.

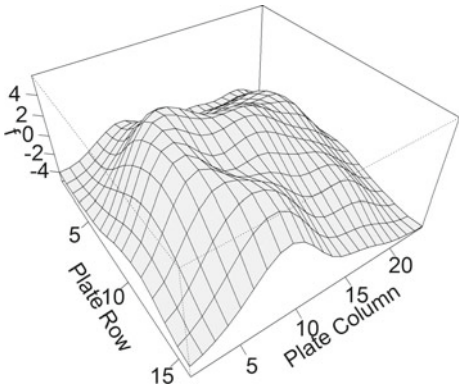


Fig. 12.16 Estimated result using the GP. The legend is the same as Fig. 12.10 (left)

Line 11: This line does centering the data `Y`.

Line 17: Considering the time consumption of MCMC, we use the point estimation that is introduced in Sect. 3.2.

Figure 12.16 is the visualization of the mean value of each hole location (`pars[,1]`). We can see that the model formula was able to estimate the smooth spatial structure for the two-dimensional locations.

12.10 Inducing Variable Method

In this section, we discuss a method that is used to speed up the estimation of a GP. One drawback of the GP is its computational complexity. Especially, if we calculate $\left(\mathbf{K} + \sigma_y^2 \mathbf{I}\right)^{-1}$ in Eqs. (12.35) and (12.36), and let the number of locations be N, then the computational cost will be the order of $\mathcal{O}\left(N^3\right)$. When we have $N = 5000$, it easily reaches the computer memory limit, and thus is impossible to do estimation in practice. Directly working with a kernel matrix with size $N \times N$ is very hard when we have a large N.

Hence, there have been multiple tricks that are proposed to reduce the computational burden in a GP. One of these methods is called an *inducing variable method*. The main idea of the inducing variable method is to take a number of locations N_i that is much smaller than the actual N, and compute the kernel matrix only using these N_i locations, which is called the inducing inputs. The location of an inducing input is denoted as X_i, and the corresponding latent output at that location is denoted as f_i, which is a parameter. Then, by predicting $\vec{Y}$ given $\overrightarrow{f_i}$ in the same manner in Sect. 12.7.3, we derive the conditional distribution of $\vec{Y}$ given $\overrightarrow{f_i}$. We use the simplified prediction Eqs. (12.38) and (12.39) because $\mathbf{k}_{pp}$ in Eq. (12.36) is a large matrix with size $N \times N$. In other words, by using the small dense kernel matrix as the latent variable $\overrightarrow{f_i}$, we build a large sparse (diagonal) kernel matrix for $\vec{Y}$. Therefore, the

inducing variable method is also called a sparse GP. More detail about the inducing variable method is over the scope of this book, but more information can be found in Quiñonero-Candela and Rasmussen (2005) and Snelson and Ghahramani (2005).

We can visualize the inducing variable method using a graphical model as in Fig. 12.17. Note that there are edges between the nodes of inducing inputs $\vec{f}_i$, but there are no edges between the nodes of $\vec{f}$. This is because we use the covariance matrix of $\vec{f}$ as the diagonal matrix.

Next, we implement it using Stan to get a better understanding of this method. Let us consider the situation where we have one-dimensional locations X and corresponding values Y, similar to Data File 12.7 and Fig. 12.11. We assume the total number of locations is $N = 2000$.

model12-10.stan

```
functions {
  (gp function defined as in model12-7b.stan)
  (gp_pred_rng defined as in model12-7d.stan)

  real gp_pred_approx_lpdf(vector outp, array[] real x, array[] realxp,
                           vector mu, vector mup, vector out,
                           real a, real rho, real s2, real sp2) {
```

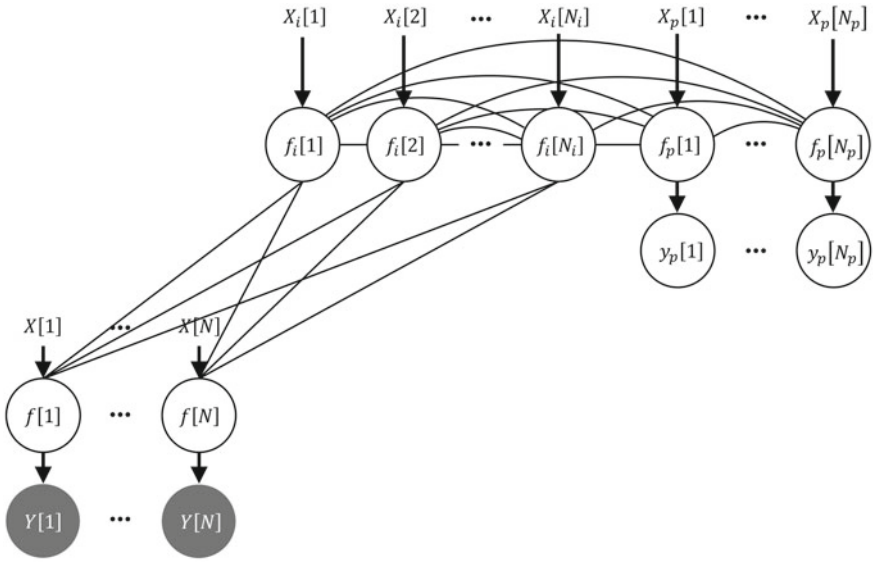


Fig. 12.17 Graphical model of inducing variable method

```
    int N = size(x);
    int Np = size(xp);
    matrix[N, N ] k_addI = add_diag(gp_exp_quad_cov(x, a, rho), s2);
    vector[Np] kpp_diag = rep_vector(square(a) + sp2, Np);
    matrix[N, Np] kp = gp_exp_quad_cov(x, xp, a, rho);
    matrix[Np,N ] coef = kp' / k_addI;
    vector[Np] mup_cond = mup + coef * (out - mu);
    vector[Np] sp_cond=sqrt(kpp_diag-rows_dot_product(coef, kp'));
    return normal_lpdf(outp | mup_cond, sp_cond);
  }
}

data {
  int<lower=1> Ni;
  int<lower=1> N;
  int<lower=1> Np;
  array[Ni] real Xi;
  array[N] real X;
  array[Np] real Xp;
  vector[Ni] Mui;
  vector[N] Mu;
  vector[Np] Mup;
  vector[N] Y;
}

parameters {
  vector[Ni] eta;
  real<lower=0> a;
  real<lower=0> rho;
  real<lower=0> s_y;
}

transformed parameters {
  vector[Ni] fi = gp(Xi, Mui, eta, a, rho, 1e-8);
}

model {
  eta ~ normal(0, 1);
  a ~ normal(0, 1);
  rho ~ normal(0.1, 0.1);
  s_y ~ normal(0, 1);
  Y[1:N]~gp_pred_approx(Xi,X,Mui,Mu,fi,a,rho,1e-8,square(s_y));
}

generated quantities {
  vector[Np] fp=gp_pred_rng(Xi,Xp,Mui,Mup,fi,a,rho,1e-8,1e-8);
  array[Np] real yp = normal_rng(fp[1:Np], s_y);
}
```

Lines 5–17: Using Eqs. (12.38), (12.35), and (12.39), we define a sparse GP with $\mu_p^{(\mathrm{cond})}[n_p]$ and $\sigma_p^{(\mathrm{cond})}[n_p]$. This function is used in line 49.

Line 11: `kpp_diag` is a `vector` that has the diagonal elements of $\mathbf{k}_{pp}$. We can write $k_{pp}[n_p, n_p] = a^2$ because we use the Gaussian kernel Eq. (12.26).

Line 14: $\mu_p[n_p]$ is computed based on Eq. (12.35) and is assigned to `mup_cond`.

Line 15: $\sigma_p[n_p]$ is computed based on Eq. (12.39) and is assigned to `sp_cond`. Because $\left(\mathbf{k}_p^T\left(\mathbf{K}+\sigma_y^2\mathbf{I}\right)^{-1}\mathbf{k}_p\right)[n_p, n_p]$ is equivalent to the inner product between the n_p-th row of $\mathbf{k}_p^T\left(\mathbf{K}+\sigma_y^2\mathbf{I}\right)^{-1}$ and n_p-th row of $\mathbf{k}_p^T$, the diagonal elements of $\mathbf{k}_p^T\left(\mathbf{K}+\sigma_y^2\mathbf{I}\right)^{-1}\mathbf{k}_p$ are calculated using the function `rows_dot_product`. This is a convenient function in Stan, and it returns a `vector` of values that are inner products between the rows of two matrices.

Line 16: This line corresponds to Eq. (12.38). $Y[n]$ follows a normal distribution with the mean and SD from lines 14–15.

Lines 21 and 24: `Ni` is the total number of inducing inputs and `Xi` is the locations of inducing inputs. There are multiple ways to determine the value of X_i. Here, we determined X_i by dividing the range [0, 1] into regular intervals. Alternatively, we can choose from those locations from where the observations are made, or find the good locations by optimization.

Line 41: This GP only uses the inducing inputs. The outputs from the GP, `fi`, is defined in `transformed parameters` block.

Line 49: When `fi` is given, outputs at the known locations X can be predicted and the distribution of `Y` is determined. This prediction uses the diagonal matrix as the approximation of the original covariance matrix, therefore we can avoid the computationally expensive matrix calculation.

Line 53: The predicted outputs at locations X_p are calculated based on the kernel matrix of the inducing inputs.

Here, we used $N_i = 15$ of inducing inputs and predicted the outputs at $N_p = 61$ locations, and estimated the parameters using the point estimation. The R and Python code are omitted. A part of the result is shown on Fig. 12.18. We confirmed that this method is fast and gives enough accuracy.

12.11 Supplementary Information and Exercises

For spatial data analysis, there are numerous methods besides the GMRF and the GP we introduced here. To know more about this topic, (Bivand et al., 2008) and (Cressie, 2015) are a good place to start.

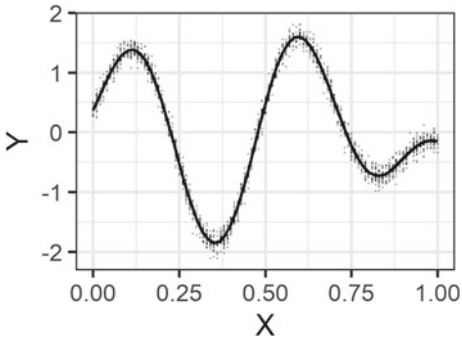


Fig. 12.18 Estimated result using inducing variable method. The black line represents the medians of estimated $\overrightarrow{f_p}$ and the small gray dots are the observed values

In Sect. 12.9, we computed a kernel matrix of size 384×384. In the case of using the Gaussian kernel for twoss-dimensional grid data in Eq. (12.27), its kernel matrix can be represented as a Kronecker product of a 16×16 kernel matrix and another 24×24 kernel matrix. Therefore, the computation can be done much faster.

It seems that a GP has been one of the main topics in Stan development team recently, and novel functions for facilitating efficient computing are being brought up in the discussion quite frequently. We suggest looking up Stan reference (see chapter "Gaussian Processes") to get the most updated information.

12.11.1 Exercises

(1) Using the kernel functions of Eqs. (12.40) to (12.43), generate the curves from the GP with some parameter settings. Plot your result that corresponds to Fig. 12.12.
(2) Use the GP for the trend component in `model11-2.stan`, and estimate the prediction intervals at Time = 21 to 23.
(3) Use the GP for the trend component in (3) in Sect. 11.9.1.
(4) Use the GP for the trend component in `model11-10.stan`, and estimate the Bayesian confidence intervals of $switch[t]$.

References

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.
Bivand, R. S., Pebesma, E. J., Gómez-Rubio, V., & Pebesma, E. J. (2008). Applied spatial data analysis with R (Vol. 747248717, pp. 237–268). New York: Springer.
Cressie, N. (2015). *Statistics for spatial data*. Wiley.
Hávard, R., & Leonhard, H. (2005). *Gaussian Markov random fields: Theory and applications*. CRC Press.

Quiñonero-Candela, J., & Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research, 6*, 1939–1959.

Rasmussen, C. E., & Williams, C. K. (2006). *Gaussian processes for machine learning*. The MIT Press.

Snelson, E., & Ghahramani, Z. (2005). Sparse Gaussian processes using pseudo-inputs. *Advances in Neural Information Processing Systems*, 18.

Chapter 13
Usages of MCMC Samples from Posterior and Predictive Distributions

We have discussed various statistical models and their implementation so far. However, the ways we used the MCMC samples from the posterior distributions and predictive distributions were only kept on very basic levels, such as computing the intervals and visualizations. In this chapter, we will introduce more advanced usages of the MCMC sample. They would be helpful in practice because it is very common to encounter the situation where we need to extract more information from the MCMC sample.

In particular, we will introduce these contents in each of the following sections:

- In Sect. 13.1, we will introduce how to do simulation using MCMC sample.
- In Sect. 13.2, we will introduce an idea of minimizing the expected loss during the decision-making process. We compute the expected loss by integrating loss function using the posterior distribution.
- In Sect. 13.3, we will introduce Thompson sampling and Bayesian optimization. We will work with a problem where we want to maximize a cumulative reward while we want to explore where to get the data from.

13.1 Simulation Based Sample Size Calculation

In this section, we will discuss how to decide the number of data points, which is necessary for an experimental design. In general, the more data points we have, the more trustable the results would be (the confidence intervals of the parameters will be smaller). However, considering the cost of collecting a large amount of data, the question would be, how many data points are needed to keep the satisfactory level of certainty on the result. This process of calculating the required number of data points is called *sample size calculation.*

For example, to get a new drug approval from the government, we need to conduct clinical trials on patients and make a statistically significant output from a Phase 3 trial. The number of patients that is subject to a clinical trial is usually set to be the

© Springer Nature Singapore Pte Ltd. 2022

K. Matsuura, *Bayesian Statistical Modeling with Stan, R, and Python*,
https://doi.org/10.1007/978-981-19-4755-1_13

minimum number of patients that makes the statistical power greater than 0.8 or 0.9. In particular, this means that if the new drug has the assumed effect, we can detect the statistical significance from the trial result with 0.8 or 0.9 probability.

When a simple model is used to detect the efficacy of a drug, it is relatively easy to analytically calculate the sample size. However, with the models becoming more complex in recent study design, using simulation to calculate the sample size is becoming a more common approach.

Let us look at an example. In order to measure the efficacy of a new drug, we conducted a Phase 2 clinical trial and randomly assigned 20 people to a group of an old drug and a group of the new drug. Assume that the old drug successfully cured 10 out of the 20 patients, and the new drug successfully cured 14 out of 20 patients. Based on this result, we want to calculate the sample size so that a Phase 3 clinical trial has at least 0.8 statistical power. Assume that each group has the same number of patients and we want to test whether the two groups differ significantly (the significance level is set to be 0.05 for the two-sided test). How should we calculate the sample size for this trial?

For the clarity, we use θ_1 to denote the probability that the old drug cures the disease, and θ_2 to denote the probability that the new drug cures the disease. We use D to denote the overall result from the Phase 2 trial. In Phase 3 trial setting, we use n, y_1, and y_2 to denote the number of patients in each group, the number of patients cured by the old drug, and the number of patients cured by the new drug, respectively.

Based on the traditional way of calculating the sample size, we estimate the parameters θ_1 and θ_2 by maximum likelihood estimation (point estimation), and obtained $\widehat{\theta}_1 = 10/20 = 0.5$ and $\widehat{\theta}_2 = 14/20 = 0.7$. Next, we can obtain the number of the patients in each group is $n = 93$ for a Phase 3 trial,[1] with the assumption that these point-estimated values are the true values of the parameters. If we use this approach, however, when 100 out of 200 patients were cured by the old drug and 140 out of 200 patients were cured by the new drug in the Phase 2 trial, it still gives the same number, $n = 93$. In other words, in the traditional sample size calculation, the uncertainty from the result of Phase 2 is not being considered, and the calculated sample size overfits to this particular result. The reason for this overfitting is that predictive distributions of y_1 and y_2 are generated by plugging the maximum likelihood estimate of the parameters into the model. These predictive distributions are written mathematically as[2]:

[1] The number of patients in each group is obtained by rounding the value calculated from the following equation (details are omitted):

$$n = \left(\frac{Z_{\alpha/2}\sqrt{(\theta_1+\theta_2)(1-(\theta_1+\theta_2)/2)} + Z_\beta\sqrt{\theta_1(1-\theta_1)+\theta_2(1-\theta_2)}}{\theta_2-\theta_1} \right)^2$$

where $Z_{\alpha/2} = 1.96$ if the significance level of the test is 0.05 (two-sided) and $Z_\beta = 0.84$ if the power is 0.8. Refer Fleiss et al. (1980) for more detail.

[2] Strictly speaking, the left-hand side is $p(y|n, D)$, but the fixed value n is omitted for simplicity.

$$p(y_1|D) = \text{Binomial}\left(y_1|n, \hat{\theta}_1\right)$$
$$p(y_2|D) = \text{Binomial}\left(y_2|n, \hat{\theta}_2\right)$$

On the other hand, under the Bayesian framework, the predictive distributions consider the uncertainty in the parameter estimates. The difference is that the predictive distribution is the average of the probability distributions for generating data over the posterior distribution of parameters (see Sect. 2.4). That is,

$$p(y_1|D) = \int \text{Binomial}(y_1|n, \theta_1)p(\theta_1|D)d\theta_1$$
$$p(y_2|D) = \int \text{Binomial}(y_2|n, \theta_2)p(\theta_2|D)d\theta_2$$

where $p(\theta_1|D)$ and $p(\theta_2|D)$ are the posterior distributions of parameters. The statistical power that computed from these predictive distributions under Bayesian framework is called *assurance*, to distinguish it from the power used in the frequentist framework (O'Hagan et al., 2005).

From now on, we assume that we already have obtained the draws from the posterior distribution for θ_1 and θ_2 using the Phase 2 data with Stan (we omit **model13-1.stan**). Consider a Phase 3 trial has sample size $n = 93$, and let us compare the power and assurance. The implementation using R and Python are shown as follows:

A part of code **run-model13-1.R**

```
(...Estimation of theta1 and theta2 by Stan...)
# N:20, Y1:10, Y2:14, n:93, N_sim:40000, N_ms:40000

calc_pval <- function(y1, y2) {
  prop.test(c(y1, y2), c(n, n), correct=FALSE)$p.value
}

y1_MLE <- rbinom(n=N_sim, size=n, prob=Y1/N)
y2_MLE <- rbinom(n=N_sim, size=n, prob=Y2/N)
y1_Bay <- rbinom(n=N_ms, size=n, prob=d_ms$theta1)
y2_Bay <- rbinom(n=N_ms, size=n, prob=d_ms$theta2)
pvals_MLE <- mapply(calc_pval, y1_MLE, y2_MLE)
pvals_Bay <- mapply(calc_pval, y1_Bay, y2_Bay)
power <- mean(pvals_MLE < pval_cut)
assurance <- mean(pvals_Bay < pval_cut)
```

A part of code **run-model13-1.py**

```
 1  (...Estimation of theta1 and theta2 by Stan...)
 2  # N:20, Y1:10, Y2:14, n:93, N_sim:40000, N_ms:40000
 3
 4  def calc_pvals(y1, y2):
 5   (...Calculate p-values using RPy2...)
 6   return pvals
 7
 8  y1_MLE = np.random.binomial(n, Y1/N, N_sim)
 9  y2_MLE = np.random.binomial(n, Y2/N, N_sim)
10  y1_Bay = np.random.binomial(n, d_ms.theta1.values, N_ms)
11  y2_Bay = np.random.binomial(n, d_ms.theta2.values, N_ms)
12  pvals_MLE = calc_pvals(y1_MLE, y2_MLE)
13  pvals_Bay = calc_pvals(y1_Bay, y2_Bay)
14  power = np.mean(pvals_MLE < pval_cut)
15  assurance = np.mean(pvals_Bay < pval_cut)
```

Lines 4–6: We define a function that computes a p-value based on the statistical test for the difference between proportions in two group in a Phase 3 trial.

Lines 8–11: The outcomes from Phase 3 trials are generated by simulation. For an easier comparison, we are computing both power and assurance from the simulation.

Lines 8–9: For the power, `N_sim` outcomes are generated probabilistically from the predictive distribution, which is constructed using the maximum likelihood estimates obtained from the Phase 2 result. Note that in this case, we use the same estimates (i.e., `Y1/N` and `Y2/N`) in each simulation. The larger the `N_sim` is, the more accurate power and assurance would be. Here it is set to be 40,000.

Lines 10–11: For the assurance, `N_ms` outcomes are generated from the predictive distribution with the posterior distribution from the Phase 2 result. Note that this time, the draws from the posterior distributions are used and thus they differ in each simulation. Here `N_ms` represents the MCMC sample size (number of draws), and is set to be 40,000.

Lines 12–13: The statistical test is conducted and the p-value is computed for each simulation data.

Lines 14–15: From the result of the simulations, the ratio of getting the statistical significance is calculated using the maximum likelihood estimation (power) and the posterior distribution (assurance).

The result from this analysis is power = 0.80 and assurance = 0.63. This way of calculating the assurance is also applicable when the model is more complex. With Bayesian modeling, we can broaden our choice of doing the simulation to determine the sample size by constructing the predictive distribution which incorporates the prior knowledge.

13.2 Bayesian Decision Theory

In this section, we introduce *Bayesian decision theory* (DeGroot, 1970). In the decision theory, we try to select the action that minimizes the loss from multiple choices with certain conditions. In the Bayesian decision theory, we select the action that minimizes the *expected loss* ($\mathbb{E}[L]$), which is calculated using the posterior distribution of the parameters and the posterior predictive distribution of data. $\mathbb{E}[L]$ can be expressed as follows:

$$\mathbb{E}[L] = \iint L(\theta, y, a)p(\theta, y|Y)d\theta dy \tag{13.1}$$

where $L(\theta, y, a)$ is the loss function that is determined by parameters θ, data y, and action a. The distribution $p(\theta, y|Y)$ is the joint distribution of θ and y estimated after obtaining the data Y. If we need to compute $\mathbb{E}[L]$ before obtaining data Y, we can just use a prior joint distribution.

From now on, we use the quarterly sales data (for four seasons) that we used in Sect. 11.2.3 as the example data. Let a random variable y be the number of purchases at the one-step-ahead time point of current time point. Now, assume that we need to make all products in advance, based on the predictive distribution of y. Let a be the number of the products that we plan to make, and let us consider what value of a would minimize the expected loss. Here we assume that we have already estimated the predictive distribution $p(y|Y)$ from the data (see Exercise 11.9.1 (2)). We assume the following loss function:

$$L(y, a) = \begin{cases} 2(y - a) & \text{if } a < y \\ 1 - \exp\big[-(a - y)\big] & \text{if } a \geq y \end{cases} \tag{13.2}$$

Equation (13.2) is shown in Fig. 13.1. The intuition of Eq. (13.2) is that when the number of the prepared products are not enough ($a < y$), the loss is $2(y - a)$, which is proportional to their difference. When there are extra products made in advance ($a \geq y$), then we use another function which represents the disposal cost ($1 - \exp\big[-(a - y)\big]$), which saturates when the difference is large enough.

Let us first compute the expected loss. For this case, the loss is only dependent on y and is independent of θ. Therefore, we can marginalize θ from the expected loss function in Eq. (13.1), and rewrite it as follows:

$$\begin{aligned} \mathbb{E}[L] &= \iint L(y, a)p(\theta, y|Y)d\theta dy \\ &= \int L(y, a)\left[\int p(\theta, y|Y)d\theta\right]dy \\ &= \int L(y, a)p(y|Y)dy \end{aligned} \tag{13.3}$$

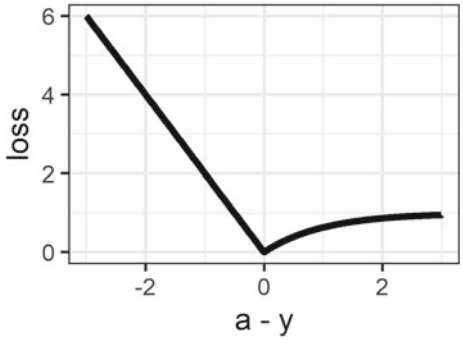


Fig. 13.1 Loss function $L(y, a)$

From this equation, we realize that the expected loss is the expectation of the loss $L(y, a)$ weighted with the predictive distribution $p(y|Y)$.

Next, we will find the a^* that minimizes the expected loss. For this case, we can easily do this by using `optim` function in R or `scipy.optimize.brent` function in Python. The implementation using R and Python are shown as follows:

A part of code **minimize-expected-loss.R**

```
1   (...Estimation by Stan...)
2   y <- fit$draws(format='df')$`yp[1]`
3
4   loss_function <- function(a) ifelse(a < y, 2*(y-a), 1-exp(-(a-y)))
5   expected_loss <- function(a) mean(loss_function(a))
6
7   a_best <- optim(median(y), expected_loss, method='Brent',
8    lower=5, upper=50)$par
```

A part of code **minimize-expected-loss.py**

```
 1 (...Estimation by Stan...)
 2 y = fit.draws_pd(['yp'])['yp[1]']
 3
 4 def loss_function(a):
 5   return(np.where(a < y, 2*(y-a), 1-np.exp(-(a-y))))
 6
 7 def expected_loss(a):
 8   return(np.mean(loss_function(a)))
 9
10 a_best = optimize.brent(expected_loss, brack=(5, 50))
```

Line 2: The draws from $p(y|Y)$ is assigned to `y`.

Line 4 in R (lines 4–5 in Python): The loss function $L(y, a)$ is defined.

Line 5 in R (lines 7–8 in Python): The expected loss function $\mathbb{E}[L]$ is defined. The draws from $p(y|Y)$ are passed to $L(y, a)$, and the average over them is calculated by `mean` function. This is equivalent to multiplying the loss function with $p(y|Y)$ and taking the integral.

Lines 7–8 in R (line 10 in Python): `expected_loss` is the function to be minimized. The lower and the upper limit of the searching range of `a` are 5 and 50. `a_best` stores a^*, and it minimizes the expected loss.

Lastly, by plotting $p(y|Y)$ with the obtained a^* together on the same plot, we get Fig. 13.2. Equation (13.2) gives a larger loss when the number of the prepared products are less than how many are actually needed, therefore the result suggests it is better to produce slightly more products than the mean of the predictive distribution. It is usually not very clear how to quantitatively determine what this "slightly more" is. However, we can solve such a question more straightforwardly by following the framework in the Bayesian decision theory.

Column: The relationship between the loss function and the summary statistics of the posterior distribution

In Sect. 13.2, we looked at an example where we considered the loss function $L(y, a)$, which is not dependent on θ. However, it is possible that sometimes we need to consider a loss function $L(\theta, a)$ that depends on θ. In such a situation, the expected loss is:

$$\mathbb{E}[L] = \iint L(\theta, a)p(\theta, y|Y)d\theta dy = \int L(\theta, a)p(\theta|Y)d\theta$$

As we will explain later, when this loss function $L(\theta, a)$ is in a relatively simple form, we can analytically solve for a^* that minimizes the expected loss.

Here, as an example of a simple loss function, we consider the a^* when using quadratic loss $(a - \theta)^2$ or absolute loss $|a - \theta|$. When it is the quadratic loss, a^* is equivalent to the posterior mean $\mathbb{E}[\theta]$ that can be calculated from $p(\theta|Y)$. The proof is as follows:

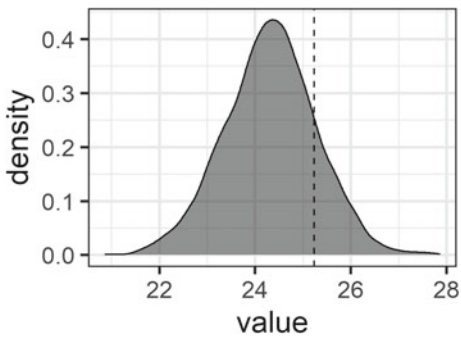


Fig. 13.2 The distribution of $p(y|Y)$ and a^* (dotted line)

$$\mathbb{E}[L] = \int (a-\theta)^2 p(\theta|Y)d\theta = \int \left(a^2 - 2a\theta + \theta^2\right)p(\theta|Y)d\theta$$
$$= a^2 - 2a\mathbb{E}[\theta] + \mathbb{E}\left[\theta^2\right]$$

By taking its partial derivative with regard to a, we can obtain $a^* = \mathbb{E}[\theta]$, which gives the minimum value of $\mathbb{E}[L]$. On the other hand, when we use the absolute loss, a^* is the posterior median. We do not explore further, but more details about loss functions can be found in Wikipedia.[3]

13.3 Thompson Sampling and Bayesian Optimization

In this section, we introduce the problem of repeating the following two steps: (1) Determine the location from which data should be obtained based on the estimation result. (2) Estimate again after obtaining the data. The objective is to select the location so that the cumulative reward is maximized. The case where the locations are discrete and independent is treated in Sect. 13.3.1, and the case where the locations are continuous and a model can be assumed is treated in Sect. 13.3.2.

13.3.1 Thompson Sampling

In this subsection, we consider the problem of deciding which spot is the best place to go fishing. There are five fishing spots, A, B, C, D, E, and we can assume they are independent. In one action, we go to one of them to fish. Assume that the total weight of fish caught in one action follows a normal distribution. Let's assume that the mean μ of the normal distribution differs among fishing spots and the SD σ is the same among fishing spots. In the first 10 actions, we go fishing twice in each of the five spots. In the next 90 actions, we determine one spot for each action according to the results of fishing. In a total of 100 actions, we want to decide the fishing spot that will give us the highest total weight of fish for the 100 actions. How can we decide where to fish?

One very practical solution to such a problem is using *Thompson sampling*. We first estimate the parameters from the accumulated data. For the next fishing, we generate a draw (a vector) from the joint posterior distribution of the means of the normal distributions (see Sect. 4.2.6), and the draw contains the five expected weights of each spot in the next fishing. We then select the spot that has the highest expectation based on the elements of the draw. Based on the result of the next fishing, we can update our model and further estimate the parameters by repeating this process.

[3] https://en.wikipedia.org/wiki/Loss_function.

Next, we will show the basic idea of how to implement Thompson sampling. From now on, we assume that we already have obtained the posterior distribution for μ of each spot using the results of fishing with Stan (we omit Model Formula 13.2 and **`model13-2.stan`**). The implementation using R and Python are shown below:

run-model13-2.R

```
library(cmdstanr)
set.seed(1234)

K <- 5
mu_true <- c(10.0, 11.5, 10.4, 12.8, 9.7)
sd_true <- 3.5
T_ini <- 10
T <- 90

fetch_y <- function(len, k) {
  return(rnorm(len, mean=mu_true[k], sd=sd_true))
}

k_cum <- rep(1:K, len=T_ini)
y_cum <- fetch_y(T_ini, k_cum)
model <- cmdstan_model('model/model13-2.stan')

for (t in 1:T) {
  data <- list(N=length(y_cum), K=K, n2k=k_cum, Y=y_cum)
  mess <- capture.output(
    fit <- model$sample(data=data, seed=123, parallel_chains=4,
                refresh=0, show_messages=FALSE))
  mu_ms <- fit$draws('mu', format='matrix')
  N_ms <- nrow(mu_ms)
  rand <- sample.int(n=N_ms, size=1)
  k_sel <- which.max(mu_ms[rand,])
  y_sel <- fetch_y(1, k_sel)
  k_cum <- c(k_cum, k_sel)
  y_cum <- c(y_cum, y_sel)
}
```

run-model13-2.py

```
import cmdstanpy
import numpy as np
import logging
from cmdstanpy.utils import get_logger
get_logger().setLevel(logging.ERROR)
np.random.seed(123)
K = 5
mu_true = np.array([10.0, 11.5, 10.4, 12.8, 9.7])
sd_true = 3.5
T_ini = 10
T = 90
```

```
12
13   def fetch_y(k, size):
14      return(np.random.normal(mu_true[k], sd_true, size))
15
16   k_cum = np.resize(np.arange(K), T_ini)
17   y_cum = fetch_y(k_cum, T_ini)
18   model = cmdstanpy.CmdStanModel(stan_file='model/model13-2.stan')
19   for _ in range(T):
20      data = {'N':len(y_cum), 'K':K, 'n2k':k_cum+1, 'Y':y_cum}
21      fit = model.sample(data=data, seed=123, parallel_chains=4,
22                 show_progress=False)
23      mu_ms = fit.draws_pd(['mu'])
24      N_ms = len(mu_ms)
25      rand = np.random.randint(0, N_ms)
26      k_sel = np.argmax(mu_ms.iloc[rand])
27      y_sel = fetch_y(k_sel, 1)
28      k_cum = np.append(k_cum, k_sel)
29      y_cum = np.append(y_cum, y_sel)
```

Line 4 in R (line 7 in Python): `K` is the number of spots.

Lines 5–6 and 10–12 in R (lines 8–9 and 13–14 in Python): These are the true settings of this simulation. The function `fetch_y` fetches the data of the result of fishing in the selected spot.

Line 20 in R (lines 3–5 and 22 in Python): To avoid the console output from the sampling function, we used `capture.output` function in R and changed the logging setting in Python.

Lines 21–22: The posterior distributions are estimated from all the cumulative data.

Lines 25–26: The fishing spot to be selected is determined. These lines correspond to Thompson sampling.

Line 25: For the next fishing, we generate the index of a draw.

Line 26: `mu_ms[rand,]` in R (`mu_ms.iloc[rand]` in Python) is a vector of length `K`, and a draw from the joint posterior distribution of the next fishing results in the `K` spots. From the `K` spots, the one with the largest value is selected.

Line 27: The next fishing result is fetched.

Lines 28–29: The data of selected spot and result is accumulated.[4] Note that for the estimation of the next fishing, all the data obtained before is used.

Our original purpose in this section was to maximize the total weights of fish in the 100 actions. Such a problem is called *cumulative reward maximization* problem, and Thompson sampling is practically known to perform well in such problem settings. However, in this example, Thompson sampling tends to select only some certain spots, and this is one of the limitations of this method. Therefore, Thompson sampling is not suitable if our purpose is to accurately judge which spot is best. Such a problem

[4] `_cum` means cumulative data.

is called *best arm identification*,[5] and some alternative algorithms have been developed. To obtain good results in best arm identification, all the spots need to be selected with a similar frequency. This is beyond the scope of this book, but more details can be found in other articles related to multi-armed bandit theory such as Slivkins (2019).

13.3.2 Bayesian Optimization

In the previous subsection, fishing spots were discrete and independent. In this subsection, we consider the problem of deciding how many kilometers away from land to fish. In other words, we consider the problem in which fishing spots are continuous and one-dimensional. After scaling the distance, let x ($0 \leq x \leq 1$) be the distance from land. Let $f(x)$ be a function representing the expected total weight of the fish. $f(x)$ is assumed to be continuous and smooth, although its shape is unknown. For such a black box function $f(x)$, the method of repeatedly exploring for the location x and exploiting the obtained data (i.e., using the posterior distribution) is called *Bayesian optimization*. Here, we assume that the purpose is the same as in the previous subsection: maximizing the cumulative reward. In the following, we will perform Bayesian optimization by assuming that $f(x)$ follows a Gaussian process (GP) in Sect. 12.7 and using Thompson sampling.

The R and Python implementation codes are the following:

run-model13-3.R

```
library(cmdstanr)
set.seed(123)

fetch_y <- function(x) {
 mu <- cos(9.5*x-2) - sin(15*x-1) + 5
 SD <- 0.25
 y <- rnorm(n=length(x), mean=mu, sd=SD)
 return(y)
}

T <- 20
Np <- 81
Xp <- seq(0, 1, len=Np)
x_sel <- 0.5
y_sel <- fetch_y(x_sel)
x_cum <- x_sel
y_cum <- y_sel
```

[5] One more possible problem is to judge which spot is sufficiently good. Such a problem is called *good arm identification*.

```
model <- cmdstan_model('../chap12/model/model12-7d.stan')

for (t in 1:T) {
  N <- length(x_cum)
  data <- list(N=N, Np=Np, X=x_cum, Xp=Xp, Mu=rep(0,N), rep(0,Np),
           Y=y_cum - mean(y_cum))
  mess <- capture.output(
    fit <- model$sample(data=data, seed=123, parallel_chains=4,
                refresh=0, show_messages=FALSE))
  fp_ms <- fit$draws('fp', format='matrix')
  N_ms <- nrow(fp_ms)
  rand <- sample.int(n=N_ms, size=1)
  np_sel <- which.max(fp_ms[rand,])
  x_sel <- Xp[np_sel]
  y_sel <- fetch_y(x_sel)
  x_cum <- c(x_cum, x_sel)
  y_cum <- c(y_cum, y_sel)
}
```

run-model13-3.py

```
import cmdstanpy
import numpy as np
from scipy.stats import beta
import logging
from cmdstanpy.utils import get_logger
get_logger().setLevel(logging.ERROR)
np.random.seed(123)

def fetch_y(x):
    mu = np.cos(9.5*x-2) - np.sin(15*x-1) + 5
    SD = 0.25
    y = np.random.normal(mu, SD, x.size)
    return(y)

T = 20
Np = 81
Xp = np.linspace(0, 1, Np)
x_sel = np.array([0.5])
y_sel = fetch_y(x_sel)
x_cum = x_sel
y_cum = y_sel
model = cmdstanpy.CmdStanModel(stan_file='../chap12/model/model12-7d.stan')

for t in range(T):
    N = len(x_cum)
    data = {'N':N, 'Np':Np, 'X':x_cum, 'Xp':Xp,
         'Mu':np.repeat(0,N), 'Mup':np.repeat(0,Np),
         'Y':y_cum - np.mean(y_cum)}
    fit = model.sample(data=data, seed=123, parallel_chains=4,
               show_progress=False)
    fp_ms = fit.draws_pd(['fp'])
    N_ms = len(fp_ms)
```

```
33      rand = np.random.randint(0, N_ms)
34      np_sel = np.argmax(fp_ms.iloc[rand])
35      x_sel = Xp[np_sel]
36      y_sel = fetch_y(x_sel)
37      x_cum = np.append(x_cum, x_sel)
38      y_cum = np.append(y_cum, y_sel)
```

Here we omit the explanation of the GP.

Lines 4–9 in R (lines 9–13 in Python): These are the true settings of this simulation. The function `fetch_y` is defined to obtain the data under the searching location `x`. This function is used in line 32 in R (line 36 in Python).

Lines 12–13 in R (lines 16–17 in Python): The searching region is scaled to be within the range [0, 1]. This range is further divided into the regular intervals and a total of `81` `Xp` is assigned as the candidate locations of search.

Line 14 in R (line 18 in Python): Here the first searching location is assigned to be `0.5` (the middle of the search region).

Line 23 in R (line 28 in Python): This centering facilitates the GP (see Sect. 12.7).

Lines 29–31 in R (lines 33–35 in Python): These lines correspond to Thompson sampling. A curve $\tilde{f}\left(x_p\right)$ is generated as a draw, then the next location with the largest value $\tilde{f}\left(x_p\right)$ is selected from the candidate locations `Xp`.

Lines 33–34 in R (lines 37–38 in Python): The data of selected location and result is accumulated. Note that for the estimation of the curve, all the data obtained before is used.

The example of the result of Bayesian optimization is shown in Fig. 13.3. With an increasing number of iterations T, the 95% Bayesian confidence intervals of $f(x)$ will be narrower. When $t = 20$, it is shown that the location that maximizes $f(x)$ is somewhere near $x = 0.3$ or 0.8.

Although we used Thompson sampling in Bayesian optimization in this subsection, other algorithms have also been proposed (Shahriari et al., 2015). In addition, we have been using the example where search region is only one dimension, but Bayesian optimization can also be applied to those data with larger dimensions of search region. The application of a higher dimensional Bayesian optimization, as well as its variants with additional conditions are still under the research.

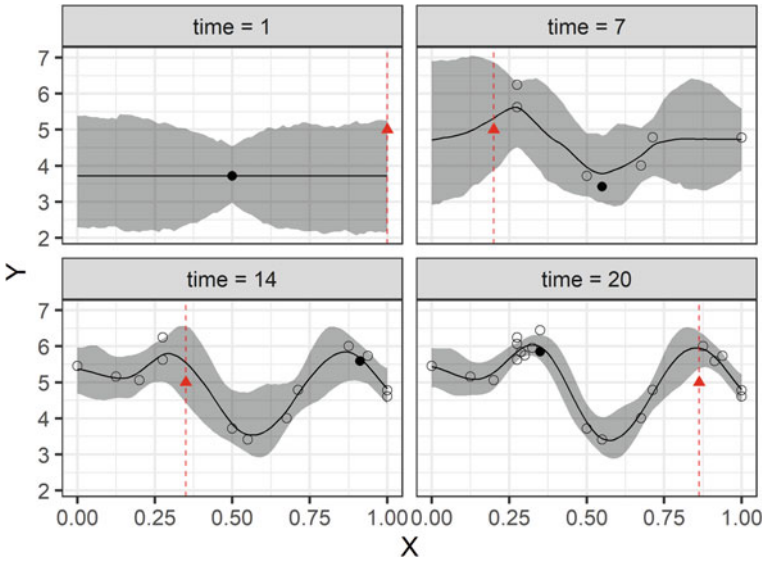


Fig. 13.3 Example of Bayesian optimization, after $t = 1, 7, 14$, and 20. For all the plots, the gray bands are the 95% confidence intervals of $f(x)$, and the black lines are the medians of $f(x)$. The solid black dots are the newly obtained data points. The hollow dots are the past data points. The triangle dots are the locations where the next iteration $(t + 1)$ will explore

References

DeGroot, M. H. (1970). *Optimal statistical decisions*. McGraw Hill.

Fleiss, J.L., Tytun, A., & Ury, H.K. (1980). A simple approximation for calculating sample sizes for comparing independent proportions. *Biometrics* 343–346.

O'Hagan, A., Stevens, J. W., & Campbell, M. J. (2005). Assurance in clinical trial design. *Pharmaceutical Statistics, 4*(3), 187–201.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & De Freitas, N. (2015). Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE, 104*(1), 148–175.

Slivkins, A. (2019). Introduction to multi-armed bandits. arXiv preprint arXiv:1904.07272.

Chapter 14
Other Advanced Topics

In this chapter, we will introduce some advanced topics that have not been mentioned in the previous chapters.

- In Sect. 14.1, we will introduce survival analysis, or more generally time-to-event analysis, which is a method that analyzes the length of time until the occurrence of an event. The key point of this analysis is using the hazard function and considering the likelihood of the censored data.
- In Sect. 14.2, we will work with matrix decomposition and dimensionality reduction. These topics may sound unrelated. However, they have close connections, and therefore, we decided to put them together into one section. These methods usually require a longer time for the estimation, and we use either MAP estimation or ADVI, instead of NUTS.
- In Sect. 14.3, we will discuss how to do model selection based on an information criterion. We especially focus on WAIC, which is a relatively newly developed information criterion. Computing WAIC for a hierarchical model is sometimes confusing to the beginners, therefore we will explain how to do this process in detail. In the end, we will compare WAIC and LOOIC.

14.1 Survival Analysis

The data analysis on survival time is called *survival analysis*. The survival time indicates the length of time until the occurrence of an event. In this section, we will explain the basic concept of survival time analysis.

As an example, let us look at the data that stores how long employees worked in a big company, where the unit of the time is month. The data also contains the start and end of the time that the employees worked in the company. Here, we define "leaving the company" as the event and "the months of working for the company" as the survival time. The observation period starts from 01/01/2014, and all the employees that joined the company after this date were recorded in this data. The

© Springer Nature Singapore Pte Ltd. 2022

K. Matsuura, *Bayesian Statistical Modeling with Stan, R, and Python*,
https://doi.org/10.1007/978-981-19-4755-1_14

recording ended on 12/31/2018. We have the comma-separated file where total of 300 employees are recorded (Data File 14.1, a row corresponds to an employee). Each column represents:

$PersonID$:	The index of each employee.
$Start$:	The time that the employee joined the company.
$Cens$:	A binary value representing a censored data (0: not censored; 1: censored).
End:	When $Cens = 0$, this date represents the time the employee left the company. When $Cens = 1$, this represents the date of the end of the observation period.
$Time$:	The length of time in months, from $Start$ to End.

Data File 14.1 Structure of `data-surv.csv`

```
  1    PersonID,Start,Cens,End,Time
  2    1,2014-01-01,0,2014-02-28,2
  3    2,2014-01-01,0,2014-06-30,6
  4    3,2014-01-01,0,2016-01-31,25
  5    4,2014-01-01,0,2018-08-31,56
  6    5,2014-01-01,0,2014-03-31,3
  7    6,2014-02-01,1,2018-12-31,60
  8    ...
301    300,2018-12-01,1,2018-12-31,2
```

Let us assume that the purpose of this analysis is to understand when the employees are likely to leave the company after joining the company. Therefore, we want to estimate the *survival function*. The value of the survival function $F(t)$ represents the probability that an employee is still working (*survival probability*) at a time point t after the employee joined the company. A curve with t on the x-axis and $F(t)$ on the y-axis is called a survival curve. The survival probability starts from 1, and decreases over time, and gradually approaches zero. The survival probability usually decreases in a similar manner as an exponential function.

We plotted three graphs in order to check the data distribution. Figure 14.1 (left) shows the survival time represented by the straight lines for the employees with $PersonID = 1$ to 16. We can confirm that there are two censored data. As we can see, the censoring accompanied with the end of the observation period is a general feature in the survival time data. Because of the censoring, we cannot simply use the regression for such data. Figure 14.1 (middle) shows the histogram of the column $Time$. A tail at a larger $Time$ is one of the features of the distribution of the survival time, as we can observe from here. Figure 14.1 (right) shows the survival function estimated from the data using the Kaplan–Meier method. The result indicates that the decrease in survival probability is faster when the $Time$ is less than about 20 months (in other words, the employees are more likely to leave the company in this time period). It also indicates that after the first 20 months, the decrease in the survival probability slows down. In this book, we will not explain the Kaplan–Meier method,

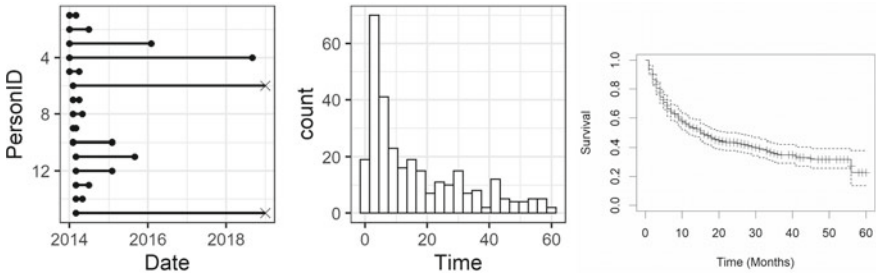


Fig. 14.1 The visualization of the survival time data. Left: an example of the survival time data. The black line represents the time that each employee worked for the company. The black dots represent where each employee joined and left the company. The black crosses represent the censoring accompanying the end of the observation period. Middle: the distribution of the survival time. We have included both the case where the employees actually left the company, as well as the censored data points, without distinguishing these two. Right: the survival function $F(t)$ estimated by the Kaplan–Meier method. The dotted lines show the 95% confidence intervals and the vertical bars represent the censored data

but the interested readers can refer Moore (2016) to find more information. One limitation of the Kaplan–Meier method is that it is a nonparametric method and it does not construct a model. Therefore, combining multiple explanatory variables and incorporating the prior knowledge is impossible. Here, we will construct our own survival function by considering the data generation mechanism.

For the survival time data, we usually use the discrete units, such as a day or a month, to measure the length of time. Therefore, for this data, we will also use a month as a unit and treat the length of time as a discrete variable as $t = 1, 2, \ldots$. When we consider the mechanism of generating the survival function, a key concept is a *hazard function*. A hazard function $h(t)$ represents the probability that the event does not occur until the point $t - 1$, and then the event occurs at the point t. Correspondingly, $1 - h(t)$ is the probability that the event still does not occur at the point t. Obviously, $F(t)$ represents the probability that an employee still works at the time point t, and it is equivalent to the probability that the event did not happen from the initial time point $t' = 1$ to the point $t' = 1$. Therefore, we can conclude the following relationships:

$$F(t) = \prod_{t'=1}^{t} \left(1 - h\left(t'\right)\right) \tag{14.1}$$

By taking the logarithm on both sides of the equation, we have the following equation:

$$\log\ F(t) = \sum_{t'=1}^{t} \log\left(1 - h\left(t'\right)\right) \tag{14.2}$$

Now let us look at how to construct the likelihood. We will first consider the case where the data does not have the censoring. As an example, consider that there is data where the event occurred at the time point $t = Time$. The likelihood of this data can be considered as the product of $F(Time - 1)$ (the probability that an employee was still working at the time point $t = Time - 1$) and $h(Time)$ (the probability that the event occurred at the time point $t = Time$). By writing this log-likelihood, we can obtain the following equation:

$$\log L = \log F(Time - 1) + \log h(Time) \tag{14.3}$$

Now we can consider the data with censoring. Consider that a censoring happened at the time point $t = Time$. In this case, we consider that an employee was still working until $t = Time$, but we do not know what happened after this time point because of the censoring. Therefore, by ignoring the likelihood after this point, we can get the log-likelihood as the following equation:

$$\log L = \log F(Time) \tag{14.4}$$

We can then obtain the overall likelihood by using either Eqs. (14.3) or (14.4) for every data point.

So far, we have explained how to derive the survival function from the hazard function, and how to calculate the log-likelihood from these two functions. Lastly, we will explain how to express the hazard function, which is the key part of the statistical modeling for survival time data. The simplest hazard function is the following function, where we assume that it does not change over time:

$$h(t) = c_0 \tag{14.5}$$

In this case, the survival function is identical to the exponential function. We can incorporate prior knowledge into the hazard function. For instance, we may know that the employees are less likely to leave the company when they have been working for the company for 2–3 years. On the other hand, when they just joined the company, or when they have been working for the company for 5 years, they are more likely to leave the company. If we have this prior knowledge, we can use the quadratic function to construct the hazard function as follows:

$$h(t) = c_0(t - c_1)^2 + c_2 \tag{14.6}$$

An additional note is that we can use a Gaussian Markov random field or a Gaussian process to construct the hazard function, especially when the amount of data is enough (see Chap. 12).

We may also want to incorporate the other explanatory variables, such as which departments the employees work on. We have several options here, and one of the most commonly used assumptions is called *proportional hazard assumption*. That is,

we assume that the hazard function can be expressed as the product of two terms. The first one is the term that depends on the time t, as we have been using so far (this part is called the baseline hazard function), and the second one is the term that depends on the explanatory variables. For instance, if we consider a case where there are two explanatory variables X_1 and X_2, we can express the hazard function as follows:

$$h(t) = h_0(t) \times \exp(b_1 X_1 + b_2 X_2) \tag{14.7}$$

where $h_0(t)$ is the baseline hazard function. When we do not have the evidence that supports that the proportional hazard assumption holds, we need to construct the hazard function properly using the explanatory variables, rather than using Eq. (14.7).

From here, we will introduce how to implement this model using Stan. We define the combination of Eqs. (14.2) to (14.5) as Model Formula 14.1. First, we will estimate the survival function using Model Formula 14.1 for the data from Data File 14.1. The implementation is as follows:

model14-1.stan

```
1   data {
2    int<lower=1> N;
3    int<lower=1> T;
4    array[N] int<lower=1, upper=T> Time;
5    array[N] int<lower=0, upper=1> Cens; // 0:event, 1:censored
6   }
7
8   parameters {
9    real<lower=0, upper=1> c0;
10   }
11
12   transformed parameters {
13    vector[T] h = rep_vector(c0, T);
14    vector[T] log_h = log(h[1:T]);
15    vector[T] log_F = cumulative_sum(log1m(h[1:T]));
16   }
17
18   model {
19    for (n in 1:N) {
20     if (Cens[n] == 0) {
21      target += log_h[Time[n]];
22      if (Time[n] >= 2) {
23       target += log_F[Time[n]-1];
24      }
25     } else {
26      target += log_F[Time[n]];
27     }
28    }
29   }
```

Line 2: `N` is the total number of employees.

Line 3: `T` is the length of the observation period. Because our observation period is from 01/01/2014 to 12/31/2018 here `T` = 60.

Lines 9 and 13: The hazard function is constructed based on Eq. (14.5).

Lines 14–15: The variables `log_h` and `log_F` are defined, in order to compute the log-likelihood easier. Line 15 corresponds to Eq. (14.2). The function `log1m(x)` is a convenient function in Stan that computes $\log(1 - x)$ with a good numeric stability. The function `cumulative_sum(v)` returns the cumulative sums as a vector. If the input vector is $v = (v_1, v_2, v_3, \ldots, v_T)$, then this function returns the vector $(v_1, v_1 + v_2, v_1 + v_2 + v_3, \ldots, v_1 + v_2 + \ldots + v_T)$.

Lines 20–25: These lines correspond to Eq. (14.3).

Lines 25–27: These lines correspond to Eq. (14.4).

We visualized the estimated survival function as in Fig. 14.2 (left), and the hazard function as in Fig. 14.2 (right). From Fig. 14.2 (left), we can see that the survival function estimated from our model is slightly different from that calculated using the Kaplan–Meier method. This suggests that the data did not fit well with this survival function. The assumption that the hazard function is constant over time is likely the issue here.

From the data and our background knowledge, we know that when the employees just joined the company, they are likely to leave the company. On the other hand, when they have been working in the company for about 20 months, they are less likely to leave the company. Hence, instead of using the assumption that the hazard function is constant over time, let us assume that the hazard function changes in a quadratic form. In other words, we will use Eq. (14.6) to replace Eq. (14.5). Here, we define the combination of Eqs. (14.2) to (14.4), and (14.6) as Model Formula 14.2. Its implementation is as follows:

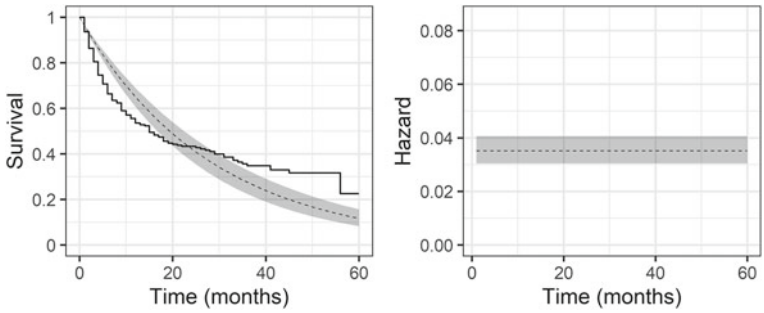


Fig. 14.2 Left: the estimated survival function. The step-like function shown as the black line is the survival function estimated using the Kaplan–Meier method. Right: the estimated hazard function. The gray bands are the 95% Bayesian confidence intervals and the black dotted lines are the median values, for both plots

model14-2.stan

```
1   (data block and model block are the same as model14-1.stan)
2
3   transformed data {
4    vector[T] Time1toT = linspaced_vector(T, 1, T);
5   }
6
7   parameters {
8    real<lower=0, upper=5> c0;
9    real<lower=0, upper=T> c1;
10   real<lower=0, upper=5> c2;
11   }
12
13   transformed parameters {
14    vector[T] h = 1e-4*c0*square(Time1toT[1:T] - c1) + 1e-2*c2;
15    vector[T] log_h = log(h[1:T]);
16    vector[T] log_F = cumulative_sum(log1m(h[1:T]));
17   }
```

Line 4: A vector of element `1`, `2`, ..., `T` is defined using `linspaced_vector` function, in order to use the vectorization in line 14.

Lines 8–10 and 14: These lines correspond to Eq. (14.6). We multiplied small numbers (`1e-4` and `1e-2`) to keep the scale of `c0` and `c2` to be close to 1.

We can obtain the survival function from the estimation result as shown in Fig. 14.3 (left), and the hazard function as shown in Fig. 14.3 (right). Comparing them to the ones in Fig. 14.2, we can see that the data fits well with the survival function.

Besides the example we used here, there is a wide range of applications of the survival analysis. For instance, we can study the length of the time until users decide to end the service contract, the time until new stores shutting down, the time until new products leaving the market, the time until the patients enter the severe stage of diseases, and many others.

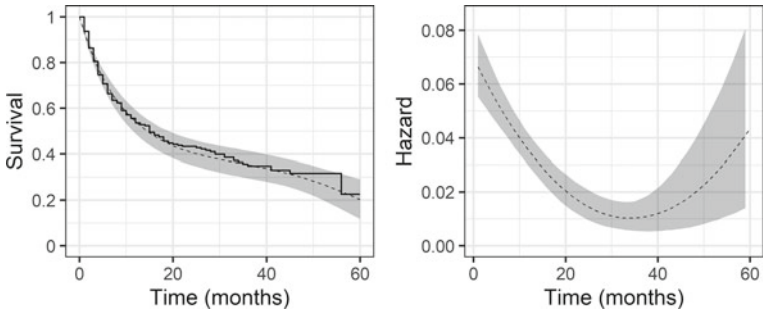


Fig. 14.3 Left: the estimated survival function. Right: the estimated hazard function. The legend is the same as Fig. 14.2

14.2 Matrix Decomposition and Dimensionality Reduction

In this section, we will introduce the method of matrix decomposition and dimensionality reduction. These methods have very close relationships even though they sound like unrelated topics. They reduce the noise in the data, and approximate the high dimensional data with the lower dimensional parameter space. The purpose of these methods is to achieve better interpretability or higher predictive performance. For the matrix decomposition, we will introduce NMF and LDA as popular examples. For the dimensionality reduction, we will introduce SNE and GPLVM. All of them are being widely used for the different data analysis applications.

14.2.1 Matrix Decomposition

Using the multiplication of two matrices with smaller ranks to represent an original matrix is called *matrix decomposition*, or *low-rank approximation*. Using the matrix decomposition, we can get the results with better interpretability, and can also estimate the structure of the denoised data, which further helps increase the prediction performance. The matrix decomposition is used, for example, in the recommendation systems to suggest certain items to certain customers in the market.

In this subsection, we will introduce the statistical model using the matrix decomposition. As an example, we will use the data from the online store of a supermarket, where it records what items were bought by which customers. We have the comma-separated file with 1407 events of purchase history (Data File 14.2, a row corresponds to a purchase event). $PersonID$ is the index of each customer and $ItemID$ is the index of each item.

Data File 14.2 Structure of `data-matrix-decomp.csv`

```
   1   PersonID,ItemID
   2   1,29
   3   1,62
   4   1,26
       ...
1407   50,80
```

Here, we consider that the purpose of this analysis is to: (1) estimate the patterns of purchased items; and (2) extract the features of the customers. From here, N represents the total number of customers and I represents the total number of items. From this data file, we have $N = 50$ and $I = 120$. We generated the cross table from this data, where it counts how many times each customer purchased each item. We stored the result as an $N \times I$ matrix $\mathbf{Y}$. The n-th row, i-th column of $\mathbf{Y}$, $Y[n, i]$, represents the times that the customer n purchased the item i. Figure 14.4 is the visualization of the matrix $\mathbf{Y}$, and the elements that are larger than or equal to 1 are

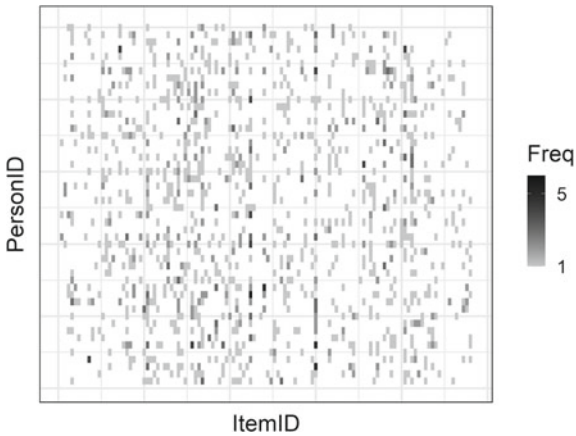


Fig. 14.4 Visualization of **Y**, colored based on the number of purchases (1 is colored as the brightest gray, and the larger numbers are colored as the darker color

colored. From this plot, we can see that the majority of this matrix is white, and thus **Y** is a sparse matrix, where only a small number of customer-item combination have non-zero values.

Nonnegative Matrix Factorization

Let us consider how **Y** was generated. Because the element $Y[n, i]$ in **Y** is the count data, here we assume that it follows a Poisson distribution.

$$Y[n, i] \sim \text{Poisson}(\lambda[n, i]) \; n = 1, \ldots, N \quad i = 1, \ldots, I \tag{14.8}$$

The n-th row, i-th column of the $N \times I$ matrix $\boldsymbol{\lambda}$ is denoted as $\lambda[n, i]$. It represents the average times the item i was purchased by the customer n. Note that $\boldsymbol{\lambda}$ is also a sparse matrix where most values are zero. Here, we consider that there are only a few patterns of items. Also, we assume that we can construct the matrix $\boldsymbol{\lambda}$ by this small number of the patterns without losing much information. Then we can decompose the matrix as follows:

$$\boldsymbol{\lambda} = \boldsymbol{\theta}\,\boldsymbol{\phi} \tag{14.9}$$

where $\boldsymbol{\theta}$ is an $N \times K$ matrix, and $\boldsymbol{\phi}$ is a $K \times I$ matrix. K is much smaller compared to I, and it corresponds to the number of patterns. We can understand that $\overrightarrow{\phi[k]}$ (the k-th row of $\boldsymbol{\phi}$) represents how likely each item is purchased in pattern k. Similarly, we can understand that $\overrightarrow{\theta[n]}$ (the n-th row of $\boldsymbol{\theta}$) represents the extent to which each customer has each pattern component.

Note that in practice, we need to try different numbers of the total patterns and discuss which the most reasonable value is. If the number of patterns is too large, it will cause overfitting, and if the number is too small, patterns that should be separated will not be separated and will be estimated as a single pattern.

The decomposition of $\boldsymbol{\lambda}$ into $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ is not unique. However, by adding extra constraints on $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, it will be unique and hence we can estimate them, although the order of the patterns will still not be unique. For instance, by adding a constraint such that $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are orthogonal, this matrix decomposition will become equivalent to principal component analysis (PCA). However, the orthogonality constraint is a very strict constraint, and $\mathbf{Y}$ cannot be approximated well. In addition, PCA allows elements of $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ to take negative values, which allows $\boldsymbol{\lambda}$ to take negative values, thus this makes the interpretation harder. Therefore, we will add a constraint such that the elements of both $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ matrices are nonnegative. The matrix decomposition with this constraint is called a *nonnegative matrix factorization* (*NMF*). In this case, for each row of $\boldsymbol{\phi}$, we also add a constraint such that the sum of the I elements equals to 1, i.e., the I elements are simplex, for an easy comparison across the patterns in $\boldsymbol{\phi}$.

Next, we will implement this model in Stan. We define the combination of Eqs. (14.8) and (14.9) as Model Formula 14.3. The implementation is as follows:

model14-3.stan

```
1   functions {
2    matrix vector_array_to_matrix(array[] vector x) {
3     matrix[size(x), rows(x[1])] y;
4     for (n in 1:size(x)) {
5      y[n] = x[n]';
6     }
7     return y;
8    }
9   }
10
11   data {
12    int<lower=1> N;
13    int<lower=1> I;
14    int<lower=1> K;
15    array[I,N] int<lower=0> Y;
16   }
17
18   parameters {
19    matrix<lower=0>[N,K] theta;
20    array[K] simplex[I] phi;
21   }
22
23   model {
24    matrix[K,I] phi_m = vector_array_to_matrix(phi);
25    matrix[N,I] lambda = theta * phi_m;
26
27    to_array_1d(Y) ~ poisson(to_vector(lambda));
28   }
```

Lines 2–8: Because Stan does not have a function that can convert an array of `vector` type into `matrix` type, we define it on our own. The function `size` is used to get the length of the array. The function `rows` returns the number of the rows. When the input is a column vector, this function returns the number of elements

in the vector. The symbol "′" on line 5 is the operator of transpose to covert a column vector to a row vector.

Line 14: `K` is the total number of patterns. Here we used `K` $= 6$.

Line 19: The matrix $\boldsymbol{\theta}$ is defined, with the constraint that each element is larger than or equal to 0.

Line 20: The matrix $\boldsymbol{\phi}$ is defined as an array of `simplex` type (which is a subtype of `vector` type), in order to make each column of $\boldsymbol{\phi}$ to be simplex.

Line 24: We convert the array of `simplex` type to `matrix` type.

Line 25: This line corresponds to Eq. (14.9). The matrix $\boldsymbol{\lambda}$ is constructed by the matrix multiplication.

Line 27: This line corresponds to Eq. (14.8). The function `to_array_1d` converts the input as a one-dimensional array in row major order. The function `to_vector` converts the input into `vector` type in column major order. In order to map the elements between these two, we declare the array `Y` as the result of transposing the $N \times I$ matrix $\mathbf{Y}$ in line 15. With this, we can now vectorize the Poisson distribution for the elements of size $N \times I$.

Because there are many parameters in this model, it will take a long time if we use NUTS, which is the standard algorithm in Stan. We therefore used MAP estimation. We will omit the code for the implementation here. Of note, label switching (see Sect. 9.1.3) may happen in this model, because $\boldsymbol{\lambda}$ will not change when we swap the indices in the patterns. In order to make the parameters identifiable, we need to add additional constraints for the parameters.

Based on the estimated result, we visualize how likely each item is purchased in each pattern in Fig. 14.5 (left). From this plot, we can learn that there are several items that are more likely to be bought in the patterns 2, 3, and 6, and we can interpret that these are the featured items for these patterns. In contrast, there is no single specific item associated with the pattern 1 and 4, and their plots look similar. As we explained earlier, when the total number of patterns is more than the actual number of patterns, we might obtain similar patterns as we observe here. In such a case, we can always keep it as it is, or consider reducing the total number of patterns. We also visualize the pattern compositions for the customer $n = 1$ and $n = 50$, as shown in Fig. 14.5 (right). From this plot, we can conclude that the main difference between the customers is in the pattern 1 to 3.

Latent Dirichlet Allocation

From here, we consider a different mechanism from Eq. (14.8) for generating an $N \times I$ matrix $\mathbf{Y}$. One approach is to assume that it follows a multinomial distribution instead of a Poisson distribution:

$$\overrightarrow{Y[n]} \sim \text{Multinomial}\left(\overrightarrow{p[n]}\right) \quad n = 1, \ldots, N \tag{14.10}$$

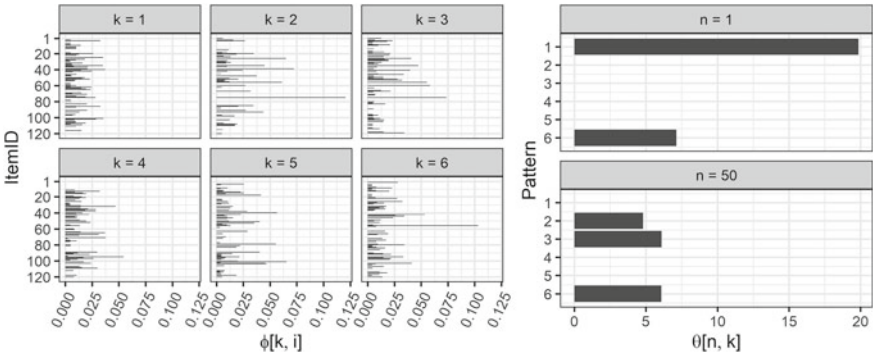


Fig. 14.5 Left: bar plot showing the probabilities of each item, $\phi[k, i]$, for each pattern. Right: the pattern compositions of the customer $n = 1$ and $n = 50$. The bars represent $\theta[1, k]$ and $\theta[50, k]$

where $\overrightarrow{Y[n]}$ is the n-th row of the matrix $\mathbf{Y}$ (hence it is a vector of length I), and $\overrightarrow{p[n]}$ is the n-th row of the matrix $\mathbf{p}$ (hence it is also a vector of length I). As in the case of NMF, we can decompose the matrix $\mathbf{p}$ as follows:

$$\mathbf{p} = \boldsymbol{\theta}\,\boldsymbol{\phi} \tag{14.11}$$

where $\boldsymbol{\theta}$ is an $N \times K$ matrix, and $\boldsymbol{\phi}$ is a $K \times I$ matrix. But here, in addition to the constraint that each row of $\boldsymbol{\phi}$ is a simplex, we also add another constraint such that each row of $\boldsymbol{\theta}$ is also a simplex. With these constraints, the element $p[n, i]$ in $\overrightarrow{p[n]}$ represents the probability that the item i is purchased by the customer n. In other words, the sum of all the elements in $\overrightarrow{p[n]}$ is 1. Let's see why this is the case. From the matrix multiplication in Eq. (14.11), if we extract the term that we calculate the element $p[n, i]$, we obtain:

$$p[n, i] = \sum_{k=1}^{K} [\theta[n, k] \times \phi[k, i]] \tag{14.12}$$

This can be explained as follows: we first roll a dice with K faces, represented by $\overrightarrow{\theta[n]}$, and record what number appeared on the upper face. We assume this number was k. Next, we roll another dice with I faces, represented by $\overrightarrow{\phi[k]}$, and again record the number on the upper face, denoted by i. Then by marginalizing k out by summing over k, we can obtain $\sum_k \mathrm{Prob}(k, i) = \mathrm{Prob}(i)$. In other words, $p[n, i]$ in Eq. (14.12) represents the probability that the number i is on the upper face. Therefore, the sum of I elements in $\overrightarrow{p[n]}$ should be 1.

In this type of purchase data, we often see that there are only a few elements in the pattern $\overrightarrow{\phi[k]}$ that have a large number. Therefore, in order to generate a dice with this tendency, we usually use a Dirichlet distribution as a prior distribution for $\overrightarrow{\phi[k]}$, as shown below (see Sects. 6.8 and 9.2.3):

$$\overrightarrow{\phi[k]} \sim \text{Dirichlet}(\vec{a}) \quad k = 1, \ldots, K \tag{14.13}$$

where $\vec{a}$ is a vector of length I, where all the elements are smaller than 1.

We define the combination of Eqs. (14.10), (14.11) and (14.13) as Model Formula 14.4. We can explain this model from another perspective. Here we want to estimate $\overrightarrow{p[n]}$, which is the probability that each customer purchases these I items. However, it will be hard to estimate the values for all the customers accurately, unless we have many data for each of the customers. Therefore, we can estimate the typical pattern $\overrightarrow{\phi[k]}$ for K patterns from all the customers, and represent $\overrightarrow{p[n]}$ by allocating these patterns in an appropriate proportion. Thus, Model Formula 14.4 is called *latent Dirichlet allocation* (*LDA*).

So far, we have explained LDA using the data of customers purchasing items. The original LDA paper was published in the field called natural language processing, where the customers correspond to the documents, the items correspond to the words, and the patterns are called "topics". LDA is also a variation in *topic models*, where we extract topics from the documents. There is a lot of fast open-source codes in different programming languages to implement the original LDA that we explained in this section. In general, however, it is not straightforward to customize LDA from these codes. Using Stan, we can easily extend and customize LDA, and we can also run the trial-and-error process to check the model easily. Of note, in the original paper, the authors used a Dirichlet distribution for $\overrightarrow{\theta[n]}$, and determined the hyperparameters for each Dirichlet distribution based on the prediction perspective. This content will be beyond the scope of this book, but readers who are interested can refer Blei et al. (2003).

The implementation of Model Formula 14.4 is as follows:

model14-4.stan

```
 (functions block is the same as model14-3.stan)

data {
  int<lower=1> N;
  int<lower=1> I;
  int<lower=1> K;
  array[N,I] int<lower=0> Y;
  vector<lower=0>[I] Alpha;
}

 parameters {
  array[N] simplex[K] theta;
  array[K] simplex[I] phi;
 }

 model {
  matrix[N,K] theta_m = vector_array_to_matrix(theta);
  matrix[K,I] phi_m  =  vector_array_to_matrix(phi);
  matrix[N,I] p = theta_m * phi_m;

  for (k in 1:K) {
```

```
22      phi[k] ~ dirichlet(Alpha);
23    }
24    for (n in 1:N) {
25      Y[n] ~ multinomial(p[n]');
26    }
27  }
```

Line 8: `Alpha` is the parameter of a Dirichlet distribution, and is passed from R or Python as a hyperparameter. The value of a hyperparameter is determined by the analyst, rather than being estimated from the data.

Lines 21–23: A Dirichlet distribution with the parameter `Alpha` is specified to be the prior distribution of $\overrightarrow{\phi[k]}$. In constant, for `theta`, we did not specify any prior distribution and decided to use a noninformative prior.

Lines 24–26: Because the parameter for a multinomial distribution has to be a `vector` type, we use "'" operator to transpose `p[n]` (`row_vector` type). Because Stan can implement the multinomial distribution by automatically obtaining the sum from `Y[n]`, there is no need to pass the total number of purchases.

In this model, we cannot use MAP estimation due to Eq. (14.13), because the density value of $\text{Dirichlet}\left(\vec{0}|\vec{a}\right)$ and $\text{Dirichlet}\left(\vec{1}|\vec{a}\right)$ would be infinite when the elements in $\vec{a}$ is smaller than 1. Therefore, we use ADVI instead (see Sect. 4.4.1).

We introduced NMF and LDA as the examples of the matrix decomposition methods. Despite the difference of using a Poisson distribution versus a multinomial distribution, these two methods have a lot of similarities. In terms of the performance, it depends on the data and we encourage the readers to try both and find the appropriate model.

Embedding

In this subsection, we introduce *embedding* as one of the interpretations of matrix factorization. The embedding uses a real number vector to express an object, such as a word, a customer and an item. We can use the vectors for other purposes. For instance, we can evaluate the similarity across the items using these vectors.

In fact, NMF in the previous subsection, is a variation of the embedding. Let us look at the two matrices $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ in NMF from the other perspective. We can convert Eq. (14.9) as follows, by focusing on the n-th row of $\boldsymbol{\theta}$, $\overrightarrow{\theta[n]}$, and the i-th column of $\boldsymbol{\phi}$, $\overrightarrow{\phi[i]}$[1]:

$$\boldsymbol{\lambda} = \boldsymbol{\theta}\,\boldsymbol{\phi}$$

[1] It should be written as $\overrightarrow{\phi[:, i]}$ because it is a column vector of the i-th column of the matrix $\boldsymbol{\phi}$. But we use $\overrightarrow{\phi[i]}$ for simplicity.

$$
= \begin{pmatrix} -\overrightarrow{\theta[1]}^T- \\ -\overrightarrow{\theta[2]}^T- \\ \vdots \\ -\overrightarrow{\theta[N]}^T- \end{pmatrix} \begin{pmatrix} | & | & & | \\ \overrightarrow{\phi[1]} & \overrightarrow{\phi[2]} & \dots & \overrightarrow{\phi[I]} \\ | & | & & | \end{pmatrix}
$$

$$
= \begin{pmatrix} \overrightarrow{\theta[1]}^T\overrightarrow{\phi[1]} & \cdots & \overrightarrow{\theta[1]}^T\overrightarrow{\phi[I]} \\ \vdots & \ddots & \vdots \\ \overrightarrow{\theta[N]}^T\overrightarrow{\phi[1]} & \cdots & \overrightarrow{\theta[N]}^T\overrightarrow{\phi[I]} \end{pmatrix}
$$

From this, we can learn that $\lambda[n, i]$ is constructed by the inner product between $\overrightarrow{\theta[n]}$ and $\overrightarrow{\phi[i]}$, i.e., $\overrightarrow{\theta[n]}^T\overrightarrow{\phi[i]}$. Because an inner product can be considered as a measurement of similarity, we can consider that how likely the item i is purchased by the customer n, is proportional to the similarity between $\overrightarrow{\theta[n]}$ and $\overrightarrow{\phi[i]}$. The vector $\overrightarrow{\theta[n]}$ of length K is compressed information of the data of the customer n. Therefore, it can be considered as the "customer vector" that represents the feature of the customer n. Likewise, $\overrightarrow{\phi[i]}$ is also a vector of length K, and can be considered as the "item vector" that represents the feature of the item i. In other words, we can interpret the mechanism of NMF as follows: we estimate the customer vectors and the item vectors separately, by defining the similarity between a customer and an item as the inner product between its customer vector and its item vector in their embedding spaces.

In NMF, we expressed the similarity using the inner product between two nonnegative vectors. We can also extend NMF by changing the definition of the similarity, and use different types of embedding.

14.2.2 Dimensionality Reduction

The method that reduces the number of dimensions while keeping as much original information as possible, is usually called *dimensionality reduction*. There are several advantages of using dimensionality reduction. For instance, we can reduce the noise and extract a more robust structure. It also enables a more interpretable visualization by projecting the high-dimensional vector obtained from the embedding onto a two-dimensional space. We can also cut down the memory storage space used to store the information. The matrix decomposition example in Sect. 14.2.1 compressed a length I vector $\overrightarrow{Y[n]}$, which is the purchasing data of the customer n, into a shorter length K vector $\overrightarrow{\theta[n]}$. There are many other methods for the dimensionality reduction, besides the matrix decomposition. Here we will introduce two methods: SNE and GPLVM.

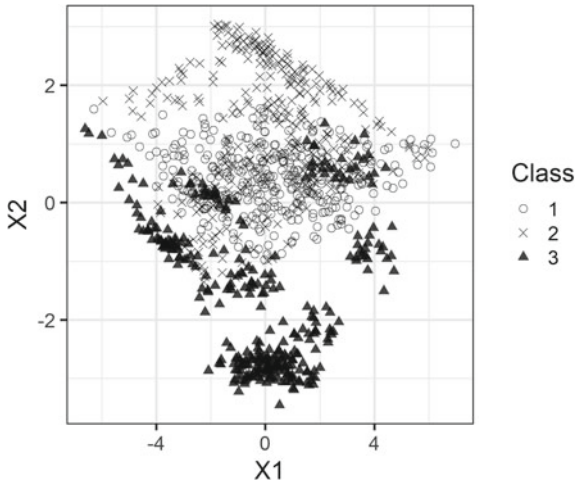


Fig. 14.6 PCA result

The data we use here is the oil flow data. This data is commonly used as an example in the field of machine learning (Bishop, 2006).[2] This is a comma-separated file for 1000 datapoints (Data File 14.3, where a row corresponds to a single data point). Each data point has 12 explanatory variables from $F01$ to $F12$ (real type) and a categorical type named $Class$ (an integer from 1 to 3).

Data File 14.3 Structure of `data-oil.csv`

```
    1   F01,F02,F03,...,F11,F12,Class
    2   0.3315,0.2156,0.6802,...,0.153,0.5856,0.2573,1
    3   0.0939,1.0089,0.0365,...,0.6085,0.0631,0.6597,2
    4   0.5184,0.2283,0.53,...,0.613,0.6705,0.5202,1
        ...
 1001   -0.0197,0.7719,0.4634,...,2.0232,0.025,-0.0265,3
```

Here, we normalized each column for the 12 explanatory variables, and used the results for the following analysis. We show the result of PCA in Fig. 14.6, where the x-axis represents the first principal component, the y-axis represents the second principal component, and the shapes of the dots represent $Class$. From this plot, we find that the dots are not separated clearly based on the $Class$. From here, we will use SNE and GPLVM instead of PCA, and project the data into 2-dimension (we call K-dimension) from the original 12-dimension (we call D-dimension), and aim to plot the similar plot as Fig. 14.6.

[2] This data can be obtained from link: http://inverseprobability.com/3PhaseData.html.
Here, we have preprocessed the original data as data-oil.csv, so that it is easy to load and implement from R and Python.

SNE

The concept of *stochastic neighbor embedding* (*SNE*) is to reduce the dimension of the data points while keeping the similarity between each data point and its neighboring data points as much as possible.

Specifically, we first express the similarity between the data point n and its neighboring data point n' using the following probability:

$$p\left(n'|n\right) = \frac{\exp\left(-\frac{\left\|\overrightarrow{Y\left[n'\right]}-\overrightarrow{Y[n]}\right\|^2}{2\,Var[n]}\right)}{\sum_{m\neq n}\exp\left(-\frac{\left\|\overrightarrow{Y[m]}-\overrightarrow{Y[n]}\right\|^2}{2\,Var[n]}\right)} \tag{14.14}$$

where $\left\|\vec{a}-\vec{b}\right\|^2$ represents the Euclidean distance between $\vec{a}$ and $\vec{b}$. $Var[n]$ is a hyperparameter that determines the neighboring data points of the data point n, and this value is provided from the analyst. Usually when the surrounding data points are denser, we adjust $Var[n]$ to be a smaller value so that the neighborhood region is smaller.

The numerator in Eq. (14.14) represents the density value at the data point $\overrightarrow{Y\left[n'\right]}$, under the assumption that $\overrightarrow{Y\left[n'\right]}$ follows a multivariate normal distribution with mean vector $\overrightarrow{Y[n]}$ and covariance matrix $Var[n]\mathbf{I}$ (the scaling factor in the multivariate normal distribution is ignored here). This density can be considered as a weighted value. We can obtain the probability by dividing the weighted value by the sum of all the weighted values of all the neighboring data points. A data point that is closer to the data point n will have a larger weight, and therefore a larger p. Because we only want to consider the similarity between two different data points, we assume $p(n|n) = 0$.

Next, we use $\overrightarrow{x[n]}$ to represent the projection of $\overrightarrow{Y[n]}$ onto the lower dimension. Similar to the case of Eq. (14.14), we can use the following probability to represent the similarity between the data point n and the neighboring data point n' after the projection.

$$q\left(n'|n\right) = \frac{\exp\left(-\left\|\overrightarrow{x\left[n'\right]}-\overrightarrow{x[n]}\right\|^2\right)}{\sum_{m\neq n}\exp\left(-\left\|\overrightarrow{x[m]}-\overrightarrow{x[n]}\right\|^2\right)} \tag{14.15}$$

Similar to the case of $p(n|n)$, we assume $q(n|n) = 0$.

In SNE, we estimate $\overrightarrow{x[n]}$ that minimizes the following value so that the similarity after the projection $q\left(n'|n\right)$ keep the original similarity $p\left(n'|n\right)$ as much as possible:

$$\text{obj} = \sum_{n=1}^{N} \sum_{n' \neq n} p(n'|n) \log \frac{p(n'|n)}{q(n'|n)} \tag{14.16}$$

where $p \log(p/q) = KL(p\|q)$ is called the Kullback–Leibler divergence (KL divergence), and it is a value that measures how the distribution q differs from the distribution p. The right-hand-side of Eq. (14.16) is the sum of the KL divergence over all the data points and their neighboring data points. We will not give detail on the KL divergence, but readers can refer to the Wikipedia page[3] for more information. The estimation can be unstable when there is no constraint on the projected space. Therefore, we add a weakly informative prior for $\overrightarrow{x[n]}$ as follows:

$$x[n,k] \sim \mathcal{N}(0,5) \; n = 1, \ldots, N \quad k = 1, \ldots, K \tag{14.17}$$

One variation of SNE is t-SNE. In the original paper of t-SNE (Maaten & Hinton, 2008), they used a Cauchy distribution with one degree of freedom, instead of a multivariate normal distribution, to calculate the similarity. This can be expressed mathematically as:

$$q(n'|n) = \frac{\left(1 + \left\|\overrightarrow{x[n']} - \overrightarrow{x[n]}\right\|^2\right)^{-1}}{\sum_{m \neq n}\left(1 + \left\|\overrightarrow{x[m]} - \overrightarrow{x[n]}\right\|^2\right)^{-1}} \tag{14.18}$$

We will now introduce an example of how to implement SNE using a multivariate normal distribution.

model14-5.stan

```
functions {
  matrix calc_prob_vector(matrix x, vector coef) {
    int N = rows(x);
    matrix[N,N-1] p;
    matrix[N,N-1] x_simi;
    for (i in 1:N) {
      int idx = 1;
      for (j in 1:N) {
        if (j == i) {
          continue;
        }
        x_simi[i,idx] = - coef[i] * squared_distance(x[i], x[j]);
        idx += 1;
      }
      p[i] = softmax(x_simi[i]')';
    }
    return(p);
```

[3] https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence.

```
18      }
19    }
20
21    data {
22      int N;
23      int D;
24      int K;
25      matrix[N,D] Y;
26      vector[N] Prec;
27    }
28
29    transformed data {
30      matrix[N,N-1] p = calc_prob_vector(Y, 0.5*Prec);
31    }
32
33    parameters {
34      matrix[N,K] x;
35    }
36
37    model {
38      matrix[N,N-1] q = calc_prob_vector(x, rep_vector(1,N));
39      target += - sum(p .* (log(p) - log(q)));
40      to_vector(x) ~ normal(0, 5);
41    }
```

Lines 2–18: The function `calc_prob_vector` is defined here. This function calculates an $N \times (N-1)$ matrix `p` from `x`, which can be either a matrix of all the data points or the all the projected data points. In order to remove the case $n' = n$, we have adjusted the indices using the `idx` variable in lines 7, 10 and 13.

Lines 12 and 15: Eqs. (14.14) and (14.15) are implemented in these two lines. The probability is computed using `softmax` function. For now, `softmax` function only accepts an input of `vector` type. Therefore, we have used "`'`" operator to transpose the `i`-th row of `x_simi` into `vector` type.

Lines 23–24: `D` and `K` are the number of dimensions before and after the projection, respectively.

Line 30: This line corresponds to Eq. (14.14). Instead of dividing by $2\,Var[n]$, we multiply its inverse. `Prec` is used because the inverse of the variance is called precision.

Line 38: This line corresponds to Eq. (14.15).

Line 39: This line corresponds to Eq. (14.16). The "`.*`" operator represents the element-wise multiplication. The log function is also an element-wise function and the sum function is the summation of all the elements in the matrix. If the inside of log is 0, an error occurs during sampling. To avoid that error, it may be necessary to add a small value such as 1e-6 to the inside of log.

Line 40: This line corresponds to Eq. (14.17).

The R and Python code to execute the Stan code is as follows:
A part of code **run-model14-5.R**

```
1   # N:1000, D:12, Y:scaled data
2
3   K <- 2
4   M <- 30
5   dist <- as.matrix(dist(Y))
6   var_vec <- numeric(N)
7   for (n in 1:N) {
8     var_vec[n] <- sort(dist[n,], partial=M)[M] / (3*3)
9   }
10  Prec <- 1/var_vec
```

A part of code **run-model14-5.py**

```
1   # N:1000, D:12, Y:scaled data
2   from scipy.spatial.distance import squareform, pdist
3   K = 2
4   M = 30
5   dist = squareform(pdist(Y))
6   var_vec = np.zeros(N)
7   for n in range(N):
8     var_vec[n] = np.sort(dist[n,])[M-1] / (3*3)
9
10  Prec = 1/var_vec
```

Line 3: Here we assumed the number of dimensions is 2 after the projection.

Lines 4–10: We determine $Var[n]$ from the data. There are multiple ways to do this, and we use the simplest method here for demonstration. First, we calculate the distance matrix of data in line 5. Next, we calculate the `M`-th closest data point from the data point n, and divided the distance by `3*3`. This ensures that if we consider a normal distribution, the distance for this data point become 3SD. The original paper adapted a more advanced method based on the entropy.

The estimated result of $\overrightarrow{x[n]}$ using a point estimation is shown in Fig. 14.7 (left). We can observe that the original classes are well separated. In PCA, we had to throw away the information after the third principal components. But in SNE, we can use the full information but still project the information onto the two dimensions.

GPLVM

A Gaussian process latent variable model (*GPLVM*) is a model that generates the high-dimensional data from a Gaussian process (GP) using low-dimensional latent variables (Titsias & Lawrence, 2010). We will not introduce a GP here, but we point readers to Chap. 12 for more detail.

From here, as in SNE, we use $\overrightarrow{x[n]}$ to represent the projection of the D-dimensional $\overrightarrow{Y[n]}$ onto a lower K-dimensioanl space. The key assumption here is that if $\overrightarrow{x[n]}$ is similar to $\overrightarrow{x[n']}$ then $\overrightarrow{Y[n]}$ is similar to $\overrightarrow{Y[n']}$. For simplicity, instead of assuming

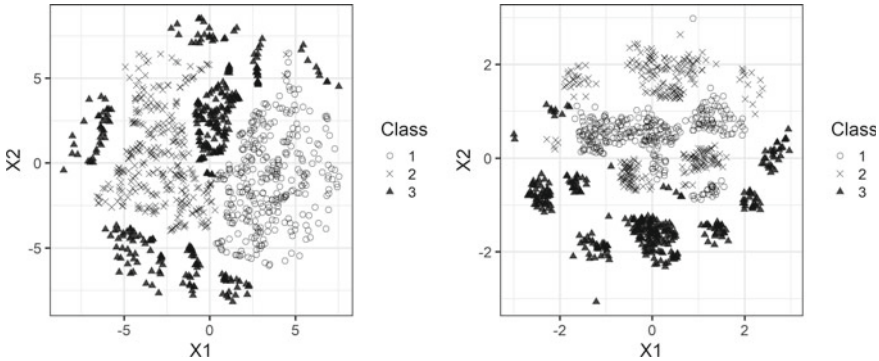


Fig. 14.7 The results of dimensionality reduction. Left: the result from SNE. The values of estimated $\overrightarrow{x[n]}$ are plotted. Right: the result from GPLVM. The median values of estimated $\overrightarrow{x[n]}$ are plotted

that $\overrightarrow{Y[n]}$ is similar to $\overrightarrow{Y[n']}$, let us assume that $Y[n, d]$ is similar to $Y[n', d]$ for each dimension $d = 1, \ldots, D$. To express this similarity assumption, we will use a GP, where we consider that the D vectors, $\overrightarrow{Y[:, 1]}$, $\overrightarrow{Y[:, 2]}$, …, $\overrightarrow{Y[:, D]}$, are generated from a single multivariate normal distribution. Mathematically, this is expressed as follows:

$$\overrightarrow{Y[:, d]} \sim \text{MultiNormal}\left(\vec{0}, \mathbf{K} + \sigma_y^2 \mathbf{I}\right) \; d = 1, \ldots, D \tag{14.19}$$

where $\overrightarrow{Y[:, d]}$ represents the column vector of the d-th column of the matrix $\mathbf{Y}$. $\mathbf{I}$ is an identity matrix of size $N \times N$. $\mathbf{K}$ is a kernel matrix of size $N \times N$, and its element at n-th row, n'-th column is as follows:

$$k\left(\overrightarrow{x[n]}, \overrightarrow{x[n']}\right) = a^2 \exp\left(-\frac{1}{2\rho^2} \sum_{k=1}^{K} \left(x[n, k] - x[n', k]\right)^2\right) \tag{14.20}$$

In Eq. (14.19), we use the same kernel matrix $\mathbf{K}$ for each $d = 1, \ldots, D$, by shifting the mean of $\overrightarrow{Y[:, d]}$ to zero and scaling $\overrightarrow{Y[:, d]}$.[4]

As in SNE, we used the following prior distribution for $\overrightarrow{x[n]}$ to stabilize the estimation:

$$x[n, k] \sim \mathcal{N}(0, 1) \qquad n = 1, \ldots, N \quad k = 1, \ldots, K \tag{14.21}$$

The advantage of GPLVM compared to SNE is that we can set different prior distributions for $\overrightarrow{x[n]}$ based on our needs. For instance, by assuming that $\overrightarrow{x[n]}$ follows

[4] We can also use different kernel matrices for each d, with different values of α and ρ. Note this would increase the number of parameters.

a mixture of multivariate normal distributions, we can do clustering in the latent space. By assuming that $\overrightarrow{x[t]}$ follows a system model in the state space model (see Chap. 11), we can estimate how $\overrightarrow{x[t]}$ changes over time in the latent space.

Next, we introduce the implementation example of GPLVM:

model14-6.stan

```
1   data {
2     int N;
3     int D;
4     int K;
5     vector[N] Mu;
6     array[D] vector[N] Y;
7   }
8
9   parameters {
10    array[N] vector[K] x;
11    real<lower=0> a;
12    real<lower=0> rho;
13    real<lower=0> s_y;
14   }
15
16   model {
17   matrix[N,N] cov = add_diag(gp_exp_quad_cov(x, a, rho), square(s_y));
18    for (n in 1:N) {
19      x[n] ~ normal(0, 1);
20    }
21    a   ~ normal(0, 1);
22    rho ~ normal(0.1, 0.1);
23    s_y ~ normal(0, 1);
24    Y[1:D] ~ multi_normal_cholesky(Mu, cholesky_decompose(cov));
25   }
```

Line 17: Building the kernel matrix. `gp_exp_quad_cov` function is a handy function that returns a kernel matrix by following the Gaussian kernel given in Eq. (14.20). Note that we need to pass `x` as an array of `vector` type, instead of `matrix` type, which is indicated in line 10.

Lines 18–20: These two lines correspond to Eq. (14.21).

Line 24: This line corresponds to Eq. (14.19), and is equivalent to the following three lines. Refer `multi_normal_cholesky` function in Stan reference for detail.

```
for (d in 1:D) {
 Y[d] ~ multi_normal_cholesky(Mu, cholesky_decompose(cov));
}
```

We used ADVI for the estimation, and the result is shown in Fig. 14.7 (right). As in the case of SNE, we can observe that the original classes are well-separated. And the data is projected onto 2-dimensional space using all the information from the data.

14.3 Model Selection Based on Information Criteria

14.3.1 Introduction of Generalization Error and WAIC

In the statistical modeling process, we generally need to consider several models. It is important to report a set of each model and the results obtained from the model rather than selecting one model from several. By investigating the results from different models, we can better understand how each assumption is associated with each result. And this is also helpful for us to learn how robust the estimated results are.

However, in some cases, we have to report the results based on a small number of models. In these cases, we need to choose the model based on some criteria, and this is a process called *model selection.* When we select a model, we usually use the information criterion as a standard measurement. Some examples of the information criterion include Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC), Widely Applicable Information Criterion (WAIC), Widely applicable Bayesian Information Criterion (WBIC) and others. AIC and WAIC measure the prediction performance, whereas BIC and WBIC measure how the model is close to the true model. For the explanation purpose, we define "good condition", where (1) the model is relatively simple and the amount of data is sufficiently large, and hence the posterior distribution can be well-approximated using a multivariate normal distribution; and (2) the fitted models include the true model. Under this "good condition", AIC is asymptotically equivalent to WAIC, and BIC is asymptotically equivalent to WBIC. In a more general case, such as the case of a hierarchical model or a model with latent variables, the "good condition" is not satisfied, and the posterior distribution could not be approximated using a normal distribution. Therefore, there is no theoretical justification on using either AIC or BIC. In such cases, using WAIC or WBIC is recommended (see Watanabe, 2018). Generally, we use WAIC when we want to select the model whose predictive distribution is close to the true data distribution, and use WBIC when we want to select the model that is close to the true model. In this book, we focus more on the predictive distribution and recommend using WAIC. From here, we will introduce the concept of WAIC and its implementation.

WAIC is an approximation of the generalization error (GE), and is a random variable that depends on data. The smaller WAIC indicates that the predictive distribution is closer to the true data distribution. Before we dig into the introduction on WAIC, let us look at how GE is expressed. GE represents the difference between the true data distribution of y and the predictive distribution of y, which is estimated from data. GE can be written as follows:

$$\mathrm{GE} = \int q(y)\big(-\log p_{\mathrm{pred}}(y)\big)dy \tag{14.22}$$

where $q(y)$ is the true data distribution, and $p_{\mathrm{pred}}(y)$ is the predictive distribution. In Bayesian inference, and $p_{\mathrm{pred}}(y)$ is as follows:

$$p_{\text{pred}}(y) = \mathbb{E}[p(y|\theta)] \\ = \int p(y|\theta)p(\theta|Y)d\theta \tag{14.23}$$

where $\mathbb{E}$ is the average over the posterior distribution of θ. $p(y|\theta)$ is the probabilistic model that generates the data Y, and it is defined by the analyst. $p(\theta|y)$ is the posterior distribution of θ, and this is estimated based on the prior distribution of θ and the data Y. If y follows a discrete probability distribution, we can simply replace the integral in the above equation with a summation.

Based on Jensen's inequality,[5] the following formula holds:

$$\int q(y)(-\log q(y))\,dy \leq \int q(y)\big(-\log p_{\text{pred}}(y)\big)dy \tag{14.24}$$

The left-hand side of Eq. (14.24) is called entropy. The entropy is an unknown value, because it is only determined by the true data distribution and does not depend on the parameters. When GE on the right-hand side gets smaller and closer to the entropy, we can consider that $p_{\text{pred}}(y)$ accurately estimates $q(y)$. By subtracting the entropy term from GE, we obtain:

$$\text{GE} - \text{entropy} = \int q(y)\big(-\log p_{\text{pred}}(y)\big)dy - \int q(y)(-\log q(y))dy \\ = \int q(y)\log\frac{q(y)}{p_{\text{pred}}(y)} \tag{14.25}$$

We can see from here that GE substantially measures the difference between the true data distribution $q(y)$ and the predictive distribution $p_{\text{pred}}(y)$ using KL divergence.

Above, we have explained that when GE is smaller, the difference between the true data distribution and the predictive distribution is smaller. However, in practice, we do not know the true data distribution $q(y)$, and need to approximate GE using the quantities that are accessible from the data and the model. These quantities are training error and functional variance. Both of these two are the random variables that depend on the data.

The *training error* T_n of the n-th data point is defined as follows:

$$T_n = -\log \mathbb{E}[p(Y[n]|\theta)] \tag{14.26}$$

where $\mathbb{E}[p(Y[n]|\theta)]$ is the value of density of the predictive distribution $\mathbb{E}[p(y|\theta)]$ at the data point $Y[n]$. We can also see this as the average of the likelihood $p(Y[n]|\theta)$ for the data point $Y[n]$ over the posterior distribution. The larger $\mathbb{E}[p(Y[n]|\theta)]$ is, the smaller the training error T_n is. In other words, the training error measures how

[5] https://en.wikipedia.org/wiki/Jensen's_inequality.

the predictive distribution fits to the data. Under the "good condition", the predictive distribution $\mathbb{E}[p(y|\theta)]$ is almost equivalent to the predictive distribution $p(y|\theta^*)$ that is computed from MLE, and therefore they are interchangeable.

The *functional variance* V_n of the n-th data point is defined as follows:

$$\begin{aligned} V_n &= \mathbb{V}\big[\log p(Y[n]|\theta)\big] \\ &= \mathbb{E}\big[(\log p(Y[n]|\theta))^2\big] - \mathbb{E}\big[\log p(Y[n]|\theta)\big]^2 \end{aligned} \tag{14.27}$$

where $\mathbb{V}$ represents the variance of the log-likelihood $\log p(Y[n]|\theta)$ whose variability depends on the posterior distribution of the parameter θ. Under the "good condition", the sum of the functional variance over all the data points is equivalent to the number of the parameters. The functional variance is a quantity that is developed in recent studies in the field of algebraic geometry and learning theory, and its importance has been receiving a lot of attention.

WAIC_n of the n-th data point is defined as follows

$$\mathrm{WAIC}_n = T_n + V_n \tag{14.28}$$

To calculate WAIC of all the data, we need to take the summation of WAIC_n over all the data points, and divide it by the total number of data points N, in order to convert the scale to the per data point unit.

$$\mathrm{WAIC} = \frac{1}{N}\sum_{n=1}^{N} \mathrm{WAIC}_n \tag{14.29}$$

When we calculate WAIC in practice, we only need the draws of the log-likelihood $\log p(Y[n]|\theta)$, under the posterior distribution of the parameters. With these values, we can compute the training error and functional variance based on Eqs. (14.26) and (14.27). We will see more details in the code **`model14-7.stan`**, **`run-model14-7.R`** and **`run-model14-7.py`**.

WAIC in Eq. (14.29) is asymptotically equivalent to GE when N increases. This is because the following formula holds:

$$\mathbb{E}_{\mathrm{data}}[\mathrm{GE}] = \mathbb{E}_{\mathrm{data}}[\mathrm{WAIC}] + \mathcal{O}\big(1/N^2\big)$$

where $\mathbb{E}_{\mathrm{data}}[f]$ represents the average over data generation. In other words, it is the expected value of f by repeating many times generating N data points from the true data distribution and measuring f. The second term on the right-hand side decreases with the order of $1/N^2$, and therefore, the larger N is, the better WAIC approximates to GE.

WAIC is not the only way to approximate GE. For instance, leave-one-out cross validation information criterion (LOOIC) is another approximation of GE. Cross

validation is commonly used in the field of machine learning. LOOIC over all the data points can be written mathematically as follows.

$$\mathrm{LOOIC} = \frac{1}{N}\sum_{n=1}^{N}\left(-\log \mathbb{E}^{(-n)}[p(Y[n]|\theta)]\right)$$

where $\mathbb{E}^{(-n)}$ is the average over the posterior distribution estimated from all the data except for the n-th data point. For LOOIC, it was also proven that with a sufficiently large N, the following holds asymptotically:

$$\mathbb{E}_{\mathrm{data}}[\mathrm{GE}] = \mathbb{E}_{\mathrm{data}}[\mathrm{LOOIC}] + \mathcal{O}\left(1/N^2\right)$$

14.3.2 Simulation Study to Evaluate Information Criteria

GE represents the theoretical prediction performance. Therefore, for evaluating the performance of information criteria, it is essential to compare them with GE. From here, we will try to get a better understanding on the information criteria by calculating GE, WAIC and LOOIC when we know the true data distribution $q(y)$.

First, we consider the multiple linear regression case and explain the process of the simulation. We assumed the true data distribution is as follows:

$$Y[n] \sim \mathcal{N}(1.3 - 3.1X_2[n] + 0.7X_3[n], 2.5) \quad n = 1, \ldots, N$$

where N is the total number of data points, Y is the data of a response variable, X_2 and X_3 are the data of explanatory variables. We fit the following model to the data:

$$Y[n] \sim \mathcal{N}(b_1 + b_2X_2 + b_3X_3, \sigma) \quad n = 1, \ldots, N$$

We estimate the parameters b_1, b_2, b_3 and σ from the data. Thus, the predictive distribution using the same explanatory variables of the n-th data point is as follows:

$$\begin{aligned} p_{\mathrm{pred}}(y) &= \mathbb{E}[p(y|\theta)] \\ &= \mathbb{E}[p(y|b_1, b_2, b_3, \sigma)] \\ &= \mathbb{E}[\mathcal{N}(y|b_1 + b_2X_2[n] + b_3X_3[n], \sigma)] \end{aligned} \tag{14.30}$$

We conducted simulation with four different settings of N ($N = 10$, 30, 100, 300). For simplicity, from here we will explain the simulation process assuming that N is fixed. First, the explanatory variable data (that is, $\{X_2[1], \ldots, X_2[N]\}$ and $\{X_3[1], \ldots, X_3[N]\}$) is generated and fixed. These explanatory variables are used

to generate $\vec{Y}$ (that is, $\{Y[1], \ldots, Y[N]\}$) N_{sim} times using random numbers. For each $\vec{Y}$, we do estimation, and calculate GE by combining the true data distribution $q(y)$ with the draws generated from the predictive distribution, which is expressed as Eq. (14.30). Further, for each $\vec{Y}$, we calculate WAIC and LOOIC using the draws from the log-likelihood of the data point $Y[n]$, which is as follows:

$$\log p(Y[n]|\theta) = \log \mathcal{N}(Y[n]|b_1 + b_2 X_2[n] + b_3 X_3[n], \sigma) \tag{14.31}$$

Because WAIC and LOOIC are the approximations of GE, we calculate their biases "WAIC − GE" and "LOOIC − GE" to evaluate them. Finally, we obtain N_{sim} of "WAIC − GE" and "LOOIC − GE", and by visualizing them as boxplots, we can see the distributions of WAIC and LOOIC.

We will introduce how to implement this simulation. First, let us look at the Stan code:

model14-7.stan

```
1   data {
2    int D;
3    int N;
4    matrix[N,D] X;
5    vector[N] Y;
6   }
7
8   parameters {
9    vector[D] b;
10    real<lower=0> sigma;
11   }
12
13   transformed parameters {
14    vector[N] mu = X*b;
15   }
16
17   model {
18    Y[1:N] ~ normal(mu[1:N], sigma);
19   }
20
21   generated quantities {
22    array[N] real yp = normal_rng(mu[1:N], sigma);
23    vector[N] log_lik;
24    for (n in 1:N) {
25     log_lik[n] = normal_lpdf(Y[n] | mu[n], sigma);
26    }
27   }
```

`D` in line 2 is the number of explanatory variables plus 1 for the intercept term (here `D` equals to 3). `N` is the number of data points. Here we focus on explaining `generated quantities` block. Line 22 corresponds to Eq. (14.30) and `yp` is defined to calculate GE. Line 25 corresponds to Eq. (14.31), and `log_lik` is defined to calculate the information criteria.

The R and Python code to implement this simulation, under the number of data points $N = 30$, is as follows:

run-model14-7.R

```
library(dplyr)
library(cmdstanr)
library(KernSmooth)
library(loo)

model <- cmdstan_model('model/model14-7.stan')
N_sim <- 10000
D <- 3
b <- c(1.3, -3.1, 0.7)
SD <- 2.5
EPS <- 1e-12

set.seed(123)
N <- 30
X <- cbind(1, matrix(runif(N*(D-1), -3, 3), N, (D-1)))
Mu <- X %*% b

d_res <- lapply(1:N_sim, function(simID) {
  set.seed(simID)
  Y <- rnorm(N, Mu, SD)
  data <- list(N=N, D=D, X=X, Y=Y)
  mess <- capture.output(
    fit <- model$sample(data=data, seed=123, iter_sampling=2500,
                parallel_chains=4, show_messages=FALSE))
  yp_ms <- fit$draws('yp', format='matrix')
  ge_by_data_point <- sapply(1:N, function(n) {
    dens <- bkde(yp_ms[,n,drop=TRUE])
    f_pred <- approxfun(dens$x, ifelse(dens$y <= EPS, EPS, dens$y),
                yleft=EPS, yright=EPS)
    f_true <- function(y) dnorm(y, mean=Mu[n], sd=SD)
    f_ge  <- function(y) f_true(y)*(-log(f_pred(y)))
    ge <- integrate(f_ge, Mu[n]-6*SD, Mu[n]+6*SD)$value
  })
  ge <- mean(ge_by_data_point)

  log_lik_ms <- fit$draws('log_lik', format='matrix')
  waic  <- waic(log_lik_ms)$waic/(2*N)
  looic <- loo(log_lik_ms)$looic/(2*N)
  data.frame(ge=ge, waic=waic, looic=looic)
 }) %>% bind_rows()
```

run-model14-7.py

```
import cmdstanpy
import numpy as np
import pandas
from scipy import stats
from scipy.stats import norm
from scipy import integrate
import arviz as az
import logging
```

```
 9  from cmdstanpy.utils import get_logger
10  get_logger().setLevel(logging.ERROR)
11  import warnings
12  warnings.filterwarnings('ignore')
13  np.random.seed(123)
14
15  model = cmdstanpy.CmdStanModel(stan_file='model/model14-7.stan')
16  N_sim = 10000
17  D = 3
18  b = np.array([1.3, -3.1, 0.7])
19  SD = 2.5
20  N = 30
21  X = np.hstack((np.ones((N,1)), np.random.uniform(-3, 3, (N, D-1))))
22  Mu = np.matmul(X, b)
23
24  res = []
25  for simID in range(N_sim):
26     np.random.seed(simID)
27     Y = np.random.normal(Mu, SD, N)
28     data = {'N':N, 'D':D, 'X':X, 'Y':Y}
29     fit = model.sample(data=data, seed=123, iter_sampling=2500,
30                 parallel_chains=4, show_progress=False)
31     yp_ms = fit.stan_variable(var='yp')
32     ge_by_data_point = []
33     for n in range(N):
34        f_pred = stats.gaussian_kde(yp_ms[:,n])
35        def f_true(y):
36           return(norm.pdf(y, Mu[n], SD))
37        def f_ge(y):
38           return(f_true(y)*(-f_pred.logpdf(y)))
39        ge, _ = integrate.nquad(f_ge, [[Mu[n]-6*SD, Mu[n]+6*SD]])
40        ge_by_data_point.append(ge)
41     ge = np.mean(np.array(ge_by_data_point))
42
43     log_lik_ms = fit.stan_variable(var='log_lik')
44     waic = - np.mean(az.waic(fit, pointwise=True).waic_i.to_numpy())
45     looic = - np.mean(az.loo(fit, pointwise=True).loo_i.to_numpy())
46     res.append([ge, waic, looic])
47
48  d_res = pandas.DataFrame(res, columns=['ge', 'waic', 'looic'])
```

Line 7 in R (line 16 in Python): `N_sim` is the number of cases of data generation. We need to make sure this number is sufficiently large, or evaluating the information criteria would be hard. Here we used 10,000 for this value.

Line 11 in R (R only): We want to clip the data at 0, so that the estimated predictive distribution does not take a (small) negative value. If we clip exactly at 0, it will throw an error when taking the logarithm, so we set a small value that is very close to 0, which is `EPS`.

Line 14 in R (line 20 in Python): `N` is the number of data points.

Lines 27–29 in R (line 34 in Python): The density function is computed from 10,000 draws from the predictive distribution. Using `bkde` function in `KernSmoot`

package in R (using `gaussian_kde` function in `scipy.stats` library in Python), we can compute the density function in a fast and stable manner.

Line 32 in R (line 39 in Python): GE is calculated by integration. We took mean $\pm$ 6SD as the range of the integral, and this will give a sufficiently close approximation, even though the theoretical range of the integral is $(-\infty, \infty)$.

Lines 37–38 in R (lines 44–45 in Python): WAIC and LOOIC of all the data are calculated using two functions, `waic` and `loo`, both in `loo` package in R (`arviz` library in Python). The calculated results are at the same scale as AIC by default. Here we divided them by $2N$ in R because we think it is preferable to convert the scale of the information criterion to the per data point unit. The calculation of WAIC is equivalently written as follows using the pure R and Python functions:

R

```
1  waic <- function(log_lik_ms) {
2   tr_error_by_data_point <- - log(colMeans(exp(log_lik_ms)))
3   func_var_by_data_point <- colMeans(log_lik_ms^2) - colMeans(log_lik_ms)^2
4
5   waic_by_data_point <- tr_error_by_data_point + func_var_by_data_point
6   waic <- mean(waic_by_data_point)
7   return(waic)
8  }
```

Python

```
1   def waic_func(log_lik_ms):
2    tr_error_by_data_point = -np.log(np.mean(np.exp(log_lik_ms), axis=0))
3    func_var_by_data_point = np.mean(log_lik_ms**2, axis=0) \
4                             - np.mean(log_lik_ms, axis=0)**2
5    waic_by_data_point = tr_error_by_data_point + func_var_by_data_point
6    waic = np.mean(waic_by_data_point)
7    return(waic)
```

Lines 2, 3–4, 5 and 6 in this code corresponds to Eqs. (14.26), (14.27), (14.28), and (14.29), respectively. The average obtained by `colMeans` function in R (`np.mean(x, axis = 0)` function in Python) corresponds to the average over the posterior distribution. The average obtained by `mean` function in line 6 corresponds to taking the summation over the data points and then dividing it by N.

To make it easier to understand, we did not show the parallelization code in **`run-model14-7.R`**. But in practice, we used the parallelization for the data points N and the number of simulations N_{sim} to implement this code. The interested readers can refer the actual code, which is available in our GitHub repository.

To see the characteristics of WAIC and LOOIC, we also did simulation using other models: a logistic regression and a nonlinear regression (Emax model). The other settings are the same as the case of multiple linear regression.

In the logistic regression, we used the following true data distribution:

$$Y[n] \sim \text{Bernoulli}(\text{inv_logit}(0.8 - 1.1X_2[n] + 0.5X_3[n])) \quad n = 1, \ldots, N$$

Then we fitted the following model to the data:

$$Y[n] \sim \text{Bernoulli}(\text{inv_logit}(b_1 + b_2 X_2[n] + b_3 X_3[n])) \quad n = 1, \ldots, N$$
$$b_1, b_2, b_3 \sim \text{Student_t}(6, 0, 5)$$

The parameters b_1, b_2, and b_3 are estimated from the data.

In the nonlinear regression, we used the following true data distribution.

$$Y[n] \sim \mathcal{N}\left(\frac{10X}{2+X}, 0.8\right) \quad n = 1, \ldots, N$$

For X, we used 0.5, 1, 2, 4, 6, 8, 12, 16, 24, and 48, and assigned the data points to each X evenly. Then we fitted the following model to the data.

$$Y[n] \sim \mathcal{N}\left(\frac{b_1 X}{b_2 + X}, \sigma\right) \quad n = 1, \ldots, N$$
$$b_1 \sim \text{Uniform}(0, 20)$$
$$b_2 \sim \text{Uniform}(0, 12)$$

The parameters b_1, b_2, and σ are estimated from the data. For the nonlinear regression model, we also conducted simulation under which the used model did not include the true data distribution. The model used for this assumption is as follows:

$$Y[n] \sim \mathcal{N}(b_1 + b_2 X, \sigma) \quad n = 1, \ldots, N$$

Again, the parameters b_1, b_2, and σ are estimated from the data. We omit the explanation of the code for the logistic regression and the nonlinear regression.

The results of the multiple linear regression, the logistic regression and the nonlinear regression are shown on Fig. 14.8. From these plots, we can learn that when the number of data points N gets larger, both WAIC and LOOIC are asymptotically equivalent to GE. Also, when N is relatively large, there is no large difference between WAIC and LOOIC. In contrast, when N is relatively small, WAIC is closer to GE than LOOIC, and therefore we can conclude that WAIC has a better property.

14.3.3 WAIC in a Hierarchical Model

Here, we will show an example of using a hierarchical model. Generally in a hierarchical model, each data point belongs to one of the groups and each group has its own mean value. Therefore, we need to determine the number of groups G, in addition to the number of the data points. Here, we used three different numbers of groups, $G = 10, 30$ and 100, and three different total number of data points per group, $N_per_G = 2, 5$ and 13. The simulation was conducted using 9 possible

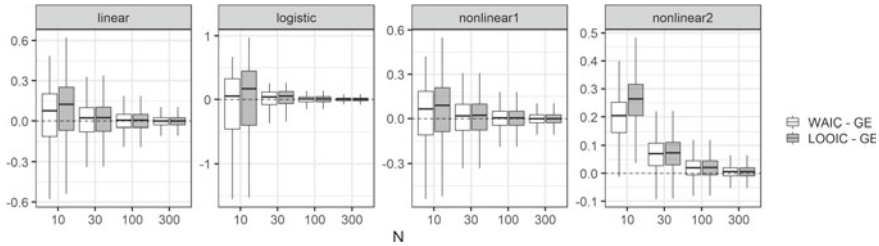


Fig. 14.8 The comparison of WAIC − GE and LOOIC − GE. Showing the case of using a multiple linear regression, a logistic regression, a nonlinear regression model for a nonlinear data, and a multiple linear regression for a nonlinear data (from left to right)

combinations of them. The following true data distribution was used:

$$\begin{aligned} \mu[g] &\sim \mathcal{N}(0, 10) & g &= 1, \ldots, G \\ Y[n] &\sim \mathcal{N}(\mu[n2g[n]], 2) & n &= 1, \ldots, N \end{aligned}$$

where $n2g[n]$ returns the group index that the n-th data point belongs to. For this data, the following model was used:

$$\begin{aligned} \mu[g] &\sim \mathcal{N}\big(\mu_{\text{all}}, \sigma_\mu\big) & g &= 1, \ldots, G \\ Y[n] &\sim \mathcal{N}\big(\mu[n2g[n]], \sigma_y\big) & n &= 1, \ldots, N \\ \mu_{\text{all}} &\sim \text{Student_t}(6, 0, 10) \\ \sigma_y &\sim \text{Student_t}^+(6, 0, 10) \end{aligned}$$

We will estimate the parameters μ_{all}, σ_μ, $\mu[1]$, …, $\mu[G]$, and σ_y from the data.

Note that in the case of a hierarchical model, we can consider multiple types of predictive distributions, depending on the problem setting. Here, we considered the following two types:

- Prediction type 1: we add a new data point to the already existing group g.
- Prediction type 2: we add a new data point to a new group.

As we have emphasized, WAIC corresponds to the predictive distribution. Therefore, WAIC will change when we use a different predictive distribution.

In the first prediction type, the predictive distribution is as follows:

$$p_{\text{pred}}(y) = \mathbb{E}[p(y|\theta)]$$

$$= \mathbb{E}\big[p\big(y|\mu[g], \sigma_y\big)\big] \tag{14.32}$$

$$= \mathbb{E}\big[\mathcal{N}\big(y|\mu[g], \sigma_y\big)\big] \tag{14.33}$$

As we can learn from $p\big(y|\mu[g], \sigma_y\big)$ in Eq. (14.32), a newly generated data point that belongs to the group g is the target of this predictive distribution. Therefore, only the data points that belong to the group g are used in the calculation of WAIC. WAIC of the n-th data point is calculated as follows:

$$\begin{aligned}\mathrm{WAIC}_n &= T_n + V_n \\ &= -\log \mathbb{E}\big[p\big(Y[n]|\mu[g], \sigma_y\big)\big] + \mathbb{V}\big[\log p\big(Y[n]|\mu[g], \sigma_y\big)\big] \\ &= -\log \mathbb{E}\big[\mathcal{N}\big(Y[n]|\mu[g], \sigma_y\big)\big] + \mathbb{V}\big[\log \mathcal{N}\big(Y[n]|\mu[g], \sigma_y\big)\big] \end{aligned} \tag{14.34}$$

To calculate WAIC for all the data points that belong to the group g, we sum over all these data points and then divide the sum by the number of the data points:

$$\mathrm{WAIC}_g = \frac{1}{N_g}\sum_{n=1}^{N_g} \mathrm{WAIC}_n \tag{14.35}$$

where N_g is the total number of data points that belong to the group g, and n is the index of the data point that belongs to the group g. If we need to calculate WAIC when we add one data point for each of all the already existing groups, we take the summation of WAIC_g:

$$\mathrm{WAIC} = \sum_{g=1}^{G} \mathrm{WAIC}_g \tag{14.36}$$

In terms of the second prediction type, the predictive distribution is as follows:

$$p_{\mathrm{pred}}(y) = \mathbb{E}[p(y|\theta)]$$

$$= \mathbb{E}\big[p\big(y|\mu_{\mathrm{all}}, \sigma_\mu, \sigma_y\big)\big] \tag{14.37}$$

$$= \mathbb{E}\left[\int_{-\infty}^{\infty} \mathcal{N}\big(y|\mu, \sigma_y\big)\,\mathcal{N}\big(\mu|\mu_{\mathrm{all}}, \sigma_\mu\big)d\mu\right] \tag{14.38}$$

The mean μ of the new group is not determined and can take any value. Therefore, we express the predictive distribution by marginalizing μ. As we can learn from $p\big(y|\mu_{all}, \sigma_\mu, \sigma_y\big)$ in Eq. (14.37), this predictive distribution does not use each group information, and any newly generated data point is the target of this predictive distribution. Therefore, all the data points are used in the calculation of WAIC. WAIC of the n-th data point is calculated as follows:

$$\begin{aligned}\mathrm{WAIC}_n &= T_n + V_n \\ &= -\log \mathbb{E}\big[p\big(Y[n]|\mu_{\mathrm{all}}, \sigma_\mu, \sigma_y\big)\big] + \mathbb{V}\big[\log p\big(Y[n]|\mu_{\mathrm{all}}, \sigma_\mu, \sigma_y\big)\big] \quad (14.39)\end{aligned}$$

We can take the summation over all the data points and divide the sum by the total number of data points N, in order to calculate WAIC for all the data:

$$\mathrm{WAIC} = \frac{1}{N}\sum_{n=1}^{N} \mathrm{WAIC}_n \quad (14.40)$$

Let us look at how to implement this idea. The Stan code is as follows:

model14-8.stan

```
1   data {
2     int G;
3     int N;
4     vector[N] Y;
5     array[N] int n2g;
6   }
7
8   parameters {
9     real mu_all;
10    real<lower=0> s_mu;
11    vector[G] mu;
12    real<lower=0> s_y;
13   }
14
15   model {
16    mu_all ~ student_t(6, 0, 10);
17    s_mu  ~ student_t(6, 0, 10);
18    mu[1:G] ~ normal(mu_all, s_mu);
19    Y[1:N] ~ normal(mu[n2g], s_y);
20   }
21
22   generated quantities {
23    array[G] real yp1 = normal_rng(mu[1:G], s_y);
24    vector[N] log_lik1;
25    real mup = normal_rng(mu_all, s_mu);
26    real yp2 = normal_rng(mup, s_y);
27    for (n in 1:N) {
28     log_lik1[n] = normal_lpdf(Y[n] | mu[n2g[n]], s_y);
29    }
30   }
```

`N` in line 3 is the total number of data points. We focus on the explanation of `generated quantities` block. `yp1` in line 23 is used for calculating GE of prediction type 1. `log_lik1` in lines 24 and 27–29 is used to calculate the information criteria of prediction type 1. `yp2` in line 26 is used to calculate GE of prediction type 2. In terms of the information criteria of prediction type 2, we need to calculate the log-likelihood using the predictive distribution with a form that includes the integral part (see Eq. (14.38)). We use R and Python to calculate this part, because

estimation for a model containing integrals is still unstable in Stan. The R code for simulation using the fixed group number and the total data points per group ($G = 10$, $N_per_G = 5$) is as follows:

run-model14-8.R

```
library(dplyr)
library(cmdstanr)
library(KernSmooth)
library(loo)

model <- cmdstan_model('model/model14-8.stan')
N_sim <- 10000
Mu_all <- 0
SD_Mu <- 10.0
SD_y <- 2.0
EPS <- 1e-12

G  <- 10
N_per_G <- 5
N  <- N_per_G * G
n2g <- data.frame(n=1:N, g=rep(1:G, times=N_per_G))
g2n_list <- split(n2g$n, n2g$g)

f_marginal <- Vectorize(function(y, mu_all, s_mu, s_y) {
  f <- function(x, y, mu_all, s_mu, s_y) {
    dnorm(x, mean=mu_all, sd=s_mu) * dnorm(y, mean=x, sd=s_y)
  }
  integrate(f, mu_all-6*s_mu, mu_all+6*s_mu,
        y, mu_all, s_mu, s_y)$value
})

d_res <- lapply(1:N_sim, function(simID) {
  set.seed(simID)
  Mu <- rnorm(G, mean=Mu_all,   sd=SD_Mu)
  Y  <- rnorm(N, mean=Mu[n2g$g], sd=SD_y)
  data <- list(N=N, G=G, n2g=n2g$g, Y=Y)
  mess <- capture.output(
    fit <- model$sample(data=data, seed=123, iter_sampling=2500,
                refresh=0, show_messages=FALSE))

  yp1_ms <- fit$draws('yp1', format='matrix')
  ge_by_group <- sapply(1:G, function(g) {
    dens <- bkde(yp1_ms[,g,drop=TRUE])
    f_pred <- approxfun(dens$x, ifelse(dens$y <= EPS, EPS, dens$y),
                yleft=EPS, yright=EPS)
    f_true <- function(y) dnorm(y, mean=Mu[g], sd=SD_y)
    f_ge  <- function(y) f_true(y)*(-log(f_pred(y)))
    ge <- integrate(f_ge, Mu[g]-6*SD_y, Mu[g]+6*SD_y)$value
  })

  log_lik1_ms <- fit$draws('log_lik1', format='matrix')
  ic_by_group <- lapply(1:G, function(g) {
    n_vec <- g2n_list[[g]]
```

```
49      waic  <- waic(log_lik1_ms[,n_vec])$waic/(2*length(n_vec))
50      looic <- loo(log_lik1_ms[,n_vec])$looic/(2*length(n_vec))
51      data.frame(waic=waic, looic=looic)
52    }) %>% bind_rows()
53
54    yp2_ms <- fit$draws('yp2', format='matrix') %>% as.vector()
55    ge_marginal <- {
56      dens <- bkde(yp2_ms)
57      f_pred <- approxfun(dens$x, ifelse(dens$y <= EPS, EPS, dens$y),
58                          yleft=EPS, yright=EPS)
59      f_true <- function(y) f_marginal(y, Mu_all, SD_Mu, SD_y)
60      f_ge  <- function(y) f_true(y)*(-log(f_pred(y)))
61      ge <- integrate(f_ge, -6*SD_Mu-6*SD_y, 6*SD_Mu+6*SD_y)$value
62    }
63
64    d_ms <- fit$draws(c('mu_all', 's_mu', 's_y'), format='df')
65    N_ms <- nrow(d_ms)
66    log_lik2 <- t(sapply(seq_len(N_ms), function(i) {
67      log(f_marginal(Y, d_ms$mu_all[i], d_ms$s_mu[i], d_ms$s_y[i]))
68    }))
69    waic  <- waic(log_lik2)$waic/(2*N)
70    looic <- loo(log_lik2)$looic/(2*N)
71
72    d1 <- data.frame(simID=simID, g=1:G, ge=ge_by_group, ic_by_group)
73    d2 <- data.frame(simID=simID, g=-1,  ge=ge_marginal, waic, looic)
74    rbind(d1, d2)
75  }) %>% bind_rows()
```

Lines 17–23: `f_marginal` defined here corresponds to $p\left(y|\mu_{\text{all}}, \sigma_\mu, \sigma_y\right)$ in Eq. (14.37). The integral part in Eq. (14.38) is computed here. This integral is then used to calculate the true data distribution of prediction type 2 in line 59, and the log-likelihood of prediction type 2 in lines 66–68.

Line 37: GE of prediction type 1 is calculated for each group.

Line 47: WAIC in Eq. (14.35) and LOOIC of prediction type 1 are calculated for each group. For this, we need to know the indices of the data points that belong to each group g. The set of the indices is created in line 17 and is assigned to `g2n_list`.

Line 55: GE of prediction type 2 is calculated.

Lines 69–70: The information criteria of prediction type 2 are computed.

To make it easier to understand, we did not do parallelization in **`run-model14-8.R`**. In practice, we used the parallelization for the number of groups G and the number of simulations N_{sim}. Also, in order to make the integral computation fast and stable, we used `RcppGSL` package. The interested readers can refer the actual code available in our GitHub repository. We omitted the Python code for simulation because the simple implementation of $p\left(y|\mu_{\text{all}}, \sigma_\mu, \sigma_y\right)$ in Eq. (14.37) was too slow. We will need to use Cython etc. to speed up the integration.

Now let us look at the results from this hierarchical model. The result of prediction type 1 is shown in Fig. 14.9. In this case, we can learn that when the number of data points per group is small, such as $N_per_G = 2$, the medians of WAIC−GE and LOOIC−GE do not approach zero even when the number of groups is large. This reflects that we cannot estimate the group mean $\mu[g]$ with a high accuracy, because

there are only $N_per_G = 2$ data points in each group. We can also learn that WAIC is closer to GE and have a better property when the number of data points per group is relatively small.

The result of prediction type 2 is shown in Fig. 14.10. The difference from prediction type 1 is that even when the number of the data points per group increases, the median of WAIC−GE and LOOIC−GE do not approach zero when the number of groups is small. This reflects that even when we increase the number of data points per group, we cannot estimate the parameters that generate the mean of each group (that is, mean μ_{all} and SD σ_μ) with high accuracy, if the number of groups is not large enough. Similar to the results in prediction type 1, we also learn that WAIC is closer to GE and has a better property when the number of groups is relatively small.

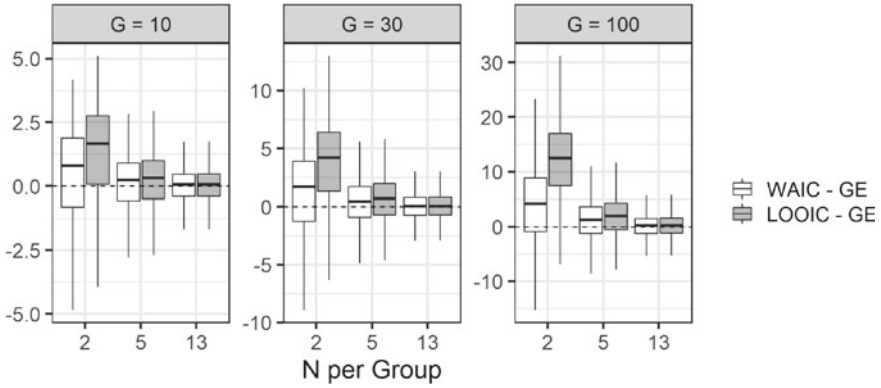


Fig. 14.9 The comparison of WAIC−GE and LOOIC−GE of prediction type 1 in a hierarchical model

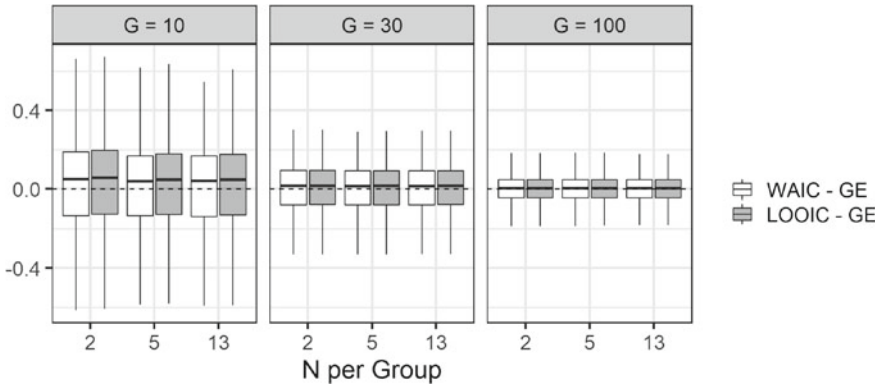


Fig. 14.10 The comparison of WAIC − GE and LOOIC − GE of prediction type 2 in a hierarchical model

From all these results, we can conclude that WAIC has a better approximation of GE compared to LOOIC, especially when the total number of data points or the groups are relatively small (in other words, the weight of each data point or each group is relatively large). On the other hand, we do not see a large difference between WAIC and LOOIC when the amount of data is relatively large. Also, from the viewpoints of the theoretical beautifulness and the computation speed, we recommend using WAIC.

Lastly, as we already mentioned in Sect. 1.7, we need to be careful when using the information criteria. Because it is a value derived from the existing data, it could lead to overfitting when we use them to do model selection. In practice, it is not easy to avoid overfitting when analyzing the complicated real-world data. We should avoid overly focusing on the information criteria without fully exploring models.

Technical Point: WBIC
WBIC is an approximation of a quantity called free energy. In contrast to the fact that WAIC focuses on the predictive distribution, we can think of WBIC as to focus on the distribution of the already obtained data. WBIC tends to choose a simpler model compared to WAIC. There are some arguments that when the true model is within the fitted model space, (W)BIC is asymptotically better in terms of prediction performance and consistency. In this book, we do not recommend (W)BIC because there is no way to know if the true model is within the fitted model space. The readers can refer to (Watanabe, 2018) for more detail about WBIC and free energy.

14.4 Supplementary Information and Exercises

In Sect. 14.2, we gave the number of patterns K in advance. We can also estimate this value using a more advanced model (nonparametric Bayesian model). This will be beyond the scope of this book, but the interested readers can refer (Ferguson, 1973).

In Sect. 14.2, we used the embedding. A large number of recent studies use deep neural network to estimate the embedding. They use a point estimation using deep learning libraries such as `TensorFlow` and `PyTorch` instead of using Stan. These libraries have lots of options for the point estimation, and using these methods can be an alternative option.

14.4.1 Exercises

(1) Simulate LDA and generate data as we did in Sect. 14.2.1. First, generate $\overrightarrow{\theta[n]}$ and $\overrightarrow{\phi[k]}$ using random numbers. Determine the total purchase counts for each customer on your own – this can be generated by a lognormal distribution, or just using constant values. Hint: you could use `rdirichlet` function from `gtools` package if need to use a Dirichlet distribution in R.

(2) Compare the WAIC of **`model8-4.stan`** and **`model9-6.stan`**. Here, we want to predict when we add one new employee to each of the four existing companies (i.e., this corresponds to prediction type 1 in Sect. 14.3.3 and use Eq. (11.36)).

References

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research, 3*, 993–1022.

Ferguson, T. S. (1973). A Bayesian analysis of some nonparametric problems. *The Annals of Statistic*s, 209–230.

Maaten, L. V. D., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research, 9*, 2579–2605.

Moore, D. F. (2016). *Applied survival analysis using R*. Springer.

Titsias, M., & Lawrence, N. D. (2010). Bayesian Gaussian process latent variable model. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 844–851.

Watanabe, S. (2018). *Mathematical theory of Bayesian statistics*. CRC Press.

Appendix
Differences from BUGS Language

- There are `data` block, `parameters` block, and `model` block.
- Variables need to be declared before they are used.
- A wide variety of variable types.
- Cannot use discrete parameters.
- From top to bottom, the code is converted to C++ code and compiled.
- A "`;`" (semicolon) is required at the end of each statement.
- The variable name cannot include ". (dot)", but can include "_ (underscore)".
- Self-assignment (`n = n + 1`) operation is possible in Stan, but not in BUGS.
- If we do not specify a prior distribution, it will automatically be a noninformative prior (a sufficiently wide uniform distribution).
- If the missing values are included in the data of the response variable, an error will occur. We need to handle them separately from the observed data.
- A normal distribution is specified as `normal(mean, SD)` in Stan, while it is specified as `dnorm(mean, inverse of variance)` in BUGS.
- A binomial distribution is specified as `binomial(number of trials, probability)` in Stan, while it is specified as `dbin(probability, number of trials)` in BUGS.
- If the data follows a Poisson distribution, we write `Y ~ poisson` (mean) in Stan and `Y ~ dpois` (mean) in BUGS; in Stan, only an integer value of `Y` are allowed, but in BUGS, the likelihood is calculated even if `Y` is a real value. This is because the Poisson distribution in BUGS has an extended definition using the gamma function.
- In BUGS, a Dirichlet distribution (`ddirch`) can be used only for a conjugate prior distribution of categorical or multinomial distributions, and a Wishart distribution (`dwish`) can be used only for a conjugate prior distribution of a multivariate normal distribution. Also, the parameters that determine the position and shape of these distributions must be fixed values; Stan has no such restrictions at all.
- Stan does not have a logit function. Use the `inv_logit` function instead.
- Stan does not have the `cut` function.

© Springer Nature Singapore Pte Ltd. 2022

K. Matsuura, *Bayesian Statistical Modeling with Stan, R, and Python*,
https://doi.org/10.1007/978-981-19-4755-1