

Junichiro Hagiwara

Time Series Analysis for the State-Space Model with R/Stan

Time Series Analysis for the State-Space Model with R/Stan

Junichiro Hagiwara

Time Series Analysis for the State-Space Model with R/Stan

Junichiro Hagiwara
Graduate School / Faculty of Information Science
and Technology
Hokkaido University
Hokkaido
Japan

ISBN 978-981-16-0710-3 ISBN 978-981-16-0711-0 (eBook)
<https://doi.org/10.1007/978-981-16-0711-0>

© Springer Nature Singapore Pte Ltd. 2021

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

To my parent—Shizuo and Sachiko.

Preface

This book is written as a reference for those who intend to perform time series analysis using the R language.

A time series is the sequentially obtained data, such as temperature or stock price, and its analysis requires consideration of the relation among data. Various approaches can be used to conduct this analysis; this book describes both deterministic and stochastic methods. We explain the deterministic method based on the moving average from the explorative viewpoints. In contrast, we explain the probabilistic method based on the state-space model, which has gained increasing attention in recent years, for conducting detailed analysis. R, which is a free software used to perform statistical analysis, is a language suitable to perform time series analysis. This book uses R libraries and related software as well as R.

Because this book presents not only basic topics but also advanced ones, a wide range of target readers, from beginners to experts in the field of time series analysis, will be interested in reading it. This book contains introductory chapters for readers who are not necessarily familiar with statistics and the R language.

One feature of this book is its emphasis on practice. Therefore, majority of the algorithm derivations, such as the derivation of Kalman filtering, is summarized in the appendices, whereas the main body focuses on explaining the meaning of the formulas and how to develop codes based on the provided examples. This type of presentation is based on my experience in this field. Even after understanding the literature on time series analysis, I cannot easily identify the appropriate manner in which the contents can be translated to actual codes at times. Although overcoming such difficulties on the reader's own is a reasonable education policy, I hope that this book can contribute to mitigate such difficulties.

The structure of this book is as follows. First, Chaps. 1–4 present the introduction, where we explain the fundamentals of statistics and handling of the time series data using R after introducing the basic concept of time series analysis; then, we present an example of time series analysis using the exploratory method. Next, Chaps. 5–9 present the fundamental analysis conducted using the state-space model. Chapters 5 and 6 describe the basics of the state-space models. Chapter 7 introduces the Wiener filter, which provides the batch solution for a linear Gaussian state-space model, from a historical viewpoint. Chapter 8 describes the Kalman filter, which

provides the sequential solution for a linear Gaussian state-space model, using the R library **dlm**. Chapter 9 describes a set of typically used linear Gaussian state-space models for conducting detailed analysis. Finally, Chaps. 10–12 present the applied analysis based on the state-space model. Chapter 10 describes a batch solution for the general state-space model using the Markov chain Monte Carlo (MCMC) method. This chapter uses the **Stan** software and also explains the forward filtering backward sampling (FFBS) technique to improve the estimation accuracy using the Kalman filter as a component. Chapter 11 describes a particle filter, which provides a sequential solution for the general state-space model. This chapter also explains an auxiliary particle filter and the Rao–Blackwellization technique to improve the estimation accuracy using the Kalman filter as a component. Chapter 12 presents a specialized analysis with respect to the data exhibiting a structural change. This chapter examines the approach in which a horseshoe distribution is used.

As for the notation, this book provides supplementary notes in a shaded box with respect to the relevant topics. As for fonts, the typewriter typefaces, such as `plot()`, boldface, such as **dlm**, and italic shapes, such as *DLM*, are used for providing code description, library names, and important indices, respectively. All the codes and data used in this book are available from the support site https://github.com/hagijyun/Time_Series_Analysis_4SSM_R_Stan.

Finally, this book has been completed with the cooperation of many people. Although I am very sorry that the names of all these people cannot be listed, I would like to express some acknowledgements.

This book has been written based on the usage of various types of software. I would like to specially thank the development team of R and **Stan** and professor Giovanni Petris of the University of Arkansas, who is one of the authors of the **dlm** library. Because I have gained valuable insights by participating in the Japanese translation of the **Stan** manual <https://github.com/stan-jp> and Japan research meeting on particle filters <https://pf.sozolab.jp/>, I am also grateful to all those involved in these projects. The content of this book is also related to the syllabus of my doctoral course; therefore, I would like to thank professor Yasutaka Ogawa, professor Takeo Ohgane, and associate professor Toshihiko Nishimura of the Hokkaido University. I would like to thank professor Motohiro Ishida of the Tokushima University for the opportunity to write this book. Professor Norikazu Ikoma of the Nippon Institute of Technology, Mr. Kentaro Matsuura, and Dr. Hiroki Itô helped to review the Japanese draft of this book, which was useful for improving its contents; I would like to thank them for their contribution. I would also like to thank Springer’s coordinators, Mr. Yutaka Hirachi and Mr. Rammohan Krishnamurthy, for their cooperation in reviewing and publishing this book.

I have written this book in the hope that sharing my experience would be helpful to everyone who intends to perform time series analysis. If there are any errors in this book, it is my responsibility alone; in such a case, I would like to make corrections through the support site.

Contents

1	Introduction	1
1.1	What Is Time Series Analysis?	1
1.2	Two Approaches in Time Series Analysis	3
1.3	Use of R	4
1.3.1	Library in R and External Software	4
1.3.2	Code and Data in This Book	5
1.4	Notation in This Book	5
	References	5
2	Fundamentals of Probability and Statistics	7
2.1	Probability	7
2.2	Mean and Variance	8
2.3	Normal Distribution	9
2.4	Relation Among Multiple Random Variables	11
2.5	Stochastic Process	14
2.6	Covariance and Correlation	14
2.7	Stationary and Nonstationary Processes	18
2.8	Maximum Likelihood Estimation and Bayesian Estimation	19
	References	21
3	Fundamentals of Handling Time Series Data with R	23
3.1	Object for Handling Time Series	23
3.2	Handling of Time Information	26
	Reference	27
4	Quick Tour of Time Series Analysis	29
4.1	Confirmation of the Purpose and Data Collection	29
4.2	Preliminary Examination of Data	31
4.2.1	Plot with Horizontal Axis as Time	31
4.2.2	Histogram and Five-Number Summary	33
4.2.3	Autocorrelation Coefficient	37
4.2.4	Frequency Spectrum	38
4.3	Model Definition	43

4.4	Specification of Parameter Values	45
4.5	Execution of Filtering, Prediction, and Smoothing	46
4.6	Diagnostic Checking for the Results	51
4.7	Guideline When Applying the State-Space Model	56
	References	57
5	State-Space Model	59
5.1	Stochastic Model	59
5.2	Definition of State-Space Model	59
5.2.1	Representation by Graphical Model	61
5.2.2	Representation by Probability Distribution	62
5.2.3	Representation by Equation	63
5.2.4	Joint Distribution of State-Space Model	63
5.3	Features of State-Space Model	65
5.4	Classification of State-Space Models	66
	References	68
6	State Estimation in the State-Space Model	69
6.1	State Estimation Through the Posterior Distribution	69
6.2	How to Obtain the State Sequentially	70
6.2.1	A Simple Example	70
6.2.2	Conceptual Diagram of Recursion	74
6.2.3	Formulation of Filtering Distribution	75
6.2.4	Formulation of Predictive Distribution	79
6.2.5	Formulation of the Smoothing Distribution	81
6.3	Likelihood and Model Selection in the State-Space Model	83
6.4	Treatment of Parameters in the State-Space Model	84
6.4.1	When Parameters are Not Regarded as Random Variables	85
6.4.2	When Parameters are Regarded as Random Variables	85
	References	87
7	Batch Solution for Linear Gaussian State-Space Model	89
7.1	Wiener Filter	89
7.1.1	Wiener Smoothing	90
7.2	Example: AR(1) Model Case	92
	References	95
8	Sequential Solution for Linear Gaussian State-Space Model	97
8.1	Kalman Filter	97
8.1.1	Kalman Filtering	98
8.1.2	Kalman Prediction	102
8.1.3	Kalman Smoothing	105
8.2	Example: Local-level Model Case	107
8.2.1	Confirmation of the Purpose and Data Collection	108
8.2.2	Preliminary Examination of Data	108

8.2.3	Model Definition	108
8.2.4	Specification of Parameter Values	110
8.2.5	Execution of Filtering, Prediction, and Smoothing	115
8.2.6	Diagnostic Checking for the Results	121
References		126
9	Introduction and Analysis Examples of a Well-Known Component Model in the Linear Gaussian State-Space Model	129
9.1	Combination of Individual Models	129
9.2	Local-Level Model	130
9.2.1	Example: Artificial Local-Level Model	131
9.3	Local-Trend Model	133
9.4	Seasonal Model	135
9.4.1	Approach from the Time Domain	136
9.4.2	Approach from the Frequency Domain	137
9.4.3	Example: CO ₂ Concentration in the Atmosphere	140
9.5	ARMA Model	150
9.5.1	Example: Japanese Beer Production	152
9.6	Regression Model	160
9.6.1	Example: Nintendo's Stock Price	161
9.6.2	Example: Flow Data of the Nile (Considering the Rapid Decrease in 1899)	165
9.6.3	Example: Family Food Expenditure (Considering Effects Depending on the Days of the Week)	167
9.7	Supplement to Modeling	175
References		176
10	Batch Solution for General State-Space Model	179
10.1	MCMC	179
10.1.1	MCMC Fundamentals	180
10.1.2	Notes on Using the MCMC Method	184
10.2	State Estimation with MCMC	185
10.3	Use of Library	187
10.3.1	Various Libraries	187
10.3.2	Example: Artificial Local-Level Model	188
10.4	Estimation Example in General State-Space Model	194
10.5	Technique for Improving Estimation Accuracy	199
10.5.1	Case in Which the Linear Gaussian State-Space Model is Partially Applicable	200
10.5.2	Example: Artificial Local-Level Model	201
10.5.3	Example: Monthly Totals of Car Drivers in the UK Killed or Injured	207
References		217

11 Sequential Solution for General State-Space Model	219
11.1 Particle Filter	219
11.1.1 Particle Filtering	221
11.1.2 Particle Prediction	225
11.1.3 Particle Smoothing	226
11.2 State Estimation with Particle Filter	229
11.2.1 Example: Artificial Local-Level Model	229
11.2.2 Attention to Numerical Computation	241
11.3 Use of Library	246
11.4 Estimation Example in General State-Space Model	247
11.4.1 Example: A Well-Known Nonlinear Benchmark Model	247
11.4.2 Application of a Particle Filter	250
11.5 Technique for Improving Estimation Accuracy	253
11.5.1 Auxiliary Particle Filter	254
11.5.2 Case in Which the Linear Gaussian State-Space Model is Partially Applicable	266
References	273
12 Example of Applied Analysis in General State-Space Model	277
12.1 Consideration of Structural Change	277
12.2 Approach Using a Kalman Filter (Known Change Point)	279
12.2.1 Time-Invariant Model Studied thus Far	279
12.2.2 Utilizing Prior Information in the Linear Gaussian State-Space Model	279
12.2.3 Numerical Result	280
12.3 Approach Using MCMC (Unknown Change Point)	283
12.3.1 Time-Invariant Model Studied thus Far	283
12.3.2 Use of a Horseshoe Distribution in the General State-Space Model	284
12.3.3 Numerical Result	286
12.4 Approach Using a Particle Filter (Unknown Change Point)	292
12.4.1 Time-Invariant Model Studied thus Far	292
12.4.2 Use of a Horseshoe Distribution in the General State-Space Model	294
12.4.3 Numerical Results	294
12.5 Real-Time Detection for an Unknown Change Point	299
References	300
Appendix A: Library in R and External Software	303
Appendix B: Library dlm	305
Appendix C: Supplement on Conditional Independence in the State-Space Model	313

Contents	xiii
Appendix D: Symbol Assignment in the Linear Gaussian State-Space Model	315
Appendix E: Algorithm Derivation	317
Appendix F: Execution of Particle Filtering with Library	335
Index	343

Chapter 1

Introduction



In this chapter, we first understand what time series analysis based on real data. We then confirm the concept of estimation target in time series analysis. Finally, we briefly describe software such as R and notation used in this book.

1.1 What Is Time Series Analysis?

Time series analysis is a method for analyzing temporally obtained sequential data. We first understand it based on real data. This chapter introduces two types of data that we will treat again in later chapters.

First data are the annual flow of the Nile on the African continent [1]. Nile is available as a built-in data set in R; these data are a series of annual observations from 1871 to 1970 (unit: ($\times 10^8$ m 3)). Figure 1.1 shows the plot.

Looking at Fig. 1.1, we cannot see a particular pattern; irregular fluctuation continues under the influence of annual climatic development. The flooding of the Nile river has a significant influence on the safety and agriculture of the area around it; it is closely related to the development of Egyptian civilization. The Aswan dam was built to enable artificial control of flooding in the modern age [2]. For various purposes, such as study of the civilization's development, safety measures, or agricultural management, it is useful to appropriately understand how much influence the flow rate roughly exercises over the past, present, and future, or what kind of trends exist, as the flow of the Nile river fluctuates every year.

The next data concern the CO₂ concentration in the atmosphere. While co2 (observations at Mauna Loa observatory in Hawaii) is available as a built-in data set in R, this book describes the latest observations in Japan. Specifically, we introduce a series of the monthly mean value (unit: (ppm)) observed from January 1987 at Ryorisaki in Iwate prefecture. These data are available at the Japan meteorological

Fig. 1.1 Data on annual flow of the Nile

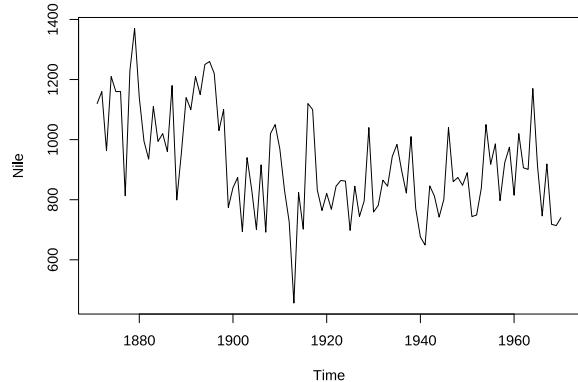
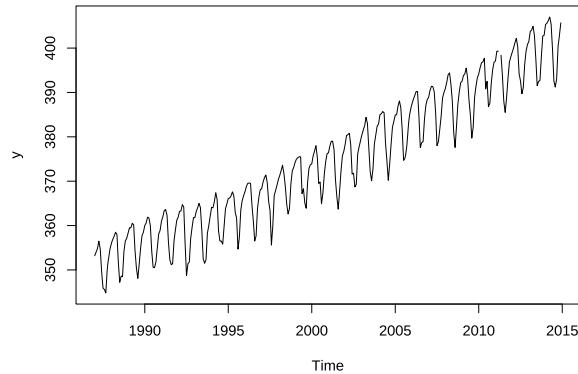


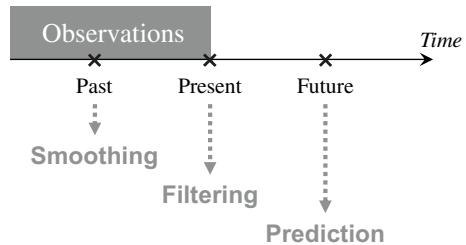
Fig. 1.2 Data on CO₂ concentration in the atmosphere



agency's website http://ds.data.jma.go.jp/ghg/kanshi/obs/co2_monthave_ryo.csv. The author has slightly modified the downloaded data for easy handling using R in this book. Figure 1.2 shows the data through December 2014.

We see that the plot for April 2011 is lacking in Fig. 1.2. This part shows where data could not be acquired as a result of the Great East Japan Earthquake that occurred at the same time. We also see the clear annual cycles in Fig. 1.2. Such a pattern is considered to be caused by the influence of various factors such as the ecosystem of plants. Photosynthesis is typically more active in summer than winter. Furthermore, there is an increasing trend, which has been recognized as one of the factors contributing to global warming in recent years. For various purposes, such as the ecology of plants or measures against global warming, it is useful to appropriately understand how much concentration has roughly occurred during the past, present, and future, or what kind of trends exist, while the CO₂ concentration in the atmosphere fluctuates every year.

According to the above examples, the general significance of time series analysis can now be considered as a work to appropriately understand (estimate) the past, present, and future values regarding events of interest, and based on those results, to obtain knowledge regarding the structure and influence of the events or to consider

Fig. 1.3 Estimation type

countermeasures to them. This book focuses on methods for estimating the past, present, and future values accurately.

These three types of estimation are classified as *smoothing*, *filtering*, and *prediction* (or *forecast*) for the past, present, and future time points, respectively; see Fig. 1.3.

There are further subdivided names for smoothing. *Fixed-interval smoothing* and *fixed-lag smoothing* consider all observations up to the present time point and lag-steps-ahead observations to the estimation time point, respectively. *Fixed-point smoothing* is smoothing focusing on a specific time point.

This book covers only a discrete time series comprising just one type of information (*univariate*) sampled at even intervals. Even though we assume the above constraints, various real data could be treated. Refer to [3–6] for handling the data exceeding these constraints such as the *multivariate* time series, simultaneously comprising multiple types of information.

1.2 Two Approaches in Time Series Analysis

In comparison with pure statistical analysis, time series analysis differs substantially in explicitly considering the relation between data at each time point. Time series data generally show some relation between data at one time point and that at another time point, unlike dice roll-outs. Regarding the data on the annual flow rate in the Nile river in the previous section, for example, we can guess that the value repeats similarly to the previous year's value because the irregular fluctuation just continues without showing any particular pattern. Moreover, regarding the data on atmospheric CO₂ concentration in the previous section, we can guess that there are annual cycles and an increasing trend. Such data correlation in the time direction is useful information and must be considered to improve estimation accuracy. For this purpose, this book describes both deterministic and stochastic methods. The deterministic method estimates the value itself at a particular time point, whereas the stochastic method estimates the uncertainty where various values can be obtained at a particular time point. Figure 1.4 shows a conceptual comparison between the two methods.

Chapter 4 describes the *deterministic method* based on the moving average in the position as exploratory analysis. While Chap. 5 and subsequent discussion will

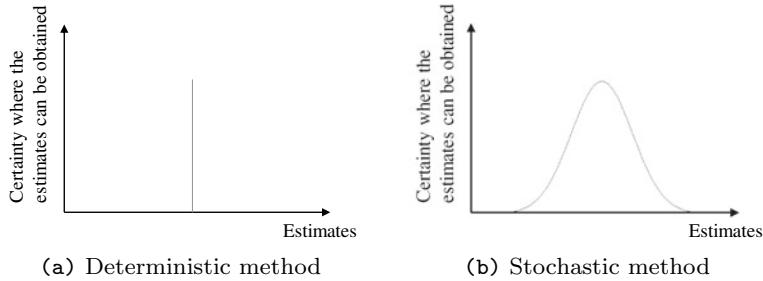


Fig. 1.4 Estimation target at a certain time point

Table 1.1 Major libraries and external software used in this book

Name	Main use in this book
dlm	Kalman filter
Stan	MCMC
pomp, NIMBLE	Particle filter

describe the *stochastic method* based on the state-space model as a deeper form of analysis. Chapter 2 also describes the fundamentals of probability and statistics briefly, which are necessary prerequisite to understand these methods.

1.3 Use of R

This section briefly summarizes information for using R in this book.

1.3.1 Library in R and External Software

The cooperative use of R and library or external software provides a strong analysis environment. Table 1.1 summarizes the major libraries and external software used in this book.

Appendix A summarizes the overview and information regarding installation for the above items. We particularly use **dlm** throughout the book; Appendix B provides a detailed description for the library.

1.3.2 Code and Data in This Book

All the code and data used in this book are available through the support site https://github.com/hagijyun/Time_Series_Analysis_4SSM_R_Stan. These are summarized with the R Markdown format by each chapter. For readers who do not use R Markdown, extract the proper R script in some manner, such as using the function `pur1()` in the library `knitr`. As of this writing, the verification environments are Windows 10 (64 bits), R 3.6.1, RStudio 1.2.5001, `dlm` 1.1-5, `Stan` 2.19.2, `pomp` 2.3, and `NIMBLE` 0.8.0. `Rtools` 3.5.0.4 and `Rcpp` 1.0.2 are also used when a library or external software requires a compiler. Note in advance that the numerical result might change in a different environment.

1.4 Notation in This Book

This section summarizes the mathematical notation used in this book.

Column vectors and matrices are denoted by bold lower-case letters, such as \mathbf{v} , and bold upper-case letters, such as \mathbf{A} , respectively. $^\top$ indicates transposition; therefore, transposed vector and matrix are denoted as \mathbf{v}^\top and \mathbf{A}^\top , respectively. The inverse and determinant of a matrix are denoted by \mathbf{A}^{-1} and $\det(\mathbf{A})$, respectively. A diagonal or block diagonal matrix is denoted as $\text{diag}(\text{element}_1, \text{element}_2, \dots)$, where an element can be a scalar or matrix. The zero vector, zero matrix, and identity matrix are denoted as $\mathbf{0}$, \mathbf{O} and \mathbf{I} , respectively. Blank elements in a matrix are assumed to be 0s. For the complex-value $z = \text{Re}(z) + i \text{Im}(z) = |z| e^{i\arg(z)}$, the real part, imaginary part, absolute value, and argument of z are denoted by $\text{Re}(z)$, $\text{Im}(z)$, $|z|$, and $\arg(z)$, respectively, where $i = \sqrt{-1}$ is the imaginary unit. The complex conjugate of z is denoted by \bar{z} . A prediction or estimation value for a particular variable \cdot is generally denoted by $\hat{\cdot}$. $\lfloor \cdot \rfloor$ indicates the largest integer not exceeding \cdot ; $\lfloor \rfloor$ is called the floor function. The set is represented as the sequence of the element between {and}. Furthermore, the definition, proportional relationship, modulo operation, integration, differentiation, partial differentiation, cumulative sum, and product of sequence are denoted by $::=$, \propto , mod , \int , $', \partial$, \sum , and \prod , respectively.

References

1. Cobb, G.W.: The problem of the Nile: conditional solution to a changepoint problem. *Biometrika* **65**(2), 243–251 (1978)
2. Dumont, H.J. (ed.): *The Nile: Origin, Environments, Limnology and Human Use*. Springer, New York (2009)
3. Harvey, A.C.: *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press, Cambridge (1989)
4. Petris, G., Petrone, S., Campagnoli, P.: *Dynamic Linear Model with R*. Springer, New York (2009)

5. Prado, R., West, M.: Time Series: Modeling, Computation, and Inference. CRC Press, New York (2010)
6. West, M., Harrison, J.: Bayesian Forecasting and Dynamic Models, 2nd edn. Springer, Berlin, Heidelberg (1997)

Chapter 2

Fundamentals of Probability and Statistics



This chapter explains the fundamentals of probability and statistics that are necessary for reading this book. The explanations are limited; see [1, 3, 6, 8] for details.

2.1 Probability

We first describe the most basic aspects of probability.

This book provides a rough definition of a random variable, i.e., an uncertain fluctuating quantity denoted by capital letters such as Y . The random variables in this book are limited to continuous values. Specific examples of random variables are referred to as *realizations* and represented by corresponding lower case letters such as y . The probability where the realization of the random variable Y falls between y_{\min} and y_{\max} is denoted as $P(y_{\min} \leq Y \leq y_{\max})$, whereas the probability where the realization of the random variable Y takes the value y is simply denoted as $P(y)$. According to the axioms of probability, a probability takes a value between 0 and 1 and the total probability is 1. The exhaustive list of probabilities under any combination of realizations for a random variable is referred to as a *probability distribution* (or just distribution). The *probability density function* (or just density function) that comprehensively covers the likelihood to obtain a value around particular realizations is generally expressed as $p()$ and treated in the same sense as the probability distribution. Popular probability distributions often have particular names, and their initial letters are assigned as notation. A quantity that characterizes a probability distribution is referred to as *parameter* and is expressed as θ . Because there might be a plurality of such quantities in general, θ is a vector with multiple elements. While parameters can change over time (*time-varying*), this book basically assumes that they are not time-varying for simplicity (*time-invariant*) until Chap. 12, where the time-varying case will be considered. The simple notation $y \sim p(\cdot)$ indicates that the random variable Y has the probability distribution $p()$; the notation $y \sim p(\theta)$ is also

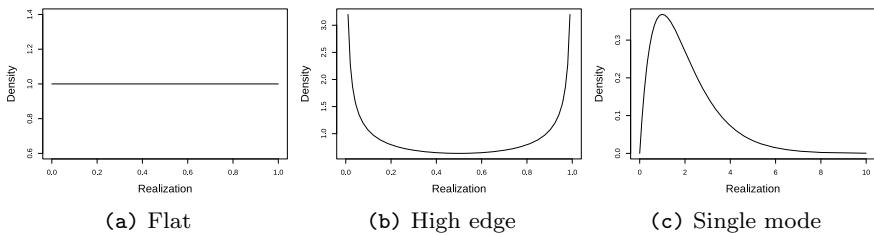


Fig. 2.1 Examples of probability density functions

used with the same meaning having explicit parameter specification. The probability density (or simply density) at point y is expressed as $p(y)$; $p(y; \theta)$ is also used with the same meaning having explicit parameter specification.

Figure 2.1 shows some examples of probability density functions.

Since the total area under the graph of the probability density function corresponds to the total probability, it always becomes 1 for every example, whereas the value on the vertical axis indicates the density and can exceed 1. The parameter definition is different for each example. For example, in Fig. 2.1a, the leftmost and rightmost values in the realizations on the horizontal axis become parameters because the density on the vertical axis takes a flat value and contains no specific information.

2.2 Mean and Variance

This section describes the mean and variance, which are often used in statistics for a probability distribution.

The *mean value* (or *expected value*) for the random variable X is defined as

$$E[X] = \int xp(x)dx. \quad (2.1)$$

The mean value is the centroid of a probability distribution and is often used as one of the representative for the probability distribution.

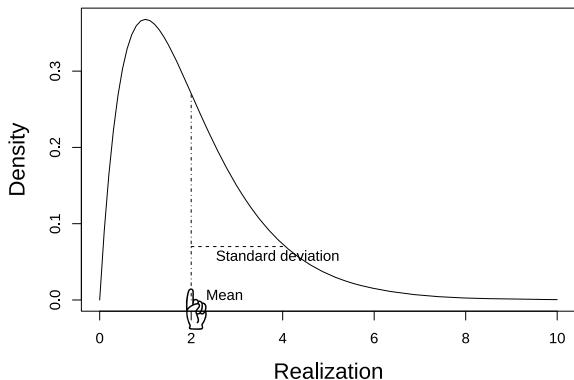
The *variance* for the random variable X is defined as

$$\text{Var}[X] = E[(X - E[X])^2] = \int (x - E[X])^2 p(x)dx. \quad (2.2)$$

The square root of the variance is referred to as the *standard deviation*. The standard deviation indicates how much the probability distribution spreads out from the mean value on average. The inverse of the variance is sometimes referred to as the *precision*.

Figure 2.2 shows an example of the mean and standard deviation for the probability distribution in Fig. 2.1c.

Fig. 2.2 Example of the probability distribution, mean, and standard deviation



Recall that the mean value implies the centroid of the probability distribution; the value corresponds to the point at which we can support a cardboard with the form of the probability distribution in good balance. On the contrary, the standard deviation corresponds to the typical spread of the probability distribution.

We also introduce simple variations in the mean and variance for the convenience of random variable transformation. Let a and b be constants and X and Y be random variables. The following relations hold for the mean and variance:

$$\mathbb{E}[aX + b] = a\mathbb{E}[X] + b, \quad (2.3)$$

$$\text{Var}[aX + b] = a^2\text{Var}[X], \quad (2.4)$$

$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y], \quad (2.5)$$

$$\text{Var}[X + Y] = \text{Var}[X] + \text{Var}[Y] + 2\mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]. \quad (2.6)$$

2.3 Normal Distribution

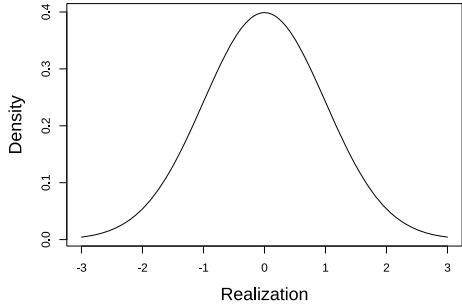
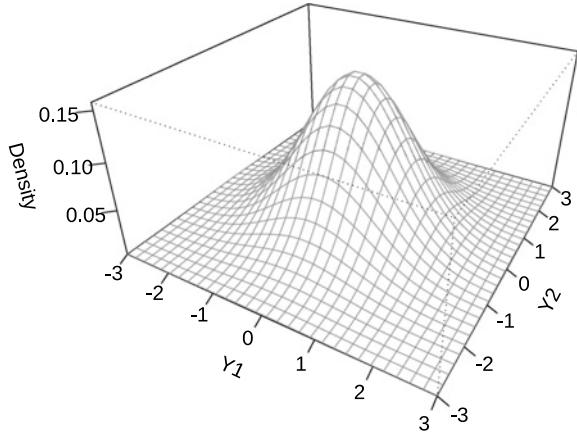
This section describes the well-known *normal distribution* (also called the *Gaussian distribution* in honor of C. F. Gauss) among a number of probability distributions.

The probability density function of the normal distribution with parameters mean μ and variance σ^2 is

$$\mathcal{N}(y; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(y - \mu)^2}{2\sigma^2}\right\}. \quad (2.7)$$

Figure 2.3 shows the normal distribution with mean zero and variance one.

The density of a normal distribution follows a symmetrical bell-curve centered on the mean.

Fig. 2.3 Normal distribution**Fig. 2.4** Two-dimensional normal distribution

The *multidimensional normal distribution* is also defined for the vector of p random variables $\mathbf{y} = (y_1, y_2, \dots, y_p)^\top$ as follows:

$$\mathcal{N}(\mathbf{y}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^p \det(\boldsymbol{\Sigma})}} \exp \left\{ -\frac{1}{2} (\mathbf{y} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{y} - \boldsymbol{\mu}) \right\}. \quad (2.8)$$

In this case, the mean $\boldsymbol{\mu}$ is a p -dimensional column vector, whereas the variance $\boldsymbol{\Sigma}$ is a $p \times p$ matrix. Figure 2.4 shows a two-dimensional normal distribution with mean vector $\mathbf{0}$ and variance identity matrix \mathbf{I} .

Since the distribution contains two random variables, its shape resembles a three-dimensional mountain. Equation (2.8) specifies the relation between random variables; free collection of a random variable sequence that has a normal distribution does not necessarily satisfy Eq. (2.8).

Finally, we describe the essential features of a normal distribution. The result of linear operation on normal random variables become normal random variables again. This characteristic is referred to as *reproducibility* for a normal distribution [4].

2.4 Relation Among Multiple Random Variables

This section describes the relation when there are multiple random variables. To help readers capture the image more easily, we explain the general case after a brief illustration of the concept based on an example.

We first explain the specific example. This example is based on the description in [3, Sect. 2]. At first, suppose a country called Ω .

As shown in Fig. 2.5, Ω has total area one and comprises prefectures A and B, wherein the use of land is divided into factories and fields.

Define

$$P(\text{Factory}, \text{Prefecture A}) \quad (2.9)$$

as the area of prefecture A in the factory region. Now, define the opposite notation in parentheses

$$P(\text{Prefecture A}, \text{Factory}) \quad (2.10)$$

as the area of the factory region in prefecture A. Both notations indicate the same area; therefore, the order in the parentheses can be ignored. The matrix

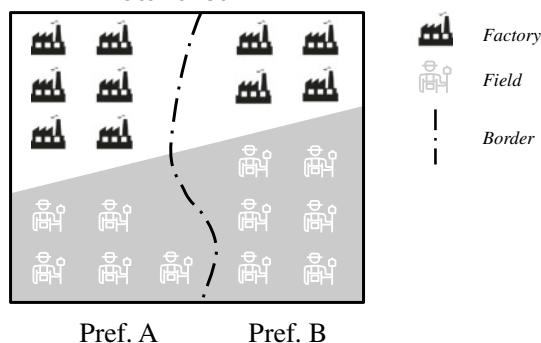
$$\begin{bmatrix} P(\text{Factory}, \text{Prefecture A}) & P(\text{Factory}, \text{Prefecture B}) \\ P(\text{Field}, \text{Prefecture A}) & P(\text{Field}, \text{Prefecture B}) \end{bmatrix}$$

covers all of Ω 's area and is the most detailed expression of it.

Next, define $P(\text{Factory})$ as the total area of the factory region over the prefectures. In addition, define $P(\text{Prefecture A})$ as the total area of prefecture A over the regions. Then, the relation

$$\begin{aligned} P(\text{Factory}) &= P(\text{Factory}, \text{Prefecture A}) + P(\text{Factory}, \text{Prefecture B}) \\ &= \sum_{\text{Prefecture}} P(\text{Factory}, \text{Prefecture}) \end{aligned} \quad (2.11)$$

Fig. 2.5 Ω country



holds. If only the total area of the factory region $P(\text{Factory})$ is known, then the actual area of the factory region in each prefecture is unknown because it can be different in prefectures A and B; rough estimates such as equal division $P(\text{Factory})/2$ are not unavailable.

Next, define $P(\text{Factory} \mid \text{Prefecture A})$ as the ratio of the factory region in prefecture A. In addition, define $P(\text{Prefecture A} \mid \text{Factory})$ as the ratio of prefecture A in the factory region. Then, the relation

$$P(\text{Factory} \mid \text{Prefecture A})P(\text{Prefecture A}) \\ = P(\text{Factory}, \text{Prefecture A}) \quad (2.12)$$

holds. Similarly, the relation

$$P(\text{Prefecture A} \mid \text{Factory})P(\text{Factory}) \\ = P(\text{Prefecture A, Factory}) \quad (2.13)$$

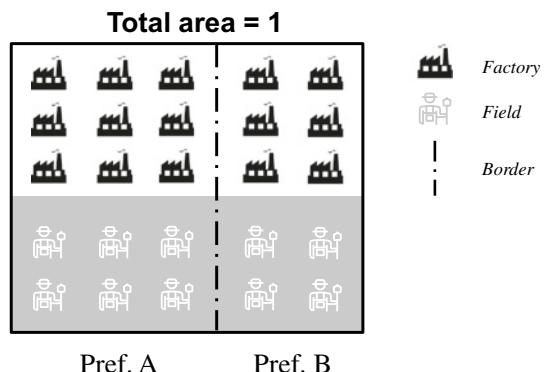
also holds. Recalling that $P(\text{Region}, \text{Prefecture})$ and $P(\text{Prefecture}, \text{Region})$ have the same meaning, we see that the relationship

$$P(\text{Factory} | \text{Prefecture A})P(\text{Prefecture A}) \\ = P(\text{Prefecture A} | \text{Factory})P(\text{Factory}) \quad (2.14)$$

holds from the above equations. Using Eq. (2.14), we can switch the viewpoint for Ω 's area between region and prefecture perspectives.

Furthermore, consider a special case wherein the ratio of the factory and field regions is uniform, as shown in Fig. 2.6.

Fig. 2.6 Ω country (where the ratio of regions is uniform)



Then, the relation

$$\begin{aligned} P(\text{Factory, Prefecture A}) &= P(\text{Factory} \mid \text{Prefecture A})P(\text{Prefecture A}) \\ &= P(\text{Factory})P(\text{Prefecture A}) \end{aligned} \quad (2.15)$$

holds because the ratio of factory and field regions does not change in prefecture A or B. The total area of the factory region $P(\text{Factory})$ corresponds to the ratio of the factory region in the entire country according to the total area one for Ω .

The above description is just an example. Based on this example, we move on to the general explanation below. When there are multiple random variables, the probability that they will coincide is referred to as the *joint probability* (or simultaneous probability). The probability distribution for the joint probability is referred to as the *joint distribution* (or simultaneous distribution). Let X and Y be the random variables; their joint distribution is denoted as

$$p(x, y) \quad (\text{or } p(y, x) \text{ with the same meaning because the order is irrelevant}). \quad (2.16)$$

When multiple random variables exist, their joint distribution is the most expressive and must be considered as the starting point for studying stochastic problems [3, 5].

When there are multiple random variables, the probability of only a specific random variable is referred to as the *marginal probability*. The probability distribution of the marginal probability is referred to as the *marginal distribution*. Let X and Y be the random variables; their marginal distribution is denoted as $p(x)$ (or $p(y)$). They are derived by integrating joint distribution as

$$p(x) = \int p(x, y)dy \quad (\text{or } p(y) = \int p(x, y)dx). \quad (2.17)$$

This operation is also called *marginalization* out. The integral in the marginalization out includes the effects of nuisance random variables on average in advance and achieves their explicit removal from the joint distribution.

When multiple random variables exist, the probability under conditions wherein the specific random variable is given is referred to as the *conditional probability*. The probability distribution for the conditional probabilities is referred to as the *conditional distribution*. Let X and Y be the random variables; the conditional distribution is denoted as $p(x \mid y)$ for a given Y or $p(y \mid x)$ for a given X . Regarding the conditional distribution, the relation

$$p(x \mid y)p(y) = p(x, y) = p(y, x) = p(y \mid x)p(x) \quad (2.18)$$

holds as a result of the *product rule of probability*. From this relation, *Bayes' theorem*

$$p(x \mid y) = \frac{p(y \mid x)p(x)}{p(y)} \quad (\text{or } p(y \mid x) = \frac{p(x \mid y)p(y)}{p(x)}) \quad (2.19)$$

is derived. Bayes' theorem abstractly reflects a change in the viewpoint of probability: let X and Y be random variables for a causal event and observation result, respectively; the theorem can be used for inferring the cause from the result.

We give further supplements to Bayes' theorem. Since the denominator of $p(x | y) = p(y | x)p(x)/p(y)$ is expanded as $p(y) = \int p(x, y)dx = \int p(y, x)dx = \int p(y | x)p(x)dx$, we can see that the left-hand side $p(x | y)$ in Bayes' theorem is theoretically derived from $p(y | x)$ and $p(x)$. These are often referred to as the *posterior distribution* $p(x | y)$, *likelihood* (exactly to say, likelihood under the condition of x) $p(y | x)$, and *prior distribution* $p(x)$. The process of obtaining the posterior distribution considering the likelihood for the prior distribution is also referred to as *Bayesian updating*.

Finally, when the joint distribution can be decomposed into a simple product of each marginal distribution as

$$p(x, y) = p(x)p(y), \quad (2.20)$$

the random variables X and Y are said to be *independent*.

2.5 Stochastic Process

This section describes basic aspects of stochastic processes.

A *stochastic process* is a series of random variables, which exactly corresponds to stochastically interpreted time series data. This book focuses on discrete-time stochastic processes. For a sequence, to express a time point t , a subscript is assigned to the random variable and its realizations such as Y_t and y_t , respectively. When a particular time is not specified in a sequence, we implicitly assume that an observation span is a natural number from one to T , whereas the probability distribution at time point zero corresponds to the prior distribution. Furthermore, we denote the time range by “:”. Therefore, sets for Y and y from time point one to T are expressed as $\{Y_1, \dots, Y_T\} = Y_{1:T}$ and $\{y_1, \dots, y_T\} = y_{1:T}$, respectively; recall the set notation $\{\dots\}$.

2.6 Covariance and Correlation

This section describes covariance and correlation, which are often used as statistics that show the relation between two random variables. Covariance and correlation are among the primary tools for investigating time series and play an important role. While covariance and correlation are essentially equivalent, we first explain covariance.

Let X and Y be random variables; the *covariance* is defined as

$$\text{Cov}[X, Y] = E[(X - E[X])(Y - E[Y])]. \quad (2.21)$$

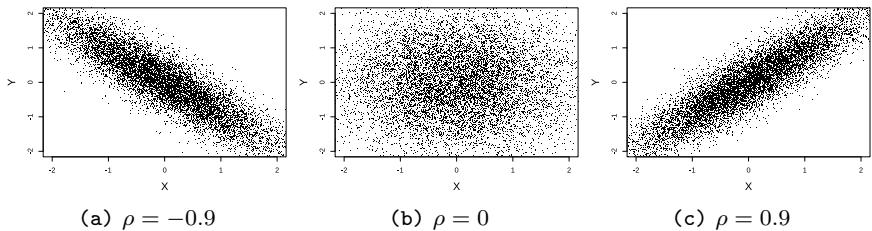


Fig. 2.7 Examples of correlation coefficient

The covariance represents the association between the random variables X and Y . It takes a positive value in the case wherein X is similarly larger (or smaller) than its mean when Y is larger (or smaller) than its mean; a negative value in the case wherein X is conversely smaller (or larger) than its mean when Y is larger (or smaller) than its mean; it takes a zero value when neither case holds. The covariance has already appeared in the above description: the last term in Eq. (2.6) expressing the variance for the sum of the two random variables is just covariance (except for the coefficient two). In addition, the elements in the variance parameter Σ in Eq. (2.8) expressing the multidimensional normal distribution are covariances.

The following normalized covariance is referred to as the *correlation coefficient*:

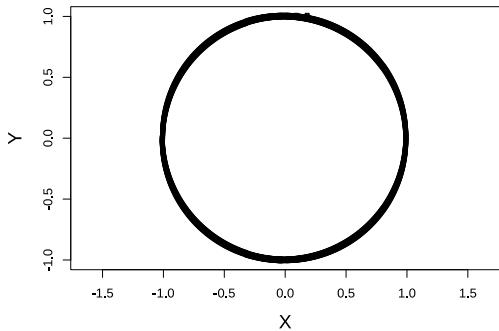
$$\rho_{t,k} = \frac{\text{Cov}[X_t, X_{t-k}]}{\sqrt{\text{Var}[X_t]}\sqrt{\text{Var}[X_{t-k}]}}. \quad (2.22)$$

The correlation coefficient takes a value from -1 to $+1$. Figure 2.7 shows some examples of the correlation coefficient.

The covariance and correlation coefficient are zero if X and Y are independent, but even if the covariance and correlation coefficient are zero, X and Y are not always independent.¹ As an example of being uncorrelated and not independent, consider the coordinate points x and y on the circle as realizations from the random variables X and Y , respectively.

Figure 2.8 shows 1,000 points generated on the circle with center at zero and radius one, whose correlation coefficient takes a value close to zero of $-1.219376e-18$. The reason for obtaining this value is that when X becomes larger than its mean, which is the center of the circle, Y becomes both larger and smaller than its mean, which is the center of the circle, and their effects are precisely same and they mutually cancel out. The correlation coefficient is a quantity just suitable for investigating a linear relation from its definition and is not suitable for understanding a nonlinear relation as in the above example. Thus, when we examine the relations among multiple kinds of data, it is not a good idea to understand the feature between two of them using

¹When X and Y have a two-dimensional normal distribution, however, X and Y are always independent if the correlation is zero. In the “special” case of a random variable that has a normal distribution, uncorrelation and independence are the same.

Fig. 2.8 Points on the circle

only covariance or correlation coefficient. We rather recommend visual inspection with a scatter plot as shown in Fig. 2.8.

Now, let X_t and Y_t be stochastic processes; the *correlation function* is defined as

$$R(t, k) = \text{E}[X_t Y_{t-k}], \quad (2.23)$$

where the time difference k is referred to as the *lag*. Subscripts such as $R_{X,Y}(t, k)$ are added when we explicitly specify the target random variables. The correlation function is equivalent to the covariance between X_t and Y_{t-k} ; Y_{t-k} is the k -shifted Y_t . Similarly to the covariance, the correlation function becomes zero when the stochastic processes X_t and Y_t are independent, but the reverse is not always true.

We move on to the topic of autocovariance and the autocorrelation coefficient. The *autocovariance* of stochastic process X_t is defined as

$$\text{Cov}[X_t, X_{t-k}] = \text{E}[(X_t - \text{E}[X_t])(X_{t-k} - \text{E}[X_{t-k}])]. \quad (2.24)$$

The autocovariance is the covariance between data and a time-shifted version and is a function of time point t and lag k .

The following normalized autocovariance is referred to as the *autocorrelation coefficient*:

$$\rho_{t,k} = \frac{\text{Cov}[X_t, X_{t-k}]}{\sqrt{\text{Var}[X_t]}\sqrt{\text{Var}[X_{t-k}]}}. \quad (2.25)$$

The autocorrelation coefficient is the correlation coefficient between data and a time-shifted version, and is a function of time point t and lag k similar to the autocovariance. The autocorrelation coefficient takes a value from -1 to $+1$ similar to the correlation coefficient; it takes the value $+1$ with $k = 0$. Figure 2.9 shows examples of the autocorrelation coefficient.

Figure 2.9a and b shows examples in cases of data without fluctuation and with a regular cycle, respectively. In each case, the left and right panels show plots for observations with the horizontal axis as time and the autocorrelation coefficient with the horizontal axis as lag, respectively. The right plots are also called correlogram;

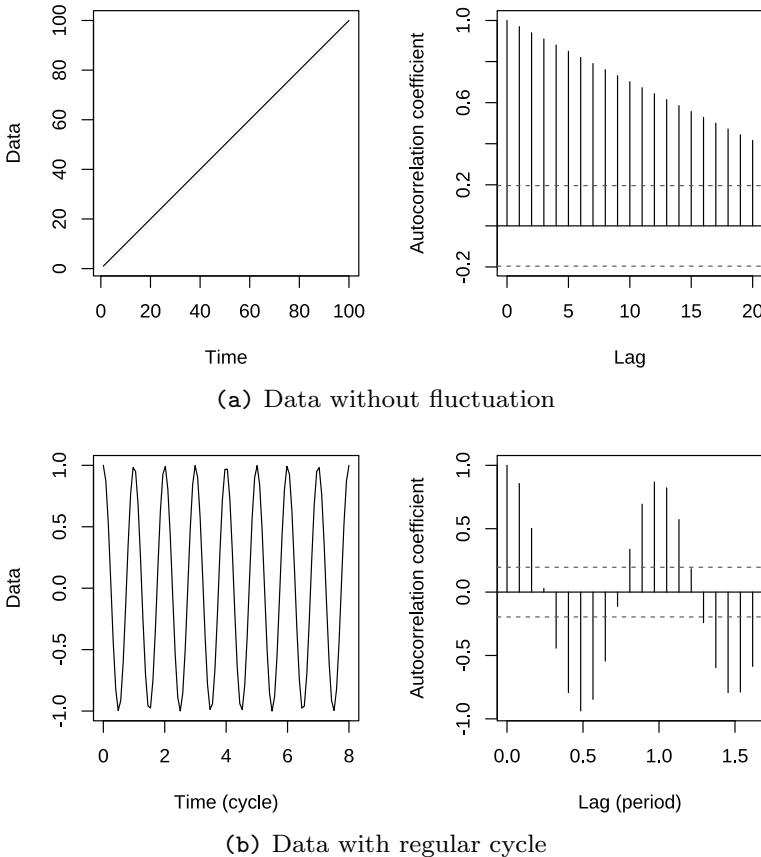


Fig. 2.9 Examples of autocorrelation coefficient

the dotted line band represents the 95% confidence interval when assuming that “autocorrelation coefficients (with $\text{lag} \neq 0$) have a standard normal distribution and are mutually independent.” In the case of data without fluctuation, a similar value continues even if time is shifted, with the result that the autocorrelation coefficient generally does not decrease easily. In case of data with a regular cycle, the autocorrelation coefficient generally becomes high at the time point of an integer multiple of the cycle.

Finally, we provide supplementary notes on calculating statistics such as mean, variance, covariance, and correlation for time series data. It is difficult to sample time series data in the real world repeatedly; we can mostly obtain the data only once. For this reason, when the statistics on time series data are derived, we primarily use calculation through different times while expecting to obtain the same result as that by calculation through different trials.

This property is called ergodicity, where the statistics through different trials coincide with those through different times.

2.7 Stationary and Nonstationary Processes

This section describes stationary and nonstationary stochastic processes.

Stationarity means that a stochastic feature does not change even if time changes. In more detail, when the probability distribution for a stochastic process is invariant independently of time point t , its stochastic process is referred to as *strictly stationary*. On the contrary, when mean, variance, autocovariance, and the autocorrelation coefficient for a stochastic process are invariant independently of time point t , the stochastic process is referred to as *weakly stationary*. Since weak stationarity is preferably used for analyzing actual data, this book expresses weak stationarity as simply *stationary* henceforth. A stochastic process that does not satisfy the stationarity condition results in a *nonstationary* one.

For a stationary stochastic process, the correlation function $R(t, k)$ and autocorrelation coefficient $\rho_{t,k}$ become $R(k)$ and ρ_k , respectively; the argument of these functions is only a lag k . In addition, a stationary stochastic process with mean zero and autocovariance/autocorrelation coefficient zero (except for lag $k = 0$) is referred to as *white noise*.

This book analyzes both stationary and nonstationary stochastic processes; we confirm the difference here. It is sometimes crucial to distinguish stationarity and nonstationarity in time series analysis.

We can roughly guess whether the time series data are stationary through visual inspection for its plot. In Chap. 1, for example, we might not obtain a clear insight into the annual flow data in Fig. 1.1, whereas we can guess the nonstationarity for the CO₂ concentration data in Fig. 1.2; the mean of the data increases as the inspection period proceeds toward recent years.

While we can roughly guess the stationarity in the above way, it is surprisingly difficult to judge it precisely.

If the data generative process is known, we can derive a strict condition for judgment. As an example, suppose the artificial data are following the *AR model* or *autoregressive model*. The AR model specifies a linear relation between current and past data. Among the AR models, the AR(1) model specifies only the relation between current and one-time-past data and is defined as

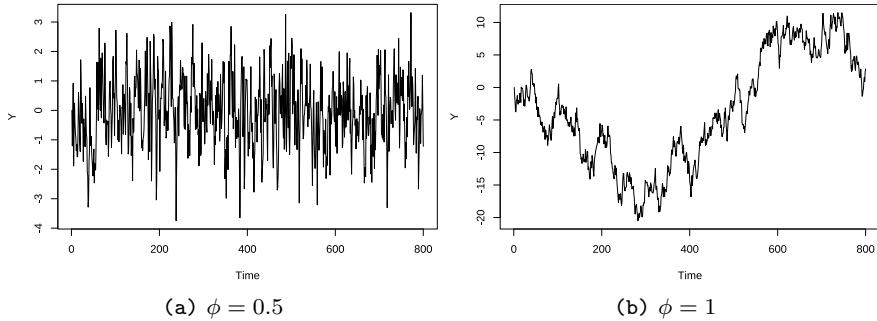


Fig. 2.10 Examples of data generated from the AR(1) model

$$Y_t = \phi Y_{t-1} + \epsilon_t, \quad (2.26)$$

where ϵ_t is white noise. Figure 2.10 shows the plots for data generated from the AR(1) model.

Figure 2.10a shows the case wherein $\phi = 0.5$, and we can see that the value is wandering around zero. On the contrary, Fig. 2.10b shows the case wherein $\phi = 1$, and we cannot see where the value goes. In fact, stationarity for the AR(1) model depends only on the value of ϕ as follows:

$$\begin{cases} \text{Stationary} & |\phi| < 1, \\ \text{Nonstationary} & \text{otherwise.} \end{cases} \quad (2.27)$$

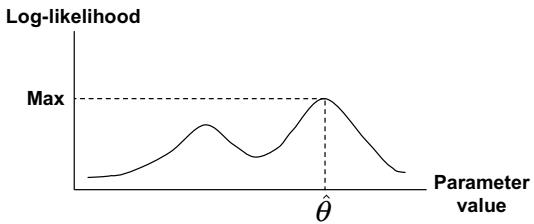
Since the data generative process is rarely known when examining stationarity, there are various test methods for general data (with some assumptions). Among them, the Augmented Dickey–Fuller (ADF) and Phillips–Perron (PP) tests are relatively popular [2, 7].

Finally, we mention the handling of stationary and nonstationary processes in this book. The state-space model that will be described in Chap. 5 and later can treat nonstationary time series data directly. Time series analysis using the state-space model does not primarily determine the stationarity; this book considers that stance as well.

2.8 Maximum Likelihood Estimation and Bayesian Estimation

We have implicitly assumed so far that the parameter θ in a probability distribution is known. The parameter is sometimes known but is generally unknown and must be specified by some means for analysis. This section explains the concept of max-

Fig. 2.11 Image of maximum likelihood estimation



imum likelihood estimation and Bayesian estimation, which are used for parameter specification in this book. Section 6.4 will describe this topic in detail again.

The likelihood of the entire stochastic process Y_t ($t = 1, 2, \dots, T$) is expressed by $p(y_1, y_2, \dots, y_T; \boldsymbol{\theta})$. The likelihood is a single numerical indicator of how an assumed probability distribution fits the realizations. The natural logarithm of the likelihood is referred to as the *log-likelihood* as

$$\ell(\boldsymbol{\theta}) = \log p(y_1, y_2, \dots, y_T; \boldsymbol{\theta}). \quad (2.28)$$

While the primary purpose for taking logarithms is that likelihood generally takes a very small value, it is also often useful from a computational perspective. The *maximum likelihood method* specifies the parameter $\boldsymbol{\theta}$ where the likelihood is maximized. Function $\log()$ is a monotonically increasing function with the result that the maximization of likelihood has the same meaning as that of log-likelihood. Let $\hat{\boldsymbol{\theta}}$ be the estimator for parameters generally; the *maximum likelihood estimation* is expressed as

$$\hat{\boldsymbol{\theta}} = \operatorname{argmax}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}). \quad (2.29)$$

Figure 2.11 shows this image.

The maximum likelihood estimation is not necessarily almighty but it has many preferred properties; it becomes a prevalent method for parameter estimation in the stochastic model.

In the above description, the parameter is not considered to be a random variable. The frequentists generally take such a viewpoint, and parameter specification through maximum likelihood estimation is one approach in frequentism. On the other hand, there is a means of considering parameter as a random variable. Bayesians generally take this viewpoint, and a parameter is estimated through *Bayesian estimation* based on Bayes' theorem under Bayesianism. There is no rule whether a parameter is to be considered as a random variable. For simplicity, this book does not regard a parameter as a random variable until Chap. 9; we will also introduce cases in which a parameter is considered as a random variable in Chap. 10 and later.

References

1. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, Berlin, Heidelberg (2006)
2. Hamilton, J.D.: Time Series Analysis. Princeton University Press, Princeton (1994)
3. Hiraoka, K., Hori, G.: Probability and Statistics for Computer Science. Ohmsha Ltd, Tokyo (2009). [in Japanese]
4. Hirota, K., Ikoma, N.: Mathematics of Stochastic Process. Asakura Publishing Co., Ltd, Tokyo (2001). [in Japanese]
5. Iwanami Data Science Publication Committee (ed.): Iwanami Data Science, vol. 1. Iwanami Shoten, Publishers, Tokyo (2015). [in Japanese]
6. Klenke, A.: Probability Theory: A Comprehensive Course. Universitext. Springer, London (2013)
7. Okimoto, T.: Time Series Analysis for Economic and Finance Data. Asakura Publishing Co., Ltd, Tokyo (2010). [in Japanese]
8. Statistics Section, Department of Social Sciences, Collage of Arts and Sciences, The University of Tokyo (ed.): Introduction to Statistics. University of Tokyo Press (1991). [in Japanese]

Chapter 3

Fundamentals of Handling Time Series Data with R



This chapter provides useful information for treating time series data with R, such as `ts` and `Date` class objects.

3.1 Object for Handling Time Series

The primary object suitable for handling time series with R is `ts` class object, where `ts` stands for “time series.” This object is structured by adding time information to data. While various extension classes have been introduced recently [1], this book uses the basic `ts` class.

An example of `ts` class object is R’s built-in dataset `Nile`, which is used for drawing Fig. 1.1 in Chap. 1. The code is as follows:

Code 3.1

```
1 > # <<Data on annual flow of the Nile>>
2 >
3 > # Display data contents
4 > Nile
5 Time Series:
6 Start = 1871
7 End = 1970
8 Frequency = 1
9  [1] 1120 1160  963 1210 1160 1160  813 1230 1370 1140  995  935 1110  994 1020
10 [16]  960 1180  799  958 1140 1100 1210 1150 1250 1260 1220 1030 1100  774  840
11 [31]  874  694  940  833  701  916  692 1020 1050  969  831  726  456  824  702
12 [46] 1120 1100  832  764  821  768  845  864  862  698  845  744  796 1040  759
13 [61]  781  865  845  944  984  897  822 1010  771  676  649  846  812  742  801
14 [76] 1040  860  874  848  890  744  749  838 1050  918  986  797  923  975  815
15 [91] 1020  906  901 1170  912  746  919  718  714  740
16 >
17 > # Plot the data
18 > plot(Nile)
```

If we type `Nile` in the console, its contents are displayed. In the display, `Start`, `End`, and `frequency` indicate the data start time (1871), end time (1970), and cycle (one year), respectively. We can draw a `ts` class object using the function `plot()`.

We use the function `ts()` to create a `ts` class object. The following R code, which is used for drawing Fig. 1.2 in Chap. 1, illustrates this:

Code 3.2

```

1 > # <<Data on CO2 concentration in the atmosphere>>
2 >
3 > # Load the data
4 > Ryori <- read.csv("CO2.csv")
5 >
6 > # Cast the data to ts class, truncating it by December 2014
7 > y_all <- ts(data = Ryori$CO2, start = c(1987, 1), frequency = 12)
8 > y <- window(y_all, end = c(2014, 12))
9 >
10 > # Display data contents
11 > y
12      Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov   Dec
13 1987 353.2 354.0 354.8 356.5 354.7 349.6 345.9 345.6 344.8 350.0 352.8 354.6
14 1988 356.0 356.9 357.6 358.5 358.0 351.7 347.2 348.6 348.5 354.1 356.5 357.1
15 1989 358.3 359.5 359.5 360.5 360.2 354.3 350.4 348.1 351.3 354.9 357.7 358.5
16 1990 360.0 360.6 361.9 361.7 360.0 354.2 350.6 350.5 351.8 355.0 358.1 358.9
17 1991 361.0 362.0 363.4 363.6 362.3 356.3 352.3 351.2 351.4 356.5 359.0 361.2
18
19 ... Ignore the interim display ...
20
21 2010 394.2 395.6 396.7 396.9 397.7 390.8 392.5 386.8 387.6 392.1 395.1 396.8
22 2011 397.1 399.3 399.3    NA 398.4 393.8 388.0 385.5 388.8 393.1 396.8 398.0
23 2012 399.0 400.0 401.2 402.2 400.3 394.2 392.6 389.7 391.1 396.3 399.3 400.8
24 2013 401.5 403.7 404.1 404.9 402.9 397.7 391.5 392.5 392.7 398.1 402.7 402.9
25 2014 405.3 405.6 406.2 407.0 405.4 399.8 392.6 391.2 393.1 400.4 402.9 405.7
26 >
27 > # Plot the data
28 > plot(y)

```

In the above code, the author has slightly modified the downloaded data from the Japan meteorological agency's website http://ds.data.jma.go.jp/ghg/kanshi/obs/co2_monthave_ryo.csv for easy handling with R and saved it as `CO2.csv`. Data are read from this file using the function `read.csv()` and stored temporarily in the data frame `Ryori`, whose variable name is derived from the reading for the name of the observation location. `Ryori$CO2` is then converted to a `ts` class object using the function `ts()`, and the result is assigned to `y_all`. The arguments `start = c(1987, 1)` and `frequency = 12` set the start time and cycle to January 1987 and 12 months, respectively. Even if `end` is not specified, it is automatically calculated from the data length. The function `window()` is then used to extract a portion of the `ts` class object `y_all`. The argument `end = c(2014, 12)` removes the data after December 2014, and the result is assigned to `y`. If we type `y` in the console, its contents are displayed similarly as `Nile`. From the contents of `y`, we can see that April 2011 corresponds to `NA`, which is not available, indicating *missing observations*.

Next, we introduce function handling of multiple `ts` class objects. This code is as follows:

Code 3.3

```

1 > # <<Function handling multiple ts class objects>>
2 >
3 > # Artificial second time series (shifting back the start time of the Nile to five
4 <- years ago)
5 > Nile2 <- ts(Nile, start = 1866)
6 >
7 > # Unite multiple time series
8 > ts.union(Nile, Nile2)
9 Time Series:
10 Start = 1866
11 End = 1970
12 Frequency = 1
13 Nile Nile2
14 1866 NA 1120
15 1867 NA 1160
16 1868 NA 963
17 1869 NA 1210
18 1870 NA 1160
19 1871 1120 1160
20 1872 1160 813
21 1873 963 1230
22 1874 1210 1370
23 1875 1160 1140
24 ... Ignore the interim display ...
25
26 1961 1020 746
27 1962 906 919
28 1963 901 718
29 1964 1170 714
30 1965 912 740
31 1966 746 NA
32 1967 919 NA
33 1968 718 NA
34 1969 714 NA
35 1970 740 NA
36 >
37 > # Artificial second time series (doubling observations of the Nile)
38 > Nile2 <- 2 * Nile
39 >
40 > # Plot multiple time series in the same area
41 > ts.plot(cbind(Nile, Nile2), ylab = "y", lty = c("solid", "dashed"))

```

In the above code, we first create an artificial secondary time series `Nile2` forcing the start time point of `Nile` back to five years earlier. The function `ts.union()` is useful when we unite multiple `ts` class objects into one while considering their time information. From the result with `ts.union()`, we can see that this function supplements `NA` at the time point where data to be united do not exist.

The function `ts.plot()` is also useful when we draw multiple `ts` class objects in the same area. For example, we now create an artificial secondary time series `Nile2` doubling the values of `Nile`. When using `ts.plot()`, we gather multiple `ts` class objects with the function `cbind()`. Figure 3.1 shows the result. We can see that Fig. 3.1 draws two time series data in the same area.

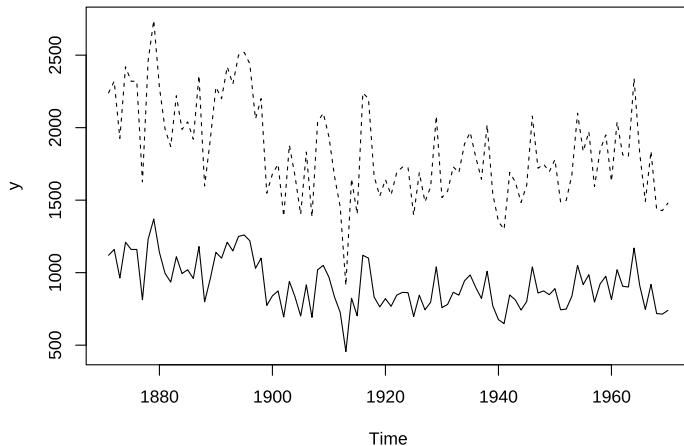


Fig. 3.1 Drawing multiple ts class objects

3.2 Handling of Time Information

This section describes how to handle time information contained in a ts class object.

The following code shows the example for Nile.

Code 3.4

```

1 > # <<Time information on ts class object>>
2 >
3 > # Start time point, end time point, and frequency
4 > tsp(Nile)
5 [1] 1871 1970     1
6 >
7 > # A sequence of time values
8 > time(Nile)
9 Time Series:
10 Start = 1871
11 End = 1970
12 Frequency = 1
13   [1] 1871 1872 1873 1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885
14   [16] 1886 1887 1888 1889 1890 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900
15   [31] 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914 1915
16   [46] 1916 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928 1929 1930
17   [61] 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1943 1944 1945
18   [76] 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960
19   [91] 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970

```

The function `tsp()` can extract or set a combination of start time, end time, and frequency in a ts class object. The function `time()` can extract a sequence of time values whose time information is the same as that of the original ts class object.

The ts class object contains only simple time information; hence, we sometimes use the other time information in combination. The primary objects suitable for handling time information in R are *Date class* and *POSIXct/POSIXlt* class objects.

The Date class is suitable for representing the year, month, and day, whereas the POSIXct/POSIXlt class is suitable for the year, month, day, hour, minute, and second. While various extension classes have been introduced recently [1], this book uses and describes only the basic Date class.

The Date class object internally holds an elapsed daily counter since January 1, 1970. We confirm the functions handling the Date class object in the following code:

Code 3.5

```

1 > # <<Date class object>>
2 >
3 > # Cast the strings to Date class object
4 > day <- as.Date("2000-01-01")
5 >
6 > # Confirm the structure
7 > str(day)
8 Date[1:1], format: "2000-01-01"
9 >
10 > # Generate consecutive Date class objects
11 > days <- seq(from = as.Date("2000-01-01"), to = as.Date("2000-01-31"),
12 +                 by = "day")
13 >
14 > # Confirm contents
15 > days
16 [1] "2000-01-01" "2000-01-02" "2000-01-03" "2000-01-04" "2000-01-05" "2000-01-06"
17 [7] "2000-01-07" "2000-01-08" "2000-01-09" "2000-01-10" "2000-01-11" "2000-01-12"
18 [13] "2000-01-13" "2000-01-14" "2000-01-15" "2000-01-16" "2000-01-17" "2000-01-18"
19 [19] "2000-01-19" "2000-01-20" "2000-01-21" "2000-01-22" "2000-01-23" "2000-01-24"
20 [25] "2000-01-25" "2000-01-26" "2000-01-27" "2000-01-28" "2000-01-29" "2000-01-30"
21 [31] "2000-01-31"
22 >
23 > # Extract information about the day of the week
24 > weekdays(days)
25 [1] "Saturday" "Sunday"   "Monday"    "Tuesday"   "Wednesday" "Thursday"
26 [7] "Friday"    "Saturday" "Sunday"    "Monday"    "Tuesday"   "Wednesday"
27 [13] "Thursday" "Friday"   "Saturday"  "Sunday"    "Monday"    "Tuesday"
28 [19] "Wednesday" "Thursday" "Friday"   "Saturday" "Sunday"    "Monday"
29 [25] "Tuesday"   "Wednesday" "Thursday"  "Friday"   "Saturday" "Sunday"
30 [31] "Monday"
```

The function `as.Date()` can cast a string into the Date class object, and the result is assigned to `day` in the above example. The function `str()` can examine the structure of an arbitrary object. We see that the result of `str(day)` is a Date class object with a length 1. The function `seq()` can generate consecutive Date class object. Also, the function `weekdays()` extracts the information regarding the day of the week from a Date class object.

Reference

1. R Supporters: Perfect R. Gijutsu-Hyohron Co., Ltd. (2017). [in Japanese]

Chapter 4

Quick Tour of Time Series Analysis



This chapter first confirms the flow of time series analysis based on examples and then moves on to a detailed explanation. We explain the analysis through the following flow:

1. confirmation of the purpose and data collection
2. preliminary examination of data
3. model definition
4. specification of parameter values
5. execution of filtering, prediction, and smoothing
6. diagnostic checking of the results
7. return to 1.

While the above flow applies to every time series analysis, an actual analysis would require multiple trial and error. At the end of this chapter, we mention the preliminarily viewpoint regarding what kind of state-space model solution is applicable to the data analyzed in this chapter; the details of the state-space model will be described through the following chapters.

4.1 Confirmation of the Purpose and Data Collection

When starting a time series analysis, we first confirm its purpose as a specified goal. If there are sufficient data, we use those. If data are insufficient or do not exist, we acquire data through the efficiently planned simulation, survey, or measurement.

Table 4.1 Purpose of analysis in this chapter

Purpose	Data
1. Remove noise as much as possible from the data while capturing data	(a), (b) (c), (d)
2. Determine the sudden change in past values	(a)
3. Predict future values	(b)

However, acquiring real-world time series data is often one time only. It appears to the author that detailed systematic plans for acquiring time series data, such as “experimental design” [9], are not studied often. In any case, while we proceed with the analysis based on the acquired data, we might need to change our initial specified goal during the investigation; it is quite common that the initial idea differs from what we actually need. In such a case, we must reexamine the analysis by appropriately reviewing the purpose.

This chapter resets the analysis purpose for the already acquired data. The target data are the following four items:

- (a) annual flow of the Nile
- (b) CO₂ concentration in the atmosphere
- (c) quarterly gas consumption in the UK
- (d) artificially generated data from a nonlinear model.

Item (a) is the R’s built-in dataset `Nile`, as mentioned in Chap. 1. Item (b) is the data published by the Japan meteorological agency, as mentioned in Chap. 1. Item (c) is R’s built-in dataset `UKgas`, which is observed from the first quarter of 1960 to fourth quarter of 1986 (unit: (million therms)). Item (d) is the generated data for an example in this book and are summarized in R’s binary file `BenchmarkNonLinearModel.RData`. We will confirm the details later in Sect. 11.4.1. Among the above mentioned items, we have prior knowledge of the item (a): the historical fact that Aswan (low) dam was built in 1899 and flow rapidly declined at that time. Table 4.1 shows the purpose of analyzing each dataset in this chapter.

The following description proceeds with an analysis based on the above data and purposes.

4.2 Preliminary Examination of Data

First, we must preliminarily examine the target data for the analysis. Such an examination must be performed without any assumptions and various viewpoints should be considered.

The author considers that inspecting the following plots and statistics are valid:

- plot with horizontal axis as time
- histogram and five-number summary
- autocorrelation coefficient
- frequency spectrum.

Needless to say, plots and figures play a particularly important role in this preliminary examination. Human visual ability can be excellent and thus must be utilized for preliminary examination.

Becoming acquainted with the rough characteristics of data helps us to determine how to deal with them. Typical examples are as follows:

- If we know a time point at which the data characteristics change drastically, we split the data at that point and analyze each period independently.
- If observations have an extensive range or a rapid increasing/decreasing trend, we take logarithms of data for easier analysis.
- If observations are likely to contain mistaken values, we eliminate such data from the analysis target.
- If the autocorrelation among the data is very low, we do need not perform a time series analysis.
- If we recognize a clear temporal pattern or periodicity in data, we must reflect its features in the model.

Through such preliminary examinations, we arrange the data for easy handling and establish an appropriate analysis plan. The following descriptions show how we actually perform such an examination.

4.2.1 Plot with Horizontal Axis as Time

We first confirm the plot with the horizontal axis representing time. This code is as follows.

Code 4.1

```

1 > # <<Plot with the horizontal axis as time>>
2 >
3 > # Preprocessing regarding plot (saving the default settings for plot, then changing
4 <- them)
5 > oldpar <- par(no.readonly = TRUE)
6 > par(mfrow = c(2, 2), oma = c(0, 0, 0, 0), mar = c(5, 3.5, 2, 1), mgp = c(2.5, 1, 0))
7 >
8 > # (a) Annual flow of the Nile
9 > plot(Nile)
10 > title(sub = "(a)", line = 4, family = "mono")
11 >
12 > # (b) CO2 concentration in the atmosphere
13 > # Load the data
14 > Ryori <- read.csv("CO2.csv")
15 >
16 > # Cast the data to ts class, truncating it by December 2014
17 > y_all <- ts(data = Ryori$CO2, start = c(1987, 1), frequency = 12)
18 > y <- window(y_all, end = c(2014, 12))
19 > y_CO2 <- y
20 > plot(y_CO2)
21 > title(sub = "(b)", line = 4, family = "mono")
22 >
23 > # (c) Quarterly gas consumption in the UK
24 > plot(UKgas)
25 > title(sub = "(c)", line = 4, family = "mono")
26 >
27 > # (d) Artificially-generated data from a nonlinear model
28 > load("BenchmarkNonLinearModel.RData")
29 > y_nonlinear <- ts(y)
30 > plot(y_nonlinear)
31 > title(sub = "(d)", line = 4, family = "mono")
32 >
33 > # Post-processing regarding plot
34 > par(oldpar)

```

The function `plot()` in R can depict the data universally. Figure 4.1 shows the result.

Figure 4.1a shows that the irregular fluctuation continues. Despite prior knowledge, we apparently cannot find a sudden change in 1899 owing to the fluctuation. It also seems that there is no particular temporal pattern. If there is no pattern, we believe that a value might repeat similarly one to the previous year's value except for irregular fluctuation. Figure 4.1b then shows a clear pattern with annual cycles and an increasing trend. While Fig. 4.1c also shows a clear pattern with annual cycles and an increasing trend the same as in Fig. 4.1b, we can further see that the level and range of observation rapidly increases as time proceeds from approximately the early 1970s. Finally, Fig. 4.1d shows that the irregular fluctuation continues. While high and low values appear to arise alternately, we cannot recognize the cycle explicitly.

Next, we consider the topic of data transformation. While such a transformation is a technique for making data more accessible, we must not abuse it if it is not really required. There are various transformations for time series analysis, *differences* and *logarithmic transformation* are commonly used. The (first) difference is defined as $y_t - y_{t-1}$ for observations y_t . This transformation often used to convert a nonstationary time series data set into a stationary one. This book utilizes the state-space

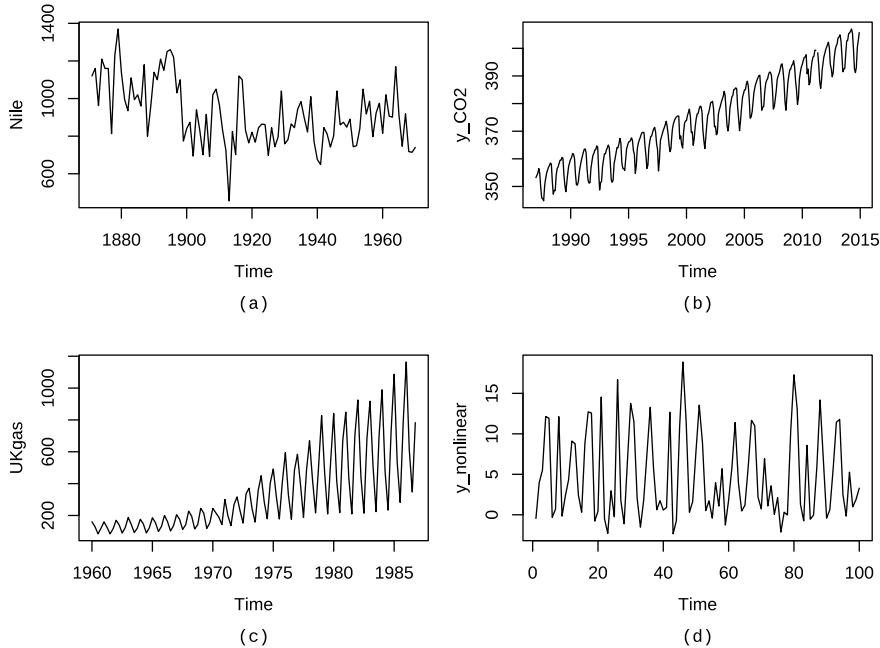


Fig. 4.1 Plot with horizontal axis as time

model for detailed analysis. This model can handle nonstationary time series directly; hence, we decide not to use the difference in this book.

On the contrary, logarithmic transformation is defined as $\log(y_t)$ for observations y_t and might be used in this book. If we expect that the data increase or decrease rapidly, i.e., exponentially, the logarithm of the data makes the trend linear, suppresses the data range reasonably, and results in easy handling. It is known empirically that some economic and natural phenomena have exponential trends. Since the data (c) in this example have such a possibility, we decide to apply the logarithmic transformation. Figure 4.2 shows the result.

Figure 4.2 shows that this operation transforms the rapid and oscillating trend into a linear and stable one; we can recognize the features of the data more clearly. We assume the logarithmically transformed (c) in the following explanations.

4.2.2 Histogram and Five-Number Summary

Next, we confirm the histogram and five-number summaries. The *histogram* is a graph that shows the frequency distribution of data. The *five-number summary* comprises of the minimum, 25%, median, 75%, and maximum values for data. They are confirmed using the following code.

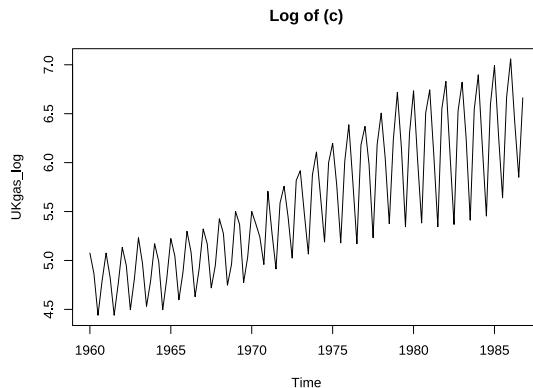


Fig. 4.2 Log quarterly gas consumption in the UK

Code 4.2

```

1 > # <<Histogram and five number summary>>
2 >
3 > # Preprocessing regarding plot (saving the default settings for plot, and then
4 >   changing them)
5 > oldpar <- par(no.readonly = TRUE)
6 > par(mfrow = c(2, 2), oma = c(0, 0, 0, 0), mar = c(5, 3.5, 2, 1), mgp = c(2.5, 1, 0))
7 >
8 > # (a) Annual flow of the Nile
9 > hist(Nile, main = "", xlab = "Data Value")
10 > title(sub = "(a)", line = 4, family = "mono")
11 > summary(Nile)
12   Min. 1st Qu. Median Mean 3rd Qu. Max.
13   456.0    798.5   893.5  919.4   1032.5  1370.0
14 >
15 > # (b) CO2 concentration in the atmosphere
16 > hist(y_CO2, main = "", xlab = "Data Value")
17 > title(sub = "(b)", line = 4, family = "mono")
18 > summary(y_CO2)
19   Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
20   344.8    361.9   373.6  374.7   388.1   407.0      1
21 >
22 > # (c) Quarterly gas consumption in the UK
23 > hist(UKgas_log, main = "", xlab = "Data Value")
24 > title(sub = "(c)", line = 4, family = "mono")
25 > summary(UKgas_log)
26   Min. 1st Qu. Median Mean 3rd Qu. Max.
27   4.440    5.032   5.398  5.579   6.152   7.060
28 >
29 > # (d) Artificially-generated data from a nonlinear model
30 > hist(y_nonlinear, main = "", xlab = "Data Value")
31 > title(sub = "(d)", line = 4, family = "mono")
32 > summary(y_nonlinear)
33   Min. 1st Qu. Median Mean 3rd Qu. Max.
34   -2.3505  0.4621  2.4743  4.8389  8.9232 18.8767
35 >
36 > # Post-processing regarding plot
37 > par(oldpar)

```

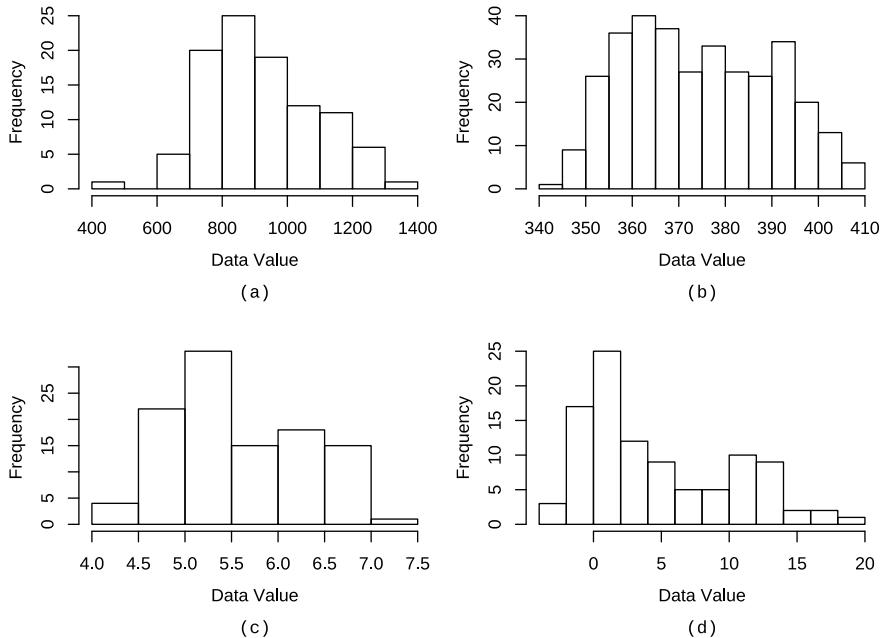


Fig. 4.3 Histogram

The functions `hist()` and `summary()` in R can derive the histogram and five-number summary, respectively. As suggested by its name, the function `summary()` is suitable for checking the outline feature of the data. Figure 4.3 shows the result of the histogram.

While each case in Fig. 4.3 has its own shape, we cannot recognize extreme outliers. Therefore, we assume that there are no outliers or errors in the data. Incidentally, we see that data (b) contain one NA from its five-number summary. That indicates a missing observation in April 2011 resulting from the Great East Japan Earthquake, as confirmed in Sect. 3.1.

Next, we confirm dealing with *outliers* and *missing observations*. While outliers can occur through contamination by error, their occurrence occasionally contains essential information. If an outlier can be attributed to error, we must assign R's NA and eliminate it from the analysis. If an outlier cannot necessarily be attributed to an error, whether we include it in the analysis or not depends on the case. The following presents an artificial example showing a case wherein outliers occur as an error. Figure 4.4 shows a plot and histogram of a series of Tokyo's daily average temperatures in August of a particular year, for which we intentionally mix a one-day error.

From the figure, we can recognize the existence of an outlier; a temperature over 60 °C is extremely unlikely. In such a case, we cannot trust the corresponding data and must replace it with NA. However, since real data sometime hide errors that are

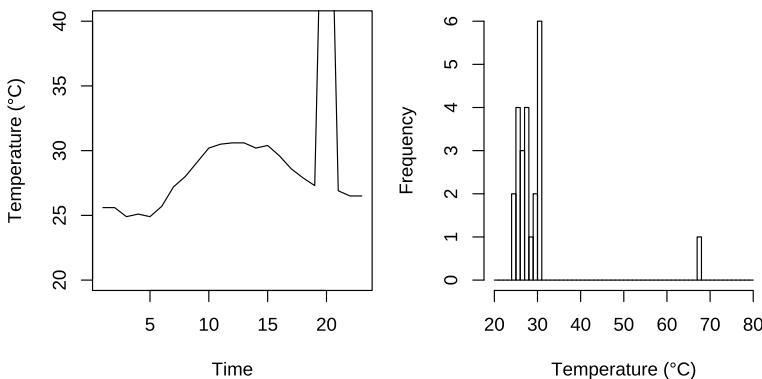


Fig. 4.4 Tokyo's daily average temperature in August of a particular year (a one-day error is introduced intentionally)

difficult to recognize, we must carefully examine the data using common sense and healthy skepticism.

Next, regarding missing observations, we must assign NA to it in R. However, some R functions cannot handle NA correctly; hence, we have to compensate for it preliminarily with similar values in such a case. Although such a process actually relies on less theoretical background and should be avoided, we supplement the missing observation of the data (b) in advance because there are restrictions on the function of the Holt–Winters method to be used later in this chapter. The following code shows such a process.

Code 4.3

```

1 > # <<Compensation for NA>>
2 >
3 > # (b) CO2 concentration in the atmosphere
4 >
5 > # Identification of NA location
6 > NA.point <- which(is.na(y_CO2))
7 >
8 > # Compensation for NA: arithmetic mean before and after the NA
9 > y_CO2[NA.point] <- (y_CO2[NA.point-1] + y_CO2[NA.point+1]) / 2

```

While there are no rules for compensating for NA, we know that NA does not exist at both the beginning and end of data; hence, we decide to assign the arithmetic mean before and after the NA for the values. In the state-space model to be described in Chap. 5 and later, we can handle the missing observations theoretically by supplementing with the predicted value; this feature is one of the attractions of the state-space model.

4.2.3 Autocorrelation Coefficient

Next, we confirm the autocorrelation coefficient. The code is as follows.

Code 4.4

```

1 > # <<Autocorrelation coefficients>>
2 >
3 > # Preprocessing regarding plot (saving the default settings for plot, then changing
4 >   them)
5 > oldpar <- par(no.readonly = TRUE)
6 > par(mfrow = c(2, 2), oma = c(0, 0, 0, 0), mar = c(5, 3.5, 2, 1), mgp = c(2.5, 1, 0))
7 >
8 > # (a) Annual flow of the Nile
9 > acf(Nile, main = "")
10 > title(sub = "(a)", line = 4, family = "mono")
11 >
12 > # (b) CO2 concentration in the atmosphere
13 > acf(y_CO2, main = "")
14 > title(sub = "(b)", line = 4, family = "mono")
15 >
16 > # (c) Quarterly gas consumption in the UK
17 > acf(UKgas_log, main = "")
18 > title(sub = "(c)", line = 4, family = "mono")
19 >
20 > # (d) Artificially-generated data from a nonlinear model
21 > acf(y_nonlinear, main = "")
22 > title(sub = "(d)", line = 4, family = "mono")
23 >
24 > # Post-processing regarding plot
25 > par(oldpar)
```

The function `acf()` in R can derive the autocorrelation coefficient. Figure 4.5 shows the autocorrelation coefficient results.

The entire data from (a) to (d) in Fig. 4.5 create the appearance that the autocorrelation coefficient does not take extremely small values everywhere. Thus, we proceed with time series analysis.

The autocorrelation coefficient sometimes becomes large for a particular lag. This phenomenon implies periodicity: for example, Fig. 4.5c shows notable peaks at the lag of every integer multiple of the cycle. Recall that data (b) similarly show periodicity in its plot with the horizontal axis representing time, but it is difficult to recognize such features in the corresponding Fig. 4.5b; while the coefficient becomes high at the lag of every integer multiple of the cycle, it is buried in the full set of high values. The reason for this is that the increasing trend is more dominant than the periodic fluctuation in these data; see Fig. 4.1b again. In such a case, we can more clearly recognize the periodicity through the frequency spectrum. This book also confirms the frequency spectrum regarding periodicity. In addition, we cannot recognize clear periodicity for data (a) and (d).

Finally, we reflect the information on the above-confirmed periodicity in the model.

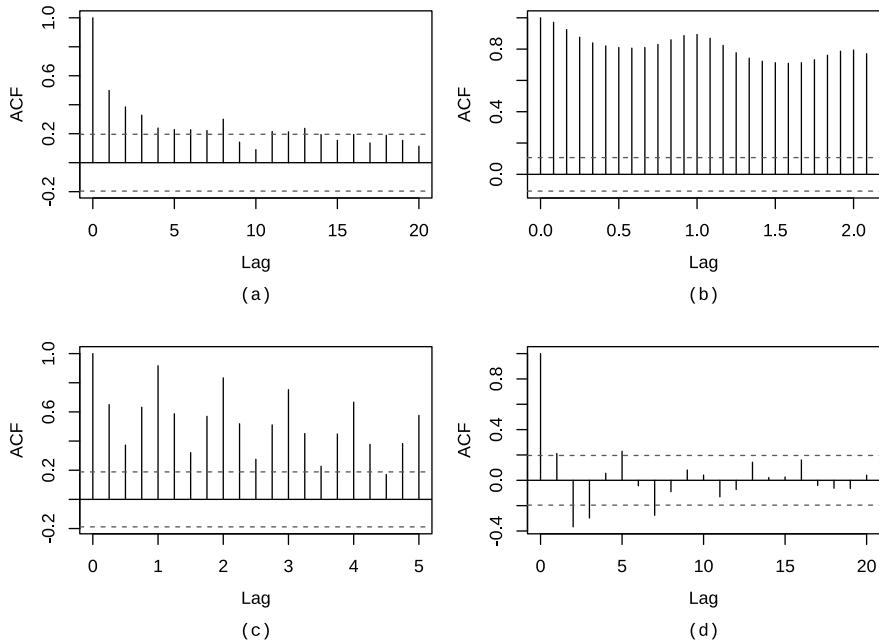


Fig. 4.5 Autocorrelation coefficient

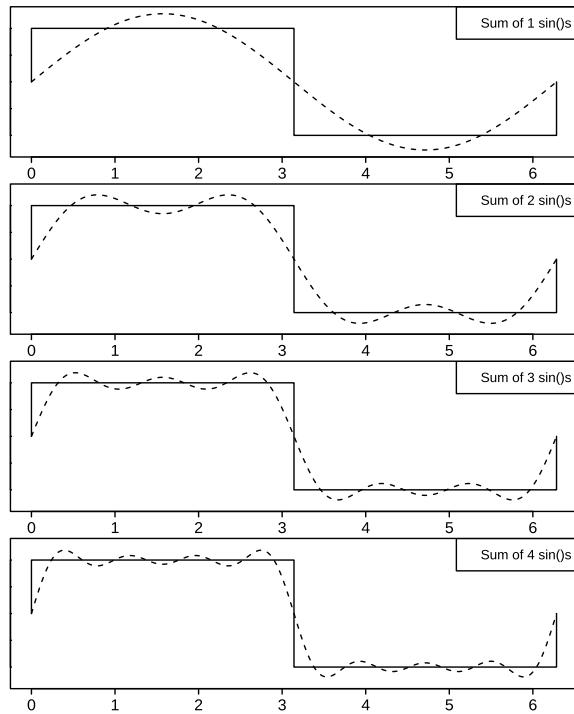
4.2.4 Frequency Spectrum

The *frequency spectrum* can be derived by converting data in the *time domain* to the *frequency domain*; it is useful for confirming periodicity. The representations in the time and frequency domains have a paired relation. While an exemplary time domain graph has the horizontal and vertical axes representing time and data value, respectively, an exemplary frequency domain graph has the horizontal and vertical axes as *frequency* (reciprocal of the cycle) and magnitude of individual frequency components, respectively. We can also confirm the periodicity using the autocorrelation coefficient in the time area. However, when a dominant trend buries a relatively small periodicity, the autocorrelation coefficient may have difficulty in distinguishing them. Consequently, this book explains the use of the frequency spectrum as well. We can recognize periodicity more readily by confirming not only the autocorrelation in the time domain but also spectrum in the frequency domain.

4.2.4.1 Fundamentals on Frequency

In general, any periodic time series data y_t can be considered as the sum of trigonometric functions with various periods. This is known as *Fourier series* expansion as established by J. B. J. Fourier. Figure 4.6 shows an example of Fourier series expansion.

Fig. 4.6 Example of Fourier series expansion



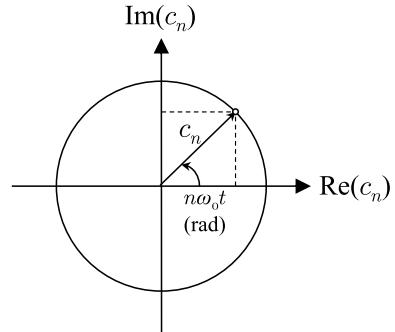
In the above example, the rectangular signal is approximated using sine functions; we see that the approximation accuracy increases with an increase in the number of synthesized sine functions.

Fourier series can be expressed using sine and cosine functions, and more briefly using complex values. Specifically, the Fourier series expansion of observations y_t with fundamental period T_0 (s) can be expressed using complex values as

$$y_t = \sum_{n=-\infty}^{\infty} c_n e^{jn\omega_0 t}, \quad (4.1)$$

where $\omega_0 = 2\pi f_0$ (rad/s) is the fundamental *angular frequency*, and $f_0 = 1/T_0$ (Hz) is the fundamental frequency. Equation (4.1) is also referred to as the complex Fourier series expansion of y_t , and c_n is the complex *Fourier coefficient*. The absolute value of the complex Fourier coefficient $|c_n|$ indicates the extent to which the data contain individual periodical components. In addition, Fig. 4.7 can be helpful in determining why the coefficient c_n takes a complex value; we regard a periodical component as a rotation operation that keeps on spinning at the relevant speed.

Fig. 4.7 Periodical component as a rotation operation



$\text{Re}[c_n]$ and $\text{Im}[c_n]$ (the real and imaginary parts of the complex Fourier coefficient c_n , respectively) represent the horizontal axis (in-phase) component and vertical axis (quadrature) component in the rotation operation, respectively. The complex Fourier coefficient c_n can be derived explicitly as

$$c_n = \frac{1}{T_0} \int_{-T_0/2}^{T_0/2} y_t e^{-in\omega_0 t} dt. \quad (4.2)$$

Expanding the fundamental period T_0 in Eq. (4.2) to infinity, we can apply the equation to arbitrary data. Such an extension is called a *Fourier transform*; the Fourier transform of y_t is generally expressed as $\mathcal{F}[y_t]$, which is also known as the frequency spectrum. The absolute value of the frequency spectrum indicates to what extent an individual periodical component is contained in the data, similar to the complex Fourier coefficient. Even though the Fourier transform converts the data in the time domain to that in the frequency domain, this operation only changes the viewpoint for the data; it never increases or decreases the information in the data. The reason for converting the data domain from time to frequency is primarily for the benefit of human beings, i.e., to provide easier understanding of periodicity, as in the above description, or to facilitate formulation and numerical calculation.

Normalizing the energy $|\mathcal{F}[y_t]|^2$ of the frequency spectrum by dividing by time and taking its limit, we can get *power spectrum*. Time series analysis in the frequency domain is often explained from the viewpoint of the power spectrum, which has a connection with the theoretical description. However, this book deals with the conversion into the frequency domain as an exploratory method; hence, we explain the Fourier transform from the viewpoint of a pure frequency spectrum.

4.2.4.2 Frequency Domain Transform

We now convert the data under analysis in the time domain to that in the frequency domain. This code is as follows.

Code 4.5

```

1 > # <<Frequency domain transform>>
2 >
3 > # User-defined function drawing frequency spectrum (tick: marking points on
4 > # horizontal axis, unit: quantity between time points)
5 > plot.spectrum <- function(dat, lab = "", sub = "",
6 > # Frequency domain transform of data
7 > dat_FFT <- abs(fft(as.vector(dat)))
8 +
9 > # Preparation for display setting about horizontal axis (frequency)
10 > data_len <- length(dat_FFT)
11 > freq_tick <- c(data_len, tick, 2)
12 +
13 > # Plot data in the frequency domain
14 > plot(dat_FFT/max(dat_FFT), type = "l", main = "",
15 > ylab = "|Standardized frequency spectrum|", ylim = c(0, y_max),
16 > xlab = sprintf("Frequency (1/%s)", lab), xlim = c(1, data_len/2), xaxt = "n")
17 > title(sub = sub, line = 4, family = "mono")
18 > axis(side = 1, at = data_len/freq_tick * unit + 1,
19 > labels = sprintf("1//%d", freq_tick), cex.axis = 0.7)
20 +
21 >
22 >
23 > # Preprocessing regarding plot (saving the default settings for plot, then changing
24 > # them)
25 > oldpar <- par(no.readonly = TRUE)
26 > par(mfrow = c(2, 2), oma = c(0, 0, 0, 0), mar = c(5, 3.5, 2, 1), mgp = c(2.5, 1, 0))
27 >
28 > # (a) Annual flow of the Nile
29 > plot.spectrum(Nile, lab = "Year", sub = "(a)")
30 >
31 > # (b) CO2 concentration in the atmosphere
32 > plot.spectrum(y_CO2, lab = "Month", sub = "(b)", tick = c(12, 6))
33 >
34 > # (c) Quarterly gas consumption in the UK
35 > plot.spectrum(UKgas_log, lab = "Month", sub = "(c)", tick = c(12, 6), unit = 3)
36 >
37 > # (d) Artificially-generated data from a nonlinear model
38 > plot.spectrum(y_nonlinear, lab = "Time point", sub = "(d)")
39 >
40 > # Ignore the display of following codes

```

The function `fft()` performs discrete-time *fast Fourier transform (FFT)* in R. Figure 4.8 shows the resulting plots in the frequency domain. We ignore the display in the negative frequency region on the horizontal axis for ease of view. We also normalize the absolute value of the frequency spectrum by its maximum value on the vertical axis to understand the contribution of the frequency component easily.

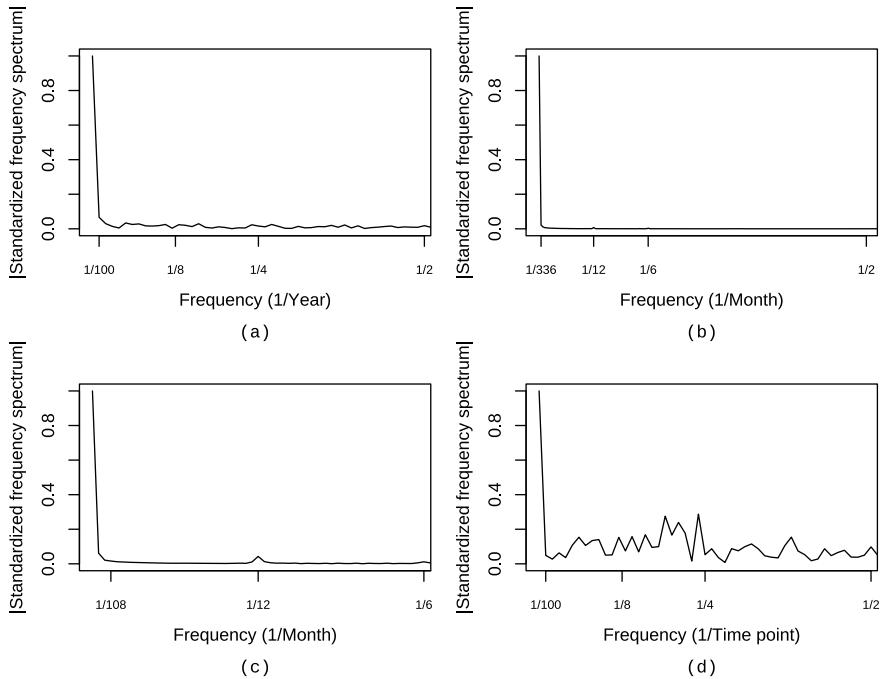


Fig. 4.8 Frequency domain transform

While every case in Fig. 4.8 has its own shape, we can see that the left edge takes the most significant value. The left edge in the plot essentially means the entire level excluding fluctuation; the cycle in the time domain becomes longer as the horizontal axis moves to the left. Consequently, the significant value at the left edge can be ignored for confirming periodicity. Figure 4.9 shows the plots whose vertical axes are scaled up for detailed confirmation.

Every case in Fig. 4.9 has its own result, and the part where the vertical axis indicates large values suggest that the corresponding periodic components are strongly contained. The data (a) appear to contain large values everywhere; hence, we cannot recognize a specific periodicity. The data (b) appear to contain large values at twelve and six months. Since the six-month cycle is incidentally derived from the twelve-month cycle comprising convex and concave halves, we can consider that only the twelve-month period is genuine. The data (c) also have large values at twelve and six months, but we consider that only the twelve-month period is genuine for the same reason. Since the data (d) appear to contain large values everywhere, we cannot recognize a specific periodicity.

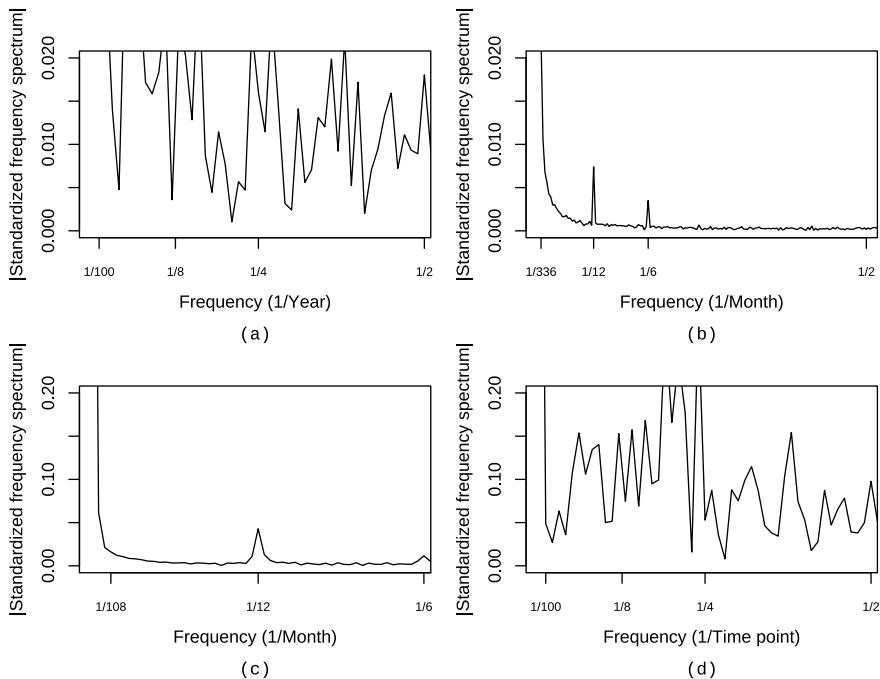


Fig. 4.9 Frequency domain transform (with scaled-up vertical axis)

As mentioned above, we can recognize periodicity more properly by confirming not only the autocorrelation in the time domain but also spectrum in the frequency domain.

4.3 Model Definition

The model is a simple structure expressing the data characteristics. It is common to think of combining simpler structures in defining a model; this is also known as *decomposition* of time series data. This chapter explicitly considers “*level + trend + season*” as a starting point since this is widely recognized as a fundamental decomposition of time series data. Additionally, the remainders resulting from subtracting these estimated components from the original time series data are referred to as *residuals*. Figure 4.10 shows an example of decomposition and residuals for time series data.

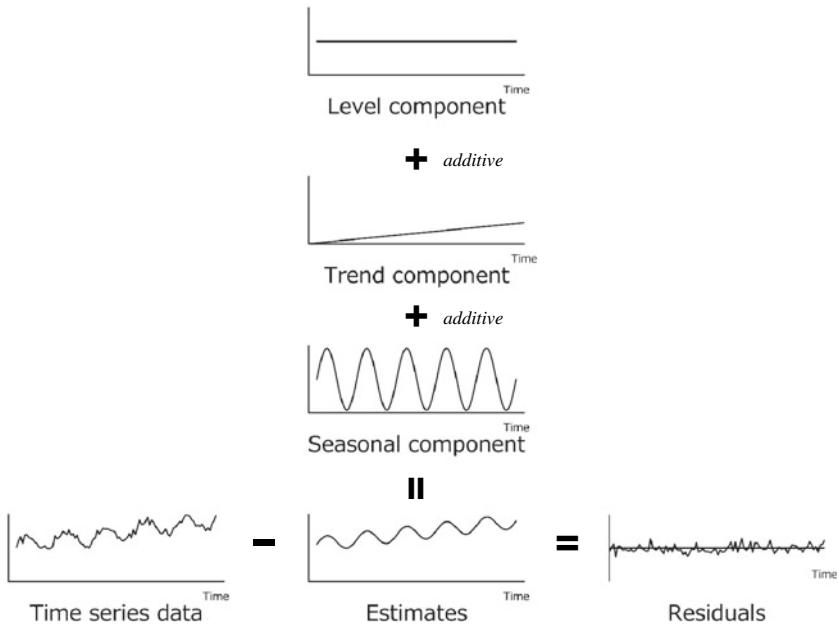


Fig. 4.10 Example of decomposition and residuals for time series data

The basic concept of modeling is the same even in the advanced state-space model. A typical model and combinations of them are the basic. In the state-space model, the model corresponding to “level + trend + season” is referred to as the basic structural model [2, 4] or standard seasonal adjustment model [7]. Chapter 9 will explain this topic in more detail.

We now actually model the data to be analyzed. Based on the knowledge obtained through preliminary examination, we make the following assumption about the data features:

- similar values continue (but are expected to have decreased rapidly in 1899) with no periodicity
- an increasing trend and periodicity of year
- an increasing trend and periodicity of year
- alternate small and large values (with an unknown mechanism) with no periodicity.

Considering the above features, we set the model as follows:

- (a) level (expecting a quick level adjustment for the sudden decrease in 1899)
- (b) level + trend + season
- (c) level + trend + season
- (d) level + trend (expecting proper tracking for alternate small and large values).

When targeting real data, we can never know the correct model except in special cases. The author recommends that the reader pragmatically regards the model as just a means of making it easier to interpret the data characteristics.

4.4 Specification of Parameter Values

Based on the model discussed above, this chapter uses the *Holt–Winters method* for analysis. The method developed by C. C. Holt and P. R. Winters is used primarily in economics [3] and has a logical connection with the stochastic estimation method. The Holt–Winters method is a kind of *exponentially weighted moving average (EWMA)*. When observations with a period p are obtained until y_t , the k -steps-ahead prediction value \hat{y}_{t+k} obtained by the *additive* Holt–Winters method is as follows:

$$\hat{y}_{t+k} = \text{level}_t + \text{trend}_t k + \text{season}_{t-p+k_p^+}, \quad (4.3)$$

$$\text{level}_t = \alpha(y_t - \text{season}_{t-p}) + (1 - \alpha)(\text{level}_{t-1} + \text{trend}_{t-1}), \quad (4.4)$$

$$\text{trend}_t = \beta(\text{level}_t - \text{level}_{t-1}) + (1 - \beta)\text{trend}_{t-1}, \quad (4.5)$$

$$\text{season}_t = \gamma(y_t - \text{level}_t) + (1 - \gamma)\text{season}_{t-p}, \quad (4.6)$$

where $k_p^+ = \lfloor (k - 1) \bmod p \rfloor + 1$ (however, $k_p^+ = p$ for $k = 0$). The prediction value \hat{y}_t for $k = 0$ corresponds to the filtering value. The terms level_t , trend_t , and season_t indicate *level component*, *trend component*, and *seasonal component*, respectively. The terms α , β , and γ are an *exponential weight*¹ for the level, trend, and seasonal component, respectively; they take values from 0 to 1.

¹ This value is also known as the forgetting or discount factor depending on the specific field.

While the Holt–Winters method is deterministic, it is known that the method becomes (almost) equivalent to the stochastic method in a specific setting: the equivalence with the autoregressive integrated moving average (ARIMA) model is described in [1, Sect. 9], whereas the equivalence with the state-space model is described in [4, Sect. 4.1.2]. Incidentally, to obtain a probabilistic confidence interval using the Holt–Winters method, one approach [5, 6] reuses the results of a Kalman filter while reconstructing the above expressions using the state-space model. Such an approach is just a detour from the viewpoint of pure state-space modeling; this book explains the Holt–Winters method in deterministic and exploratory terms.

Even if we define the model and choose a method, there generally remain things that need to be decided before performing the analysis. This section discusses such matters as parameters in a broad sense. While using the stochastic method, we just specify the parameters of the probability distribution for the model; even when we are using the deterministic method, there are matters to decide because most processes are structured for general purpose. However, what the parameters are and how to determine them depends on the analytical method. This section confirms the case of the deterministic Holt–Winters method; we will see an example of the stochastic state-space model in Chap. 5 and later.

In the Holt–Winters method, the exponential weights α , β , and γ in Eqs. (4.4)–(4.6) are basic parameters, respectively, and are typically determined to minimize the sum of the squared one-step-ahead prediction error over all time points. Note that the terms level_t , trend_t , and season_t are just estimates based on the specified parameters α , β , and γ , and are not parameters. The function `HoltWinters()` in R is available for the Holt–Winters method; it can perform the specification of parameter values and estimation simultaneously; the next section explains them together.

4.5 Execution of Filtering, Prediction, and Smoothing

Once the model and parameters are specified, we perform the analysis process, i.e., filtering, prediction, and smoothing. This section describes the case of the deterministic Holt–Winters method; we will explain the example of the stochastic state-space model in Chap. 5 and later. The Holt–Winters method can perform filtering and prediction. The code for filtering with the Holt–Winters method is as follows.

Code 4.6

```

1 > # <<Holt-Winters method>>
2 >
3 > # (a) Annual flow of the Nile
4 > HW_Nile <- HoltWinters(Nile, beta = FALSE, gamma = FALSE)
5 > str(HW_Nile)
6 List of 9
7 $ fitted      : Time-Series [1:99, 1:2] from 1872 to 1970: 1120 1130 1089 1119 1129
8   .. attr(*, "dimnames")=List of 2
9   ...$ : NULL
10  ...$ : chr [1:2] "xhat" "level"
11 $ x          : Time-Series [1:100] from 1871 to 1970: 1120 1160 963 1210 1160 1160
12   .. attr(*, "dimnames")=List of 2
13   ...$ : chr [1:2] "x" "level"
14   ...$ : num 813 1230 1370 1140 ...
15 $ alpha       : num 0.247
16 $ beta        : logi FALSE
17 $ gamma       : logi FALSE
18 $ coefficients: Named num 805
19   .. attr(*, "names")= chr "a"
20 $ seasonal    : chr "additive"
21 $ SSE         : num 2038872
22 $ call        : language HoltWinters(x = Nile, beta = FALSE, gamma = FALSE)
23 - attr(*, "class")= chr "HoltWinters"
24 >
25 > # (b) CO2 concentration in the atmosphere
26 > HW_CO2 <- HoltWinters(y_CO2)
27 >
28 > # (c) Quarterly gas consumption in the UK
29 > HW_UKgas_log <- HoltWinters(UKgas_log)
30 >
31 > # Plot the filtering value
32 >
33 > # Preprocessing regarding plot (saving the default settings for plot, and then
34   .. changing them)
35 > oldpar <- par(no.readonly = TRUE)
36 > par(mfrow = c(2, 2), oma = c(0, 0, 0, 0), mar = c(5, 3.5, 2, 1), mgp = c(2.5, 1, 0))
37 > mygray <- "#80808080"
38 > plot(HW_Nile      , main = "", col = mygray, col.predicted = "black",
39 +       lty.predicted = "dashed")
40 > title(sub = "(a)", line = 4, family = "mono")
41 >
42 > plot(HW_CO2      , main = "", col = mygray, col.predicted = "black",
43 +       lty.predicted = "dashed")
44 > title(sub = "(b)", line = 4, family = "mono")
45 >
46 > plot(HW_UKgas_log, main = "", col = mygray, col.predicted = "black",
47 +       lty.predicted = "dashed")
48 > title(sub = "(c)", line = 4, family = "mono")
49 >
50 > plot(HW_nonlinear, main = "", col = mygray, col.predicted = "black",
51 +       lty.predicted = "dashed")
52 > title(sub = "(d)", line = 4, family = "mono")
53 >
54 > # Post-processing regarding plot
55 > par(oldpar)

```

As mentioned in the previous section, the R function `HoltWinters()` for the Holt–Winters method performs the specification of parameter values and filtering simultaneously. Although the parameters of the Holt–Winters method are exponential weights α , β , and γ , if there is something unused among them, we set FALSE to the corresponding arguments `alpha`, `beta`, and `gamma`. The return value from `HoltWinters()` is assigned to the variable `HW_Nile`; hence, we confirm the contents. The variable is a list with the name of the class `HoltWinters`. The specified parameter values are stored in the list elements `alpha`, `beta`, and `gamma`, whereas the element `fitted` contains the filtering value for each component and sum of the filtering values for all components separately. When `plot()` is applied to the return value from `HoltWinters()`, the filtering value summing all the components is depicted over the original data. Figure 4.11 shows the result of drawing the data as a solid gray line and filtering value as a black dashed line.

Figure 4.11 shows the various results; we will examine them together in the next section. Here, we will proceed ahead.

We now confirm each component in the result of the Holt–Winters method separately. This code is as follows.

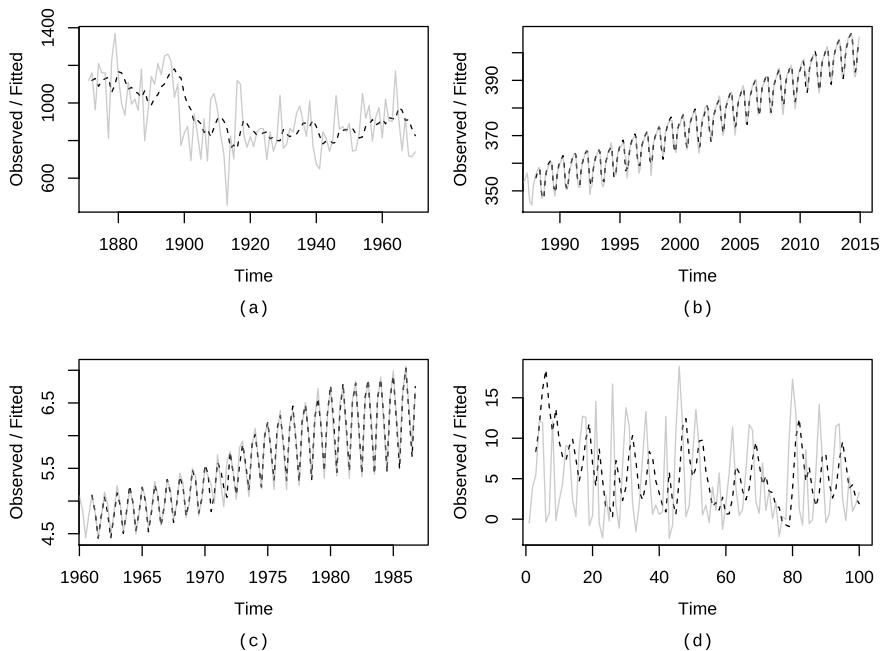


Fig. 4.11 Holt–Winters method

Code 4.7

```

1 > # <<Holt-Winters method (per component)>>
2 >
3 > # (a) Annual flow of the Nile
4 > HW_out <- HW_Nile
5 > HW_decomp <- ts.union(y = HW_out$x,
6 +                         Level = HW_out$fitted[, "level"],
7 +                         Residuals = residuals(HW_out))
8 > plot(HW_decomp, main = "", cex.lab = 1.3, cex.axis = 1.5, mar = c(0, 5, 1, 1))
9 >
10 > # (b) CO2 concentration in the atmosphere
11 > HW_out <- HW_CO2
12 > HW_decomp <- ts.union(y = HW_out$x,
13 +                         Level = HW_out$fitted[, "level"] + HW_out$fitted[, "trend"],
14 +                         Season = HW_out$fitted[, "season"],
15 +                         Residuals = residuals(HW_out))
16 > plot(HW_decomp, main = "", cex.lab = 1.3, cex.axis = 1.5, mar = c(0, 5, 1, 1))
17 >
18 > # (c) Quarterly gas consumption in the UK
19 > HW_out <- HW_UKgas_log
20 > HW_decomp <- ts.union(y = HW_out$x,
21 +                         Level = HW_out$fitted[, "level"] + HW_out$fitted[, "trend"],
22 +                         Season = HW_out$fitted[, "season"],
23 +                         Residuals = residuals(HW_out))
24 > plot(HW_decomp, main = "", cex.lab = 1.3, cex.axis = 1.5, mar = c(0, 5, 1, 1))
25 >
26 > # (d) Artificially-generated data from a nonlinear model
27 > HW_out <- HW_nonlinear
28 > HW_decomp <- ts.union(y = HW_out$x,
29 +                         Level = HW_out$fitted[, "level"] + HW_out$fitted[, "trend"],
30 +                         Residuals = residuals(HW_out))
31 > plot(HW_decomp, main = "", cex.lab = 1.3, cex.axis = 1.5, mar = c(0, 5, 1, 1))

```

Using the function `ts.union()`, we unite the original data, estimates for each component, and residuals that indicate the original data minus the sum of the filtering values for all components. When a component includes both level and trend in this example, the author summed them as a newly defined level for easy confirmation of the plot. The function `residuals()` is also used to obtain residuals. The residuals are necessary for diagnostic checking of the results. Figure 4.12 shows the results of the above code.

The plots show the various results; we will examine them together in the next section. Here, we will proceed ahead.

Recalling the descriptions in Table 4.1, we recognize that purposes two and three mention the past and future values, respectively, in addition to the present value in purpose one. Specifically, purpose two must confirm the past change point in the data (a). While smoothing can consider relative future information and is valid for accurate estimation of such past values, the standard Holt–Winters method cannot perform smoothing; hence, we examine the change point through the filtering result. On the contrary, the purpose three must predict future values in the data (b). For this purpose, we can predict the value in 2015 (one year in the future from the end of data) using the following code.

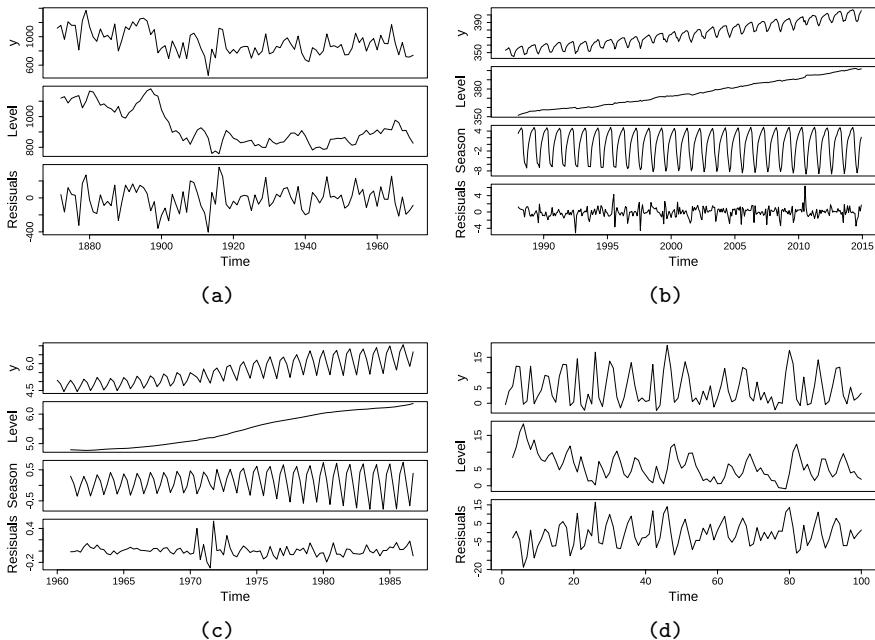


Fig. 4.12 Holt-Winters method (per component)

Code 4.8

```

1 > # <<Holt-Winters method (forecast)>>
2 >
3 > # (b) CO2 concentration in the atmosphere
4 > HW_predict <- predict(HW_CO2, n.ahead = 12)
5 > str(HW_predict)
6 Time-Series [1:12, 1] from 2015 to 2016: 405 407 407 408 407 ...
7 - attr(*, "dimnames")=List of 2
8   ..$ : NULL
9   ..$ : chr "fit"
10 >
11 > # Plot observations along with filtering and prediction values
12 > plot(HW_CO2, HW_predict, main = "Filtering and prediction with Holt-Winters method",
13 +       col = mygray, col.predicted = "black", lty.predicted = "dashed")
14 >
15 > # Plot observations in 2015 as well
16 > y_CO2_2015 <- window(y_all, start = 2015)
17 > lines(y_CO2_2015, col = mygray)

```

Applying the function `predict()` to the return value from `HoltWinters()` yields the prediction value. The argument `n.ahead` is set to the number of time points for prediction. The result `HW_predict` contains `ts` class object for the year 2015. Figure 4.13 shows the plot for the predicted values.

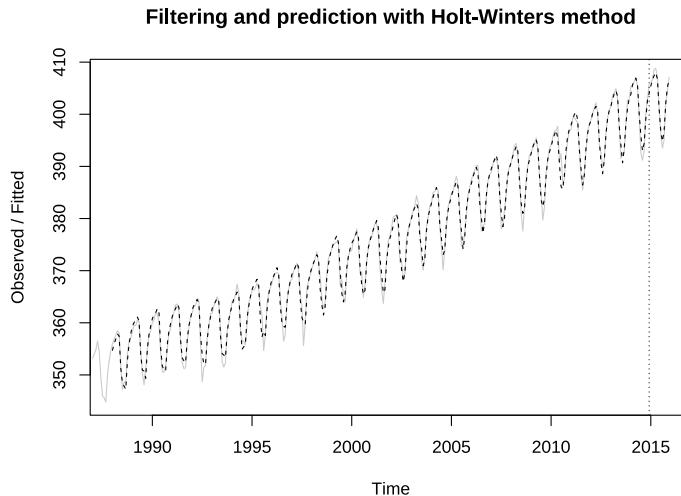


Fig. 4.13 Holt–Winters method (forecast)

If we set the return value from `HoltWinters()` and `predict()` into the argument of the `plot()` in comma-separated style, they are drawn together in the same area. The resulting plot draws the original data, including the observations in 2015, with a solid gray line and both filtering and prediction values with a dashed black line.

Through the above, we have obtained entire results for analysis purposes to be able to examine them together in the next section.

4.6 Diagnostic Checking for the Results

Finally, we perform diagnostic checking for the analysis results. This checking also includes a review of the assumption and process in the analysis. While checking is the last ordered procedure in the analysis, the analysis rarely finishes at once; we typically must repeat the analysis procedure several times based on the results. Some literature refers to the contents in this section as “diagnosis (of the model).”

We now reconfirm the results obtained for the purpose in Table 4.1. Note that the following some figures reproduce the other ones for readability.

For analyzing purposes one, “Remove noise as much as possible from the data while capturing data” and two, “Determine any sudden change in past values” in Table 4.1, we rely on Figs. 4.14 and 4.15. Since the correct result is unknown except for the exceptional data (d) for which the answer is known, we confirm the findings from the viewpoint of whether it is easy to interpret the features assumed in the model definition.

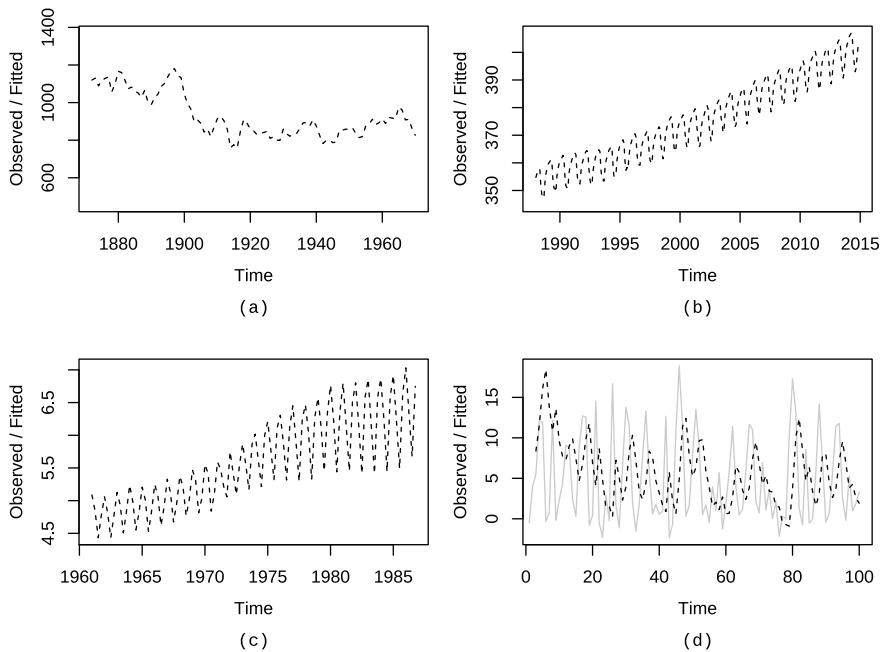


Fig. 4.14 Holt–Winters method (reproduced Fig. 4.11)

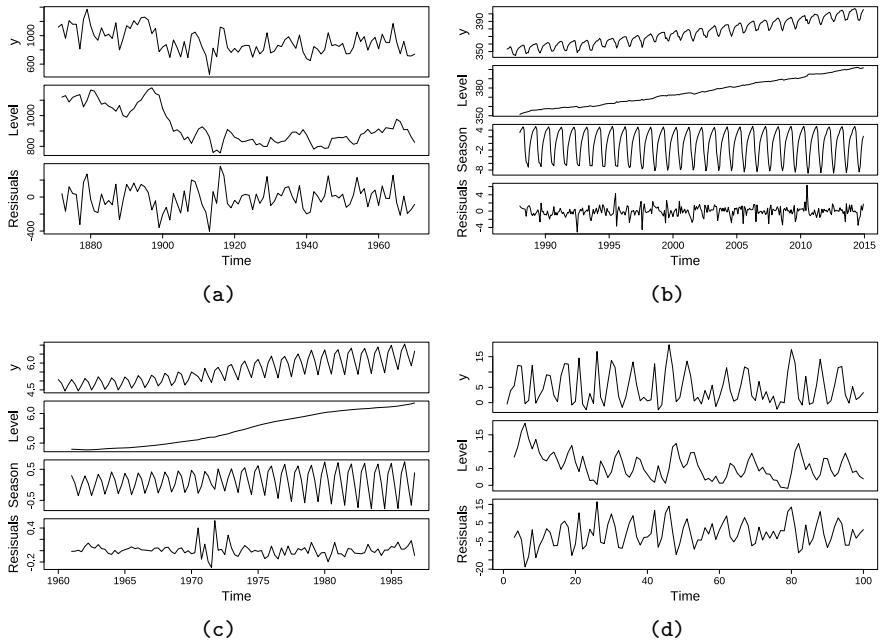


Fig. 4.15 Holt–Winters method (per component) (reproduced Fig. 4.12)

The result for data (a) makes a less strange impression; nearly similar values continue with fluctuation except for the decrease around 1899. While there are some parts where estimates are far from the data and the residuals are consequently large, as a whole, the residuals are not too large and have no remaining periodic patterns. Extremely large residuals or remaining correlations indicate that the corresponding factors are not adequately considered in the model; hence, the modeling is not sufficient. Thus, residuals are used to confirming whether problems remain in the analysis.

Even in the case of the state-space model to be described in Chap. 5 and later, residuals are useful for diagnostic checking. On the contrary, the state-space model enables more theoretical treatment for the residuals compared to the exploratory method. We will explain the details in Sect. 8.2.6.

For purpose two “Determine the sudden change in past values,” the decrease from the year 1899 appears to be very gentle and not as rapid as expected. To capture the sudden decrease accurately, we must modify the model further.

Subsequent results for data (b) and (c) make no strange impression. Similarly to (a), while there are some parts where estimates are sometimes far from the data and residuals are consequently large, as a whole, the residuals are not considerably large and have no remaining periodic patterns.

Final result for data (d) unfortunately makes a strange impression. We can recognize that the estimates are far from the correct answer shown in Fig. 4.16 and the residuals take large values as a whole. This result is natural because we perform the analysis without accurate knowledge of the nonlinear data generative process. Though such a generative process is rarely known in practice, if possible, we must actively include the knowledge in the modeling for accurate analysis.

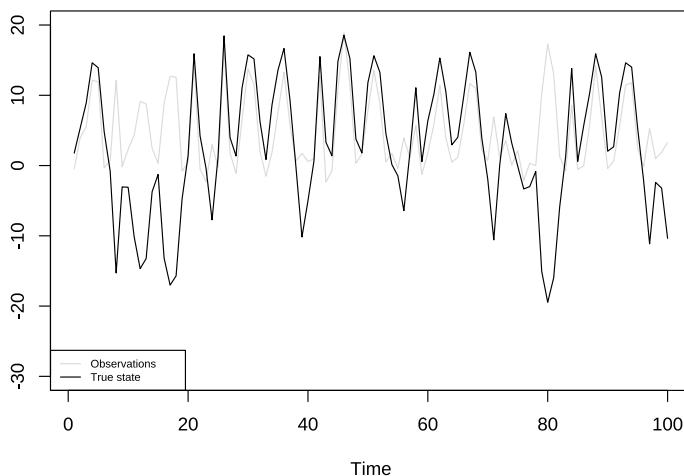


Fig. 4.16 A well-known benchmark model (reproduced Fig. 11.9)

While we have relied on visual inspection of the residuals thus far, it is also common to confirm the autocorrelation coefficient for residuals. The autocorrelation coefficient for residuals in this example can be obtained with the following code.

Code 4.9

```

1 > # <<Diagnostic checking for the results (autocorrelation of residuals)>>
2 >
3 > # Preprocessing regarding plot (saving the default settings for plot, then changing
4 >   ↪ them)
5 > oldpar <- par(no.readonly = TRUE)
6 > par(mfrow = c(2, 2), oma = c(0, 0, 0, 0), mar = c(5, 3.5, 2, 1), mgp = c(2.5, 1, 0))
7 >
8 > acf(residuals(HW_Nile)      , main = "")
9 > title(sub = "(a)", line = 4, family = "mono")
10 >
11 > acf(residuals(HW_CO2)       , main = "")
12 > title(sub = "(b)", line = 4, family = "mono")
13 >
14 > acf(residuals(HW_UKgas_log), main = "")
15 > title(sub = "(c)", line = 4, family = "mono")
16 >
17 > acf(residuals(HW_nonlinear), main = "")
18 > title(sub = "(d)", line = 4, family = "mono")
19 >
20 > # Post-processing regarding plot
21 > par(oldpar)

```

Figure 4.17 shows the results, which are consistent with the previous discussion: the residuals do not have a large correlation about (a), (b), and (c), except for (d).

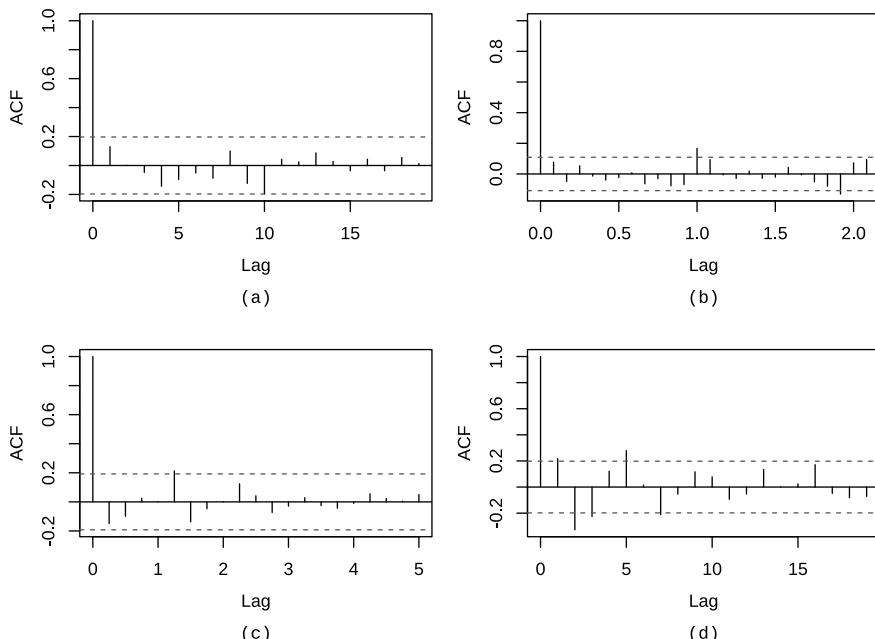


Fig. 4.17 Autocorrelation coefficient for residuals

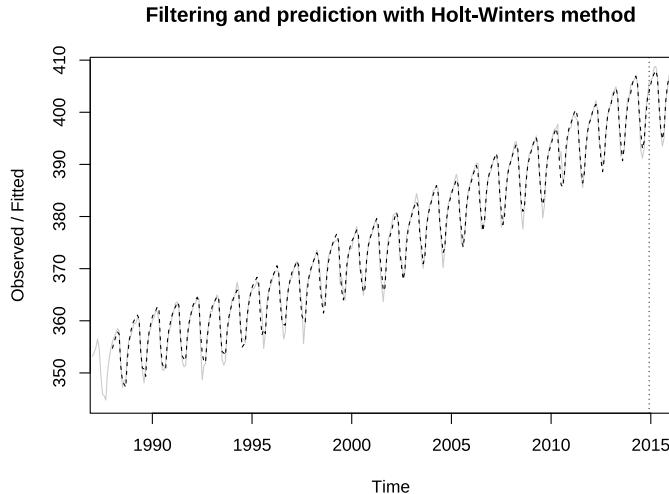


Fig. 4.18 Holt–Winters method (forecast) (reproduced Fig. 4.13)

As for purpose three “Predict future values” in the Table 4.1, we now rely on Fig. 4.18. The result makes no strange impression; compare it with real observations in 2015.

We further confirm the numeric indicator regarding prediction. Although there are various such indicators [8], and *mean squared error* is often used owing to the connection with theory, this book introduces *mean absolute percentage error (MAPE)* for intuitive understanding.

The MAPE of estimates \hat{y}_t is defined for observations y_t as

$$\text{MAPE} = \frac{1}{K - k} \sum_{t=k}^{K-1} \frac{|\hat{y}_t - y_t|}{y_t}, \quad (4.7)$$

where k and $K - 1$ are the start and end time points in the evaluation duration, respectively. The closer to 0 this value is, the more accurate the prediction becomes. The code for obtaining MAPE is as follows.

Code 4.10

```

1 > # <<Diagnostic checking for the results (prediction performance)>>
2 >
3 > # User-defined function for mean absolute percentage error (MAPE)
4 > MAPE <- function(true, pred){
5 +   mean(abs(pred - true) / true)
6 + }
7 >
8 > # The MAPE of the prediction value
9 > MAPE(true = y_CO2_2015, pred = HW_predict)
10 [1] 0.001754518

```

The above code first defines a user-defined function `MAPE()` for calculating MAPE. As a result, we obtain a sufficiently small MAPE value 0.001754518 and can accept the prediction as reasonable.

In the above manner, we have performed diagnostic checking for the estimation results. We can also recognize that further challenges remain in several cases for improving the estimation accuracy. Based on such insight, the analysis is repeated.

4.7 Guideline When Applying the State-Space Model

Until now, this chapter has walked through a time series analysis based on the deterministic method. For further detailed analysis, this book recommends the use of the stochastic state-space model; we will explain the details in the following chapters sequentially. Before diving into the details, this last section preliminarily shows a rough guideline about what type of solution the state-space model would be able to apply to the data discussed in this chapter.

The state-space model has two types of solution: the *batch solution* expending specific computational power treats collected offline data, while the *sequential solution* efficiently treats sequentially obtained data. The batch solution is suitable for fixed-interval smoothing, and the sequential solution is suitable for fixed-lag smoothing and filtering. In addition, the state-space model can be roughly classified into the *linear Gaussian state-space model* and the *general state-space model*. For the linear Gaussian state-space model, the efficient solving algorithm is well known, while the general state-space model usually requires more computational power than the former. This book introduces the *Wiener filter* (Chap. 7) and *Kalman filter* (Chap. 8) as batch and sequential solutions for the linear Gaussian state-space model, respectively. Furthermore, this book introduces the solution using *MCMC* (Chap. 10) and *particle filter* (Chap. 11) as batch and sequential solutions for the general state-space model, respectively. The relationship among the above is summarized in Table 4.2.

Note that this book treats the Wiener filter primarily from a historical perspective and focuses on the Kalman filter as the solution for the linear Gaussian state-space model.

Table 4.2 Solutions for the state-space model introduced in this book

	Batch type	Sequential type
For the linear Gaussian state-space model	Wiener filter (Chap. 7)	Kalman filter Chap. 8
For the general state-space model	Solution using MCMC (Chap. 10)	Particle filter (Chap. 11)

In an actual analysis, we must select a proper solution. If a simple linear Gaussian state-space model can adequately solve a problem, it is sufficient. Even though we might encounter a severe problem, the author believes that the solution for the linear Gaussian state-space model is worth trying at least for comparison. Whether the linear Gaussian state-space model is sufficient depends on the problem; it is theoretically known that the linear Gaussian state-space model can adequately deal with typical issues such as the basic “level + trend + season” model and consideration of “known” structural change. Based on such knowledge, we can set the solution-applying guidelines for the data discussed in this chapter as follows:

- The solution for the linear Gaussian state-space model can deal adequately with the following problems:
 - Accurately estimate the current values on the data (a), (b), and (c) (however, the parameters are not considered as random variables).
 - Consider the past structural change point on the data (a) with prior knowledge.
 - Predict the future values on the data (b).
- The solution for general state-space model is required for the following problems:
 - Accurately estimate the current values on the data (d).
 - Consider the past or current structural change point on the data (a) without prior knowledge.
 - Estimate both data value and parameters jointly as random variables on all data.

The following chapters will explain the details regarding how to overcome problems that are difficult to solve with the deterministic and exploratory methods described in this chapter. We sometimes omit some analysis steps to focus on a particular topic to be explained in the later chapters.

References

1. Brockwell, P.J., Davis, R.A.: *Introduction to Time Series and Forecasting*, 3rd edn. Springer, New York (2016)
2. Durbin, J., Koopman, S.J.: *Time Series Analysis by State Space Methods*, 2nd edn. Oxford University Press, Oxford (2012)
3. Goodwin, P.: The Holt–Winters approach to exponential smoothing: 50 years old and going strong. *Foresight: Int. J. Appl. Forecast.* **19**(Fall), 30–33 (2010)
4. Harvey, A.C.: *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press, Cambridge (1989)
5. Hyndman, R.J., Khandakar, Y.: Automatic time series forecasting: the forecast package for R. *J. Stat. Softw.* **26**(3), 1–22 (2008)
6. Hyndman, R.J., Koehler, A.B., Ord, J.K., Snyder, R.D.: *Forecasting with Exponential Smoothing: The State Space Approach*. Springer Series in Statistics. Springer, Berlin (2008)

7. Kitagawa, G.: Introduction to Time Series Modeling. CRC Press, New York (2009)
8. Petris, G., Petrone, S., Campagnoli, P.: Dynamic Linear Model with R. Springer, New York (2009)
9. Statistics Section, Department of Social Sciences, Collage of Arts and Sciences, The University of Tokyo (ed.): Introduction to Statistics. University of Tokyo Press (1991). [in Japanese]

Chapter 5

State-Space Model



This chapter mentions the stochastic model and defines the state-space model as a stochastic model. We also explain the features and classification of the state-space model.

5.1 Stochastic Model

This chapter explains the state-space model; this section first emphasizes that this model is based on a flexible stochastic model. Such a model considers observations as samples that are obtained from a probability distribution, and the estimation target becomes not a specific value but the distribution. For example, Fig. 5.1 applies that concept to the data on the annual flow of the Nile.

Although the drawing might be somewhat difficult to discern, Fig. 5.1 shows the observations and its probability distribution in two-dimensional (2D) and three-dimensional (3D) plots, respectively.¹ As can be seen from Fig. 5.1, the probability distribution of the estimation target is generally different for each time point.

5.2 Definition of State-Space Model

The *state-space model* interprets series data that have a relation with each other in stochastic terms.

¹This figure assumes the normal density for the probability distribution and draws the estimated result using the library **dlm**.

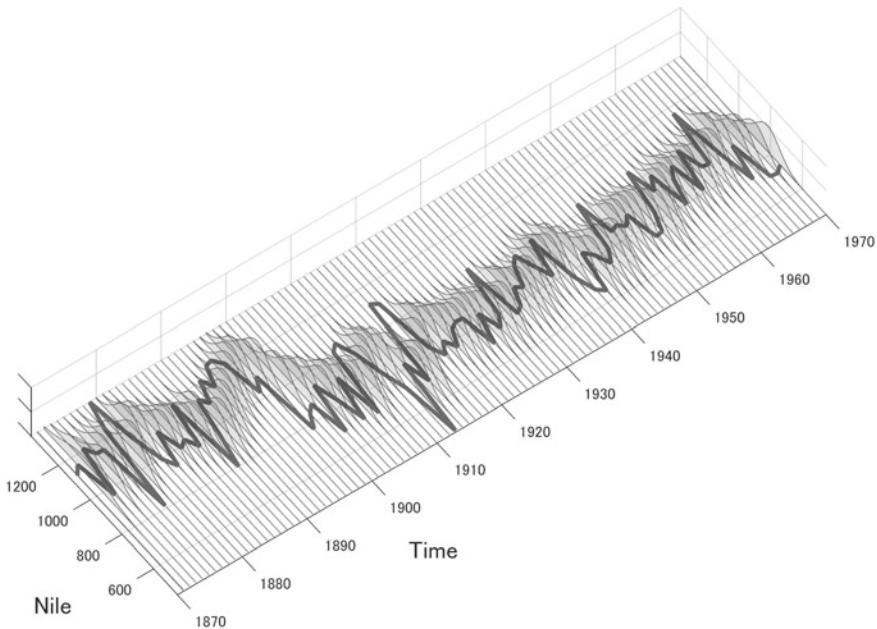


Fig. 5.1 Stochastic treatment for the data on the annual flow of the Nile

First, this model introduces latent random variables that are not directly observed in addition to directly observed data. Such a latent random variable is called a *state*. Regarding such a state, we can freely choose any quantity that makes interpretation easier for analysis; we will provide examples in Chap. 9. We assume the following association for a state, i.e., a *Markov property*:

A state is related only to the state at the previous time point.

We also assume the following property for observations:

An observation at a certain time point depends only on the state at the same time point.

The above descriptions provide the assumptions for the state-space model. We can represent these assumptions from three different perspectives. We then confirm them for a deeper understanding.

This book takes a consistent stance wherein Bayesian estimation is applied to a state [5]. For this reason, we assume that the initial value of the state is also drawn from the prior distribution. In contrast, unlike the state, the maximum likelihood estimation is applied to the parameters without regarding them as random variables until Chap. 9; we will also explain the case wherein Bayesian estimation is used on the parameters, regarding them as random variables in Chap. 10 and later. The terms “state” and “parameters” in this book can also be respectively referred to as “parameters” and “hyperparameters” [4, 6, 7, 9] or “extensive parameters” and “intensive parameters” [1], depending on the references.

5.2.1 Representation by Graphical Model

The first representation is based on a graphical model and corresponds to Fig. 5.2.

In the figure, the time durations are $1, \dots, T$, whereas the state and observation at time point t are denoted by \mathbf{x}_t and y_t , respectively. Also, \mathbf{x}_0 is a state reflecting prior knowledge. When there are p types of elements in the state, \mathbf{x}_t is a p -dimensional column vector. This book assumes only univariate observations; therefore, y_t is a scalar and not denoted in bold. On the contrary, if we consider multivariate observations in general, the observations are an m -dimensional column vector and in bold.

Henceforth, we denote both random variables and their realizations in lower-case letters. Even though such a policy is adopted, the context can be expected to prevent confusion.

Figure 5.2 is called a *directed acyclic graph (DAG)*. The Markov property for the state is expressed by the fact that \mathbf{x}_t is directly connected with only its neighbors. In addition, the fact that y_t is directly connected with only \mathbf{x}_t indicates an assumption for the observations.

$$\begin{array}{ccccccccccccc} \mathbf{x}_0 & \rightarrow & \mathbf{x}_1 & \rightarrow & \mathbf{x}_2 & \rightarrow & \cdots & \rightarrow & \mathbf{x}_{t-1} & \rightarrow & \mathbf{x}_t & \rightarrow & \mathbf{x}_{t+1} & \rightarrow & \cdots & \rightarrow & \mathbf{x}_{T-1} & \rightarrow & \mathbf{x}_T \\ \downarrow & & \downarrow & & & & & & \downarrow & & \downarrow & & & & & & & \downarrow & & \downarrow \\ y_1 & & y_2 & & & & & & y_{t-1} & & y_t & & y_{t+1} & & & & y_{T-1} & & y_T \end{array}$$

Fig. 5.2 DAG representation of the state-space model

5.2.2 Representation by Probability Distribution

The second representation is based on the probability distribution, corresponding to the following equations:

$$p(\mathbf{x}_t \mid \mathbf{x}_{0:t-1}, y_{1:t-1}) = p(\mathbf{x}_t \mid \mathbf{x}_{t-1}), \quad (5.1)$$

$$p(y_t \mid \mathbf{x}_{0:t}, y_{1:t-1}) = p(y_t \mid \mathbf{x}_t). \quad (5.2)$$

The left-hand side of Eq. (5.1) describes the distribution of \mathbf{x}_t , given all quantities from 0 to $t - 1$, whereas that of Eq. (5.2) describes the distribution of y_t , given all quantities from 0 to $t - 1$ and \mathbf{x}_t . Equation (5.1) shows that when \mathbf{x}_{t-1} is given, \mathbf{x}_t is independent of $\mathbf{x}_{0:t-2}, y_{1:t-1}$. This relation is referred to as *conditional independence* on state, and Fig. 5.3 shows its image overlaid on the DAG representation. This property holds at any time point other than t : for a particular time point, when a state at one earlier time point is given, the state at that time point is independent of all quantities before that time point (except for the state at one time point earlier).

Equation (5.2) also shows that when \mathbf{x}_t is given, y_t is independent of $y_{1:t-1}, \mathbf{x}_{0:t-1}$. This relationship is called *conditional independence* on observations, and Fig. 5.4 shows its image overlaid on the DAG representation. This property also holds at any time point other than t : for a particular time point, when a state is given at that time point, the observation at that time point is independent of all quantities before that time point.

Motivated by the abovementioned DAG representation, the other conditional independence also holds, such as

$$p(\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:T}) = p(\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t}). \quad (5.3)$$

This book reuses Eq. (5.3) in other parts; its derivation is summarized in Appendix C.

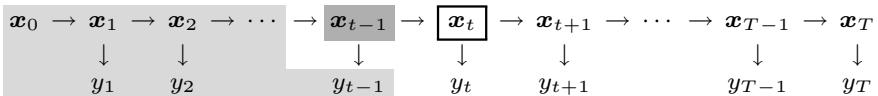


Fig. 5.3 Conditional independence on state (when \blacksquare is given, \square is independent of \blacksquare)

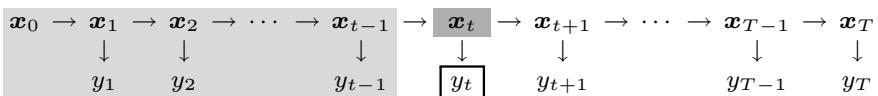


Fig. 5.4 Conditional independence on observations (when \blacksquare is given, \square is independent of \blacksquare)

5.2.3 Representation by Equation

The third representation is based on equations, corresponding to following equations:

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{w}_t), \quad (5.4)$$

$$y_t = h(\mathbf{x}_t, v_t), \quad (5.5)$$

where, $f(\cdot)$ and $h(\cdot)$ are arbitrary functions. \mathbf{w}_t and v_t are mutually independent white noise called *state noise* (or *system noise/process noise*) and *observation noise* (or *measurement noise*), respectively. The noise \mathbf{w}_t is a p -dimensional column vector. Equation (5.4) is referred to as the *state equation* (or *system equation*) and is equivalent to Eq. (5.1). Equation (5.5) is also known as the *observation equation* (or *measurement equation*) and is equivalent to Eq. (5.2).

The state equation is a stochastic differential equation for the state. The state noise \mathbf{w}_t is the tolerance for a temporal change of state; the impression created by the word “noise” is misleading. The state equation potentially specifies a temporal pattern that allows for distortion.

The observation equation, on the contrary, suggests that when observations are obtained from a state, observation noise v_t is added. While the observation noise also has an aspect of supplementing the model definition, it is mostly interpreted as typical noise different from the state noise: the less the observation noise becomes, the more accurate the observations become.

5.2.4 Joint Distribution of State-Space Model

Based on the above description, we derive the joint distribution of the state-space model. Because the joint distribution of all random variables in the model plays an essential role in representing all the features of the model, this book derives the formula in a step by step manner. The joint distribution of the state-space model is defined as

$$p(\text{all random variables}) = p(\mathbf{x}_{0:T}, y_{1:T}),$$

where the distribution on the right-hand side is further transformed as follows:

$$p(\mathbf{x}_{0:T}, y_{1:T})$$

move the random variables to be decomposed to the front considering ease of viewing

$$= p(y_{1:T}, \mathbf{x}_{0:T})$$

decompose $y_{1:T}$

$$= p(y_T, y_{1:T-1}, \mathbf{x}_{0:T})$$

apply the product rule of probability regarding considering $y_{1:T-1}, \mathbf{x}_{0:T}$ as a single block

$$= p(y_T | y_{1:T-1}, \mathbf{x}_{0:T}) p(y_{1:T-1}, \mathbf{x}_{0:T})$$

move the random variables to be decomposed in the last term to the front considering ease of viewing

$$= p(y_T | y_{1:T-1}, \mathbf{x}_{0:T}) p(\mathbf{x}_{0:T}, y_{1:T-1})$$

decompose $\mathbf{x}_{0:T}$

$$= p(y_T | y_{1:T-1}, \mathbf{x}_{0:T}) p(\mathbf{x}_T, \mathbf{x}_{0:T-1}, y_{1:T-1})$$

apply the product rule of probability to the last term regarding $\mathbf{x}_{0:T-1}, y_{1:T-1}$ as a single block

$$= p(y_T | y_{1:T-1}, \mathbf{x}_{0:T}) p(\mathbf{x}_T | \mathbf{x}_{0:T-1}, y_{1:T-1}) p(\mathbf{x}_{0:T-1}, y_{1:T-1})$$

in the above manner, apply the product rule of probability to the last term repeatedly

$$= \left(\prod_{t=2}^T p(y_t | y_{1:t-1}, \mathbf{x}_{0:t}) p(\mathbf{x}_t | \mathbf{x}_{0:t-1}, y_{1:t-1}) \right) p(\mathbf{x}_{0:1}, y_1)$$

replace the order of the random variables in the last term considering ease of viewing

$$= \left(\prod_{t=2}^T p(y_t | y_{1:t-1}, \mathbf{x}_{0:t}) p(\mathbf{x}_t | \mathbf{x}_{0:t-1}, y_{1:t-1}) \right) p(y_1, \mathbf{x}_{0:1})$$

apply the product rule of probability to the last term again

$$= \left(\prod_{t=2}^T p(y_t | y_{1:t-1}, \mathbf{x}_{0:t}) p(\mathbf{x}_t | \mathbf{x}_{0:t-1}, y_{1:t-1}) \right) p(y_1 | \mathbf{x}_{0:1}) p(\mathbf{x}_1 | \mathbf{x}_0) p(\mathbf{x}_0)$$

move the last term to the front considering ease of viewing

$$= p(\mathbf{x}_0) \left(\prod_{t=2}^T p(y_t | y_{1:t-1}, \mathbf{x}_{0:t}) p(\mathbf{x}_t | \mathbf{x}_{0:t-1}, y_{1:t-1}) \right) p(y_1 | \mathbf{x}_{0:1}) p(\mathbf{x}_1 | \mathbf{x}_0)$$

from the assumption for the state-space model, i.e., Eqs. (5.2) and (5.1)

$$= p(\mathbf{x}_0) \left(\prod_{t=2}^T p(y_t | \mathbf{x}_t) p(\mathbf{x}_t | \mathbf{x}_{t-1}) \right) p(y_1 | \mathbf{x}_1) p(\mathbf{x}_1 | \mathbf{x}_0)$$

simplify considering ease of viewing

$$= p(\mathbf{x}_0) \prod_{t=1}^T p(y_t | \mathbf{x}_t) p(\mathbf{x}_t | \mathbf{x}_{t-1}). \quad (5.6)$$

Equation (5.6) states that the state-space model is determined by a simple product by assumed prior distribution $p(\mathbf{x}_0)$, observation equation $p(y_t | \mathbf{x}_t)$, and state equation $p(\mathbf{x}_t | \mathbf{x}_{t-1})$. This result is a natural consequence of the assumptions for the state-space model.

5.3 Features of State-Space Model

This section describes some features of the state-space model explained in the previous section.

First, the introduction of a latent variable, i.e., the state, enables the state-space model to be used as the basis for easily constructing a complicated model by combining multiple states; recall that we can arbitrarily choose a state convenient for interpretation. Such a framework improves the flexibility of modeling. We confirm examples in Chap. 9.

Next, the state-space model represents the relation among observations via states indirectly rather than by direct connection among observations: Fig. 5.2 shows that y_t s are connected not directly but through \mathbf{x}_t s. This feature contrasts with the *ARMA model* (*autoregressive moving average model*) which directly represents the relation among observations. The ARMA(p, q) model is a stochastic one often used in time series analysis; it is defined as

$$y_t = \sum_{j=1}^p \phi_j y_{t-j} + \sum_{j=1}^q \psi_j \epsilon_{t-j} + \epsilon_t, \quad (5.7)$$

where p and q are nonnegative integers known as the *order of AR* and *order of MA* respectively, ϕ_j and ψ_j are real numbers known as the *AR coefficient* and *MA coefficient*, respectively, and ϵ_t is white noise. ARMA(1, 0) model has no MA term and is also known as the AR(1) model, as already mentioned in Sect. 2.7. Furthermore, the ARMA model for the d -th differences time series becomes an *ARIMA model* (*autoregressive integrated moving average model*) and is denoted by ARIMA(p, d, q). The ARIMA and state-space models are closely related; the models often used in the

state-space model can be defined as the ARIMA model [4, Appendix 1], and the ARIMA model can also be defined as one of the state-space models, as explained in Sect. 9.5. Thus, the author believes that there is no essential difference between the two models from the viewpoint of capturing the temporal pattern of data stochastically. However, their underlying philosophies are different. Since the ARIMA model represents the relation among observations directly, the modeling problem requires a *black box approach*. On the contrary, since the state-space model represents the relation among observations indirectly in terms of states, the modeling problem results in a *white box approach*: the analyst assumes as much as possible regarding the factors generating the association among data and reflects them in the model through the corresponding latent variables. While the choice of modeling approach is ultimately a matter of preference, this book recommends the state-space model in consideration of its flexibility, such as easy handling of nonstationary processes, missing observations, and the regression component. Note that the above discussion is based on [3, Sect. 3.10.1].

While the state-space model has been studied explicitly by R. E. Bellman and R. E. Kalman in control theory since the 1950s, the equivalent model appears also to have been examined previously [8, 10]. It is also well recognized that the state-space model is equivalent to the hidden Markov model [2, 7], which has been developed primarily in speech recognition; however, the random variable to be estimated in the hidden Markov model is discrete.

5.4 Classification of State-Space Models

First, when the state-space model does not have special restrictions on its operations or probability distributions, the model is referred to as the *general state-space model*. The general state-space model can be further classified. This book explains the classification shown in Table 5.1.

As can be seen from the above table, the classification is based on the setting of Eqs. (5.4) and (5.5).

Table 5.1 Classification of state-space models

	Either w_t or v_t in Eqs. (5.4) and (5.5) is non-Gaussian	Both w_t and v_t in Eqs. (5.4) and (5.5) are Gaussian
Either f or h in Eqs. (5.4) and (5.5) is nonlinear	Nonlinear non-Gaussian (Chaps. 10 and 11)	Nonlinear Gaussian (Chaps. 10 and 11)
Both f and h in Eqs. (5.4) and (5.5) are linear	Linear non-Gaussian (Chaps. 10 and 11)	Linear Gaussian (Chaps. 7 and 8)

If either f or h is a nonlinear function and either \mathbf{w}_t or v_t has a non-Gaussian distribution, then the state-space model is referred to as a *nonlinear non-Gaussian state-space model*. This model is a kind of general state-space model, but can be used with the same meaning.

If either f or h is a nonlinear function and both \mathbf{w}_t and v_t have Gaussian distributions, then the state-space model is referred to as a nonlinear Gaussian state-space model.

If both f and h are linear functions and either \mathbf{w}_t or v_t has a non-Gaussian distribution, then the state-space model is referred to as a linear non-Gaussian state-space model.

Furthermore, if both f and h are linear functions and both \mathbf{w}_t and v_t have Gaussian distributions, then the state-space model is referred to as a *linear Gaussian state-space model*. The state equation and observation equation for the linear Gaussian state-space model are expressed as follows (they are repeatedly referred to in this book and emphasized by framing):

$$\boxed{\mathbf{x}_t = \mathbf{G}_t \mathbf{x}_{t-1} + \mathbf{w}_t, \quad \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{W}_t),} \quad (5.8)$$

$$\boxed{y_t = \mathbf{F}_t \mathbf{x}_t + v_t, \quad v_t \sim \mathcal{N}(0, V_t),} \quad (5.9)$$

where \mathbf{G}_t is a $p \times p$ state transition matrix (or state evolution matrix/system matrix), \mathbf{F}_t is a $1 \times p$ observation matrix (or measurement matrix), \mathbf{W}_t is a $p \times p$ covariance matrix for the state noise, and V_t is a variance for the observation noise. The prior distribution is also expressed as $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{m}_0, \mathbf{C}_0)$, where \mathbf{m}_0 is the p -dimensional mean vector and \mathbf{C}_0 is a $p \times p$ covariance matrix. All the results from the linear operation in Eqs. (5.8) and (5.9) have a normal distribution according to the reproducibility for normal distribution mentioned in Sect. 2.3. In addition, Eqs. (5.8) and (5.9) can be expressed equivalently with a probability distribution as follows (they are also repeatedly referred to in this book and emphasized by framing):

$$\boxed{p(\mathbf{x}_t | \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{G}_t \mathbf{x}_{t-1}, \mathbf{W}_t),} \quad (5.10)$$

$$\boxed{p(y_t | \mathbf{x}_t) = \mathcal{N}(\mathbf{F}_t \mathbf{x}_t, V_t),} \quad (5.11)$$

where $p(\mathbf{x}_0) = \mathcal{N}(\mathbf{m}_0, \mathbf{C}_0)$. Parameters in the linear Gaussian state-space model are summarized as $\boldsymbol{\theta} = \{\mathbf{G}_t, \mathbf{F}_t, \mathbf{W}_t, V_t, \mathbf{m}_0, \mathbf{C}_0\}$. The linear Gaussian state-space model is also referred to as a *dynamic linear model (DLM)*. DLM is not only easy to understand and handle but also is practically important. The DLM alone can cover many problems, and even in the general state-space model, its conditional model can be reduced to a linear Gaussian state-space model; in such a case, the DLM is to be used in part to solve an entire problem.

The symbols assigned for the linear Gaussian state-space model in expressions (5.8) and (5.9) are not universally standard but vary depending on the literature. This book mainly complies with [10–12] considering the consistency with the functions in **dlm** and **Stan**. For information, Appendix D summarizes the correspondence relationship with notations in the primary literature. Incidentally, regarding the state noise, while it appears to be more common to express it as the product of a matrix and vector rather than as a reduced vector such as w_t , as shown in this book, the two representations are equivalent.

References

1. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, Berlin, Heidelberg (2006)
2. Cappé, O., Moulines, E., Rydén, T.: Inference in Hidden Markov Models. Springer, Berlin, Heidelberg (2005)
3. Durbin, J., Koopman, S.J.: Time Series Analysis by State Space Methods, 2nd edn. Oxford University Press, Oxford (2012)
4. Harvey, A.C.: Forecasting, Structural Time Series Models and the Kalman Filter. Cambridge University Press, Cambridge (1989)
5. Iba, Y. (ed.): The Expanding World of Bayesian Modeling. Iwanami Shoten, Tokyo (2018). [in Japanese]
6. Ishiguro, M., Matsumoto, T., Inui, T., Tanabe, K.: Hierarchical Bayesian Model and Related Topics—Application to Time Series, Images and Cognition. Iwanami Shoten, Tokyo (2004). [in Japanese]
7. Iwanami Data Science Publication Committee (ed.): Iwanami Data Science, vol. 6. Iwanami Shoten, Tokyo (2017). [in Japanese]
8. Kailath, T., Sayed, A.H., Hassibi, B.: Linear Estimation. Prentice Hall, Upper Saddle River, NJ (2000)
9. Kitagawa, G., Gersch, W.: Smoothness Priors Analysis of Time Series. Springer, New York, NY (1996)
10. Petris, G., Petrone, S., Campagnoli, P.: Dynamic Linear Model with R. Springer, New York, NY (2009)
11. Prado, R., West, M.: Time Series: Modeling, Computation, and Inference. CRC Press, Boca Raton, FL (2010)
12. West, M., Harrison, J.: Bayesian Forecasting and Dynamic Models, 2nd edn. Springer, Berlin, Heidelberg (1997)

Chapter 6

State Estimation in the State-Space Model



This chapter first explains the general concept of state estimation in the state-space model. Then, using a simple example, we describe a method for obtaining the marginal posterior distribution of the state sequentially. Then, we formulate them generally. This result is the basis of the Kalman filter (Chap. 8) and particle filter (Chap. 11), which are explained in later chapters. This chapter also describes the likelihood of the state-space model and treatment of parameters.

6.1 State Estimation Through the Posterior Distribution

The state in the state-space model is estimated based on observations through the current time point t . Thus, the distribution to be estimated becomes a conditional distribution as

$$p(\text{state} \mid y_{1:t}). \quad (6.1)$$

Equation (6.1) is generally referred to as the *posterior distribution of the state*.

For the “state” in Eq. (6.1), we must first consider all time points as

$$p(\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{t'}, \dots \mid y_{1:t}). \quad (6.2)$$

Equation (6.2) is referred to as the *joint posterior distribution of the state*. The estimation type is classified depending on the focused time point t' as follows:

$$\begin{cases} \text{smoothing} & \text{for } t' < t, \\ \text{filtering} & \text{for } t' = t, \\ \text{prediction} & \text{for } t' > t. \end{cases}$$

The joint posterior distribution in Eq. (6.2) is regarded as the estimation target in the solution using MCMC (Chap. 10) and one of the smoothing algorithms of the particle filter (Chap. 11). Now, if we want to exclude the states whose time points differ from the focused t' , we perform marginalization as

$$p(\mathbf{x}_{t'} \mid y_{1:t}) = \int p(\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{t'}, \dots \mid y_{1:t}) d\mathbf{x}_{\text{except } t'}. \quad (6.3)$$

Equation (6.3) is referred to as the *marginal posterior distribution of the state*. Depending on the focused time point t' , Eq. (6.3) is as follows:

$$\begin{cases} \text{smoothing distribution for } t' < t, \\ \text{filtering distribution for } t' = t, \\ \text{predictive distribution for } t' > t. \end{cases}$$

The marginal posterior distribution in Eq. (6.3) is regarded as the estimation target in all subsequent chapters. There are several means of deriving the marginal posterior distribution of the state. For example, in the solution using MCMC (Chap. 10) and one of the smoothing algorithms of the particle filter (Chap. 11), we simply marginalize the obtained joint posterior distribution. In addition, there is a direct method of obtaining the marginal posterior distribution. The Wiener filter (Chap. 7) is one such method; it is classified into a batch solution in the frequency domain. An efficient derivation method in the time domain is also known, and this approach provides the basis of a sequential solution such as the Kalman filter (Chap. 8) and particle filter (Chap. 11). The next section formulates the recursions for obtaining the marginal posterior distribution of the state in Eq. (6.3).

The joint posterior distribution of the state $p(\mathbf{x}_{0:t} \mid y_{1:t})$ is sometimes referred to as the smoothing distribution with the focus on the time points before the current time point t . The name is the same as that of the smoothing distribution in the marginal posterior distribution of the state, but this book uses the name as long as no misunderstanding can arise.

6.2 How to Obtain the State Sequentially

6.2.1 A Simple Example

The method for obtaining the state sequentially is based on the repetition of state transitions and Bayesian updating, that is, the posterior distribution \propto prior distribution \times likelihood. This section introduces a simple example for the reader to obtain

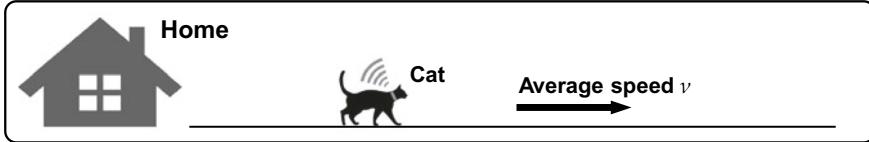


Fig. 6.1 Moving cat

the image before the recursive formulation. This example explains only the filtering case because filtering is the most fundamental mechanism providing the basis for prediction and smoothing. Filtering includes one-step-ahead prediction partially, and smoothing results from filtering.

We now move on to a concrete explanation. Let there be a domestic cat with a collar with a GPS tracker function for monitoring its behavior, as shown in Fig. 6.1.

Although an actual cat's behavior is fickle, and modeling it appears to be difficult, for simplicity, we assume a case wherein the cat moves straight away from home at an average speed v . We formulate this problem in terms of the linear Gaussian state-space model as

$$x_t = x_{t-1} + v + w_t, \quad w_t \sim \mathcal{N}(0, \sigma_w^2), \quad (6.4)$$

$$y_t = x_t + v_t, \quad v_t \sim \mathcal{N}(0, \sigma_v^2), \quad (6.5)$$

where $p(x_0) \sim \mathcal{N}(m_0, C_0)$, $m_0 = \hat{m}_1$ is the guessed position at $t = 1$ before the beginning of observation from $t = 1$ based on prior knowledge, and variance C_0 represents the degree of confidence in the guess. In Eqs. (6.4) and (6.5), x_t is the cat's position at time point t , v is the average velocity (constant), w_t is the fluctuation in velocity during movement, y_t is an observation at time t , and v_t is the noise at the observation. In this example, every distribution of interest becomes a normal distribution in accordance with the reproducibility of normal distributions. Using these characteristics, we derive the process of obtaining the state sequentially. First, adding such a sequential process to the bottom of Fig. 6.1 yields Fig. 6.2. In the figure, the progress of time is shown in the downward direction and movement of the cat in the rightward direction.

In Fig. 6.2, we first estimate the cat's position at the beginning of movement, i.e., at $t = 1$ when observation starts. Based on prior knowledge, we multiply the prior distribution by the likelihood based on observation at $t = 1$ to obtain the posterior distribution at $t = 1$ (Bayesian updating). Since the consideration of likelihood increases the accuracy, the variance of the posterior distribution is smaller than that of the prior distribution. This posterior distribution is the result of estimating where the cat is at the beginning of the observation.

Subsequently, we estimate the cat's position at the next time point $t = 2$ after it moves. The prior distribution at $t = 2$ corresponds to the posterior distribution at $t = 1$. Since the cat is moving at this time, we reflect the movement in this prior distribution (state transition) based on the average speed. The variance of the

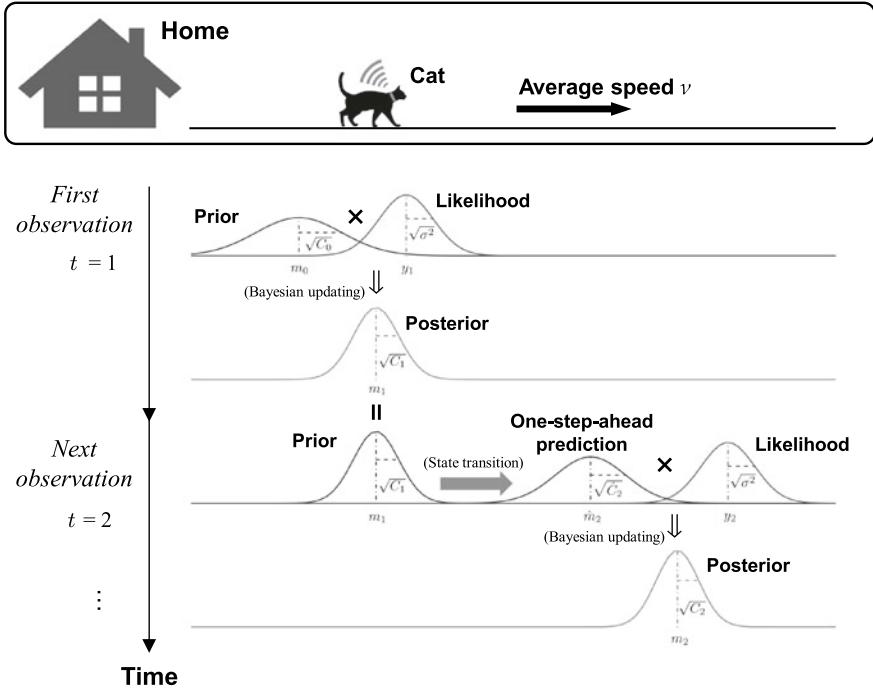


Fig. 6.2 Estimate the cat's position

one-step-ahead predictive distribution is unavoidably greater than that of the prior distribution because a fluctuation w_t in the moving speed increases the uncertainty of the movement. This one-step-ahead predictive distribution predicts the situation after the movement; hence, we multiply this predictive distribution by the likelihood based on the observation at $t = 2$, resulting in a posterior distribution at $t = 2$ (Bayesian updating). Since the consideration of likelihood increases the accuracy, the variance of the posterior distribution is less than that of the one-step-ahead predictive distribution. This posterior distribution is the result of estimating where the cat is at the next time point.

In such a manner, the target distribution for estimation is updated while repeating the state transition and Bayesian updating. We now confirm the same contents through the expansion of the equation.

First, Bayesian updating at time point $t = 1$ is expressed from Eqs. (2.19) and (2.7) as follows:

$$\begin{aligned} p(x_1 | y_1) &\propto p(x_1) \times p(y_1 | x_1) \\ &= \frac{1}{\sqrt{2\pi C_0}} \exp \left\{ -\frac{(x_1 - m_0)^2}{2C_0} \right\} \times \frac{1}{\sqrt{2\pi \sigma^2}} \exp \left\{ -\frac{(y_1 - x_1)^2}{2\sigma^2} \right\} \end{aligned}$$

focus on the estimation target x_1 only

$$\begin{aligned} &\propto \exp \left\{ -\frac{x_1^2 - 2m_0x_1}{2C_0} - \frac{-2y_1x_1 + x_1^2}{2\sigma^2} \right\} \\ &= \exp \left\{ -\left(\frac{1}{2C_0} + \frac{1}{2\sigma^2} \right) x_1^2 + \left(\frac{2m_0x_1}{2C_0} + \frac{2y_1x_1}{2\sigma^2} \right) \right\} \\ &= \exp \left\{ -\frac{1}{2} \left(\frac{1}{C_0} + \frac{1}{\sigma^2} \right) x_1^2 + \left(\frac{m_0}{C_0} + \frac{y_1}{\sigma^2} \right) x_1 \right\}. \end{aligned} \quad (6.6)$$

The posterior distribution at $t = 1$ is also expressed from Eq. (2.7) as follows:

$$p(x_1 | y_1) = \frac{1}{\sqrt{2\pi C_1}} \exp \left\{ -\frac{(x_1 - m_1)^2}{2C_1} \right\}$$

focus on the estimation target x_1 again

$$\propto \exp \left\{ -\frac{x_1^2 - 2m_1x_1}{2C_1} \right\} = \exp \left\{ -\frac{1}{2} \frac{1}{C_1} x_1^2 + \frac{m_1}{C_1} x_1 \right\}. \quad (6.7)$$

Comparing Eqs. (6.6) and (6.7) yields the following relationships:

$$\frac{1}{C_1} = \frac{1}{C_0} + \frac{1}{\sigma^2}, \quad (6.8)$$

$$m_1 = C_1 \frac{1}{C_0} m_0 + C_1 \frac{1}{\sigma^2} y_1 = \frac{\frac{1}{C_0}}{\frac{1}{C_0} + \frac{1}{\sigma^2}} m_0 + \frac{\frac{1}{\sigma^2}}{\frac{1}{C_0} + \frac{1}{\sigma^2}} y_1 \quad (6.9)$$

deform for recursion

$$\begin{aligned} &= \frac{\sigma^2 m_0 + C_0 y_1}{\sigma^2 + C_0} \\ &= \frac{\sigma^2 + C_0 - C_0}{\sigma^2 + C_0} m_0 + \frac{C_0}{\sigma^2 + C_0} y_1 \\ &= m_0 + \frac{C_0}{\sigma^2 + C_0} (y_1 - m_0). \end{aligned} \quad (6.10)$$

Equation (6.8) shows that the precision, i.e., the inverse of the variance of the posterior distribution is the sum of the precisions of the prior distribution and the likelihood. Such a fact implies that the precision or variance always increases or decreases through Bayesian updating because the certainty increases by considering observations. On the contrary, Eq. (6.9) shows that the mean for the posterior distribution is a weighted sum of the means for prior distribution and the likelihood; each weight is proportional to each precision. The deformed Eq. (6.10) for recursion also shows that the prediction is corrected based on the observations; recall that m_0 is \hat{m}_1 .

Next, from the state equation (6.4) and Eqs. (2.6) and (2.5), the state transition at time point $t = 2$ is expressed as follows:

$$\begin{aligned}\hat{C}_2 &= \text{Var}[x_1] + \text{Var}[\nu] + \text{Var}[w_2] + (\text{all covariance } 0) \\ &= C_1 + 0 + \sigma_w^2 = C_1 + \sigma_w^2,\end{aligned}\quad (6.11)$$

$$\hat{m}_2 = \mathbb{E}[x_1] + \mathbb{E}[\nu] + \mathbb{E}[w_2] = m_1 + \nu + 0 = m_1 + \nu.\quad (6.12)$$

The above expression (6.11) shows that the variance always increases with the state transition.

Finally, we confirm the Bayesian updating at time point $t = 2$. Since the mechanism of Bayesian updating remains the same as that at $t = 1$, we can obtain the result by replacing $m_0 (= \hat{m}_1)$, $C_0 (= \hat{C}_1)$, m_1 , C_1 , and y_1 at $t = 1$ by \hat{m}_2 , \hat{C}_2 , m_2 , C_2 , and y_2 at $t = 2$, respectively. Specifically, the Bayesian updating at time point $t = 2$ is derived from Eqs. (6.8) and (6.10) as follows:

$$\frac{1}{C_2} = \frac{1}{\hat{C}_2} + \frac{1}{\sigma^2} = \frac{1}{C_1 + \sigma_w^2} + \frac{1}{\sigma^2},\quad (6.13)$$

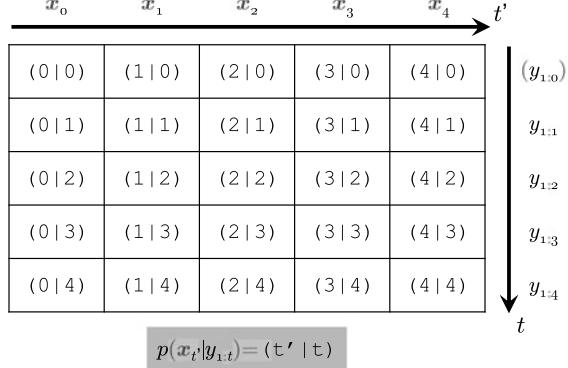
$$m_2 = \hat{m}_2 + \frac{\hat{C}_2}{\sigma^2 + \hat{C}_2} (y_2 - \hat{m}_2) = (m_1 + \nu) + \frac{C_1 + \sigma_w^2}{\sigma^2 + C_1 + \sigma_w^2} (y_2 - (m_1 + \nu)).\quad (6.14)$$

The above result corresponds to a simple example of Kalman filtering. We subsequently supplement Eq. (6.14) for estimating the mean. This equation has a prediction-error correction structure: the mean \hat{m}_2 for the one-step-ahead predictive distribution is corrected by observations y_2 . This structure is a general result obtained by the combination of state transition and Bayesian updating. In addition, the equation can take the form $\star y_2 + (1 - \star) \hat{m}_2$; therefore, it also has a structure of an exponentially weighted moving average. Such a viewpoint suggests that estimating the mean of the state is equivalent to the exponentially weighted moving average (with optimal weight \star). This insight is crucial knowledge established in general. Even in a complicated problem, it would be helpful to examine the analyzed results based on the notion that “point estimation of the state corresponds to the moving average.” The denominator in the weight \star corresponds to a variance of one-step-ahead predictive likelihood according to the observation equation (6.5), i.e., the sum of the variance \hat{C}_2 of the one-step-ahead predictive distribution and variance σ^2 of the observation noise.

6.2.2 Conceptual Diagram of Recursion

When explaining recursion for the marginal posterior distribution of the state in Eq. (6.3), we often use a conceptual diagram showing the relation among distribu-

Fig. 6.3 Relationship among distributions



tions [5]. This book also uses a conceptual diagram as a supplement for formulating recursions; hence, we provide a preliminary explanation in advance. Figure 6.3 shows the diagram where the horizontal and vertical axes indicate t' and t , respectively, and each cell is labeled with the simplified notation $(t' | t)$ representing the posterior distribution in Eq. (6.3). Note that $y_{1:0}$ does not exist and is assumed to be an empty set. We show the calculation flow for the recursion by adding arrows in this figure. While the recursion implies that such a flow reaches the destination via each cell with an upward, downward, leftward, or rightward step, the upward direction is not considered because the time progress is expressed as downward.

6.2.3 Formulation of Filtering Distribution

We now formulate the recursions for the marginal posterior distribution of the state. First, we start from the formulation of the filtering distribution because filtering is the most fundamental mechanism that provides the basis of prediction and smoothing as explained at the beginning of this section.

For filtering, $t' = t$ is set in Eq. (6.3), the filtering distribution becomes $p(x_t | y_{1:t})$.

Since recursive formulas for the filtering distribution require expressing the element about $t - 1$ clearly, we deform $y_{1:t}$ in the filtering distribution to ensure that its elements through $t - 1$ can be expressed explicitly as follows:

$$\begin{aligned} p(x_t | y_{1:t}) \\ = p(x_t | y_t, y_{1:t-1}) \end{aligned}$$

from the relation $p(a, b | c) = p(a | b, c)p(b | c)$

$$= \frac{p(x_t, y_t | y_{1:t-1})}{p(y_t | y_{1:t-1})}$$

replace the order of the first two variables in the numerator considering the ease of viewing

$$= \frac{p(y_t, \mathbf{x}_t | y_{1:t-1})}{p(y_t | y_{1:t-1})}$$

from the relation $p(a, b | c) = p(a | b, c)p(b | c)$ again

$$= \frac{p(y_t | \mathbf{x}_t, y_{1:t-1})p(\mathbf{x}_t | y_{1:t-1})}{p(y_t | y_{1:t-1})}$$

from the conditional independence in Eq. (5.2)

$$= \frac{p(y_t | \mathbf{x}_t)p(\mathbf{x}_t | y_{1:t-1})}{p(y_t | y_{1:t-1})}, \quad (6.15)$$

where $p(\mathbf{x}_t | y_{1:t-1})$ in the numerator is referred to as the *one-step-ahead predictive distribution*, and the denominator $p(y_t | y_{1:t-1})$ is referred to as the *one-step-ahead predictive likelihood*. We further deform both considering the ease of viewing.

The relation $p(a, b | c) = p(a | b, c)p(b | c)$ is also a kind of Bayes' theorem, which can be shown as follows.

We first apply the product rule of probability to $p(a, b, c)$ considering a and b as a single block:

$$p(a, b, c) = p(a, b | c)p(c).$$

We then apply the product rule of probability to $p(a, b, c)$ considering b and c as a single block:

$$p(a, b, c) = p(a | b, c)p(b, c)$$

apply the product rule of probability to the last term $p(b, c)$

$$= p(a | b, c)p(b | c)p(c).$$

From the above, $p(a, b | c)p(c) = p(a | b, c)p(b | c)p(c)$ holds. Thus, we have $p(a, b | c) = p(a | b, c)p(b | c)$.

First, since the one-step-ahead predictive distribution $p(\mathbf{x}_t | y_{1:t-1})$ is the distribution of the state \mathbf{x}_t , based on the conditional independence on the state in Eq. (5.1), we clarify the relation with \mathbf{x}_{t-1} as follows:

$$p(\mathbf{x}_t \mid y_{1:t-1})$$

apply the marginalization in reverse

$$= \int p(\mathbf{x}_t, \mathbf{x}_{t-1} \mid y_{1:t-1}) d\mathbf{x}_{t-1}$$

from the relation $p(a, b \mid c) = p(a \mid b, c)p(b \mid c)$

$$= \int p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, y_{1:t-1}) p(\mathbf{x}_{t-1} \mid y_{1:t-1}) d\mathbf{x}_{t-1}$$

from the conditional independence in Eq. (5.1)

$$= \int p(\mathbf{x}_t \mid \mathbf{x}_{t-1}) p(\mathbf{x}_{t-1} \mid y_{1:t-1}) d\mathbf{x}_{t-1}. \quad (6.16)$$

Next, since the one-step-ahead predictive likelihood $p(y_t \mid y_{1:t-1})$ is the distribution of observation y_t , based on the conditional independence of the observations in Eq. (5.2), we clarify the relation with \mathbf{x}_t as follows:

$$p(y_t \mid y_{1:t-1})$$

apply marginalization in reverse

$$= \int p(y_t, \mathbf{x}_t \mid y_{1:t-1}) d\mathbf{x}_t$$

from the relation $p(a, b \mid c) = p(a \mid b, c)p(b \mid c)$

$$= \int p(y_t \mid \mathbf{x}_t, y_{1:t-1}) p(\mathbf{x}_t \mid y_{1:t-1}) d\mathbf{x}_t$$

from conditional independence in Eq. (5.2)

$$= \int p(y_t \mid \mathbf{x}_t) p(\mathbf{x}_t \mid y_{1:t-1}) d\mathbf{x}_t. \quad (6.17)$$

From the above, the filtering recursion is summarized as follows:

$$\text{filtering distribution} \quad p(\mathbf{x}_t \mid y_{1:t}) = p(\mathbf{x}_t \mid y_{1:t-1}) \frac{p(y_t \mid \mathbf{x}_t)}{p(y_t \mid y_{1:t-1})}, \quad (6.18)$$

one-step-ahead
predictive distribution

$$\begin{aligned} p(\mathbf{x}_t \mid y_{1:t-1}) \\ = \int p(\mathbf{x}_t \mid \mathbf{x}_{t-1}) p(\mathbf{x}_{t-1} \mid y_{1:t-1}) d\mathbf{x}_{t-1}, \end{aligned} \quad (6.19)$$

one-step-ahead
predictive likelihood

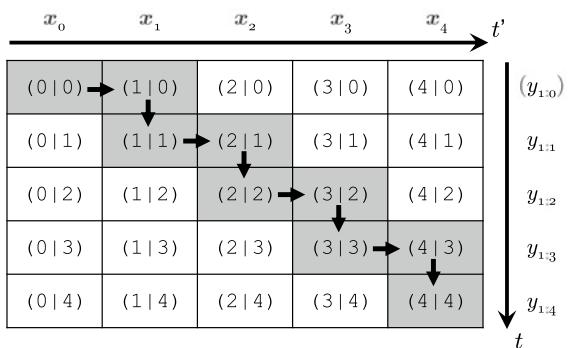
$$\begin{aligned} p(y_t \mid y_{1:t-1}) \\ = \int p(y_t \mid \mathbf{x}_t) p(\mathbf{x}_t \mid y_{1:t-1}) d\mathbf{x}_t. \end{aligned} \quad (6.20)$$

The points about Eqs. (6.18), (6.19), and (6.20) are as follows:

- Filtering distribution in Eq. (6.18): the one-step-ahead predictive distribution is corrected based on the likelihood $p(y_t \mid \mathbf{x}_t)$.
- One-step-ahead predictive distribution in Eq. (6.19): the filtering distribution one time before is transitioned in the time-forward direction based on the state equation.
- One-step-ahead predictive likelihood in Eq. (6.20): the one-step-ahead predictive distribution is converted in the domain of observations based on the observation equation; this likelihood also corresponds to the normalization factor for the numerators in Eq. (6.18).

Now add the filtering expression in a conceptual diagram Fig. 6.3. From the above explanation, filtering can be achieved by ① obtaining the one-step-ahead predictive distribution through the state transition from the filtering distribution one time before and then ② correcting the obtained distribution based on the observations. The flow of this processing in the filtering is shown in Fig. 6.4. The processes ① and ② are represented by the horizontal (right) and vertical arrows, respectively, and their combination describes the entire filtering process through a stepwise shape.

Fig. 6.4 Relationship among distributions (filtering)



6.2.4 Formulation of Predictive Distribution

We now consider k -steps-ahead prediction with $t' = t + k$ in Eq. (6.3). The corresponding k -steps-ahead *predictive distribution* is $p(\mathbf{x}_{t+k} | y_{1:t})$. Since recursive formulas for the predictive distribution require expressing the element of $t + k - 1$ clearly, we deform the predictive distribution to ensure that the element \mathbf{x}_{t+k-1} can be expressed explicitly as follows:

$$p(\mathbf{x}_{t+k} | y_{1:t})$$

apply marginalization in reverse

$$= \int p(\mathbf{x}_{t+k}, \mathbf{x}_{t+k-1} | y_{1:t}) d\mathbf{x}_{t+k-1}$$

from the relation $p(a, b | c) = p(a | b, c)p(b | c)$

$$= \int p(\mathbf{x}_{t+k} | \mathbf{x}_{t+k-1}, y_{1:t}) p(\mathbf{x}_{t+k-1} | y_{1:t}) d\mathbf{x}_{t+k-1}$$

from the conditional independence suggested by Eq. (5.1)

$$= \int p(\mathbf{x}_{t+k} | \mathbf{x}_{t+k-1}) p(\mathbf{x}_{t+k-1} | y_{1:t}) d\mathbf{x}_{t+k-1}. \quad (6.21)$$

Thus, the recursion for the predictive distribution is expressed as follows:

$$\begin{aligned} \text{predictive distribution } & p(\mathbf{x}_{t+k} | y_{1:t}) \\ &= \int p(\mathbf{x}_{t+k} | \mathbf{x}_{t+k-1}) p(\mathbf{x}_{t+k-1} | y_{1:t}) d\mathbf{x}_{t+k-1}. \end{aligned} \quad (6.22)$$

The point about Eq. (6.22) is as follows:

- The k -steps-ahead predictive distribution in Eq. (6.22): the $k - 1$ -steps-ahead predictive distribution is transitioned in the forward time direction based on the state equation.

Hence, the following relation continues to hold in a chain:

- The $k - 1$ -steps-ahead predictive distribution: the $k - 2$ -steps-ahead predictive distribution is transitioned in the forward time direction based on the state equation.
 - The $k - 2$ -steps-ahead predictive distribution: the $k - 3$ -steps-ahead predictive distribution is transitioned in the forward time direction based on the state equation.
- ...

The cumulative relation of the above chains is summarized as follows:

- Starting from the one-step-ahead predictive distribution and repeating the transition in the forward time direction $k - 1$ times based on the state equation, we obtain the k -steps-ahead predictive distribution.

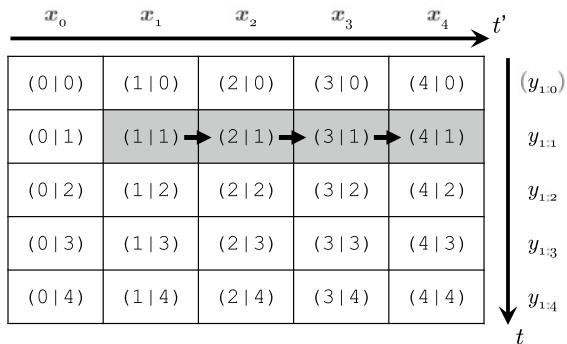
Recall that if the filtering distribution one time before is transitioned in the forward time direction based on the state equation, we can obtain the one-step-ahead predictive distribution. Therefore, we can conclude as follows:

- Starting from the filtering distribution $p(x_t | y_{1:t})$ and repeating the transition in the forward time direction k times based on the state equation, we obtain the k -steps-ahead predictive distribution.

We explain how k -steps-ahead prediction is expressed in the conceptual diagram. From the above explanation, the k -steps-ahead prediction can be achieved by repeating the state transition k times for the filtering distribution. The flow of this processing in the k -steps-ahead prediction is shown in Fig. 6.5. For example, the processing to obtain three-steps-ahead prediction from the current time point $t = 1$ is represented by repeating the horizontal (right) arrow three times.

Comparing the k -steps-ahead prediction with filtering, we can see that the state transition (horizontal arrow) is repeated without correction based on the observations (vertical arrow). Hence, while the accuracy of prediction gradually decreases as k increases, the most likely distributions continue to be generated based on the information through the filtering point. Incidentally, the state-space model applies this concept directly to the missing observations. Therefore, the missing observations in the state-space model are compensated for with prediction values. This is the background showing that it is easy to handle missing observations in the state-space model.

Fig. 6.5 Relationship among distributions (prediction)



6.2.5 Formulation of the Smoothing Distribution

We now consider smoothing. This book focuses on the popular fixed-interval smoothing; hence, we replace the t and t' in Eq. (6.3) with T and t , respectively.

For fixed-lag and -point smoothings, see [1, 3].

In this case, the smoothing distribution is $p(\mathbf{x}_t | y_{1:T})$. To perform smoothing, we first assume that filtering has been completed through the time point T . In addition, we assume that sequential updating in the reverse time direction. The recursive formulas for the smoothing distribution require expressing the element about $t + 1$ clearly. Hence, we deform the smoothing distribution to ensure that the element \mathbf{x}_{t+1} can be expressed explicitly as follows:

$$p(\mathbf{x}_t | y_{1:T})$$

apply marginalization in reverse

$$= \int p(\mathbf{x}_t, \mathbf{x}_{t+1} | y_{1:T}) d\mathbf{x}_{t+1}$$

from the relation $p(a, b | c) = p(a | b, c)p(b | c)$

$$= \int p(\mathbf{x}_t | \mathbf{x}_{t+1}, y_{1:T}) p(\mathbf{x}_{t+1} | y_{1:T}) d\mathbf{x}_{t+1}$$

replace the order of the first and second terms considering the ease of viewing

$$= \int p(\mathbf{x}_{t+1} | y_{1:T}) p(\mathbf{x}_t | \mathbf{x}_{t+1}, y_{1:T}) d\mathbf{x}_{t+1}$$

apply the conditional independence in Eq. (5.3) to the last term $p(\mathbf{x}_t | \mathbf{x}_{t+1}, y_{1:T})$

$$= \int p(\mathbf{x}_{t+1} | y_{1:T}) p(\mathbf{x}_t | \mathbf{x}_{t+1}, y_{1:t}) d\mathbf{x}_{t+1}$$

apply the relation $p(a, b | c) = p(a | b, c)p(b | c)$ to the last term $p(\mathbf{x}_t | \mathbf{x}_{t+1}, y_{1:t})$

$$= \int p(\mathbf{x}_{t+1} | y_{1:T}) \frac{p(\mathbf{x}_t, \mathbf{x}_{t+1} | y_{1:t})}{p(\mathbf{x}_{t+1} | y_{1:t})} d\mathbf{x}_{t+1}$$

modify the order of variables in the numerator $p(\mathbf{x}_t, \mathbf{x}_{t+1} | y_{1:t})$ considering the ease of viewing

$$= \int p(\mathbf{x}_{t+1} | y_{1:T}) \frac{p(\mathbf{x}_{t+1}, \mathbf{x}_t | y_{1:t})}{p(\mathbf{x}_{t+1} | y_{1:t})} d\mathbf{x}_{t+1}$$

apply the relation $p(a, b | c) = p(a | b, c)p(b | c)$ to the numerator $p(\mathbf{x}_{t+1}, \mathbf{x}_t | y_{1:t})$

$$= \int p(\mathbf{x}_{t+1} | y_{1:T}) \frac{p(\mathbf{x}_{t+1} | \mathbf{x}_t, y_{1:t})p(\mathbf{x}_t | y_{1:t})}{p(\mathbf{x}_{t+1} | y_{1:t})} d\mathbf{x}_{t+1}$$

from the fact that $p(\mathbf{x}_t | y_{1:t})$ in the numerator does not depend on \mathbf{x}_{t+1}

$$= p(\mathbf{x}_t | y_{1:t}) \int p(\mathbf{x}_{t+1} | y_{1:T}) \frac{p(\mathbf{x}_{t+1} | \mathbf{x}_t, y_{1:t})}{p(\mathbf{x}_{t+1} | y_{1:t})} d\mathbf{x}_{t+1}$$

apply conditional independence as suggested by Eq. (5.1) to the numerator $p(\mathbf{x}_{t+1} | \mathbf{x}_t, y_{1:t})$

$$= p(\mathbf{x}_t | y_{1:t}) \int p(\mathbf{x}_{t+1} | y_{1:T}) \frac{p(\mathbf{x}_{t+1} | \mathbf{x}_t)}{p(\mathbf{x}_{t+1} | y_{1:t})} d\mathbf{x}_{t+1}. \quad (6.23)$$

Thus, the recursion for the smoothing distribution can be expressed as follows:

$$\begin{aligned} \text{smoothing distribution } & p(\mathbf{x}_t | y_{1:T}) \\ &= p(\mathbf{x}_t | y_{1:t}) \int \frac{p(\mathbf{x}_{t+1} | \mathbf{x}_t)}{p(\mathbf{x}_{t+1} | y_{1:t})} p(\mathbf{x}_{t+1} | y_{1:T}) d\mathbf{x}_{t+1}. \end{aligned} \quad (6.24)$$

The point about Eq. (6.24) is as follows:

- Smoothing distribution in Eq. (6.24): the filtering distribution is corrected based on the smoothing distribution at one time ahead.

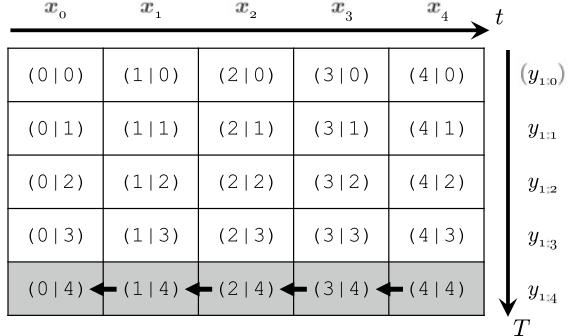
Hence, the following relation continues to hold in a chain:

- Smoothing distribution at time point $T - 1$: the filtering distribution at time point $T - 1$ is corrected based on the smoothing distribution at time point T .
- Smoothing distribution at time point $T - 2$: the filtering distribution at time point $T - 2$ is corrected based on the smoothing distribution at time point $T - 1$.
- ...

The smoothing distribution at time point T corresponds to the filtering distribution at time point T .

We explain how smoothing is expressed in the conceptual diagram. From the above explanation, the smoothing at each time point can be achieved by repeating the smoothing recursion in the reverse time direction, starting from the filtering at time point T . Figure 6.6 shows the flow of this processing in the smoothing. For example, supposing a situation at $T = 4$, the processing to obtain the smoothing distribution at $t = 0$ is represented by repeating the horizontal (left) arrow four times.

Fig. 6.6 Relationship among distributions (smoothing)



6.3 Likelihood and Model Selection in the State-Space Model

While this book has briefly mentioned likelihood, this section explains likelihood again because this book treats it as a critical criterion for specifying models or parameters.

First, as described in Sect. 2.8, the likelihood for an entire time series y_t ($t = 1, 2, \dots, T$) is expressed as $p(y_1, y_2, \dots, y_T; \theta)$. This term can be decomposed as follows:

$$\begin{aligned} p(y_1, y_2, \dots, y_T; \theta) &= p(y_{1:T}; \theta) \\ &= p(y_T, y_{1:T-1}; \theta) \end{aligned}$$

apply the product rule of probability

$$= p(y_T \mid y_{1:T-1}; \theta)p(y_{1:T-1}; \theta)$$

apply the product rule of probability to the last term repeatedly \dots

$$= \prod_{t=1}^T p(y_t \mid y_{1:t-1}; \theta), \quad (6.25)$$

where $y_{1:0}$ does not exist and is assumed to be the empty set \emptyset : $p(y_1 \mid y_{1:0}; \theta) = p(y_1 \mid \emptyset; \theta) = p(y_1; \theta)$.

Taking the logarithm of the likelihood in Eq. (6.25) yields the log-likelihood as

$$\begin{aligned} \ell(\theta) &= \log p(y_1, y_2, \dots, y_T; \theta) = \log p(y_{1:T}; \theta) \\ &= \sum_{t=1}^T \log p(y_t \mid y_{1:t-1}; \theta). \end{aligned} \quad (6.26)$$

From the above Eq. (6.26), we see that the log-likelihood for the entire time series can be represented as the sum of the logarithms of the one-step-ahead predictive likelihoods. Recalling that the filtering recursion derives the one-step-ahead predictive likelihood during its processing, we see that the likelihood can be calculated efficiently through filtering.

Next, we examine the meaning of likelihood in the state-space model.

Based on the fact that the state equation in the state-space model represents the time correlation among data, we can regard Eq. (6.25) or (6.26) as a numerical indicator of the extent to which observations fit the model considering the time pattern. In addition, from Eq. (6.25) or (6.26), we see that the likelihood for the entire time series comprises the one-step-ahead predictive likelihood; hence, we can regard the likelihood as an indicator considering the predictive ability to provide future values based only on the past values (however, specific observations, not multiple ones). From such characteristics, this book treats the likelihood as a model selecting criterion.

Concerning more advanced topics, information criteria such as Akaike information criterion (AIC) and widely applicable information criterion (WAIC) are often used for model selection. For details of the information criteria, see [2, 5–7].

Incidentally, Eq. (6.25) or (6.26) is a representation wherein the state does not appear explicitly; hence, we can consider that it has been marginalized or averaged over the state. For this reason, Eq. (6.25) or (6.26) is sometimes referred to as the *marginal likelihood* or *log marginal likelihood*, respectively. From a practical viewpoint, we can consider that marginalization suppresses the extreme values depending on individual states and makes it easier to obtain stable values [4].

6.4 Treatment of Parameters in the State-Space Model

While we have assumed that parameters are implicitly known, such a case is rare; parameters are usually unknown in real situations. When the parameters are unknown, we must specify their values in some manner for the estimation of time series. Several methods have been developed for such a purpose. This section describes the basic idea regarding typical methods.

6.4.1 When Parameters are Not Regarded as Random Variables

The approach is based on frequentism. In this case, before estimating the time series, we typically specify the point estimates for the parameter using the maximum likelihood method. A *maximum likelihood estimator* $\hat{\boldsymbol{\theta}}$ for the parameters in the state-space model is expressed from Eq. (6.26) as follows:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \ell(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \log p(y_{1:T}; \boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{t=1}^T \log p(y_t | y_{1:t-1}; \boldsymbol{\theta}). \quad (6.27)$$

According to the above, we obtain the maximum likelihood estimator for parameters based on a particular number of observations. This is achieved through several methods: a direct search for different values of parameters, quasi-Newton method, when a numerical optimization is applicable, and the expectation maximization (EM) algorithm.

6.4.2 When Parameters are Regarded as Random Variables

The approach is based on Bayesianism. Unlike the case of frequentism, treating parameters as random variables makes it possible to consider their uncertainties more appropriately. In this case, for the estimation of a time series, Bayesian estimation is applied to both parameters and state.

In a Bayesian estimation, a parameter becomes a kind of state in the state-space model: we treat the augmented state $= \{\mathbf{x}, \boldsymbol{\theta}\}$ comprising the original state \mathbf{x} and added parameters $\boldsymbol{\theta}$. Thus, the joint distribution of the state-space model is extended from Eq. (5.6) as follows:

$$\begin{aligned} & p(\mathbf{x}_{0:T}, y_{1:T}, \boldsymbol{\theta}) \\ &= p(\mathbf{x}_0, \boldsymbol{\theta}) \prod_{t=1}^T p(y_t | \mathbf{x}_t, \boldsymbol{\theta}) p(\mathbf{x}_t | \mathbf{x}_{t-1}, \boldsymbol{\theta}) \end{aligned}$$

apply the product rule of probability to the first term $p(\mathbf{x}_0, \boldsymbol{\theta})$

$$= p(\mathbf{x}_0 | \boldsymbol{\theta}) p(\boldsymbol{\theta}) \prod_{t=1}^T p(y_t | \mathbf{x}_t, \boldsymbol{\theta}) p(\mathbf{x}_t | \mathbf{x}_{t-1}, \boldsymbol{\theta}). \quad (6.28)$$

The posterior distribution of the state is also extended from Eqs. (6.2) and (6.3). Now, the joint posterior distribution of state and parameter is examined. Specifically, Eq. (6.2) is extended as

$$p(\mathbf{x}_{0,1}, \dots, t', \dots, \theta | y_{1:t}) = p(\mathbf{x}_{0,1}, \dots, t', \dots | \theta, y_{1:t}) p(\theta | y_{1:t}), \quad (6.29)$$

and Eq. (6.3) is also extended as

$$p(\mathbf{x}_{t'}, \theta | y_{1:t}) = p(\mathbf{x}_{t'} | \theta, y_{1:t}) p(\theta | y_{1:t}). \quad (6.30)$$

If only the parameter is of interest, then we marginalize out the state in the above equations to obtain the marginal posterior distribution $p(\theta | y_{1:t})$. Furthermore deriving the point estimator of the parameter, we can obtain it by maximizing this marginal posterior distribution $p(\theta | y_{1:t})$ as follows:

$$\operatorname{argmax}_{\theta} p(\theta | y_{1:t})$$

from Bayes' theorem

$$= \operatorname{argmax}_{\theta} \frac{p(y_{1:t} | \theta) p(\theta)}{p(y_{1:t})}$$

recall that \propto denotes the proportional relationship

$$\propto \operatorname{argmax}_{\theta} p(y_{1:t} | \theta) p(\theta). \quad (6.31)$$

The above expression is a *maximum a posteriori probability (MAP)* estimation. In addition, taking the logarithm of Eq. (6.31), we have the following equation:

$$\operatorname{argmax}_{\theta} \log p(\theta | y_{1:t}) \propto \operatorname{argmax}_{\theta} \{\log p(y_{1:t} | \theta) + \log p(\theta)\}. \quad (6.32)$$

Comparing Eq. (6.32) with (6.27), we see that the MAP estimation is achieved by adding the correction with θ 's prior distribution $p(\theta)$ to the maximum likelihood estimation. On the contrary, it can be said that the MAP estimation without correction with the prior distribution is equivalent to the maximum likelihood estimation.

Regarding the parameter estimation, an alternative approach examines the marginal posterior distribution $p(\theta | y_{1:t})$ directly and then selects its representative value, such as the mode, mean, and median, as the point estimator. Such an approach is safer because it provides an overview of the distribution as well.

Finally, we supplement the augmented state-space model. The augmented state-space model includes parameters as random variables in addition to the original state; hence, it becomes a general state-space model even if the original model is linear Gaussian. Therefore, such an augmented state-space model cannot be solved directly using the Wiener and Kalman filters, which are solutions for the linear Gaussian state-space model, and we must use other methods, such as solution using MCMC and particle filter; see also Sect. 4.7.

References

1. Cappé, O., Moulines, E., Rydén, T.: *Inference in Hidden Markov Models*. Springer (2005)
2. Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B., Vehtari, A., Rubin, D.B.: *Bayesian Data Analysis*, 3rd edn. CRC Press (2013)
3. Higuchi, T., Ueno, G., Nakano, S., Nakamura, K., Yoshida, R.: *Introduction to Data Assimilation—Next Generation Simulation Technology—*. Asakura Publishing Co. Ltd, Shinjuku, Tokyo (2011). [in Japanese]
4. Iba, Y.: Learning and hierarchy—from the Bayesian statistics perspective—. *Condensed Matter. Phys. Rep.* **65**(5), 657–677 (1996). [in Japanese]
5. Kitagawa, G.: *Introduction to Time Series Modeling*. CRC Press (2009)
6. Konishi, S., Kitagawa, G.: *Information Criteria and Statistical Modeling*. Springer (2008)
7. Watanabe, S.: *Mathematical Theory of Bayesian Statistics*. CRC Press (2018)

Chapter 7

Batch Solution for Linear Gaussian State-Space Model



This chapter describes the batch estimation method for the case wherein a particular number of observations already exist in the linear Gaussian state-space model. This solution is called the Wiener filter. The Wiener filter has the historical significance of being the basis of the Kalman filter; this chapter briefly outlines its features.

7.1 Wiener Filter

The optimal batch estimation method for the linear Gaussian state-space model is known as the *Wiener filter* named after N. Wiener, who first derived this expression clearly; the word “optimal” means minimizing the mean squared error between the point estimates and true values of data to be estimated.

A. N. Kolmogorov also derived an equivalent result almost at the same time; hence, this filter is sometimes referred to as the Kolmogorov filter, Wiener–Kolmogorov filter, and linear minimum mean square error (LMMSE) filter depending on the specialized field.

The Wiener filter assumes the stationarity of the time series and was first derived in the frequency domain without using the state-space model. This filter is typically classified as batch type solution, and we also explain it from that perspective; see Sect. 4.7. Regarding the derivation of the Wiener filter [3–5], Appendix E.1 summarizes the details.

We now confirm the relation between the Wiener and Kalman filters. While we have already mentioned that the Wiener filter was derived initially without using the state-space model, it can be rewritten into a sequential solution based on the state-space model with time-invariant parameters. This approach was clarified after the initial derivation, and its consistency with the Kalman filter has been confirmed [3]. In other words, the Wiener and Kalman filters yield the same results for a stationary process.

This book briefly explains the outline of the Wiener filter in accordance with its historical significance, because the author recognizes the Kalman filter as a more critical solution for the linear Gaussian state-space model in general. While the Kalman filter is a sequential solution, it can also be applied to cases involving a particular number of observations. Furthermore, in contrast to the Wiener filter, the Kalman filter can handle nonstationary processes adequately.

Given the above description, what is the benefit of the Wiener filter currently beyond its historical significance? In answer to this question, the author believes that the Wiener filter has a substantial advantage in numerical computation. For example, to reduce the computational complexity for multivariate time series data, [2] has proposed an approximate two-step solution, which first performs one-dimensional filtering in the series direction and then in the time direction. In addition, the Wiener filter can avoid the accumulation of numerical errors due to sequential processing such as the Kalman filter.

7.1.1 *Wiener Smoothing*

We now present the formulation of the Wiener filter. While the Wiener filter can formulate each of smoothing, filtering, and prediction, the formulation of smoothing is the most natural and straightforward among them. For this reason, this book focuses on the formulation of smoothing for the Wiener filter.



Fig. 7.1 Observation and Wiener filter

Filtering and prediction for the Wiener filter in the frequency domain require a technique called spectral factorization [3–5] introduced by Wiener, which has not been explained in this book. Incidentally, the developing history of the Wiener filter is opposite to that of the Kalman filter; the Kalman filter was initially published as a filtering theory; its use in smoothing was formulated later.

First, consider the system shown in Fig. 7.1.

This figure assumes that the following equations hold:

$$y_t = x_t + v_t, \quad (7.1)$$

$$d_t = h_t \circledast y_t. \quad (7.2)$$

The above equations state that an independent white noise v_t is added to the original data x_t , followed by an observation y_t . Next, the observations y_t are passed to the Wiener filter h_t ; then, the desired signal d_t is derived as the point estimates for the original data x_t . Regarding the Wiener filter, its transfer function $H(z) = \mathcal{Z}[h_t]$ is expressed as

$$\begin{aligned} H(z) &= \frac{S_{xx}(z)}{S_{xx}(z) + S_{vv}(z)} \\ &= \frac{1}{1 + S_{vv}(z)/S_{xx}(z)}. \end{aligned} \quad (7.3)$$

In the above description, we have used several terms such as the *convolution* operator \circledast , *transfer function*, $\mathcal{Z}[\cdot]$ representing the *z-transform* (a kind of frequency domain transform), and *power spectrum* $S(z)$ without detailed explanation. See Appendix E.1 for relevant information and the derivation of the above equation.

We examine the meaning of the transfer function in Eq. (7.3). If there is no noise, then $S_{vv}(z) = 0$ and $H(z) = 1$ holds; hence, we can rely entirely on the observations. On the contrary, if the noise is dominant, such as $S_{vv}(z) = \infty$, then $H(z) = 0$ holds; hence, we cannot rely on the observations. In the case of finite noise, with an increase in the noise, $H(z)$ decreases according to the power ratio of the noise to the original data. Thus, we can understand that the noisier a frequency band is, the more the Wiener filter suppresses the observations in that frequency band.

7.2 Example: AR(1) Model Case

This section examines the results of the Wiener filter and its equivalence with the Kalman filter through a simple example.

First, suppose that the original data x_t follow the AR(1) model¹. Specifically, we assume the following linear Gaussian state-space model:

$$x_t = \phi x_{t-1} + w_t, \quad w_t \sim \mathcal{N}(0, W), \quad (7.4)$$

$$y_t = x_t + v_t, \quad v_t \sim \mathcal{N}(0, V), \quad (7.5)$$

where $|\phi| < 1$ is assumed for satisfying stationarity. In this case, we can derive the time domain expressed desired signal d_t analytically in Eq. (7.2) with the Wiener smoothing as follows:

$$d_t = \frac{(\phi^{-1} - \beta)(\phi - \beta)}{1 - \beta^2} \sum_{k=-\infty}^{\infty} \beta^{|k|} y_{t+k}, \quad (7.6)$$

where

$$\beta = \frac{\left(\frac{1}{r\phi} + \phi^{-1} + \phi\right) - \sqrt{\left(\frac{1}{r\phi} + \phi^{-1} + \phi\right)^2 - 4}}{2},$$

$$r = V/W.$$

A detailed derivation for the above is summarized in Appendix E.1. Equation (7.6) implies an exponentially weighted moving average centered on y_t .

Next, we pragmatically confirm the equivalence between the Wiener and Kalman filters using the following code.

¹ The formulation here is based on Example 5.1 in [6].

Code 7.1

```

1 > # <<Wiener and Kalman smoothings for the AR(1) model>>
2 >
3 > # Preprocessing
4 > set.seed(23)
5 > library(dlm)
6 >
7 > # Setting of state space model including AR(1)
8 > W <- 1
9 > V <- 2
10 > phi <- 0.98    # AR(1) coefficient
11 > mod <- dlmModPoly(order = 1, dW = W, dV = V, CO = 100)
12 > mod$GG[1, 1] <- phi
13 >
14 > # Generate observations using Kalman prediction
15 > t_max <- 100
16 > sim_data <- dlmForecast(mod = mod, nAhead = t_max, sampleNew = 1)
17 > y <- sim_data$newObs[[1]]
18 >
19 > # Kalman smoothing
20 > dlmSmoothed_obj <- dlmSmooth(y = y, mod = mod)
21 > s <- dropFirst(dlmSmoothed_obj$s)
22 >
23 > # Winner smoothing
24 > # Set coefficients
25 > r <- V / W
26 > b <- 1/(r*phi) + 1/phi + phi
27 > beta <- (b - sqrt(b^2 - 4)) / 2
28 >
29 > # Supplement the minimum required dummy 0s before and after the finite observations
30 > y_expand <- c(rep(0, t_max - 1), y, rep(0, t_max - 1))
31 >
32 > # Execution of Winner smoothing
33 > d <- (1/phi - beta)*(phi - beta) / (1 - beta^2) *
34 +     filter(method = "convolution",
35 +         filter = beta^abs(-(t_max-1):(t_max-1)), x = y_expand
36 +     )
37 >
38 > # Remove dummy NAs from the result
39 > d <- d[!is.na(d)]
40 >
41 > # Plot results
42 > ts.plot(cbind(y, d, s),
43 +           lty = c("solid", "dashed", "solid"),
44 +           col = c("lightgray", "red", "blue"),
45 +           ylab = "")
46 > # Legend
47 > legend(legend = c("Observations", "Winner smoothing", "Kalman smoothing"),
48 +           lty = c("solid", "dashed", "solid"),
49 +           col = c("lightgray", "red", "blue"),
50 +           x = "topright", text.width = 17, cex = 0.6)

```

The above code uses the functions `dlmModPoly()`, `dlmForecast()`, and `dlmSmooth()` of the library **dlm** for the setting of the state-space model, generation of observations, and Kalman smoothing, respectively; see Appendix B for an overview of the significant functions in **dlm**. We perform Wiener smoothing using R's generic linear filtering function `filter()`. While the Wiener smoothing assumes infinite observations, the actual observations are finite. This example provisionally relaxes this assumption using dummy observations. We perform Wiener smoothing

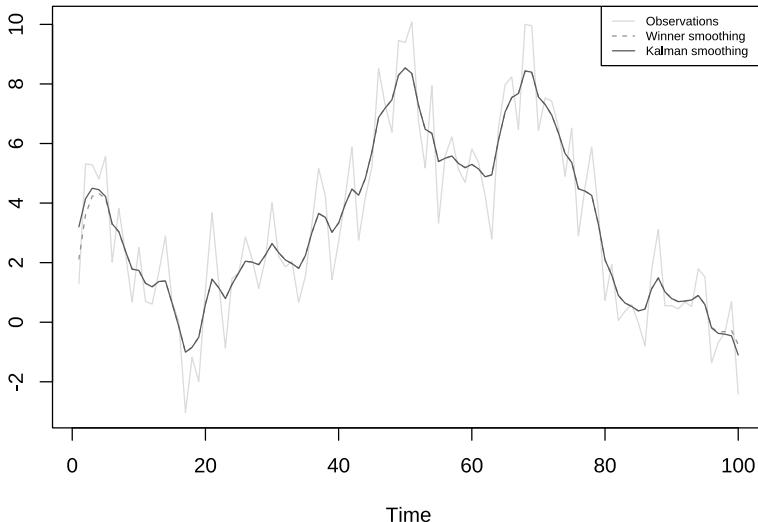


Fig. 7.2 Wiener and Kalman smoothing for the AR(1) model

after adding the minimum dummy “0” at both ends of the observations and then remove the part corresponding to the dummy from the result. Figure 7.2 shows the comparison result between the desired signal using Wiener smoothing and estimated mean for the smoothing distribution using Kalman smoothing.

As shown in Fig. 7.2, the results of Wiener and Kalman smoothings are almost identical except for the two ends of the data; the graph lines are almost overlapped and cannot be distinguished.

The deviation of estimates at both ends arises because the stationarity of the observations has been compromised contrary to the assumption for Wiener smoothing. In this example, dummy padding with 0s makes the statistical characteristics around both ends of the observations incompatible with the assumption. However, the deviation of the estimates does not continue for long but is resolved immediately, because, for the AR(1) model used in this example, the correlation between the data rapidly decreases. As the estimation time point moves away from the endpoints of data, the influence of nonstationarity at the endpoints becomes practically insignificant.

On the other hand, Kalman smoothing does not assume the stationarity of data and is not affected by the endpoints of observations.

While this book finishes an explanation of the Wiener filter here, its related study continues even now as a result of the influence of the historical background in the field of audio and image processing [1]. In addition, the seasonal adjustment method TRAMO [6] which was developed by a Spanish bank uses the Wiener filter for economic time series analysis; it is implemented in the R library **seasonal**.

References

1. Asano, F.: *Array Signal Processing for Acoustics—Localization, Tracking and Separation of Sound Sources—*. Corona Publishing Co., Ltd (2011). [in Japanese]
2. Hanzo, L., Münster, M., Choi, B.J., Keller, T.: *OFDM and MC-CDMA for Broadband Multi-User Communications, WLANs and Broadcasting*. Wiley-IEEE Press, Hoboken, New Jersey (2003)
3. Kailath, T., Sayed, A.H., Hassibi, B.: *Linear Estimation*. Prentice Hall (2000)
4. Katayama, T.: (New Edition) *Applied Kalman filter*. Asakura Publishing Co., Ltd (2000). [in Japanese]
5. Nishiyama, K.: *Optimal Filtering*. Baifukan Co., Ltd (2001). [in Japanese]
6. Takaoka, M.: *Economic Time Series and Seasonal Adjustment Method*. Asakura Publishing Co., Ltd (2015). [in Japanese]

Chapter 8

Sequential Solution for Linear Gaussian State-Space Model



This chapter describes the sequential estimation method for the case wherein the observations are obtained sequentially in the linear Gaussian state-space model. This solution is called the Kalman filter. Although the Kalman filter is a sequential solution, it can also be applied to the case wherein a particular number of observations exist.

8.1 Kalman Filter

The optimal sequential estimation method for the linear Gaussian state-space model is referred to as *Kalman filter*, named after Kalman, who first derived this expression clearly; the word “optimal” here means to minimize the mean squared error between the point estimates and true values of data to be estimated. In contrast to the Wiener filter described in the previous chapter, the Kalman filter can handle a nonstationary process without any problem.

This chapter explains filtering, smoothing, and prediction using the Kalman filter [1, 8, 10, 11, 16]. Since the Kalman filter performs only linear operations, every distribution of interest becomes a normal distribution from the reproducibility for normal distributions described in Sect. 2.3.

We now repeat the definition of the linear Gaussian state-space model, which has already been mentioned in Sect. 5.4, as a starting point for the explanation in this chapter:

$$\mathbf{x}_t = \mathbf{G}_t \mathbf{x}_{t-1} + \mathbf{w}_t, \quad \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{W}_t), \quad (8.1)$$

$$y_t = \mathbf{F}_t \mathbf{x}_t + v_t, \quad v_t \sim \mathcal{N}(0, V_t), \quad (8.2)$$

where \mathbf{G}_t is the $p \times p$ state transition matrix, \mathbf{F}_t is the $1 \times p$ observation matrix, \mathbf{W}_t is the $p \times p$ covariance matrix of the state noise, and V_t is the variance of the obser-

vation noise. Regarding the prior distribution, we also assume that $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{m}_0, \mathbf{C}_0)$, where \mathbf{m}_0 is the p -dimensional mean vector and \mathbf{C}_0 is the $p \times p$ covariance matrix. Note that listing all the parameters in this linear Gaussian state-space model yields $\boldsymbol{\theta} = \{\mathbf{G}_t, \mathbf{F}_t, \mathbf{W}_t, V_t, \mathbf{m}_0, \mathbf{C}_0\}$.

8.1.1 Kalman Filtering

We now explain Kalman filtering based on the description in Sect. 6.2.3.

As mentioned previously, the filtering distribution, one-step-ahead predictive distribution, and one-step-ahead predictive likelihood are all normal distributions in the Kalman filter; hence, we define them as follows:

$$\begin{aligned} \text{filtering distribution} & \quad \mathcal{N}(\mathbf{m}_t, \mathbf{C}_t), \text{ for the mean vector } \mathbf{m}_t \text{ and} \\ & \quad \text{covariance matrix } \mathbf{C}_t, \end{aligned} \quad (8.3)$$

$$\begin{aligned} \text{one-step-ahead} & \quad \mathcal{N}(\mathbf{a}_t, \mathbf{R}_t), \text{ for the mean vector } \mathbf{a}_t \text{ and} \\ \text{predictive distribution} & \quad \text{covariance matrix } \mathbf{R}_t, \end{aligned} \quad (8.4)$$

$$\begin{aligned} \text{one-step-ahead} & \quad \mathcal{N}(f_t, Q_t), \text{ for the mean } f_t \text{ and} \\ \text{predictive likelihood} & \quad \text{variance } Q_t. \end{aligned} \quad (8.5)$$

The procedure for obtaining the filtering distribution at time point t from that at time point $t - 1$ is as follows; see Appendix E.2 for its detailed derivation.

Algorithm 8.1 Kalman filtering

0. filtering distribution at time point $t - 1$: $\mathbf{m}_{t-1}, \mathbf{C}_{t-1}$
 1. update procedure at time point t
 - *one-step-ahead predictive distribution*
(Mean) $\mathbf{a}_t \leftarrow \mathbf{G}_t \mathbf{m}_{t-1}$
(Covariance) $\mathbf{R}_t \leftarrow \mathbf{G}_t \mathbf{C}_{t-1} \mathbf{G}_t^\top + \mathbf{W}_t$
 - *one-step-ahead predictive likelihood*
(Mean) $f_t \leftarrow \mathbf{F}_t \mathbf{a}_t$
(Covariance) $Q_t \leftarrow \mathbf{F}_t \mathbf{R}_t \mathbf{F}_t^\top + V_t$
 - *Kalman gain*
 $\mathbf{K}_t \leftarrow \mathbf{R}_t \mathbf{F}_t^\top \mathbf{Q}_t^{-1}$
 - *State update*
(Mean) $\mathbf{m}_t \leftarrow \mathbf{a}_t + \mathbf{K}_t [y_t - f_t]$
(Covariance) $\mathbf{C}_t \leftarrow [\mathbf{I} - \mathbf{K}_t \mathbf{F}_t] \mathbf{R}_t$
 2. filtering distribution at time point t : $\mathbf{m}_t, \mathbf{C}_t$
-

Repeating the procedure of Algorithm 8.1 from $t = 1$, we obtain the filtering distribution at every time point. The filtering distribution at time point 0 corresponds to the prior distribution. The procedure in Algorithm 8.1 is consistent with the description in Sect. 6.2.3 as follows:

- One-step-ahead predictive distribution: the filtering distribution one time before is transitioned in the time-forward direction based on the state equation.
- One-step-ahead predictive likelihood: the one-step-ahead predictive distribution is converted in the domain of observations based on the observation equation.
- Filtering distribution: the one-step-ahead predictive distribution is corrected based on the likelihood.

Herein, we supplement Kalman filtering.

Focus on a $p \times p$ matrix called the *Kalman gain* in Algorithm 8.1. As explained in Sect. 6.2.3, the filtering distribution is obtained by correcting the one-step-ahead predictive distribution. The Kalman gain is related to the degree of this correction. Specifically, if we define the *prediction error* $e_t = y_t - f_t$ as the difference between the observations and mean of the one-step-ahead predictive likelihood, the gain corresponds to the weight of the extent to which this prediction error is reflected in the correction. The ratio of W_t to V_t (also known as the signal-to-noise ratio) has a dominant influence on the Kalman gain. We consider a simple example wherein $p = 1$, $\mathbf{G}_t = [1]$, and $\mathbf{F}_t = [1]$. The dimension of the state $p = 1$ implies that all the vectors and matrices of interest become scalar. This model is referred to as the *local-level model*; we will explain the details in Sect. 9.2. If $V_t = 0$ holds in this model, for example, then the observations can be trusted fully; hence, we must completely reflect the prediction error containing the influence of observations in the correction. In this case, the value of the Kalman gain becomes 1 and $m_t = y_t$ holds; hence the filtering distribution is determined only by the observations. On the contrary, if $V_t = \infty$ holds, then the observations cannot be trusted at all; hence, we must completely ignore the prediction error containing the influence of observations in the correction. In this case, the value of the Kalman gain becomes 0 and $m_t = a_t$ holds; hence, the filtering distribution is determined only by the one-step-ahead prediction. If V_t is between 0 and ∞ , then the Kalman gain is to take a value between 1 and 0 compromising both the necessity based on the observation equation and possibility based on the state equation. Furthermore, we can confirm that the estimation of the mean is equivalent to the exponentially weighted moving average because $m_t = m_{t-1} + K_t[y_t - m_{t-1}] = K_t y_t + (1 - K_t)m_{t-1}$ holds in this model. The Kalman gain converges to a time-invariant constant in many models for stationary time series; such a Kalman filter is called a *stationary Kalman filter*.

Now, we implement the contents of Algorithm 8.1 with simple code. For data and model, we use the flow data of the Nile and abovementioned local-level model, respectively. This code is as follows.

Code 8.1

```

1 > # <<Kalman filtering (from scratch)>>
2 >
3 > # Set the flow data of the Nile to observations
4 > y <- Nile
5 > t_max <- length(y)
6 >
7 > # Function performing Kalman filtering for one time point
8 > Kalman_filtering <- function(m_t_minus_1, C_t_minus_1, t){
9 +   # One-step-ahead predictive distribution
10 +   a_t <- G_t %*% m_t_minus_1
11 +   R_t <- G_t %*% C_t_minus_1 %*% t(G_t) + W_t
12 +
13 +   # One-step-ahead predictive likelihood
14 +   f_t <- F_t %*% a_t
15 +   Q_t <- F_t %*% R_t %*% t(F_t) + V_t
16 +
17 +   # Kalman gain
18 +   K_t <- R_t %*% t(F_t) %*% solve(Q_t)
19 +
20 +   # State update
21 +   m_t <- a_t + K_t %*% (y[t] - f_t)
22 +   C_t <- (diag(nrow(R_t)) - K_t %*% F_t) %*% R_t
23 +
24 +   # Return the mean and variance of the filtering distribution (and also
25 +   # one-step-ahead predictive distribution)
26 +   return(list(m = m_t, C = C_t,
27 +               a = a_t, R = R_t))
28 + }
29 >
30 > # Set parameters of the linear Gaussian state space (all 1 x 1 matrices)
31 > G_t <- matrix(1, ncol = 1, nrow = 1); W_t <- matrix(exp(7.29), ncol = 1, nrow = 1)
32 > F_t <- matrix(1, ncol = 1, nrow = 1); V_t <- matrix(exp(9.62), ncol = 1, nrow = 1)
33 > m0 <- matrix(0, ncol = 1, nrow = 1); CO <- matrix(1e+7, ncol = 1, nrow = 1)
34 >
35 > # Calculate the mean and variance of the filtering distribution (and also
36 +   # one-step-ahead predictive distribution)
37 > m <- rep(NA_real_, t_max); C <- rep(NA_real_, t_max)
38 > a <- rep(NA_real_, t_max); R <- rep(NA_real_, t_max)
39 >
40 > # Time point: t = 1
41 > KF <- Kalman_filtering(m0, CO, t = 1)
42 > m[1] <- KF$m; C[1] <- KF$C
43 > a[1] <- KF$a; R[1] <- KF$R
44 >
45 > # Time point: t = 2 to t_max
46 > for (t in 2:t_max){
47 +   KF <- Kalman_filtering(m[t-1], C[t-1], t = t)
48 +   m[t] <- KF$m; C[t] <- KF$C
49 +   a[t] <- KF$a; R[t] <- KF$R
50 + }
51 >
52 > # Ignore the display of following codes

```

In the above code, we first set the flow data of the Nile to observations. We then define the function `Kalman_filtering()`, which performs Kalman filtering for one time point. Subsequently, the parameters of the linear Gaussian state-space model are specified. We set the values to be obtained with the library `dlm` in later Sect. 8.2,

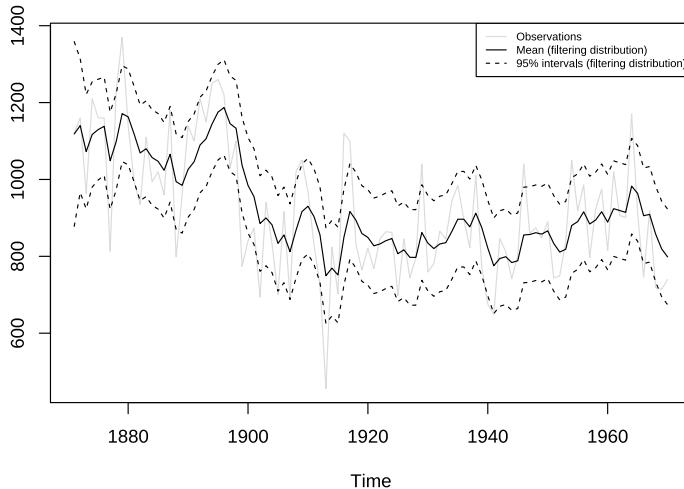


Fig. 8.1 Kalman filtering for a local-level model (flow data of the Nile)

where parameters are specified using the maximum likelihood method assuming that the variances \mathbf{W}_t of the state noise and V_t of the observation noise are time-invariant. For filtering, we use the function `Kalman_filtering()` for every time point to obtain the mean and variance of the filtering distribution. In addition, we save the results of the one-step-ahead predictive distribution through Kalman filtering in advance because they are to be reused later in Kalman smoothing. Figure 8.1 shows the plotting result of Kalman filtering. This figure displays the mean and 95% probability intervals of the estimated distribution.

This book regards a probability interval as a Bayesian *confidence interval*: when we use the $(1 - \alpha)\%$ probability intervals for some probability distribution, it corresponds to the remaining central portion of $(1 - \alpha)\%$ after removing each area of $\alpha/2\%$ from both ends of the distribution.

Furthermore, we derive the likelihood in the linear Gaussian state-space model. The log-likelihood for the entire time series can be obtained from the one-step-ahead predictive likelihood, as in Eq. (6.26). Recalling that the one-step-ahead predictive likelihood in the linear Gaussian state-space model is a normal distribution with mean f_t and variance Q_t yields

$$\begin{aligned}\ell(\boldsymbol{\theta}) &= \sum_{t=1}^T \log p(y_t \mid y_{1:t-1}; \boldsymbol{\theta}) \\ &:= -\frac{1}{2} \sum_{t=1}^T \log |Q_t| - \frac{1}{2} \sum_{t=1}^T (y_t - f_t)^2 / Q_t.\end{aligned}\quad (8.6)$$

The definition of the log-likelihood in the linear Gaussian State-space model varies in the literature. The principal difference lies in the constant term $-\frac{1}{2}T \log 2\pi$, which has been removed in the above definition [16]. For this reason, the value of the log-likelihood in this book differs from that calculated according to the exact expression of the normal distribution by the constant term; this book's value is greater than the latter. Regarding the likelihood, the relative value is more important than the absolute value; hence there is no problem, even with the definition in this book.

Finally, We supplement the prediction error e_t . The prediction error e_t is the same as the residuals described in Sect. 4.3 and is also referred to as *innovations*. If the model is appropriate, the innovations theoretically have an independent and identical normal distribution [16]. Thus, the innovations are used to diagnose whether the model is appropriate.

The above phrase “the samples have independent and identical distribution” means that “the samples obtained from the same distribution are mutually independent.” The term independent and identical distribution is abbreviated as i.i.d.

We supplement the brief history of the study of innovations: it was conceived by H. Wold, named by N. Wiener, and thoroughly examined by T. Kailath [9]. While this book does not describe the innovation form [9, 16] of the linear Gaussian state-space model, nevertheless, the innovations are useful for acquiring an advanced understanding of the estimation problem.

8.1.2 Kalman Prediction

We now explain Kalman prediction based on the description in Sect. 6.2.4.

The k -steps-ahead predictive distribution in the linear Gaussian state-space model is also the normal distribution; hence, we define it as follows:

k-steps-ahead predictive distribution $\mathcal{N}(\mathbf{a}_t(k), \mathbf{R}_t(k))$, for the mean vector $\mathbf{a}_t(k)$ and covariance matrix $\mathbf{R}_t(k)$. (8.7)

The procedure for obtaining the k -steps-ahead predictive distribution at time point $t + k$ from that at time point $t + (k - 1)$ is as follows; see Appendix E.2 for its detailed derivation.

Algorithm 8.2 Kalman prediction

0. the $k - 1$ -steps-ahead predictive distribution at time point $t + (k - 1)$: $\mathbf{a}_t(k - 1), \mathbf{R}_t(k - 1)$
 1. updating procedure at time point $t + k$
 - *the k -steps-ahead predictive distribution*

(Mean) $\mathbf{a}_t(k) \leftarrow \mathbf{G}_{t+k} \mathbf{a}_t(k - 1)$
(Covariance) $\mathbf{R}_t(k) \leftarrow \mathbf{G}_{t+k} \mathbf{R}_t(k - 1) \mathbf{G}_{t+k}^\top + \mathbf{W}_{t+k}$
 2. the k -steps-ahead predictive distribution at time point $t + k$: $\mathbf{a}_t(k), \mathbf{R}_t(k)$
-

Repeating the procedure of Algorithm 8.2 from $k = 1$ yields the predictive distribution at time point $t + k$. The 0-steps-ahead predictive distribution at time t corresponds to the filtering distribution at time t . The procedure in Algorithm 8.2 is consistent with the description in Sect. 6.2.4 as follows:

- the k -steps-ahead predictive distribution: the $k - 1$ -steps-ahead predictive distribution is transitioned based on the state equation.

Herein, we supplement Kalman prediction.

As mentioned in Sect. 6.2.4, missing observations in the state-space model are compensated for by the prediction. Thus, even if there are missing observations, we can proceed with the filtering procedure by assuming that Kalman gain = \mathbf{O} in Algorithm 8.1; we eventually regard the one-step-ahead predictive distribution as a filtering distribution. In addition, according to the above property, if we intentionally add multiple missing observations at the end of data, the filtering of such data results in the long-term prediction.

Now, we implement the contents of Algorithm 8.2 in simple code. Similarly to the Kalman filtering case, we use the flow data of the Nile and local-level model as observations and model, respectively. The code is as follows.

Code 8.2

```

1 > # <<Kalman prediction (from scratch)>>
2 >
3 > # Assuming Kalman filtering completed
4 >
5 > # Prediction period
6 > t <- t_max      # From the last time point
7 > nAhead <- 10    # Ten time points ahead
8 >
9 > # Function for one-step-ahead Kalman prediction (k = 1)
10 > Kalman_prediction <- function(a_t0, R_t0){
11 +   # One-step-ahead predictive distribution
12 +   a_t1 <- G_t_plus_1 %*% a_t0
13 +   R_t1 <- G_t_plus_1 %*% R_t0 %*% t(G_t_plus_1) + W_t_plus_1
14 +
15 +   # Return the mean and variance of the one-step-ahead predictive distribution
16 +   return(list(a = a_t1, R = R_t1))
17 + }
18 >
19 > # Set parameters of the linear Gaussian state space (time-invariant)
20 > G_t_plus_1 <- G_t; W_t_plus_1 <- W_t
21 >
22 > # Calculate the mean and variance for the k-steps-ahead predictive distribution
23 >
24 > # Allocate memory for state (mean and covariance)
25 > a_ <- rep(NA_real_, t_max + nAhead); R_ <- rep(NA_real_, t_max + nAhead)
26 >
27 > # k = 0 (zero-step-ahead predictive distribution corresponds to filtering
28 > # distribution)
28 > a_[t + 0] <- m[t]; R_[t + 0] <- C[t]
29 >
30 > # k = 1 to nAhead
31 > for (k in 1:nAhead){
32 +   KP <- Kalman_prediction(a_[t + k-1], R_[t + k-1])
33 +   a_[t + k] <- KP$a; R_[t + k] <- KP$R
34 + }
35 >
36 > # Ignore the display of following codes

```

This code assumes that Kalman filtering has been completed. In the above code, we first set the prediction period. We then define the function `Kalman_prediction()`, which performs one-step-ahead Kalman prediction. Subsequently, we set the parameters necessary for prediction, such as the state transition matrix and variance of the state noise. In this example, supposing a time-invariant model, we set the same values as for Kalman filtering. For prediction, we use the function `Kalman_prediction()` for the prediction period to obtain the mean and variance of the k -steps-ahead predictive distribution. Figure 8.2 shows the plotting results of Kalman prediction. This figure displays the mean and 95% probability intervals of the estimated distribution.

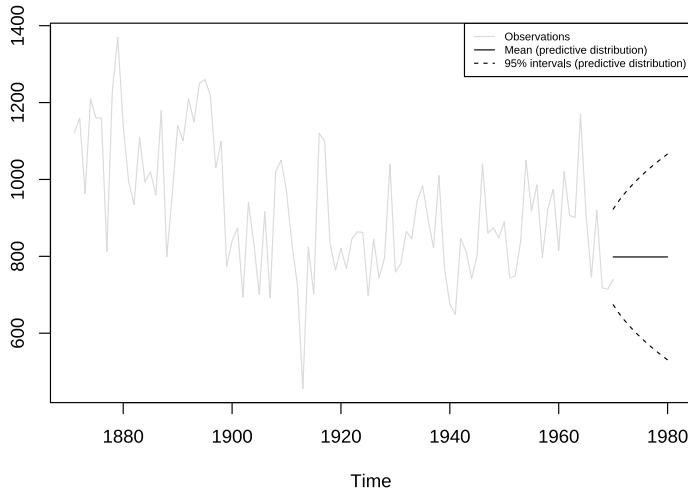


Fig. 8.2 Kalman prediction for the local-level model (flow data of the Nile)

8.1.3 *Kalman Smoothing*

We now explain Kalman smoothing based on the description in Sect. 6.2.5. As mentioned in Sect. 6.2.5, this book focuses on fixed-interval smoothing; hence, we assume that Kalman filtering through time T has been completed before smoothing.

Regarding the fixed-lag and -point smoothings in Kalman filter, see [1, 8–10, 18].

The smoothing distribution in the linear Gaussian state-space model is also the normal distribution; hence, we define it as follows:

$$\text{smoothing distribution } \mathcal{N}(s_t, S_t), \text{ for the mean vector } s_t \text{ and} \\ \text{covariance matrix } S_t. \quad (8.8)$$

The procedure for obtaining the smoothing distribution at time t from that at time $t + 1$ is as follows; see Appendix E.2 for its detailed derivation.

Algorithm 8.3 Kalman smoothing

-
0. smoothing distribution at time $t + 1$: s_{t+1} , S_{t+1}
 1. update procedure at time t
 - *Smoothing gain*
 $A_t \leftarrow C_t G_{t+1}^\top R_{t+1}^{-1}$
 - *State update*
 (Mean) $s_t \leftarrow m_t + A_t [s_{t+1} - a_{t+1}]$
 (Covariance) $S_t \leftarrow C_t + A_t [S_{t+1} - R_{t+1}] A_t^\top$
 2. smoothing distribution at time t : s_t , S_t
-

Repeating the procedure of Algorithm 8.3 from $t = T - 1$ in the reverse time direction, we obtain the smoothing distribution at every time point. Note that the smoothing distribution at time T corresponds to the filtering distribution at time T . The procedure in Algorithm 8.3 is consistent with the description in Sect. 6.2.5 as follows:

- Smoothing distribution: the filtering distribution is corrected based on the smoothing distribution at one time point ahead.

Herein, we supplement Kalman smoothing.

The algorithm described in Algorithm 8.3 is a popular *RTS algorithm* (RTS; Rauch–Tung–Striebel) [17]. The fixed-interval smoothing in the Kalman filter can be also achieved using other algorithms, such as the Bryson–Frazier (BF) algorithm [2, 3, 14], which is similar to the RTS algorithm and two-filter formula [4, 5, 12], which utilizes the form of an information filter. While they are all equivalent, they differ in the way they are calculated.

Now, we implement the contents of Algorithm 8.3 in simple code. Similar to the Kalman filtering and prediction cases, we use the flow data of the Nile and local-level model as observations and model, respectively. This code is as follows.

Code 8.3

```

1 > # <<Kalman smoothing (from scratch)>>
2 >
3 > # Assuming Kalman filtering completed
4 >
5 > # Function performing Kalman smoothing for one time point
6 > Kalman_smoothing <- function(s_t_plus_1, S_t_plus_1, t){
7 +   # Smoothing gain
8 +   A_t <- C[t] %*% t(G_t_plus_1) %*% solve(R[t+1])
9 +
10 +  # State update
11 +  s_t <- m[t] + A_t %*% (s_t_plus_1 - a[t+1])
12 +  S_t <- C[t] + A_t %*% (S_t_plus_1 - R[t+1]) %*% t(A_t)
13 +
14 +  # Return the mean and variance of the smoothing distribution
15 +  return(list(s = s_t, S = S_t))
16 +
17 >
18 > # Find the mean and variance of the smoothing distribution
19 >
20 > # Allocate memory for state (mean and covariance)
21 > s <- rep(NA_real_, t_max); S <- rep(NA_real_, t_max)
22 >
23 > # Time point: t = t_max
24 > s[t_max] <- m[t_max]; S[t_max] <- C[t_max]
25 >
26 > # Time point: t = t_max - 1 to 1
27 > for (t in (t_max-1):1){
28 +   KS <- Kalman_smoothing(s[t+1], S[t+1], t = t)
29 +   s[t] <- KS$s; S[t] <- KS$S
30 +
31 >
32 > # Ignore the display of following codes

```

This code assumes that Kalman filtering has been completed. In the above code, we first define the function `Kalman_smoothing()`, which performs Kalman smoothing for one time point. For smoothing, we then use the function `Kalman_smoothing()` for every time point to obtain the mean and variance of the smoothing distribution. Figure 8.3 shows the plotting result of Kalman smoothing. This figure displays the mean and 95% probability intervals of the estimated distribution.

8.2 Example: Local-level Model Case

We now illustrate time series analysis using a Kalman filter through a simple example. In the previous section, we use the scratched codes reflecting the algorithm description to perform Kalman filter. However, henceforth, we use the generic library **dlm** for ease of application. We proceed below with an explanation based on the flow of analysis explained in Chap. 4.

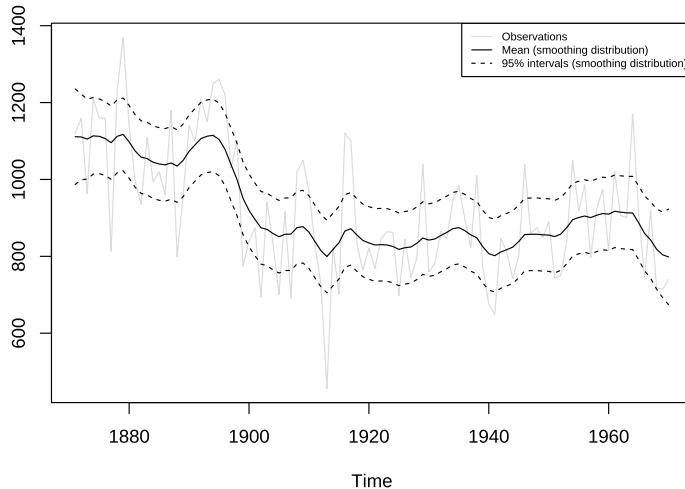


Fig. 8.3 Kalman smoothing for the local-level model (flow data of the Nile)

8.2.1 Confirmation of the Purpose and Data Collection

This example uses the R built-in dataset `Nile`. We set the purpose of the analysis as estimating the past, present, and future values accurately.

8.2.2 Preliminary Examination of Data

This step is identical with that in Chap. 4; hence, we ignore its description.

8.2.3 Model Definition

We define here the model for analysis. We adopt the local-level model consistent with the descriptions in Sect. 4.3, where we have examined the Nile data from an exploratory perspective; the next chapter explains the details of this model as one of the often used models for the linear Gaussian state-space model. The local-level model is defined as follows:

$$x_t = x_{t-1} + w_t, \quad w_t \sim \mathcal{N}(0, W), \quad (8.9)$$

$$y_t = x_t + v_t, \quad v_t \sim \mathcal{N}(0, V). \quad (8.10)$$

The above model corresponds to the linear Gaussian state-space model that has the settings $p = 1$, $\mathbf{G}_t = [1]$, and $\mathbf{F}_t = [1]$ for its general definition in Eqs. (5.8) and (5.9). The state dimension $p = 1$ results in all related vectors and matrices being scalar. Furthermore, for simplicity, we assume a time-invariant model, i.e., a model whose parameters do not depend on time: $W_t = W$ and $V_t = V$. The state equation (8.9) for the local-level model states that the current observations are roughly the same as the previous ones and that there is no particular time pattern. Thus, we can understand that the knowledge obtained through the preliminary examination is adequately reflected in the model except for the sudden decrease in 1899.

The local-level model is similar to the AR(1) model in Eqs. (7.4) and (7.5), which are described in Sect. 7.2. The difference lies in the AR(1) coefficient ϕ for the state x_{t-1} : the coefficient of the local-level model is one. Such a coefficient leads to the nonstationary time series for the state known as a *random walk*. For this reason, the local-level model is also called a *random walk plus noise model*.

The prior distribution $p(x_0) = \mathcal{N}(m_0, C_0)$ is a part of the model definition along with the state and observation equations; recall the joint distribution of the state-space model in the Eq. (5.6) described in Sect. 5.2.4. In this book, if there is no particular knowledge of a prior distribution, mean is set to an arbitrary finite value such as 0 or observations' average, whereas variance is set to an arbitrary finite value such as a sufficiently large value according to the data. Even such a pragmatic policy does not cause a problem [19]. While there are further approaches to pursue the setting of the prior distribution, this book ignores those details.

For example, the following approaches are available:

- A strict infinite variance
⇒ The derivation of the Kalman filter and likelihood for this case is described in [3, 14].
- A precise mean consistent with the observations
⇒ When the model contains a seasonal component, this book's approach might degrade the estimation accuracy for the first cycle of the seasonality, but a precise mean in the prior distribution can avoid such degradation. The basic idea of this approach is reuse of the result by preliminary execution of the smoothing. Fixed-point smoothing can also be used for this mechanism.

We then define the local-level model using the library **dlm**. This code is as follows.

Code 8.4

```

1 > # <<Define local-level model>>
2 >
3 > # Preprocessing
4 > library(dlm)
5 >
6 > # Setting of state space model
7 > mod <- dlmModPoly(order = 1)
8 >
9 > # Confirmation of model contents
10 > str(mod)
11 List of 10
12 $ m0 : num 0
13 $ C0 : num [1, 1] 1e+07
14 $ FF : num [1, 1] 1
15 $ V : num [1, 1] 1
16 $ GG : num [1, 1] 1
17 $ W : num [1, 1] 1
18 $ JFF: NULL
19 $ JV : NULL
20 $ JGG: NULL
21 $ JW : NULL
22 - attr(*, "class")= chr "dlm"

```

In the above code, we use the function `dlmModPoly()` prepared in **dLM**. The arguments of `dlmModPoly()` are `order`, `dW`, `dV`, `m0`, and `C0`. We omit the particular setting; hence, default values are applied to them except for `order`. We set `order = 1` because the local-level model is generally a polynomial model of order one; the details of the polynomial model are described in Sect. 9.3. The arguments `dW` and `dV` indicate the variances W and V of the state and observation noises, respectively. After the setting of the default value to them, they are updated to appropriate values through the specification of parameter values to be described later. The arguments `m0` and `C0` indicate the mean m_0 and variance C_0 of the prior distribution, respectively. While the default values 0 and 10^7 are set to them, the mean is finite and the variance is sufficiently larger than the data variance `var(Nile) = 28637.95`; hence, we need not update them. The prior's default values can be applied to all examples in this book without any problem for the same reason.

Confirming the contents of `mod`, i.e., the return values from `dlmModPoly()`, we see that it is a list with `dlm` class. The elements of the list are `m0`, `C0`, `FF`, `V`, `GG`, `W`, `JFF`, `JV`, `JGG`, and `JW`. In these elements, those with a name starting from `J` in the second half are used to specify the time-varying model. Since this example assumes the time-invariant local-level model, we need not change their settings from the default `NULL`. The first-half elements `m0`, `C0`, `FF`, `V`, `GG`, and `W` correspond to the parameters \mathbf{m}_0 , \mathbf{C}_0 , \mathbf{F}_t , \mathbf{V}_t , \mathbf{G}_t , and \mathbf{W}_t in Eqs. (8.1) and (8.2), respectively.

8.2.4 Specification of Parameter Values

As described in Sect. 6.4, when the model contains unknown parameters, we must specify their values in some manner for time series estimation. In this example, the

parameters W and V are unknown and lack any specific clues; hence we must specify these values. We attempt to use the maximum likelihood method based on Sect. 6.4.1 because the data `Nile` are of a sufficient quantity.

Now, we specify the parameters in the local-level model using the library `dlm`. This code is as follows.

Code 8.5

```

23 > # <<Specification of parameter values in local-level model>>
24 >
25 > # User-defined function to define and build a model
26 > build_dlm <- function(par) {
27 +   mod$W[1, 1] <- exp(par[1])
28 +   mod$V[1, 1] <- exp(par[2])
29 +
30 +   return(mod)
31 }
32 >
33 > # Maximum likelihood estimation of parameters, confirming the search results with
34 <- three different initial values
35 > lapply(list(c(0, 0), c(1, 10), c(20, 3)), function parms){
36 +   dlmMLE(y = Nile, parm = parms, build = build_dlm)
37 + }
38 [[1]]
39 [[1]]$par
40 [1] 7.291951 9.622437
41
42 [[1]]$value
43 [1] 549.6918
44
45 [[1]]$counts
46 function gradient
47      34      34
48
49 [[1]]$convergence
50 [1] 0
51
52 [[1]]$message
53 [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
54
55 [[2]]
56 [[2]]$par
57 [1] 7.291992 9.622452
58
59 [[2]]$value
60 [1] 549.6918
61
62 [[2]]$counts
63 function gradient
64      22      22
65
66 [[2]]$convergence
67 [1] 0
68
69 [[2]]$message
70 [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
71
72
73 [[3]]
74 [[3]]$par
75 [1] 7.291948 9.622437
76

```

```

77 [[3]]$value
78 [1] 549.6918
79
80 [[3]]$counts
81 function gradient
82      40      40
83
84 [[3]]$convergence
85 [1] 0
86
87 [[3]]$message
88 [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
89
90
91 >
92 > # Maximum likelihood estimation of parameters, including the Hessian matrix in
93 > # return value
94 > fit_dlm <- dlmMLE(y = Nile, parm = c(0, 0), build = build_dlm, hessian = TRUE)
95 >
96 > # Find the (asymptotic) standard error in the maximum likelihood estimation from
97 > # the Hessian matrix using the delta method
98 > exp(fit_dlm$par) * sqrt(diag(solve(fit_dlm$hessian)))
99 [1] 1280.170 3145.999
100 >
101 >
102 > # Set the maximum likelihood estimates of parameters in the model
103 > mod <- build_dlm(fit_dlm$par)
104 >
105 > # Confirmation of the results
106 > mod
107 $FF
108      [,1]
109 [1,]    1
110
111 $V
112      [,1]
113 [1,] 15099.8
114
115 $GG
116      [,1]
117 [1,]    1
118
119 $W
120      [,1]
121 [1,] 1468.432
122
123 $m0
124      [,1]
125 [1,] 1e+07

```

The above code continues from the former Code 8.4. In the above code, we use the function `d1mMLE()` prepared in `d1m`. The function `d1mMLE()` uses the R general-purpose optimization function `optim()` internally to minimize the negative log-likelihood; hence, its arguments and return values are consistent with those of `optim()`.

We now explain the contents of the above code from the beginning in order.

First, the user-defined function `build_d1m()` is prepared for defining and building a model. Specifically, this function updates the unknown parameters W and V in `mod` by the values in argument `par` and returns the entire `mod`. Since both W and V are variances and do not take negative values, an exponential transformation is applied to `par` to ensure that the function `optim()` can avoid an unnecessary search of the negative region. When the function `build_d1m()` references an object `mod` internally, R copies the `mod` internally and automatically, which is already defined in Code 8.4. Consequently, the operations in the function are applied to the copied `mod` with the result that the content of the original `mod` is never changed.

Next, we explain `d1mMLE()`. The arguments of `d1mMLE()` are `y`, `parm`, and `build`. We set the `Nile` to the observations `y`. The argument `parm` is the initial searching values for the parameters in the maximum likelihood estimation and we can set the arbitrary finite value to those. The argument `build` is a user-defined function for defining and building a model; hence, we set the previously explained `build_d1m()`. Regarding the algorithm performing the maximum likelihood estimation numerically, the L-BFGS method, a kind of quasi-Newton method, is used by default.

Confirming the return value from `d1mMLE()`, we see that it is a list. The element `par` contains the result of maximum likelihood estimation before exponential transformation. The element `convergence` is a flag indicating the convergence situation in the maximum likelihood estimation, and the value 0 confirms that convergence has been achieved. However, even if this value becomes 0, a problem might still remain; hence, we must examine the result. In particular, the maximum likelihood estimation is a nonlinear optimization in principle, so there is always a possibility of falling into a local optimum. Although various methods have been proposed to avoid such a local optimum, there is no universal solution currently; therefore, it is safe to change the initial value at least several times and compare the results. This example tries three different initial values, for which we obtain roughly the same result. Thus, we decide not to pursue the matter further. If the results were very different, we would need to pay attention. In such a case, the author does not recommend choosing one result with the maximum log-likelihood. If a stable result cannot be obtained, we can infer that the inadequacy might remain in the examination process thus far; hence, we must reconsider the problem by going back to the model definition or even some earlier stage.

Herein, we explain the *standard error* in the maximum likelihood estimation. The standard error in the maximum likelihood estimation means the average fluctuation of the estimates when we hypothesize other observations that have the same statistics. We can derive the standard error for each parameter in the maximum likelihood estimation from the *Hessian matrix* for the log-likelihood function.

The Hessian matrix contains the second-order partial derivatives of a function as its elements. In this example, the log-likelihood function $\ell()$ contains the parameters W and V ; hence, the Hessian matrix is as follows:

$$H = - \begin{bmatrix} \frac{\partial^2 \ell(W, V)}{\partial W \partial W} & \frac{\partial^2 \ell(W, V)}{\partial W \partial V} \\ \frac{\partial^2 \ell(W, V)}{\partial V \partial W} & \frac{\partial^2 \ell(W, V)}{\partial V \partial V} \end{bmatrix}. \quad (8.11)$$

According to the asymptotic properties of the maximum likelihood estimation [6, Chap. 13][13], if there is an inverse matrix for the Hessian matrix of the negative log-likelihood function, its diagonal elements have an asymptotic variance for each parameter.

When we set the additional argument `hessian = TRUE` to the function `d1m MLE()`, it is passed through the function `optim()`, and the Hessian matrix evaluated at the minimum value of the negative log-likelihood is to be included in the return values. However, this example cannot use them directly because of the exponentially transformed parameters. Thus, we use the *delta method* to obtain the standard error of the parameters after exponential transformation [16].

The basic idea of the delta method lies in Taylor expansions for the statistics of interest; these are simply truncated at order 1. Specifically, focusing on a single θ , we have

$$\begin{aligned} \text{Var}[\exp(\theta)] &\approx \text{Var} \left[\exp(\hat{\theta}) + \frac{\exp'(\hat{\theta})}{1!} (\theta - \hat{\theta}) \right] \\ &= \exp^2(\hat{\theta}) \text{Var}[\theta - \hat{\theta}] = \exp^2(\hat{\theta}) \text{Var}[\theta]. \end{aligned} \quad (8.12)$$

In this example, the diagonal elements of H^{-1} have $\text{Var}[\theta]$; hence, we obtain the standard error of a parameter after exponential transformation as

$$\begin{aligned} & \exp(\text{maximum likelihood estimates of a parameter}) \\ & \times \sqrt{\text{corresponding element of } \text{diag}(H^{-1})}. \end{aligned} \quad (8.13)$$

We have obtained the results that the standard errors of maximum likelihood estimation are 1280.170 and 3145.999 for W and V , respectively.

Finally, we set the maximum likelihood estimation results of the parameters to the model. We have the estimates 1468.432 and 15099.8 for the parameters W and V , respectively. Comparing these estimates with their standard errors reveals that the errors are comparable with the corresponding estimates, especially W . While there is no rule that the standard error must be within a specific range of values, the author thinks roughly that the estimation is stable if the error is less than one digit compared to the estimates. From such a viewpoint, this result appears to be of some concern. However, some literature [15] points out that the maximum likelihood estimation of the parameters in the linear Gaussian state-space model tends to yield a large standard error; therefore, we decide not to pursue these results any further. To examine this point more deeply, the Bayesian estimation assuming the parameters as random variables would be applied to enable appropriate consideration of the uncertainty of the parameter.

8.2.5 *Execution of Filtering, Prediction, and Smoothing*

Now, we perform filtering, prediction, and smoothing. We examine the results through the plot with the mean and 95% probability intervals for each estimated distribution.

8.2.5.1 **Filtering**

The operation of Kalman filtering is based on Algorithm 8.1. We perform the processing using the library **dlm**. This code is as follows.

Code 8.6

```

126 > # <<Kalman filtering>>
127 >
128 > # Filtering process
129 > dlmFiltered_obj <- dlmFilter(y = Nile, mod = mod)
130 >
131 > # Confirmation of the results
132 > str(dlmFiltered_obj, max.level = 1)
133 List of 9
134 $ y : Time-Series [1:100] from 1871 to 1970: 1120 1160 963 1210 1160 ...
135   ↪ 1370 1140 ...
136 $ mod:List of 10
137   ..- attr(*, "class")= chr "dlm"
138 $ m : Time-Series [1:101] from 1870 to 1970: 0 1118 1140 1072 1117 ...
139 $ U.C:List of 101
140 $ D.C: num [1:101, 1] 3162.3 122.8 88.9 76 70 ...
141 $ a : Time-Series [1:100] from 1871 to 1970: 0 1118 1140 1072 1117 ...
142 $ U.R:List of 100
143 $ D.R: num [1:100, 1] 3162.5 128.6 96.8 85.1 79.8 ...
144 $ f : Time-Series [1:100] from 1871 to 1970: 0 1118 1140 1072 1117 ...
145 - attr(*, "class")= chr "dlmFiltered"
146 >
147 > # Find the mean and standard deviation of the filtering distribution
148 > m <- dropFirst(dlmFiltered_obj$m)
149 > m_sdev <- sqrt(
150   +           dropFirst(as.numeric(
151   +             dlmSvd2var(dlmFiltered_obj$U.C, dlmFiltered_obj$D.C)
152   +           )))
153 >
154 > # Find 2.5% and 97.5% values for 95% intervals of the filtering distribution
155 > m_quant <- list(m + qnorm(0.025, sd = m_sdev), m + qnorm(0.975, sd = m_sdev))
156 >
157 > # Plot results
158 > ts.plot(cbind(Nile, m, do.call("cbind", m_quant)),
159   +         col = c("lightgray", "black", "black", "black"),
160   +         lty = c("solid", "solid", "dashed", "dashed"))
161 >
162 > # Legend
163 > legend(legend = c("Observations", "Mean (filtering distribution)", "95% intervals
164   + (filtering distribution)"),
165   +         lty = c("solid", "solid", "dashed"),
166   +         col = c("lightgray", "black", "black"),
167   +         x = "topright", text.width = 32, cex = 0.6)

```

The above code continues from the former Code 8.5. In the above code, we use the function `dlmFilter()` prepared in **dlm**. The arguments of `dlmFilter()` are `y` and `mod`. We set the `Nile` to the observations `y`. Since the argument `mod` is a list representing a model, we set the previous `mod`, whose parameter values are already specified. Confirming the contents of the `dlmFiltered_obj`, that is, the return value from `dlmFilter()` we see that it is a list with the `dlmFiltered` class. The elements of the list are `y`, `mod`, `m`, `U.C`, `D.C`, `a`, `U.R`, `D.R`, and `f`. The elements `y`, `mod`, `m`, `U.C`, `D.C`, `a`, `U.R`, `D.R`, and `f` correspond to the input observations, a list for model, mean of filtering distribution, two singular value decompositions for the covariance matrix of filtering distribution, mean of one-step-ahead predictive distribution, two singular value decompositions for the covariance matrix of one-step-ahead predictive distribution, and mean of one-step-ahead predictive likelihood,

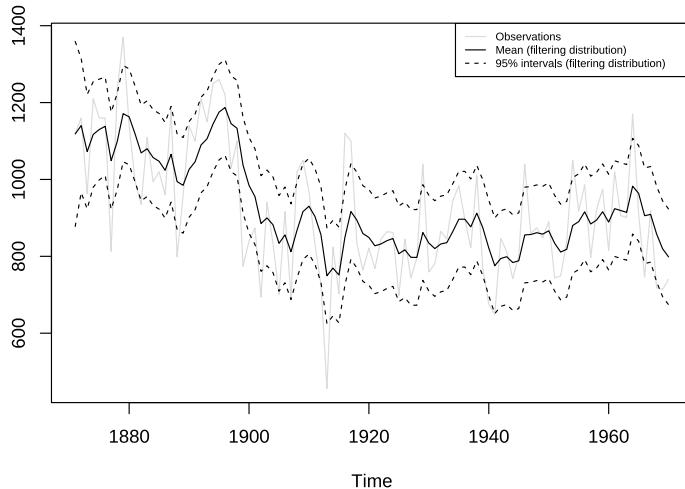


Fig. 8.4 Result of Kalman filtering

respectively. Since an object corresponding to the prior distribution is added at the beginning of m , $U.C$, and $D.C$, their length is one more than that of the observations: this example takes the value 101 because `length(Nile) = 100`.

Subsequently, we derive the mean and standard deviation of the filtering distribution from `dlmFiltered_obj`. The mean of the filtering distribution is m . We drop a first element corresponding to the prior distribution in advance using the utility function `dropFirst()` of `dlm` for ease of plotting. Since the covariance of the filtering distribution is decomposed into the elements $U.C$ and $D.C$, we retrieve the original covariance matrix using the utility function `dlmSvd2var()` of `dlm`. We then cast the retrieved list of the 1×1 matrices into a vector by `as.numeric`; recall the state dimension $p = 1$ in this example. Then, we drop a first element corresponding to the prior distribution just as in the case of the mean using the utility function `dropFirst()` of `dlm`.

Finally, we derive the 2.5% and 97.5% values of the filtering distribution to confirm the 95% intervals. In this example, based on the fact that the filtering distribution is a normal distribution, we use the function `qnorm()` to obtain the quantiles of the distribution.

Figure 8.4 shows the results for the above description.

The local-level model does not consider the sudden decrease in 1899 explicitly; hence, we cannot detect the corresponding change point accurately as a natural result.

8.2.5.2 Prediction

The operation of Kalman prediction is based on Algorithm 8.2. We perform the processing using the library `dlm`. This code is as follows.

Code 8.7

```

167 > # <<Kalman prediction>>
168 >
169 > # Prediction processing
170 > dlmForecasted_obj <- dlmForecast(mod = dlmFiltered_obj, nAhead = 10)
171 >
172 > # Confirmation of the results
173 > str(dlmForecasted_obj, max.level = 1)
174 List of 4
175 $ a: Time-Series [1:10, 1] from 1971 to 1980: 798 798 798 798 798 ...
176 ..- attr(*, "dimnames")=List of 2
177   $ R:List of 10
178   $ f: Time-Series [1:10, 1] from 1971 to 1980: 798 798 798 798 798 ...
179   ..- attr(*, "dimnames")=List of 2
180   $ Q:List of 10
181 >
182 > # Find the mean and standard deviation of the predictive distribution
183 > a <- ts(data = dlmForecasted_obj$a, start = c(1971, 1))
184 > a_sdev <- sqrt(
185 +     as.numeric(
186 +         dlmForecasted_obj$R
187 +     )
188 + )
189 >
190 > # Find 2.5% and 97.5% values for 95% intervals of the predictive distribution
191 > a_quant <- list(a + qnorm(0.025, sd = a_sdev), a + qnorm(0.975, sd = a_sdev))
192 >
193 > # Plot results
194 > ts.plot(cbind(Nile, a, do.call("cbind", a_quant)),
195 +           col = c("lightgray", "black", "black", "black"),
196 +           lty = c("solid", "solid", "dashed", "dashed"))
197 >
198 > # Legend
199 > legend(legend = c("Observations", "Mean (predictive distribution)", "95% intervals
200   ↪ (predictive distribution)"),
201 +           lty = c("solid", "solid", "dashed"),
202 +           col = c("lightgray", "black", "black"),
203 +           x = "topright", cex = 0.6)

```

The above code continues from the former Code 8.6. In the above code, we use the function `dlmForecast()` prepared in **dlm**. The arguments of `dlmForecast()` are `mod`, `nAhead`, and `sampleNew`. The argument `mod` takes a list representing a model or an object with the `dlmFiltered` class. This example predicts the future from the end point of filtering; hence, this argument is set to `dlmFiltered_obj` obtained from the previous Kalman filtering. The argument `nAhead` is the number of steps ahead for a prediction, and this example sets this argument to ten. Regarding the argument `sampleNew`, if we set this argument to an integer, the return value contains the samples for the number of trials. This example does not use such samples; hence, we let the setting be the default, i.e., `FALSE`.

The Code 7.1 generated the analysis data by setting `sampleNew = 1`.

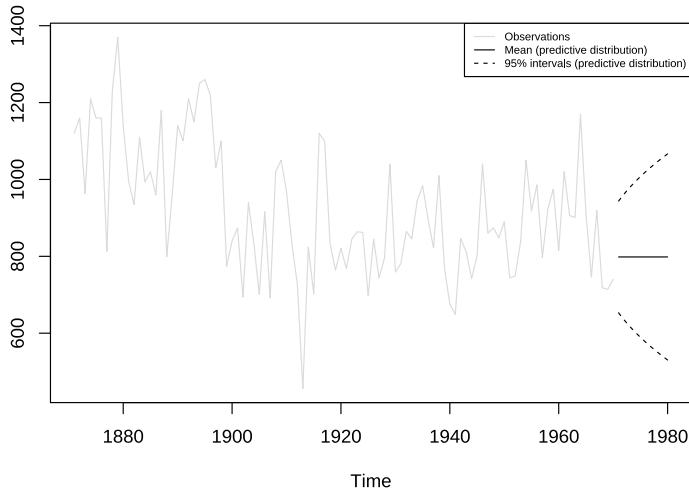


Fig. 8.5 Result of Kalman prediction

Confirming the contents of the `dlmForecasted_obj`, i.e., the return value from `dlmForecast()`, we see that it is a list. The elements of the list are α , R , f , and Q , which correspond to the mean vector and covariance matrix of the k -steps-ahead predictive distribution and the mean vector and covariance matrix of the k -steps-ahead prediction likelihood, respectively.

Subsequently, we derive the mean and standard deviation of the predictive distribution from `dlmForecasted_obj`. The mean of the predictive distribution is α . We cast it into the R `ts` class object in advance for ease of plotting. The variance of the predictive distribution is R . We also cast the list of the 1×1 matrices into a vector by `as.numeric`; recall the state dimension $p = 1$ in this example.

Finally, we derive the 2.5% and 97.5% values of the predictive distribution to confirm the 95% intervals. In this example, based on the fact that the predictive distribution is also a normal distribution, we use the function `qnorm()` to obtain the quantile of the distribution.

Figure 8.5 shows the results for the above description.

Comparing the Fig. 8.5 with Fig. 8.4 of the filtering result, we can see that the probability intervals regarding prediction continue expanding as time passes, and the uncertainty increases.

8.2.5.3 Smoothing

The operation of Kalman smoothing is based on Algorithm 8.3. We perform the processing using the library `dlm`. This code is as follows.

Code 8.8

```

203 > # <<Kalman smoothing>>
204 >
205 > # Smoothing processing
206 > dlmSmoothed_obj <- dlmSmooth(y = Nile, mod = mod)
207 >
208 > # Confirmation of the results
209 > str(dlmSmoothed_obj, max.level = 1)
210 List of 3
211 $ s : Time-Series [1:101] from 1870 to 1970: 1111 1111 1111 1105 1113 ...
212 $ U.S:List of 101
213 $ D.S: num [1:101, 1] 74.1 63.5 56.9 53.1 50.9 ...
214 >
215 > # Find the mean and standard deviation of the smoothing distribution
216 > s <- dropFirst(dlmSmoothed_obj$s)
217 > s_sdev <- sqrt(
218 +     dropFirst(as.numeric(
219 +         dlmSvd2var(dlmSmoothed_obj$U.S, dlmSmoothed_obj$D.S)
220 +     )))
221 + )
222 >
223 > # Find 2.5% and 97.5% values for 95% intervals of the smoothing distribution
224 > s_quant <- list(s + qnorm(0.025, sd = s_sdev), s + qnorm(0.975, sd = s_sdev))
225 >
226 > # Plot results
227 > ts.plot(cbind(Nile, s, do.call("cbind", s_quant)),
228 +           col = c("lightgray", "black", "black", "black"),
229 +           lty = c("solid", "solid", "dashed", "dashed"))
230 >
231 > # Legend
232 > legend(legend = c("Observations", "Mean (smoothing distribution)", "95% intervals
233 <- (smoothing distribution)"),
234 +           lty = c("solid", "solid", "dashed"),
235 +           col = c("lightgray", "black", "black"),
236 +           x = "topright", cex = 0.6)

```

The above code continues from the former Code 8.7. In the above code, we use the function `dlmSmooth()` prepared in **dlm**. The arguments of `dlmSmooth()` are `y` and `mod`. We set the `Nile` to the observations `y`. The argument `mod` is a list representing a model; hence, we set the previous `mod` already specified. When the Kalman filtering has been completed in advance, we may set the object with the `dlmFiltered` class into the argument `y` to obtain the same result. Confirming the contents of the `dlmSmoothed_obj`, that is, the return value from `dlmSmooth()`, we see that it is a list. The elements of the list are `s`, `U.S`, and `D.S`. The elements `s`, `U.S`, and `D.S` are the mean of the smoothing distribution and two singular value decompositions for the covariance matrix of the smoothing distribution, respectively. Since an object corresponding to the prior distribution is added at the beginning, their length is one more than that of the observations: this example takes the value 101 because `length(Nile) = 100`.

Subsequently, we derive the mean and standard deviation of the smoothing distribution from `dlmSmoothed_obj`. The mean of the smoothing distribution is `s`. We drop a first element corresponding to the prior distribution in advance using the utility function `dropFirst()` of **dlm** for ease of plotting. The covariance of the smoothing distribution is decomposed into the elements `U.S` and `D.S`; hence, we retrieve

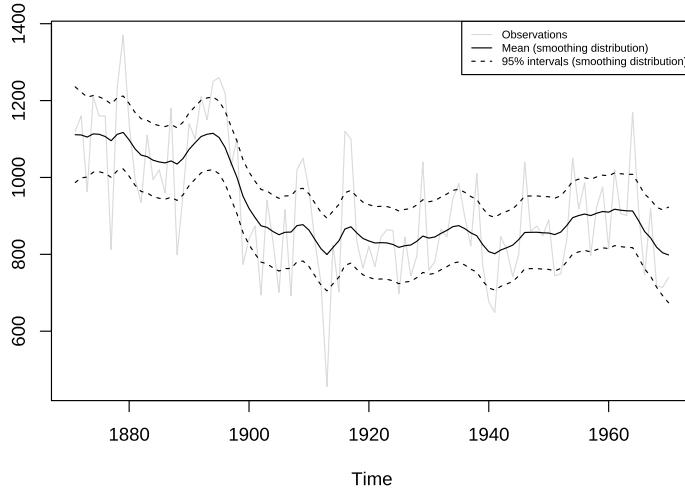


Fig. 8.6 Result of Kalman smoothing

the original covariance matrix using the utility function `dlmSvd2var()` of **dlm**. Next, we cast the retrieved list of the 1×1 matrices into a vector by `as.numeric`; recall the state dimension $p = 1$ in this example. We then drop a first element corresponding to the prior distribution just as in the case of the mean using the utility function `dropFirst()` of **dlm**.

Finally, we derive the 2.5% and 97.5% values of the smoothing distribution to confirm the 95% intervals. In this example, based on the fact that the smoothing distribution is also a normal distribution, we use the function `qnorm()` to obtain the quantiles of the distribution.

Figure 8.6 shows the results for the above description.

Comparing the Fig. 8.6 with Fig. 8.4 of the filtering result, we see that the mean value and the probability intervals become smoother and more narrow, respectively. While the filtering is based on the past and current information, the smoothing can further consider the relative future information; it can generally improve the estimation accuracy. However, even smoothing cannot capture the sudden decrease in 1899. To capture such a situation adequately, we must incorporate particular information into the model: Sect. 9.6.2 will introduce a method using a regression model.

8.2.6 Diagnostic Checking for the Results

Finally, we explain diagnostic checking for the results. We use the likelihood and prediction error as diagnostic tools.

8.2.6.1 Likelihood

As mentioned in Sect. 6.3, this book treats the likelihood as a model selection criterion. Thus, when changing the model and premise, we compare their likelihoods and explore a condition wherein the likelihood is highest in principle. Since, for the linear Gaussian state-space model, Eq. (8.6) provides the likelihood of the entire time series, we calculate this value.

We now derive the actual likelihood value using the library **dlm**. This code is as follows.

Code 8.9

```

236 > # <<Likelihood in linear Gaussian state space model>>
237 >
238 > # Calculation of "negative" log-likelihood
239 > dlmLL(y = Nile, mod = mod)
240 [1] 549.6918

```

The above code continues from the former Code 8.8. In the above code, we use the function `dlmLL()` prepared in **dLM**.

The function `dlmMLE()` described in Sect. 8.2.4 internally uses `dlmLL()`. Thus, the return value from `dlmMLE()` actually contains the (negative) log-likelihood value as an element `value`; see Code 8.5. However, for applied studies, we sometimes change the parameters intentionally to check the log-likelihood value; hence, this book also explains `dlmLL()`.

The arguments of `dlmLL()` are `y` and `mod`. We set `Nile` to the observations `y`. The argument `mod` is a list representing a model; hence, we set the previous `mod` already specified.

Regarding the log-likelihood value, we obtain the result 549.6918. If changing the model and premise further, we compare the result values and explore the best condition. The function `dlmLL()` returns a “*negative*” log-likelihood; the sign of this value is reversed from Eq. (8.6).

8.2.6.2 Innovations (Prediction Error/Residuals)

As mentioned in Sect. 8.1.1, the difference between observations and the mean of the one-step-ahead predictive likelihood, i.e., $e_t = y_t - f_t$, is called innovations, pre-

diction error, or residuals. If the model is appropriate, the innovations theoretically follow an independent and identical normal distribution. Note that $e_t/\sqrt{Q_t}$ becomes *normalized innovations*, because the variance of the innovations is the same as Q_t (variance of the one-step-ahead predictive likelihood) from its definition. Therefore, for the appropriate model, the normalized innovations theoretically follow an independent and identical normal standard distribution. In accordance with this property, this book diagnoses the model through the correlation and normality for the standardized innovations.

In the model diagnosis, the autocorrelation coefficient is a particularly important indicator; if the model cannot explain the relation among the data adequately, the standardized innovations contain substantial residual autocorrelation. Encountering such a case, we must repeat the model modification until less residual correlation is achieved in principle. Incidentally, when there might be an outlier or structural change, the standardized innovations reasonably take an unusually large value at such a point. If we incorporate the corresponding information in the model, we might be able to improve estimation accuracy. We must judge whether an outlier or structural change is involved in the model, not automatically but carefully.

While this book does not pursue other indices similar to innovations, it is also possible to define values that could be called “smoothing error” corresponding to the smoothed state and observation noises as follows:

$$\begin{aligned} E[\mathbf{w}_t \mid y_{1:T}] &= \mathbf{s}_t - \mathbf{G}_t \mathbf{s}_{t-1}, \\ E[\mathbf{v}_t \mid y_{1:T}] &= \mathbf{y}_t - \mathbf{F}_t \mathbf{s}_t. \end{aligned}$$

The normalized values of the above for each covariance are proposed as auxiliary residuals [3, 7, 14]. Unlike the standardized innovations, the auxiliary residuals have a correlation among them and are not an index based on predictability; hence, it appears to be inappropriate to use them directly for model diagnosis. The index would be rather useful for precise checking of an outlier or structural changes.

We now diagnose the model with innovations using the library **dlm**. This code is as follows.

Code 8.10

```

241 > # <<Model diagnosis using innovations>>
242 >
243 > # Adjust display area
244 > oldpar <- par(no.readonly = TRUE)
245 > par(oma = c(0, 0, 0, 0)); par(mar = c(4, 4, 3, 1))
246 >
247 > # Confirmation of autocorrelation
248 > tsdiag(object = dlmFiltered_obj)
249 > par(oldpar)                                # Restore parameters regarding display
250 >
251 > # Confirm normality
252 > # Get standardized innovations
253 > e <- residuals(object = dlmFiltered_obj, sd = FALSE)
254 >
255 > # Confirmation of the results
256 > e
257 Time Series:
258 Start = 1871
259 End = 1970
260 Frequency = 1
261 [1] 0.353882059 0.234347637 -1.132356160 0.920990056 0.293678981 0.208635337
262 [7] -2.253572772 1.259032474 1.892476740 -0.217341987 -1.168902405 -1.274143563
263 [13] 0.285525795 -0.598907228 -0.257811351 -0.607000658 1.087882179 -1.857139370
264 [19] -0.253479119 1.082258192 0.514608573 1.143623092 0.420243462 1.004779540
265 [25] 0.806186369 0.312247137 -1.094920935 -0.314871097 -2.502167297 -1.374265341
266 [31] -0.770458633 -1.818880663 0.380717419 -0.466439568 -1.261597906 0.573222303
267 [37] -1.140514421 1.449290868 1.271364973 0.367565138 -0.692068900 -1.238867224
268 [43] -2.789292186 0.519416299 -0.469282598 2.568373713 1.743294475 -0.589405024
269 [49] -0.905802083 -0.266834820 -0.564863230 0.122437752 0.222128119 0.148887241
270 [55] -0.1035113562 0.266628842 -0.508263319 -0.010258329 1.692519755 -0.717199228
271 [61] -0.372431518 0.312263613 0.089544673 0.755407088 0.832415412 0.004006761
272 [67] -0.519616041 0.928982856 -0.984249216 -1.383364390 -1.202138127 0.491394148
273 [73] 0.123305797 -0.397331863 0.119827082 1.753036709 0.030865019 0.120167580
274 [79] -0.093067758 0.224410162 -0.852741827 -0.590230865 0.187451524 1.614486965
275 [85] 0.263740703 0.667105819 -0.827838843 0.271075485 0.561004302 -0.703558407
276 [91] 0.912596736 -0.125338671 -0.126711287 1.781342993 -0.491841367 -1.517108371
277 [97] 0.093301043 -1.332051596 -1.004275772 -0.554991814
278 >
279 > # Display Q-Q plot
280 > qqnorm(e)
281 > qqline(e)      # Guideline through 25% and 75% points does not always have a
  ↪ 45-degree inclination

```

The above code continues from the former Code 8.9. We check the autocorrelation using the method of **dlm** prepared for the generic function **tsdiag()**. The argument of **tsdiag()** is **object**. We set the previously obtained **dlmFiltered_obj** with the **dlmFiltered** class into the argument **object**.

Figure 8.7 shows three plotting results by **tsdiag()**: the normalized innovations, the autocorrelation coefficient, and the p-values of the Ljung–Box test in order from the top.

We first confirm whether the standardized innovations have an extreme bias and unusually large values. We see that such features do not remain in this result. In addition, the result does not have a high autocorrelation coefficient. For the p-value of the Ljung–Box test, a dotted line shows a guide of the significance level 0.05.

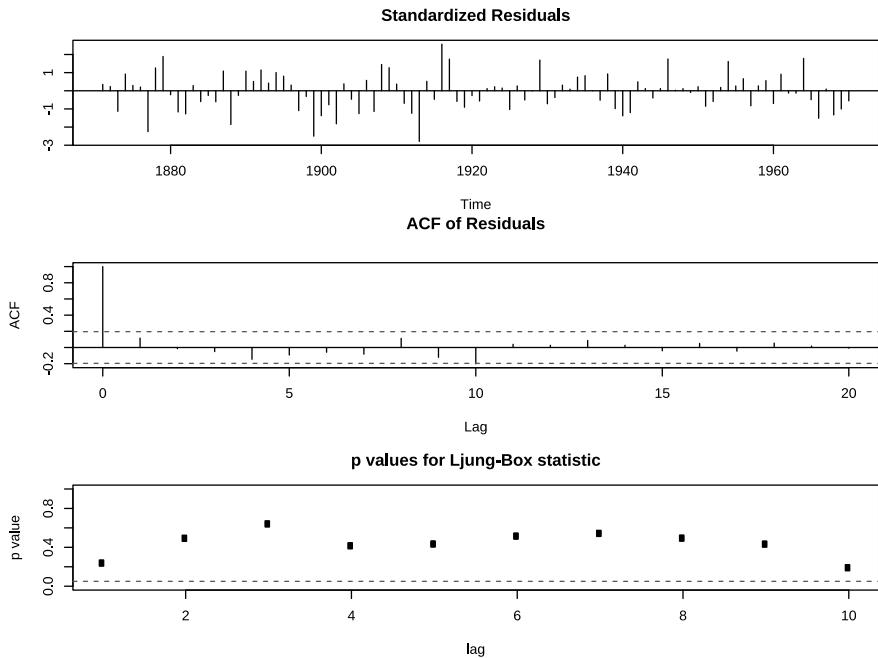


Fig. 8.7 Result of `tsdiag()`

Since the null hypothesis in this test is “data sequence is independent,” we cannot reject this hypothesis from the result.

We then confirm normality using the method of `dlm` prepared for the generic function `residuals()`. The arguments of `residuals()` are `object` and `sd`. We set the previously obtained `dlmFiltered_obj` with the `dlmFiltered` class to the argument `object`. While the argument `sd` indicates whether to include the variance of innovations, that is, the variance of the one-step-ahead predictive likelihood, in the return value, this example does not use the variance; hence, we set the argument to `FALSE`. Confirming the contents of the return value `e` from `residuals()`, we see that it is a time series of standardized innovations. Subsequently, we use the functions `qqnorm()` and `qqline()` to draw a Q-Q plot of the standardized innovations. Figure 8.8 shows the Q-Q plot.

The Q-Q plot takes the quantiles of the theoretical normal distribution on the horizontal axis and those of the data on the vertical axis. Thus, as the resulting plot approaches the diagonally upper right guideline, the data follow a more normal distribution. As Fig. 8.8 shows that the plotting result fits the guideline well, especially within $\pm 1\sigma$ range, we see that the normalized innovations almost have the normal distribution. While for normality, we can also use other test methods, such as the Shapiro–Wilk test, which is available in R as `shapiro.test()`, this book omits the details for simplicity.

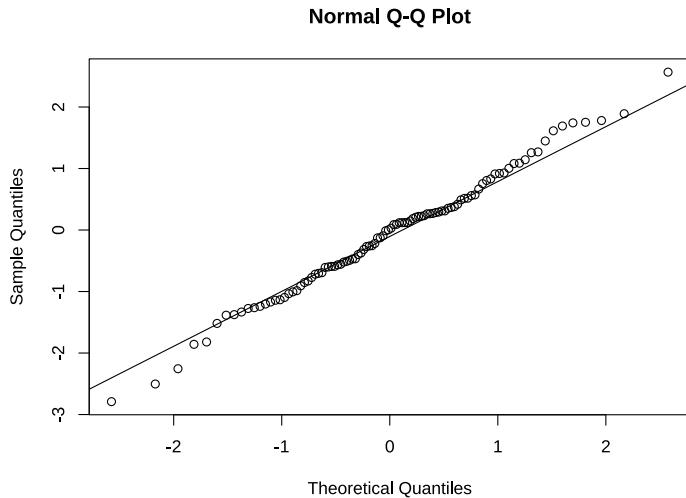


Fig. 8.8 Q-Q plot

References

1. Anderson, B.D.O., Moore, J.B.: Optimal Filtering. In: Kailath, T. (ed.) Information and System Sciences. Prentice-Hall (1979)
2. Bryson, A.E., Frazier, M.: Smoothing for linear and nonlinear dynamic systems. In: Proceedings of the optimum system synthesis conference, pp. 353–364 (1963)
3. Durbin, J., Koopman, S.J.: Time Series Analysis by State Space Methods, 2nd edn. Oxford University Press, Oxford (2012)
4. Fraser, D.C.: A new technique for the optimal smoothing of data. Sc.D. thesis, Massachusetts Institute of Technology, Cambridge, Mass. (1967)
5. Fraser, D.C., Potter, J.E.: The optimum linear smoother as a combination of two optimum linear filters. IEEE Trans. Automat. Control **14**(4), 387–390 (1969)
6. Hamilton, J.D.: Time Series Analysis. Princeton University Press (1994)
7. Harvey, A.C., Koopman, S.J.: Diagnostic checking of unobserved-components time series models. J. Bus. Econ. Statist. **10**(4), 377–389 (1992)
8. Higuchi, T., Ueno, G., Nakano, S., Nakamura, K., Yoshida, R.: Introduction to Data Assimilation—Next Generation Simulation Technology—. Asakura Publishing Co., Ltd (2011). [in Japanese]
9. Kailath, T., Sayed, A.H., Hassibi, B.: Linear Estimation. Prentice Hall (2000)
10. Katayama, T.: (New Edition) Applied Kalman filter. Asakura Publishing Co., Ltd (2000). [in Japanese]
11. Kitagawa, G.: Introduction to Time Series Modeling. CRC Press (2009)
12. Mayne, D.Q.: A solution of the smoothing problem for linear dynamic systems. Automatica **4**(2), 73–92 (1966)
13. Noda, K., Miyaoka, E.: Fundamentals of Mathematical Statistics. Kyoritsu Shuppan Co., Ltd (1992). [in Japanese]
14. Nomura, S.: Kalman Filter—Time Series Prediction and State Space Model using R—. Kyoritsu Shuppan Co., Ltd (2016). [in Japanese]
15. Petris, G.: State Space Models in R (user! 2011 tutorial). <https://user2011.r-project.org/tutorials/Petris.html> (2011)

16. Petris, G., Petrone, S., Campagnoli, P.: *Dynamic Linear Model with R*. Springer (2009)
17. Rauch, H.E., Tung, F., Striebel, C.T.: Maximum likelihood estimates of linear dynamic systems. *AIAA J.* **3**(8), 1445–1450 (1965)
18. Särkkä, S.: *Bayesian Filtering and Smoothing*, Institute of Mathematical Statistics Textbooks. Cambridge University Press, Cambridge (2013)
19. Tanizaki, H.: Application of State Space Model to Economics—Estimation of US-Japan Macro Econometric Model by Time-Varying Parameter Model—. Nippon Hyoron Sha Co., Ltd (1993). [in Japanese]

Chapter 9

Introduction and Analysis Examples of a Well-Known Component Model in the Linear Gaussian State-Space Model



This chapter explains typical models used in the state-space model individually with examples. As described in Chap. 5, the state-space model assumes the Markov property wherein the state has a relation with only itself at a previous time point. This property appears to be a very strong constraint at first glance. However, if we reconsider the states from viewpoints, such as multiple time points or kinds, we realize that a wide range of models can be expressed [5, Appendix A]. We discuss some of these models in this chapter. This chapter assumes that if there are unknown parameters, then the parameters are specified through maximum likelihood estimation without being regarded as random variables.

9.1 Combination of Individual Models

Time series data are typically comprise several components with differing properties. We have already examined the concept of *component decomposition* in Sect. 4.3. In the state-space model, it is easy to combine the individual basic model as parts. While the contents of an individual model are explained in the subsequent section, this section describes how to combine such a particular model in advance, i.e., the additive combination in the linear Gaussian state-space model.

We distinguish I types of individual model in the linear Gaussian state-space model by assigning the superscript (i) to the variables as follows:

$$\mathbf{x}_t^{(i)} = \mathbf{G}_t^{(i)} \mathbf{x}_{t-1}^{(i)} + \mathbf{w}_t^{(i)}, \quad \mathbf{w}_t^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{W}^{(i)}), \quad (9.1)$$

$$y_t^{(i)} = \mathbf{F}_t^{(i)} \mathbf{x}_t^{(i)} + v_t^{(i)}, \quad v_t^{(i)} \sim \mathcal{N}(0, V^{(i)}), \quad (9.2)$$

where the prior distribution is defined by $\mathbf{x}_0^{(i)} \sim \mathcal{N}(\mathbf{m}_0^{(i)}, \mathbf{C}_0^{(i)})$. Note that *the dimension of the state vector differs for each model* such as $p_1, \dots, p_i, \dots, p_I$.

Now, consider the above individual models as independent components constituting observations. In such a case, we can briefly express the final model in a single state equation (5.8) and observation equation (5.9) by setting the following:

$$\mathbf{x}_t = \begin{bmatrix} \mathbf{x}_t^{(1)} \\ \vdots \\ \mathbf{x}_t^{(I)} \end{bmatrix}, \mathbf{G}_t = \begin{bmatrix} \mathbf{G}_t^{(1)} & & \\ & \ddots & \\ & & \mathbf{G}_t^{(I)} \end{bmatrix}, \mathbf{W}_t = \begin{bmatrix} \mathbf{W}_t^{(1)} & & \\ & \ddots & \\ & & \mathbf{W}_t^{(I)} \end{bmatrix}, \quad (9.3)$$

$$y_t = \sum_{i=1}^I y_t^{(i)}, \quad \mathbf{F}_t = \left[\mathbf{F}_t^{(1)} \cdots \mathbf{F}_t^{(I)} \right], \quad V_t = \sum_{i=1}^I V_t^{(i)}. \quad (9.4)$$

The above settings are important, and hence are emphasized with framing; we refer to them later repeatedly. In addition, the prior distribution is defined as

$$\mathbf{x}_0 = \begin{bmatrix} \mathbf{x}_0^{(1)} \\ \vdots \\ \mathbf{x}_0^{(I)} \end{bmatrix}, \quad \mathbf{m}_0 = \begin{bmatrix} \mathbf{m}_0^{(1)} \\ \vdots \\ \mathbf{m}_0^{(I)} \end{bmatrix}, \quad \mathbf{C}_0 = \begin{bmatrix} \mathbf{C}_0^{(1)} & & \\ & \ddots & \\ & & \mathbf{C}_0^{(I)} \end{bmatrix}.$$

The library **dLM** prepares a method for generic function `+` to achieve the above combination; while R can treat `+` as an operator, it is actually a function [15]. For example, we can realize a combination using the expression “an object with `dLM` class `+` another object with `dLM` class.”

Incidentally, we have already mentioned that the basic structural model or standard seasonal adjustment model expresses “level + trend + season” in the state-space model; they are achieved by combining three models to be described in subsequent sections, namely Sect. 9.2 or Sect. 9.3 and Sect. 9.4.

9.2 Local-Level Model

A *local-level model* is also referred to as a *random walk plus noise model* and is suitable for expressing a model that does not have a particular time pattern and retains similar values for the short term. We have already applied this model to the flow data of the Nile in Sect. 8.2. We reprint here the state and observation equations

with time-invariant state and observation noises; they are the same as Eqs. (8.9) and (8.10):

$$x_t = x_{t-1} + w_t, \quad w_t \sim \mathcal{N}(0, W), \quad (9.5)$$

$$y_t = x_t + v_t, \quad v_t \sim \mathcal{N}(0, V). \quad (9.6)$$

These correspond to Eqs. (5.8) and (5.9) with the following settings:

$$\mathbf{x}_t = [x_t], \quad \mathbf{G}_t = [1], \quad \mathbf{W}_t = W, \quad (9.7)$$

$$\mathbf{F}_t = [1], \quad \mathbf{V}_t = V. \quad (9.8)$$

9.2.1 Example: Artificial Local-Level Model

Consider an artificial local-level model with parameters $W = 1$, $V = 2$, $m_0 = 10$, and $C_0 = 9$. We can achieve such a model through the following settings in the component decomposition expression by Eqs. (9.3) and (9.4):

- Eqs. (9.7) and (9.8) are applied to the first component model.

This code is as follows.

Code 9.1

```

1 > # <<Generate artificial data that obey local-level model>>
2 >
3 > # Preprocessing
4 > set.seed(23)
5 > library(dlm)
6 >
7 > # Setting of local-level model
8 > W <- 1
9 > V <- 2
10 > m0 <- 10
11 > C0 <- 9
12 > mod <- dlmModPoly(order = 1, dW = W, dV = V, m0 = m0, C0 = C0)
13 >
14 > # Generate observations using Kalman prediction
15 > t_max <- 200
16 > sim_data <- dlmForecast(mod = mod, nAhead = t_max, sampleNew = 1)
17 > y <- sim_data$newObs[[1]]
18 >
19 > # Cast the result to ts class
20 > y <- ts(as.vector(y))
21 >
22 > # Plot results
23 > plot(y, ylab = "y")
```

Just as in Code 7.1, we generate the data by setting the argument `sampleNew = 1` for the function `dlmForecast()` in library **dlm**. The time series length `t_max` is set to 200. Figure 9.1 shows the data plot.

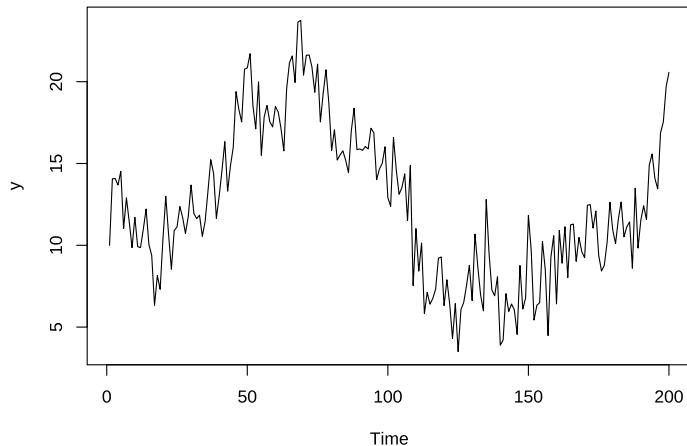


Fig. 9.1 Artificial data obeying local-level model

Now, we analyze this model using the Kalman filter. We ignore the code display because it is nearly the same as in Sect. 8.2. We reuse the generated data later; hence, we save it with the other related information, such as the model and results of filtering and smoothing as `ArtificialLocalLevelModel.RData` in advance. Figures 9.2, 9.3, and 9.4 shows the plot of the results of filtering, prediction, and smoothing, respectively. Each figure displays the mean and 95% probability intervals for the estimated distribution.

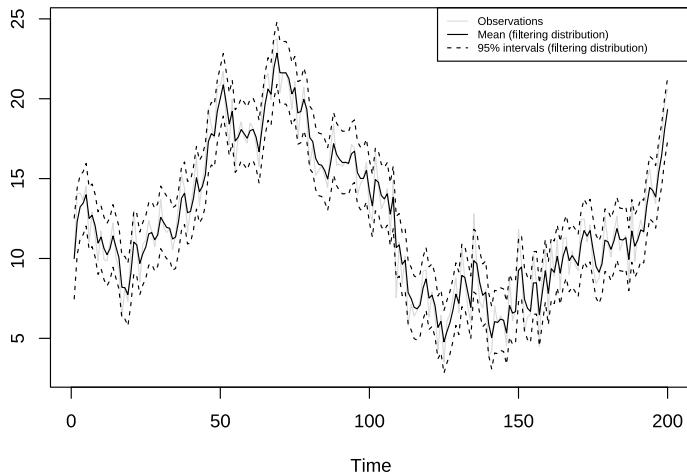


Fig. 9.2 Analysis of artificial local-level model with a Kalman filter (filtering)

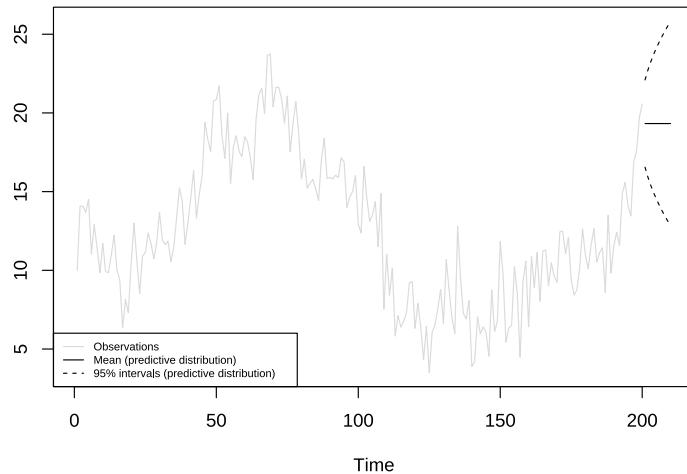


Fig. 9.3 Analysis of artificial local-level model with a Kalman filter (prediction)

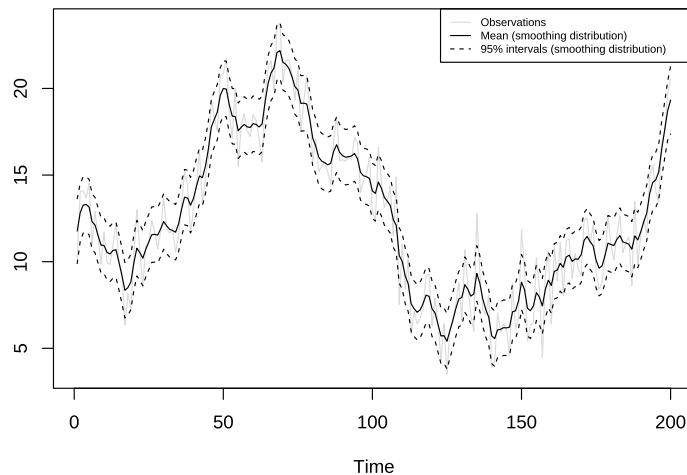


Fig. 9.4 Analysis of artificial local-level model with a Kalman filter (smoothing)

9.3 Local-Trend Model

A *local-trend model* is also known as a *linear growth model* and can express a linear slope in the data. Thus, this model is suitable for describing time series data with a trend of rising and falling for a particular term. Regarding some data in Chap. 4, for example, we can capture the entire increasing trend excluding periodicity: (b) CO₂ concentration in the atmosphere and (c) quarterly gas consumption in the UK.

The local-trend model is generally a polynomial model of order two. Based on the fact that the local-level model explained in the previous section can also be

interpreted as a polynomial model of order one, we describe the general *polynomial model* in advance. A state in the polynomial model comprises N elements, i.e., $[x_t^{(N)}, \dots, x_t^{(n)}, \dots, x_t^{(1)}]^\top$, and we distinguish each element with a superscript (n) and refer to N as the order of the polynomial model. The state and observation equations for the polynomial model with time-invariant state and observation noises are as follows:

$$x_t^{(n)} = x_{t-1}^{(n)} + x_{t-1}^{(n-1)} + w_t^{(n)}, \quad (n = N, \dots, 2) \quad w_t^{(n)} \sim \mathcal{N}(0, W^{(n)}), \quad (9.9)$$

$$x_t^{(1)} = x_{t-1}^{(1)} + w_t^{(1)}, \quad w_t^{(1)} \sim \mathcal{N}(0, W^{(1)}), \quad (9.10)$$

$$y_t = x_t^{(N)} + v_t, \quad v_t \sim \mathcal{N}(0, V). \quad (9.11)$$

These correspond to Eqs. (5.8) and (5.9) with the following settings:

$$\mathbf{x}_t = \begin{bmatrix} x_t^{(N)} \\ \vdots \\ x_t^{(2)} \\ x_t^{(1)} \end{bmatrix}, \quad \mathbf{G}_t = \begin{bmatrix} 1 & 1 \\ & 1 & 1 \\ & & \ddots & \ddots \\ & & & 1 & 1 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}, \quad \mathbf{W}_t = \begin{bmatrix} W^{(N)} & & & \\ & \ddots & & \\ & & W^{(2)} & \\ & & & W^{(1)} \end{bmatrix}, \quad (9.12)$$

$$\mathbf{F}_t = [1, 0, \dots, 0], \quad V_t = V. \quad (9.13)$$

The name of the polynomial model comes from its forecast function $E[y_{t+k} | y_{1:t}]$, which is a polynomial of degree $N - 1$ for k [11].

Setting $w_t^{(n)} = 0$ in Eq. (9.9) reveals the relation $x_t^{(n)} - x_{t-1}^{(n)} = x_{t-1}^{(n-1)}$. Based on this property, we can interpret the meaning of each element in the state as follows:

$x_t^{(N)}$	level for short term
$x_t^{(N-1)}$	slope (trend) for short term
$x_t^{(N-2)}$	curvature (degree of curve) for short term
\vdots	

We typically use polynomial models of the order up to $N = 1$ or 2, corresponding to the local-level model or local-trend model, respectively. Polynomial models of the order $N = 3$ and more are not generally used because with an increase in N , the estimation result overfits the noise and becomes difficult to interpret. However, we can use them if the data generative process is clearly compatible.

The polynomial model with the setting $W^{(N)} = \dots = W^{(2)} = 0$ is particularly referred to the *integrated random walk* model [11]. The integrated random walk model simplifies the polynomial model because it reduces the number of parameters compared to the standard polynomial model; its suitability finally depends on the problem.

Incidentally, this model is equivalent to taking the N -th difference of the data at preprocessing; hence, we can recognize that the integrated random walk model in the state-space model corresponds to the difference at preprocessing in the ARIMA model.

We return to the topic of the local-trend model again. The state and observation equations of the local-trend model with time-invariant state and observation noises are as follows:

$$x_t^{(2)} = x_{t-1}^{(2)} + x_{t-1}^{(1)} + w_t^{(2)}, \quad w_t^{(2)} \sim \mathcal{N}(0, W^{(2)}), \quad (9.14)$$

$$x_t^{(1)} = x_{t-1}^{(1)} + w_t^{(1)}, \quad w_t^{(1)} \sim \mathcal{N}(0, W^{(1)}), \quad (9.15)$$

$$y_t = x_t^{(2)} + v_t, \quad v_t \sim \mathcal{N}(0, V). \quad (9.16)$$

These correspond to Eqs. (5.8) and (5.9) with the following settings:

$$\mathbf{x}_t = \begin{bmatrix} x_t^{(2)} \\ x_t^{(1)} \end{bmatrix}, \quad \mathbf{G}_t = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{W}_t = \begin{bmatrix} W^{(2)} & 0 \\ 0 & W^{(1)} \end{bmatrix}, \quad (9.17)$$

$$\mathbf{F}_t = [1, 0], \quad \mathbf{V}_t = V. \quad (9.18)$$

The example of the local-trend model is illustrated with the seasonal model in the next section.

9.4 Seasonal Model

The *seasonal model* is also known as *periodical model* and is suitable for expressing a model that has a clear periodic pattern in the observations. This section explains two approaches for defining the model: time domain and frequency domain approaches. While both the approaches are substantially equivalent, their expressions are different; each approach has its own advantages.

9.4.1 Approach from the Time Domain

This approach allows intuitive interpretation of state elements and is easy to understand. In addition, we can easily extend this model to a more applied model, such as the one considering the similarity with the most recent specific day of the week [16].

At first, in the time domain, the periodic data repeat almost the same pattern for each cycle. From this viewpoint, we can assume that the sum of the seasonal components for one cycle becomes almost the same as that in other cycles. Let the cycle be s and the seasonal component be x_t ; then, such a property is expressed as follows:

$$\sum_{t=1}^s x_t = \text{almost constant that does not differ from the sum in another cycle.} \quad (9.19)$$

In the above equation, the state-space model often treats the offset for the full seasonal components in terms of the polynomial model; hence, the “constant” can be considered as 0. However, we remain the concept of fluctuation that makes the model flexible. When we assume that such a fluctuation has a normal distribution with mean 0, Eq. (9.19) becomes as follows:

$$\sum_{t=1}^s x_t = w_t, \quad w_t \sim \mathcal{N}(0, W). \quad (9.20)$$

This Eq. (9.20) imposes a constraint on the seasonality.

Next, as a specific explanation, consider the data with quarter seasonality. Let the seasonal component at time point t be $[x_{t-4}, x_{t-3}, x_{t-2}, x_{t-1}]^\top = [x_t^{(1Q)}, x_t^{(2Q)}, x_t^{(3Q)}, x_t^{(4Q)}]^\top$. Note that the first element $x_{t-4} = x_t^{(1Q)}$ is only reflected to the observations finally. The time transitions of these elements are expressed as follows:

$$x_t^{(1Q)} \leftarrow x_{t-1}^{(4Q)}, \quad (9.21)$$

$$x_t^{(2Q)} \leftarrow x_{t-1}^{(1Q)}, \quad (9.22)$$

$$x_t^{(3Q)} \leftarrow x_{t-1}^{(2Q)}, \quad (9.23)$$

$$x_t^{(4Q)} \leftarrow x_{t-1}^{(3Q)}. \quad (9.24)$$

Recalling that the value of 4Q can be defined by those of 1Q, 2Q, and 3Q from the constraint in Eq. (9.20), we further reduce the above equations. The right-hand side of Eq. (9.21) can be rewritten with the values of 1Q, 2Q, and 3Q. In addition, Eq. (9.24) is actually unnecessary because there are Eqs. (9.21), (9.22), and (9.23). In accordance with the above, the time transitions are simplified as

$$\begin{aligned} x_t^{(1Q)} &\leftarrow -x_{t-1}^{(1Q)} - x_{t-1}^{(2Q)} - x_{t-1}^{(3Q)} + w_t, \\ x_t^{(2Q)} &\leftarrow x_{t-1}^{(1Q)}, \\ x_t^{(3Q)} &\leftarrow x_{t-1}^{(2Q)}. \end{aligned}$$

Finally, we generally define the state and observation equations based on the above descriptions. The state and observation equations for the seasonal model in the time-domain approach, with time-invariant state and observation noises, correspond to Eqs. (5.8) and (5.9) with the following settings:

$$\begin{aligned} \boldsymbol{x}_t &= \begin{bmatrix} x_t^{(1)} \\ x_t^{(2)} \\ \vdots \\ x_t^{(s-1)} \end{bmatrix}, \quad \boldsymbol{G}_t = \begin{bmatrix} -1 & \cdots & -1 & -1 \\ 1 & & & \\ & \ddots & & \\ & & 1 & \end{bmatrix}, \quad \boldsymbol{W}_t = \begin{bmatrix} W & & & \\ 0 & & & \\ & \ddots & & \\ & & & 0 \end{bmatrix}, \quad (9.25) \\ \boldsymbol{F}_t &= [1, 0, \dots, 0], \quad V_t = V. \quad (9.26) \end{aligned}$$

9.4.2 Approach from the Frequency Domain

This approach is based on Fourier series expansion and can easily adjust the number N of frequency components to be considered. While there are various kinds of seasonalities, the gradual seasonality can lead to a more natural interpretation than the complicated one in many cases, and the former is further able to suppress the overfitting to the noise. The setting low N in this model can achieve such a gradual seasonality and even sometimes improve the marginal likelihood as a result.

First, as we have discussed the Fourier series in Sect. 4.2.4, arbitrary periodic data can generally be represented by the infinite sum of its frequency components. Let the periodic real-valued signal be γ_t ; then, from Eq. (4.1), its complex-valued Fourier series can be expressed as follows:

$$\begin{aligned} \gamma_t &= \sum_{n=-\infty}^{\infty} c_n e^{in\omega_0 t} \\ &= c_0 + \sum_{n=1}^{\infty} \bar{c}_n e^{-in\omega_0 t} + \sum_{n=1}^{\infty} c_n e^{in\omega_0 t}, \end{aligned} \quad (9.27)$$

where $\omega_0 = 2\pi/s$ is the fundamental angular frequency and s is the (fundamental) period. We proceed with further consideration of each term in Eq. (9.27) to reduce the expression.

The first term is a component with frequency 0, i.e., no periodicity, implying an offset for the entire periodic signal¹. This term is often treated with a polynomial model in the state-space model and can be ignored.

The second and third terms contribute to the negative and positive frequency components, respectively, and negative and positive frequencies abstractly imply clockwise and counterclockwise rotations, respectively. If we treat the real-valued periodic signals in the state-space model, we can ignore either of them because the second and third terms have a complex conjugate relationship; this relation makes the imaginary part of real-valued signal γ_t become zero. Thus, if the observation matrix doubles the real part (in-phase component) and neglects the imaginary part (quadrature-phase component) of either term, the other becomes unnecessary. We decide here to ignore the third term, contributing to the positive frequency component.

In addition, while the Fourier series is generally an infinite sum, if and only if the period s is a rational number, it can be represented by a finite sum through $\lfloor s/2 \rfloor$ from the sampling theorem; recall that the periodic signal to be expanded in this case is represented through its samples at discrete time points with equal intervals.

In accordance with the above description, we redefine Eq. (9.27) with a finite sum of only negative frequency components as follows:

$$\begin{aligned} \gamma_t &:= \sum_{n=1}^N 2\bar{c}_n e^{-in\omega_0 t} \\ &= \sum_{n=1}^N \gamma_t^{(n)}, \end{aligned} \quad (9.28)$$

where we set $\gamma_t^{(n)} = 2\bar{c}_n e^{-in\omega_0 t}$ for each frequency component at time t .

Next, considering the time transition of $\gamma_t^{(n)}$, we have

$$\begin{aligned} \gamma_{t+1}^{(n)} &= 2\bar{c}_n e^{-in\omega_0(t+1)} \\ &= 2\bar{c}_n e^{-in\omega_0 t} e^{-in\omega_0} \\ &= \gamma_t^{(n)} e^{-in\omega_0} \\ &= \left(\operatorname{Re}(\gamma_t^{(n)}) + i \operatorname{Im}(\gamma_t^{(n)}) \right) (\cos(n\omega_0) - i \sin(n\omega_0)) \\ &= \left(\operatorname{Re}(\gamma_t^{(n)}) \cos(n\omega_0) + \operatorname{Im}(\gamma_t^{(n)}) \sin(n\omega_0) \right) \\ &\quad + i \left(-\operatorname{Re}(\gamma_t^{(n)}) \sin(n\omega_0) + \operatorname{Im}(\gamma_t^{(n)}) \cos(n\omega_0) \right). \end{aligned} \quad (9.29)$$

We can rewrite the above equation by representing the real and imaginary parts of $\gamma_t^{(n)}$ as elements of the vector:

¹ Engineers usually refer to this as a direct current component.

$$\begin{bmatrix} \operatorname{Re}(\gamma_{t+1}^{(n)}) \\ \operatorname{Im}(\gamma_{t+1}^{(n)}) \end{bmatrix} = \begin{bmatrix} \cos(n\omega_0) & \sin(n\omega_0) \\ -\sin(n\omega_0) & \cos(n\omega_0) \end{bmatrix} \begin{bmatrix} \operatorname{Re}(\gamma_t^{(n)}) \\ \operatorname{Im}(\gamma_t^{(n)}) \end{bmatrix} \\ = \text{rotation matrix}^{(n)} \begin{bmatrix} \operatorname{Re}(\gamma_t^{(n)}) \\ \operatorname{Im}(\gamma_t^{(n)}) \end{bmatrix}, \quad (9.30)$$

where rotation matrix⁽ⁿ⁾ = $\begin{bmatrix} \cos(n\omega_0) & \sin(n\omega_0) \\ -\sin(n\omega_0) & \cos(n\omega_0) \end{bmatrix}$. We see that this is a state equation wherein the state for each frequency component is represented by $\begin{bmatrix} \operatorname{Re}(\gamma_t^{(n)}) \\ \operatorname{Im}(\gamma_t^{(n)}) \end{bmatrix}$. However, the state noise is not yet considered. Note that we ignore $\operatorname{Im}(\gamma_t^{(n)})$ eventually and assign $\operatorname{Re}(\gamma_t^{(n)})$ only to the observations.

Finally, we define the state and observation equations generally based on the above descriptions. The state and observation equations for the seasonal model in the frequency-domain approach with time-invariant state and observation noises correspond to Eqs. (5.8) and (5.9) with the following settings:

$$\mathbf{x}_t = \begin{bmatrix} \operatorname{Re}(\gamma_t^{(1)}) \\ \operatorname{Im}(\gamma_t^{(1)}) \\ \operatorname{Re}(\gamma_t^{(2)}) \\ \operatorname{Im}(\gamma_t^{(2)}) \\ \vdots \\ \operatorname{Re}(\gamma_t^{(N-1)}) \\ \operatorname{Im}(\gamma_t^{(N-1)}) \\ \operatorname{Re}(\gamma_t^{(N)}) \\ \operatorname{Im}(\gamma_t^{(N)}) \end{bmatrix},$$

$$\mathbf{G}_t = \begin{bmatrix} \text{rotation matrix}^{(1)} & & & & \\ & \text{rotation matrix}^{(2)} & & & \\ & & \ddots & & \\ & & & \text{rotation matrix}^{(N-1)} & \\ & & & & \text{rotation matrix}^{(N)} \end{bmatrix},$$

$$\mathbf{W}_t = \begin{bmatrix} W^{(1)} & & & & \\ & W^{(1)} & & & \\ & & W^{(2)} & & \\ & & & W^{(2)} & \\ & & & & \ddots \\ & & & & & W^{(N-1)} \\ & & & & & & W^{(N-1)} \\ & & & & & & & W^{(N)} \\ & & & & & & & & W^{(N)} \end{bmatrix}, \quad (9.31)$$

$$\mathbf{F}_t = [1, 0, 1, 0, \dots, 1, 0, 1, 0],$$

$$\mathbf{V}_t = V, \quad (9.32)$$

where N is an arbitrary natural number, the rotation matrix⁽ⁿ⁾ = $\begin{bmatrix} \cos(n\omega_0) & \sin(n\omega_0) \\ -\sin(n\omega_0) & \cos(n\omega_0) \end{bmatrix}$, $\omega_0 = 2\pi/s$, and s is a (fundamental) period. If the period s is a rational number, the maximum value of N becomes $\lfloor s/2 \rfloor$. In particular, if s is an even number, the rotation matrix^(N) becomes diagonal because of $N\omega_0 = \frac{s}{2}2\pi/s = \pi$ at $N = s/2$;

therefore, $\text{Re}(\gamma_t^{(N)})$ and $\text{Im}(\gamma_t^{(N)})$ no longer have a relation. In such a case, we can remove $\text{Im}(\gamma_t^{(N)})$ from the above equation in advance because it is eventually ignored through the observation equation. Based on this decision, when s is an even number, the settings for the linear Gaussian state-space model become as follows:

$$\begin{aligned} \mathbf{x}_t &= \begin{bmatrix} \text{Re}(\gamma_t^{(1)}) \\ \text{Im}(\gamma_t^{(1)}) \\ \text{Re}(\gamma_t^{(2)}) \\ \text{Im}(\gamma_t^{(2)}) \\ \vdots \\ \text{Re}(\gamma_t^{(N-1)}) \\ \text{Im}(\gamma_t^{(N-1)}) \\ \text{Re}(\gamma_t^{(N)}) \end{bmatrix}, \\ \mathbf{G}_t &= \begin{bmatrix} \text{rotation matrix}^{(1)} & & & & & \\ & \text{rotation matrix}^{(2)} & & & & \\ & & \ddots & & & \\ & & & \text{rotation matrix}^{(N-1)} & & \\ & & & & \cos(\pi) & \\ \mathbf{W}_t &= \begin{bmatrix} W^{(1)} & & & & & \\ & W^{(1)} & & & & \\ & & W^{(2)} & & & \\ & & & W^{(2)} & & \\ & & & & \ddots & \\ & & & & & W^{(N-1)} \\ & & & & & & W^{(N-1)} \\ & & & & & & & W^{(N)} \end{bmatrix}, \end{aligned} \quad (9.33)$$

$$\begin{aligned} \mathbf{F}_t &= [1, 0, 1, 0, \dots, 1, 0, 1], \\ V_t &= V. \end{aligned} \quad (9.34)$$

9.4.3 Example: CO₂ Concentration in the Atmosphere

Now, we analyze CO₂ concentration data in the atmosphere, mentioned in Chap. 4, using the linear Gaussian state-space model. As noted in Sect. 4.3, these data show an annual seasonality and an increasing trend. Thus, we apply a combination of the local-trend and seasonal models. Based on this decision, we compare three model variations.

9.4.3.1 Local-Trend Model + Seasonal Model (Time-Domain Approach)

This case uses the seasonal model with the time-domain approach. We can achieve such a model through the following setting in the component decomposition Eqs. (9.3) and (9.4):

- Eqs. (9.17) and (9.18) are applied to the first component model.

- Eqs. (9.25) and (9.26) are applied to the second component model.

This code is as follows.

Code 9.2

```

1 > # <<Local-trend model + seasonal model (time-domain approach)>>
2 >
3 > # Preprocessing
4 > library(dlm)
5 >
6 > # Load the data
7 > Ryori <- read.csv("CO2.csv")
8 >
9 > # Cast the data to ts class, truncating it by December 2014
10 > y_all <- ts(data = Ryori$CO2, start = c(1987, 1), frequency = 12)
11 > y <- window(y_all, end = c(2014, 12))
12 >
13 > # Model setting: local-trend model + seasonal model (time-domain approach)
14 > build_dlm_CO2a <- function(par) {
15 +   return(
16 +     dlmModPoly(order = 2, dW = exp(par[1:2]), dV = exp(par[3])) +
17 +     dlmModSeas(frequency = 12, dW = c(exp(par[4]), rep(0, times = 10)), dV = 0)
18 +   )
19 + }
20 >
21 > # Maximum likelihood estimation of parameters and confirmation of the results
22 > fit_dlm_CO2a <- dlmMLE(y = y, parm = rep(0, 4), build = build_dlm_CO2a)
23 > fit_dlm_CO2a
24 $par
25 [1] -2.50347101 -21.93720237 -0.09833124 -5.18269911
26
27 $value
28 [1] 330.4628
29
30 $counts
31 function gradient
32      39      39
33
34 $convergence
35 [1] 0
36
37 $message
38 [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
39
40 >
41 > # Set the maximum likelihood estimates of parameters in the model
42 > mod <- build_dlm_CO2a(fit_dlm_CO2a$par)
43 >
44 > # Kalman filtering
45 > dlmFiltered_obj <- dlmFilter(y = y, mod = mod)
46 > dlmFiltered_obja <- dlmFiltered_obj           # Save under a different name for
47   ↪ later comparison of prediction values
48 >
49 > mu <- dropFirst(dlmFiltered_obj$m[, 1])
50 > gamma <- dropFirst(dlmFiltered_obj$m[, 3])
51 >
52 > # Plot results
53 > oldpar <- par(no.readonly = TRUE)
54 > par(mfrow = c(3, 1)); par(oma = c(2, 0, 0, 0)); par(mar = c(2, 4, 1, 1))

```

```

55 | > ts.plot(    y, ylab = "Observations")
56 | > ts.plot(   mu, ylab = "Level component", ylim = c(350, 405))
57 | > ts.plot(gamma, ylab = "Seasonal component" , ylim = c( -9, 6))
58 | > mtext(text = "Time", side = 1, line = 1, outer = TRUE)
59 | > par(oldpar)
60 |
61 | > # Confirm the log-likelihood
62 | > -dlmLL(y = y, mod = mod)
63 | [1] -330.4628

```

As described in Chap. 3, we have downloaded the data from the Japan meteorological agency's website and saved it as the file CO2.csv with slight modification for ease of handling in R. The data are read from this file, and the observations through December 2014 are set to y . Then, we prepare the user-defined function `build_dlm_CO2a()` for creating the model. The functions `d1mModPoly()` and `d1mModSeas()` in the library `d1m` are combined using `+`. The local-trend model is realized by setting `order = 2` in the argument of `d1mModPoly()`. The arguments `dW` and `dV` indicate the diagonal elements $W^{(2)}$ and $W^{(1)}$ in the covariance of the state noise and variance V of the observation noise, respectively. These are respectively set to `exp(par[1:2])` and `exp(par[3])` based on the argument `par` of `build_dlm_CO2a()`. The arguments `m0` and `C0` are set to the default values, i.e., $\mathbf{0}$ and $10^7\mathbf{I}$, respectively, because the setting is omitted; these values can be used in this example without any problem. The function `d1mModSeas()` sets the seasonal model in the time-domain approach. The arguments of this function are `frequency`, `dW`, `dV`, `m0`, and `C0`. The argument `frequency` indicates the period s ; hence, we set `frequency = 12` as the annual cycle. The arguments `dW` and `dV` indicate the diagonal elements $W, 0, \dots, 0$ in the covariance of the state noise and variance V of the observation noise, respectively. The argument `dW` is set to `c(exp(par[4]), rep(0, times = 10))` based on the argument `par` of `build_dlm_CO2a()`. The argument `dV` has already been set through `d1mModPoly()`; hence, we set `0` here. The arguments `m0` and `C0` indicate the mean vector \mathbf{m}_0 and covariance \mathbf{C}_0 of the prior distribution, respectively. They are set to the default values, i.e., $\mathbf{0}$ and $10^7\mathbf{I}$, respectively, because the setting is omitted; these values can be used in this example without any problem. Subsequently, the maximum likelihood estimation is performed on the parameters, and the result is assigned to `fit_dlm_CO2a`. We cannot see any particular problem in the result. We then set the maximum likelihood estimates of the parameters in the model to perform Kalman filtering. Figure 9.5 shows the plotting results.

The figure shows three plots, namely observations, level component (mean of the filtering distribution), and seasonal component (mean of the filtering distribution) in order from the top. The combination of individual models in the state-space model easily enables such component decomposition for time series data. From the level and seasonal components, we understand that the estimation accuracy for the first year deteriorates because the mean vector of the prior distribution is set to $\mathbf{0}$ and consistency with the data are not particularly considered. Such a situation can be mitigated through mean vector refinement of the prior distribution or Kalman smoothing, but we do not pursue this issue here. Seeing April 2011, we also understand that the

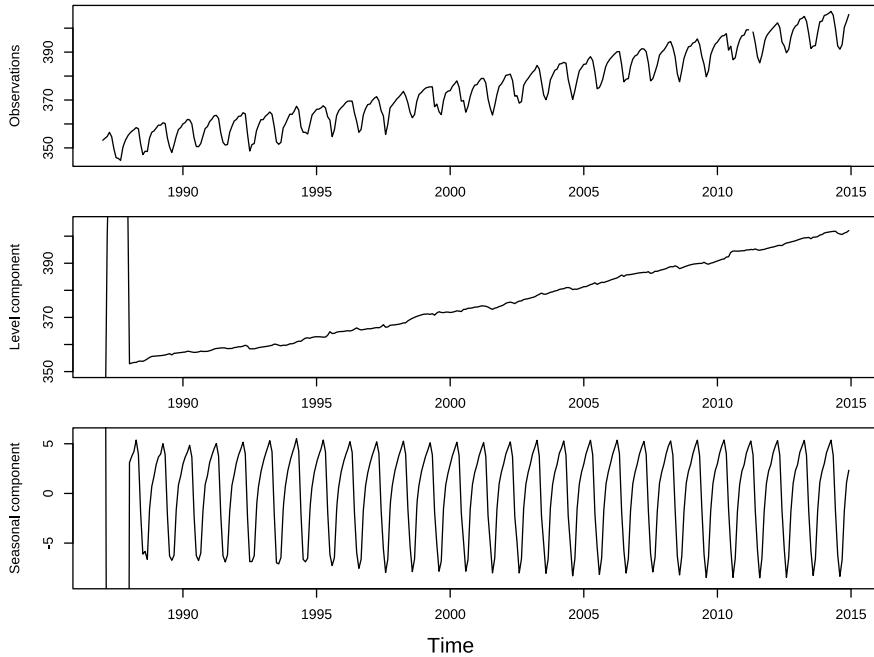


Fig. 9.5 Filtering result by the local-trend model + seasonal model (time-domain approach)

estimation is performed without any problem even if observations at that point are missing. Finally, we confirm the log-likelihood value, -330.4628 . If we change the model further, the log-likelihood value must be compared.

9.4.3.2 Local-Level Model + Seasonal Model (Time-Domain Approach)

Now, consider an alternative model wherein we substitute the local-level model for the local-trend model for comparison. We can achieve such a model through the following setting in the component decomposition Eqs. (9.3) and (9.4):

- Eqs. (9.7) and (9.8) are applied to the first component model.
- Eqs. (9.25) and (9.26) are applied to the second component model.

The shaded part indicates the difference between the above model and the first model. This code is written as follows.

Code 9.3

```

64 > # <<Local-level model + seasonal model (time-domain approach)>>
65 >
66 > # Model setting: local-level model + seasonal model (time-domain approach)
67 > build_dlm_CO2b <- function(par) {
68 +   return(
69 +     dlmModPoly(order = 1, dW = exp(par[1]), dV = exp(par[2])) +
70 +     dlmModSeas(frequency = 12, dW = c(exp(par[3]), rep(0, times = 10)), dV = 0)
71 +   )
72 + }
73 >
74 > # Ignore the display of following codes

```

The above code continues from the former Code 9.2. First, we prepare the user-defined function `build_dlm_CO2b()` for creating the model. However, unlike the first user-defined function `build_dlm_CO2a()`, the local-level model is now realized by setting `order = 1` in the argument of `dlmModPoly()`. The shaded part of the code also indicates the difference between the above code and the first code. Subsequently, maximum likelihood estimation is performed on the parameters, and the result is assigned to `fit_dlm_CO2b`. We cannot recognize any particular problem in the result. We then set the maximum likelihood estimates of the parameters in the model to perform Kalman filtering. Figure 9.6 shows the plotting results.

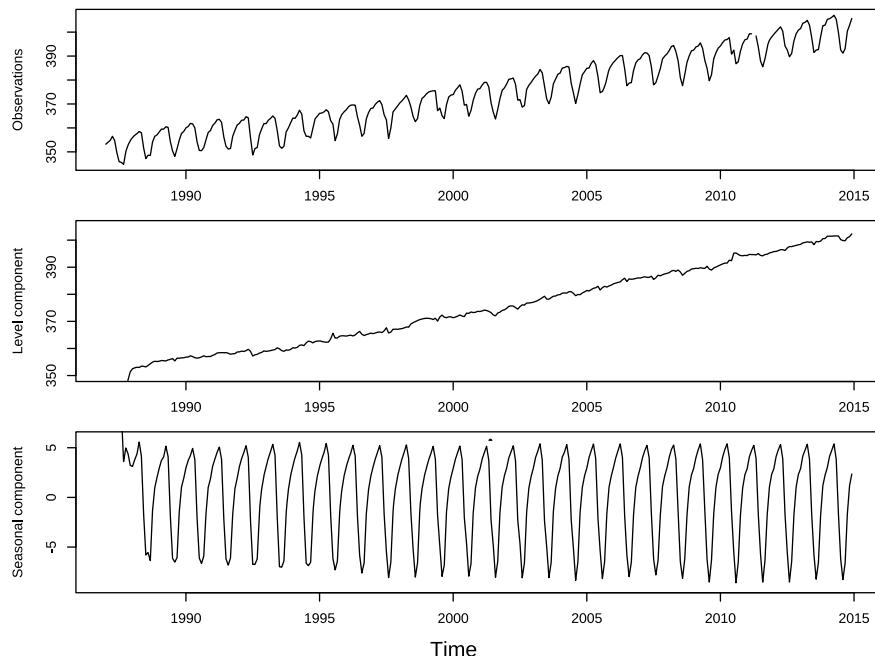


Fig. 9.6 Filtering result by the local-level model + seasonal model (time-domain approach)

The above figure shows three plots: observations, level component (mean of the filtering distribution), and seasonal component (mean of the filtering distribution) in order from the top. Seeing the level and seasonal components, we can understand that the estimation accuracy for the first year deteriorates similarly in Fig. 9.5. When we compare the above figure with Fig. 9.5, the level component appears to be more jagged. Finally, we confirm the log-likelihood value, -340.1425 . Since this value is less than -330.4628 , which was obtained from the first “local-trend model + seasonal model (time-domain approach),” we understand that the previous model considering the trend is more appropriate.

9.4.3.3 Local-Trend Model + Seasonal Model (Frequency-Domain Approach)

The final case uses the seasonal model in the frequency-domain approach. We can achieve such a model through the following setting in the component decomposition Eqs. (9.3) and (9.4):

- Eqs. (9.17) and (9.18) are applied to the first component model.
- Eqs. (9.31) and (9.32) are applied to the second component model.

The shaded part indicates the difference between the above and first models. This code is as follows.

Code 9.4

```

75 > # <<Local-trend model + seasonal model (frequency domain approach)>>
76 >
77 > # Model setting: local-trend model + seasonal model (frequency domain approach)
78 > build_dlm_CO2c <- function(par) {
79 +   return(
80 +     dlmModPoly(order = 2, dW = exp(par[1:2]), dV = exp(par[3])) +
81 +     dlmModTrig(s = 12, q = 2, dW = exp(par[4]), dV = 0)
82 +   )
83 + }
84 >
85 > # Ignore the display of following codes

```

The above code continues from the former Code 9.3. First, we prepare the user-defined function `build_dlm_CO2c()` for creating the model. However, unlike the first user-defined function `build_dlm_CO2a()`, `dlmModTrig()` is used instead of `dlmModSeas()`. The shaded part in the code also indicates the difference between the above and first codes. The function `dlmModTrig()` in library **dlm** can set the seasonal model in the frequency-domain approach. The arguments of this function are `s`, `q`, `dV`, `dW`, `m0`, `C0`, `om`, and `tau`. This example does not use `om` and `tau` because they are set when the fundamental period `s` is not an integer. The argument `s` indicates the fundamental period `s`, which is an integer in this example, and we set `s = 12` as the annual cycle. The argument `q` indicates the value N of the number of frequency components we leave in order from the low frequency. Such a value must be optimized, and this example sets 2 as the optimum value; we

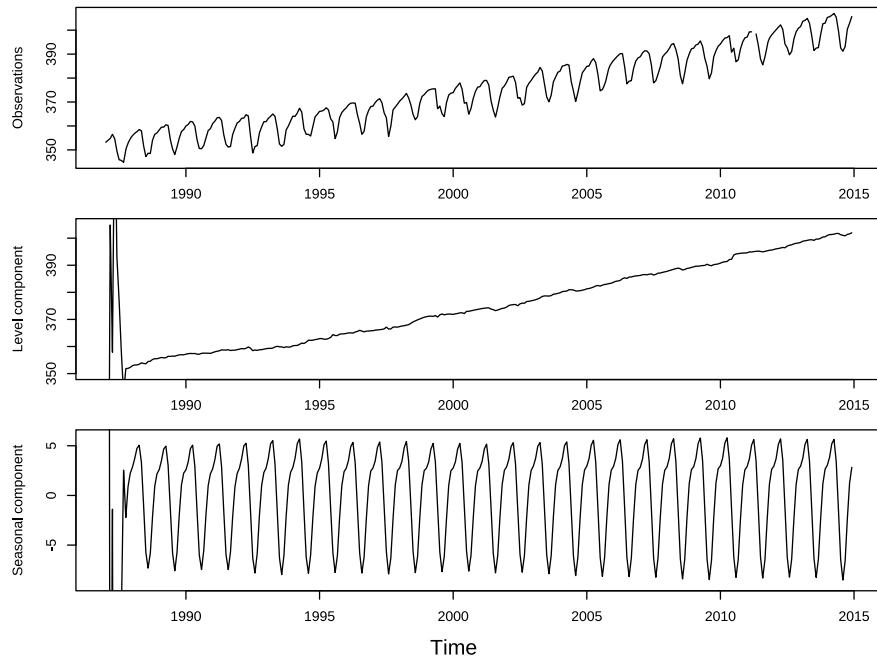


Fig. 9.7 Filtering result by the local-trend model + seasonal model (frequency-domain approach)

have previously compared the marginal likelihood through the various settings. The arguments `dW` and `dV` indicate the diagonal elements in the covariance of the state noise and the variance V of the observation noise, respectively. The seasonal model in the frequency-domain approach can set the variance of the state noise individually as $W^{(1)}, \dots, W^{(N)}$ for each frequency component. However, if there is no particular prior information, we typically set the unique value W to them. Since there are also no specific informations in this example, `dW` is set to `exp(par[4])` as a unique value based on the argument `par` of `build_dlm_CO2c()`. The argument `dV` has already been set through `dlmModPoly()`; hence, we set 0 here. The arguments `m0` and `C0` indicate the mean vector \mathbf{m}_0 and covariance \mathbf{C}_0 of the prior distribution, respectively. They are set to the default values, that is, $\mathbf{0}$ and $10^7\mathbf{I}$, respectively, because the setting is omitted; these values can be used in this example with no problem. Subsequently, maximum likelihood estimation is performed on the parameters, and the result is assigned to `fit_dlm_CO2c`. We cannot see any particular problem in the result. We then set the maximum likelihood estimates of the parameters in the model to perform Kalman filtering. Figure 9.7 shows the plotting results.

The figure shows three plots: observations, level component (mean of the filtering distribution), and seasonal component (mean of the filtering distribution) in order from the top. Seeing the level and seasonal components, we understand that the estimation accuracy for the first year deteriorates similarly in Fig. 9.5. When we compare the above figure with Fig. 9.5, the lower values of the seasonal component appear to

be more consistent particularly during the initial periods. Finally, we confirm the log-likelihood value, -284.6092 . Since this value is greater than -330.4628 obtained from the first “local-trend model + seasonal model (time-domain approach),” we understand that this model is more appropriate. This result is obtained because we truncate the seasonal components with higher frequencies to suppress overfitting the signal distortion.

Furthermore, we perform the prediction from 2015 using this model. The data `y_all` including observations `y` through December 2014 also contain the preliminary data for the year 2015; hence, we use these data as a correct answer. The prediction code is as follows.

Code 9.5

```

86 > # <<Forecasts from 2015>>
87 >
88 > # Kalman prediction
89 > dlmForecasted_object <- dlmForecast(mod = dlmFiltered_obj, nAhead = 12)
90 >
91 > # Find the standard deviation and the 2.5% and 97.5% values of the prediction value
92 > f_sd <- sqrt(as.numeric(dlmForecasted_object$Q))
93 > f_lower <- dlmForecasted_object$f + qnorm(0.025, sd = f_sd)
94 > f_upper <- dlmForecasted_object$f + qnorm(0.975, sd = f_sd)
95 >
96 > # Unite the entire observation along with the mean, 2.5%, and 97.5% values of the
  ← prediction values into ts class object
97 > y_union <- ts.union(y_all, dlmForecasted_object$f, f_lower, f_upper)
98 >
99 > # Ignore the display of following codes

```

The above code continues from the former Code 9.4. The function `dlmForecast()` performs the prediction for one year from the end of `y`. From the result, we calculate some statistics for comparison and unite them with `y_all` using the R function `ts.union()`. Figure 9.8 shows the plotting results.

The prediction result of Fig. 9.8 is considered to be almost reasonable.

We also compare the prediction results during 2015 for the three models considered thus far. For convenience of comparison, we refer to these three models as (a), (b), and (c):

- (a) local-trend + season (time-domain approach)
- (b) local-level + season (time-domain approach)
- (c) local-trend + season (frequency-domain approach).

Since we have saved the objects for each filtering result individually as `dlmFiltered_obja`, `dlmFiltered_objb`, and `dlmFiltered_objc` through previous codes, applying the function `dlmForecast()` to them, we can obtain the prediction results for each model; the result is finally united into a `ts` type object. This code is as follows.

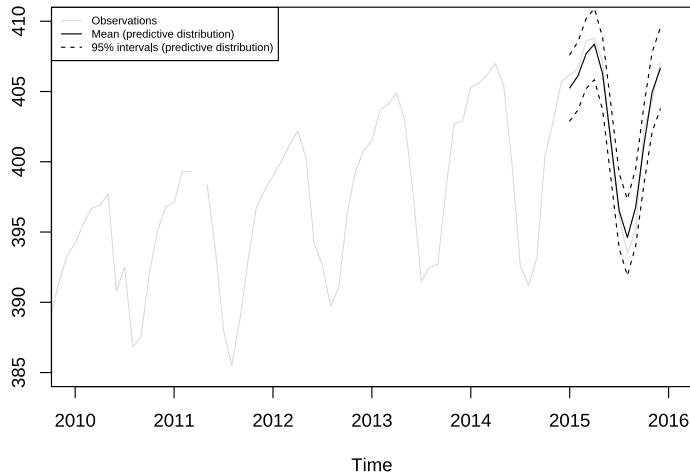


Fig. 9.8 Prediction results from 2015

Code 9.6

```

100 > # <<Comparison of forecasts from 2015 among three models>>
101 >
102 > # Find the mean, 2.5%, and 97.5% values of the prediction values for each of models
103 <- a, b, and c
104 f_all <- lapply(list(dlmFiltered_obja, dlmFiltered_objb, dlmFiltered_objc),
105 + function(x){
106 +   # Kalman prediction
107 +   dlmForecasted_object <- dlmForecast(mod = x, nAhead = 12)
108 +   # Find the standard deviation and the 2.5% and 97.5% values of the prediction
109 +   value
110 +   f_sd <- sqrt(as.numeric(dlmForecasted_object$Q))
111 +   f_lower <- dlmForecasted_object$f + qnorm(0.025, sd = f_sd)
112 +   f_upper <- dlmForecasted_object$f + qnorm(0.975, sd = f_sd)
113 +
114 +   # Combine the results
115 +   return(ts.union(
116 +     mean = dlmForecasted_object$f,
117 +     lower = f_lower,
118 +     upper = f_upper
119 +   )))
120 >
121 > # Combine the prediction results for each model into ts class
122 > names(f_all) <- c("a", "b", "c")
123 > y_pred <- do.call("ts.union", f_all)
124 >
125 > # Extract 2015 data from the entire observation
126 > y_true <- window(y_all, start = 2015)
127 >
128 > # Ignore the display of following codes

```

Figure 9.9 shows the results from Code 9.6.

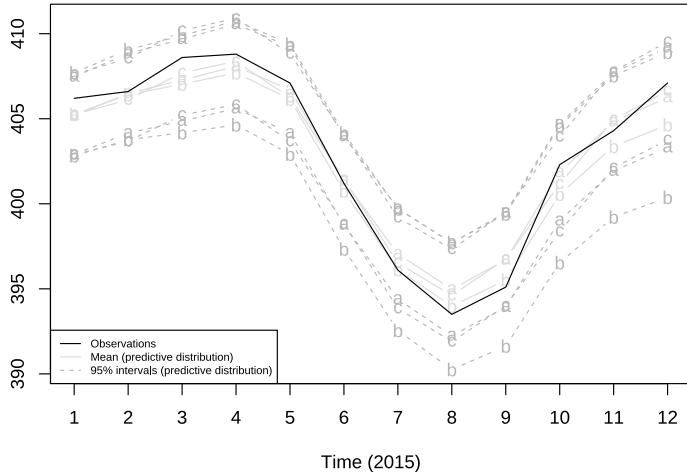


Fig. 9.9 Comparison of predictions from 2015 among three models

Seeing Fig. 9.9, it is clear that the probability intervals of model b) are greater than those of models a) and c), while the results of models a) and c) appear to be similar. Regarding the numerical index, we then confirm the MAPE described in Chap. 4. Code 9.7 compares MAPEs for 2015.

Code 9.7

```

129 > # <<Comparison of MAPEs from 2015 among three models>>
130 > MAPE(true = y_true, pred = y_pred[, "a.mean"])
131 [1] 0.001990842
132 > MAPE(true = y_true, pred = y_pred[, "b.mean"])
133 [1] 0.002313552
134 > MAPE(true = y_true, pred = y_pred[, "c.mean"])
135 [1] 0.001923007

```

We reuse here the user-defined function `MAPE()` created in Chap. 4. We see that the MAPE is in the same order as the result of the marginal likelihood, while the difference in the MAPE between models a) and c) is very small consistently with the impression from Fig. 9.9.

Incidentally, it is worth noting that the model a) uses eleven states to express seasonality, whereas the model c) uses only four states: the number eleven is derived from the “12-month period – constraint on the time domain period,” and the number four is also derived from “two artificially set residual frequency components × each real and imaginary part.” In other words, the model c) can achieve an equivalent (actually slightly better) MAPE while reducing the number of states in the model.

As can be seen, the appropriate use of the seasonal model in the frequency-domain approach can sometimes reduce computational complexity without degrading prediction performance.

9.5 ARMA Model

The *ARMA model*, already mentioned in Sect. 5.3, can be defined as an individual model in the state-space model. This section defines the ARMA(p, q) model as follows:

$$y_t = \sum_{j=1}^r \phi_j y_{t-j} + \sum_{j=1}^{r-1} \psi_j \epsilon_{t-j} + \epsilon_t, \quad (9.35)$$

where $r = \max\{p, q + 1\}$, $\phi_j = 0$ at $j > p$, $\psi_j = 0$ at $j > q$, and ϵ_t is white noise with variance σ^2 . The above definition is equivalent to Eq. (5.7), while the notation is somewhat different.

If the original time series data are nonstationary, we sometimes take the multiple-times difference of the data to achieve stationarity before applying the ARMA model. Such a model is called the *ARIMA model*. In the state-space model, we typically apply the polynomial and seasonal models to the nonstationary components; hence, this section focuses on the ARMA model. Regarding the representation of the ARIMA model in the state-space model, see [3, 11].

In the state-space model, it is common to treat the correlation between data by primarily combining the polynomial and seasonal models introduced thus far. However, with actual data, irregular fluctuations can remain even with such a combinational model for various reasons. For example, unexpected events may not be fully explained by such a model. In the state-space model, the ARMA model is effective in capturing and expressing such residual correlation. We typically use a low-order stationary AR model for this purpose.

We now consider the expression of Eq. (9.35) in the state-space model. While there are several expressions of ARMA model with state-space form, this book introduces one of them [3, 7, 11] known as the *observable canonical form* [1, 4]. The state and observation equations for the ARMA model correspond to Eqs. (5.8) and (5.9) with the following settings:

$$\mathbf{x}_t = \begin{bmatrix} x_t^{(1)} \\ x_t^{(2)} \\ \vdots \\ x_t^{(r-1)} \\ x_t^{(r)} \end{bmatrix}, \quad \mathbf{G}_t = \begin{bmatrix} \phi_1 & 1 & & \\ \phi_2 & & 1 & \\ \vdots & & \ddots & \\ \phi_{r-1} & & & 1 \\ \phi_r & 0 & 0 & \cdots & 0 \end{bmatrix}, \quad \mathbf{W}_t = \mathbf{R}\mathbf{R}^\top \sigma^2, \quad (9.36)$$

$$\mathbf{F}_t = [1, 0, \dots, 0, 0], \quad V_t = 0, \quad (9.37)$$

where $\mathbf{R} = [1, \psi_1, \dots, \psi_{r-2}, \psi_{r-1}]^\top$, and σ^2 is the variance of the white noise ϵ_t . Note that the above setting leads to $\mathbf{w}_t = \mathbf{R}\epsilon_t$.

It can be difficult to understand intuitively that the above definition becomes the ARMA(p, q) model in Eq. (9.35); hence, we complement the deformation of the expression. First, the state equation is expanded as

$$\begin{aligned} x_t^{(1)} &= \phi_1 x_{t-1}^{(1)} + x_{t-1}^{(2)} + \epsilon_t, \\ x_t^{(2)} &= \phi_2 x_{t-1}^{(1)} + x_{t-1}^{(3)} + \psi_1 \epsilon_t, \\ &\vdots \\ x_t^{(r-1)} &= \phi_{r-1} x_{t-1}^{(1)} + x_{t-1}^{(r)} + \psi_{r-2} \epsilon_t, \\ x_t^{(r)} &= \phi_r x_{t-1}^{(1)} + \psi_{r-1} \epsilon_t. \end{aligned}$$

We can obtain the following relationship by replacing $x_{t-1}^{(2)}$ in the first expression above with the variable derived from the second expression:

$$x_t^{(1)} = \phi_1 x_{t-1}^{(1)} + \phi_2 x_{t-2}^{(1)} + x_{t-2}^{(3)} + \psi_1 \epsilon_{t-1} + \epsilon_t.$$

Repeating such a substitution successively eventually yields

$$x_t^{(1)} = \phi_1 x_{t-1}^{(1)} + \cdots + \phi_r x_{t-r}^{(1)} + \psi_1 \epsilon_{t-1} + \cdots + \psi_{r-1} \epsilon_{t-(r-1)} + \epsilon_t.$$

The observation equation $y_t = x_t^{(1)}$ finally yields the following:

$$y_t = \phi_1 y_{t-1} + \cdots + \phi_r y_{t-r} + \psi_1 \epsilon_{t-1} + \cdots + \psi_{r-1} \epsilon_{t-(r-1)} + \epsilon_t.$$

Recalling $r = \max\{p, q+1\}$, we see that the above relationship becomes Eq. (9.35).

9.5.1 Example: Japanese Beer Production

Herein, we analyze the data on Japanese beer production. The original data are available from the brewers association of Japan's website <http://www.brewers.or.jp/data/doko-list.html>; this example treats the monthly series of Japanese internal taxable shipping volume (unit: (kl)). First, we examine the data preliminarily. We read the analyzing data from the file BEER.csv; the author has saved the data from January 2003 to December 2013 in advance. This code is as follows.

Code 9.8

```

1 > # <<Japanese beer production>>
2 >
3 > # Preprocessing
4 > library(dlm)
5 >
6 > # Load the data
7 > beer <- read.csv("BEER.csv")
8 >
9 > # Cast the data to ts class
10 > y <- ts(beer$Shipping_Volume, frequency = 12, start = c(2003, 1))
11 >
12 > # Plot
13 > plot(y)
14 >
15 > # Log-transform the data
16 > y <- log(y)
17 >
18 > # Plot log-transformed data
19 > plot(y, ylab = "log(y)")
```

The data are read from the file and assigned to y . Figure 9.10a shows the plotting result.

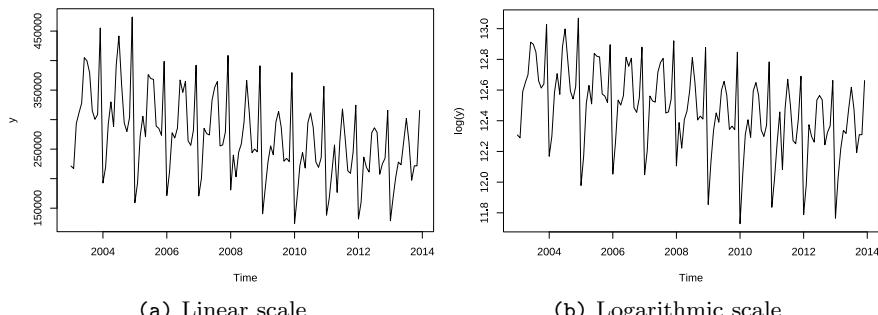


Fig. 9.10 Japanese beer production

According to Fig. 9.10a the range of data fluctuation appears to decrease with increasing time. It is known empirically that the characteristics of such economic data often show an exponential tendency. Thus, we decide to apply the logarithmic transformation typically used for preprocessing in economic time series analysis. Figure 9.10b shows the transformation result.

We also see the annual cycle and decreasing trend for the essential characteristics of the time series. The annual cycle arises from the beer demand, which tends to increase in the summer and at the end of the year [9]. In addition, the decreasing trend can be affected by young people's lack of interest in beer [8]. Moreover, we can expect various other factors, such as adult population, price revision, weather, tax, economy, competing products, and natural disasters.

Then, we consider the model to be applied. First, we model the annual cycle. The cycle has extremely sharp changes; such a change in the time domain corresponds to the high-frequency component in the frequency domain. Therefore, we can assume that the periodic pattern in these data has an essentially high-frequency component. Seasonality can be modeled using two approaches, i.e., the time and frequency domains. The frequency-domain approach appears not to have any advantage in this example because the data originally have high-frequency components that must not be truncated. Thus, we decide to use the seasonal model in the time-domain approach; its state elements are more intuitive and easier to understand. We also apply the local-trend model to the decreasing trend as a first step.

9.5.1.1 Local-Trend Model + Seasonal Model (Time-domain Approach)

Based on the above descriptions, we first combine the local-trend model and the seasonal model in the time-domain approach. We can achieve such a model through the following setting in component decomposition Eqs. (9.3) and (9.4):

- Eqs. (9.17) and (9.18) are applied to the first component model.
- Eqs. (9.25) and (9.26) are applied to the second component model.

This code is as follows.

Code 9.9

```

20 > # <<Japanese beer production: local-trend model + seasonal model (time-domain
21   approach)>>
22 >
23 > # Model setting: local-trend model + seasonal model (time-domain approach)
24 > build_dlm_BEERa <- function(par){
25 +   return(
26 +     dlmModPoly(order = 2, dW = exp(par[1:2]), dV = exp(par[3])) +
27 +     dlmModSeas(frequency = 12, dW = c(exp(par[4]), rep(0, times = 10)), dV = 0)
28 +   )
29 + }
30 >
31 > # Maximum likelihood estimation of parameters and confirmation of the results
32 > fit_dlm_BEERa <- dlmMLE(y = y, parm = rep(0, 4), build = build_dlm_BEERa)
33 > fit_dlm_BEERa
34 $par
35 [1] -1.248291 -2.552020 -2.054341 -3.564919
36
37 $value
38 [1] 126.3795
39
40 $counts
41 function gradient
42      5      5
43
44 $convergence
45 [1] 0
46
47 $message
48 [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
49
50 >
51 > # Set the maximum likelihood estimates of parameters in the model
52 > mod <- build_dlm_BEERa(fit_dlm_BEERa$par)
53 >
54 > # Kalman smoothing
55 > dlmSmoothed_obj <- dlmSmooth(y = y, mod = mod)
56 >
57 > # Mean of the smoothing distribution
58 > mu <- dropFirst(dlmSmoothed_obj$s[, 1])
59 > gamma <- dropFirst(dlmSmoothed_obj$s[, 3])
60 >
61 > # Plot results
62 > oldpar <- par(no.readonly = TRUE)
63 > par(mfrow = c(3, 1)); par(oma = c(2, 0, 0, 0)); par(mar = c(2, 4, 1, 1))
64 > ts.plot(    y, ylab = "Observations (log-transformed)")
65 > ts.plot(  mu, ylab = "Level component")
66 > ts.plot(gamma, ylab = "Seasonal component")
67 > mtext(text = "Time", side = 1, line = 1, outer = TRUE)
68 > par(oldpar)
69 >
70 > # Confirm the log-likelihood
71 > -dlmLL(y = y, mod = mod)
72 [1] -126.3795

```

The above code continues from the former Code 9.8. The user-defined function `build_dlm_BEERa()` is used to create the model. Subsequently, maximum likelihood estimation is performed on the parameters, and the result is assigned to `fit_dlm_BEERa`. We cannot see any particular problem in the result. We then set

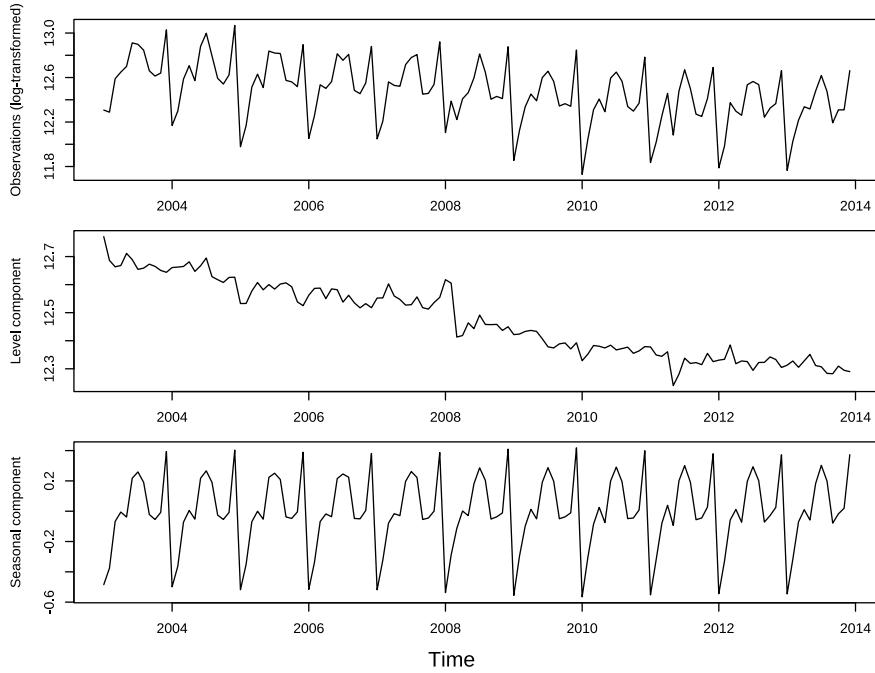


Fig. 9.11 Smoothing result by the local-trend model + seasonal model (time-domain approach)

the maximum likelihood estimates of parameters in the model to perform Kalman smoothing. Figure 9.11 shows the plotting results.

The above figure shows three plots: observations, level component (mean of the smoothing distribution), and seasonal component (mean of the smoothing distribution) in order from the top. Unlike filtering case, we can mitigate the degradation of estimation accuracy for the first period because this result is obtained by smoothing. We extract the seasonal component in reasonably consistent shape. On the contrary, the level component reflects the distortion in observations and appears to be quite jagged. Finally, we confirm the log-likelihood value, -126.3795 .

9.5.1.2 Local-Level Model + Seasonal Model (Time-Domain Approach)

In the previous examination, the level component was not as smooth as expected despite smoothing; hence, we now improve the results. For this purpose, an intentional reduction of the state noise about the level component might be useful. However, such a method has a poor theoretical background. Recall that the local-trend model can consider the short-term slope in the data. This model is originally inclined to follow the irregular increase and decrease of observations and may automatically track the distortion, which we would like to interpret separately from a smooth trend.

To mitigate such tracking, we simplify the model and apply the local-level model instead of the local-trend model. We achieve such a model through the following setting in the component decomposition Eqs. (9.3) and (9.4):

- Eqs. (9.7) and (9.8) are applied to the first component model.
- Eqs. (9.25) and (9.26) are applied to the second component model.

The shaded part indicates the difference between the above and first models. This code is as follows.

Code 9.10

```

72 > # <<Japanese beer production: local-level model + seasonal model (time-domain
73   approach)>>
74 >
75 > # Model setting: local-level model + seasonal model (time-domain approach)
76 > build_dlm_BEERb <- function(par){
77   +
78   +   return(
79   +     dlmModPoly(order = 1, dW = exp(par[1]), dV = exp(par[2])) +
80   +       dlmModSeas(frequency = 12, dW = c(exp(par[3]), rep(0, times = 10)), dV = 0)
81   +   )
82   +
83 >   +
84 >   # Ignore the display of following codes

```

The above code continues from the former Code 9.9. First, we prepare the user-defined function `build_dlm_BEERb()` for creating the model. Unlike the first user-defined function `build_dlm_BEERa()`, the local-level model is used instead of the local-trend model. The shaded part in the code indicates the difference between the above code and the first code. Subsequently, maximum likelihood estimation is performed on the parameters, and the result is assigned to `fit_dlm_BEERb`. We can see that the result has warnings starting with the phrase `Warning in dlmLL`. These warnings are due to the library `dlm`. When the estimates of the observation noise approach zero, the library intentionally adds a minimal value 10^{-6} to it. This is considered not to affect the results in this example, so we ignore them here. We cannot see any particular problem in other items such as convergence. We then set the maximum likelihood estimates of the parameters in the model to perform Kalman smoothing. Figure 9.12 shows the plotting results.

The figure shows three plots: observations, level component (mean of the smoothing distribution), and seasonal component (mean of the smoothing distribution) in order from the top. We extract the seasonal component in reasonably consistent shape. The jaggedness of the level component also appears to be mitigated considerably. Finally, we confirm the log-likelihood value, 106.1595. This value is greater than -126.3795 in the first “local-trend model + seasonal model (time-domain approach).” Hence, we understand that this model is more appropriate.

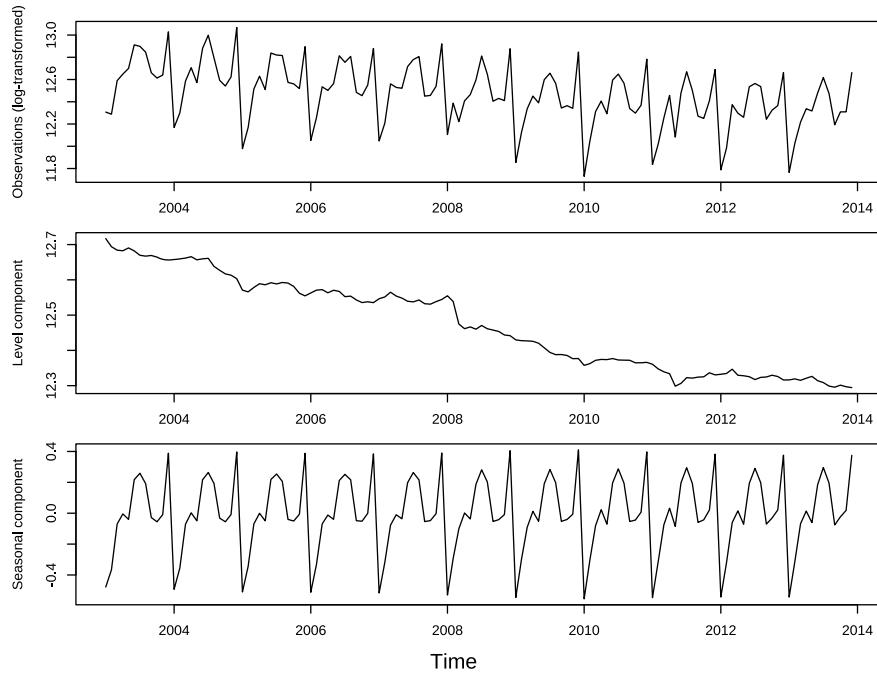


Fig. 9.12 Smoothing result by the local-level model + seasonal model (time-domain approach)

9.5.1.3 Local-Level Model + Seasonal Model (Time-Domain Approach) + ARMA Model

While the jaggedness of the level component has been mitigated through the previous examination, we try improving the result further by adding the ARMA model. The extent to which we must eliminate such jaggedness is ultimately a subjective judgment, but we can confirm the quality of the results using the marginal likelihood. The ARMA model captures the residual correlations due to various factors working together: adult population, price revision, weather, tax, economy, competing products, and natural disasters. For simplicity, we decide here to use the purest AR(1) model to capture the irregular deviation from the supposed smoothing level. We achieve such a model through the following setting in the component decomposition Eqs. (9.3) and (9.4):

- Eqs. (9.7) and (9.8) are applied to the first component model.
- Eqs. (9.25) and (9.26) are applied to the second component model.
- Eqs. (9.36) and (9.37) are applied to the third component model.

The shaded part indicates the difference between the above model and the second model. This code is as follows.

Code 9.11

```

83 > # <<Japanese beer production: considering AR(1) component>>
84 >
85 > # Model setting: local-level model + seasonal model (time-domain approach) + AR(1)
86 <-> model
87 > build_dlm_BEERc <- function(par){
88 +   return(
89 +     dlmModPoly(order = 1, dW = exp(par[1]), dV = exp(par[2])) +
90 +     dlmModSeas(frequency = 12, dW = c(exp(par[3]), rep(0, 10)), dV = 0) +
91 +     dlmModARMA(ar = ARtransPars(par[4]), sigma2 = exp(par[5]))
92 +   )
93 + }
94 >
95 > # Ignore the display of following codes

```

The above code continues from the former Code 9.10. First, we prepare the user-defined function `build_dlm_BEERc()` for creating the model. Unlike the second user-defined function `build_dlm_BEERb()`, the AR(1) model is added. The shaded part in the code indicates the difference between the above code and the second code. The function `dlmModARMA()` in the library **dlm** can set the AR(1) model. The arguments of `dlmModARMA()` are `ar`, `ma`, `sigma2`, `dV`, `m0`, and `C0`. The argument `ar` indicates the AR coefficient ϕ_j and must be specified preserving stationarity. For such a purpose, the library **dlm** provides a utility function `ARtransPars()` that approximates the AR coefficients [6]. This example uses the function; hence, the `ar` is set to `ARtransPars(par[4])` based on the argument `par` of `build_dlm_BEERc()`. The argument `ma` indicates the MA coefficient ψ_j . Because this example does not use it, the default value `NULL` is applied. The argument `sigma2` indicates the variance σ^2 of the white noise and is set to `exp(par[5])` based on the argument `par` of `build_dlm_BEERc()`. The argument `dV` indicates the variance V of the observation noise. Since this value has already been set through other component models, the setting is omitted; the default value, that is 0, is applied to this argument. The arguments `m0` and `C0` indicate the mean vector \mathbf{m}_0 and covariance \mathbf{C}_0 of the prior distribution, respectively. They are set to the default values, that is, $\mathbf{0}$ and $10^7\mathbf{I}$, respectively, because the setting is omitted; these values can be used in this example with no problem. Subsequently, maximum likelihood estimation is performed on the parameters, and the result is assigned to `fit_dlm_BEERc`. We see that the result has warnings. They are the same as that in the case of previous `fit_dlm_BEERb` and considered not to affect the results in this example; hence, we ignore them here. We cannot see any particular problem in other items such as convergence. Incidentally, we can confirm that the estimation result of AR(1) coefficient satisfies the stationary condition through the fact that `ARtransPars(fit_dlm_BEERc$par[4])` leads $| -4.907038 \times 10^{-5} | < 1$. We then set the maximum likelihood estimates of the parameters in the model to perform Kalman smoothing. Figure 9.13 shows the plotting results.

The figure shows four plots: observations, level component (mean of the smoothing distribution), seasonal component (mean of the smoothing distribution), and AR(1) component (mean of the smoothing distribution) in order from the top. We extract the seasonal component in a reasonably consistent shape. It is considered that

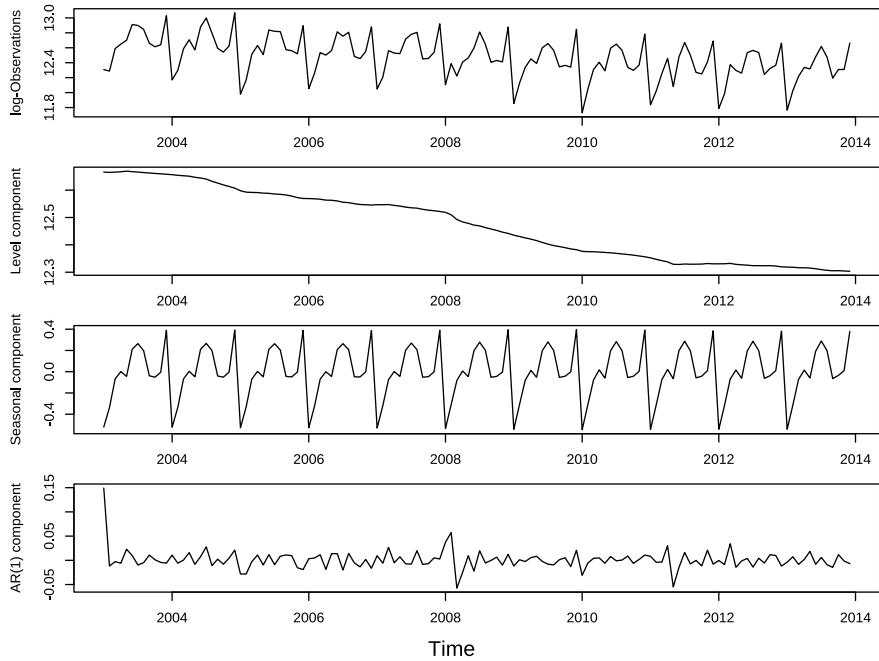


Fig. 9.13 Smoothing result by the local-level model + seasonal model (time-domain approach) + AR(1) model

the level component becomes almost smooth, while the AR(1) component reflects irregular distortion. Specifically, the significant distortion of the AR(1) component at the beginning of 2008 is hypothesized to be rush demand and its backlash, which are associated with the price rise implemented at that time. However, we cannot determine the causality easily; hence, this book stops pursuing variation factors further. Finally, we confirm the log-likelihood value, 152.6782. This value is greater than 106.1595 in the second “local-level model + seasonal model (time-domain approach).” Hence, we understand that this model is more appropriate. In this manner, the ARMA model can extract the total distortion that we want to distinguish from other components; it can also improve the likelihood of the model.

Incidentally, we complement the role of the AR model. In the above example, the AR(1) coefficient takes a small value. This result indicates that the ARMA model captures fine distortion. On the other hand, if the AR(1) coefficient takes a greater value, then the ARMA model can capture rough distortion. The ARMA model is often used for the latter purpose.

9.6 Regression Model

The *regression model* is used to enhance the explanation capability of the model by considering the relation with the event, which might be useful for describing time series data. We call the value quantifying such an event by some means an *explanatory variable*, such as temperature, population, or an indicator of 0/1 showing the occurrence or nonoccurrence of an event (this is also called an *intervention variable* [2]). The state-space model also considers the regression component as one of the individual models.

This section describes the *regression model* based on the following linear regression:

$$y_t = \alpha_t + \beta_t^{(1)} x_t^{(1)} + \cdots + \beta_t^{(n)} x_t^{(n)} + \cdots + \beta_t^{(N)} x_t^{(N)} + v_t, \quad (9.38)$$

where α_t is a *regression intercept*, $\beta_t^{(n)}$ is a *regression coefficient*, and $x_t^{(n)}$ is an explanatory variable. We assume that all of these variables can change over time. The regression intercept α_t and coefficient $\beta_t^{(n)}$ are stochastic and must be estimated, while the explanatory variable $x_t^{(n)}$ is deterministic. Incidentally, the setting $N = 1$ and $N > 1$ leads to *single regression* and *multiple regression*, respectively.

In this book, symbol x is assigned to the stochastic state variable in principle, which is different from the deterministic explanatory variable x' .

The state and observation equations for the linear regression model in Eq. (9.38) (with time-invariant state and observation noises) correspond to Eqs. (5.8) and (5.9) with the following settings:

$$\mathbf{x}_t = \begin{bmatrix} \alpha_t \\ \beta_t^{(1)} \\ \vdots \\ \beta_t^{(N)} \end{bmatrix}, \quad \mathbf{G}_t = \mathbf{I}, \quad \mathbf{W}_t = \begin{bmatrix} W^{(\alpha)} & & & \\ & W^{(1)} & & \\ & & \ddots & \\ & & & W^{(N)} \end{bmatrix}, \quad (9.39)$$

$$\mathbf{F}_t = [1, x_t^{(1)}, \dots, x_t^{(N)}], \quad V_t = V, \quad (9.40)$$

where it is assumed that the regression intercept α_t and coefficient $\beta_t^{(n)}$ are uncorrelated and follow an independent random walk. The above dynamic linear regression model estimates the time-varying regression intercept and coefficient. This is the advantage of this model and differs from standard static linear regression. The setting $\mathbf{W}_t = \mathbf{O}$ eliminates the temporal transition of the regression intercept and coefficient and renders the above model a standard static linear regression model.

9.6.1 Example: Nintendo's Stock Price

We now examine the beta value for Nintendo's stock price. The beta value is an index showing how the individual stock price changes aggressively compared to the market average. While a beta value of one indicates that the price change is the same as the market average, if the beta value is greater or less than one, the price change is more aggressive or conservative, respectively, than the market average. We proceed with the analysis based on a single regression model supposing that Nintendo's stock price, the Nikkei stock average, and the beta correspond to the observations, an explanatory variable, and a regression coefficient, respectively.

The stock price data are available from the Yahoo! JAPAN finance website <http://finance.yahoo.co.jp/>; we treat here the weekly closing price (unit: (yen)). The author has saved the data from October 2013 to November 2016 as NINTENDO.csv and NIKKEI225.csv, and we examine them preliminarily. This code is as follows.

Code 9.12

```

1 > # <<Nintendo's stock price>>
2 >
3 > # Preprocessing
4 > library(dlm)
5 >
6 > # Load the data
7 > NINTENDO <- read.csv("NINTENDO.csv")
8 > NINTENDO$Date <- as.Date(NINTENDO$Date)
9 >
10 > NIKKEI225 <- read.csv("NIKKEI225.csv")
11 > NIKKEI225$Date <- as.Date(NIKKEI225$Date)
12 >
13 > # Set observations and explanatory variables
14 > y      <- NINTENDO$Close
15 > x_dash <- NIKKEI225$Close
16 >
17 > # Ignore the display of following codes

```

First, the data regarding Nintendo's stock price and the Nikkei stock average are read from the files and assigned to the observations `y` and an explanatory variable `x_dash`, respectively. Figure 9.14 shows their plots.

According to Fig. 9.14, we see that both continue changing, but their situation depends on time. They are similar in 2014 and first half of 2016, whereas differ in 2015 and latter half of 2016.

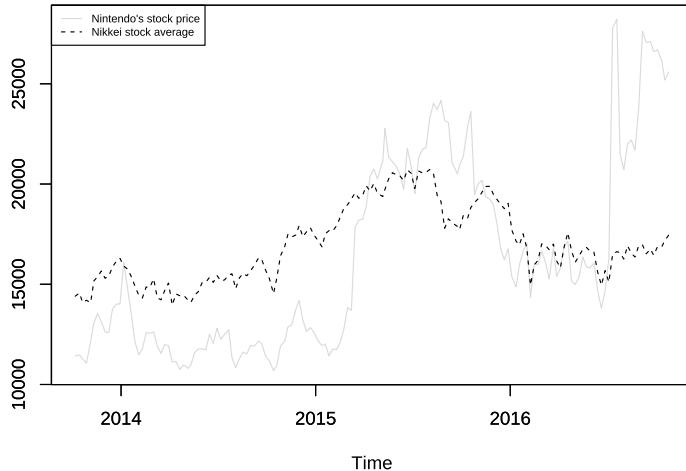


Fig. 9.14 Nintendo's stock price and the Nikkei stock average

We then analyze using the regression model. We can achieve such a model through the following setting in the component decomposition Eqs. (9.3) and (9.4):

- Eqs. (9.39) and (9.40) are applied to the first component model.

This code is as follows.

Code 9.13

```

18 > # <<Beta for Nintendo's stock price>>
19 >
20 > # Model setting: regression model
21 > build_dlm_REG <- function(par) {
22 +   dlmModReg(X = x_dash, dW = exp(par[1:2]), dV = exp(par[3]))
23 + }
24 >
25 > # Maximum likelihood estimation of parameters and confirmation of the results
26 > fit_dlm_REG <- dlmMLE(y = y, parm = rep(0, 3), build = build_dlm_REG)
27 > fit_dlm_REG
28 $par
29 [1] 5.607036e-07 -5.186503e+00 -8.297637e-07
30
31 $value
32 [1] 1233.212
33
34 $counts
35 function gradient
36     8          8
37
38 $convergence
39 [1] 0
40
41 $message
42 [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
43

```

```

44 >
45 > # Set the maximum likelihood estimates of parameters in the model
46 > mod <- build_dlm_REG(fit_dlm_REG$par)
47 > str(mod)
48 List of 11
49 $ m0 : num [1:2] 0 0
50 $ C0 : num [1:2, 1:2] 1e+07 0e+00 0e+00 1e+07
51 $ FF : num [1, 1:2] 1 1
52 $ V : num [1, 1] 1
53 $ GG : num [1:2, 1:2] 1 0 0 1
54 $ W : num [1:2, 1:2] 1 0 0 0.00559
55 $ JFF: num [1, 1:2] 0 1
56 $ JV : NULL
57 $ JGG: NULL
58 $ JW : NULL
59 $ X : num [1:160, 1] 14405 14562 14088 14202 14087 ...
60 - attr(*, "class")= chr "dlm"
61 >
62 > # Kalman smoothing
63 > dlmSmoothed_obj <- dlmSmooth(y = y, mod = mod)
64 >
65 > # Ignore the display of following codes

```

The above code continues from the former Code 9.12. First, we prepare the user-defined function `build_dlm_REG()` for creating the model. The function `dlmModReg()` in library **dlm** can set up the regression model. The arguments of `dlmModReg()` are `X`, `addInt`, `dW`, `dV`, `m0`, and `C0`. The argument `X` indicates a vector of explanatory variables $x_t^{(n)}$ and is set to `x_dash`. The argument `addInt` indicates whether we are to include the regression intercept, and the default value `TRUE` is applied to it because this example considers the intercept. The arguments `dW` and `dV` indicate the diagonal elements $W^{(x)}$, $W^{(1)}$, ..., $W^{(N)}$ in the covariance of the state noise and the variance V of the observation noise, respectively. We respectively set `exp(par[1:2])` and `exp(par[3])` to them based on the argument `par` of `build_dlm_REG()`. The arguments `m0` and `C0` indicate the mean vector \mathbf{m}_0 and covariance \mathbf{C}_0 of the prior distribution, respectively. These are set to the default values, that is, $\mathbf{0}$ and $10^7\mathbf{I}$, respectively, because the setting is omitted; these values can be used in this example with no problem. Subsequently, the maximum likelihood estimation is performed to parameters, and the result is assigned to `fit_dlm_REG`. We cannot see any particular problem in the result. We then set the maximum likelihood estimates of the parameters in the model.

Now, we confirm the contents of `mod` especially. We find the elements `$JFF` and `$X` that have not been set in the former models; see also the shaded code. These elements are used for the time-varying model, and we see that the regression model has a time-varying parameter F_t .

While this book has assumed that the parameters in the stochastic model are time-invariant until the last Chap. 12 in principle for simplicity, the regression model is an exception in having time-varying parameters. Appendix B summarizes the information regarding the setting of the time-varying model by the library `dlm`.

Next, we perform Kalman smoothing. Figure 9.15 shows the plotting result.

According to the plot, we see that the beta in this example looks basically like its stock price. The time points marked on the horizontal axis also indicate the following events related to Nintendo:

- March 17, 2015: Notice regarding business and capital alliance with DeNA Co., Ltd.
- July 6, 2016: Mobile game “Pokémon GO” launched in Australia, New Zealand, and the United States.

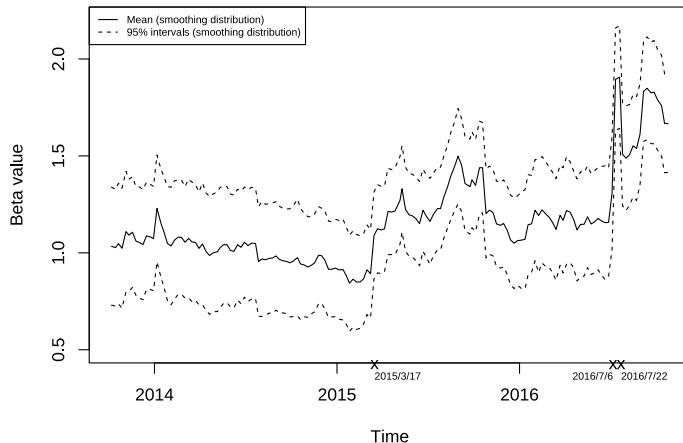


Fig. 9.15 Beta of Nintendo’s stock price (smoothing distribution)

- July 22, 2016: Notice regarding the impact of “Pokémon GO” on the consolidated financial forecast.

The above events appear to affect the beta. However, since we cannot determine the causality easily, this book stops pursuing the possible factors further.

9.6.2 Example: Flow Data of the Nile (Considering the Rapid Decrease in 1899)

We now reexamine the flow data of the Nile. While we have analyzed these data based on the state-space model in Sect. 8.2, we could not capture adequately the rapid decrease in 1899. Although a simple means such as data division might be sufficient for these data, we attempt here to extend the model to capture structural changes even without such a division. For this purpose, we use the regression model to consider the rapid decrease in 1899. Specifically, in the regression model, we set an explanatory variable to zero through 1899, that is, without a dam, and one after 1899, i.e., with a dam. An explanatory variable indicating the presence or absence of such an event is sometimes referred to as an intervention variable. We consider a model that contains such a regression component as well as the local-level component [10].

Incidentally, we can obtain an equivalent model by including the regression intercept in the regression model instead of the local-level model. However, this example explains the combined model with the local-level model to enable readers to become accustomed to a combination of models.

We can achieve such a model through the following setting in the component decomposition Eqs. (9.3) and (9.4):

- Eqs. (9.7) and (9.8) are applied to the first component model.
- Eqs. (9.39) and (9.40) are applied to the second component model.

This code is as follows.

Code 9.14

```

1 > # <<Apply local-level model + regression model (intervention variable) to flow data
2 > # of the Nile>>
3 > # Preprocessing
4 > set.seed(123)
5 > library(dlm)
6 >
7 > # Flow data of the Nile
8 > y <- Nile
9 > t_max <- length(y)
10 >
11 > # Set the explanatory variable (intervention variable)
12 > x_dash <- rep(0, t_max) # All initial value 0s (no dam)
13 > x_dash[which(1899 <= time(y))] <- 1 # All 1s after 1899 (with dam)
14 >
15 > # Function building local-level model + regression model (intervention variable)
16 > build_dlm_DAM <- function(par) {
17 +   return(
18 +     dlmModPoly(order = 1, dV = exp(par[1]), dW = exp(par[2])) +
19 +     dlmModReg(X = x_dash, addInt = FALSE, dW = exp(par[3]), dV = 0)
20 +   )
21 + }
22 >
23 > # Maximum likelihood estimation of parameters
24 > fit_dlm_DAM <- dlmMLE(y = y, parm = rep(0, 3), build = build_dlm_DAM)
25 > modtv <- build_dlm_DAM(fit_dlm_DAM$par)
26 >
27 > # Kalman smoothing
28 > dlmSmoothed_obj <- dlmSmooth(y = y, mod = modtv)
29 >
30 > # Mean and variance of the smoothing distribution
31 > stv <- dropFirst(dlmSmoothed_obj$s)
32 > stv_var <- dlmSvd2var(dlmSmoothed_obj$U.S, dlmSmoothed_obj$D.S)
33 > stv_var <- stv_var[-1]
34 >
35 > # Mean for estimator
36 > s <- stv[, 1] + x_dash * stv[, 2] # Consider also x_dash
37 >
38 > # 95% intervals of level estimator (finding 2.5% and 97.5% values)
39 > coeff <- cbind(1, x_dash)
40 > s_sdev <- sqrt(sapply(seq_along(stv_var), function(ct){ # Consider covariance
41 +   coeff[ct, ] %*% stv_var[[ct]] %*% t(coeff[ct, , drop = FALSE]
42 +   }))
43 > s_quant <- list(s + qnorm(0.025, sd = s_sdev), s + qnorm(0.975, sd = s_sdev))
44 >
45 > # Ignore the display of following codes

```

In the above code, we first set the explanatory variable `x_dash`. Then, we prepare the user-defined function `build_dlm_DAM()` for creating the model. The function `dlmModReg()` in the library `dlm` sets the regression model. The argument `X` of this function is set to `x_dash`. In addition, the regression intercept is not used in this example; hence, the argument `addInt` is set to `FALSE`. The argument `dW` is set to `exp(par[3])` based on the argument `par` of `build_dlm_DAM()`. Since the argument `dV` has already been set through the local-level model, we set `0` here. The arguments `m0` and `C0` are set to the default values, i.e., $\mathbf{0}$ and $10^7\mathbf{I}$, respectively, because the setting is omitted; these values can be used in this example.

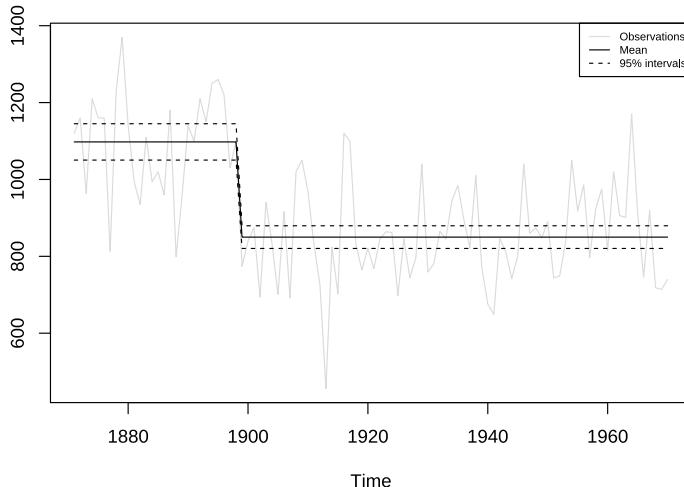


Fig. 9.16 Flow data of the Nile (considering the rapid decrease in 1899 as an intervention variable)

with no problem. Subsequently, maximum likelihood estimation is performed on the parameters, and the result is assigned to `fit_dlm_DAM`. We then perform Kalman smoothing based on this result. The estimator in this model is the sum of two random variables, that is, the level + regression components; hence, we calculate its mean and variance based on Eqs. (2.5) and (2.6). Figure 9.16 shows the plotting results.

According to Fig. 9.16, we see that the rapid decrease in 1899 can be captured adequately. We can also see that the estimates through and after 1899 together become more stable and closer to the constants.

As can be seen from the above description, we can consider the known change points adequately in terms of the regression component in the state-space model. Alternatively, there is another approach using the time-varying state noise for the same purpose; we will discuss that approach in Chap. 12. Incidentally, since it is difficult to analyze the unknown change, we require a general state-space model for such data; we will also discuss such an example in Chap. 12.

9.6.3 Example: Family Food Expenditure (Considering Effects Depending on the Days of the Week)

We now examine family food expenditures in Japan. The expenditure on food expenses is a continuous occurrence and affected by various factors such as prices and the economy. While it has aspects in common with beer production in Sect. 9.5.1

from the viewpoint of human ingestion data, it is significantly different in whether it has a life-threatening effect. In other words, if food is not eaten frequently, we die, but even if the beer is not consumed at all, we still stay alive. The data on food expenditure in Japan are available at the Japanese ministry of internal affairs and communications' website <http://www.stat.go.jp/data/kakei/2.htm>. These data come from a household economics survey conducted by a statistics bureau; this example uses the monthly food expenditure per two-or-more-person household (unit: (yen)). The values of these data do not contain expenses of eating out. First, we examine the data preliminarily. We read the analyzing data from the file FOOD.csv; the author has saved the data from January 2003 to December 2013 in advance. This code is as follows.

Code 9.15

```

1 > # <<Family expenditure (food)>>
2 >
3 > # Preprocessing
4 > library(dlm)
5 >
6 > # Load the data
7 > food <- read.csv("FOOD.csv")
8 >
9 > # Cast the data to ts class
10 > y <- ts(food$Expenditure, frequency = 12, start = c(2000, 1))
11 >
12 > # Plot
13 > plot(y)
14 >
15 > # Log-transform the data
16 > y <- log(y)
17 >
18 > # Plot log-transformed data
19 > plot(y, ylab = "log(y)")
```

The data are read from the file and assigned to y . Figure 9.17a shows the plotting result.

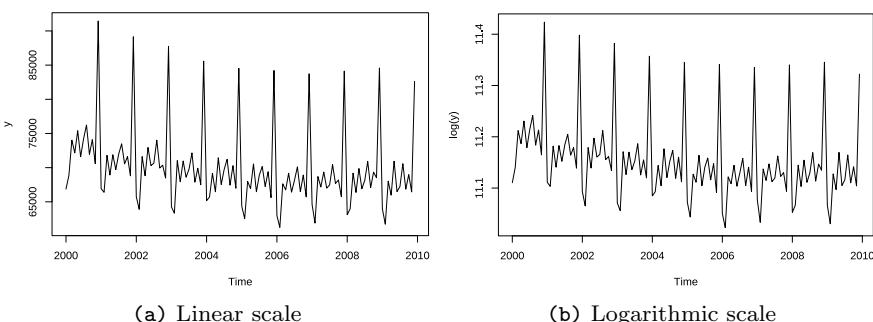


Fig. 9.17 Food expenditure

As shown in Fig. 9.17a, the level appears to be saturated after a moderate decreasing trend, except for the annual cycle. We also see that the annual cycle has the remarkable property of being the highest in December and the lowest in January or February. This property is the same as that of beer production; we can guess that both are influenced by Japanese culture. Although logarithmic transformation appears to be unnecessary according to the plot, we apply the transformation to the data in this example for consistency with the previous analysis by the statistics bureau of the Japanese ministry of internal affairs (http://www.stat.go.jp/data/kakei/longtime/pdf/rev_sa.pdf). Figure 9.17b shows the plot on the logarithm of data.

Regarding the model for these data, we first apply the local-trend model to the entire decreasing trend. As for the annual cycle, we apply the seasonal model with the time-domain approach considering the sharp change between the year-end and new year similarly to beer production. We can achieve such a model through the following setting in the component decomposition Eqs. (9.3) and (9.4):

- Eqs. (9.17) and (9.18) are applied to the first component model.
- Eqs. (9.25) and (9.26) are applied to the second component model.

Because the analyzing code based on the above model is very close to the Code 9.9, we omit its display. Figure 9.18 shows the smoothing result.

The figure shows three plots: observations, level component (mean of the smoothing distribution), and seasonal component (mean of the smoothing distribution) in order from the top. We confirm the log-likelihood value, 312.291.

While this result provides a reasonable impression, we focus on the influence of the day of the week for further improvement. If the food expenditure did not change depending on the day of the week, we would not have to take such an effect into account. However, if the food expenditure considerably changes depending on the day of the week, we should consider such an effect because the total number of “day of the week” in one month varies according to the month, e.g., the number of Mondays may vary in different months. Since the detailed data including daily food expenditure have been investigated in recent years, we preliminarily check the daily data of June 2009 obtained from the mentioned cite <http://www.stat.go.jp/data/kakei/2.htm>. Figure 9.19 shows the plot of the data.

According to Fig. 9.19, we find that the values on Saturdays and Sundays are significantly higher than those on other days. If we regard Saturdays and Sundays as typical holidays, we can suppose that food is inclined to be purchased on holidays intensively. Based on this assumption, we can expect that the national holidays also have the same inclination, while there are no national holidays in June 2009.

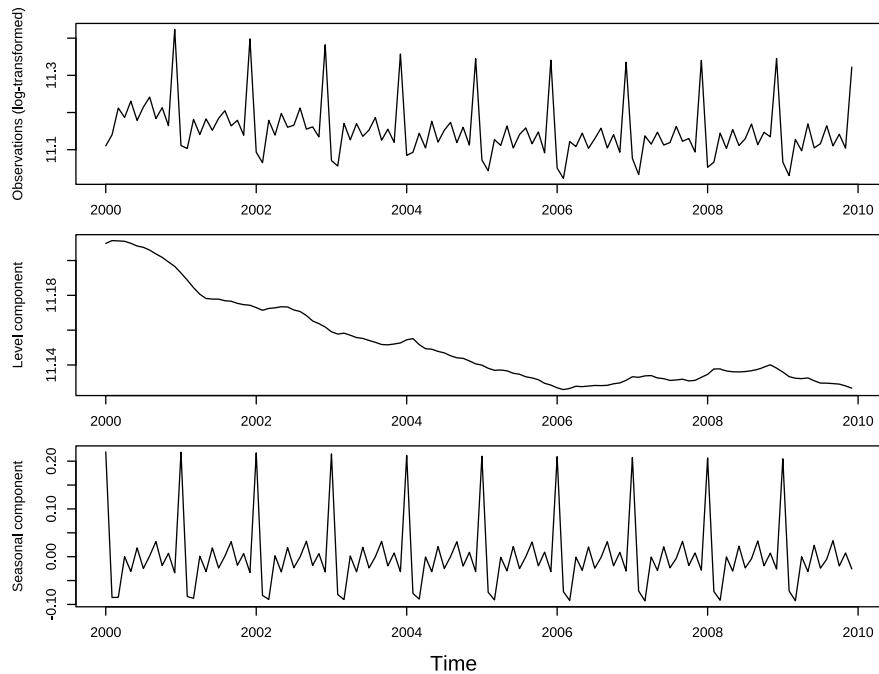


Fig. 9.18 Food expenditure analysis with the local-trend model + seasonal model (time-domain approach)

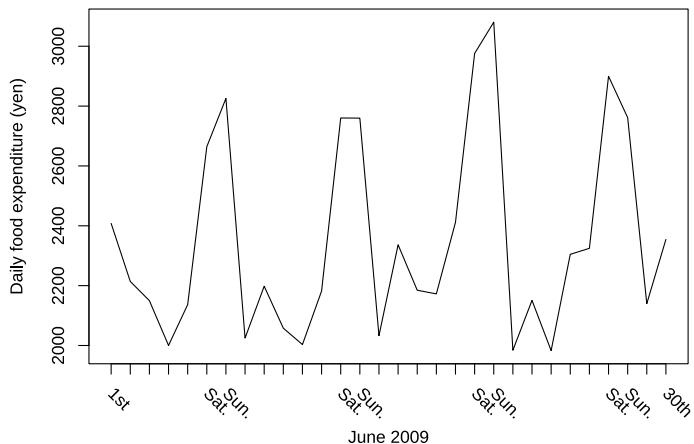


Fig. 9.19 Daily food expenditure of June 2009

The above influence is called a calendar effect, and the state-space model can take its impact into account through the regression component, whose explanatory variable is set to the deterministic calendar information. While there are various means for the setting of such information, for monthly data, an explanatory variable is typically set to the monthly total number for each day of the week. This is often referred to as a trading-day effect [7, 13]. While we can apply such primary trading-day effect to this issue straightforwardly, the effect has several variations; hence, we now examine one more suitable to this example. Figure 9.19 implies that the distinction between weekdays and holidays such as Saturdays, Sundays, and national holidays mostly has a significant impact on food expenditure. Thus, this example considers the following regression component whose explanatory variables are set to the monthly total number for the weekdays and holidays:

$$\begin{aligned} \text{trading-day effect} = & \beta^{(1)} \times \text{monthly total number for the weekdays} \\ & + \\ & \beta^{(2)} \times \text{monthly total number for the holidays}. \end{aligned} \quad (9.41)$$

We assume time-invariant regression coefficients in the above effect because time-invariant coefficients result in static regression, which is easily interpreted and typically used. Furthermore, we assume that the sum of regression coefficients is zero to ensure the unique time series decomposition. Specifically, this example supposes the following constraint:

$$\beta^{(1)} + \beta^{(2)} = 0. \quad (9.42)$$

In accordance with this constraint, we reduce Eq. (9.41) as follows:

$$\begin{aligned} \text{trading-day effect} = & \beta^{(1)} \times \text{monthly total number for weekdays} \\ & + \\ & \beta^{(2)} \times \text{monthly total number for holidays} \\ = & \beta^{(1)} \times (\text{monthly total number for weekdays} \\ & - \text{monthly total number for holidays}). \end{aligned} \quad (9.43)$$

Thus, the trading-day effect in this example has an explanatory variable, “the number of weekdays more than holidays in one month.” Hereafter, we call this effect a “weekday effect.” The code setting the explanatory variable for this weekday effect is as follows.

Code 9.16

```

20 > # <<Set explanatory variable (weekday effect)>>
21 >
22 > # User-defined function returning weekdays and holidays in Japan
23 > jholidays <- function(days){
24 +   # Use of is.jholiday()
25 +   library(Nippon)
26 +
27 +   # Obtain the day of the week
28 +   DOW <- weekdays(days)
29 +
30 +   # Consider Saturdays, Sundays, and other public holidays, including their
31 +   ← compensations, as holidays
32 +   holidays <- (DOW %in% c("Saturday", "Sunday")) | is.jholiday(days)
33 +
34 +   # Overwrite the day of the week with "HOLIDAY" or "WEEKDAY"
35 +   DOW[holidays] <- "Holiday"
36 +   DOW[!holidays] <- "Weekday"
37 +
38 +   return(DOW)
39 +
40 > # Sequence of the date during the examination period
41 > days <- seq(from = as.Date("2000/1/1"), to = as.Date("2009/12/31"), by = "day")
42 >
43 > # Aggregate the number of weekdays or holidays for every month
44 > monthly <- table(substr(days, start = 1, stop = 7), jholidays(days))
45 >
46 > # Explanatory variable (difference between the total number of weekdays and
47 > ← holidays in a month)
47 > x_dash_weekday <- monthly[, "Weekday"] - monthly[, "Holiday"]

```

The above code continues from the former Code 9.15. The code first prepares the user-defined function `jholidays()`, which returns the judgment result on weekday or holiday in Japan using the library **Nippon**. We then use this function to calculate the explanatory variable for the weekday effect and set the result to `x_dash_weekday`.

Incidentally, we notice that the examination period in this example includes leap years such as 2000, 2004, and 2008. We must consider the influence of February 29 as a calendar effect. The regression component can also consider the influence of a leap year. Specifically, the effect can be taken into account by setting the explanatory variable to one and zero in February of a leap and otherwise, respectively. Hereafter, we call this effect the “leap-year effect.” We assume the leap-year effect is also a static regression for facilitating our interpretation, as in the case of the weekday effect. The code setting the explanatory variable for the leap-year effect is as follows.

Code 9.17

```

48 > # <<Set explanatory variable (leap year effect)>>
49 >
50 > # Data length
51 > t_max <- length(y)
52 >
53 > # February in a leap year during the examination period
54 > LEAPYEAR_FEB <- (c(2000, 2004, 2008) - 2000)*12 + 2
55 >
56 > # Explanatory variable (February in a leap year only corresponds to 1)
57 > x_dash_leapyear <- rep(0, t_max)           # All initial value 0s
58 > x_dash_leapyear[LEAPYEAR_FEB] <- 1          # February in leap year corresponds to 1

```

The above code continues from the former Code 9.16. The code sets the explanatory variable for the leap-year effect to `x_dash_leapyear`.

We add the regression model described above to the first discussed model, that is, the local-trend model + seasonal model (time-domain approach). Here, we simplify the first model in advance, because we expect that the regression model is to capture fine fluctuations residing in the observations. Specifically, we replace the local-trend model by the local-level model and set the state noise of the seasonal model in the time-domain approach to zero. We can achieve such a model through the following setting in component decomposition Eqs. (9.3) and (9.4):

- Eqs. (9.7) and (9.8) are applied to the first component model.
- Eqs. (9.25) and (9.26) are applied to the second component model (however, the state noise is equal to zero).
- Eqs. (9.39) and (9.40) are applied to the third component model.

The shaded part indicates the difference between the above model and the first model. The analyzing code based on the above model is as follows.

Code 9.18

```

59 > # <<Food expenditure analysis with local-level model + seasonal model (time-domain
60   <- approach) + regression model>>
61 >
62 > # Bind explanatory variables (weekday and leap year effects)
63 > x_dash <- cbind(x_dash_weekday, x_dash_leapyear)
64 >
65 > # Function building local-level model + seasonal model (time-domain approach) +
66   <- regression model
67 > build_dlm_FOODb <- function(par) {
68   +   return(
69   +     dlmModPoly(order = 1, dW = exp(par[1]), dV = exp(par[2])) +
70   +     dlmModSeas(frequency = 12, dW = c(0, rep(0, times = 10)), dV = 0) +
71   +     dlmModReg(X = x_dash, addInt = FALSE, dV = 0)
72   +   )
73 }

```

```

72 >
73 > # Maximum likelihood estimation of parameters
74 > fit_dlm_FOODb <- dlmMLE(y = y, parm = rep(0, 2), build = build_dlm_FOODb)
75 >
76 > # Set the maximum likelihood estimates of parameters in the model
77 > mod <- build_dlm_FOODb(fit_dlm_FOODb$par)
78 > -dlmLL(y = y, mod = mod)
79 [1] 319.8928
80 >
81 > # Kalman filtering
82 > dlmSmoothed_obj <- dlmSmooth(y = y, mod = mod)
83 >
84 > # Mean of the smoothing distribution
85 > mu <- dropFirst(dlmSmoothed_obj$s[, 1])
86 > gamma <- dropFirst(dlmSmoothed_obj$s[, 3])
87 > beta_w <- dropFirst(dlmSmoothed_obj$s[, 13])[t_max] # Time-invariant
88 > beta_l <- dropFirst(dlmSmoothed_obj$s[, 14])[t_max] # Time-invariant
89 >
90 > # Confirmation of the results
91 > cat(beta_w, beta_l, "\n")
92 -0.003098287 0.03920194
93 >
94 > # Mean of regression component
95 > reg <- x_dash %*% c(beta_w, beta_l)
96 > tsp(reg) <- tsp(y)
97 >
98 > # Plot results
99 > oldpar <- par(no.readonly = TRUE)
100 > par(mfrow = c(4, 1)); par(oma = c(2, 0, 0, 0)); par(mar = c(2, 4, 1, 1))
101 > ts.plot( y, ylab = "log-Observations")
102 > ts.plot( mu, ylab = "Level component")
103 > ts.plot(gamma, ylab = "Seasonal component")
104 > ts.plot( reg, ylab = "Regression component")
105 > mtext(text = "Time", side = 1, line = 1, outer = TRUE)
106 > par(oldpar)

```

The above code continues from the former Code 9.17. We first unite two explanatory variables set up thus far, that is, the weekday effect and the leap-year effect, into one `x_dash`. Then, we prepare the user-defined function `build_dlm_FOODb()` for creating the model. Comparing to the first model which led Fig. 9.18 (however, we omitted the code display), we see that the local-trend model is replaced by the local-level model, and the regression model is added. The shaded part indicates the difference between the above code and the previous code. Figure 9.20 shows the results with the above code.

The figure shows four plots: observations, level component (mean of the smoothing distribution), seasonal component (mean of the smoothing distribution), and regression component (mean of the smoothing distribution) in order from the top. We find that the regression component grows in February of the leap year. We also find that the regression component can absorb the impact of February 29 comparing the level component with that in Fig. 9.18. Regarding the regression coefficients, we can obtain the rational values consistent with our assumption: the coefficient `beta_w` of the weekday effect takes the negative value -0.003098287 , which indicates that the expenditure on weekdays decreases compared with that on holidays, while the coefficient `beta_l` of the leap-year effect takes the value 0.03920194 , approximately $1/30$, which indicates one day in a month. The marginal log-likelihood

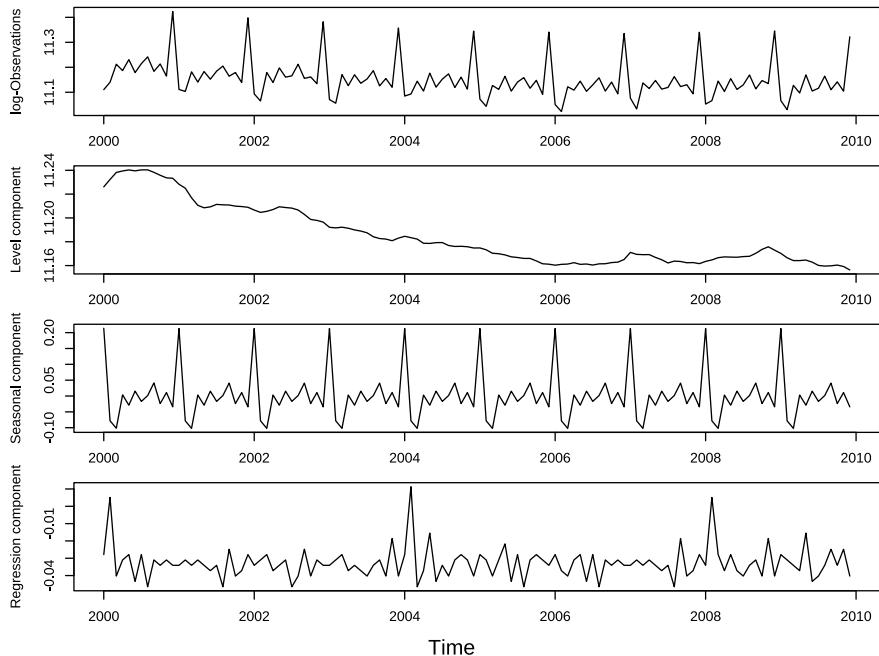


Fig. 9.20 Food expenditure analysis with the local-level model + seasonal model (time-domain approach) + regression model

value 319.8928 is somewhat improved from the result obtained using the first model. However, according to the above figure, the model appears to have room for further improvement. For example, according to the previous analysis by the statistics bureau of the Japanese ministry of internal affairs and communications (http://www.stat.go.jp/data/kakei/longtime/pdf/rev_sa.pdf), the food expenditure appears to be affected by other kinds of expenditure depending on the day of the week on the last day in the month. However, for simplicity, this book stops pursuing such effects further.

9.7 Supplement to Modeling

This chapter has described individual models typically used in the linear Gaussian state-space model. In addition to those, various models such as the time-varying coefficient AR model [7] and models for multivariate time series [11] are available; this book omits further explanation. The nonlinear model generally has no typical type, in contrast to the linear models explained thus far; hence, we must examine the nonlinear modeling carefully according to the individual problem.

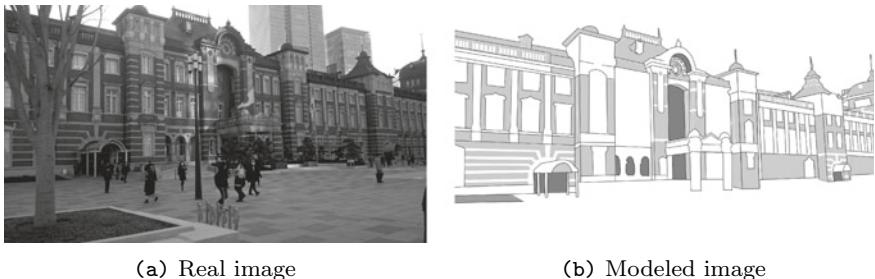


Fig. 9.21 The real and model

For modeling an actual problem, we typically combine partial models. As described through the examples in this chapter, while the likelihood becomes a guide to such combination, we must examine the problem according to individual issues; we cannot avoid trial and error. Because there is no panacea for modeling at this moment, the author supplements the discussion with a personal opinion on basic guidelines for modeling.

First, before modeling, it is important to perform sufficient preliminary examination of data. Such an examination is a preliminary analysis and can be regarded as a type of *exploratory data analysis* [12, 14]. In the author's opinion, the data analysis such as time series analysis is similar to cooking. The preliminary examination of data corresponds to preparing food material and establishing a rough cooking direction; it is an essential first step in deciding the quality of cooking.

Next, as mentioned in Sect. 4.3, when targeting real data, we can never know the correct model except in special cases. The author recommends that the reader should pragmatically regard the model as just a decision to make it easier to interpret the characteristics of data.

Finally, we must improve the model in a step by step manner; it is safe to start with a simple one at the beginning. Furthermore, it is essential to keep the model as concise as possible. Simplicity is the key to enhancing our understanding and improving the accuracy of results. In the author's opinion, modeling is similar to the cartoon shown in Fig. 9.21. The author believes that unnecessary elements must be removed to clarify a point in question and highlight its essence.

References

1. Adachi, S., Maruta, I.: Fundamentals of Kalman Filter. Tokyo Denki University Press (2012). [in Japanese]
2. Commandeur, J.J., Koopman, S.J.: An Introduction to State Space Time Series Analysis. Oxford University Press (2007)
3. Durbin, J., Koopman, S.J.: Time Series Analysis by State Space Methods, 2nd edn. Oxford University Press (2012)

4. Hirota, K., Ikoma, N.: Mathematics of Stochastic Process. Asakura Publishing Co., Ltd (2001). [in Japanese]
5. Iba, Y., Tanemura, M., Omori, Y., Wago, H., Sato, S., Takahashi, A.: Computational Statistics II: Markov Chain Monte Carlo Method and Related Topics. Iwanami Shoten Publishers, Tokyo, Japan (2005). [in Japanese]
6. Jones, M.C.: Randomly choosing parameters from the stationarity and invertibility region of autoregressive-moving average models. *J. Royal Statist. Soc. Series C (Appl. Statist.)* **36**(2), 134–138 (1987)
7. Kitagawa, G.: Introduction to Time Series Modeling. CRC Press (2009)
8. M1/F1 Research Institute: Verification of young people’s “lack of interest in beer”. Analysis report vol. 22 (2009). [in Japanese] (http://m1f1.jp/wp-content/uploads/2013/03/topic_090728.pdf) is already dead link as of 2018-04-28)
9. Noujuu, M.: Beer demand forecast and seasonal variation. *Manage. Sci. Operat. Res.* **43**(8), 426–430 (1998). [in Japanese]
10. Petris, G., Petrone, S.: State space models in *R. J. Statist. Softw. Art.* **41**(4), 1–25 (2011)
11. Petris, G., Petrone, S., Campagnoli, P.: Dynamic Linear Model with R. Springer (2009)
12. Shibata, R.: Data Assay and Data Science. Kindai Kagaku Sha Co., Ltd (2015). [in Japanese]
13. Takaoka, M.: Economic Time Series and Seasonal Adjustment Method. Asakura Publishing Co., Ltd (2015). [in Japanese]
14. Tukey, J.W.: Exploratory Data Analysis. Addison-Wesley (1977)
15. Wickham, H.: Advanced R. CRC Press (2014)
16. Yamaguchi, R., Tsuchiya, E., Higuchi, T.: Factorization of restaurant sales using state space model. *Manage. Sci. Operat. Res.* **49**(5), 52–60 (2004). [in Japanese]

Chapter 10

Batch Solution for General State-Space Model



This chapter describes the batch estimation method for the case wherein a specific number of observations already exist in the general state-space model. This solution uses the Markov chain Monte Carlo (MCMC) method. Recently, the excellent MCMC libraries have become widespread; this book focuses on the explanation using the software **Stan**. In addition, regarding the technique for improving estimation accuracy, we describe the forward filtering backward sampling (FFBS) algorithm, which uses the Kalman filter as a part of the solution.

10.1 MCMC

We have described the fundamental formulation for the estimation of the general state-space model in Chap. 6. In case of the linear Gaussian state-space model, we can derive its analytical solution “fortunately;” the details have been explained in Chaps. 7 and 8. On the contrary, in case of the general state-space model, we cannot derive its analytical solution, except for rare cases. Conventionally, in an age without rich computing power, a device such as the *conjugate prior distribution*, which is in the same distribution family as the posterior distribution, has often been used to make the analytical calculation possible. On the contrary, currently, when computing power is abundant, the approximate numerical solutions are typically applied. Obtaining the distribution of interest numerically, we must overcome the calculation of integrals as we see from the filtering distribution in Eq. (6.18), predictive distribution in Eq. (6.22), and (marginal) smoothing distribution in Eq. (6.24). There are several approaches for this purpose.

An approach using numerical integration known as a grid filter has been confirmed to be a practical method when the state dimension is not high [11, 12, 24].

There is also an approach using random numbers. References [4, 16] have proposed an approach using *importance sampling*.

Importance sampling draws a sample from a surrogate distribution enabling easy sampling, that is, *proposal distribution*. We then derive the statistics regarding the distribution of interest using these samples with correction in accordance with the likelihood of the distribution of interest.

The basic idea of this approach is that the proposal distribution for the general state-space model is approximated using the linear Gaussian state-space model. The Laplace approximation [1] is used for this purpose, and Kalman filtering and smoothing are used for sampling from the proposal distribution and calculation of the statistics. R library **KFAS** implements such an approach and can estimate the smoothing distribution alone under the condition wherein the observations have the particular exponential-family distribution when all the states are given [7, 16].

As a further approach using random numbers, *Markov chain Monte Carlo (MCMC)* method with high versatility has been used commonly in recent years; hence, this book proceeds with an explanation of the approach using MCMC.

MCMC generates a random number from the distribution of interest through an exploration primarily around the high-density region of the distribution based on the Markov chain framework. The basic mechanism of the algorithm will be reviewed shortly.

In the approach using MCMC, we obtain samples directly from the distribution of interest, such as the filtering, predictive, and smoothing distributions, in the general state-space model. Among them, the treatment of smoothing is the most natural and straightforward. When deriving the filtering and predictive distributions directly using the MCMC approach, we must restart the MCMC process from the beginning whenever data are obtained sequentially, with the result that the computational efficiency is poor. For this reason, it is common to use the sequential solution for filtering and prediction in the general state-space model. The next chapter describes such a solution in detail.

Based on the above explanation, this chapter focuses on “smoothing for a particular number of observations using MCMC.”

10.1.1 MCMC Fundamentals

Herein, we denote the probability distribution of interest as $p(\theta)$ and explain some specific methods for obtaining samples from it using MCMC [1, 8, 13, 14, 17].

First, we describe the most basic *Metropolis method*. The algorithm is as follows:

Algorithm 10.1 Metropolis method

0. Initialization: prepare $\theta^{(0)}$.
 1. for $i = 1, \dots, I$:
 - a. Set as $\theta = \theta^{(i-1)}$. Then, draw a proposal, $\tilde{\theta}$, from $q(\cdot | \theta)$.
 - b. Calculate the acceptance ratio, $\alpha = \frac{p(\tilde{\theta})}{p(\theta)}$.
 - c. If $\alpha \geq 1$, then accept the proposal $\tilde{\theta}$ by setting $\theta^{(i)} = \tilde{\theta}$.
 - d. If $\alpha < 1$, then accept the proposal $\tilde{\theta}$ with a probability proportional to α :
 - by setting $\theta^{(i)} = \tilde{\theta}$ in case of acceptance or
 - by setting $\theta^{(i)} = \theta^{(i-1)}$ in case of rejection.
 2. $\{\theta^{(i)}\} (i = 1, \dots, I)$ is a sample from $p(\theta)$.
-

In this chapter, I denotes the total number of iterations in the calculation of a Markov chain and $\cdot^{(i)}$ is the realization of a variable at a particular iteration i .

The Metropolis method uses a proposal distribution similar to importance sampling. Regarding the proposal distribution, $q(\cdot | \theta)$, we assume that the new proposal parameter $\tilde{\theta}$ can be drawn from it with the given current parameter, θ . For example, suppose a normal distribution whose mean¹ is set to the current parameter, θ , as $q(\cdot | \theta)$. In this case, sampling from $q(\cdot | \theta)$ implies a random walk, which is a type of Markov chain. Additionally, the algorithm of the Metropolis method is not deterministic, but is instead randomized based on the acceptance ratio of α . This algorithm is a type of Monte Carlo method. The Metropolis method is a Monte Carlo method that has a randomized algorithm based on the Markov chain such as a random walk. Therefore, it is an MCMC.

We then confirm the behavior of the Metropolis method using an example in which the distribution of interest is a one-dimensional standard normal distribution. Figure 10.1 shows the image of the trajectory.

From Fig. 10.1, we find some typical features of the MCMC. First, a certain number of steps are required for the proposal to start from the initial value and wander around in high density. Although these early-period samples are from the distribution of interest, they tend to be concentrated in low-density areas. Thus, it is safer to not actively use them as representative samples for the distribution of interest. In practice, they are usually discarded as unsatisfactory samples during a burn-in (warmup) period. Additionally, there is a correlation between proposals: a proposal at a particular step takes a value similar to the step before. Thinning is a well-known technique for such situations. It periodically discards samples to reduce the correlation between samples. Note that we must adjust the degree of burn-in and thinning, depending on the problem.

Although this example selects a random walk as $q(\cdot | \theta)$, determining its step size is a difficult problem. For example, if the step size is too large, the acceptance ratio

¹ The variance can be any value, but it should be tuned according to the problem for efficient sampling.

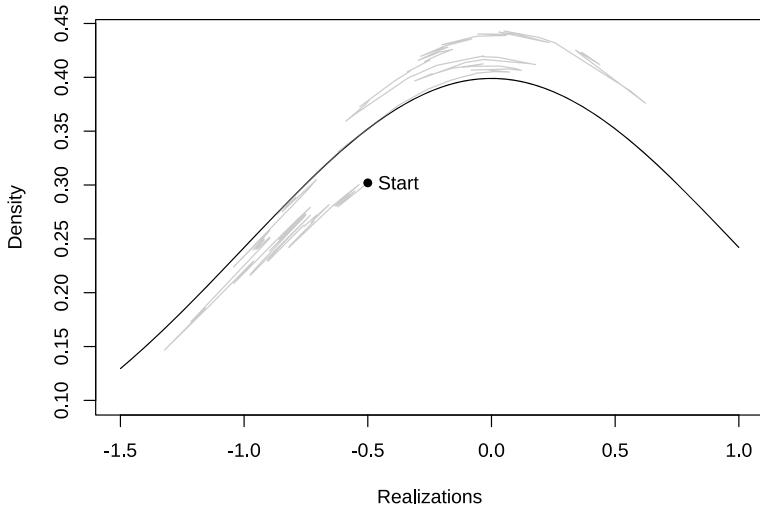


Fig. 10.1 Drawing a sample from one-dimensional standard normal distribution using the Metropolis method

typically decreases, resulting in more computation waste. On the contrary, if the step size is too small, the acceptance ratio typically increases. However, only a minuscule portion of the distribution will be searched. Many extensions have been proposed to avoid this dilemma and improve the efficiency of the MCMC methods. The Hybrid (Hamiltonian) Monte Carlo method [3] is one such extension, which introduces the concept of the Hamiltonian from physics and can obtain useful samples without decreasing the acceptance ratio.

Next, we explain the *Metropolis–Hastings (M–H) method*, which is an extension of the Metropolis method. In the previous description of the Metropolis method, we implicitly assumed that the density of the proposal distribution was symmetric as with

$$q(\tilde{\boldsymbol{\theta}} \mid \boldsymbol{\theta}) = q(\boldsymbol{\theta} \mid \tilde{\boldsymbol{\theta}}).$$

The above condition holds in the random walk example. However, it does not always hold. The M–H method is an extension of this point. The algorithm for this method is given by Algorithm 10.2.

Comparing the Algorithm 10.2 with the Algorithm 10.1, we see that the calculation of the acceptance ratio has been changed to incorporate the asymmetry of the proposal density.

Finally, we discuss the *Gibbs method*, which is a type of M–H method. In the literature, the Gibbs method is often referred to as Gibbs sampling or the Gibbs sampler. The Gibbs method sets the proposal distribution of the M–H method to a *full conditional distribution*. This distribution is one wherein every random variable in the model is given except for one type. For example, let the parameters be $\boldsymbol{\theta} = \{\theta_1, \dots, \theta_k\}$. Then, its full conditional distribution is expressed as

$$q(\cdot \mid \boldsymbol{\theta}) = p(\cdot \mid \boldsymbol{\theta}_{\text{except for } k}).$$

Algorithm 10.2 M–H method

-
0. Initialization: prepare $\theta^{(0)}$.
 1. for $i = 1, \dots, I$:
 - a. Set as $\theta = \theta^{(i-1)}$. Then, draw a proposal, $\tilde{\theta}$, from $q(\cdot | \theta)$.
 - b. Calculate the acceptance ratio, $\alpha = \frac{p(\tilde{\theta})q(\theta | \tilde{\theta})}{p(\theta)q(\tilde{\theta} | \theta)}$.
 - c. If $\alpha \geq 1$, then accept the proposal $\tilde{\theta}$ by setting $\theta^{(i)} = \tilde{\theta}$.
 - d. If $\alpha < 1$, then accept the proposal $\tilde{\theta}$ with a probability proportional to α :
 - by setting $\theta^{(i)} = \tilde{\theta}$ in case of acceptance or
 - by setting $\theta^{(i)} = \theta^{(i-1)}$ in case of rejection.
 2. $\{\theta^{(i)}\}$ ($i = 1, \dots, I$) is a sample from $p(\theta)$.
-

Thus, only $\tilde{\theta}_k$ is drawn and updated for a certain iteration step. Note that the remaining parameters are not changed during this iteration step, resulting in $\tilde{\theta}_{\text{except for } k} = \theta_{\text{except for } k}$.

The acceptance ratio in the Gibbs method is as follows:

$$\begin{aligned}\alpha &= \frac{p(\tilde{\theta})q(\theta | \tilde{\theta})}{p(\theta)q(\tilde{\theta} | \theta)} \\ &= \frac{p(\tilde{\theta})p(\theta_k | \tilde{\theta}_{\text{except for } k})}{p(\theta)p(\tilde{\theta}_k | \theta_{\text{except for } k})} \\ &= \frac{p(\tilde{\theta}_k, \tilde{\theta}_{\text{except for } k})p(\theta_k | \tilde{\theta}_{\text{except for } k})}{p(\theta_k, \theta_{\text{except for } k})p(\tilde{\theta}_k | \theta_{\text{except for } k})}\end{aligned}$$

from Bayes' theorem

$$= \frac{p(\tilde{\theta}_k | \tilde{\theta}_{\text{except for } k})p(\tilde{\theta}_{\text{except for } k})p(\theta_k | \tilde{\theta}_{\text{except for } k})}{p(\theta_k | \theta_{\text{except for } k})p(\theta_{\text{except for } k})p(\tilde{\theta}_k | \theta_{\text{except for } k})}$$

$$\begin{aligned}\text{recall } \tilde{\theta}_{\text{except for } k} &= \theta_{\text{except for } k} \\ &= 1.\end{aligned}$$

Therefore, the Gibbs method always accepts the proposal. However, this is not necessarily linked to adequate sampling.

We then confirm the behavior of the Gibbs method with an example of a two-dimensional standard normal distribution with a covariance of zero. Figure 10.2 shows the situation.

From Fig. 10.2, we see a mechanism of the Gibbs method. Every full conditional distribution for a two-dimensional normal distribution becomes a one-dimensional normal distribution. Thus, if we consider the cross-section of a two-dimensional normal distribution cut based on the realization obtained at an iteration step, a full conditional distribution can be observed. The Gibbs method continues to obtain a new sample from such a cross-sectional distribution.

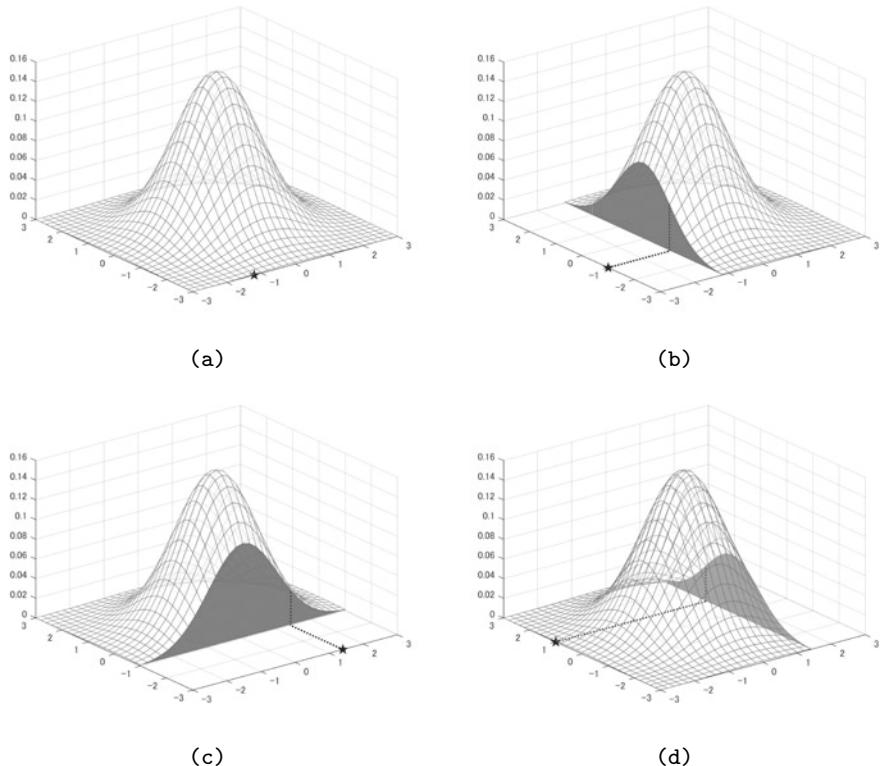


Fig. 10.2 Drawing a sample from two-dimensional standard normal distribution using the Gibbs method

10.1.2 Notes on Using the MCMC Method

Herein, we supplement the precautions in the use of MCMC. Since the output of MCMC is a random number, we might encounter difficulty in judging success or failure immediately. Thus, at least, we must always check the convergence of the Markov chain before using the result. While there are various means of confirming convergence, visual inspection of the multiple Markov chains is useful. For such inspection, we first draw a figure known as a *trace plot* with the number of searching steps on the horizontal axis and realizations of the sample on the vertical axis. We then confirm convergence by inspecting the mixing of multiple Markov chains in this trace plot. Figure 10.3 shows such examples.

Each panel of Fig. 10.3 plots three Markov chains. Figure 10.3a is a trace plot with good convergence. When the Markov chain convergence is good, similar sample values rarely continue; hence, the fluctuation of the trace plot becomes intense. This property ensures that various regions of the distribution of interest are being explored as the searching step progresses. Thus, overlaying the multiple Markov chains over

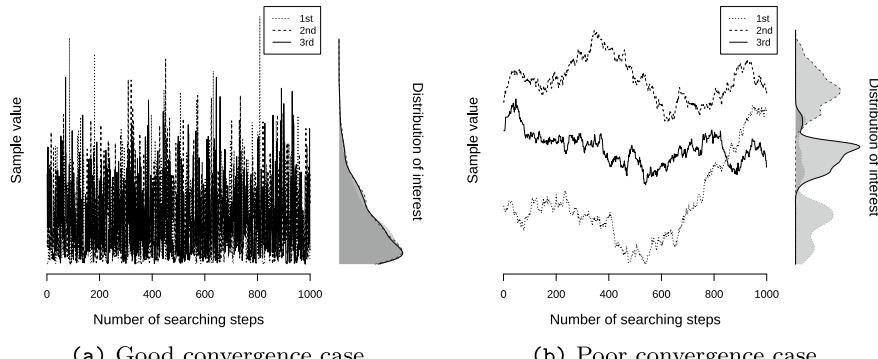


Fig. 10.3 Trace plots of Markov chains

the same trace plot leads to more mixing. On the contrary, Fig. 10.3b is a trace plot with poor convergence. When the Markov chain convergence is insufficient, similar sample values often continue with the result that the fluctuation of the trace plot becomes flat. This property ensures that only narrow regions of the distribution of interest are being explored even as the searching step progresses. Thus, overlaying the multiple Markov chains over the same trace plot leads to less mixing.

10.2 State Estimation with MCMC

In smoothing using MCMC, we usually consider the joint posterior distribution $p(\mathbf{x}_{0:T} | y_{1:T})$ as an estimation target. Furthermore, when the parameter is regarded as a random variable, the joint posterior distribution becomes $p(\mathbf{x}_{0:T}, \boldsymbol{\theta} | y_{1:T})$. This section treats the joint posterior distribution $p(\mathbf{x}_{0:T}, \boldsymbol{\theta} | y_{1:T})$ as an estimation target.

We now explain how to draw samples from a joint posterior distribution using MCMC. While there are various MCMC algorithms, we herein describe an example assuming the Gibbs method [18]. The Gibbs method repeats sampling from the full conditional distribution. The following is an algorithm:

Algorithm 10.3 Gibbs method

0. Initialization: prepare $\theta^{(0)}$.
 1. for $i = 1, \dots, I$:
 - a. Draw $x_{0:T}^{(i)}$ from $p(x_{0:T} | \theta = \theta^{(i-1)}, y_{1:T})$.
 - b. Draw $\theta^{(i)}$ from $p(\theta | x_{0:T} = x_{0:T}^{(i)}, y_{1:T})$.
 2. $\{x_{0:T}^{(i)}, \theta^{(i)}\}$ ($i = 1, \dots, I$) is a sample from $p(x_{0:T}, \theta | y_{1:T})$.

Regarding 1-a in the above algorithm, although $p(\mathbf{x}_{0:T} \mid \boldsymbol{\theta} = \boldsymbol{\theta}^{(i-1)}, \mathbf{y}_{1:T})$ is generally an arbitrary distribution, especially when the distribution follows the linear Gaussian state-space model, we can draw the sample more efficiently using *forward filtering backward sampling (FFBS)*. As the name suggests, this method performs Kalman filtering once in the time forward direction and then draws a sample based on Kalman smoothing in the time reverse direction. It is sometimes referred to as *simulation smoothing* because it corresponds to a simulation version of Kalman smoothing. The specific FFBS algorithm is as follows; see Appendix E.3 for its detailed derivation:

Algorithm 10.4 FFBS algorithm

0. Perform Kalman filtering with the parameter $\boldsymbol{\theta}^{(i-1)}$.
 1. Draw \mathbf{x}_T from $\mathcal{N}(\mathbf{m}_T, \mathbf{C}_T)$.
 2. for $t = T - 1, \dots, 0$:
 - *smoothing gain*
 $\mathbf{A}_t \leftarrow \mathbf{C}_t \mathbf{G}_{t+1}^\top \mathbf{R}_{t+1}^{-1}$
 - Draw \mathbf{x}_t from $\mathcal{N}(\mathbf{h}_t, \mathbf{H}_t)$,
 where $\mathbf{h}_t \leftarrow \mathbf{m}_t + \mathbf{A}_t [\mathbf{x}_{t+1} - \mathbf{a}_{t+1}]$ and
 $\mathbf{H}_t \leftarrow \mathbf{C}_t + \mathbf{A}_t [\mathbf{O} - \mathbf{R}_{t+1}] \mathbf{A}_t^\top$.
 3. $\{\mathbf{x}_t\} (t = T, \dots, 0)$ is a sample $\mathbf{x}_{0:T}^{(i)}$ from $p(\mathbf{x}_{0:T} \mid \boldsymbol{\theta} = \boldsymbol{\theta}^{(i-1)}, \mathbf{y}_{1:T})$.
-

Comparing FFBS and Kalman smoothing Algorithms (10.4 and 8.3) enables a deeper understanding of FFBS as a simulation version of Kalman smoothing.

If we scratch the code, we write the above algorithms into specific code. Herein, we reconsider the features of such a scratching approach again. While the scratching approach has the advantage of extensibility, its implementation is not always easy. Especially for the Gibbs method, we must pay attention to the setting of the distribution in the modeling according to the problem to ensure that we can easily draw a sample from the full conditional distribution. Such a scheme reduces modeling flexibility and can be difficult with a complex model (except for statisticians).

In contrast to the scratching approach, there is an approach that utilizes libraries. Although the advantages and disadvantages of the approach utilizing libraries are opposite to those of the scratching approach, general-purpose libraries have become readily available in recent years. In addition, the use of excellent MCMC libraries is becoming more widespread; hence this book focuses on the approach that utilizes libraries.

10.3 Use of Library

10.3.1 Various Libraries

R provides a variety of MCMC libraries, which we can classify roughly into standalone libraries in R and external software used in conjunction with R. The author has confirmed some of these and briefly introduces their features below.

The libraries **dlm** and **bsts** [5, 20, 25] are R standalone libraries; these consider the state-space model explicitly. The library **dlm** prepares functions that jointly estimate the state, state noise, and observation noise for the univariate linear Gaussian state-space model. The user must write code individually for setting the model and analyzing the results. In these functions, the Gibbs method with FFBS is implemented using the R language. The library **bsts** can jointly estimate the state and various parameters for typical univariate state-space models that are mainly focused on the linear Gaussian one. This library prepares abundant functions for adding individual models and is structured to be easy to use on the whole. It cooperates internally with the Bayesian estimation library **Boom** written in C++, and FFBS is also implemented. Incidentally, even though **Prophet** [19] is also an R/Python library, this is a type of easy-to-use wrapper for **Stan** which is described below.

On the contrary, external software working in conjunction with R includes **WinBUGS**, **OpenBUGS**, **JAGS**, **Stan**, and **NIMBLE**. Although they have some differences, we can generally regard them as programming languages suitable for statistical modeling. The softwares **WinBUGS**, **OpenBUGS**, and **JAGS** use the BUGS language notation, whereas **Stan** and **NIMBLE** use a similar notation. While the implementation of each software has its own characteristics [9], we have not described their details.

This book uses external software in conjunction with R as libraries considering an ability to customize the contents of processing. While the choice of library might depend on preference, this book uses **Stan** [15, 21, 26] considering sampling efficiency and flexibility.

The software **Stan** implements a hybrid (Hamiltonian) Monte Carlo method as its MCMC algorithm, and its sampling efficiency generally seems superior than that of the other software. In addition, **Stan** has full functionality as a programming language.

Incidentally, while this chapter does not treat **NIMBLE**, Appendix F mentions it as one of the libraries that implement the particle filter.

10.3.2 Example: Artificial Local-Level Model

Before estimating the general state-space model using **Stan**, we first explain its basic notation and behavior. For this purpose, we perform smoothing for the linear Gaussian state-space model with known parameters using **Stan** and then compare the results with those obtained using Kalman smoothing. For the model and data, we use the artificial local-level model and data that have been prepared using Code 9.1 in Sect. 9.2. Since the parameters are known in this example, smoothing for the state-space model using MCMC corresponds to drawing samples from the joint posterior distribution $p(\mathbf{x}_{0:T} | y_{1:T})$. This code for **Stan** is as follows.

Code 10.1

```

1 // model10-1.stan
2 // Model: specification (local-level model with known parameters)
3
4 data{
5   int<lower=1> t_max;    // Time series length
6   vector[t_max] y;        // Observations
7
8   cov_matrix[1] W;        // Variance of state noise
9   cov_matrix[1] V;        // Variance of observation noise
10  real m0;                // Mean of prior distribution
11  cov_matrix[1] C0;       // Variance of prior distribution
12 }
13
14 parameters{
15   real x0;                // State [0]
16   vector[t_max] x;        // State [1:t_max]
17 }
18
19 model{
20   // Likelihood part
21   /* Observation equation; see also equation (5.11) */
22   for (t in 1:t_max){
23     y[t] ~ normal(x[t], sqrt(V[1, 1]));
24   }
25
26   // Prior part
27   /* Prior distribution for state */
28   x0 ~ normal(m0, sqrt(C0[1, 1]));
29
30   /* State equation; see also equation (5.10) */
31   x[1] ~ normal(x0, sqrt(W[1, 1]));
32   for (t in 2:t_max){
33     x[t] ~ normal(x[t-1], sqrt(W[1, 1]));
34   }
35 }
```

The naming of variables used in the code is consistent with those used in the explanation thus far. The code description in **Stan** is divided into multiple blocks, which are explained in order. The first `data` block sets the constants passed from R. This example sets the time series length `t_max`, data `y`, variance of the state noise `W`, variance of the observation noise `V`, mean `m0` of the prior distribution, and variance `C0` of the prior distribution. The next `parameters` block declares the

variables to be sampled. This example declares the state x_0 for the prior distribution and state \mathbf{x} . The subsequent model block is the core part of **Stan** and describes the posterior distribution. Specifically, we transform the posterior distribution of interest using Bayes' theorem and express it separately in the likelihood and prior distribution parts. For example, let $p(x | y)$ be the posterior distribution, it leads to $p(x | y) \propto p(y | x)p(x)$; hence, we write $p(y | x)$ in the likelihood part and $p(x)$ in the prior distribution part. Since the posterior distribution of interest in this example is the joint posterior distribution $p(\mathbf{x}_{0:T} | y_{1:T})$, the specific transformation using Bayes' theorem is as follows:

$$\begin{aligned} p(\mathbf{x}_{0:T} | y_{1:T}) &= p(\mathbf{x}_{0:T}, y_{1:T}) / p(y_{1:T}) \\ &\propto p(\mathbf{x}_{0:T}, y_{1:T}) = \underbrace{p(y_{1:T} | \mathbf{x}_{0:T})}_{\text{likelihood part}} \underbrace{p(\mathbf{x}_{0:T})}_{\text{prior distribution part}} \end{aligned}$$

from Eq. (5.6)

$$\begin{aligned} &= p(\mathbf{x}_0) \prod_{t=1}^T p(y_t | \mathbf{x}_t) p(\mathbf{x}_t | \mathbf{x}_{t-1}) \\ &= \underbrace{\prod_{t=1}^T p(y_t | \mathbf{x}_t)}_{\text{likelihood part}} \underbrace{\prod_{t=1}^T p(\mathbf{x}_t | \mathbf{x}_{t-1}) p(\mathbf{x}_0)}_{\text{prior distribution part}}. \end{aligned} \quad (10.1)$$

Thus, the likelihood part $p(y_{1:T} | \mathbf{x}_{0:T})$ corresponds to the cumulative product $\prod_{t=1}^T p(y_t | \mathbf{x}_t)$ of the observation equation, and the prior distribution part $p(\mathbf{x}_{0:T})$ corresponds to the cumulative product $\prod_{t=1}^T p(\mathbf{x}_t | \mathbf{x}_{t-1})$ of the state equation times the prior distribution $p(\mathbf{x}_0)$. We write each of these terms based on the probability distribution representation of the observation and state equations in Eqs. (5.11) and (5.10) and the prior distribution specification for the state. When writing the code, we can use the easily understood notation “~,” as in the BUGS language. In addition, it is possible to ignore the code description on $p(\mathbf{x}_0)$ and $p(\mathbf{x}_1 | \mathbf{x}_0)$ in the prior distribution part and to apply the *noninformative prior distribution* (sufficiently wide uniform distribution) to them through the default setting in **Stan** [9, 15]. However, this book writes code faithfully according to the definition of the posterior distribution. While it is further possible to utilize **Stan**'s vectorization feature positively and to write code without `for` [15, 21], this book focuses on the standard code description.

Next, the R code for executing Code 10.1 is as follows.

Code 10.2

```

1 > # <<Smoothing for local-level model using MCMC (known parameters)>>
2 >
3 > # Preprocessing
4 > set.seed(123)
5 > library(rstan)
6 >
7 > # Presetting of Stan: HDD storage of compiled code and parallel computation
8 > rstan_options(auto_write = TRUE)
9 > options(mc.cores = parallel::detectCores())
10 >
11 > # Load data on artificial local-level model
12 > load(file = "ArtificialLocalLevelModel.RData")
13 >
14 > # Model: generation and compilation
15 > stan_mod_out <- stan_model(file = "model10-1.stan")
16 >
17 > # Smoothing: execution (sampling)
18 > fit_stan <- sampling(object = stan_mod_out,
19 +                         data = list(t_max = t_max, y = y,
20 +                                     W = mod$W, V = mod$V,
21 +                                     m0 = mod$m0, C0 = mod$C0),
22 +                         pars = c("x"),
23 +                         seed = 123
24 +                     )
25 >
26 > # Confirmation of the results
27 > oldpar <- par(no.readonly = TRUE); options(max.print = 99999)
28 > fit_stan
29 Inference for Stan model: model10-1.
30 4 chains, each with iter=2000; warmup=1000; thin=1;
31 post-warmup draws per chain=1000, total post-warmup draws=4000.
32
33      mean se_mean    sd   2.5%   25%   50%   75%   97.5% n_eff Rhat
34 x[1]    11.76     0.01  0.95   9.91  11.12  11.76  12.40   13.63  4237     1
35 x[2]    12.82     0.01  0.84  11.20  12.26  12.82  13.36   14.50  4212     1
36
37 ... Ignore the interim display ...
38
39 x[199]   18.70     0.01  0.85  17.06  18.12  18.70  19.27   20.36  4818     1
40 x[200]   19.35     0.01  0.99  17.45  18.66  19.36  20.01   21.27  5121     1
41 lp__ -197.74    0.25 10.02 -218.44 -204.34 -197.44 -190.69 -179.59  1572     1
42
43 Samples were drawn using NUTS(diag_e) at Sat Apr 25 16:00:34 2020.
44 For each parameter, n_eff is a crude measure of effective sample size,
45 and Rhat is the potential scale reduction factor on split chains (at
46 convergence, Rhat=1).
47 > par(oldpar)
48 > tmp_tp <- traceplot(fit_stan, pars = c(sprintf("%d", 100), "lp__"), alpha = 0.5)
49 > tmp_tp + theme(aspect.ratio = 3/4)
50 >
51 > # Extract necessary sampling results
52 > stan_mcmc_out <- rstan::extract(fit_stan, pars = "x")
53 > str(stan_mcmc_out)
54 List of 1
55 $ x: num [1:4000, 1:200] 13 12.4 11.3 10.5 11.3 ...
56 ..- attr(*, "dimnames")=List of 2
57 ... . $ iterations: NULL
58 ... . $ : NULL
59 >
60 > # Calculate the mean, 2.5%, and 97.5% values while marginalizing
61 > s_mcmc <- colMeans(stan_mcmc_out$x)
62 > s_mcmc_quant <- apply(stan_mcmc_out$x, 2, FUN = quantile, probs=c(0.025, 0.975))
63 >
64 > # Ignore the display of following codes

```

In the above code, we first load the library **rstan** to link R and **Stan**. After various presettings of **Stan**, we read the data on an artificial local-level model prepared with Code 9.1. The description thus far is preparation; the following specifically describes the cooperation with **Stan**. First, we read Code 10.1 using the function `stan_model()`, compile it, and save the result in `stan_mod_out`. Then, the function `sampling()` performs sampling, and the result is assigned to `fit_stan`. Regarding the function `sampling()`, the arguments `object`, `data`, `pars`, and `seed` are set to a compiled object, data passed to **Stan**, variables to be included in the return value, and random seed ensuring reproducibility, respectively. The other arguments, such as the numbers of Markov chains or iterations, are set to default values. The returned object `fit_stan` includes information other than sampling results. We can confirm their summary information through the console display of `fit_stan` such as some statistics regarding both sampled variables and log posterior probability density (`lp__`) used inside **Stan**. Explicitly, we can confirm the mean, standard error, standard deviation, quantile, effective sample size, and \hat{R} by rows. The last two statistics are the indices for measuring the convergence of the Markov chain. The *effective sample size* in MCMC indicates the sample size hypothesizing no correlation among samples, and if it is extremely small compared with the actual sample size, some concern remains in the convergence of the Markov chain. The actual sample size in this example is $4 \times 2,000/2 = 4,000$ from four Markov chains and 2,000 iterations per chain; the half iteration is discarded as the warm-up period in the default setting. Thus, the estimated effective sample size is about 4,000 at the maximum², and we can recognize that the result in this example has no particular problem. In addition, \hat{R} is a value based on the ratio of the sample variance within Markov chain to that between Markov chain, and if this value is not close to 1, some concern remains in the convergence of the Markov chain.

Let W and B be the within- and between-Markov chain sequence variances, respectively; [6] defines \hat{R} as

$$\sqrt{1 + \frac{B/W - 1}{\text{sample size of one Markov chain sequence}}}.$$

This book regards the condition $\hat{R} < 1.1$ as a convergence guideline of the Markov chain based on [6]. We can also recognize that the \hat{R} in this example has no particular problem. Regarding the `fit_stan`, we can further confirm the trajectory of the sample, i.e., a trace plot, using the function `traceplot()`. Figure 10.4 shows the results of the trace plot.

² The estimated effective sample size can be larger than the actual one because autocorrelation can be estimated negative [21].

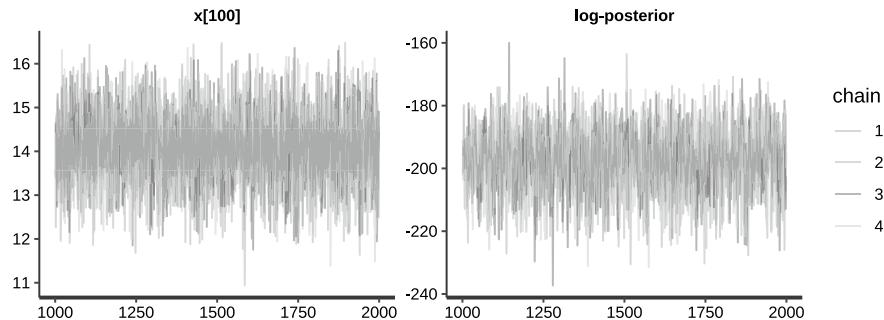


Fig. 10.4 Trace plots (linear Gaussian state-space model with known parameters)

As a typical example, we select two of them, namely state and log posterior probability density at time point 100 and depict them. The trace plot's horizontal and vertical axes show the number of iterations in the Markov chain and value of the sample, respectively. As mentioned in the beginning of this chapter, the insufficient mixing of multiple Markov chains in a trace plot indicates that some concern remains in the convergence of the chain. While the black and white plot may be somewhat difficult to recognize, there are no particular problems in the selected trace plots of this example. Next, we apply **rstan**'s function `extract()` for the object `fit_stan` to extract only the required sampling result and assign the result to the `stan_mcmc_out`. Checking the `stan_mcmc_out`, we see that it is a list whose element `$x` stores the sampling result. The element `$x` is a matrix whose rows and columns indicate the iteration number and time, respectively.

Next, we compare this result of smoothing using MCMC with that obtained by Kalman smoothing assuming all the known parameters. While the smoothing result using **Stan** is a sample from the joint posterior distribution, result obtained using Kalman smoothing is a statistic of the marginal posterior distribution. Thus, we must obtain the required statistics by marginalizing the sample of the joint posterior distribution for the comparison. The marginalization of a sample from the joint posterior distribution is not difficult: we can achieve the marginalization at a particular time point t' by considering only the sample at time point t' while ignoring samples at other time points. The statistics previously confirmed on the console display of `fit_stan` are also marginalized. In this example, we recalculate the statistics of the marginal posterior distribution from `stan_mcmc_out` and set the mean and 2.5 and 97.5% values to `s_mcmc` and `s_mcmc_sdev`, respectively. Figure 10.5 shows the comparison plots between the results from smoothing using MCMC and Kalman smoothing.

As shown in Fig. 10.5, we see that the two results are almost identical; the graph lines overlap and cannot be distinguished from each other.

Through the above simple example, we have confirmed that the **Stan** can accurately estimate the linear Gaussian state-space model.

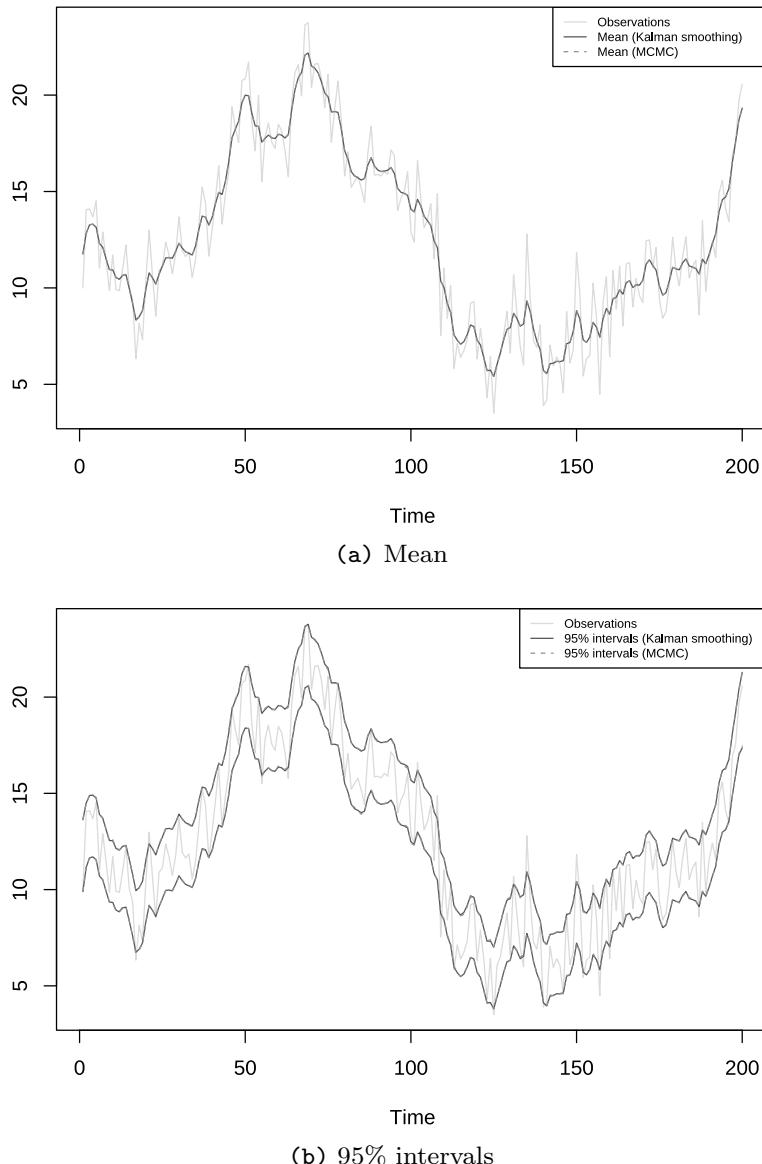


Fig. 10.5 Smoothing using MCMC and Kalman smoothing (linear Gaussian state-space model with known parameters)

10.4 Estimation Example in General State-Space Model

The example in the previous section was involved the linear Gaussian state-space model with known parameters; we can actually solve such a problem using a Kalman filter. We now consider a general state-space model as a linear Gaussian state-space model with unknown parameters treated as random variables. As already mentioned in Sect. 6.4.2, even if the original state-space model is linear Gaussian, we must solve the augmented state-space model whose state contains unknown parameters as random variables using the estimation method described in this and the next chapters. For the model and data in this section, we again use the artificial local-level model and data that have been prepared with Code 9.1 in Sect. 9.2. While this section assumes that the parameters of the prior distribution for the state are known as in the previous section, we suppose a situation wherein the variances of the state noise and observation noise are unknown and estimated jointly with the state. The estimation target in this example is now a joint posterior distribution $p(\mathbf{x}_{0:T}, \boldsymbol{\theta} | y_{1:T})$, and we obtain samples from that distribution. This code for **Stan** is as follows.

Code 10.3

```

1 // model10-2.stan
2 // Model: specification (local-level model with unknown parameters)
3
4 data{
5     int<lower=1> t_max;      // Time series length
6     vector[t_max] y;         // Observations
7
8     real m0;                // Mean of prior distribution
9     cov_matrix[1] CO;        // Variance of prior distribution
10 }
11
12 parameters{
13     real x0;                // State [0]
14     vector[t_max] x;         // State [1:t_max]
15
16     cov_matrix[1] W;         // Variance of state noise
17     cov_matrix[1] V;         // Variance of observation noise
18 }
19
20 model{
21     // Likelihood part
22     /* Observation equation */
23     for (t in 1:t_max){
24         y[t] ~ normal(x[t], sqrt(V[1, 1]));
25     }
26
27     // Prior part
28     /* Prior distribution for state */
29     x0 ~ normal(m0, sqrt(CO[1, 1]));
30
31     /* State equation */
32     x[1] ~ normal(x0, sqrt(W[1, 1]));
33
34     for (t in 2:t_max){
35         x[t] ~ normal(x[t-1], sqrt(W[1, 1]));
36     }
37
38     /* Prior distribution for W and V: noninformative prior distribution (utilizing the
39      ↪ default setting) */
40 }
```

At first, comparing the above code with the previous Code 10.1, we recognize that the variance W of the state noise and variance V of the observation noise are moved from the `data` block to the `parameters` block; see the shaded code. We then explain the `model` block. Since the distribution of interest in this example is a joint posterior distribution $p(\mathbf{x}_{0:T}, \boldsymbol{\theta} | y_{1:T})$, we can specifically transform it according to Bayes' theorem as follows:

$$\begin{aligned} p(\mathbf{x}_{0:T}, \boldsymbol{\theta} | y_{1:T}) &= p(\mathbf{x}_{0:T}, \boldsymbol{\theta}, y_{1:T}) / p(y_{1:T}) \\ &\propto p(\mathbf{x}_{0:T}, \boldsymbol{\theta}, y_{1:T}) = \underbrace{p(y_{1:T} | \mathbf{x}_{0:T}, \boldsymbol{\theta})}_{\text{likelihood part}} \underbrace{p(\mathbf{x}_{0:T}, \boldsymbol{\theta})}_{\text{prior distribution part}} \end{aligned}$$

from Eq. (6.28)

$$= p(\mathbf{x}_0 | \boldsymbol{\theta}) p(\boldsymbol{\theta}) \prod_{t=1}^T p(y_t | \mathbf{x}_t, \boldsymbol{\theta}) p(\mathbf{x}_t | \mathbf{x}_{t-1}, \boldsymbol{\theta})$$

regarding the first term $p(\mathbf{x}_0 | \boldsymbol{\theta})$, the prior state \mathbf{x}_0 and the parameter $\boldsymbol{\theta}$ are typically considered mutually independent

$$= \underbrace{\prod_{t=1}^T p(y_t | \mathbf{x}_t, \boldsymbol{\theta})}_{\text{likelihood part}} \underbrace{\prod_{t=1}^T p(\mathbf{x}_t | \mathbf{x}_{t-1}, \boldsymbol{\theta}) p(\mathbf{x}_0) p(\boldsymbol{\theta})}_{\text{prior distribution part}}. \quad (10.2)$$

Thus, the likelihood part $p(y_{1:T} | \mathbf{x}_{0:T}, \boldsymbol{\theta})$ corresponds to the cumulative product $\prod_{t=1}^T p(y_t | \mathbf{x}_t, \boldsymbol{\theta})$ of the observation equation, and the prior distribution part $p(\mathbf{x}_{0:T}, \boldsymbol{\theta})$ corresponds to the cumulative product $\prod_{t=1}^T p(\mathbf{x}_t | \mathbf{x}_{t-1}, \boldsymbol{\theta})$ of the state equation times the prior distributions $p(\mathbf{x}_0)$ and $p(\boldsymbol{\theta})$. The descriptions of the cumulative product $\prod_{t=1}^T p(y_t | \mathbf{x}_t, \boldsymbol{\theta})$ of the observation equation, the cumulative product $\prod_{t=1}^T p(\mathbf{x}_t | \mathbf{x}_{t-1}, \boldsymbol{\theta})$ of the state equation, and the state prior distribution $p(\mathbf{x}_0)$ are the same as those in Code 10.1. Regarding the $p(\boldsymbol{\theta})$, we assume that the prior parameters are typically mutually independent unless there is special a priori information; hence, $p(\boldsymbol{\theta}) = p(W, V) = p(W)p(V)$ holds. If there is further a priori information on $p(W)$ and $p(V)$, we must reflect that in the code. However, because we have no such particular knowledge in this example, we apply the noninformative prior distribution (sufficiently wide uniform distribution). If we omit the setting for the prior distribution, the **Stan** automatically applies the noninformative prior distribution; hence, we use this default feature and omit the description. Consequently, the description in the `model` part is the same as Code 10.1.

Next, the R code for executing Code 10.3 is as follows.

Code 10.4

```

1 > # <<Smoothing for local-level model using MCMC (unknown parameter)>>
2
3 ... Ignore the interim display ...
4
5 > # Model: generation and compilation
6 > stan_mod_out <- stan_model(file = "model10-2.stan")
7 >
8 > # Smoothing: execution (sampling)
9 > fit_stan <- sampling(object = stan_mod_out,
10 +                         data = list(t_max = t_max, y = y,
11 +                                     m0 = mod$m0, CO = mod$CO),
12 +                         pars = c("W", "V", "x"),
13 +                         seed = 123
14 +                     )
15 Warning message:
16 Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians
17 ↪ may be unreliable.
18 Running the chains for more iterations may help. See
19 http://mc-stan.org/misc/warnings.html#bulk-ess
20 >
21 > # Confirmation of the results
22 > oldpar <- par(no.readonly = TRUE); options(max.print = 99999)
23 > fit_stan
24 Inference for Stan model: model10-2.
25 4 chains, each with iter=2000; warmup=1000; thin=1;
26 post-warmup draws per chain=1000, total post-warmup draws=4000.
27
28      mean se_mean    sd   2.5%    25%    50%    75%   97.5% n_eff Rhat
29 W[1,1]  0.97    0.01  0.26   0.57   0.78   0.93   1.12   1.59   510  1.01
30 V[1,1]  2.07    0.01  0.32   1.50   1.85   2.06   2.27   2.76  1598  1.00
31 x[1]    11.78   0.02  0.98   9.93  11.13  11.77  12.43  13.76  3591  1.00
32 x[2]    12.81   0.01  0.86  11.07  12.25  12.81  13.37  14.52  3523  1.00
33
34 ... Ignore the interim display ...
35
36 x[199]  18.62   0.02  0.89  16.90  18.03  18.62  19.24  20.35  3475  1.00
37 x[200]  19.24   0.02  1.05  17.16  18.53  19.26  19.94  21.30  3342  1.00
38 lp__   -261.44  0.98 19.06 -298.47 -274.76 -261.89 -248.50 -223.34   376  1.02
39 Samples were drawn using NUTS(diag_e) at Sat Apr 25 16:00:50 2020.
40 For each parameter, n_eff is a crude measure of effective sample size,
41 and Rhat is the potential scale reduction factor on split chains (at
42 convergence, Rhat=1).
43 >
44 > # Ignore the display of following codes

```

The content of the above code is almost the same as that of the former Code 10.2. However, the variables W and V are removed from the argument `data` of the function `sampling()` and are added to the argument `pars` because this example estimates both the state and parameter in the linear Gaussian state-space model as the augmented state in the general state-space model. We recognize that the effective sample size and \hat{R} have no particular problem by checking the console display of

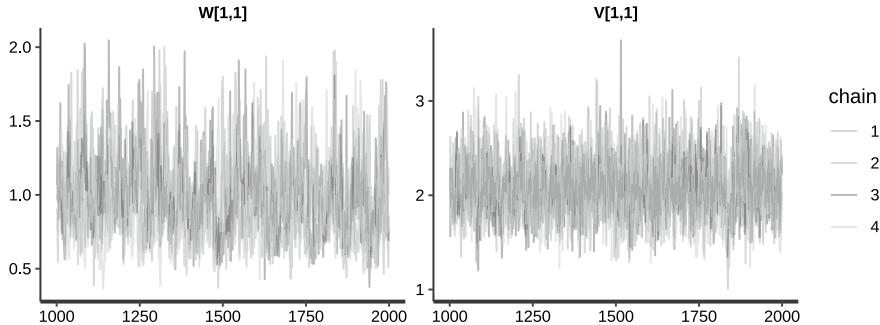
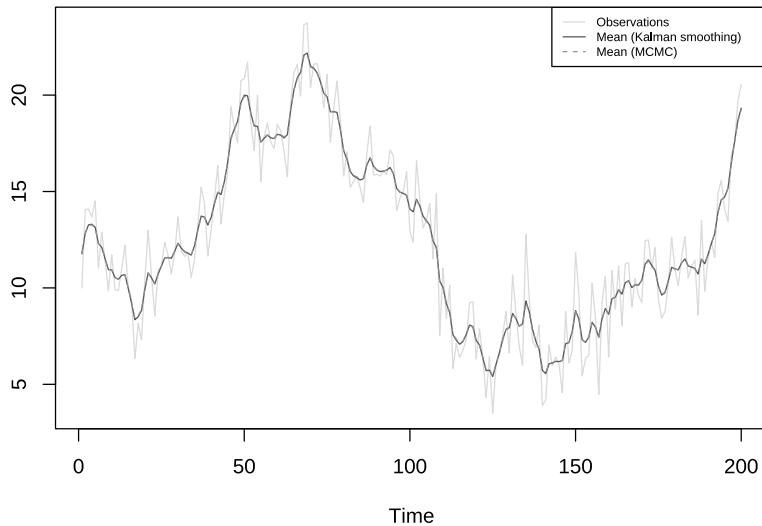


Fig. 10.6 Trace plots (linear Gaussian state-space model with unknown parameters)

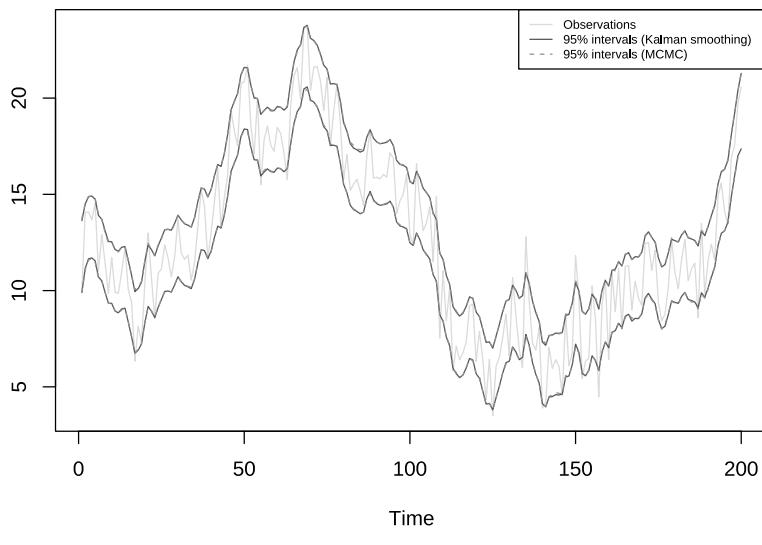
`fit_stan`. The estimated mean values for the variances of the state and observation noises are 0.97 and 2.07, respectively, and their true values are 1 and 2, respectively. Thus, we can confirm that the mean values for the variances can be estimated sufficiently accurately, and their standard errors are also not especially large. However, confirming the result carefully, we see that the effective sample size and \hat{R} are worse than those obtained using Code 10.2. In particular, the `lp__` has a noticeable tendency. This is because the estimation difficulty is increasing. We now estimate the general state-space model unlike the previous Code 10.2, and the number of variables to be estimated has also increased. Among the parameters, the variance of the state noise is a sensitive quantity that directly concerns the dynamic characteristics of the model; hence, the author guesses that its estimation difficulty might influence the evaluation of the log posterior probability density. Figure 10.6 shows the results of the trace plots.

For simplicity, we select the two variances of the state and observation noises and display their trace plots as in the above graphs. We recognize that the result in this example has no particular problem. However, carefully confirming the results, we see a slight unevenness especially in the variance of the state noise. This result is consistent with the result regarding the effective sample size. Figure 10.7 shows the comparison plots between the results by smoothing using MCMC and Kalman smoothing with all known parameters.

As shown in Fig. 10.7, we see that the two results are almost identical; the graph lines overlap and cannot be distinguished from each other.



(a) Mean



(b) 95% intervals

Fig. 10.7 Smoothing using MCMC and Kalman smoothing (linear Gaussian state-space model with unknown parameters)

We compare the marginal smoothing distribution $p(\mathbf{x}_t | y_{1:T})$ in the above. However, the derivation method differs as follows:

$$\text{Smoothing using MCMC: } \int p(\mathbf{x}_t, \boldsymbol{\theta} | y_{1:T}) d\boldsymbol{\theta}$$

$$\text{Kalman smoothing: } p(\mathbf{x}_t | y_{1:T}; \boldsymbol{\theta} = \text{true value})$$

Thus, while both the results are expected to be similar, they are not always the same [10]. Later chapters will perform a similar comparison, that is, when estimating the linear Gaussian state-space model with unknown parameters as the general state-space model, we first estimate parameters as random variables and then marginalize out the estimated parameters; finally, we compare the result with that obtained by the linear Gaussian state-space model with known parameters. While all that kind of comparisons in this book yield consistent results, we cannot always obtain such results in principle.

Through the above simple example, we have confirmed that **Stan** can also estimate the general state-space model accurately.

10.5 Technique for Improving Estimation Accuracy

According to the above description, it is actually possible to estimate the general state-space model using MCMC. MCMC can achieve ideal performance when its Markov chain has infinite iterations. However, the number of iterations is finite in actual implementation; therefore, the performance degrades compared to the ideal situation. Various improved algorithms or implementations have been proposed to maintain performance with fewer repetitions [6, 8]. For example, an advanced algorithm known as the replica exchange method has been developed to avoid the situation wherein it is difficult for Markov chain to get out of the local region of the target distribution. This book explains an improved method for the estimation of the general state-space model using MCMC when the linear Gaussian state-space model is partially applicable.

10.5.1 Case in Which the Linear Gaussian State-Space Model is Partially Applicable

In the analysis of the general state-space model, the linear Gaussian state-space model is sometimes, but not always, partially applicable. In such a case, we can use the Kalman filter as a part of the solution. This approach concentrates the finite MCMC ability on the area that cannot be solved using a Kalman filter. In addition, a Kalman filter is an analytical solution and can improve the total estimation accuracy. Thus, this book recommends using this approach if possible.

Specifically, we can use this approach when the state in the general state-space model is divided into parts that conform to the linear Gaussian state-space model and others. This corresponds to the case in which we regard the unknown parameters in the linear Gaussian state-space model as random variables. In such a situation, if parameters are already known or fixed by means such as maximum likelihood estimation, we can treat the model just as the linear Gaussian state-space model. However, the parameters have actually to be estimated jointly with the states, i.e., the joint posterior distribution $p(\mathbf{x}_{0:T}, \boldsymbol{\theta} | \mathbf{y}_{1:T})$ is the estimation target; hence, we must use the general state-space model. Herein, recall the feature that once the parameters are specified, we can reconsider the model just as a linear Gaussian state-space model. The description below introduces two methods that make use of this feature.

The first method is a simple one, wherein only parameters are estimated as random variables using MCMC and the estimation result is used in a Kalman filter. In this case, the posterior distribution obtained by MCMC is $p(\boldsymbol{\theta} | \mathbf{y}_{1:T})$. Herein, we obtain $p(\boldsymbol{\theta} | \mathbf{y}_{1:T}) \propto p(\mathbf{y}_{1:T} | \boldsymbol{\theta})p(\boldsymbol{\theta})$ from Bayes' theorem. Since the first term $p(\mathbf{y}_{1:T} | \boldsymbol{\theta})$ is the likelihood of the linear Gaussian state-space model, we can use Eq. (8.6). For the second term $p(\boldsymbol{\theta})$, if there is no particular prior information, we apply the noninformative prior distribution to it. Finally, we plug the point estimates of $\boldsymbol{\theta}$ such as the representative value of estimated $\boldsymbol{\theta}$ into the Kalman filter.

Stan prepares a function `gaussian_dlm_obs()` to calculate the likelihood of the linear Gaussian state-space model. This function was implemented as Dr. Jeffrey B. Arnold's contribution and executes Kalman filtering internally. This function in the **Stan** 2.19.2 does not yet support time-varying models and missing observations.

The second method draws a sample from the joint posterior distribution $p(\mathbf{x}_{0:T}, \boldsymbol{\theta} | y_{1:T})$ using FFBS. The joint posterior distribution is expanded from Bayes' theorem as $p(\mathbf{x}_{0:T}, \boldsymbol{\theta} | y_{1:T}) = p(\mathbf{x}_{0:T} | \boldsymbol{\theta}, y_{1:T})p(\boldsymbol{\theta} | y_{1:T})$. The first term $p(\mathbf{x}_{0:T} | \boldsymbol{\theta}, y_{1:T})$ has a condition wherein the $\boldsymbol{\theta}$ is given and follows the linear Gaussian state-space model. For sampling from this distribution, we use the FFBS described in Algorithm 10.4. Regarding the second term $p(\boldsymbol{\theta} | y_{1:T})$, we draw the sample as in the first method. As the final arranged order, the samples are first drawn from the second term $p(\boldsymbol{\theta} | y_{1:T})$, and the samples from the first term $p(\mathbf{x}_{0:T} | \boldsymbol{\theta}, y_{1:T})$ are then drawn using FFBS; the later sample indicates the reproduced state.

The library **dlm** provides the function `dlmBSample()` to execute FFBS. This function calculates Kalman smoothing internally.

When considering parameters in the linear Gaussian state-space model as random variables and estimating the state and parameters jointly, we must essentially use MCMC only for the parameter estimation; both the methods mentioned above focus on the parameters as estimation targets using MCMC. In the above description, the parameter sampling from $p(\boldsymbol{\theta} | y_{1:T})$ is common as first step in both methods. In addition, we have already explained the Kalman filter for the first method in Chap. 8. Thus, we describe the second method using FFBS below.

10.5.2 Example: Artificial Local-Level Model

Again, we consider an example wherein we estimate the state and parameters jointly in the linear Gaussian state-space model with unknown parameters. For the model and data, we reuse the artificial local-level model and data prepared with Code 9.1 in Sect. 9.2. While we applied MCMC directly to the same problem for drawing samples of the state in Sect. 10.4, we now reproduce the samples of the state using FFBS after parameter estimation. As in the former example, we assume that the mean and variance of the prior distribution for the state are known. Because MCMC here draws only the samples regarding the variances of the state and observation noises, the joint posterior distribution to be estimated becomes $p(\boldsymbol{\theta} | y_{1:T}) = p(\mathbf{W}, V | y_{1:T})$. This code for **Stan** is as follows.

Code 10.5

```

1 // model10-3.stan
2 // Model: specification (local-level model with unknown parameters, utilizing Kalman
   ↪ filter)
3
4 data{
5     int<lower=1> t_max;    // Time series length
6     matrix[1, t_max] y;    // Observations
7
8     matrix[1, 1] G;        // State transition matrix
9     matrix[1, 1] F;        // Observation matrix
10    vector[1] m0;         // Mean of prior distribution
11    cov_matrix[1] CO;      // Variance of prior distribution
12 }
13
14 parameters{
15     cov_matrix[1] W;       // Variance of state noise
16     cov_matrix[1] V;       // Variance of observation noise
17 }
18
19 model{
20     // Likelihood part
21     /* Function calculating likelihood of the linear Gaussian state space model */
22     y ~ gaussian_dlm_obs(F, G, V, W, m0, CO);
23
24     // Prior part
25     /* Prior distribution for W and V: noninformative prior distribution (utilizing the
       ↪ default setting) */
26 }
```

Comparing with the Code 10.3, we recognize that the description of the state transition matrix G and the observation matrix F is added in the `data` block. Note that the shaded part indicates the difference between the above code and the previous code. These values are used as arguments of the function `gaussian_dlm_obs()`, which calculates the likelihood of the linear Gaussian state-space model. For compatibility with the specification of this function, the observations y and mean $m0$ for the prior distribution are declared as a matrix with one row and vector with one element, respectively. In the subsequent `parameters` block, the description of the states $x0$ and x that existed in Code 10.3 has been deleted. In the last `model` block, the function `gaussian_dlm_obs()` is used to describe the likelihood part. In the prior distribution part, the description concerning the state that existed in Code 10.3 has been deleted.

Next, the R code for executing Code 10.5 is as follows.

Code 10.6

```

1 > # <<Smoothing for local-level model using MCMC (unknown parameters and use of
2   ← Kalman filter)>>
3 >
4 > # Preprocessing
5 > set.seed(123)
6 > library(rstan)
7 >
8 > # Presetting of Stan: HDD storage of compiled code and parallel computation
9 > rstan_options(auto_write = TRUE)
10 > options(mc.cores = parallel::detectCores())
11 >
12 > # Load data on artificial local-level model
13 > load(file = "ArtifiticialLocalLevelModel.RData")
14 >
15 > # Model: generation and compilation
16 > stan_mod_out <- stan_model(file = "model10-3.stan")
17 >
18 > # Smoothing: execution (sampling)
19 > dim(mod$m0) <- 1           # Set the explicit dimension in the case of only one
20   ← element vector
21 > fit_stan <- sampling(object = stan_mod_out,
22   +                         data = list(t_max = t_max, y = matrix(y, nrow = 1),
23   +                                 G = mod$G, F = t(mod$F),
24   +                                 m0 = mod$m0, CO = mod$CO),
25   +                         pars = c("W", "V"),
26   +                         seed = 123
27   +                     )
28 >
29 > # Confirmation of the results
30 > fit_stan
31 Inference for Stan model: model10-3.
32 4 chains, each with iter=2000; warmup=1000; thin=1;
33 post-warmup draws per chain=1000, total post-warmup draws=4000.
34
35          mean se_mean    sd    2.5%    25%    50%    75%   97.5% n_eff Rhat
36 W[1,1]    0.98    0.01 0.27    0.56    0.79    0.94    1.13    1.61   2160     1
37 V[1,1]    2.06    0.01 0.33    1.48    1.83    2.04    2.27    2.75   2010     1
38 lp__ -235.52    0.03 1.02 -238.39 -235.89 -235.20 -234.79 -234.52   1504     1
39
40 Samples were drawn using NUTS(diag_e) at Sat Apr 25 16:01:10 2020.
41 For each parameter, n_eff is a crude measure of effective sample size,
42 and Rhat is the potential scale reduction factor on split chains (at
43 convergence, Rhat=1).
44 > tmp_tp <- traceplot(fit_stan, pars = c("W", "V"), alpha = 0.5)
45 > tmp_tp + theme(aspect.ratio = 3/4)
```

The content of the above code is almost the same as that of Code 10.4 until the function `sampling()`. The principal difference is as follows. In front of the function `sampling()`, we make `mod$m0` a single element vector by providing the explicit dimension. In the argument `data` of the function `sampling()`, for accordance with the specification of `gaussian_dlm_obs()`, we cast the observations `y` into a one-row matrix and transpose the observation matrix `mod$F`. In addition, the state `x` is deleted from the augment `pars`. Regarding the sampling results, we see that the effective sample size and \hat{R} have no particular problem by checking the console display of the `fit_stan`. The estimated mean values for the variances of the state and observation noises are 0.98 and 2.06, respectively; their true values are one and two, respectively. Thus, we can confirm that the mean values for the variances can be estimated sufficiently accurately, and their standard errors are also not large. Confirming the results carefully, we can also see that the effective sample size and \hat{R} are improved compared to those obtained using Code 10.4. Figure 10.8 shows the result of the trace plot.

The trace plots for the variances of the state and observation noises are displayed in Fig. 10.8. We recognize that the result in this example has no particular problem. Confirming the plots carefully, we see that unevenness in the variance of state noise is slightly improved compared to Fig. 10.6. This result is consistent with that from the effective sample size and \hat{R} .

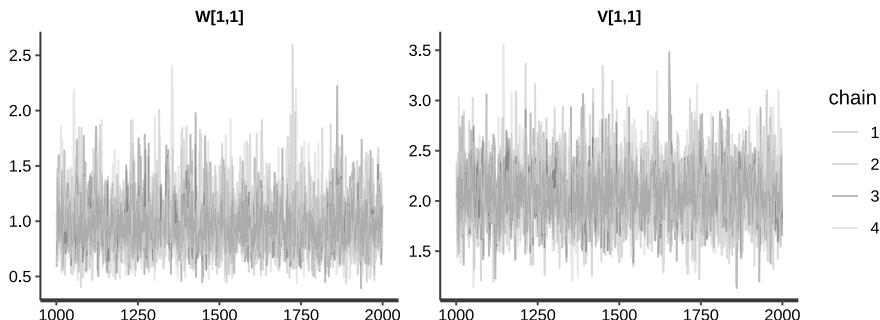


Fig. 10.8 Trace plots (linear Gaussian state-space model with unknown parameters and the use of a Kalman filter)

We then reproduce the state sample using FFBS. This R code is as follows.

Code 10.7

```

44 > # <<Smoothing for local-level model using MCMC (state draw with FFBS)>>
45 >
46 > # Preprocessing
47 > set.seed(123)
48 > library(dlm)
49 >
50 > # Extract necessary sampling results
51 > stan_mcmc_out <- rstan::extract(fit_stan, pars = c("W", "V"))
52 >
53 > # FFBS preprocessing: setting of MCMC iteration step and progress bar
54 > it_seq <- seq_along(stan_mcmc_out$V[, 1, 1])
55 > progress_bar <- txtProgressBar(min = 1, max = max(it_seq), style = 3)
56 >
57 > # FFBS main processing: draw of state
58 > x_FFBS <- sapply(it_seq, function(it){
59 +   # Display progress bar
60 +   setTxtProgressBar(pb = progress_bar, value = it)
61 +
62 +   # Set W and V values in the model
63 +   mod$W[1, 1] <- stan_mcmc_out$W[it, 1, 1]
64 +   mod$V[1, 1] <- stan_mcmc_out$V[it, 1, 1]
65 +
66 +   # FFBS execution
67 +   return(dlmBSSample(dlmFilter(y = y, mod = mod)))
68 + })
69 |=====| 100%
70 >
71 > # FFBS post-processing: removal of x0 and transposition (row means time direction
72 > # according to the output of Stan)
73 > x_FFBS <- t(x_FFBS[-1, ])
74 >
75 > # Calculate the mean, 2.5%, and 97.5% values while marginalizing
76 > s_FFBS <- colMeans(x_FFBS)
77 > s_FFBS_quant <- apply(x_FFBS, 2, FUN = quantile, probs=c(0.025, 0.975))
78 > # Ignore the display of following codes

```

The above code continues from the former Code 10.6. In the main process of FFBS, the state is reproduced for each MCMC iteration step. Because this process is a somewhat time-consuming, the progress bar is displayed. In the processing, realizations of the state and observation noises at a particular MCMC iteration step i are set into the model, and state samples are then reproduced using the function `dlmBSSample()`. The argument of the function `dlmBSSample()` is `modfilt`, which is the output of the function `dlmFilter()`. Since the state and observation noises in the model vary at every iteration step of MCMC, the function `dlmFilter()` is executed each time. Figure 10.9 compares the state result using FFBS with that of Kalman smoothing where all parameters are assumed known.

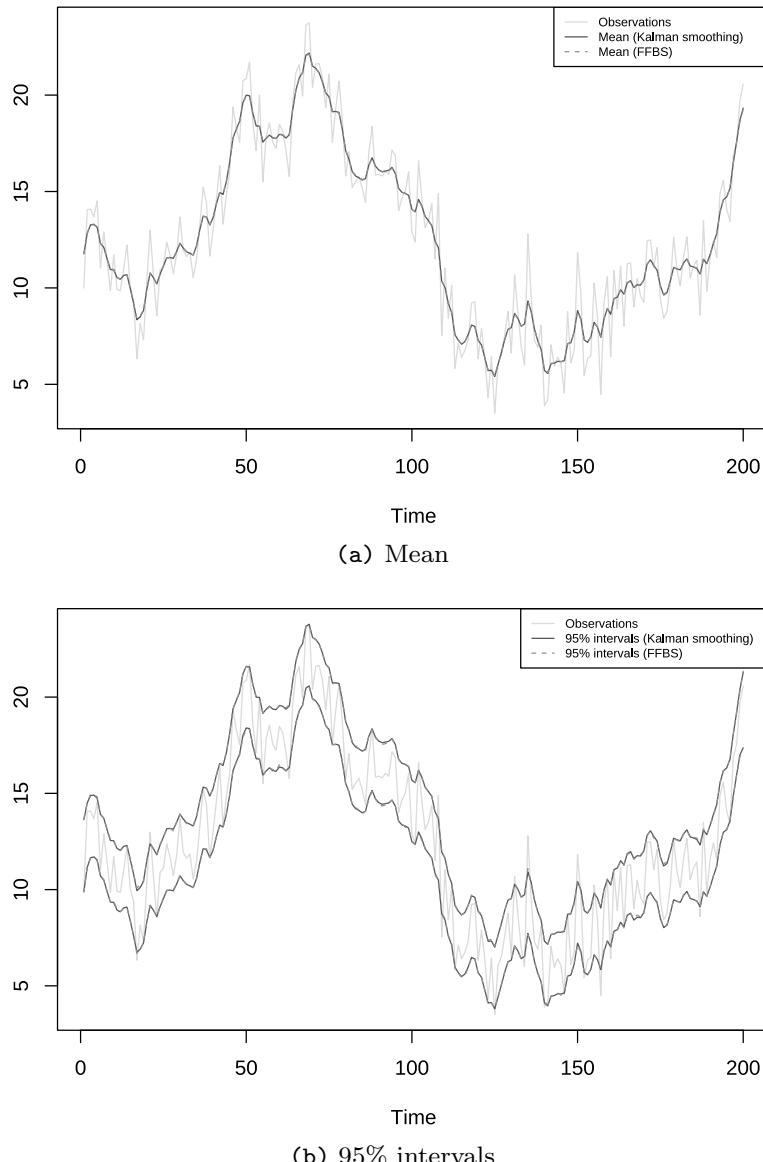


Fig. 10.9 FFBS and Kalman smoothing (linear Gaussian state-space model with unknown parameters)

As shown in Fig. 10.9, we see that the results of the two methods are almost identical; the graph lines overlap and cannot be distinguished from each other.

Through the above simple example, we have confirmed a method for improving estimation accuracy when the linear Gaussian state-space model is partially applicable.

10.5.3 Example: Monthly Totals of Car Drivers in the UK Killed or Injured

Since the local-level model described in the previous example is a simple one, the degree of improvement using the Kalman filter in parts is modest. On the contrary, the previously mentioned method can increase the effectiveness of improvement as the model becomes complicated, such as with many states, deep hierarchy, and strong nonlinearity. We introduce here an analysis example of monthly totals of car drivers in the UK killed or injured. While various models can be applied to the data, we analyze them with a local-level model + seasonal model (time-domain approach) in accordance with [2, Sect. 4.2]. Commandeur and Koopman [2, Sect. 4.2] refers to this model as stochastic level and seasonal. This model has more states than the local-level model; we are to treat more complexity. For the estimation of this model using **Stan**, [22] points out the difficulty of MCMC convergence when we sample the state and parameters jointly. Recall that the local-level model + seasonal model (time-domain approach) is a linear Gaussian state-space model; when we estimate the state and parameters jointly based on this model, the linear Gaussian state-space model is partially applicable. Thus, we can improve the estimation accuracy and stability with the previously mentioned method using a Kalman filter as parts.

We then move on to the actual examination. First, we preliminarily examine the data, specify parameters using the maximum likelihood estimation, and apply Kalman smoothing to the data. This code is as follows.

Code 10.8

```

1 > # <<Monthly totals of car drivers in UK killed or injured: Kalman smoothing>>
2 >
3 > # Preprocessing
4 > set.seed(123)
5 > library(dlm)
6 >
7 > # Log-transform the data and set the time series length
8 > y <- log(UKDriverDeaths)
9 > t_max <- length(y)
10 >
11 > # Plot with the horizontal axis as time
12 > plot(y)
13 >
14 > # Model template
15 > mod <- dlmModPoly(order = 1) + dlmModSeas(frequency = 12)
16 >
17 > # User-defined function to define and build a model
18 > build_dlm_UKD <- function(par) {
19 +   mod$W[1, 1] <- exp(par[1])
20 +   mod$W[2, 2] <- exp(par[2])
21 +   mod$V[1, 1] <- exp(par[3])

```

```

22 +
23 +   return(mod)
24 +
25 >
26 > # Maximum likelihood estimation of parameters
27 > fit_dlm_UKD <- dlmMLE(y = y, parm = rep(0, times = 3), build = build_dlm_UKD)
28 Warning messages:
29 1: In dlmLL(y = y, mod = mod, debug = debug) :
30   a numerically singular 'V' has been slightly perturbed to make it nonsingular
31 2: In dlmLL(y = y, mod = mod, debug = debug) :
32   a numerically singular 'V' has been slightly perturbed to make it nonsingular
33 3: In dlmLL(y = y, mod = mod, debug = debug) :
34   a numerically singular 'V' has been slightly perturbed to make it nonsingular
35 4: In dlmLL(y = y, mod = mod, debug = debug) :
36   a numerically singular 'V' has been slightly perturbed to make it nonsingular
37 5: In dlmLL(y = y, mod = mod, debug = debug) :
38   a numerically singular 'V' has been slightly perturbed to make it nonsingular
39 6: In dlmLL(y = y, mod = mod, debug = debug) :
40   a numerically singular 'V' has been slightly perturbed to make it nonsingular
41 7: In dlmLL(y = y, mod = mod, debug = debug) :
42   a numerically singular 'V' has been slightly perturbed to make it nonsingular
43 >
44 > # Model setting and its confirmation
45 > mod <- build_dlm_UKD(fit_dlm_UKD$par)
46 > cat(diag(mod$W)[1:2], mod$V, "\n")
47 0.0009456339 1.841854e-10 0.003513928
48 >
49 > # Smoothing processing
50 > dlmSmoothed_obj <- dlmSmooth(y = y, mod = mod)
51 >
52 > # Mean of the smoothing distribution
53 > mu <- dropFirst(dlmSmoothed_obj$s[, 1])
54 > gamma <- dropFirst(dlmSmoothed_obj$s[, 2])
55 >
56 > # Plot results
57 > oldpar <- par(no.readonly = TRUE)
58 > par(mfrow = c(3, 1)); par(oma = c(2, 0, 0, 0)); par(mar = c(2, 4, 1, 1))
59 > ts.plot( y, ylab = "Observations (log-transformed)")
60 > ts.plot( mu, ylab = "Level component")
61 > ts.plot(gamma, ylab = "Seasonal component")
62 > mtext(text = "Time", side = 1, line = 1, outer = TRUE)
63 > par(oldpar)

```

We apply a logarithmic transformation to the R built-in `UKDriverDeaths` in accordance with [2, Sect. 4.2]. Figure 10.10 shows the plot for the logarithms of the data.

We can recognize the whole level up and down and the distorted annual cycle. We apply **dlm** functions `dlmModPoly()` and `dlmModSeas()` to this data for setting the local-level model + seasonal model (time-domain approach). We use the default setting of these functions; therefore, the mean vector and covariance matrix in the prior distribution of the state are set to $\mathbf{0}$ and $10^7 \mathbf{I}$, respectively. In the subsequent maximum likelihood estimation for parameters, warnings are issued. These warnings are due to the library **dlm**: when the estimates for the variance of observation noise approach zero, a minimal quantity (10^{-6}) is intentionally added to the value. Because these are considered not to affect the estimation results in this example seriously, we ignore them here. As a result, the variances of the state noises for the level and seasonal components and the variance of the observation noise are 0.0009456339, $1.841854e-10$, and 0.003513928, respectively. These values are not

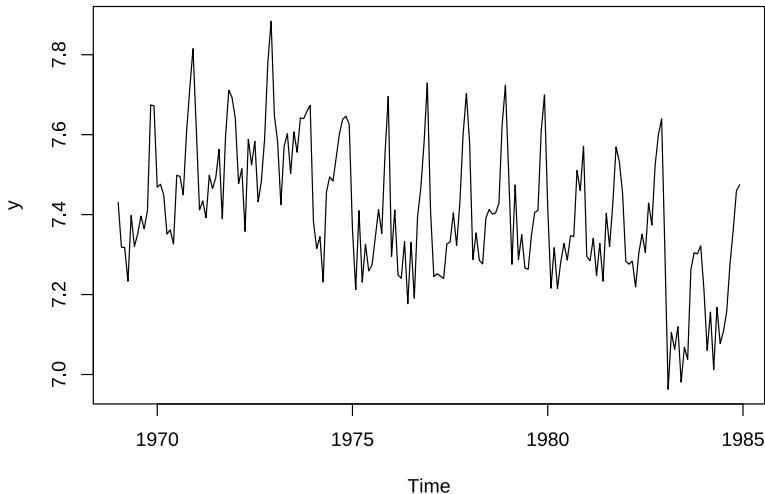


Fig. 10.10 Monthly totals of car drivers in the UK killed or injured (log-transformed)

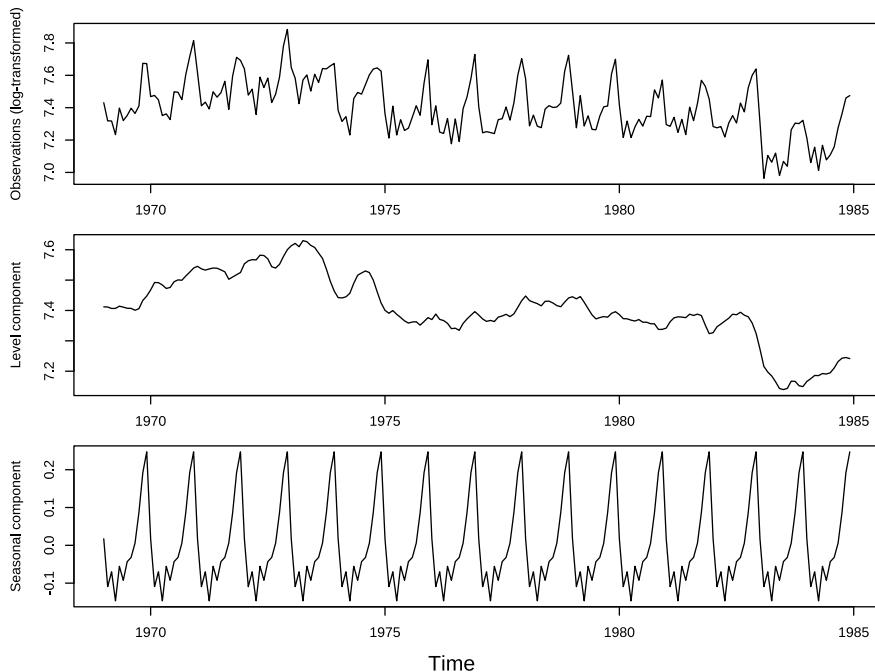


Fig. 10.11 Monthly totals of car drivers in the UK killed or injured (Kalman smoothing)

necessarily correct answers because of real data analysis but can be used as a guide. Figure 10.11 shows the results of Kalman smoothing using the above parameters.

Figure 10.11 shows three plots: observations, level component (mean of the smoothing distribution), and seasonal component (mean of the smoothing distribution) in order from the top.

Subsequently, we draw samples from a joint posterior distribution using **Stan**. First, we explain the direct means of sampling the state and parameters jointly using MCMC. This code for **Stan** is as follows.

Code 10.9

```

1 // model10-4.stan
2 // Model: specification (local-level model + seasonal model (time-domain approach))
3
4 data{
5     int<lower=1> t_max;           // Time series length
6     vector[t_max] y;             // Observations
7
8     vector[12] m0;               // Mean vector of prior distribution
9     cov_matrix[12] CO;          // Covariance matrix of prior distribution
10 }
11
12 parameters{
13     real x0_mu;                // State (level component) [0]
14     vector[11] x0_gamma;        // State (seasonal component) [0]
15     vector[t_max] x_mu;         // State (level component) [1:t_max]
16     vector[t_max] x_gamma;       // State (seasonal component) [1:t_max]
17
18     real<lower=0> W_mu;        // Variance of state noise (level component)
19     real<lower=0> W_gamma;      // Variance of state noise (seasonal component)
20     cov_matrix[1] V;            // Covariance matrix of observation noise
21 }
22
23 model{
24     // Likelihood part
25     /* Observation equation */
26     for (t in 1:t_max){
27         y[t] ~ normal(x_mu[t] + x_gamma[t], sqrt(V[1, 1]));
28     }
29
30     // Prior part
31     /* Prior distribution for state (level component) */
32     x0_mu ~ normal(m0[1], sqrt(CO[1, 1]));
33
34     /* State equation (level component) */
35     x_mu[1] ~ normal(x0_mu, sqrt(W_mu));
36     for(t in 2:t_max){
37         x_mu[t] ~ normal( x_mu[t-1], sqrt(W_mu));
38     }
39
40     /* Prior distribution for state (seasonal component) */
41     for (p in 1:11){
42         x0_gamma[p] ~ normal(m0[p+1], sqrt(CO[(p+1), (p+1)]));
43     }
44
45     /* State equation (seasonal component) */
46     x_gamma[1] ~ normal(-sum(x0_gamma[1:11]),
47                           , sqrt(W_gamma));
48     for(t in 2:11){
49         x_gamma[t] ~ normal(-sum(x0_gamma[t:11])-sum(x_gamma[1:(t-1)]),
50                               , sqrt(W_gamma));
51     }
52     for(t in 12:t_max){
53         x_gamma[t] ~ normal(
54                         -sum(x_gamma[(t-11):(t-1)]),
55                         , sqrt(W_gamma));
56     }
57
58     /* Prior distribution for W and V: noninformative prior distribution (utilizing the
59      ← default setting) */
60 }
```

The author created the above code referring to [15, 23]. The `parameters` block declares the states and parameters to be estimated. The states consist of two kinds of component: level component and seasonal component. The subsequent `model` block describes the likelihood part and prior distribution part on the joint posterior distribution. For the prior distribution of the state at time point zero, we use the values stored in the `mod`, that is, the mean $\mathbf{0}$ and covariance $10^7 \mathbf{I}$. Our settings for the state equation regarding the seasonal component are based on Eq. (9.25). We omit describing the prior distribution for the parameters `W_mu`, `W_gamma`, and `V`; hence, the default noninformative prior distribution is applied to them.

The R code for executing Code 10.9 is as follows.

Code 10.10

```

64 > # <<Monthly totals of car drivers in UK killed or injured: state sampling with
65   ← MCMC>>
66 >
67 > # Preprocessing
68 > set.seed(123)
69 > library(rstan)
70 >
71 > # Presetting of Stan: HDD storage of compiled code and parallel computation
72 > rstan_options(auto_write = TRUE)
73 > options(mc.cores = parallel::detectCores())
74 >
75 > # Model: generation and compilation
76 > stan_mod_out <- stan_model(file = "model10-4.stan")
77 >
78 > # Smoothing: execution (sampling)
79 > fit_stan <- sampling(object = stan_mod_out,
80 +                           data = list(t_max = t_max, y = y, m0 = mod$m0, C0 = mod$C0),
81 +                           pars = c("W_mu", "W_gamma", "V"),
82 +                           seed = 123
83 +                         )
84 Warning messages:
85 1: There were 13 divergent transitions after warmup. Increasing adapt_delta above 0.8
86   ← may help. See
87   http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
88 2: There were 293 transitions after warmup that exceeded the maximum treedepth.
89   ← Increase max_treedepth above 10. See
90   http://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded
91 3: There were 4 chains where the estimated Bayesian Fraction of Missing Information
92   ← was low. See
93   http://mc-stan.org/misc/warnings.html#bfmi-low
94 4: Examine the pairs() plot to diagnose sampling problems
95

```

```

92 5: The largest R-hat is 1.08, indicating chains have not mixed.
93 Running the chains for more iterations may help. See
94 http://mc-stan.org/misc/warnings.html#r-hat
95 6: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and
96 → medians may be unreliable.
97 Running the chains for more iterations may help. See
98 http://mc-stan.org/misc/warnings.html#bulk-ess
99 7: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and
100 → tail quantiles may be unreliable.
101 Running the chains for more iterations may help. See
100 http://mc-stan.org/misc/warnings.html#tail-ess
101 >
102 > # Confirmation of the results
103 > fit_stan
104 Inference for Stan model: model10-4.
105 4 chains, each with iter=2000; warmup=1000; thin=1;
106 post-warmup draws per chain=1000, total post-warmup draws=4000.
107
108      mean se_mean    sd   2.5%   25%   50%   75% 97.5% n_eff Rhat
109 W_mu     0.00     0.0  0.00    0.00   0.00   0.00   0.0  0.0  353 1.01
110 W_gamma  0.00     0.0  0.00    0.00   0.00   0.00   0.0  0.0  39  1.08
111 V[1,1]   0.00     0.0  0.00    0.00   0.00   0.00   0.0  0.0  1013 1.00
112 lp__ 1868.01   12.8 77.81 1729.92 1808.57 1862.26 1927.4 2021.4 37  1.09
113
114 Samples were drawn using NUTS(diag_e) at Sat Apr 25 16:03:01 2020.
115 For each parameter, n_eff is a crude measure of effective sample size,
116 and Rhat is the potential scale reduction factor on split chains (at
117 convergence, Rhat=1).
118 > tmp_tp <- traceplot(fit_stan, pars = c("W_mu", "W_gamma", "V"), alpha = 0.5)
119 > tmp_tp + theme(aspect.ratio = 3/4)

```

The above code continues from the former Code 10.8. Confirming the results, we see that deterioration of the convergence is detected while executing `sampling()` and warnings are issued. From the result `fit_stan`, we also confirm that both effective sample size and \hat{R} worsen for the variance of the state noise regarding the seasonal component and the log posterior probability. Figure 10.12 shows the trace plots; we see that the mixing of estimation results is not good for the variances of state noises (especially regarding the seasonal component).

Next, we explain how to sample only parameters using MCMC and then reproduce the state samples using FFBS. This code for **Stan** is as follows.

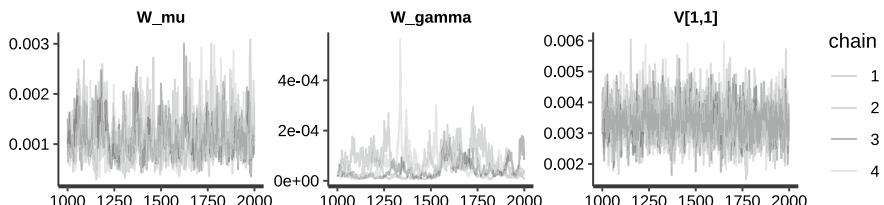


Fig. 10.12 Trace plots (monthly totals of car drivers in the UK killed or injured: state sampling using MCMC)

Code 10.11

```

1 // model10-5.stan
2 // Model: specification (local-level model + seasonal model (time-domain approach),
2 ← utilizing Kalman filter)
3
4 data{
5     int<lower=1> t_max;           // Time series length
6     matrix[1, t_max] y;          // Observations
7
8     matrix[12, 12] G;            // State transition matrix
9     matrix[12, 1] F;             // Observation matrix
10    vector[12] m0;              // Mean vector of prior distribution
11    cov_matrix[12] CO;           // Covariance matrix of prior distribution
12 }
13
14 parameters{
15     real<lower=0> W_mu;         // Variance of state noise (level component)
16     real<lower=0> W_gamma;       // Variance of state noise (seasonal component)
17     cov_matrix[1] V;             // Covariance matrix of observation noise
18 }
19
20 transformed parameters{
21     matrix[12, 12] W;           // Covariance matrix of state noise
22
23     for (k in 1:12){           // In Stan's matrices, the column-major access is
24         ← effective
25         for (j in 1:12){
26             if (j == 1 && k == 1){ W[j, k] = W_mu;      }
27             else if (j == 2 && k == 2){ W[j, k] = W_gamma;  }
28             else{                      W[j, k] = 0;        }
29         }
30     }
31
32 model{
33     // Likelihood part
34     /* Function calculating likelihood of the linear Gaussian state space model */
35     y ~ gaussian_dlm_obs(F, G, V, W, m0, CO);
36
37     // Prior part
38     /* Prior distribution for W and V: noninformative prior distribution (utilizing the
39     ← default setting) */
}

```

In this code, the `parameters` block declares the parameters to be estimated. In the subsequent `transformed parameters` block, we can redefine useful parameters by using constants and parameters declared in the `data` and `parameters` blocks, respectively. Here, we build a covariance matrix for the state noise. The last `model` block uses the function `gaussian_dlm_obs()`. For the prior distribution of the state at time point 0, we use the values stored in `mod`, that is, mean $\mathbf{0}$ and covariance $10^7 \mathbf{I}$. We omit any description of the prior distribution for parameters; hence, the default noninformative prior distribution is applied to them.

The R code for executing Code 10.11 is as follows.

Code 10.12

```

120 > # <<Monthly totals of car drivers in UK killed or injured: no state sampling with
121 >   MCMC>>
122 >
123 > # Preprocessing
124 > set.seed(123)
125 > library(rstan)
126 >
127 > # Presetting of Stan: HDD storage of compiled code and parallel computation
128 > rstan_options(auto_write = TRUE)
129 > options(mc.cores = parallel::detectCores())
130 >
131 > # Model: generation and compilation
132 > stan_mod_out <- stan_model(file = "model10-5.stan")
133 >
134 > # Smoothing: execution (sampling)
135 > fit_stan <- sampling(object = stan_mod_out,
136 >   data = list(t_max = t_max, y = matrix(y, nrow = 1),
137 >     G = mod$G, F = t(mod$F),
138 >     m0 = mod$m0, CO = mod$CO),
139 >   pars = c("W_mu", "W_gamma", "V"),
140 >   seed = 123
141 > )
142 >
143 > # Confirmation of the results
144 > fit_stan
145 Inference for Stan model: model10-5.
146 4 chains, each with iter=2000; warmup=1000; thin=1;
147 post-warmup draws per chain=1000, total post-warmup draws=4000.
148
149       mean se_mean    sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
150 W_mu      0.00    0.00 0.00  0.00  0.00  0.00  0.0  0.00  2191 1.00
151 W_gamma   0.00    0.00 0.00  0.00  0.00  0.00  0.0  0.00  3240 1.00
152 V[1,1]    0.00    0.00 0.00  0.00  0.00  0.00  0.0  0.00  2172 1.00
153 lp__     233.25   0.04 1.33 229.71 232.69 233.59 234.2 234.72  1292 1.01
154
155 Samples were drawn using NUTS(diag_e) at Sat Apr 25 16:04:49 2020.
156 For each parameter, n_eff is a crude measure of effective sample size,
157 and Rhat is the potential scale reduction factor on split chains (at
158 convergence, Rhat=1).
159 > tmp_tp <- traceplot(fit_stan, pars = c("W_mu", "W_gamma", "V"), alpha = 0.5)
160 > tmp_tp + theme(aspect.ratio = 3/4)
161 >
162 > # Confirm the mean of the marginal distribution as the estimation result
163 > cat(summary(fit_stan)$summary["W_mu", "mean"],
164 >   summary(fit_stan)$summary["W_gamma", "mean"],
165 >   summary(fit_stan)$summary["V[1,1]", "mean"], "\n")
0.001134877 5.002168e-05 0.003340325

```

The above code continues from the former Code 10.10. From the result `fit_stan`, we confirm that both the effective sample size and \hat{R} are improved. Figure 10.13 shows the trace plots; we see that the mixing of estimation results is improved especially for the variance of the state noise regarding the seasonal component.

For the parameter estimation result (mean of the marginal distribution), the variances of the state noises for the level & seasonal components and the variance of the observation noise are 0.001134877 & 5.002168e-05 and 0.003340325, respectively. While the variance of the state noise for the seasonal component is greater than

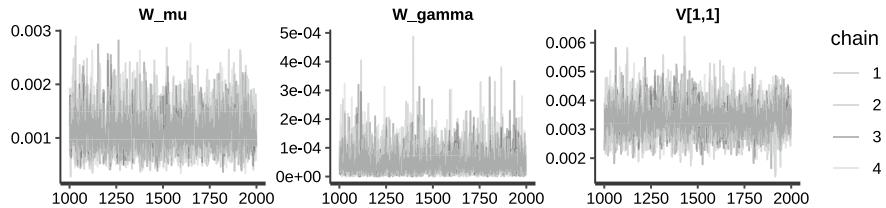


Fig. 10.13 Trace plots (monthly totals of car drivers in the UK killed or injured: no MCMC state sampling)

1.841854e-10 of the maximum likelihood estimates, we obtain other values close to the maximum likelihood estimates.

Last, we reproduce the state sample using FFBS and compare its statistics to those obtained using Kalman smoothing with parameters specified by maximum likelihood estimation. This code is as follows.

Code 10.13

```

166 > # <<Monthly totals of car drivers in UK killed or injured: state draw with FFBS>>
167 >
168 > # Preprocessing
169 > set.seed(123)
170 > library(dlm)
171 >
172 > # Extract necessary sampling results
173 > stan_mcmc_out <- rstan::extract(fit_stan, pars = c("W_mu", "W_gamma", "V"))
174 >
175 > # FFBS preprocessing: setting of MCMC iteration step and progress bar
176 > it_seq <- seq_along(stan_mcmc_out$V[, 1, 1])
177 > progress_bar <- txtProgressBar(min = 1, max = max(it_seq), style = 3)
178 >
179 > # FFBS main processing: draw of state
180 > x_FFBS <- lapply(it_seq, function(it){
181 +   # Display progress bar
182 +   setTxtProgressBar(pb = progress_bar, value = it)
183 +
184 +   # Set W and V values in the model
185 +   mod$W[1, 1] <- stan_mcmc_out$W_mu[it]
186 +   mod$W[2, 2] <- stan_mcmc_out$W_gamma[it]
187 +   mod$V[1, 1] <- stan_mcmc_out$V[it, 1, 1]
188 +
189 +   # FFBS execution
190 +   return(dlmBSSample(dlmFilter(y = y, mod = mod)))
191 + })
192 |=====| 100%
193 >
194 > # FFBS post-processing: removal of x0 and transposition (row means time direction
195 > # according to the output of Stan)
196 > x_mu_FFBS <- t(sapply(x_FFBS, function(x){ x[-1, 1] }))
197 > x_gamma_FFBS <- t(sapply(x_FFBS, function(x){ x[-1, 2] }))
198 >
199 > # Calculate the mean while marginalizing
200 > mu_FFBS <- colMeans( x_mu_FFBS)
201 > gamma_FFBS <- colMeans(x_gamma_FFBS)
202 >
203 > # Ignore the display of following codes

```

The above code continues from the former Code 10.12. As with Code 10.7, in the main process of FFBS, the state is reproduced for each MCMC iteration step. Figure 10.14 compares the state result using FFBS with that obtained using Kalman smoothing where parameters are specified by maximum likelihood estimation.

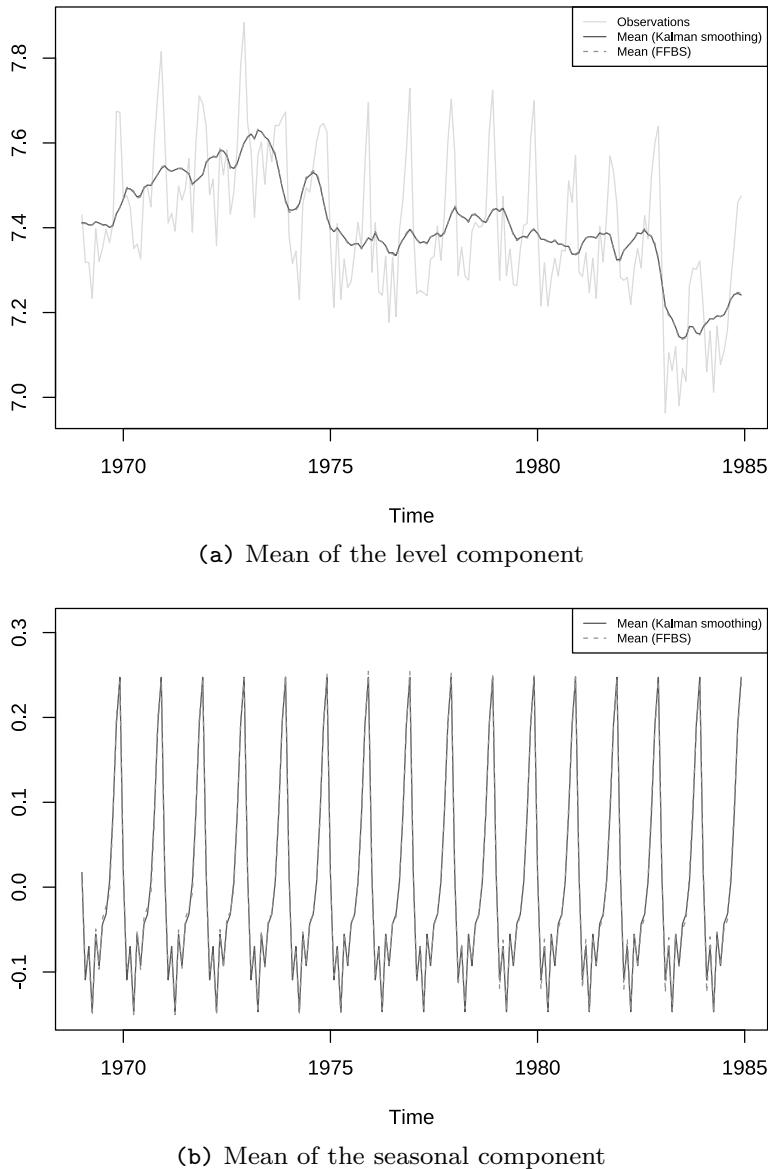


Fig. 10.14 FFBS and Kalman smoothing (monthly totals of car drivers in the UK killed or injured)

As shown in Fig. 10.14, we see that the two sets of results are almost identical; the graph lines overlap and cannot be distinguished from each other. Confirming carefully, we can see a slight deviation for the graph regarding the mean of the seasonal component. This result appears to be influenced by the fact that the variance of the state noise regarding the seasonal component is estimated to be greater than the maximum likelihood estimates. However, we cannot see a big difference overall for the mean of the marginal smoothing distribution.

Through the above example, we have confirmed that the stability of the estimation for a complex model can be improved by using a Kalman filter as parts when the linear Gaussian state-space model is partially applicable.

References

1. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, Berlin (2006)
2. Commandeur, J.J., Koopman, S.J.: An Introduction to State Space Time Series Analysis. Oxford University Press, Oxford (2007)
3. Duane, S., Kennedy, A., Pendleton, B.J., Roweth, D.: Hybrid monte carlo. Phys. Lett. B **195**(2), 216–222 (1987)
4. Durbin, J., Koopman, S.J.: Time Series Analysis by State Space Methods, 2nd edn. Oxford University Press, Oxford (2012)
5. Fitting Bayesian Structural Time Series with the bsts R Package. <http://www.unofficialgoogledatascience.com/2017/07/fitting-bayesian-structural-time-series.html>. Visited on 26 Sept 2017
6. Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B., Vehtari, A., Rubin, D.B.: Bayesian Data Analysis, 3rd edn. CRC Press, Boca Raton (2013)
7. Helske, J.: KFAS: Exponential family state space models in R. J. Stat. Softw. **78**(10), 1–39 (2017)
8. Iba, Y., Tanemura, M., Omori, Y., Wago, H., Sato, S., Takahashi, A.: Computational Statistics II: Markov Chain Monte Carlo Method and Related Topics. Iwanami Shoten, Tokyo (2005). [in Japanese]
9. Iwanami Data Science Publication Committee (ed.): Iwanami Data Science, vol. 1. Iwanami Shoten, Tokyo (2015). [in Japanese]
10. Iwanami Data Science Publication Committee (ed.): Iwanami Data Science, vol. 6. Iwanami Shoten, Tokyo (2017). [in Japanese]
11. Kitagawa, G.: Introduction to Time Series Modeling. CRC Press, Boca Raton (2009)
12. Kitagawa, G., Gersch, W.: Smoothness Priors Analysis of Time Series. Springer, Berlin (1996)
13. Konishi, S., Ochi, Y., Omori, Y.: Computational Statistics Method—Bootstrap/EM Algorithm/MCMC—. Asakura Publishing Co., Ltd., Tokyo (2008). [in Japanese]
14. Kubo, T.: Introduction to Statistical Modeling for Data Analysis—Generalized Linear Model/Hierarchical Bayesian Model/MCMC. Iwanami Shoten, Tokyo (2012). [in Japanese]
15. Matsuuwa, K.: Bayesian Statistical Modeling using Stan and R. Kyoritsu Shuppan Co., Ltd., Tokyo (2016). [in Japanese]
16. Nomura, S.: Kalman Filter—Time Series Prediction and State Space Model using R—. Kyoritsu Shuppan Co., Ltd., Tokyo (2016). [in Japanese]
17. Okumura, H., Makiyama, K., Uryu, S.: Introduction to Bayesian Statistics with R. Gijutsu Hyohron Co., Ltd., Tokyo (2018). [in Japanese]
18. Petris, G., Petrone, S., Campagnoli, P.: Dynamic Linear Model with R. Springer, Berlin (2009)
19. Profet: Forecasting at Scale. <https://facebook.github.io/prophet/>. Visited on 11 Apr 2020

20. [R] How to Use bsts Package. <http://ill-identified.hatenablog.com/entry/2017/09/08/001002>. Visited on 26 Sept 2017. [in Japanese]
21. Stan Development Team: Stan Modeling Language Users Guide and Reference Manual (Version 2.14.0) (2016). <http://mc-stan.org>
22. State space model with R: Reproduce the examples in the book “An Introduction to State Space Time Series Analysis” with rstan. <http://sinhrks.hatenablog.com/entry/2015/05/28/071124>. Visited on 21 Apr 2017. [in Japanese]
23. Sinhrks/GitHub: https://raw.githubusercontent.com/sinhrks/stan-statespace/master/models/fig04_06.stan. Visited on 21 Apr 2017. [comments in Japanese]
24. The Institute of Statistical Mathematics: TSSS: Time Series Analysis with State Space Model (2014). <http://jasp.ism.ac.jp/ism/TSSS/>. R package version 1.0.1 based on the program by Genshiro Kitagawa
25. Toyoda, H.: Bayesian statistics from the basics—A practical introduction to the Hamiltonian Monte Carlo method. Asakura Publishing Co., Ltd., Tokyo (2015). [in Japanese]
26. Try Using the bsts Package for Estimating the Bayesian Structural Time Series Model. <http://tjo.hatenablog.com/entry/2017/03/07/190000>. Visited on 26 Sept 2017. [in Japanese]

Chapter 11

Sequential Solution for General State-Space Model



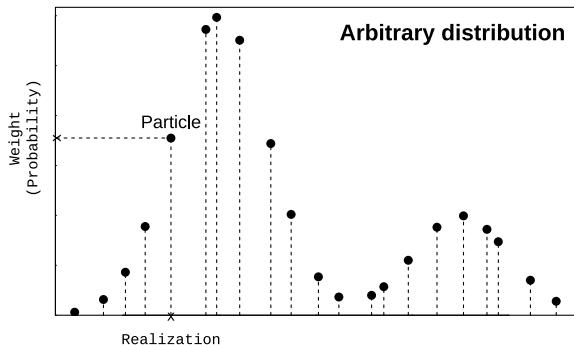
This chapter describes the sequential estimation method for the case wherein the observations are obtained sequentially in the general state-space model. While various methods have been proposed as solutions, this book introduces the particle filter. Regarding the implementation of the particle filter, this book mentions both the scratching and library-based approaches. However, the so-called standard library is not yet wide-spread; hence, this chapter focuses on the scratching approach. The chapter also describes the techniques for improving estimation accuracy, such as an auxiliary particle filter and Rao–Blackwellization. Although a particle filter is a sequential solution, it can be applied also for the case wherein there are a specific number of observations.

11.1 Particle Filter

The previous chapter described the MCMC-based method as a batch solution for the general state-space model. This method can in principle be used with sequentially obtained data. However, for such a case, we must recalculate MCMC for all time points as time progresses and this is inefficient. Regarding the sequential solution for the general state-space model, there have been several proposals such as the applied version of the Kalman filter. Among them, the proposed method in the earliest era was the extended Kalman filter [30], which was initially studied in NASA's Apollo program and used practically.¹ The Gaussian-sum filter [23] approximates the distribution of interest by a Gaussian mixture and utilizes the Kalman filter for its filtering. In addition, the unscented Kalman filter [18] is the basis for a method proposed in robotics, and the ensemble Kalman filter [9] was proposed for meteorology. All the

¹ The source code of perhaps the world's first extended Kalman filter used in the Apollo program is published at https://github.com/chrislgarry/Apollo-11/blob/master/Luminary099/KALMAN_FILTER.agc.

Fig. 11.1 Concept of particle filter



above methods sequentially estimate the statistics, such as mean and covariance for the posterior distribution of interest. On the contrary, a *particle filter* [11, 20] approximates the posterior distribution of interest directly using many *particles* comprising pairs of realizations and weights, as shown in Fig. 11.1. There is also another representation wherein the weights of the particles are converted to the equivalent number of them; these are in principle identical.

If necessary, the particle filter can also derive various statistics after directly approximating the posterior distribution. While each of the abovementioned methods has its own characteristics, this book focuses on the explanation of the particle filter considering its theoretical universality. For information on the extended, unscented, and ensemble Kalman filters, see [8, 13, 19].

Although the principle of particle filters has been proposed a long time ago [1, 12, 41], the method appears to be used since [11, 20] actually owing to the improvement in computing power. The particle filter is also referred to as the *sequential Monte Carlo method* from its principle and *SIR method* (SIR; sequential importance resampling) from the viewpoint of a sequential version of importance sampling.

This section proceeds to explain filtering, prediction, and smoothing using particle filters [3, 5–7, 13–16, 22, 25, 33, 35, 37, 40].

While the particle filter is a suitable solution for sequentially obtained data, it is an approximate solution unlike the analytical solution provided by the Kalman filter; the approximation errors accumulate as processing proceeds. Thus, the estimation accuracy might be degraded in very long continuous operation. For the prevention of such deterioration, it would be better practically to restart the filter process after every specified period [33].

11.1.1 Particle Filtering

Particle filtering is interpretable in various ways. The author considers the most general view to be a sequential version of importance sampling. Thus, this book first gives a brief explanation of importance sampling as follows.

For calculating statistics regarding the distribution of interest, *importance sampling* draws a sample from a surrogate distribution (*proposal distribution*), which is easy to sample, and then applies a correction with the likelihood of the distribution of interest. We explain an example step by step. An expectation of a function $f(\mathbf{x})$ concerning a random variable \mathbf{x} is calculated as

$$\mathbb{E}[f(\mathbf{x})] = \int f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x}.$$

If sampling from $p(\mathbf{x})$ is easy, the above equation can be approximated as

$$\mathbb{E}[f(\mathbf{x})] \approx \frac{1}{N} \sum_{n=1}^N f(\mathbf{x}^{(n)}),$$

where the superscript (n) to \mathbf{x} denotes the serial number of the sample, and N is the sample size. However, unfortunately, sampling from $p(\mathbf{x})$ is not always easy. Importance sampling uses a surrogate distribution (proposal distribution) that is easy to sample to overcome such a situation. Let the proposal distribution be $q(\cdot)$ explicitly, we get

$$\begin{aligned} \mathbb{E}[f(\mathbf{x})] &= \int f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \\ &= \int f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x}) d\mathbf{x} \end{aligned}$$

define $w(\mathbf{x}) = p(\mathbf{x})/q(\mathbf{x})$

$$\begin{aligned} &= \int f(\mathbf{x}) w(\mathbf{x}) q(\mathbf{x}) d\mathbf{x} \\ &\approx \frac{1}{N} \sum_{n=1}^N w(\mathbf{x}^{(n)}) f(\mathbf{x}^{(n)}). \end{aligned}$$

Thus, $\mathbb{E}[f(\mathbf{x})]$ can be approximated using samples from the proposal distribution. We call $w(\mathbf{x})$ an importance function, which conveys the meaning of both canceling the influence of the proposal distribution and performing a correction with the likelihood of the distribution of interest. From this mechanism, we recognize that if samples are drawn from a proposal distribution that is far from the distribution of interest, there are many samples for which the likelihood is low in the importance function.

This results in substantial waste in the approximate calculation. Thus, if the sample size is constant, the closer the proposal distribution $q(\cdot)$ is to the distribution $p(\cdot)$ of interest, the better the approximation accuracy becomes.

The above is a brief explanation of importance sampling. Applying this mechanism to filtering and arranging it in such a manner that the weights in the discrete approximation of filtering distribution can be updated sequentially, we obtain the particle filtering algorithm; see Appendix E.4 for the detailed derivation. Specifically, the procedure for obtaining the filtering distribution at time point t from that at time point $t - 1$ is as follows.

Algorithm 11.1 Particle filtering

0. filtering distribution at time point $t - 1$: $\left\{ \text{realizations } \mathbf{x}_{t-1}^{(n)}, \text{weights } \omega_{t-1}^{(n)} \right\}_{n=1}^N$
 1. update procedure at time point t
 - **for** $n = 1$ **to** N **do**
 - a. *realizations*
Draw $\mathbf{x}_t^{(n)}$ from the proposal distribution $q\left(\mathbf{x}_t \mid \mathbf{x}_{0:t-1}^{(n)}, \mathbf{y}_{1:t}\right)$.
 - b. *weights*

$$\omega_t^{(n)} \leftarrow \omega_{t-1}^{(n)} \frac{p\left(\mathbf{x}_t^{(n)} \mid \mathbf{x}_{t-1}^{(n)}\right) p\left(\mathbf{y}_t \mid \mathbf{y}_{1:t-1}, \mathbf{x}_t^{(n)}\right)}{q\left(\mathbf{x}_t^{(n)} \mid \mathbf{x}_{0:t-1}^{(n)}, \mathbf{y}_{1:t}\right)} \quad (11.1)$$
 - end for
 - *normalization of the weights*: $\omega_t^{(n)} \leftarrow \omega_t^{(n)} / \sum_{n=1}^N \omega_t^{(n)}$
 - *resampling*
Sample from the set $\{1, \dots, N\}$ with replacement, with a probability proportional to $\omega_t^{(n)}$ ($n = 1, \dots, N$), and prepare the index sequence $\mathbf{k} = \{k_1, \dots, k_n, \dots, k_N\}$ for resampling.
Relabel $\mathbf{x}_t^{(k)} = \left\{ \mathbf{x}_t^{(k_1)}, \dots, \mathbf{x}_t^{(k_n)}, \dots, \mathbf{x}_t^{(k_N)} \right\}$ to obtain the realizations $\mathbf{x}_t^{(n)}$, and reset every weight $\omega_t^{(n)}$ to $1/N$.
 2. filtering distribution at time point t : $\left\{ \text{realizations } \mathbf{x}_t^{(n)}, \text{weights } \omega_t^{(n)} \right\}_{n=1}^N$
-

In the above algorithm, the total number of particles is assumed to be N . In addition, this chapter denotes the particle index of the variable by the superscript to the variable $^{(n)}$. Note that the previous chapters denoted the variable type in modeling or the iteration step in the Markov chain with such a superscript. Repeating the procedure of Algorithm 11.1 from $t = 1$, we obtain the filtering distribution at every time point. The filtering distribution at time point zero corresponds to the prior distribution, and we usually set $\mathbf{x}_0^{(n)}$ to the any realization drawn from the prior distribution and set $\omega_0^{(n)} = 1/N$. The procedure in Algorithm 11.1 is consistent with the description in Sect. 6.2.3 as follows:

- One-step-ahead predictive distribution: corresponds to $\omega_{t-1}^{(n)} p(\mathbf{x}_t^{(n)} | \mathbf{x}_{t-1}^{(n)})$ in Eq. (11.1).
- One-step-ahead predictive likelihood: is taken into account through the normalization of the weights.
- Filtering distribution: corresponds to the correction by $p(y_t | y_{1:t-1}, \mathbf{x}_t^{(n)})$ in Eq. (11.1).

Herein, we supplement the Algorithm 11.1.

The step 1-a uses the proposal distribution $q(\cdot)$ to draw realizations. Since the proposal distribution is just a surrogate for the distribution of interest, its effect is eventually canceled by the division in Eq. (11.1). While we can choose the proposal distribution freely in principle, its density must not be zero where the distribution of interest has non-zero density. In addition, since the number of particles is finite, we must use a distribution similar to the one of interest as much as possible; such a choice is actually difficult. Incidentally, if we derived the optimal proposal distribution that minimized the variation of weight $\omega_t^{(n)}$, the distribution $q(\cdot)$ depended only on \mathbf{x}_{t-1} and y_t from the conditional independence in the state-space model; hence, we would have $q(\mathbf{x}_t^{(n)} | \mathbf{x}_{0:t-1}^{(n)}, y_{1:t}) = p(\mathbf{x}_t^{(n)} | \mathbf{x}_{t-1}^{(n)}, y_t)$ [33, 35]. While what kind of distribution is to be used pragmatically as the proposal distribution depends on the problem, the state equation is often used. Such a choice can reduce Eq. (11.1) as follows:

$$\begin{aligned}\omega_t^{(n)} &\leftarrow \omega_{t-1}^{(n)} \frac{p(\mathbf{x}_t^{(n)} | \mathbf{x}_{t-1}^{(n)}) p(y_t | y_{1:t-1}, \mathbf{x}_t^{(n)})}{q(\mathbf{x}_t^{(n)} | \mathbf{x}_{0:t-1}^{(n)}, y_{1:t})} \\ &= \omega_{t-1}^{(n)} \frac{p(\mathbf{x}_t^{(n)} | \mathbf{x}_{t-1}^{(n)}) p(y_t | y_{1:t-1}, \mathbf{x}_t^{(n)})}{p(\mathbf{x}_t^{(n)} | \mathbf{x}_{t-1}^{(n)})} \\ &= \omega_{t-1}^{(n)} p(y_t | y_{1:t-1}, \mathbf{x}_t^{(n)}).\end{aligned}\quad (11.2)$$

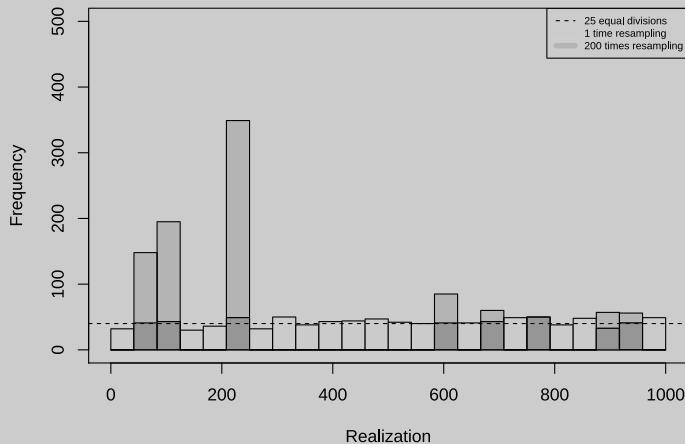
The above type of particle filter is also called a *bootstrap filter* or *Monte Carlo filter*. When the proposal distribution is set to the state equation, information regarding observations $y_{1:t}$ (especially y_t) is not taken into consideration. This might cause the degrading of estimation performance with a limited number of particles. The auxiliary particle filter is popular as a proposal for mitigating such concerns; this chapter explains it later as one of the techniques for improving estimation accuracy.

We can ignore the term $y_{1:t-1}$ of $p(y_t | y_{1:t-1}, \mathbf{x}_t^{(n)})$ in Eq. (11.1) with conditional independence assumed for the state-space model. However, we intentionally left the term considering the alignment with the later description of Rao–Blackwellization.

The *resampling* process reuses a limited number of particles efficiently. Although this process is not necessary in principle if the number of particles would be infinite, it is required practically because the number of particles is finite. The resampling process eliminates particles with small weights in the filtering distribution and accumulates those with large weights in the filtering distribution. As a result, many parti-

cles are preplaced around the area where the weights would be likely to grow at the next time point; we can prevent the degrading of the approximation accuracy for the distribution. The resampling process can be implemented through the index operation; hence, this book describes the process from such a viewpoint. For example, if every weight $\omega_t^{(n)}$ is equal, an index sequence k for resampling ideally provides a uniform result, such as $k = \{1, 2, 3, \dots, N - 2, N - 1, N\}$. On the contrary, if only the weight $\omega_t^{(2)}$ is prominent, the index sequence for resampling yields a result that contains many instances of two, such as $k = \{1, 2, 2, 2, 2, 3, \dots\}$. We can finally reselect the realizations consistently with the unevenness of $\omega_t^{(n)}$ by relabeling them which are selected in accordance with the value of k . Since resampling reduces the diversity of particles, excessive resampling can degrade estimation performance. While this book assumes for simplicity that resampling is always performed, the condition or timing for its execution is flexible. For example, we can perform the resampling only if the effective number of particles decreases to a specified threshold.

Repeated resampling can cause *particle degeneracy*. Particle degeneracy is a phenomenon wherein the weights of particles are biased toward a specific few particles. While we can prevent degeneracy pragmatically using sufficient number of particles, we confirm the phenomenon through the following simple experiment to enhance understanding.



First, assume an integer sequence $1, 2, \dots, 1,000$, and let each value be a particle index. When we express this sequence by a histogram with 25 equal divisions of bins, first, it becomes a completely flat distribution as depicted by the dotted line in the above figure. Next, resampling once with equal weight for every particle results in a light gray (red if colored) histogram. Ideally, the same sequence must be reproduced because every particle has equal weight. However, duplication occurs due to the uncertainty while resampling; hence, a slight bias occurs even though resampling was conducted only once. In addition,

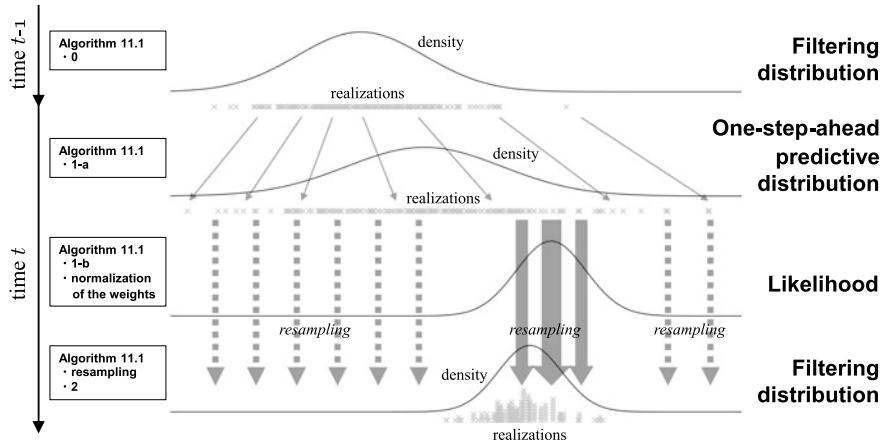


Fig. 11.2 Bootstrap or Monte Carlo filter example

tion, the result of resampling 200 times under the same conditions becomes a dark gray (blue if colored) histogram. We see that duplication while resampling is repeated, and the final result contains a heavy bias.

Finally, we provide a conceptual diagram for particle filtering. Assume a bootstrap or Monte Carlo filter in which resampling is performed at every time point. Figure 11.2 depicts the algorithm flow of particle filtering in such a situation.

In this case, every particle has equal weight; hence, the diagram can express directly the filtering mechanism of one-step-ahead prediction and correction with likelihood. There is another well-known representation of Fig. 11.2 in which time progress is expressed in the horizontal direction, and the consideration of likelihood is shown by the pictorial particle sizes [17].

11.1.2 Particle Prediction

We now explain particle prediction based on the description in Sect. 6.2.4.

The procedure for obtaining the k -steps-ahead predictive distribution at time point $t + k$ from that at time point $t + (k - 1)$ is as follows; see Appendix E.4 for its detailed derivation.

Algorithm 11.2 Particle prediction

-
0. $k - 1$ -steps-ahead predictive distribution at time point $t + (k - 1)$: $\left\{ \begin{array}{l} \text{realizations } \mathbf{x}_{t+(k-1)}^{(n)}, \\ \text{weights } \omega_{t+(k-1)}^{(n)} \end{array} \right\}_{n=1}^N$
1. update procedure at time point $t + k$
- **for** $n = 1$ to N **do**
 - a. *realizations*
Draw $\mathbf{x}_{t+k}^{(n)}$ from the state equation $p(\mathbf{x}_{t+k} | \mathbf{x}_{t+(k-1)}^{(n)})$.
 - b. *weights*
$$\omega_{t+k}^{(n)} \leftarrow \omega_{t+(k-1)}^{(n)} \quad (11.3)$$
- end for**
2. k -steps-ahead predictive distribution at time point $t + k$: $\left\{ \begin{array}{l} \text{realizations } \mathbf{x}_{t+k}^{(n)}, \text{ weights} \\ \omega_{t+k}^{(n)} \end{array} \right\}_{n=1}^N$
-

Repeating the procedure of Algorithm 11.2 from $k = 1$, we obtain the predictive distribution at time point $t + k$. Note that the 0-steps-ahead predictive distribution at time point t corresponds to the filtering distribution at time point t . The procedure in Algorithm 11.2 is consistent with the description in Sect. 6.2.4 as follows:

- the k -steps-ahead predictive distribution: the $k - 1$ -steps-ahead predictive distribution is transitioned based on the state equation.

Herein, we supplement particle prediction.

We can interpret the algorithm of particle prediction as that of particle filtering without observations y_t in Algorithm 11.1. Based on this view, we apply the state equation to the proposal distribution in step 1-a as in the bootstrap or Monte Carlo filter. In the subsequent step 1-b, the state equation is canceled by the proposal distribution, and there is no correction by the likelihood; hence, the weight remains unchanged. Consequently, both normalization of the weights and resampling are unnecessary.

11.1.3 Particle Smoothing

We now explain particle smoothing based on the description in Sect. 6.2.5. While many methods have been developed for particle smoothing, this book describes two typical ones. These methods reselect the filtered particles considering the relative future information compared with the filtering time point. We omit the description of the other proposed algorithms [3, 5, 7, 15, 25, 37], such as one based on the two-filter formula. This book focuses on fixed-interval smoothing; hence, we assume that

particle filtering through time point T has been completed prior to smoothing. The smoothing distribution at time point T corresponds to the filtering distribution at time point T .

11.1.3.1 Kitagawa Algorithm

This algorithm is based on fixed-lag smoothing [20]. It assumes an augmented state, which is the expansion of a normal state and contains the past state as well. This expansion results in past particles repeatedly undergoing resampling based on the criteria at the current time point. Using such a mechanism, we achieve smoothing by reselecting the filtered particles; their weights are set to those at the current time point. This algorithm is quite natural from the viewpoint that defines a particle as a group including its history [14, 15, 25, 33]. Since this algorithm is based on fixed-lag smoothing, we can perform fixed-interval smoothing by extending the lag length in principle to the end of data. However, such an approach is likely to result in particle degeneracy because resampling is applied repeatedly to the past particles up to the number for all time points of the data. Particle degeneracy brings about a reduction in the effective number of particles, preventing us from maintaining adequate estimation performance. Therefore, we must avoid such a situation practically by not setting an extremely long lag length. Since the autocorrelation property of time series data is locally concentrated in general, such a setting is valid even if the lag length is short. Incidentally, this algorithm appears not to have an established name as of this writing. For example, depending on the literature, there are various names, such as SIR smoothing [37], fixed-lag smoothing [7], and poor man's smoothing [5]. Based on the notation in [27], this book refers to this algorithm as the *Kitagawa algorithm* in honor of Dr. Genshiro Kitagawa who first proposed this approach clearly.

11.1.3.2 FFBSi (Forward Filtering Backward Simulation) Algorithm

This algorithm is similar to Kalman smoothing (RTS algorithm) [10]. First, recall that particle filtering completes through time point T before smoothing. Hence, the smoothing distribution at time point T can be initialized based on the filtering distribution at time point T . Herein, let the smoothing and filtering weights be $\rho_t^{(n)}$ and $\omega_t^{(n)}$, respectively, and the smoothing and filtering realizations be $\mathbf{x}_t^{(n)}$ and $\mathbf{b}_t^{(n)}$, respectively. The initialization is as follows: the weight $\rho_T^{(n)}$ is set to $\omega_T^{(n)}$, and realization $\mathbf{b}_T^{(n)}$ is obtained through resampling $\mathbf{x}_T^{(n)}$ with a probability proportional to $\omega_T^{(n)}$. The procedure for obtaining the smoothing distribution at time point t from that at $t + 1$ is as follows; see Appendix E.4 for its detailed derivation.

Algorithm 11.3 Particle smoothing (FFBSi algorithm)

-
0. smoothing distribution at time point $t + 1$: $\left\{ \text{realizations } \mathbf{b}_{t+1}^{(n)}, \text{ weights } \rho_{t+1}^{(n)} \right\}_{n=1}^N$
 1. update procedure at time point t

- **for** $n = 1$ **to** N **do**

Compute the smoothing weights based on the state equation $p(\mathbf{x}_{t+1} | \mathbf{x}_t^{(n)})$:

$$\rho_t^{(n)} \leftarrow \omega_t^{(n)} p(\mathbf{b}_{t+1}^{(n)} | \mathbf{x}_t^{(n)}). \quad (11.4)$$

end for

- *normalization of weighs*: $\rho_t^{(n)} \leftarrow \rho_t^{(n)} / \sum_{n=1}^N \rho_t^{(n)}$

- *resampling*

Sample from the set $\{1, \dots, N\}$ with replacement, with a probability proportional to $\rho_t^{(n)}$ ($n = 1, \dots, N$), and prepare the index sequence $\mathbf{k} = \{k_1, \dots, k_n, \dots, k_N\}$ for resampling.

Relabel $\mathbf{x}_t^{(k)} = \{\mathbf{x}_t^{(k_1)}, \dots, \mathbf{x}_t^{(k_n)}, \dots, \mathbf{x}_t^{(k_N)}\}$ to obtain the realizations $\mathbf{b}_t^{(n)}$, and reset every weight $\rho_t^{(n)}$ to $1/N$.

2. smoothing distribution at time point t : $\left\{ \text{realizations } \mathbf{b}_t^{(n)}, \text{ weights } \rho_t^{(n)} \right\}_{n=1}^N$
-

Repeating the procedure of Algorithm 11.3 from $t = T - 1$ in the reverse time direction yields the smoothing distribution at every time point. The procedure in Algorithm 11.3 is consistent with the description in Sect. 6.2.5 as follows:

- Smoothing distribution: the filtering distribution $\omega_t^{(n)}$ in Eq. (11.4) is corrected based on the smoothing distribution at one time ahead.

Herein, we supplement the FFBSi algorithm.

The above algorithm gives the result of a one-shot trial for the joint smoothing distribution $p(\mathbf{x}_{0:T} | y_{1:T})$. If we obtain the marginal smoothing distribution $p(\mathbf{x}_t | y_{1:T})$, we must apply the marginalization to multiple trial results.

Equation (11.4) requires known parameters regarding the state equation.

In the above resampling step, we can reselect the filtered realizations consistently with the unevenness of $\rho_t^{(n)}$ by relabeling them that are selected according to the value of \mathbf{k} .

Compared to the previous Kitagawa algorithm, we can avoid the situation in which past particles undergo resampling directly and repeatedly. Thus, this algorithm is likely to suppress particle degeneracy.

This algorithm is also known as forward filtering backward smoothing [7]. Based on [5], this book chooses the name *FFBSi algorithm* (FFBSi; forward filtering backward simulation) to distinguish it from the FFBS described in Chap. 10.

11.2 State Estimation with Particle Filter

Regarding the implementation of the particle filter, there are both scratching and library-based approaches as with the MCMC. We again reconsider the merits and demerits of these approaches. While libraries are easy to use, it is not easy to extend their features. For example, the libraries introduced in this book do not yet have advanced features such as combination with a Kalman filter as of this writing. If we want to introduce such functions, we must modify the source code of the relevant library or revise it at a similar level. On the contrary, although the scratching approach requires effort to build features oneself, the functionality can be extended easily. The scratching and library-based approaches have their own merits and demerits, and it is best to use them properly according to the reader's purposes. However, the author believes that the scratch of the particle filter is not expensive; hence, this book focuses on the scratching approach.

11.2.1 Example: Artificial Local-Level Model

We now explain the actual buildup of the particle filter. First, we apply the particle filter to the linear Gaussian state-space model with known parameters and compare the results with those obtained using a Kalman filter. For the model and data, we use an artificial local-level model and data that are prepared using Code 9.1 in Sect. 9.2. We set the number of particles for the particle filter to 10,000 and apply the state equation to the proposal distribution for state drawing.

11.2.1.1 Filtering

We first explain particle filtering. This code is as follows.

Code 11.1

```

1 > # <<Particle filtering (from scratch) for local-level model with known parameters>>
2 >
3 > # Preprocessing
4 > set.seed(4521)
5 >
6 > # Presetting of particle filter
7 > N <- 10000           # Number of particles
8 >
9 > # Load data on artificial local-level model
10 > load(file = "ArtifitcialLocalLevelModel.RData")
11 >
```

```

12 > # *Note: Assuming that the time point of the prior distribution corresponds to one,
13 <- we regard the shifted time points (from 2 to t_max+1) as the original ones (from
14 <- 1 to t_max).
15 >
16 > # Data formatting (adding the forefront dummy corresponding to prior distribution)
17 > y <- c(NA_real_, y)
18 >
19 > # Save the index sequence for resampling at every time point
20 > k <- matrix(1:N, nrow = N, ncol = t_max+1)
21 >
22 > # Setting of prior distribution
23 > x <- matrix(NA_real_, nrow = t_max+1, ncol = N)
24 > x[1, ] <- rnorm(N, mean = mod$m0, sd = sqrt(mod$C0))
25 >
26 > # Particle (realizations)
27 > w <- matrix(NA_real_, nrow = t_max+1, ncol = N)
28 > w[1, ] <- 1 / N
29 >
30 > # Time-forward processing
31 > for (t in (1:t_max)+1){
32 +   # State equation: generate particles (realizations)
33 +   x[t, ] <- rnorm(N, mean = x[t-1, ], sd = sqrt(mod$W))
34 +
35 +   # Observation equation: updating particle (weight)
36 +   w[t, ] <- w[t-1, ] * dnorm(y[t], mean = x[t, ], sd = sqrt(mod$V))
37 +
38 +   # Normalization of weight
39 +   w[t, ] <- w[t, ] / sum(w[t, ])
40 +
41 +   # Resampling
42 +
43 +   # Index sequence for resampling
44 +   k[, t] <- sample(1:N, prob = w[t, ], replace = TRUE, size = N)
45 +
46 +   # Particle (realizations): relabeling with the resampling index sequence
47 +   x[t, ] <- x[t, k[, t]]
48 +
49 +   # Particle (weight): reset
50 +   w[t, ] <- 1 / N
51 + }
52 >
53 > # Result formatting: removing the forefront corresponding to prior distribution,
<- etc.
54 > y <- ts(y[-1])
55 > k <- k[, -1, drop = FALSE]
56 > x <- x[-1, , drop = FALSE]
57 > w <- w[-1, , drop = FALSE]
58 >
59 > # Find mean, 2.5%, and 97.5% values
60 > scratch_m <- sapply(1:t_max, function(t){
61 +   mean(x[t, ])
62 + })
63 > scratch_m_quant <- lapply(c(0.025, 0.975), function(quant){
64 +   sapply(1:t_max, function(t){
65 +     quantile(x[t, ], probs = quant)
66 +   })
67 + })
68 >
69 > # Ignore the display of following codes

```

In the above code, the number N of particles is set to 10,000, and then the data prepared in Code 9.1 are loaded. This example assumes that the time point of the prior distribution corresponds to 1; hence, the original time points $1, \dots, t_{\max}$ are shifted by +1 and treated as $2, \dots, t_{\max}+1$. For easy handling in conjunction with these time points, we add one dummy at the beginning of the observations. We then preassign the necessary areas for the index sequence for resampling, realizations for particles, and weights for particles. From the viewpoint of execution speed (especially in R), it is better to reserve an area for variables of known size. We save the index sequence for resampling at all the time points because it is necessary for executing the Kitagawa algorithm in later smoothing. The realizations and weights at time point 1 are initialized based on the prior distribution. Subsequently to the preparation, the following `for` loop describes the processing of particle filtering. Inside the loop, we describe the processing along with Algorithm 11.1. The realizations are first drawn from the state equation, whose probability distribution representation is Eq. (5.10). The weights are then updated based on the observation equation, whose probability distribution representation is Eq. (5.11). In addition, we normalize the weights and perform resampling. The resampling process is implemented using the R function `sampLe()` and yields the index sequence for resampling. This function implements the sampling method based on the multinomial distribution; hence, it is called multinomial resampling. Particle-by-particle processing is achieved through the vectorization feature of R. While we could use a loop to implement such a processing, the vectorization has an advantage in terms of execution speed; the loop in R is particularly slow. The filtering result is formatted by removing the forefront corresponding to the prior distribution, and we can then finally obtain the summary statistics. Figure 11.3 shows comparison plots between the results of the above code and those of Kalman filtering with known parameters.

As shown in Fig. 11.3, the results of the two are almost identical; the graph lines overlap and cannot be distinguished from each other.

Through the above simple example, we have confirmed that the scratching code of particle filtering can estimate the linear Gaussian state-space model accurately.

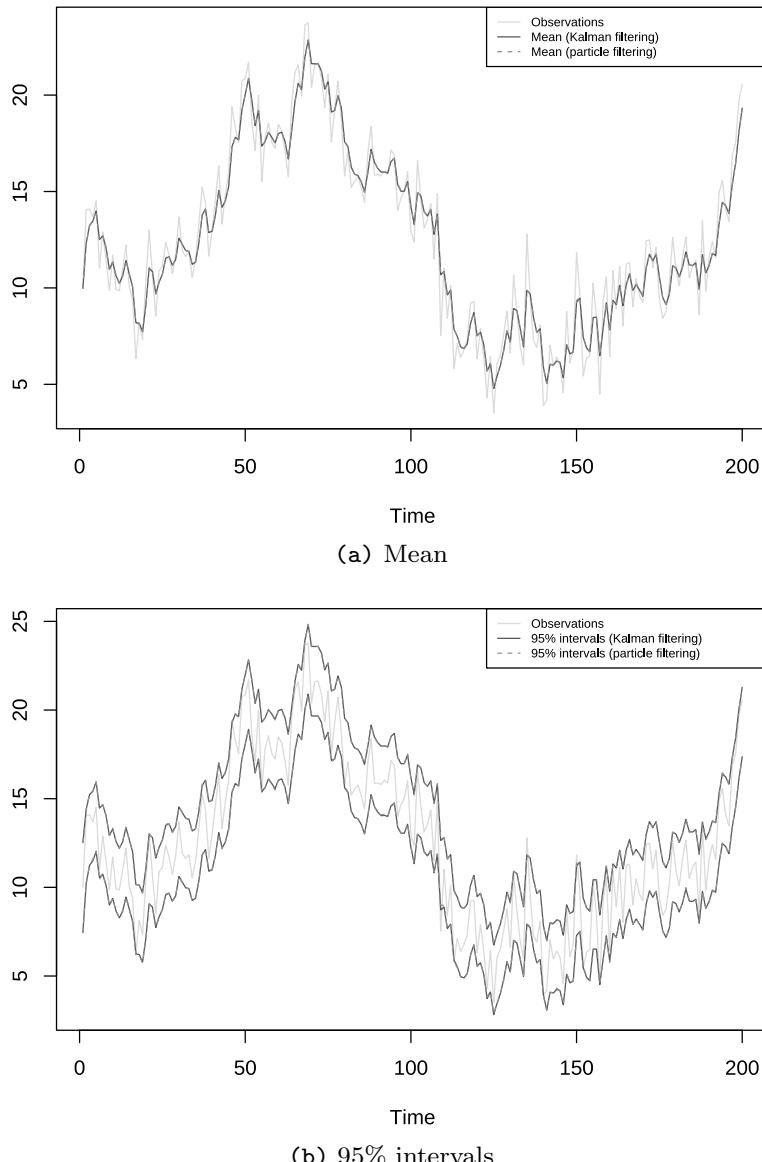


Fig. 11.3 Particle filtering from scratch and Kalman filtering (linear Gaussian state-space model with known parameters)

11.2.1.2 Prediction

We now confirm particle prediction. This code is as follows.

Code 11.2

```

70 > # <<Particle prediction (from scratch) for local-level model with known parameters>>
71 >
72 > # Preprocessing
73 > set.seed(4521)
74 >
75 > # Append data area at the future time points
76 > x <- rbind(x, matrix(NA_real_, nrow = 10, ncol = N))
77 > w <- rbind(w, matrix(NA_real_, nrow = 10, ncol = N))
78 >
79 > # Time-forward processing
80 > for (t in t_max+(1:10)){
81 +   # State equation: generate particles (realizations)
82 +   x[t, ] <- rnorm(N, mean = x[t-1, ], sd = sqrt(mod$W))
83 +
84 +   # Update particle (weight)
85 +   w[t, ] <- w[t-1, ]
86 + }
87 >
88 > # Find mean, 2.5%, and 97.5% values
89 > scratch_a      <- sapply(t_max+(1:10), function(t){
90 +   mean(x[t, ])
91 + })
92 > scratch_a_quant <- lapply(c(0.025, 0.975), function(quant){
93 +   sapply(t_max+(1:10), function(t){
94 +     quantile(x[t, ], probs = quant)
95 +   }))
96 + }
97 >
98 > # Ignore the display of following codes

```

The above code continues from the former Code 11.1. We first preassign the predicting area regarding future realizations and weights for ten time points from the end of the observations. Subsequent to this preparation, the following `for` loop describes the processing of particle prediction. The processing along with Algorithm 11.2 is described inside the loop. The realizations are first drawn from the state equation, whose probability distribution representation is Eq. (5.10). We continue applying the same value to the weights while the loop. We finally obtain the summary statistics regarding the particles for future time points. Figure 11.4 shows plots comparing the results obtained from the above code with those obtained from Kalman prediction with known parameters.

As shown in Fig. 11.4, we see that the two results are almost identical; the graph lines overlap and cannot be distinguished from each other.

Through the simple example above, we have confirmed that the scratching code of particle prediction can estimate the linear Gaussian state-space model accurately.

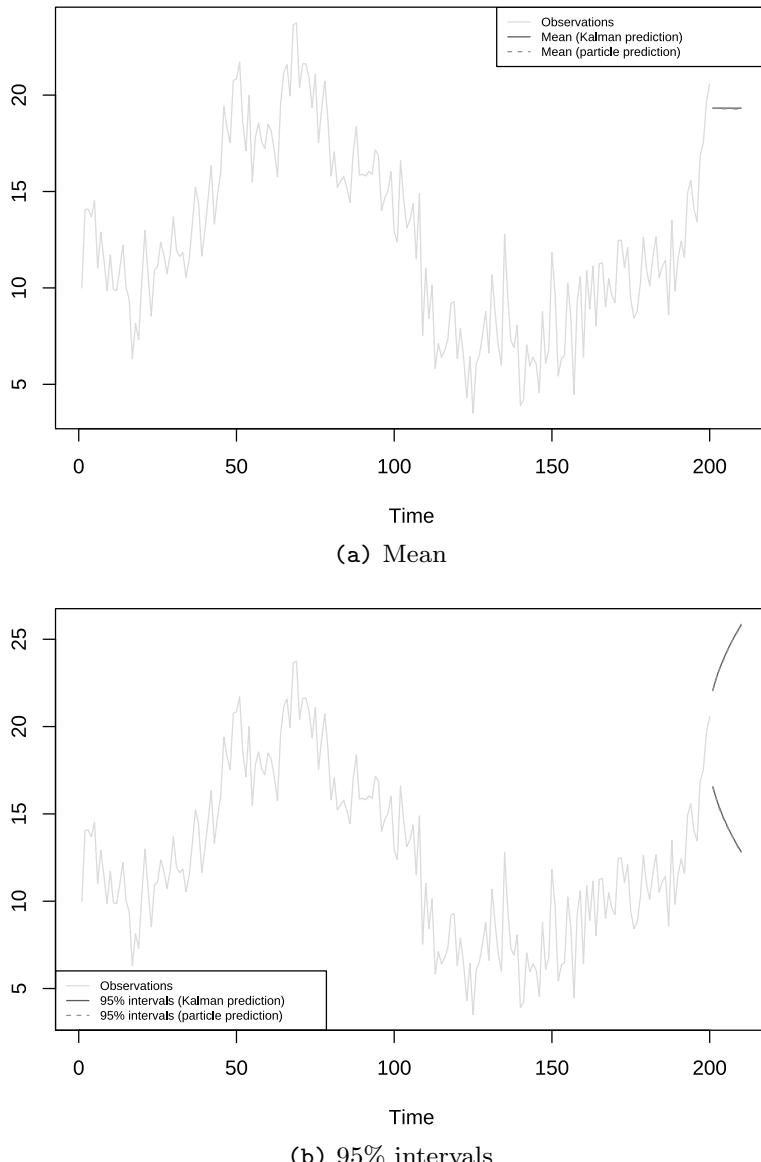


Fig. 11.4 Particle prediction from scratch and Kalman prediction (linear Gaussian state-space model with known parameters)

11.2.1.3 Smoothing

Finally, we explain particle smoothing, beginning with the Kitagawa algorithm. This code is as follows.

Code 11.3

```

99 > # <<Particle smoothing (from scratch and Kitagawa algorithm) for local-level model
100> # with known parameters>
101>
102> # Preprocessing
103> set.seed(4521)
104>
105> # User-defined function obtaining indices to reselect filtering particles
106> # considering future information
107> smoothing_index <- function(t_current){
108+   # Index sequence at current time t_current
109+   index <- 1:N
110+
111+   # Virtual repetition of resampling from t_current + 1 to t_max
112+   for (t in (t_current+1):t_max){      # Upper limitation leads to fixed-lag
113+     smoothing
114+     index <- index[k[, t]]
115+   }
116+
117>
118> # Reselect filtering particles considering future information
119> ki <- sapply(1:(t_max-1), function(t){ x[t, smoothing_index(t)] })
120> ki <- t(cbind(ki, x[t_max, ]))          # Add smoothing distribution at the last time
121# point
122>
123> scratch_s           <- sapply(1:t_max, function(t){
124+   mean(ki[t, ])
125+ })
126> scratch_s_quant    <- lapply(c(0.025, 0.975), function(quant){
127+   sapply(1:t_max, function(t){
128+     quantile(ki[t, ], probs = quant)
129+   })
130+ })
131>
132> # Ignore the display of following codes

```

The above code continues from the former Code 11.2. The Kitagawa algorithm repeats resampling for the past particles based on the criteria at the current time point. This processing is performed using the user-defined function `smoothing_index()` after filtering. Since the index sequence for resampling has been saved for every time point through the filtering, this function can later be used to reproduce the process of applying resampling to the index at a particular time point virtually and repeatedly. Based on the return values from this function, we reselect the filtering particles for smoothing at every time point except for the final time point, and save the results in `ki`. The smoothing distribution at the final time point is equal to the filtering distribution at that time point; hence, we can set the last value without using this function. We can finally obtain the summary statistics on the smoothing results. Figure 11.5 shows comparison plots between the results from the above code and those from Kalman smoothing with known parameters.

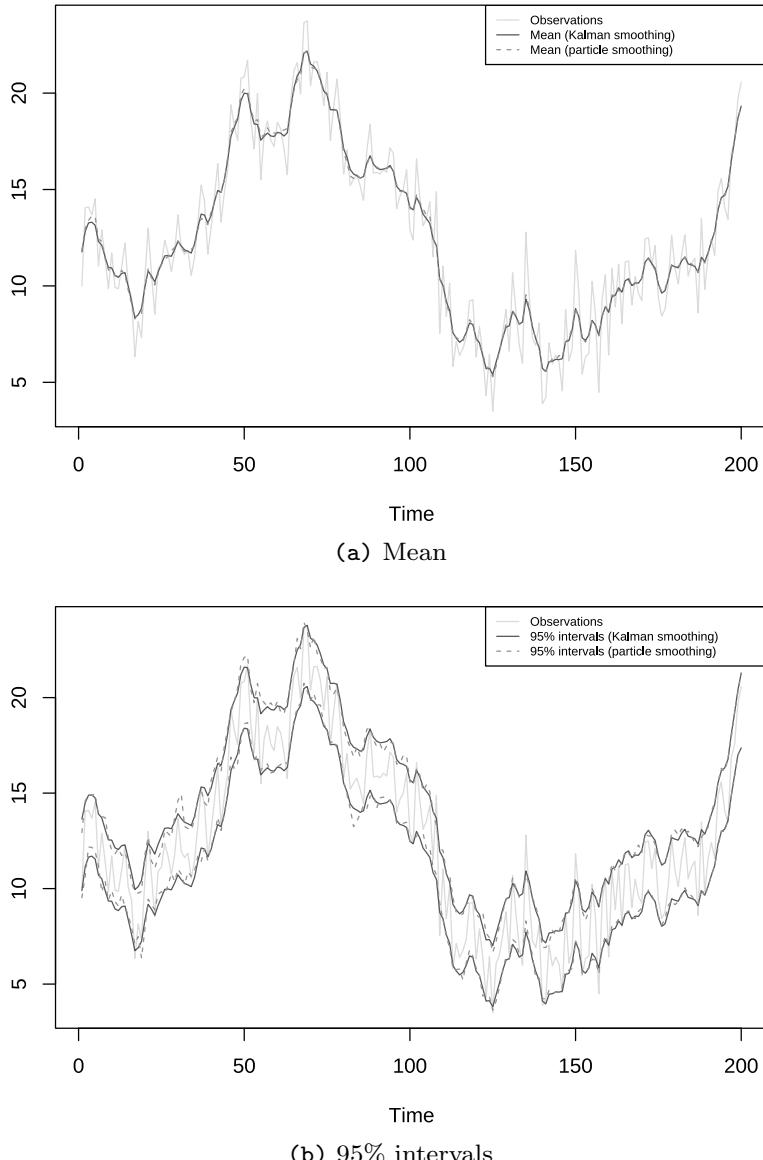


Fig. 11.5 Particle smoothing from scratch (Kitagawa algorithm) and Kalman smoothing (linear Gaussian state-space model with known parameters)

As shown in Fig. 11.5, we see that the two results are almost identical. However, looking carefully, we see slight differences. These difference are considered to be caused by an increase in the Monte Carlo error due to smoothing in addition to filtering.

We then confirm the FFBSi algorithm. This code is as follows.

```

Code 11.4
133 > # <<Particle smoothing (from scratch and FFBSi algorithm) for local-level model
134 >   <<with known parameters>>
135 >
136 > # Preprocessing
137 > set.seed(4521)
138 >
139 > # Maximum value of trial (path)
140 > path_max <- 500
141 >
142 > # Setting of the progress bar
143 > progress_bar <- txtProgressBar(min = 2, max = path_max, style = 3)
144 >
145 > # Smoothing particles (realizations)
146 > b <- array(NA_real_, dim = c(t_max, N, path_max))
147 >
148 > # Smoothing particles (weight)
149 > rho <- matrix(NA_real_, nrow = t_max, ncol = N)
150 > rho[t_max, ] <- w[t_max, ]
151 >
152 > # By trial (path)
153 > for (path in 1:path_max){
154 +   # Display progress bar
155 +   setTxtProgressBar(pb = progress_bar, value = path)
156 +
157 +   # Initialize realizations for smoothing distribution at t_max
158 +   b[t_max, , path] <- sample(x[t_max, ],
159 +                               prob = w[t_max, ], replace = TRUE, size = N)
160 +
161 +   # Time-reverse processing
162 +   for (t in (t_max-1):1){
163 +     # Weight
164 +     rho[t, ] <- w[t, ] * dnorm(b[t+1, , path], mean = x[t, ], sd = sqrt(mod$W))
165 +
166 +     # Normalization of weight
167 +     rho[t, ] <- rho[t, ] / sum(rho[t, ])
168 +
169 +     # Resampling
170 +
171 +     # Find indices to reselect filtering particles considering future information
172 +     FFBSi_index <- sample(1:N, prob = rho[t, ], replace = TRUE, size = N)
173 +
174 +     # Reselect filtering particles considering future information
175 +     b[t, , path] <- x[t, FFBSi_index]
176 +
177 +     # Reset weight
178 +     rho[t, ] <- 1 / N
179 +   }
180 + }
181 > ====== | 100%
182 > # Find mean, 2.5%, and 97.5% values
183 > scratch_s <- sapply(1:t_max, function(t){
184 +   mean(b[t, ,])
185 + })
186 >
187 > scratch_s_quant <- lapply(c(0.025, 0.975), function(quant){
188 +   sapply(1:t_max, function(t){
189 +     quantile(b[t, ,], probs = quant)
190 +   })
191 + })
192 >
193 > # Ignore the display of following codes

```

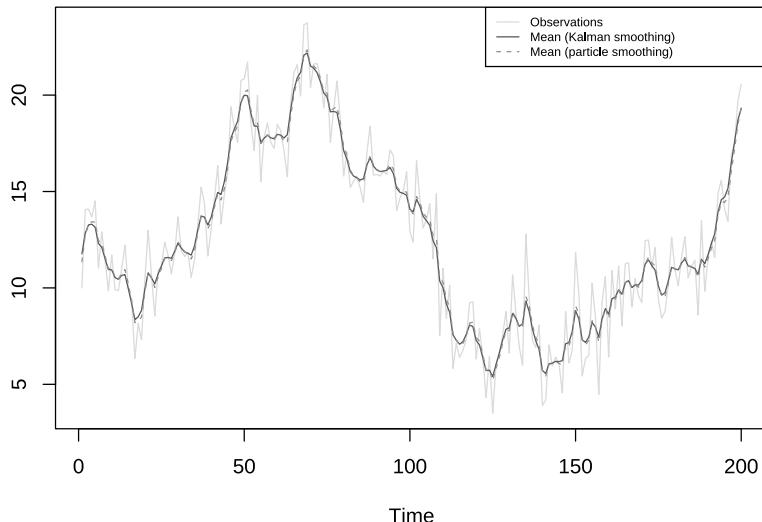
The above code continues from the former Code 11.3. First, we set the maximum number of trials for marginalizing the joint smoothing distribution to 500. We display the progress bar because the repetition of each trial takes substantial time. Next, we preassign the necessary areas for realizations and weights in the smoothing distribution. Subsequently to this preparation, the following nested `for` loop describes the primary processing of particle smoothing. The first loop repeats the trial, and immediately after the loop is entered, the realizations of the smoothing distribution at time point t_{max} are initialized by resampling the filtering distribution at time point t_{max} . In the second loop, the processing of particle smoothing (FFBSi algorithm) is described along with Algorithm 11.3. Specifically, the smoothing weights are calculated based on the state equation, whose probability distribution representation is Eq. (5.10). In addition, we normalize the weights and perform resampling. Through this resampling, the filtering particles are reselected in consideration of future information. In this manner, we first obtain the joint smoothing distribution for the number of trials, and then derive the summary statistics regarding the marginal smoothing distribution. Such a derivation is computationally expensive: even if we obtain the mean only at a particular time point, we need the calculation for 5,000,000 samples = 10,000 particles \times 500 trials. Figure 11.6 shows comparison plots between the results of the above code with those of Kalman smoothing with known parameters.

As shown in Fig. 11.6, we see that the two results are almost identical. However, confirming carefully, we can recognize a slight difference. As with the Kitagawa algorithm, this difference is considered to be caused by an increase in the Monte Carlo error due to smoothing in addition to filtering.

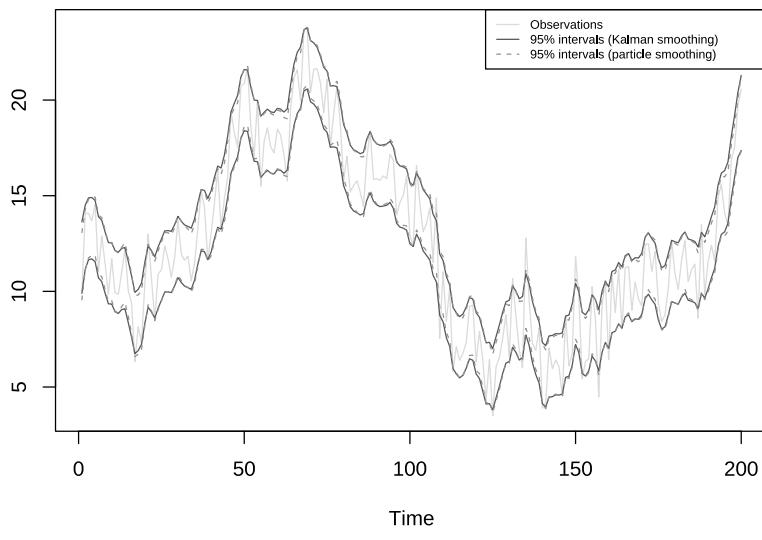
Finally, we compare the Kitagawa algorithm and the FFBSi algorithm. We intentionally reduce the number of particles to 150 to make it easier to confirm the particle degeneracy. Figure 11.7 shows the comparison of 95% values for each smoothing distribution.

From Fig. 11.7, we easily recognize the particle degeneracy in the Kitagawa algorithm. We also see that the more backward time goes, the more the results deviate from those obtained from Kalman smoothing: the 2.5% and 97.5% values become the same before approximately the time point 110. On the contrary, the particle degeneracy is more suppressed in the FFBSi algorithm, and the deviation from the result obtained from Kalman smoothing is also mitigated. Since this example intentionally reduces the number of particles, we clearly recognize such a difference. If the number of particles is sufficiently large and lag length is not excessively long, the Kitagawa algorithm yields an adequate result.² The Kitagawa algorithm has a further advantage of being applicable even when the parameters regarding the state equation are unknown; hence, it is better to select the algorithm to be used depending on the problem.

² In the author's experience, to obtain stable results, the number of particles must be greater than 1,000, whereas the lag length must be less than 100. However, such conditions depend on the problem, and hence, the author recommends that readers verify the results with a variety of settings.

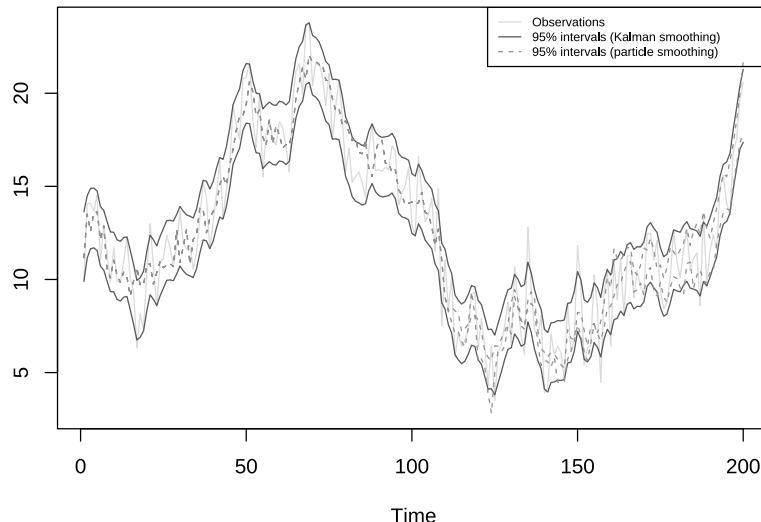


(a) Mean

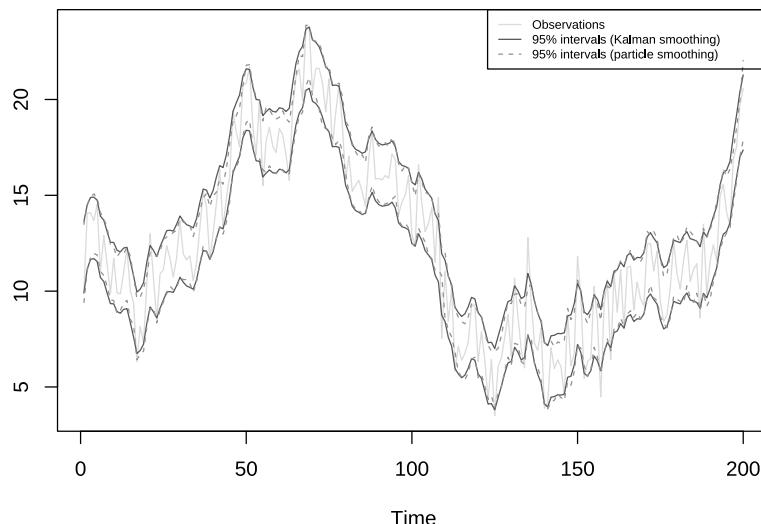


(b) 95% intervals

Fig. 11.6 Particle smoothing from scratch (FFBSI algorithm) and Kalman smoothing (linear Gaussian state-space model with known parameters)



(a) Kitagawa algorithm



(b) FFBSi algorithm

Fig. 11.7 Kitagawa and FFBSi algorithms (linear Gaussian state-space model with known parameters)

Through the above simple example, we have confirmed that the scratching code of particle smoothing can accurately estimate the linear Gaussian state-space model.

11.2.2 Attention to Numerical Computation

We have thus far confirmed the fundamental behavior of the particle filter. While all codes were implemented simply, they did not pose any concern regarding numerical computation because only simple problems were considered. However, the implementation can affect estimation performance when the problem becomes more complicated. We introduce below some techniques for minimizing degrading in estimation performance.

11.2.2.1 Calculation in the Log-Domain

The weights of a particle filter can be very small. In such a case, a naive implementation tends to cause underflow. For this reason, dealing with the weight values in the log-domain to the extent possible can assist in suppressing accuracy deterioration in the numerical calculation. However, when operations in the linear-domain cannot be avoided, we inevitably take the exponent of the log-weights to convert them into the linear domain. Regarding the normalization of log-values in the linear domain, there is a well-known technique that utilizes scaling for suppressing the underflow as much as possible. This technique is known as *logsumexp* and is adopted in this book. Specifically, letting the vector of the logarithmic values be \mathbf{l} , its normalization in the linear domain is expressed by $\exp(\mathbf{l}) / \sum \exp(\mathbf{l})$. Thus, we can convert the result again into the log-domain as follows:

$$\log \left(\frac{\exp(\mathbf{l})}{\sum \exp(\mathbf{l})} \right) = \log (\exp(\mathbf{l})) - \log \left(\sum \exp(\mathbf{l}) \right)$$

let l_{\max} be the maximum value of \mathbf{l}

$$= \mathbf{l} - \log \left(\sum \exp(\mathbf{l} - l_{\max} + l_{\max}) \right)$$

the last term l_{\max} can be factored out from \sum

$$\begin{aligned} &= \mathbf{l} - \left\{ \log \left(\sum \exp(\mathbf{l} - l_{\max}) \right) + \log (\exp(l_{\max})) \right\} \\ &= \mathbf{l} - l_{\max} - \log \left(\sum \exp(\mathbf{l} - l_{\max}) \right) \end{aligned}$$

furthermore, we divide \mathbf{l} into l_{\max} and other elements

$$\begin{aligned} &= \mathbf{l} - l_{\max} - \log \left(\sum \exp (\{l_{\max}, \mathbf{l}_{\text{Except max}}\} - l_{\max}) \right) \\ &= \mathbf{l} - l_{\max} - \log \left(\sum \exp (\{0, \mathbf{l}_{\text{Except max}} - l_{\max}\}) \right) \\ &= \mathbf{l} - l_{\max} - \log \left(1 + \sum \exp (\mathbf{l}_{\text{Except max}} - l_{\max}) \right). \end{aligned} \quad (11.5)$$

The code for normalization in the linear domain based on Eq. (11.5) is as follows.

Code 11.5

```

1 > # <<Logsumexp>>
2 >
3 > # Normalization in the linear domain (input value: unnormalized log vector, return
4 >   value: normalized log vector)
5 > normalize <- function(l){
6 +   # Number for which the input log vector takes its maximum value
7 +   max_ind <- which.max(l)
8 +
9 +   # Suppress underflow as much as possible by applying scaling
10 +  return(
11 +    l - l[max_ind] -
12 +    log1p(sum(exp(l[-max_ind] - l[max_ind]))))
13 + )
+ }
```

The above code defines a user-defined function `normalize()` for normalization in the linear domain. The argument `l` of this function is a logarithmic vector before normalization, and the return value is a logarithmic vector after normalization. Because R provides a function `log1p()` for calculating $\log(1+x)$ accurately even under the condition $|x| \ll 1$, we use this function. While the accuracy of `log1p(x)` can be degraded compared to the usual `log(1+x)` when `x` is close to -1 , this function can be used without problems because the `x` in this problem never becomes negative. The implementation in the above code refers to the R-help mailing list (<https://stat.ethz.ch/pipermail/r-help/2011-February/269205.html>).

11.2.2.2 Resampling

We have used the R function `sample()` for resampling in the scratching code of a particle filter. This function implements *multinomial resampling* (also known as random resampling), and an uncertain factor is included in the results. However, unlike the typical usage, the resampling in the particle filter does not require such an uncertain factor because it is used only to increase or decrease the number of realizations according to their weights. Resampling without uncertainty is rather useful for

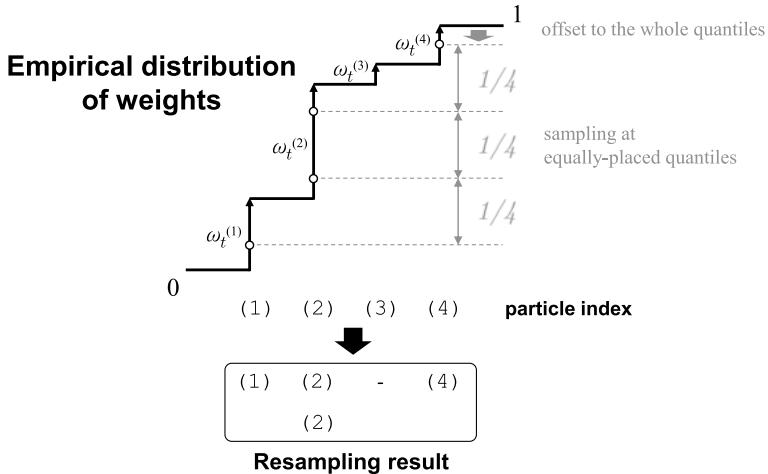


Fig. 11.8 Systematic resampling example

improving estimation accuracy. For this reason, various methods have been proposed to suppress the effects of such Monte Carlo errors. We introduce *systematic resampling* (also known as deterministic stratified resampling) [20], *stratified resampling* [20], and *residuals resampling* [29] below.

Systematic resampling performs sampling at equal intervals on the empirical distribution of the weights, as shown in the example in Fig. 11.8.

While it is common to use a random number for the offset to whole quantiles, there is another variation that uses a deterministic offset. While presorting the weights can further improve the bias of the results, it is not a necessary process.

Stratified resampling is similar to systematic resampling. This method performs sampling at unequally spaced quantiles on the empirical distribution of the weights. The unequal space comes from the addition of a random offset to each quantile.

The concept of residuals resampling is as follows. While the number of whole particles multiplied by the weights is not necessarily an integer, this value indicates the effective number for each particle after resampling. Thus, we can first reserve the integer part of that value deterministically and then apply any resampling method to the aggregation of the remaining fractional parts. These are combined as the final result. Compared with systematic resampling, reserving the integer part of the effective particle number is the same and the handling of the fractional parts is different.

Which resampling method is best for the particle filter? The answer depends on the problem and is not fixed [4]. However, the systematic resampling appears to be generally known [7]. Thus, this book uses systematic resampling as mentioned below. The code for systematic resampling is as follows.

Code 11.6

```

1 > # <<Systematic resampling>>
2 >
3 > # User-defined function for systematic resampling (N: number of particles, w:
   ← standardized log weight vector)
4 > sys_resampling <- function(N, w){
5 +   # Restore w to the linear domain value
6 +   w <- exp(w)
7 +
8 +   # Define the step function returning the particle number according to the
   ← empirical cumulative distribution of the weight (y has one more element than x)
9 +   sfun <- stepfun(x = cumsum(w), y = 1:(N+1))
10 +
11 +   # Sampling at even interval (applying offset to all the quantiles with runif())
12 +   sfun((1:N - runif(n = 1)) / N)
13 + }
```

The code above describes a user-defined function `sys_resampling()` for systematic resampling. The argument `N` is the number of particles, and the argument `w` is the normalized logarithmic weights vector. This function uses a generic R function `stepfun()` to generate a step-like function. We set the empirical distribution of the weights to the argument `x` and particle identity number to the argument `y` to specify a function that returns the identity number according to the distribution. The return object from the `stepfun()` is a function, and it is stored in the `sfun`. Resampling is finally achieved by setting the equally spaced quantiles to the argument of the prepared function `sfun()`.

11.2.2.3 Improved Code

We improve Code 11.1 based on the abovementioned techniques: calculation in the log-domain and effective resampling. This code is as follows.

Code 11.7

```

1 > # <<Particle filtering (improved version) for local-level model with known
2   <- parameters>>
3 >
4 > # Preprocessing
5 > set.seed(4521)
6 >
7 > # Presetting of particle filter
8 > N <- 10000           # Number of particles
9 >
10 > # Load data on artificial local-level model
11 > load(file = "ArtificialLocalLevelModel.RData")
12 >
13 > # *Note: Assuming that the time point of the prior distribution corresponds to one,
14   <- we regard the shifted time points (from 2 to t_max+1) as the original ones (from
15   <- 1 to t_max).
16 >
17 > # Data formatting (adding the forefront dummy corresponding to prior distribution)
18 > y <- c(NA_real_, y)
19 >
20 > # Save the index sequence for resampling at every time point
21 > k <- matrix(1:N, nrow = N, ncol = t_max+1)
22 >
23 > # Setting of prior distribution
24 > x <- matrix(NA_real_, nrow = t_max+1, ncol = N)
25 > x[1, ] <- rnorm(N, mean = mod$m0, sd = sqrt(mod$C0))
26 >
27 > # Particle (realizations)
28 > w <- matrix(NA_real_, nrow = t_max+1, ncol = N)
29 > w[1, ] <- log(1 / N)
30 >
31 > # Time-forward processing
32 > for (t in 1:t_max+1){
33 +   # State equation: generate particles (realizations)
34 +   x[t, ] <- rnorm(N, mean = x[t-1, ], sd = sqrt(mod$W))
35 +
36 +   # Observation equation: updating particle (weight)
37 +   w[t, ] <- w[t-1, ] +
38 +             dnorm(y[t], mean = x[t, ], sd = sqrt(mod$V), log = TRUE)
39 +
40 +   # Normalization of weight
41 +   w[t, ] <- normalize(w[t, ])
42 +
43 +   # Resampling
44 +
45 +   # Index sequence for resampling
46 +   k[, t] <- sys_resampling(N = N, w = w[t, ]) # Systematic resampling
47 +
48 +   # Particle (realizations): relabeling with the resampling index sequence
49 +   x[t, ] <- x[t, k[, t]]
50 +
51 +   # Particle (weight): reset
52 +   w[t, ] <- log(1 / N)
53 +
54 > # Ignore the display of following codes

```

The shaded part indicates the difference between the above code and Code 11.1. While the display is ignored, we can obtain the same result as in Fig. 11.3. The effect of this improvement is not immediately apparent because the local-level model with known parameters is a simple model. We emphasize that this improvement is more useful for a complex model, and this book regards Code 11.7 as reference code for particle filtering hereafter.

We supplement the execution performance of the particle filter. The author considers that the vectorization feature of R can achieve sufficient performance as far as the examples in this book. However, if the number of particles increases further, we might need further implementation improvement. While the typical approaches involve the application of parallel processing and porting to a high-performance language, it would be safer to compare the performance before and after the change. Especially, pay attention to parallel processing because it contains traps such as overhead. Incidentally, although the author has examined the application of parallel processing and library **Rcpp** to the examples in this book, they were found to be unable to achieve substantial benefit because the number of particles is not so large.

11.3 Use of Library

In recent years, use of general-purpose libraries capable of executing a particle filter has increased. In comparison with to the MCMC library, there appears to be no standard one as of this writing. In addition, the implementation of extended features depends on the library. However, because libraries typically have the advantage of being easy to use, some libraries within the author's knowledge are briefly introduced below. Table 11.1 summarizes the features of some R libraries as well as providing the author's opinion. The author chooses general-purpose libraries that have been updated since 2020.

See also Appendix F for simple examples of **pomp** and **NIMBLE**. These libraries have their own advantages and disadvantages; hence, it would be better ultimately to choose one based on a reader's preferences.

Table 11.1 Major R libraries that can execute a particle filter

Features	pomp	NIMBLE
Registration to CRAN	Yes	Yes
Model definition	R or C (described within R code)	BUGS extended language (described within R code)
Algorithm modification	Correction at source code level	R-like proprietary language enables the extension.
Care for execution speed	Compiled in C at runtime (if using C)	Compiled in C++ at runtime
Others	A lot of related functions are provided.	While focused on MCMC, the particle filter has also been implemented after version 0.5-1.

11.4 Estimation Example in General State-Space Model

The example in the previous section involved the case of a linear Gaussian state-space model with known parameters; we can actually solve such a problem using a Kalman filter. This section examines a well-known nonlinear benchmark model as a general state-space model. Recall that we once examined this model through data (d) in Chap. 4.

11.4.1 Example: A Well-Known Nonlinear Benchmark Model

This model was initially introduced in [2] and has been formulated in several papers as a typical example of a nonlinear model in time series analysis. The state and observation equations for this model are as follows:

$$x_t = \frac{1}{2}x_{t-1} + \frac{25x_{t-1}}{1+x_{t-1}^2} + 8 \cos(1.2t) + w_t, \quad w_t \sim \mathcal{N}(0, W), \quad (11.6)$$

$$y_t = \frac{x_t^2}{20} + v_t, \quad v_t \sim \mathcal{N}(0, V), \quad (11.7)$$

where we assume $x_0 \sim \mathcal{N}(m_0, C_0)$ for the prior distribution. While the state value alternates from time to time between positive and negative areas based on the state equation (11.6), information on the sign of the state is lost in the observation equation (11.7). Although we have already analyzed the data of this model using the exploratory method in Chap. 4, this chapter reconsiders it in the general state-space model.

We first explain the data generative process. This code is as follows.

Code 11.8

```

1 > # <<Well-known benchmark model>>
2 >
3 > # Preprocessing
4 > set.seed(23)
5 > library(dlm)
6 >
7 > # Set the parameters
8 > W <- 1
9 > V <- 2
10 > m0 <- 10
11 > C0 <- 9
12 >
13 > # Nonlinear function in state equation
14 > f <- function(x, t){
15 +   1/2 * x + 25 * x / (1 + x^2) + 8 * cos(1.2 * t)
16 + }
17 >
18 > # Nonlinear function in observation equation
19 > h <- function(x){
20 +   x^2 / 20
21 + }
22 >
23 > # Time series length
24 > t_max <- 100
25 >
26 > # Initialization of data (+1 considering prior distribution)
27 > x_true <- rep(NA_real_, times = t_max + 1)
28 > y <- rep(NA_real_, times = t_max + 1)
29 >
30 > # Data generation
31 > # *Note: Assuming that the time point of the prior distribution corresponds to one,
32 > # we regard the shifted time points (from 2 to t_max+1) as the original ones (from
33 > # 1 to t_max).
34 > x_true[1] <- m0                      # Set mean to the realization from prior
35 > # distribution
36 > for (it in (1:t_max)+1){               # Time update
37 +   # State equation
38 +   x_true[it] <- f(x_true[it - 1], it) + rnorm(n = 1, sd = sqrt(W))
39 + }
40 >
41 > # Observation equation
42 > y[it] <- h(x_true[it]) + rnorm(n = 1, sd = sqrt(V))
43 >
44 >
45 > # Data formatting (removing the forefront corresponding to prior distribution)
46 > x_true <- x_true[-1]
47 > y <- y[-1]
48 >
49 > # Ignore the display of following codes

```

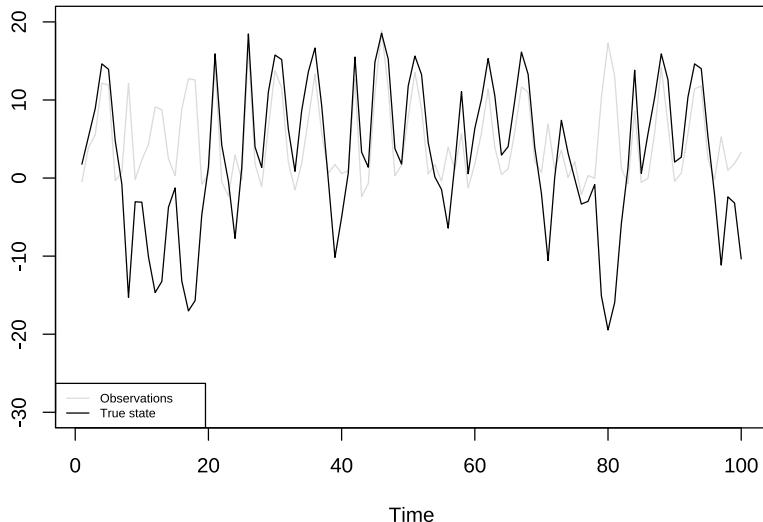


Fig. 11.9 A well-known benchmark model

The above code assumes the case with parameters $W = 1$, $V = 2$, $m_0 = 10$, and $C_0 = 9$. The nonlinear functions f and h in the state and observation equations, respectively, are described as user-defined functions. We set the time series length t_max to 100. The realization of the prior distribution is set to its mean parameter. Assuming that the time point for the prior distribution corresponds to 1, we regard the shifted time points $(2, \dots, t_max+1)$ as the original ones $(1, \dots, t_max)$ when generating the data. After data generation, the result is formatted by removing the forefront corresponding to the prior distribution. The data and related objects are saved together as `BenchmarkNonLinearModel.RData` at the end of the code. Figure 11.9 shows the plots of the observations and the true value of the state.

As can be seen from Fig. 11.9, even if the true value of the state falls in the negative area, the observations always comprise positive values. Thus, we cannot determine whether the state is truly positive or negative through the observations alone.

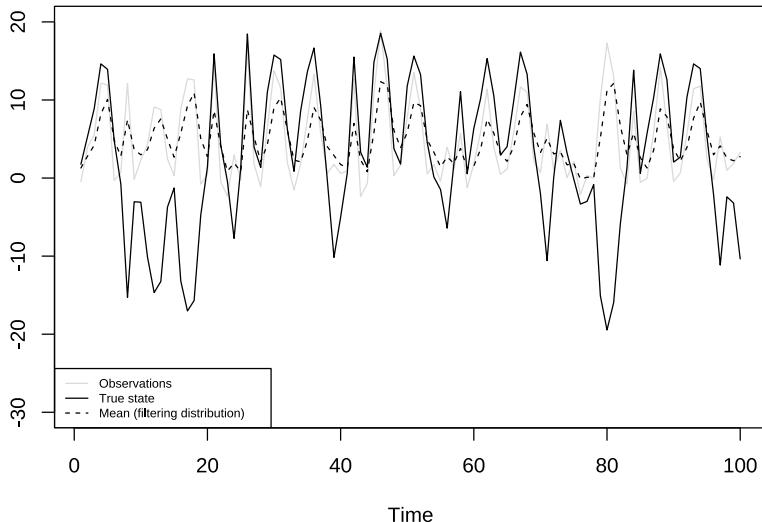


Fig. 11.10 Analyzing the well-known benchmark model with local-level model

We then analyze these data using the local-level model. Since the local-level model is a linear model, applying it to a nonlinear model is unreasonable. Thus, we expect that the estimation cannot be appropriately performed as in the case of Chap. 4. While we ignore the code display, Fig. 11.10 shows a plot of the filtering result when the true values of parameters W , V , m_0 , and C_0 are set.

As can be seen from Fig. 11.10, the result cannot achieve an appropriate estimation accuracy. In particular, the mean of the filtering distribution always takes positive values. As mentioned in Sect. 4.7, we require an approach based on the general state-space model to handle such a nonlinear model properly.

11.4.2 Application of a Particle Filter

We now analyze the well-known nonlinear benchmark model using a particle filter. For the particle filter, we set the number of particles to 10,000 and apply the state equation to the proposal distribution for drawing the state. This code is as follows.

Code 11.9

```

1 > # <<Particle filtering for a well-known nonlinear benchmark model>>
2 >
3 > # Preprocessing
4 > set.seed(4521)
5 >
6 > # Presetting of particle filter
7 > N <- 10000           # Number of particles
8 >
9 > # *Note: Assuming that the time point of the prior distribution corresponds to one,
  <- we regard the shifted time points (from 2 to t_max+1) as the original ones (from
  <- 1 to t_max).
10 >
11 > # Data formatting (adding the forefront dummy corresponding to prior distribution)
12 > y <- c(NA_real_, y)
13 >
14 > # Save the index sequence for resampling at every time point
15 > k <- matrix(1:N, nrow = N, ncol = t_max+1)
16 >
17 > # Setting of prior distribution
18 >
19 > # Particle (realizations)
20 > x <- matrix(NA_real_, nrow = t_max+1, ncol = N)
21 > x[1, ] <- rnorm(N, mean = m0, sd = sqrt(C0))
22 >
23 > # Particle (weight)
24 > w <- matrix(NA_real_, nrow = t_max+1, ncol = N)
25 > w[1, ] <- log(1 / N)
26 >
27 > # Time-forward processing
28 > for (t in (1:t_max)+1){
29 +   # State equation: generate particles (realizations)
30 +   x[t, ] <- rnorm(N, mean = f(x = x[t-1, ], t = t), sd = sqrt(W))
31 +
32 +   # Observation equation: updating particle (weight)
33 +   w[t, ] <- w[t-1, ] +
34 +     dnorm(y[t], mean = h(x = x[t, ]), sd = sqrt(V), log = TRUE)
35 +
36 +   # Normalization of weight
37 +   w[t, ] <- normalize(w[t, ])
38 +
39 +   # Resampling
40 +
41 +   # Index sequence for resampling
42 +   k[, t] <- sys_resampling(N = N, w = w[t, ])    # Systematic resampling
43 +
44 +   # Particle (realizations): relabeling with the resampling index sequence
45 +   x[t, ] <- x[t, k[, t]]
46 +
47 +   # Particle (weight): reset
48 +   w[t, ] <- log(1 / N)
49 +
50 >
51 > # Ignore the display of following codes

```

The content of the code above is almost the same as that of Code 11.7. The principal differences are that the state and observation equations are nonlinear; the shaded part indicates the difference. The shaded codes are described based on the

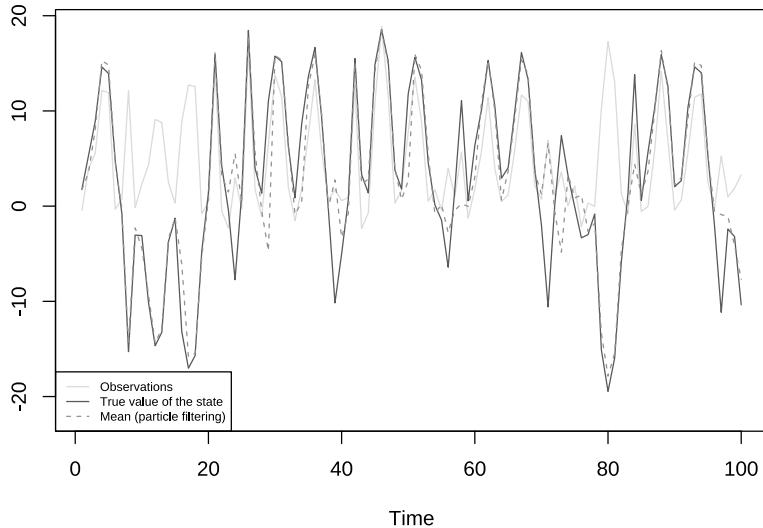


Fig. 11.11 Analyzing a well-known benchmark model using a particle filter

following probability distribution representations for the state equation (11.6) and the observation equation (11.7):

$$\begin{aligned} p(\mathbf{x}_t \mid \mathbf{x}_{t-1}) &= \mathcal{N}\left(\frac{1}{2}x_{t-1} + \frac{25x_{t-1}}{1+x_{t-1}^2} + 8 \cos(1.2t), W\right) \\ &= \mathcal{N}(f(x_{t-1}), W), \end{aligned} \quad (11.8)$$

$$\begin{aligned} p(y_t \mid \mathbf{x}_t) &= \mathcal{N}\left(\frac{x_t^2}{20}, V\right) \\ &= \mathcal{N}(h(x_t), V). \end{aligned} \quad (11.9)$$

Fig. 11.11 shows a plot of the filtering results obtained using the above code.

Figure 11.11 yields more appropriate estimation results in comparison with Fig. 11.10. An annoying aspect of this model is that the observations alone do not reveal which of the positive or negative area the true state exists in. Regarding this point, the particle filter can consider the nonlinear property; hence, it is possible to improve the distinction accuracy for the positive or negative area. For further detailed examination of this result, we plot the figure showing the temporal transition of the filtering distribution; see Fig. 11.12.³

In Fig. 11.12, the thick line shows the true value of the state and three-dimensionally overlaid mountain shows the filtering distribution. As can be seen from Fig. 11.12, the shape of the distribution is sometimes bimodal and is divided into positive and

³ We use the R function `density()` and MATLAB to estimate the density and draw a graph, respectively.

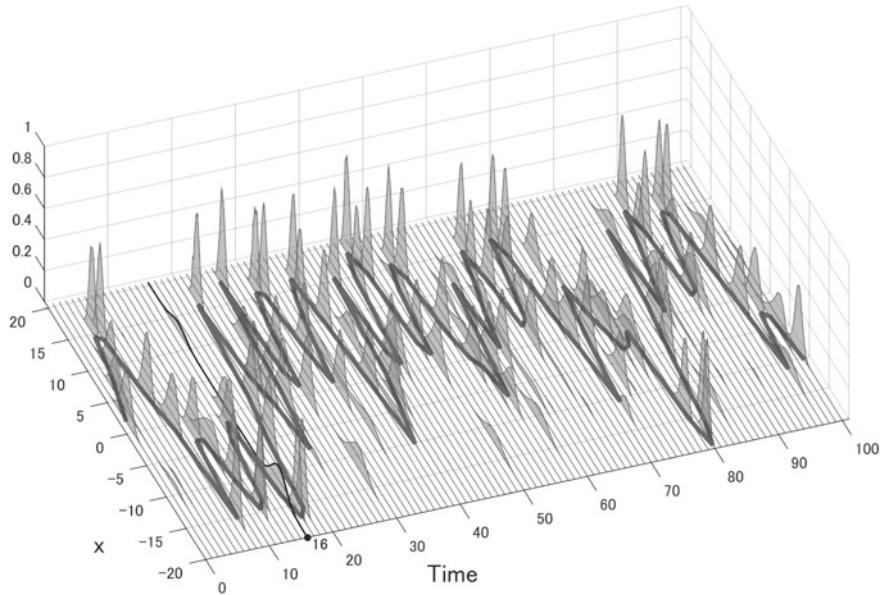


Fig. 11.12 Temporal transition of the filtering distribution for a well-known benchmark model

negative areas. This result indicates the situation wherein the filtering estimates for the state might fall in either a positive or negative area. In such a situation, while the Kalman filter forcibly regards the filtering distribution as a unimodal normal distribution, the particle filter can track the shape of the distribution as it is and estimate it more appropriately. For example, at the time point 16, we see that such a property makes it possible to estimate the state in the negative area correctly.

11.5 Technique for Improving Estimation Accuracy

According to the above description, it is actually possible to estimate the general state-space model using a particle filter. A particle filter can achieve ideal performance when the number of particles is infinite. However, in actual implementation the number of particles is finite; hence, performance degrades in comparison to the ideal situation. Especially in the case of complicated problems such as those with a large number of states, particle degeneracy can occur. For maintaining performance with fewer particles, various improvement methods have been proposed from algorithms to implementations, such as particle filter with MCMC [3, 7, 15, 40], Gaussian particle filter [26], and merging particle filter [32]. Among these, this book explains the auxiliary particle filter and Rao–Blackwellization. Note that this book omits their derivations; refer to [3, 5–7, 16, 25, 33, 35, 37] for details.

11.5.1 Auxiliary Particle Filter

The *auxiliary particle filter* is a technique that considers the influence of the current observations when drawing the state from the proposal distribution in particle filtering. The auxiliary particle filter was first proposed by [34]. While various extensions have since been developed [39], we introduce a basic algorithm based on the first proposal [34]. In filtering using the auxiliary particle filter, the procedure for obtaining the filtering distribution at time point t from that at time point $t - 1$ is as follows.

Algorithm 11.4 Auxiliary particle filtering

0. filtering distribution at time point $t - 1$: $\{realizations \mathbf{x}_{t-1}^{(n)}, weights \omega_{t-1}^{(n)}\}_{n=1}^N$
 1. update procedure at time point t
 - (*equivalent*) resampling
Sample from the set $\{1, \dots, N\}$ with replacement, with a probability proportional to $\omega_{t-1}^{(n)} p(y_t | y_{1:t-1}, \hat{\mathbf{x}}_t^{(n)})$ ($n = 1, \dots, N$), and prepare the index sequence $\mathbf{k} = \{k_1, \dots, k_n, \dots, k_N\}$ for resampling, where the best guess $\hat{\mathbf{x}}_t^{(n)}$ for the state is set to $E[\mathbf{x}_t | \mathbf{x}_{t-1}^{(n)}]$.
 - for $n = 1$ to N do
 - a. realizations
Draw $\mathbf{x}_t^{(n)}$ from the state equation $p(\mathbf{x}_t | \mathbf{x}_{t-1}^{(k_n)})$.
 - b. weights
$$\omega_t^{(n)} \leftarrow \frac{p(y_t | y_{1:t-1}, \mathbf{x}_t^{(k_n)})}{p(y_t | y_{1:t-1}, \hat{\mathbf{x}}_t^{(k_n)})} \quad (11.10)$$
 - end for
 - normalization of the weights: $\omega_t^{(n)} \leftarrow \omega_t^{(n)} / \sum_{n=1}^N \omega_t^{(n)}$
-
2. filtering distribution at time point t : $\{realizations \mathbf{x}_t^{(n)}, weights \omega_t^{(n)}\}_{n=1}^N$
-

When comparing Algorithm 11.4 with Algorithm 11.1, we see that the auxiliary particle filtering “first” executes the resampling to ensure that the influence of the current observations can be considered. This resampling is actually performed in two steps. First, we derive only the index sequence for resampling, which is referred to as the *auxiliary variable* sequence according to [34]. Then, we realize resampling by performing the subsequent processing based on the auxiliary variable. Through this mechanism, we prescreen the particles that would likely yield consistent results in light of the current observations. Therefore, even if we draw the realizations of

the state from the state equation, we can obtain a sample which takes the current observations into account. However, this algorithm requires the best guess regarding the current state to obtain the auxiliary variable. For this value, Algorithm 11.4 applies the conditional expectation derived from the state equation. As mentioned above, the auxiliary particle filter intentionally biases the finite particles representing the prior distribution. However, the pure method in Algorithm 11.4 can cause an excessively intensive bias, thereby reducing the diversity of the particles in advance [39]. For this reason, while it has been confirmed that auxiliary particle filtering using Algorithm 11.4 can almost obtain good performance, there is also a limitation of causing performance degrading. Thus, we must always examine the results. In some literature, the auxiliary particle filtering involves “again” the normal resampling at the last step of the algorithm [33]. Using such a version results in resampling twice. We believe that the last resampling is not necessarily required; hence, this book chooses the version without a final resampling. In addition, according to this policy, the auxiliary particle filtering in this book does not reset the weights to $1/N$ at every time point.

11.5.1.1 Example: Flow Data of the Nile

We now examine the actual behavior of an auxiliary particle filter. We apply the local-level model to the flow data of the Nile as in Sect. 12.5 of [8] and compare the filtering results using the bootstrap and auxiliary particle filters. For comparison, we examine the mean of the filtering distribution and the effective sample size.

The *effective sample size* in the particle filter is derived from the weights and defined as

$$\text{ESS}_t = 1 / \sum_{n=1}^N (\omega_t^{(n)})^2. \quad (11.11)$$

Equation (11.11) takes the maximum value N if all weights have no bias and are equally $1/N$. On the contrary, the equation takes the minimum value of one in the case of the most severe bias for the weights: a weight for one particle is 1 and those for the other particles are zeros. Thus, the effective sample size takes a value from 1 to N according to the bias for the weights.

The code for applying the local-level model to the flow data of the Nile and performing particle filtering is as follows.

Code 11.10

```

1 > # <<Apply local-level model to flow data of the Nile (particle filtering)>>
2 >
3 > # Preprocessing
4 > set.seed(4521)
5 > library(dlm)
6 >
7 > # Flow data of the Nile
8 > y <- Nile
9 > t_max <- length(y)
10 >
11 > # Function building local-level model
12 > build_dlm <- function(par) {
13 +   dlmModPoly(order = 1, dV = exp(par[1]), dW = exp(par[2]))
14 + }
15 >
16 > # Maximum likelihood estimation of parameters
17 > fit_dlm <- dlmMLE(y = y, parm = rep(0, 2), build = build_dlm)
18 > mod <- build_dlm(fit_dlm$par)
19 >
20 > # Presetting of particle filter
21 > N <- 10000          # Number of particles
22 >
23 > # *Note: Assuming that the time point of the prior distribution corresponds to one,
24 > # we regard the shifted time points (from 2 to t_max+1) as the original ones (from
25 > # 1 to t_max).
26 >
27 > # Data formatting (adding the forefront dummy corresponding to prior distribution)
28 > y <- c(NA_real_, y)
29 >
30 > # Save the value of effective sample size at every time point
31 > ESS <- rep(N, times = t_max+1)
32 >
33 > # Setting of prior distribution
34 >
35 > # Particle (realizations)
36 > x <- matrix(NA_real_, nrow = t_max+1, ncol = N)
37 > x[1, ] <- rnorm(N, mean = mod$m0, sd = sqrt(mod$c0))
38 >
39 > # Particle (weight)
40 > w <- matrix(NA_real_, nrow = t_max+1, ncol = N)
41 > w[1, ] <- log(1 / N)
42 >
43 > # Time-forward processing: auxiliary particle filter
44 > for (t in (1:t_max)+1){
45 +   # (equivalent) Resampling
46 +   # Auxiliary variable sequence
47 +   probs <- w[t-1, ] + dnorm(y[t], mean = x[t-1, ], sd = sqrt(mod$V), log = TRUE)
48 +   k <- sys_resampling(N = N, w = normalize(probs))
49 +   # State equation: generate particles (realizations)
50 +   x[t, ] <- rnorm(N, mean = x[t-1, k], sd = sqrt(mod$W))
51 +   #
52 +   # Observation equation: updating particle (weight)
53 +   w[t, ] <- dnorm(y[t], mean = x[t, ], sd = sqrt(mod$V), log = TRUE) -
54 +             dnorm(y[t], mean = x[t-1, k], sd = sqrt(mod$V), log = TRUE)
55 +   #
56 +   # Normalization of weight
57 +   w[t, ] <- normalize(w[t, ])
58 +   #
59 +   # Effective sample size

```

```

60 +   ESS[t] <- 1 / crossprod(exp(w[t, ]))
61 +
62 >
63 > # Result formatting: removing the forefront corresponding to prior distribution,
64 >   etc.
64 > y <- ts(y[-1])
65 > ESS <- ts(ESS[-1])
66 > x <- x[-1, , drop = FALSE]
67 > w <- w[-1, , drop = FALSE]
68 >
69 > # Save effective sample size, and then calculate mean
70 > APF_ESS <- ESS
71 > APF_m <- sapply(1:t_max, function(t){ weighted.mean(x[t, ], w = exp(w[t, ])) })
72 >
73 > # Ignore the display of following codes

```

In the above code, we first use the library **dlm** to prepare the model. Through the default setting of the **dlm** function, the mean and variance for the prior distribution are set to zero and 10^7 , respectively. The other parameters are specified using the maximum likelihood method. We then perform presetting for the particle filter. The number of particles is set to 10,000. Since particle smoothing (Kitagawa algorithm) is not applied in this example, we need not save the index sequence for resampling, i.e., the auxiliary variable sequence, at every time point. So we omit the preassign of the matrix for such purpose. On the contrary, because the effective sample size is saved at each time point for later comparison, we preassign the area for them. The following **for** loop describes the processing of the auxiliary particle filter. Inside the loop, we describe the processing along with Algorithm 11.4. The best guess for the state is $E[x_t | \mathbf{x}_{t-1}^{(n)}]$, and it becomes $\mathbf{x}_{t-1}^{(n)}$ in the case of the local-level model. The effective sample size is calculated based on Eq. (11.11) after normalization of the weights. Finally, we format the results and obtain the effective sample size and mean of the filtering distribution. Furthermore, for comparison, we similarly perform bootstrap filtering based on Code 11.7 to obtain the effective sample size and mean of the filtering distribution. Figure 11.13 shows the plots for the results.

Figure 11.13a shows a comparison of the mean of the filtering distribution. We obtain almost the same results. Subsequently, Fig. 11.13b shows a comparison of the effective sample size. This figure is equivalent to Fig. 12.3 in [8], and we see that the effective sample size in the auxiliary particle filter has increased than that in the bootstrap filter. While the effect of this difference on estimation performance is limited in such a simple model, the more complicated the problem is, the more the difference influences the performance.

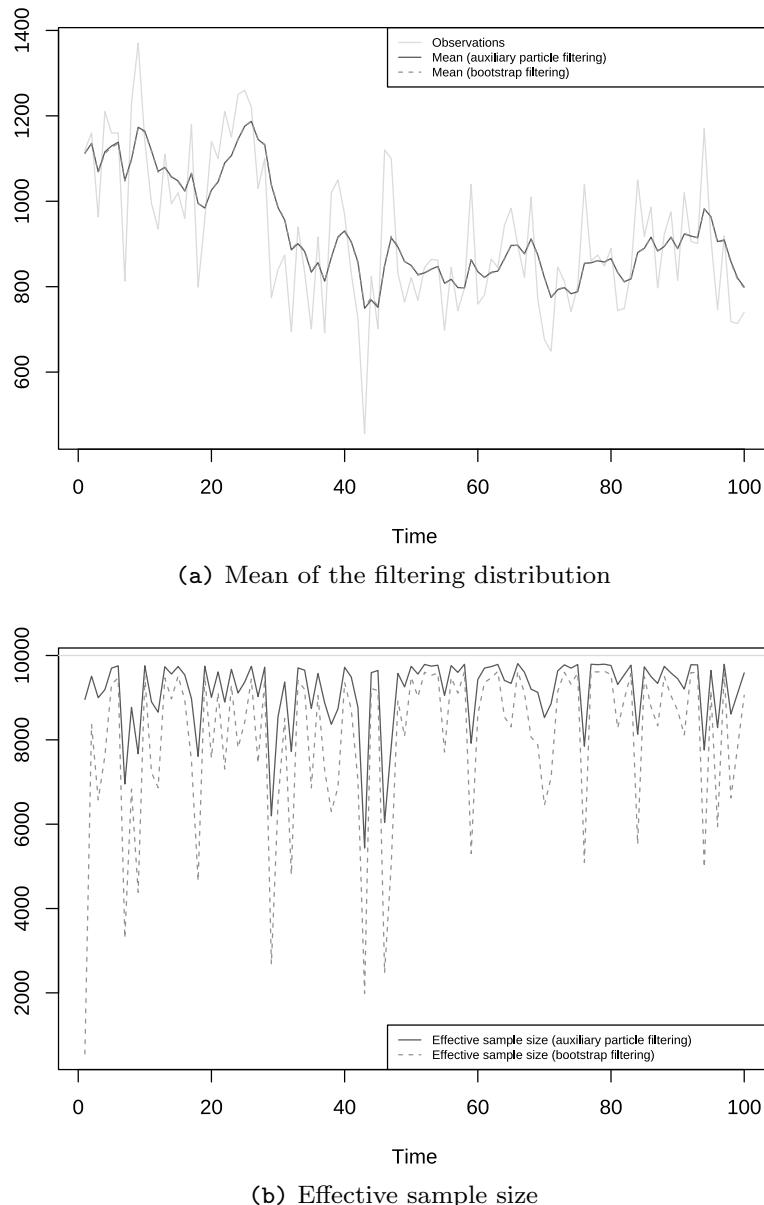


Fig. 11.13 Applying the local-level model to the flow data of the Nile

11.5.1.2 Usage in the Liu and West Filter

We now explain the *Liu and West filter* [28] as an example application of the auxiliary particle filter. The Liu and West filter is a kind of algorithm that jointly estimates the state and parameters using a particle filter. While we confirm the details of the Liu and West filter later, we herein consider the parameter estimation using a particle filter.

Various methods have been proposed for parameter estimation using a particle filter [3, 15, 35, 37, 40]. First, if there is a specific number of observations, we can in principle specify the parameters using the maximum likelihood method. In a particle filter in which resampling is performed at every time point, we can obtain the likelihood (except for the constant coefficients) through the product of expectation regarding weight across all time points. While the numerical maximization of likelihood in a particle filter is not straightforward owing to the Monte Carlo error, some proposals are based on this approach. This book considers sequential parameter estimation, which is applicable even in a real-time problem. Furthermore, we suppose joint estimation of states and parameters; a state partly includes the parameter as a random variable.

In a particle filter, an estimation approach that regards the parameters as random variables and includes them as a part of the state is known as *self-organizing* [21].

When a time-invariant parameter is estimated, its state noise is in principle zero. In this case, the particle filter continues reusing the realizations from the prior distribution even if time progresses; hence, degeneracy to a specific particle arises immediately. While this approach is valid if sufficient knowledge can be reflected in the prior distribution, such knowledge is not always available. According to the principle of a particle filter, if the realizations of parameters continue to be refreshed through the use of some mechanism, we could track the correct parameters as time progresses. For this purpose, [24] has proposed using “moderate” state noise by assuming the temporal transition of parameters as in the local-level model. This approach is also valid if an adequate variance of the state noise is almost predictable in advance. However, it can be difficult to guess the moderate variance because the

value depends on the problem and also is generally unknown. If we use a large value, the variance in the parameter estimates increases, prompting a concern that the estimation accuracy deteriorates. The Liu and West filter was proposed at an early time in the history of particle filter development to resolve such a concern. This filter uses a technique called *kernel smoothing* [38] as a means of refreshing the realizations without increasing the variance for the parameter estimates. Kernel smoothing first applies the artificial moving average to the realization cloud at the previous time point. Using this operation, we can once reduce the variance while maintaining the mean. This variance reduction is then used as “funds:” we draw new particle realizations from the continuous distribution whose variance is set to the same value as the variance reduction. Thus, we can refresh the realizations without increasing the total variance. In the Liu and West filter, this technique is incorporated into the particle filter; hence, resampling is repeated in the situation in which the total variance does not increase. As a result, the diversity of the particles gradually decreases, and the estimates eventually converge. Based on this property, we can understand that the estimates must approach the true value sufficiently before convergence. Therefore, we cannot select the prior distribution entirely freely on the parameters, but must have some relevant knowledge. Incidentally, the first proposed version of the Liu and West filter [28] used the auxiliary particle filter as the particle filter; hence, we explain the filter in accordance with convention. It is not always necessary to do this [31, 36]. Thus far, regarding parameter estimation using a particle filter, we mentioned the positioning and overview of the Liu and West filter; hence, we proceed with the details.

First, the kernel smoothing algorithm at time point t is as follows.

Algorithm 11.5 Kernel smoothing algorithm

1. update procedure at time point t
 - *artificial moving average for parameters*
 mean: $\mu^{(n)} \leftarrow a\theta_{t-1}^{(n)} + (1-a)\text{E}_{\omega_{t-1}^{(n)}}[\theta_{t-1}^{(n)}]$
 variance reduction: $\gamma \leftarrow (1-a^2)\text{Var}_{\omega_{t-1}^{(n)}}[\theta_{t-1}^{(n)}]$
 - **for** $n = 1$ **to** N **do**
 - a. *realizations (parameters)*
 Draw $\theta_t^{(n)}$ from a continuous proposal distribution with mean $\mu^{(n)}$ and variance γ .**end for**
-

While the above algorithm assumes the estimation of the time-invariant parameter θ , the parameters are refreshed at each time point. Thus, they are denoted by θ_t , whose subscript t can distinguish them from each other. The exponential weight for the artificial moving average is set to a . While the weight a takes a value from zero to one and should be adjusted in actual problems, [28] generally recommends a value from 0.974 to 0.995. In addition, $E_\star[\cdot]$ and $\text{Var}_\star[\cdot]$ is the weighted mean and weighted variance, respectively, based on the weight \star . While [28] uses the normal distribution as the continuous proposal distribution, we can actually use an arbitrary distribution, as in [33].

Then, the code for kernel smoothing is as follows.

Code 11.11

```

1 > # <<Kernel smoothing>>
2 >
3 > # User-defined function to perform artificial moving average for parameters
4 > kernel_smoothing <- function(realization, w, a){
5 +   # Restore w to the linear domain value
6 +   w <- exp(w)
7 +
8 +   # Weighted mean and variance
9 +   mean_realization <- weighted.mean( realization , w)
10 +  var_realization <- weighted.mean((realization - mean_realization)^2, w)
11 +
12 +  # Mean and variance decrease through artificial moving average
13 +  mu <- a * realization + (1 - a) * mean_realization
14 +  sigma2 <- (1 - a^2) * var_realization
15 +
16 +  return(list(mu = mu, sigma = sqrt(sigma2)))
17 + }
```

In the above code, a user-defined function for executing an artificial moving average for the parameters is described as `kernel_smoothing()`. The arguments of the function are `realization`, `w`, and `a`, which are the vector of realizations, the vector of normalized logarithmic weights, and the value of exponential weight for the moving average, respectively. The contents of the process are described in accordance with the “artificial moving average for parameters” in Algorithm 11.5. The variance reduction is finally converted to standard deviation form and included in the return value.

Next, the algorithm of the Liu and West filter is as follows.

Algorithm 11.6 Liu and West filter

-
0. filtering distribution at time point $t - 1$: $\left\{ \text{realizations } \boldsymbol{\theta}_{t-1}^{(n)}, \mathbf{x}_{t-1}^{(n)}, \text{weights } \omega_{t-1}^{(n)} \right\}_{n=1}^N$
 1. update procedure at time point t

- *artificial moving average for parameters*
 mean: $\boldsymbol{\mu}^{(n)} \leftarrow a \boldsymbol{\theta}_{t-1}^{(n)} + (1 - a) \mathbb{E}_{\omega_{t-1}^{(n)}} [\boldsymbol{\theta}_{t-1}^{(n)}]$
 variance reduction: $\gamma \leftarrow (1 - a^2) \text{Var}_{\omega_{t-1}^{(n)}} [\boldsymbol{\theta}_{t-1}^{(n)}]$
 - *(equivalent) resampling*
 Sample from the set $\{1, \dots, N\}$ with replacement, with a probability proportional to $\omega_{t-1}^{(n)} p(y_t | y_{1:t-1}, \boldsymbol{\theta}_t^{(n)}, \hat{\mathbf{x}}_t^{(n)})$ ($n = 1, \dots, N$), and prepare index sequence $\mathbf{k} = \{k_1, \dots, k_n, \dots, k_N\}$ for resampling, where let the best guess for the parameters and state be $\hat{\boldsymbol{\theta}}_t^{(n)} = \boldsymbol{\mu}^{(n)}$ and $\hat{\mathbf{x}}_t^{(n)} = \mathbb{E}[\mathbf{x}_t | \mathbf{x}_{t-1}^{(n)}, \boldsymbol{\mu}^{(n)}]$, respectively.
 - **for** $n = 1$ **to** N **do**
 - a-1. *realizations (parameters)*
 Draw $\boldsymbol{\theta}_t^{(n)}$ from a continuous proposal distribution with mean $\boldsymbol{\mu}^{(k_n)}$ and variance γ .
 - a-2. *realizations (state)*
 Draw $\mathbf{x}_t^{(n)}$ from the state equation $p(\mathbf{x}_t | \mathbf{x}_{t-1}^{(k_n)}, \boldsymbol{\theta}_t^{(n)})$.
 - b. *weights*

$$\omega_t^{(n)} \leftarrow \frac{p(y_t | y_{1:t-1}, \boldsymbol{\theta}_t^{(n)}, \mathbf{x}_t^{(n)})}{p(y_t | y_{1:t-1}, \hat{\boldsymbol{\theta}}_t^{(k_n)}, \hat{\mathbf{x}}_t^{(k_n)})} \quad (11.12)$$
 - end for**
 - *normalization of the weights*: $\omega_t^{(n)} \leftarrow \omega_t^{(n)} / \sum_{n=1}^N \omega_t^{(n)}$
-
2. filtering distribution at time point t : $\left\{ \text{realizations } \boldsymbol{\theta}_t^{(n)}, \mathbf{x}_t^{(n)}, \text{weights } \omega_t^{(n)} \right\}_{n=1}^N$

The Liu and West filter derives the filtering distribution $p(\mathbf{x}_t, \boldsymbol{\theta} | y_{1:t})$, which includes the parameters as a part of the state. Thus, the realization of the particle now becomes a set consisting of the parameter and the state. The content of the process is a combination of the auxiliary particle filter in Algorithm 11.4 and the kernel smoothing in Algorithm 11.5 basically. The best guesses for the parameters and state required to derive the auxiliary variable sequence are set to $\hat{\boldsymbol{\theta}}_t^{(n)} = \boldsymbol{\mu}^{(n)}$ and $\hat{\mathbf{x}}_t^{(n)} = \mathbb{E}[\mathbf{x}_t | \mathbf{x}_{t-1}^{(n)}, \boldsymbol{\mu}^{(n)}]$, respectively.

We now confirm the actual behavior of the Liu and West filter. For the model and data, we use the artificial local-level model and the generated data, which are prepared with Code 9.1 in Sect. 9.2. This example jointly estimates the state and parameters (the variances W and V of the state and observation noises, respectively). This code is as follows.

Code 11.12

```

1 > # <<Local-level model with known parameters (Liu and West filter)>>
2 >
3 > # Preprocessing
4 > set.seed(4521)
5 >
6 > # Load data on artificial local-level model
7 > load(file = "ArtifititalLocalLevelModel.RData")
8 >
9 > # Presetting of particle filter
10 > N <- 10000           # Number of particles
11 > a <- 0.975            # Exponential weight in artificial moving average for
   ↪  parameters
12 > W_max <- 10 * var(diff(y))    # Guess maximum value for parameter W
13 > V_max <- 10 * var(      y )    # Guess maximum value for parameter V
14 >
15 > # *Note: Assuming that the time point of the prior distribution corresponds to one,
   ↪  we regard the shifted time points (from 2 to t_max+1) as the original ones (from
   ↪  1 to t_max).
16 >
17 > # Data formatting (adding the forefront dummy corresponding to prior distribution)
18 > y <- c(NA_real_, y)
19 >
20 > # Setting of prior distribution
21 >
22 > # Particle (realizations): parameter W
23 > W      <- matrix(NA_real_, nrow = t_max+1, ncol = N)
24 > W[1, ] <- log(runif(N, min = 0, max = W_max))      # Log domain
25 >
26 > # Particle (realizations): parameter V
27 > V      <- matrix(NA_real_, nrow = t_max+1, ncol = N)
28 > V[1, ] <- log(runif(N, min = 0, max = V_max))      # Log domain
29 >
30 > # Particle (realizations): state
31 > x <- matrix(NA_real_, nrow = t_max+1, ncol = N)
32 > x[1, ] <- rnorm(N, mean = 0, sd = sqrt(1e+7))      # Prior distribution with
   ↪  unknown parameters
33 >
34 > # Particle (weight)
35 > w <- matrix(NA_real_, nrow = t_max+1, ncol = N)
36 > w[1, ] <- log(1 / N)
37 >
38 > # Time-forward processing: kernel smoothing + auxiliary particle filter
39 > for (t in 1:t_max){}
40 >   # Artificial moving average for parameters
41 >   W_ks <- kernel_smoothing(realization = W[t-1, ], w = w[t-1, ], a = a)
42 >   V_ks <- kernel_smoothing(realization = V[t-1, ], w = w[t-1, ], a = a)
43 >
44 >   # (equivalent) Resampling
45 >
46 >   # Auxiliary variable sequence
47 >   probs <- w[t-1, ] +
48 >     dnorm(y[t], mean = x[t-1, ], sd = sqrt(exp(V_ks$mu)), log = TRUE)
49 >   k <- sys_resampling(N = N, w = normalize(probs))
50 >
51 >   # Draw realizations of parameters from a continuous proposal distribution
   ↪  (refreshment)
52 >   W[t, ] <- rnorm(N, mean = W_ks$mu[k], sd = W_ks$sigma)

```

```

53 + V[t, ] <- rnorm(N, mean = V_ks$mu[k], sd = V_ks$sigma)
54 +
55 + # State equation: generate particles (realizations)
56 + x[t, ] <- rnorm(N, mean = x[t-1, k], sd = sqrt(exp(W[t, ])))
57 +
58 + # Observation equation: updating particle (weight)
59 + w[t, ] <- dnorm(y[t], mean = x[t, ], sd = sqrt(exp(V[t, ]))), log = T) -
60 +           dnorm(y[t], mean = x[t-1, k], sd = sqrt(exp(V_ks$mu[k])), log = T)
61 +
62 + # Normalization of weight
63 + w[t, ] <- normalize(w[t, ])
64 +
65 >
66 > # Ignore the display of following codes

```

In the above code, we first set the number of particles to 10,000. The exponential weight a for the artificial moving average in kernel smoothing is set to 0.975 according to the example in [33] (however, there is no particular intention). In addition, we must reflect our knowledge in the prior distribution of the parameters to the extent possible; hence, we apply a uniform distribution whose maximum value is guessed from the data. We handle the parameters in the log-domain to suppress the degrading of calculation accuracy. The mean and variance in the prior distribution of the state are also unknown, and they are set to zero and 10^7 , respectively. We describe the specific processing for the Liu and West filter inside the `for` loop after the above preparation. This description is in accordance with that in Algorithm 11.6. The user-defined function `kernel_smoothing()` described in Code 11.11 is used for the artificial moving average for the parameters. The best guess for the state is $\hat{x}_t^{(n)} = E[x_t | x_{t-1}^{(n)}, \mu^{(n)}] = x_{t-1}^{(n)}$ because this example applies the local-level model. In addition, we use the normal distribution as a continuous proposal distribution for refreshing the parameter realizations. Finally, we format the obtained results and calculate various summary statistics. Figures 11.14 and 11.15 show the plotting results. Figure 11.14 shows the plots for the mean and 95% intervals of the state.

As can be seen from the 95% intervals in Fig. 11.14b, the estimation accuracy before time point 50 is not sufficient, because the convergence of the jointly estimated parameters is not entirely achieved through approximately that time point and affects the estimation accuracy of the state.

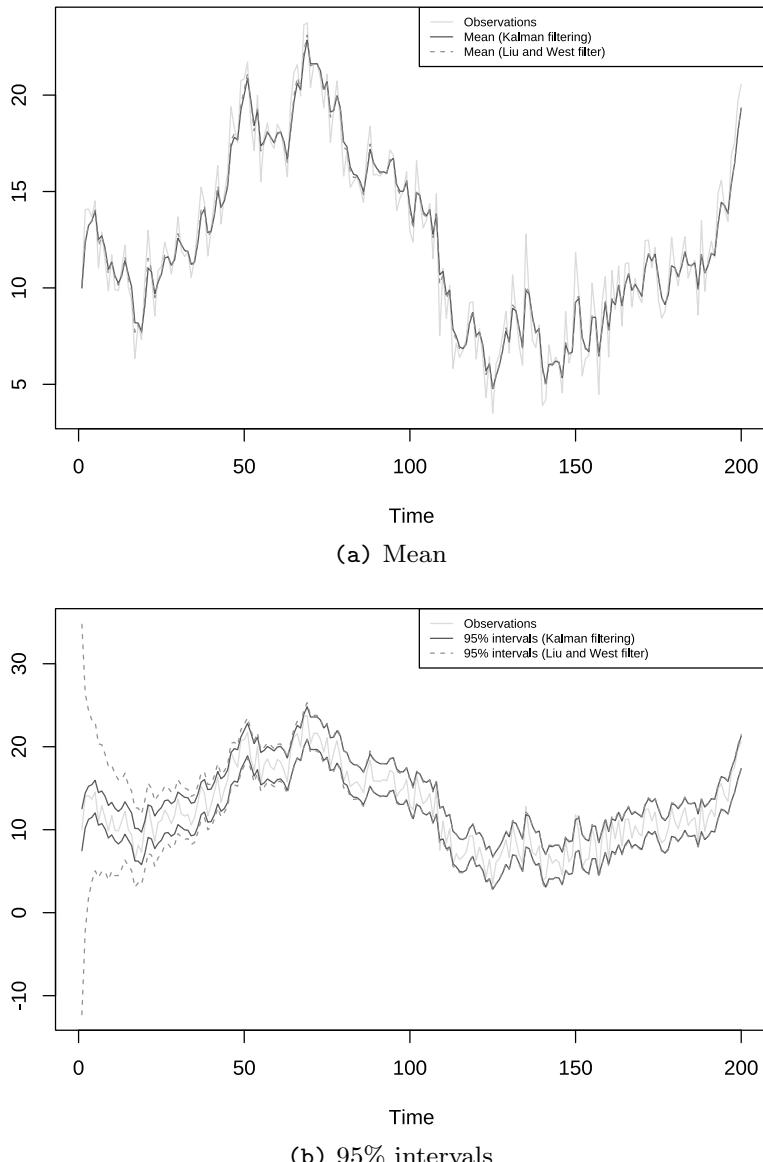


Fig. 11.14 Local-level model with known parameters (Liu and West filter)

In addition, Fig. 11.15 shows the plots for the mean and 95% intervals of the parameters W and V .

While both results require some time to converge, we can eventually estimate values close to the true ones.

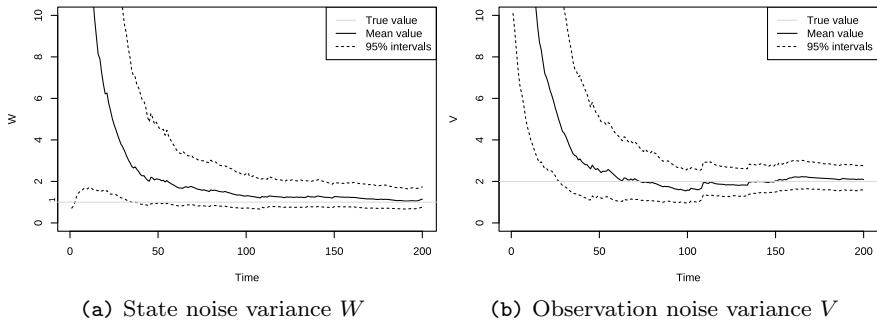


Fig. 11.15 Estimation results of parameters (Liu and West filter)

Through the above simple example, we have confirmed that the Liu and West filter can jointly estimate the state and parameters sequentially.

11.5.2 Case in Which the Linear Gaussian State-Space Model is Partially Applicable

This topic for the case of MCMC was mentioned in Sect. 10.5.1, and the same concept also applies in the case of the particle filter. In the analysis of the general state-space model, the linear Gaussian state-space model is sometimes partially applicable (but not always). We can use a Kalman filter as a part of the solution in such a case. This approach focuses the limited particle filter capability on the part that cannot be solved using a Kalman filter. Such an approach to the particle filter is referred to as *Rao–Blackwellization*. In addition, a Kalman filter is an analytical solution and can contribute to the improvement of estimation accuracy. Thus, this book recommends this approach if possible.

A Rao–Blackwellized particle filter is sometimes referred to as a *marginalized particle filter*. It is also known as a *mixture Kalman filter* because it amounts to a hybridization of Kalman and particle filters.

Specifically, we can use Rao–Blackwellization when the state in the general state-space model is divided into parts that conform to the linear Gaussian state-space model and others. This corresponds to the case in which we regard the unknown parameters in the linear Gaussian state-space model as the random variables. In such a situation, if parameters were already known or fixed through means such

as maximum likelihood estimation, we could treat the model as a linear Gaussian state-space model. However, the parameters are now estimated jointly with the states, that is, the joint filtering distribution $p(\mathbf{x}_t, \boldsymbol{\theta} | y_{1:t})$ of the states and parameters is the estimation target; hence, we must use the general state-space model. Once the parameters are specified, we can reconsider the model as a linear Gaussian state-space model. Based on this insight, decomposing the joint filtering distribution as $p(\mathbf{x}_t, \boldsymbol{\theta} | y_{1:t}) = p(\mathbf{x}_t | y_{1:t}, \boldsymbol{\theta})p(\boldsymbol{\theta} | y_{1:t})$ by Bayes' theorem, we see that the Kalman and particle filters can be applied to the first and second terms, respectively. This is the basic concept of Rao–Blackwellization.

Specifically, Rao–Blackwellization modifies the particle filter as follows:

- The realizations of the state \mathbf{x}_t now become their parameters, i.e., the mean \mathbf{m}_t and variance \mathbf{C}_t of the filtering distribution, and the Kalman filter for one time point is used to derive them.
- The likelihood given the state \mathbf{x}_t , i.e., the observation equation, now becomes the one-step-ahead predictive likelihood given parameters $\boldsymbol{\theta}$, and a Kalman filter for one time point is also used to derive them.

We consider later the application of Rao–Blackwellization to the Liu and West filter; here, we prepare for the explanation in advance. Regarding the Kalman filter for one time point in Rao–Blackwellization, we represent this process at time point t as follows:

- Kalman filtering for one time point: $\mathcal{KF}(\text{observations}_t, \text{state}_{t-1}, \text{parameters}_t)$

We implicitly assume the model to be applied and omit the explicit notation in the above. The processing of Kalman filtering for one time point follows Algorithm 8.1. This code is as follows.

Code 11.13

```

1 > # <<Kalman filtering at one time point>>
2 >
3 > # User-defined function performing Kalman filtering for one time point
4 > Kalman_filtering <- function(y, state, param){
5 +   # Obtain the result for all particles initially (number of particles N is set to
6 +   # that in the parent environment)
6 +   res <- sapply(1:N, function(n){
7 +     # Model setting: mod in the parent environment is automatically copied as base
8 +     mod$m0 <-      state$m0[n]
9 +     mod$C0 <-      state$C0[n]
10 +    mod$W  <- exp(param$ W[n])    # W is the log domain value
11 +    mod$V  <- exp(param$ V[n])    # V is the log domain value
12 +
13 +    # Execute Kalman filtering for one time point
14 +    KF_out <- dlmFilter(y = y, mod = mod)
15 +
16 +    # Concatenate the required values
17 +    return(
18 +      c(
19 +        # Derivation of state (the mean and variance of filtering distribution)
20 +        m = KF_out$m[2],                      # "1" in the state
21 +        # corresponds to the prior distribution
22 +        C = dlmSvd2var(KF_out$U.C, KF_out$D.C)[[2]], # "1" in the state
23 +        # corresponds to the prior distribution
24 +
25 +        # For the calculation of the one-step-ahead predictive likelihood
26 +        f = KF_out$f,
27 +        Q = mod$FF %*% dlmSvd2var(KF_out$U.R, KF_out$D.R)[[1]] %*% t(mod$FF) +
28 +          mod$V
29 +      )
30 +
31 +    # Integrate everything into a list for easy handling
32 +    return(list(m = res["m", ], C = res["C", ], f = res["f", ], Q = res["Q", ]))
33 +  )

```

The code above describes a user-defined function `Kalman_filtering()` performing Kalman filtering for one time point under the assumption that the number of states is one and the parameters are only variances for the state and observation noises. Although we have defined the function with the same name in Code 8.1, the above code now modifies the contents utilizing the library `dlm`. The arguments of the function are `y`, `state`, and `param`, which are the observations at the current time point, the realizations for the state at the previous time point, and the values of parameters at the current time point, respectively. Regarding the argument type, we suppose that the observations are scalar, and that the state and parameters are the list of vectors; each element of the vector corresponds to the one particle. Inside this function, the Kalman filtering process is performed for all particles. For the calculation of Kalman filtering, we use the library `dlm` function `dlmFilter()`. From the return value of this function, we extract the mean m_t and the variance C_t of the filtering distribution (for obtaining the realization of the state) and also extract the mean f_t and the variance Q_t of the one-step-ahead predictive likelihood (for calculating the likelihood value). We finally combine them in a list for the return value in consideration of easy handling.

11.5.2.1 Application to the Liu and West Filter

We now consider Rao–Blackwellization of the Liu and West filter as a specific example. This algorithm is as follows.

Algorithm 11.7 Rao–Blackwellized Liu and West Filter

0. filtering distribution at time point $t - 1$: $\left\{ \text{realizations } \boldsymbol{\theta}_{t-1}^{(n)}, \mathbf{m}_{t-1}^{(n)}, \mathbf{C}_{t-1}^{(n)}, \text{weights } \omega_{t-1}^{(n)} \right\}_{n=1}^N$
 1. update procedure at time point t
 - *artificial moving average for parameters*
mean: $\boldsymbol{\mu}^{(n)} \leftarrow a \boldsymbol{\theta}_{t-1}^{(n)} + (1 - a) \mathbb{E}_{\omega_{t-1}^{(n)}} [\boldsymbol{\theta}_{t-1}^{(n)}]$
variance reduction: $\gamma \leftarrow (1 - a^2) \text{Var}_{\omega_{t-1}^{(n)}} [\boldsymbol{\theta}_{t-1}^{(n)}]$
 - *(equivalent) resampling*
Sample from the set $\{1, \dots, N\}$ with replacement, with a probability proportional to $\omega_{t-1}^{(n)} p(y_t | y_{1:t-1}, \hat{\boldsymbol{\theta}}_t^{(n)})$ ($n = 1, \dots, N$), and prepare index sequence $\mathbf{k} = \{k_1, \dots, k_n, \dots, k_N\}$ for resampling, where let the best guess for the parameters be $\hat{\boldsymbol{\theta}}_t^{(n)} = \boldsymbol{\mu}^{(n)}$, and calculate the one-step-ahead predictive likelihood $p(y_t | y_{1:t-1}, \hat{\boldsymbol{\theta}}_t^{(n)})$ as $\mathcal{N}(y_t; f_t^{(n)}, Q_t^{(n)})$ using $f_t^{(n)}$ and $Q_t^{(n)}$ obtained from $\mathcal{KF}(\text{observations}_t = y_t, \text{state}_{t-1} = \{\mathbf{m}_{t-1}^{(n)}, \mathbf{C}_{t-1}^{(n)}\}, \text{parameters}_t = \boldsymbol{\mu}^{(n)})$.
 - **for** $n = 1$ **to** N **do**
 - a-1. *realizations (parameters)*
Draw $\boldsymbol{\theta}_t^{(n)}$ from a continuous proposal distribution with mean $\boldsymbol{\mu}^{(k_n)}$ and variance γ .
 - a-2. *realizations (state)*
Obtain $\mathbf{m}_t^{(n)}$ and $\mathbf{C}_t^{(n)}$ from $\mathcal{KF}(\text{observations}_t = y_t, \text{state}_{t-1} = \{\mathbf{m}_{t-1}^{(k_n)}, \mathbf{C}_{t-1}^{(k_n)}\}, \text{parameters}_t = \boldsymbol{\theta}_t^{(n)})$.
 - b. *weights*
$$\omega_t^{(n)} \leftarrow \frac{p(y_t | y_{1:t-1}, \boldsymbol{\theta}_t^{(n)})}{p(y_t | y_{1:t-1}, \hat{\boldsymbol{\theta}}_t^{(k_n)})}, \quad (11.13)$$
where the numerator is calculated as $\mathcal{N}(y_t; f_t^{(n)}, Q_t^{(n)})$ using the results in step “a-2” and the denominator is calculated as $\mathcal{N}(y_t; f_t^{(k_n)}, Q_t^{(k_n)})$ using the results in step “(equivalent) resampling.”
2. filtering distribution at time point t : $\left\{ \text{realizations } \boldsymbol{\theta}_t^{(n)}, \mathbf{m}_t^{(n)}, \mathbf{C}_t^{(n)}, \text{weights } \omega_t^{(n)} \right\}_{n=1}^N$
-

We focus on the differences from Algorithm 11.6. First, the realizations of the state \mathbf{x}_t now become their parameters, that is, the mean \mathbf{m}_t and variance \mathbf{C}_t of the filtering distribution. They are derived using the Kalman filter. Next, the observation equation (the likelihood given the state) in the steps “(equivalent) resampling” and “weights” is changed to the one-step-ahead predictive likelihood given the parameters. They are also derived using the Kalman filter.

We then examine the actual behavior of the Rao–Blackwellized Liu and West filter. For the model and data, we use the artificial local-level model and the data prepared using Code 9.1 in Sect. 9.2. This code is as follows.

Code 11.14

```

1 > # <<Rao-Blackwellized Liu and West filter>>
2 >
3 > # Preprocessing
4 > set.seed(4521)
5 >
6 > # Load data on artificial local-level model
7 > load(file = "ArtificialLocalLevelModel.RData")
8 > m_org <- m      # Save the existing variable m distinguished from new one for the
   ← mean in particles representing the filtering distribution
9 >
10 > # Presetting of particle filter
11 > N <- 1000          # Number of particles
12 > a <- 0.975         # Exponential weight in artificial moving average for
   ← parameters
13 > W_max <- 10 * var(diff(y))    # Guess maximum value for parameter W
14 > V_max <- 10 * var(      y )    # Guess maximum value for parameter V
15 >
16 > # *Note: Assuming that the time point of the prior distribution corresponds to one,
   ← we regard the shifted time points (from 2 to t_max+1) as the original ones (from
   ← 1 to t_max).
17 >
18 > # Data formatting (adding the forefront dummy corresponding to prior distribution)
19 > y <- c(NA_real_, y)
20 >
21 > # Setting of prior distribution
22 >
23 > # Particle (realizations): parameter W (log domain)
24 > W      <- matrix(NA_real_, nrow = t_max+1, ncol = N)
25 > W[1, ] <- log(runif(N, min = 0, max = W_max))      # Log domain
26 >
27 > # Particle (realizations): parameter V (log domain)
28 > V      <- matrix(NA_real_, nrow = t_max+1, ncol = N)
29 > V[1, ] <- log(runif(N, min = 0, max = V_max))      # Log domain
30 >
31 > # Particle (realizations): state (the mean and variance of the filtering
   ← distribution)
32 > m <- matrix(NA_real_, nrow = t_max+1, ncol = N)
33 > m[1, ] <- 0                      # Prior distribution with
   ← unknown parameters
34 > C <- matrix(NA_real_, nrow = t_max+1, ncol = N)
35 > C[1, ] <- 1e-7                  # Prior distribution with
   ← unknown parameters
36 >
37 > # Particle (weight)
38 > w <- matrix(NA_real_, nrow = t_max+1, ncol = N)
39 > w[1, ] <- log(1 / N)
40 >
41 > # Setting of the progress bar
42 > progress_bar <- txtProgressBar(min = 2, max = t_max+1, style = 3)
43 >
44 > # Time-forward processing: kernel smoothing + auxiliary particle filter +
   ← Rao-Blackwellization

```

```

45 > for (t in (1:t_max)+1){
46 +   # Display progress bar
47 +   setTxtProgressBar(pb = progress_bar, value = t)
48 +
49 +   # Artificial moving average for parameters
50 +   W_ks <- kernel_smoothing(realization = W[t-1, ], w = w[t-1, ], a = a)
51 +   V_ks <- kernel_smoothing(realization = V[t-1, ], w = w[t-1, ], a = a)
52 +
53 +   # (equivalent) Resampling
54 +
55 +   # Kalman filtering for one time point -> auxiliary variable sequence
56 +   KF_aux <- Kalman_filtering(y = y[t],
57 +                                 state = list(m0 = m[t-1, ], C0 = C[t-1, ]),
58 +                                 param = list(W = W_ks$mu, V = V_ks$mu)
59 +                               )
60 +   probs <- w[t-1, ] +
61 +             dnorm(y[t], mean = KF_aux$f, sd = sqrt(KF_aux$Q), log = TRUE)
62 +   k <- sys_resampling(N = N, w = normalize(probs))
63 +
64 +
65 +   # Draw realizations of parameters from a continuous proposal distribution
66 +   ← (refreshment)
66 +   W[t, ] <- rnorm(N, mean = W_ks$mu[k], sd = W_ks$sigma)
67 +   V[t, ] <- rnorm(N, mean = V_ks$mu[k], sd = V_ks$sigma)
68 +
69 +   # State: Kalman filtering for one time point -> derivation of particles
69 +   ← (realizations)
70 +   KF <- Kalman_filtering(y = y[t],
71 +                           state = list(m0 = m[t-1, k], C0 = C[t-1, k]),
72 +                           param = list(W = W[t, ], V = V[t, ])
73 +                         )
74 +   m[t, ] <- KF$m
75 +   C[t, ] <- KF$C
76 +
77 +   # Update particle (weight)
78 +   w[t, ] <- dnorm(y[t], mean = KF$f, sd = sqrt(KF$Q), log = T) -
79 +             dnorm(y[t], mean = KF_aux$f[k], sd = sqrt(KF_aux$Q[k]), log = T)
80 +
81 +   # Normalization of weight
82 +   w[t, ] <- normalize(w[t, ])
83 }
84 ====== | 100%
85 >
86 > # Ignore the display of following codes

```

We focus on the differences from Code 11.12. We first reduce the number of particles to 1,000 because Rao–Blackwellization is expected to result in performance improvement. In addition, we use the mean m and variance C of the filtering distribution as the realizations of the state. Inside the `for` loop after preparation, we describe the specific processing of the Rao–Blackwellized Liu and West filter. Since this processing is somewhat heavy, we display the progress bar. The content of the process is in accordance with Algorithm 11.7. The user-defined function `Kalman_filtering()` in Code 11.13 is used to execute Kalman filtering for one time point. Finally, we format the obtained results and calculate various summary statistics (however, the code display is omitted). Figures 11.16 and 11.17 show the their plotting results. Figure 11.16 shows the plots for the mean and 95% intervals of the state.

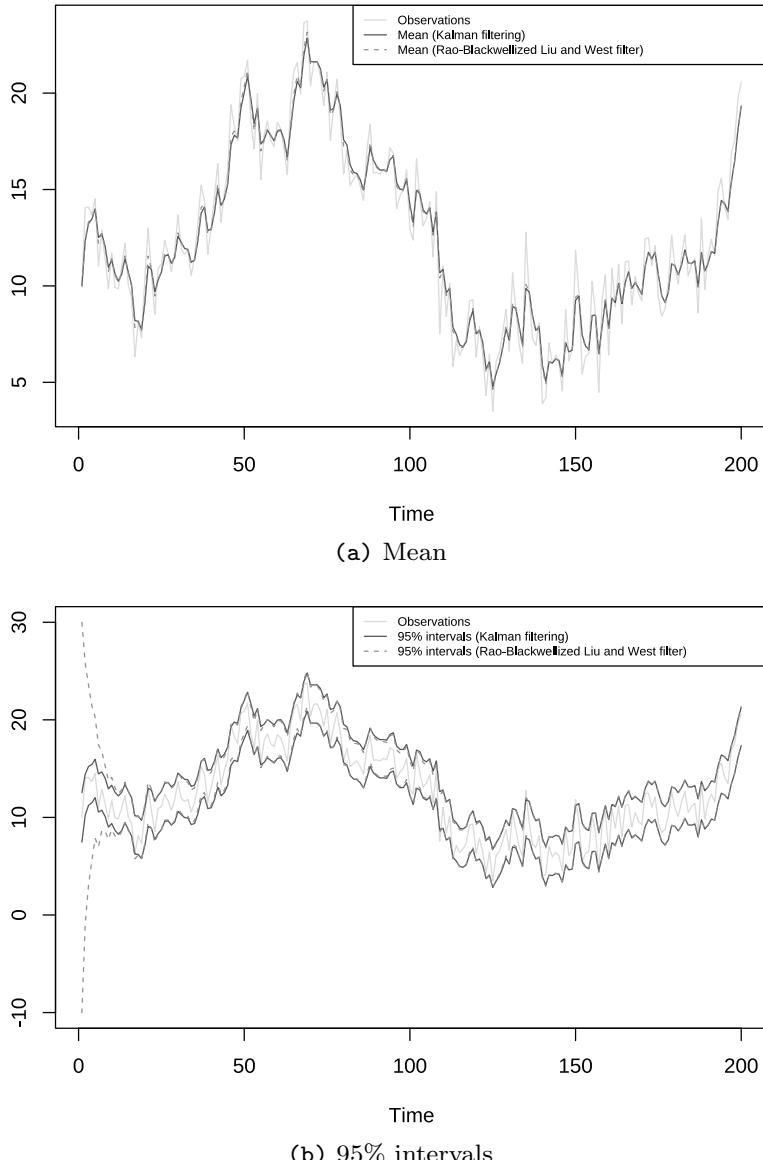


Fig. 11.16 Local-level model with known parameters (Rao-Blackwellized Liu and West filter)

Comparing Fig. 11.16b with Fig. 11.14b, we see that the estimation accuracy at the initial time points is improved for the 95% interval results. This is because the accuracy of the jointly estimated parameters is improved at the initial time point. In

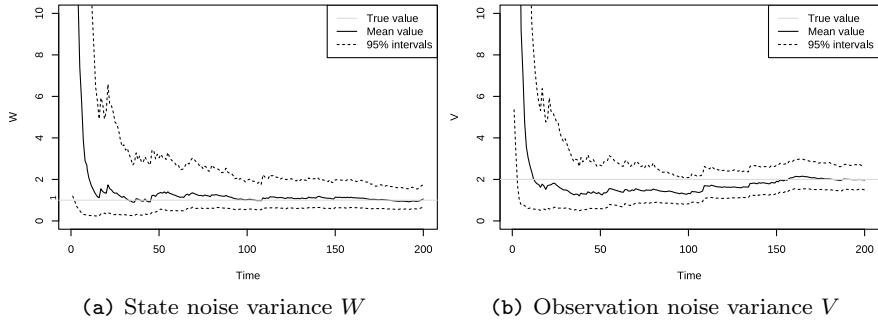


Fig. 11.17 Estimation result of parameters (Rao–Blackwellized Liu and West filter)

addition, Fig. 11.17 shows the plots for the mean and 95% intervals for the parameters W and V .

Comparing Fig. 11.17 with Fig. 11.15, we can see that the convergence performance is improved.

Through the above simple example, we have confirmed that estimation performance can be improved by applying Rao–Blackwellization to the Liu and West filter even if the number of particles is reduced to 1/10. Incidentally, the effect of Rao–Blackwellization would be more apparent for more complex problems.

References

1. Akashi, H., Kumamoto, H., Nose, K.: Application of Monte Carlo method to optimal control for linear systems under measurement noise with Markov dependent statistical property. *Int. J. Control* **22**(6), 821–836 (1975)
 2. Andrade Netto, M.L., Gimeno, L., Mendes, M.J.: On the optimal and suboptimal nonlinear filtering problem for discrete-time systems. *IEEE Trans. Automat. Control* **23**(6), 1062–1067 (1978)
 3. Cappé, O., Godsill, S.J., Moulines, E.: An overview of existing methods and recent advances in sequential Monte Carlo. *Proc. IEEE* **95**(5), 899–924 (2007)
 4. Douc, R., Cappé, O., Moulines, E.: Comparison of resampling schemes for particle filtering. In: *Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis (ISPA)*, pp. 64–69 (2005)
 5. Douc, R., Moulines, E., Stoerger, D.: *Nonlinear Time Series: Theory, Methods and Applications with R Examples*. Texts in Statistical Science. CRC Press (2014)
 6. Doucet, A., Freitas, N.d., Gordon, N. (eds.): *Sequential Monte Carlo Methods in Practice*. Information Science and Statistics. Springer, Berlin (2001)
 7. Doucet, A., Johansen, A.M.: A tutorial on particle filtering and smoothing: fifteen years later. In: Crisan, D., Rozovskii, B. (eds.) *The Oxford Handbook of Nonlinear Filtering*, pp. 656–704. Oxford University Press (2011)
 8. Durbin, J., Koopman, S.J.: *Time Series Analysis by State Space Methods*, 2nd edn. Oxford University Press (2012)

9. Evensen, G.: Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics. *J. Geophys. Res.: Oceans* **99**(C5), 10143–10162 (1994)
10. Godsill, S.J., Doucet, A., West, M.: Monte Carlo smoothing for nonlinear time series. *J. Am. Stat. Assoc.* **99**(465), 156–168 (2004)
11. Gordon, N.J., Salmond, D.J., Smith, A.F.M.: Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proc. F (Radar Signal Processing)* **140**(2), 107–113 (1993)
12. Handschin, J.E., Mayne, D.Q.: Monte Carlo techniques to estimate the conditional expectation in multi-stage non-linear filtering. *Int. J. Control* **9**(5), 547–559 (1969)
13. Higuchi, T., Ueno, G., Nakano, S., Nakamura, K., Yoshida, R.: Introduction to Data Assimilation—Next Generation Simulation Technology—. Asakura Publishing Co., Ltd. (2011). [in Japanese]
14. Iba, Y., Tanemura, M., Omori, Y., Wago, H., Sato, S., Takahashi, A.: Computational Statistics II: Markov Chain Monte Carlo Method and Related Topics. Iwanami Shoten, Publishers (2005). [in Japanese]
15. Ikoma, N.: Particle filter—from basics to recent trends. *J. Inst. Syst., Control Inf. Eng.* **59**(5), 164–173 (2015). [in Japanese]
16. Ikoma, N.: Particle filter ~ generic non-Gaussian filter. *J. Soc. Instrum. Control Eng.* **56**(9), 644–649 (2017). [in Japanese]
17. Iwanami Data Science Publication Committee (ed.): Iwanami Data Science, vol. 6. Iwanami Shoten, Publishers (2017). [in Japanese]
18. Julier, S.J., Uhlmann, J.K., Durrant-Whyte, H.F.: A new approach for filtering nonlinear systems. *Proc. Am. Control Conf.* **3**, 1628–1632 (1995)
19. Katayama, T.: Nonlinear Kalman filter. Asakura Publishing Co., Ltd. (2011). [in Japanese]
20. Kitagawa, G.: Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *J. Comput. Graph. Stat.* **5**(1), 1–25 (1996)
21. Kitagawa, G.: A self-organizing state-space model. *J. Am. Stat. Assoc.* **93**(443), 1203–1215 (1998)
22. Kitagawa, G.: Introduction to Time Series Modeling. CRC Press (2009)
23. Kitagawa, G., Gersch, W.: Smoothness Priors Analysis of Time Series. Springer, Berlin (1996)
24. Kitagawa, G., Sato, S.: Monte Carlo smoothing and self-organising state-space model. In: Sequential Monte Carlo Methods in Practice, pp. 177–195. Springer, Berlin (2001)
25. Kitagawa, G., Takemura, A. (eds.): Statistical Science of Mathematics and Computation. Statistical Science in the 21st century III. University of Tokyo Press (2008). [in Japanese]
26. Kotecha, J.H., Djurić, P.M.: Gaussian particle filtering. *IEEE Trans. Signal Processing* **51**(10), 2592–2601 (2003)
27. Lindsten, F., Bunch, P., Särkkä, S., Schön, T.B., Godsill, S.J.: Rao-Blackwellized particle smoothers for conditionally linear Gaussian models. *IEEE J. Sel. Top. Signal Processing* **10**(2), 353–365 (2016)
28. Liu, J., West, M.: Combined parameter and state estimation in simulation-based filtering. In: Sequential Monte Carlo Methods in Practice, pp. 197–223. Springer, Berlin (2001)
29. Liu, J.S., Chen, R.: Sequential Monte Carlo methods for dynamic systems. *J. Am. Stat. Assoc.* **93**(443), 1032–1044 (1998)
30. McGee, L.A., Schmidt, S.F.: Discovery of the Kalman filter as a practical tool for aerospace and industry. Technical memorandum 86847, NASA (1985)
31. Nakano, M., Sato, S., Takahashi, A., Takahashi, S.: Optimal portfolio with particle filtering. Discussion papers CIRJE-J-277, Center for International Research on the Japanese Economy (2016). [in Japanese]
32. Nakano, S., Ueno, G., Higuchi, T.: Merging particle filter for sequential data assimilation. *Nonlinear Processes Geophys.* **14**(4), 395–408 (2007)
33. Petris, G., Petrone, S., Campagnoli, P.: Dynamic Linear Model with R. Springer, Berlin (2009)
34. Pitt, M.K., Shephard, N.: Filtering via simulation: auxiliary particle filters. *J. Am. Stat. Assoc.* **94**(446), 590–599 (1999)
35. Prado, R., West, M.: Time Series: Modeling, Computation, and Inference. CRC Press (2010)

36. Rios, M.P., Lopes, H.F.: The extended Liu and West filter: Parameter learning in Markov switching stochastic volatility models. In: Zeng, Y., Wu, S. (eds.) State-Space Models: Applications in Economics and Finance, pp. 23–61. Springer, Berlin (2013)
37. Särkkä, S.: Bayesian Filtering and Smoothing. Institute of Mathematical Statistics Textbooks, Cambridge University Press (2013)
38. West, M.: Approximating posterior distributions by mixture. *J. R. Stat. Soc. Ser. B (Methodol.)* **55**(2), 409–422 (1993)
39. Whiteley, N., Johansen, A.M.: Auxiliary particle filtering: recent developments. In: Barber, D., Cemgil, A.T., Chiappa, S. (eds.) Bayesian Time Series Models, pp. 52–81. Cambridge University Press (2011)
40. Yano, K.: A tutorial on particle filters: filters, smoothing, and parameter estimation. *J. Jpn. Stat. Soc.* **44**(1), 189–216 (2014). [in Japanese]
41. Yoshimura, T., Soeda, T.: The application of Monte Carlo methods to the nonlinear filtering problem. *IEEE Trans. Automat. Control* **17**(5), 681–684 (1972)

Chapter 12

Example of Applied Analysis in General State-Space Model



This chapter discusses approaches to appropriately consider the structural change in the flow data of the Nile. As one such approach, we have already introduced the regression model in Sect. 9.6.2. However, we have treated the change point as known. In this chapter, we first review the approach in the case of a known change point and then extend it to the case wherein the change point is unknown. Furthermore, based on such an examination, we explain real-time detection of an unknown change point.

12.1 Consideration of Structural Change

This chapter discusses the flow data of the Nile again as time series data with *structural change*. This chapter aims to properly consider the rapid decrease in the flow in 1899, which is associated with the construction of the Aswan (Low) Dam. As a study step, we first review the case with the known *change point* and then extend the idea to the case wherein the change point is unknown. Figure 12.1 shows the plot of this data again.

Incidentally, although this chapter examines a method that appropriately considers the structural change included in time series data, such change point detection is also known as a type of *anomaly detection*. Various methods have been studied and proposed for such detection, and discussions are still ongoing [1]. Some R libraries [3, 10, 16, 18] that can perform anomaly detection have been recently developed.

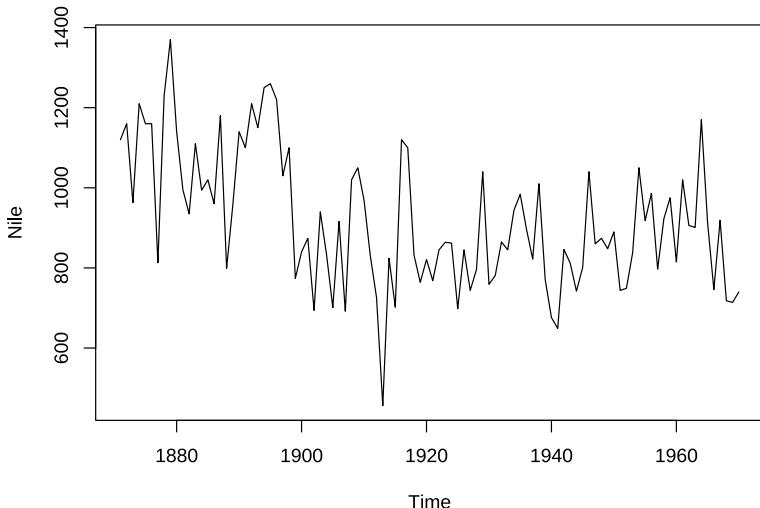


Fig. 12.1 Data on the annual flow of the Nile (reproduced)

Regarding time series data with a known change point, we have already introduced a method using the regression model in Sect. 9.6.2. This chapter now discusses an alternative approach that considers time-varying state noise variance. By considering the parameters as time-varying, we can improve the ability to follow the structural change. Thus far in this book, while we have considered that the parameters are basically time-invariant, we henceforth treat the time-varying case as well. Then, note that we refer to the term time-varying or time-invariant parameter as simply time-varying or time-invariant, respectively.

Furthermore, we extend the concept of the time-varying parameter for the case wherein the change point is unknown. We apply a fat-tailed distribution to the state noise specifically. Such a distribution makes it easy for the state noise to take a very large value. Since the state noise is the tolerance for the temporal change of the state, a situation wherein its value occasionally becomes large can adequately explain the fact that the state value changes suddenly and significantly.

As a solution in the following discussion, while a Kalman filter is applied to the “known” change point, MCMC and a particle filter are applied to the “unknown” change point. In addition, for the estimation type, we use fixed-interval smoothing to compare the results under identical conditions.

At the end of this chapter, we also consider real-time detection for an unknown change point.

By applying the above discussion of the state noise to that of the observation noise, we can handle rare outliers properly. However, this book omits the explanation.

12.2 Approach Using a Kalman Filter (Known Change Point)

This section explains how to consider properly a known change point in a structural change using a Kalman filter.

12.2.1 Time-Invariant Model Studied thus Far

For comparison, we first look back on the results obtained using the time-invariant local-level model studied thus far. This local-level model is defined by Eqs. (8.9) and (8.10) as follows:

$$x_t = x_{t-1} + w_t, \quad w_t \sim \mathcal{N}(0, W), \quad (12.1)$$

$$y_t = x_t + v_t, \quad v_t \sim \mathcal{N}(0, V). \quad (12.2)$$

Since we have already analyzed the above in Sect. 8.2, we repeat only the results for the mean of the smoothing distribution in Fig. 12.2.

We can recognize that the tracking performance for the rapid change in 1899 is not sufficient. We have specified the maximum likelihood estimates 1468.432 and 15099.8 for the variances W and V of the state and observation noises, respectively. While these results are not necessarily correct because of the real data analysis, this chapter considers them as a guideline.

12.2.2 Utilizing Prior Information in the Linear Gaussian State-Space Model

We now modify the definition of the state noise in the local-level model. Specifically, we suppose that the variance of the state noise increases “only” in 1899 to ensure that we can include the influence of rapid change in that year to the model appropriately [14]. While such an approach makes the variance of the state noise time-varying,

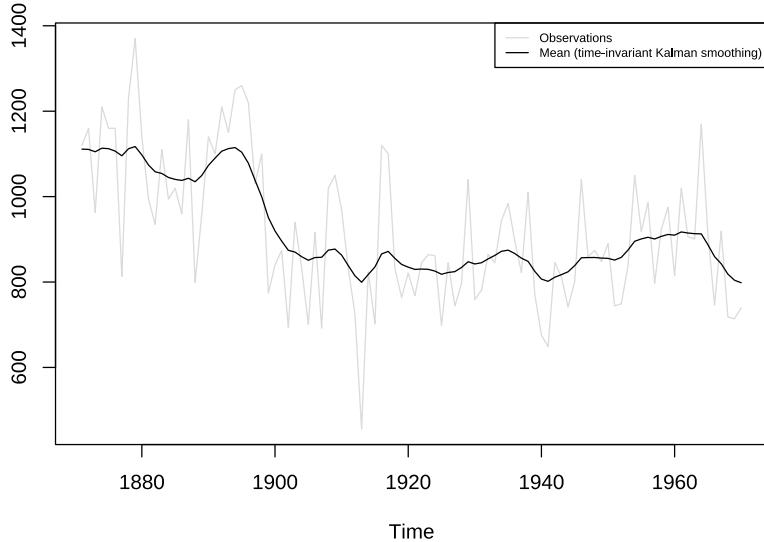


Fig. 12.2 Results of time-invariant Kalman smoothing

we treat the change point as not unknown but known. Hence, the model still follows the linear Gaussian state-space model. Such a local-level model is defined as follows:

$$x_t = x_{t-1} + w_t, \quad w_t \sim \mathcal{N}(0, W_t), \quad (12.3)$$

$$y_t = x_t + v_t, \quad v_t \sim \mathcal{N}(0, V). \quad (12.4)$$

Comparing the above equations with Eqs. (12.1) and (12.2), we see that the variance W is changed to W_t . Specifically, W_t is set as $W_{\text{other than } 1899} = W$ and $W_{1899} = \lambda^2 W$. While W is time-invariant and is common throughout all the years, the factor λ^2 exists only for 1899. Incidentally, we refer to a Kalman filter for the linear Gaussian state-space models with such time-varying parameters as a *time-varying Kalman filter*.

12.2.3 Numerical Result

The code for executing the time-varying Kalman filter for Eqs. (12.3) and (12.4) is as follows.

Code 12.1

```

1 > # <<Apply local-level model to flow data of the Nile (time-varying Kalman filter)>>
2 >
3 > # Preprocessing
4 > set.seed(123)
5 > library(dlm)
6 >
7 > # Flow data of the Nile
8 > y <- Nile
9 > t_max <- length(y)
10 >
11 > # Function building local-level model (time-varying variance of state noise)
12 > build_dlm <- function(par) {
13 +   tmp <- dlmModPoly(order = 1, dV = exp(par[1]))
14 +
15 +   # Variance of state noise refers to the first column of X
16 +   tmp$JW <- matrix(1, nrow = 1, ncol = 1)
17 +
18 +   # Store the variance of state noise in the first column of X
19 +   tmp$X <- matrix(exp(par[2]), nrow = t_max, ncol = 1)
20 +
21 +   # Allow increase of state noise only in 1899
22 +   j <- which(time(y) == 1899)
23 +   tmp$X[j, 1] <- tmp$X[j, 1] * exp(par[3])
24 +
25 +   return(tmp)
26 }
27 >
28 > # Maximum likelihood estimation of parameters
29 > fit_dlm <- dlmMLE(y = y, parm = rep(0, 3), build = build_dlm)
30 > modtv <- build_dlm(fit_dlm$par)
31 > as.vector(modtv$X); modtv$V
32 [1] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
33 [7] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
34 [13] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
35 [19] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
36 [25] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.035191e+04 6.709260e-02
37 [31] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
38 [37] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
39 [43] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
40 [49] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
41 [55] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
42 [61] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
43 [67] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
44 [73] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
45 [79] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
46 [85] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
47 [91] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
48 [97] 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02 6.709260e-02
49 [,1]
50 [1,] 16301.65
51 >
52 > # Kalman smoothing
53 > dlmSmoothed_obj <- dlmSmooth(y = y, mod = modtv)
54 >
55 > # Mean of the smoothing distribution
56 > stv <- dropFirst(dlmSmoothed_obj$s)

```

```

57 >
58 > # Plot
59 > ts.plot(cbind(y, stv),
60 +           lty=c("solid", "solid"),
61 +           col=c("lightgray", "black"))
62 >
63 > # Legend
64 > legend(legend = c("Observations", "Mean (time-varying Kalman smoothing)"),
65 +           lty = c("solid", "solid"),
66 +           col = c("lightgray", "black"),
67 +           x = "topright", cex = 0.6)

```

At the beginning of the above code, we prepare the user-defined function `build_dlm()` to create a time-varying model. Inside this function, we first create the model template `tmp` using function `dlmModPoly()`. We then set the `tmp$JW` to one (a 1×1 matrix) considering the time-varying variance of the state noise. Regarding the time-varying variance, the values are stored in the first column of `tmp$X`. Thus, we set the unique time-invariant base value to all of them and then overwrite the specific value to ensure that it can have a positive factor only in 1899. Next, the maximum likelihood method specifies the parameters using this model-building function `build_dlm()`. The variances W_t of the state noise are specified as $6.709260\text{e-}02$ except in 1899; the variance in 1899 is $6.035191\text{e+}04$, and the variance V of the observation noise is specified as 16301.65 . Figure 12.3 shows a plot for the mean of the smoothing distribution obtained from the Kalman smoother with the above-specified parameters.

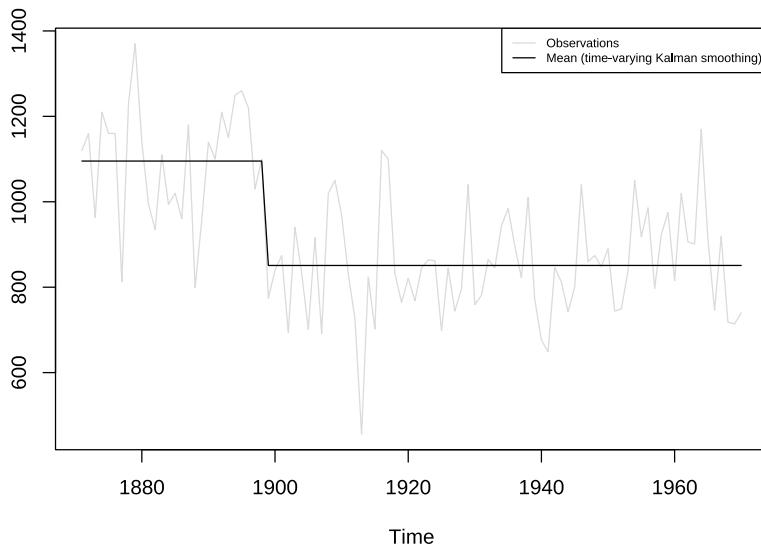


Fig. 12.3 Time-varying Kalman smoothing results

Comparing Fig. 12.3 with Fig. 12.2, we see that the result can sharply track the rapid change in 1899. Furthermore, we see that the estimated level is almost constant before and after 1899. While this is not necessarily the correct answer because of the real data analysis, this chapter considers this result as a guideline for tracking the rapid change in 1899.

12.3 Approach Using MCMC (Unknown Change Point)

This section explains how to consider properly an unknown change point in a structural change using MCMC-based solutions.

12.3.1 Time-Invariant Model Studied thus Far

For comparison, we first look back on the results obtained using the time-invariant local-level model studied thus far. The definition for the local-level model does not differ from Eqs. (12.1) and (12.2). Herein, we regard the parameters W and V as random variables and estimate them jointly with the state. Regarding this model, we first estimate only the parameters W and V using MCMC based on the explanation in Sect. 10.5. However, we then estimate the state through not FFBS but by substituting the mean of the parameter estimates into the Kalman filter [9]. The principal reason for using such a simple method is to match the comparison conditions with the following description regarding the particle filter.

Regarding Stan code, we can directly reuse Code 10.5 for parameter estimation. In addition, regarding R code, we can reuse Code 10.6 almost as it is. Therefore, we omit their display. The mean of the variances W and V are estimated as 2865.51 and 14606.24, respectively. While the variance W is slightly larger than the result specified by the maximum likelihood method, the variance V is close to its maximum likelihood estimates (MLE). Figure 12.4 shows the results for the mean of the state in the smoothing distribution.

We recognize that the tracking performance for the rapid change in 1899 is not sufficient. Compared with the results of time-invariant Kalman smoothing, we see that the entire fluctuation is slightly intense because the estimation result for the variance W of the state noise is greater than its maximum likelihood estimates.

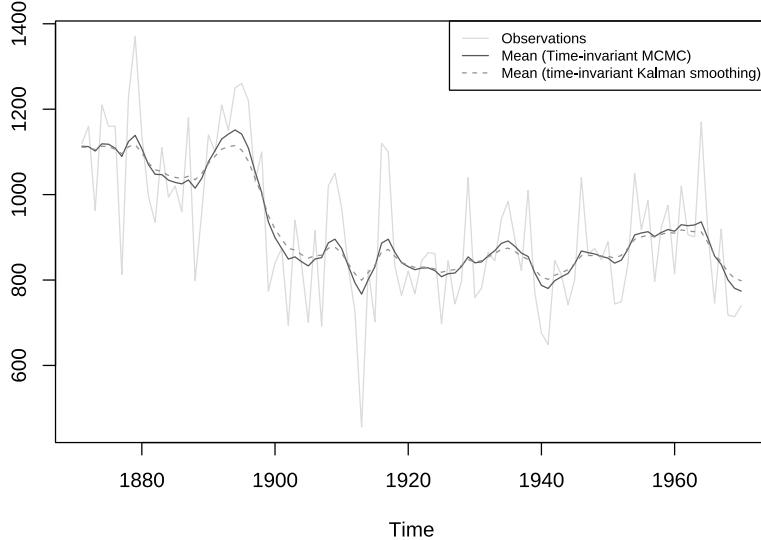


Fig. 12.4 Time-invariant MCMC results

12.3.2 Use of a Horseshoe Distribution in the General State-Space Model

In the previous section, we specified a time-varying model based on the prior information regarding a historical fact. However, such information is not always available. To properly consider a rarely occurring structural change even without prior information, we try to apply a distribution other than the normal distribution to the state noise. If we can assume the degree of “rare” to some extent, it might be better to use mixtures of Gaussians [11]. On the contrary, it is typical to use a *fat-tailed distribution* in a circumstance wherein we cannot clearly assume such a degree. While the *t*-distribution and Cauchy distribution (*t*-distribution with one degree of freedom) are often used as a fat-tailed distribution, the *horseshoe distribution* has recently been proposed. This distribution can enhance the thickness of the tail and concentration around the mode [5–7]. The horseshoe distribution is defined through a scale mixture of the normal distribution as follows:

$$\text{Horseshoe distribution (center} = 0, \text{scale} = \sigma) = \mathcal{N}(0, \lambda_t^2 \sigma^2), \quad (12.5)$$

$$\lambda_t \sim \mathcal{C}^+(\text{center} = 0, \text{scale} = 1), \quad (12.6)$$

where \mathcal{C}^+ is the Cauchy distribution whose support is limited positive and is referred to as a half-Cauchy distribution. The above Eq. (12.5) implies that the normal distribution becomes a horseshoe distribution by considering the time-varying factor λ_t , which follows the half-Cauchy distribution, to the scale (standard deviation) of

the normal distribution. As can be seen from Eq. (12.6), the time-varying factor is usually less than one but is sometimes large; hence, while the horseshoe distribution almost yields a value smaller than σ , it sometimes yields a large value.

The scale mixture of the normal distribution is useful. If we set $\lambda_t^2 \sim \text{inverse gamma distribution } (\nu/2, \nu/2)$ instead of using Eq. (12.6), we obtain a t -distribution with ν degrees of freedom [2, Sect. 2.3.7]. Reference [14] uses such an approach to detect a structural change in time series data.

The horseshoe distribution is suitable for expressing the *sparsity* because of the thickness of the tail and the high concentration around the mode. While the Laplace (double exponential) distribution is also used for expressing sparsity, the horseshoe distribution can enhance further the concentration around the mode. For confirmation of such a property, Fig. 12.5 compares the density of the Cauchy and Laplace distributions with that of the horseshoe distribution; the parameters for all distributions are set to center = 0 and scale = 1.

Since we cannot express the horseshoe distribution analytically, we show the simulation frequency for a sample size of 50,000. The horseshoe distribution can increase the concentration around the mode while maintaining the tail thickness same as that in the Cauchy and Laplace distributions.

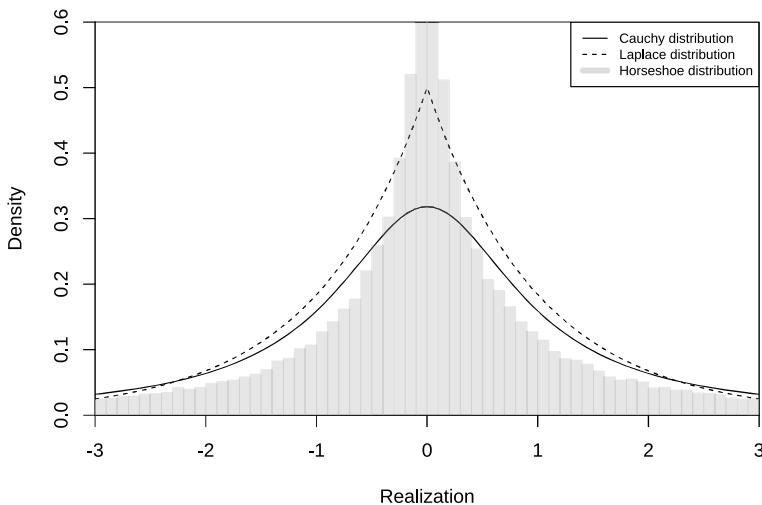


Fig. 12.5 Frequency of horseshoe distribution

A further feature of the horseshoe distribution is its computational advantage. Since this distribution is expressed as a scale mixture of the normal distribution from the definition, we can use the linear Gaussian state-space model conditionally. Herein, the scale factor λ_t in Eq. (12.5) is time-varying. This implies that the solution requires a time-varying Kalman filter.

Based on the abovementioned attractive features, we apply the horseshoe distribution to the state noise. In this case, the local-level model is defined as follows:

$$x_t = x_{t-1} + w_t, \quad w_t \sim \text{Horseshoe distribution}(\text{center} = 0, \text{scale} = \sqrt{W}), \quad (12.7)$$

$$y_t = x_t + v_t, \quad v_t \sim \mathcal{N}(0, V). \quad (12.8)$$

Comparing the above with Eqs. (12.3) and (12.4), we see that the distribution for w_t is changed from the normal distribution to the horseshoe distribution. Thus, while w_t usually takes a value smaller than \sqrt{W} , it sometimes takes a large value.

12.3.3 Numerical Result

We now examine an example wherein the state noise follows the horseshoe distribution. When we use a horseshoe distribution, the conditional linear Gaussian state-space model requires a time-varying Kalman filter for the solution. The function `gaussian_dlm_obs()` in **Stan** 2.19.2 does not yet support the time-varying linear Gaussian state-space model. Regarding this point, while future revision is expected, Dr. Jeffrey B. Arnold, who contributed to the implementation of `gaussian_dlm_obs()`, has implemented a time-varying Kalman filter as a user-defined function. We can download the source code from <https://raw.githubusercontent.com/jrnold/dlm-shrinkage/master/stan/includes/dlm.stan> and then decide to use this function in the analysis below.

This source code contains many other utility functions such as one executing the FFBS in the `generated_quantities` block. The author recommends that readers confirm their contents.

When the state noise of the local-level model follows a horseshoe distribution, the code for **Stan** is as follows.

Code 12.2

```

1 // mode12-1.stan
2 // Model: specification (local-level model + horseshoe distribution with unknown
2 ← parameters, utilizing time-varying Kalman filter)
3
4 /*
5 * The contents of the forefront function block are borrowed from the following site
5 ← (partly modified to avoid the latest rstan's warning)
6 * https://raw.githubusercontent.com/jrnold/dlm-shrinkage/master/stan/includes/dlm.st
6 ← an
7 */
8
9 functions{
10    // --- mode: stan ---
11    // --- BEGIN: dlm ---
12
13 ... Ignore the interim display ...
14
15 // --- END: dlm ---
16 }
17
18 data{
19    int<lower = 1>          t_max; // Time series length
20    vector[t_max]            y; // Observations
21
22    int                      miss[t_max]; // Indicator for missing values
23    real                     m0; // Mean of prior distribution
24    real<lower = 0>          CO; // Variance of prior distribution
25 }
26
27 parameters{
28    vector<lower = 0>[t_max] lambda; // Standard deviation of state noise
28 ← (time-varying factor)
29    real<lower = 0>          W_sqrt; // Standard deviation of state noise
29 ← (time-invariant base)
30    real<lower = 0>          V_sqrt; // Standard deviation of observation noise
30 ← (time-invariant)
31 }
32
33 transformed parameters{
34    vector[t_max]            W; // Variance of state noise (time-varying)
35    vector[t_max]            V; // Variance of observation noise (time-varying)
36    real                      log_lik; // Log-likelihood through all time points
37
38 // Variance of state noise (different value is set at every time point)
39 for (t in 1:t_max) {
40    W[t] = pow(lambda[t] * W_sqrt, 2);
41 }
42
43 // Variance of observation noise (unique value is set at all time points)
44 V = rep_vector(pow(V_sqrt, 2), t_max);
45
46 // The final log-likelihood
47 log_lik = sum(dlm_local_level_loglik(t_max, y, miss, V, W, m0, CO));
48 }
49
50 model{
51    // Likelihood part

```

```

52 target += log_liik;
53
54 // Prior part
55 /* Standard deviation of state noise (time-varying factor) */
56 lambda ~ cauchy(0, 1);
57
58 /* Standard deviation of state noise (time-invariant base) and standard deviation
59 ← of observation noise (time-invariant): noninformative prior distribution */
59 }

```

At the beginning of the above code, the `functions` block describes the user-defined functions; owing to its excessive length, we ignore the interim display. In the block, we define various functions, such as a generic function that performs Kalman filtering to time-varying linear Gaussian state-space models and specialized function `dlm_local_level_loglik()` for time-varying local-level modes. This example uses the latter specialized function. The arguments of this function are the time series length `t_max`, data `y`, missing observations index `miss`, time-varying variance `V` of the observation noise, time-varying variance `W` of the state noise, mean `m0` of the prior distribution, and variance `C0` of the prior distribution. In the following `data` block, we set `t_max`, `y`, `miss`, `m0`, and `C0`. These data types are consistent with the arguments of the function `dlm_local_level_loglik()`. In the following `parameters` block, we declare the scale factor `lambda` for the standard deviation of the state noise, standard deviation `W_sqrt` (time-invariant base) of the state noise, and standard deviation `V_sqrt` (time-invariant) of the observation noise. In the following `transformed parameters` block, we first transform the parameters declared in the `parameters` block to ensure that they match the arguments of the function `dlm_local_level_loglik()`. Specifically, the time-varying variance `W` of the state noise is set according to Eq. (12.5), and the time-varying variance `V` of the observation noise is set to a unique value at all time points. In addition, since the function `dlm_local_level_loglik()` can obtain the vectors of log-likelihoods for all time points, we sum them to derive the entire log-likelihood `log_liik`. In the following `model` block, we describe the posterior distribution. In the likelihood part, we use the notation “`target += log-likelihood`,” which directly describes the log-likelihood instead of “`observations ~ likelihood`.” In the prior distribution part, we apply the half-Cauchy distribution to the prior distribution for the `lambda` based on Eq. (12.6); we added the condition `<lower = 0>` at the declaration of `lambda`, resulting in the range of its sample being limited positive. We omit the description of the prior distributions for the standard deviation (time-invariant base) of the state noise and standard deviation (time-invariant) of the observation noise; hence, the noninformative prior distributions are applied to them.

The R code for executing Code 12.2 is as follows.

Code 12.3

```

68 > # <<Apply local-level model to flow data of the Nile (time-varying MCMC: horseshoe
69 > prior>>
70 >
71 > # Preprocessing
72 > set.seed(123)
73 > library(rstan)
74 >
75 > # Presetting of Stan: HDD storage of compiled code and parallel computation
76 > rstan_options(auto_write = TRUE)
77 > options(mc.cores = parallel::detectCores())
78 >
79 > # Model: generation and compilation
80 > stan_mod_out <- stan_model(file = "model12-1.stan")
81 >
82 > # Smoothing: execution (sampling)
83 > fit_stan <- sampling(object = stan_mod_out,
84 +                         data = list(t_max = t_max, y = y,
85 +                                     miss = as.integer(is.na(y)),
86 +                                     m0 = modtv$m0, C0 = modtv$C0[1, 1]),
87 +                         pars = c("lambda", "W_sqrt", "V_sqrt"),
88 +                         seed = 123
89 +                     )
90 >
91 > # Confirmation of the results
92 > oldpar <- par(no.readonly = TRUE); options(max.print = 99999)
93 > print(fit_stan, probs = c(0.025, 0.5, 0.975))
94 Inference for Stan model: model12-1.
95 4 chains, each with iter=2000; warmup=1000; thin=1;
96 post-warmup draws per chain=1000, total post-warmup draws=4000.
97
98      mean se_mean     sd    2.5%     50%    97.5% n_eff Rhat
99 lambda[1]   3.47    0.28 13.43   0.05    1.02   21.08   2284    1
100 ... Ignore the interim display ...
101
102 lambda[100]  3.04    0.22 12.76   0.04    0.97   16.45   3235    1
103 W_sqrt       4.64    0.10  4.21   0.40    3.41   16.34   1911    1
104 V_sqrt      128.00   0.17 10.04  109.81  127.63  148.66   3613    1
105 lp__      -768.07   0.26  9.28 -786.94 -767.75 -751.17  1232    1
106
107 Samples were drawn using NUTS(diag_e) at Sat Apr 25 17:01:54 2020.
108 For each parameter, n_eff is a crude measure of effective sample size,
109 and Rhat is the potential scale reduction factor on split chains (at
110 convergence, Rhat=1).
111 > par(oldpar)
112 > tmp_tp <- traceplot(fit_stan, pars = c("W_sqrt", "V_sqrt"), alpha = 0.5)
113 > tmp_tp + theme(aspect.ratio = 3/4)
114 >
115 > # Modify the model copied from Kalman filter
116 > modtv_MCMC <- modtv
117 > modtv_MCMC$X[, 1] <- (summary(fit_stan)$summary[ 1:100, "mean"] *
118 +                           summary(fit_stan)$summary["W_sqrt", "mean"])^2
119 > modtv_MCMC$V[1, 1] <- (summary(fit_stan)$summary["V_sqrt", "mean"])^2
120 > as.vector(modtv_MCMC$X); modtv_MCMC$V
121 [1] 259.31065 143.73234 163.84145 127.17156 195.93944 161.00466
122 [7] 142.31396 171.30339 117.29581 210.72479 208.56738 172.30525

```

```

123 [13] 153.44244   164.69939   156.99696   148.62297   141.67938   141.47493
124 [19] 182.99432   342.03580   321.00342   275.24018   200.67405   202.29153
125 [25] 170.35374   239.66732   5293.58784  10608.18678  303593.72100  2364.88097
126 [31] 412.69492   185.79406   154.96661   128.55400   137.75630   223.15976
127 [37] 118.35416   173.16014   131.71138   145.50848   197.78681   200.15788
128 [43] 166.11988   176.76204   144.47785  330.03324   129.26414   186.39054
129 [49] 207.45509   127.36625   109.10133   102.76346   118.77724   119.82864
130 [55] 124.77253   137.68412   141.48371   176.00457   134.52922   138.40540
131 [61] 103.76063   141.27565   120.21805   119.40603   127.04966   189.09129
132 [67] 108.42100   97.90159   170.66662   164.46438   148.04669   131.86951
133 [73] 145.94902   137.91529   185.44609   195.15200   134.31272   107.16847
134 [79] 132.48612   125.44544   114.98999   142.29877   174.64198   252.68935
135 [85] 133.57347   164.06345   140.43189   138.53463   106.49137   134.24959
136 [91] 144.15681   116.64762   154.21627   156.86225   672.75611   545.18915
137 [97] 284.05554   483.23529   267.37267   199.37083
138 [,1]
139 [1,] 16384.74
140 >
141 > # Kalman smoothing
142 > dlmSmoothed_obj <- dlmSmooth(y = y, mod = modtv_MCMC)
143 >
144 > # Mean of the smoothing distribution
145 > stv_MCMC <- dropFirst(dlmSmoothed_obj$s)
146 >
147 > # Plot
148 > ts.plot(cbind(y, stv_MCMC, stv),
149 +           lty=c("solid", "solid", "dashed"),
150 +           col=c("lightgray", "blue", "red"))
151 >
152 > # Legend
153 > legend(legend = c("Observations", "Mean (time-varying MCMC: horseshoe
154 > distribution)", "Mean (time-varying Kalman smoothing)"),
155 +           lty = c("solid", "solid", "dashed"),
156 +           col = c("lightgray", "blue", "red"),
+           x = "topright", cex = 0.7)

```

The above code continues from Code 12.1. Since the contents are almost the same as Code 10.6, we focus on the differences. The data type for the argument of the function `sampling()` conforms to the specification of the function `dlm_local_level_loglik()` in Code 12.2. The argument `miss` is a vector of all 0s because the data `y` in this example have no missing observations. We ignore the interim console display of the estimation result `fit_stan` because there are many estimated parameters. From the result, we can confirm that neither the effective sample size nor \hat{R} have any particular problem. Figure 12.6 shows the trace plots for the `W_sqrt` and `V_sqrt`; there is no particular problem with their mixing.

Finally, we derive the smoothing distribution of the state using a time-varying Kalman filter. We copy the time-varying model `modtv` prepared in Code 12.1 to the `modtv_MCMC`, and then we make necessary corrections to it. Specifically, we substitute the estimated mean values using MCMC into the `X` and `V`, which store the time-varying variance of the state noise and time-invariant variance of the observation noise, respectively. Confirming the contents of the time-varying variance of the state noise, we see that it takes the maximum value of 303593.721 in 1899 (29th-time point) and smaller values at other time points. In addition, the variance of the obser-

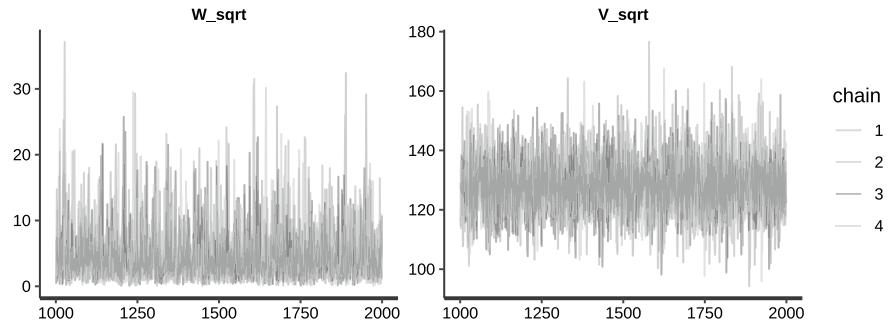


Fig. 12.6 Trace plots (time-varying MCMC: horseshoe distribution)

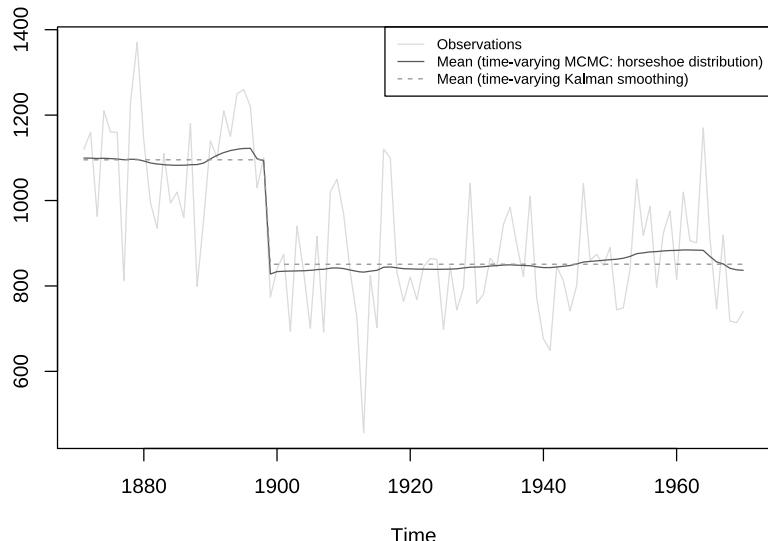


Fig. 12.7 Time-varying MCMC: results using a horseshoe distribution

vation noise takes the value of 16384.74, which is close to the maximum likelihood estimates. Figure 12.7 shows the plot of the mean of the smoothing distribution.

Confirming Fig. 12.7, we see that the results can sharply track the rapid change in 1899. Furthermore, we see that the level before and after 1899 shows some fluctuation compared with that obtained using time-varying Kalman smoothing.

Incidentally, while we have confirmed the results when the state noise follows the horseshoe distribution, [13] has introduced a local-level model whose state noise follows a time-varying Cauchy distribution as a similar model. Although the author applied this model to the Nile data and could confirm almost the same result, we ignore the details.

12.4 Approach Using a Particle Filter (Unknown Change Point)

This section explains how to consider properly an unknown change point in a structural change using a particle filter.

12.4.1 Time-Invariant Model Studied thus Far

For comparison, we first look back on the results of the time-invariant local-level model examined thus far. The definition for the local-level model is not different from Eqs. (12.1) and (12.2). Herein, we regard the parameters W and V as random variables and estimate them jointly with the state. Regarding this model, we first estimate the parameters W and V using the Rao–Blackwellized Liu and West filter as in Sect. 11.5. We then estimate the smoothing distribution of the state by substituting the parameter estimates into the Kalman filter. Specifically, we use the mean values at the last time point in the filtering as the plug-in values for the time-invariant parameters W and V according to [4, 15].

The straightforward application of the particle smoothing algorithm described in Sect. 11.1.3 to the Rao–Blackwellized particle filter results in the smoothing state being represented through reselection of the filtering state. In this case, we cannot directly correct the state value unlike the Kalman smoothing. For this reason, several smoothing algorithms specialized for the Rao–Blackwellized particle filter using Kalman smoothing have been proposed [8, 12, 17]. However, as of this writing, there appears to be no standard algorithm. This chapter adopts a simple method of substituting the mean value of the posterior distribution with the parameters into the Kalman filter to derive the smoothing distribution of the state while matching the comparison conditions with the MCMC-based solution.

Regarding the Rao–Blackwellized Liu and West filter, we can reuse Code 11.4 almost as it is. Therefore, we have not shown the code. Figure 12.8 shows the estimation result for parameters.

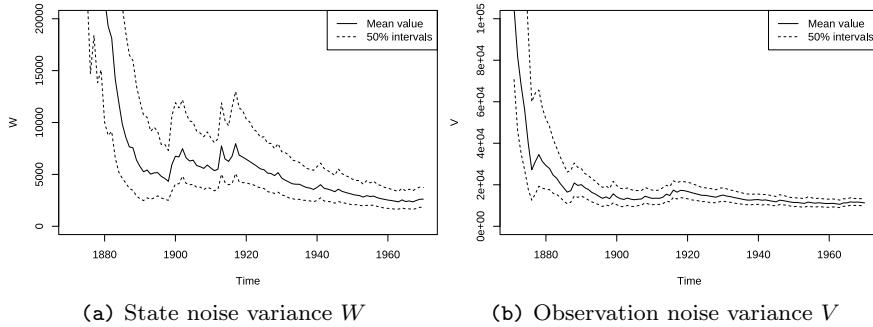


Fig. 12.8 Parameter estimation results (time-invariant Rao–Blackwellized Liu and West filter)

Comparing each mean value at the last time point with the maximum likelihood estimates, we see that the variance $W(2615.615)$ of the state noise is slightly greater than the MLE, and the variance $V(11288.07)$ of the observation noise is almost the same as MLE. Confirming the estimation results regarding the variance W of the state noise carefully, we see that the value increases in response to the rapid change in 1899 and the temporary large fluctuations around 1916. However, since we assume a normal distribution for the state noise, their values have not reached an extreme increase. In addition, owing to the influence of such fluctuations, the estimates for W still appear to be large even at the last time point. Subsequently, Fig. 12.9 shows the results on the smoothing distribution of the state.

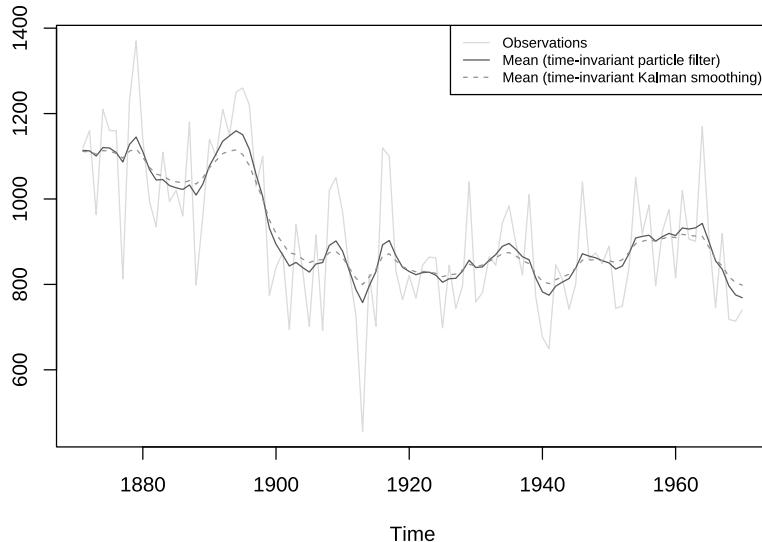


Fig. 12.9 Time-invariant particle filter results

We recognize that the tracking performance for the rapid change in 1899 is not sufficient. In comparison with the time-invariant Kalman smoothing result, we see that the entire fluctuation is slightly intense because the estimation result for the variance W of the state noise is greater than the maximum likelihood estimates.

12.4.2 Use of a Horseshoe Distribution in the General State-Space Model

In the previous section, we described the model whose state noise follows horseshoe distribution in Eqs. (12.7) and (12.8). We also confirmed its numerical results using MCMC. We can apply the same model to the particle filter and here confirm the result. However, some cautions are called for in regard to such an implementation owing to its unique nature; hence, we supplement them.

First, we assume the Rao–Blackwellized Liu and West filter for the particle filter to be used as the base. The Liu and West filter uses kernel smoothing to estimate the parameters, and the parameter estimates converge as time passes. While this scheme works well with time-invariant parameters, it does not work well with time-varying ones. The horseshoe distribution contains a time-varying factor related to the scale parameter, and it must be prepared for an unexpected change point. Thus, the convergence of such a value is undesirable, requiring us to modify the treatment. For this purpose, we decide not to apply kernel smoothing to the time-varying factor and merely to draw its sample from the half-Cauchy distribution at each time point. We also apply particle smoothing based on the Kitagawa algorithm to the time-varying factor and use the mean of its marginal smoothing distribution as their point estimates. Incidentally, we adopt “1” as the best guess for the time-varying factor at the current time point based on Eq. (12.6).

12.4.3 Numerical Results

We now analyze an example wherein the state noise of the local-level model follows the horseshoe distribution. This code is as follows.

Code 12.4

```

157 > # <<Apply local-level model to flow data of the Nile (time-varying particle filter:
158 >   → horseshoe prior)>>
159 >
160 > # Preprocessing
161 > set.seed(123)
162 >
163 > # Presetting of particle filter
164 > N <- 10000           # Number of particles
165 > a <- 0.975          # Exponential weight in artificial moving average for
   → parameters
166 > W_max <- 10 * var(diff(y))    # Guess maximum value for parameter W
167 > V_max <- 10 * var(      y )  # Guess maximum value for parameter V
168 >
169 > # *Note: Assuming that the time point of the prior distribution corresponds to one,
   → we regard the shifted time points (from 2 to t_max+1) as the original ones (from
   → 1 to t_max).
170 >
171 > # Data formatting (adding the forefront dummy corresponding to prior distribution)
172 > y <- c(NA_real_, y)
173 >
174 > # Save the index sequence for resampling at every time point
175 > k_save <- matrix(1:N, nrow = N, ncol = t_max+1)
176 >
177 > # Setting of prior distribution
178 >
179 > # Particle (realizations): variance of state noise (time-varying factor)
180 > lambda2      <- matrix(NA_real_, nrow = t_max+1, ncol = N)
181 > lambda2[1, ] <- log(rcauchy(N)^2)           # Log domain (no actual
   → impact on estimation)
182 >
183 > W      <- matrix(NA_real_, nrow = t_max+1, ncol = N)
184 > W[1, ] <- log(runif(N, min = 0, max = W_max))      # Log domain
185 >
186 > # Particle (realizations): parameter V (time-invariant)
187 > V      <- matrix(NA_real_, nrow = t_max+1, ncol = N)
188 > V[1, ] <- log(runif(N, min = 0, max = V_max))      # Log domain
189 >
190 > # Particle (realizations): state (the mean and variance of the filtering
   → distribution)
191 > m <- matrix(NA_real_, nrow = t_max+1, ncol = N)
192 > m[1, ] <- modtv$m0           # Prior distribution with
   → unknown parameters
193 > C <- matrix(NA_real_, nrow = t_max+1, ncol = N)
194 > C[1, ] <- modtv$C0           # Prior distribution with
   → unknown parameters
195 >
196 > # Particle (weight)
197 > w <- matrix(NA_real_, nrow = t_max+1, ncol = N)
198 > w[1, ] <- log(1 / N)
199 >
200 > # Setting of the progress bar
201 > progress_bar <- txtProgressBar(min = 2, max = t_max+1, style = 3)
202 >
203 > # Time-forward processing: kernel smoothing + auxiliary particle filter +
   → Rao-Blackwellization

```

```

204 > for (t in (1:t_max)+1){
205 +   # Display progress bar
206 +   setTxtProgressBar(pb = progress_bar, value = t)
207 +
208 +   # Artificial moving average for parameters
209 +   W_ks <- kernel_smoothing(realization = W[t-1, ], w = w[t-1, ], a = a)
210 +   V_ks <- kernel_smoothing(realization = V[t-1, ], w = w[t-1, ], a = a)
211 +
212 +   # (equivalent) Resampling
213 +
214 +   # Kalman filtering for one time point -> auxiliary variable sequence
215 +   KF_aux <- Kalman_filtering(y = y[t],
216 +                                 state = list(m0 = m[t-1, ], CO = C[t-1, ]),
217 +                                 param = list(W = log(1)+W_ks$mu, V = V_ks$mu)
218 +                                 )
219 +   probs <- w[t-1, ] +
220 +     dnorm(y[t], mean = KF_aux$f, sd = sqrt(KF_aux$Q), log = TRUE)
221 +   k <- sys_resampling(N = N, w = normalize(probs))
222 +   k_save[, t] <- k           # Save indices at every time point for particle
223 →   smoothing (Kitagawa algorithm)
224 +
225 +   # Refresh all samples at each time point
226 +   lambda2[t, ] <- log(rcauchy(N)^2)
227 +
228 +   # Draw realizations of parameters from a continuous proposal distribution
229 →   (refreshment)
230 +
231 +   W[t, ] <- rnorm(N, mean = W_ks$mu[k], sd = W_ks$sigma)
232 +   V[t, ] <- rnorm(N, mean = V_ks$mu[k], sd = V_ks$sigma)
233 +
234 +   # State: Kalman filtering for one time point -> derivation of particles
235 →   (realizations)
236 +   KF <- Kalman_filtering(y = y[t],
237 +                           state = list(m0 = m[t-1, k], CO = C[t-1, k]),
238 +                           param = list(W = lambda2[t, ]+W[t, ], V = V[t, ]))
239 +
240 +   m[t, ] <- KF$m
241 +   C[t, ] <- KF$C
242 +
243 +   # Update particle (weight)
244 +   w[t, ] <- dnorm(y[t], mean = KF$f, sd = sqrt(KF$Q), log = T) -
245 +     dnorm(y[t], mean = KF_aux$f[k], sd = sqrt(KF_aux$Q[k]), log = T)
246 +
247 +   # Normalization of weight
248 +   w[t, ] <- normalize(w[t, ])
249 +
250 |=====| 100%
251 >
252 > # Ignore the display of following codes

```

The above code continues from Code 12.3. Since the content is an extension of Code 11.14, we focus on the differences. We first increase the number of particles to 10,000 because we expect this problem to be a difficult task that captures the structural change. In addition, to perform particle smoothing based on the Kitagawa algorithm, we prepare a variable `k_save` for saving the resampling index at every time point. Furthermore, we prepare a variable `lambda2` for the (logarithmic) time-varying factor relevant to the variance of the state noise, and its prior distribution is set to the (logarithmic) square of the sample from the Cauchy distribution. This setting of the prior distribution actually has no impact on the estimation because the samples

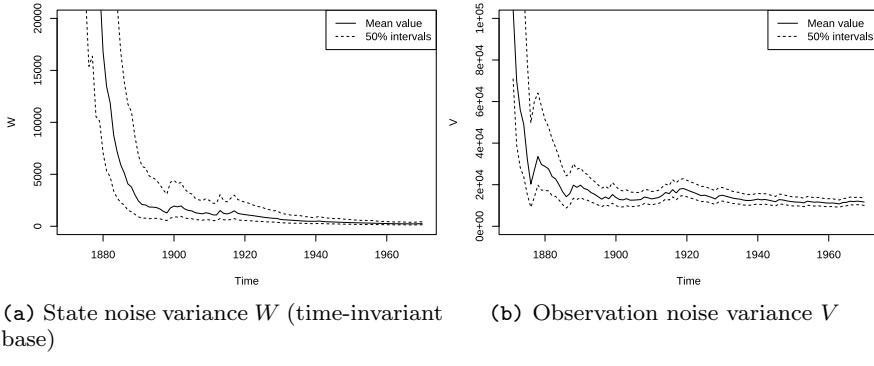


Fig. 12.10 Parameter estimation results (time-varying Rao–Blackwellized Liu and West filter)

in the `lambda2` are fully refreshed at each time point in this implementation; see the corresponding shaded code.

Figure 12.10 shows the parameter estimation results.

Comparing the mean of the variance V at the last time point with the maximum likelihood estimates, we see that the obtained value 11671.96 is almost the same as MLE. Confirming the estimation results regarding the variance W (time-invariant base) carefully, we see that while the value responds to the rapid change in 1899 and the temporary fluctuations around 1916, their impact is not so tremendous. This is because the time-varying factor to W absorbs the impact. Thus, the estimates for W can become stably convergent. In addition, Fig. 12.11 shows the smoothing distribution of the time-varying factor λ_t^2 to the W .

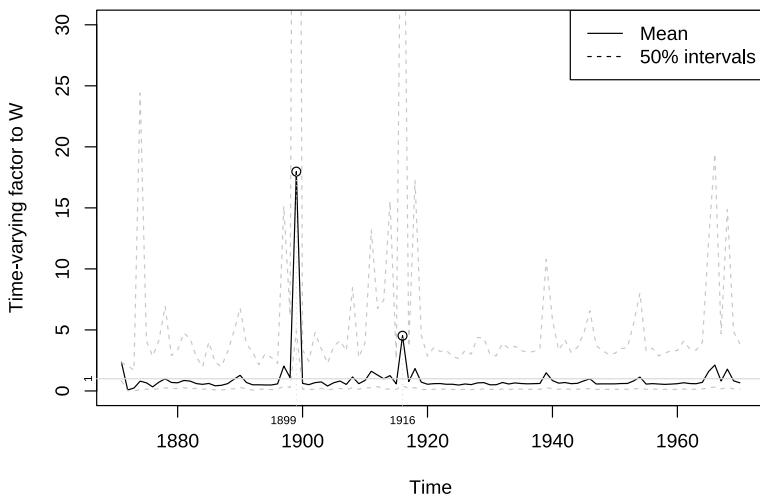


Fig. 12.11 Parameter estimation results (time-varying factor λ_t^2 to the W)

Examining the mean in Fig. 12.11, we see that while the value is almost always less than 1, it takes a large value in 1899 and 1916. Although an unusually large value in 1899 is expected, it is unexpected that the mean takes a somewhat large value in response to the temporary fluctuation around 1916. Regarding this point, we will have to eventually examine the estimation results for the state as well.

Subsequently, we obtain the smoothing distribution of the state using the time-varying Kalman filter. In the code, we copy the time-varying model `modtv` prepared in Code 12.1 and then correct it as necessary. Specifically, we substitute the parameter estimated using the time-varying Rao–Blackwellized Liu and West filter into the X and V , which store the time-varying variance of the state noise and the variance of the observation noise, respectively. We omit the code display here. Figure 12.12 shows the obtained smoothing distribution of the state.

Examining Fig. 12.12, we see that the result can sharply track the rapid change in 1899. While the time-varying factor λ_t^2 to W takes a somewhat large value around 1916, an estimation result of the state results in no significant fluctuation. In addition, we recognize that the level before and after 1899 shows some fluctuation compared with the result obtained through time-varying Kalman smoothing. Comparing Fig. 12.12 with Fig. 12.7, the result of estimating the same model using MCMC, we see that the results are almost consistent.

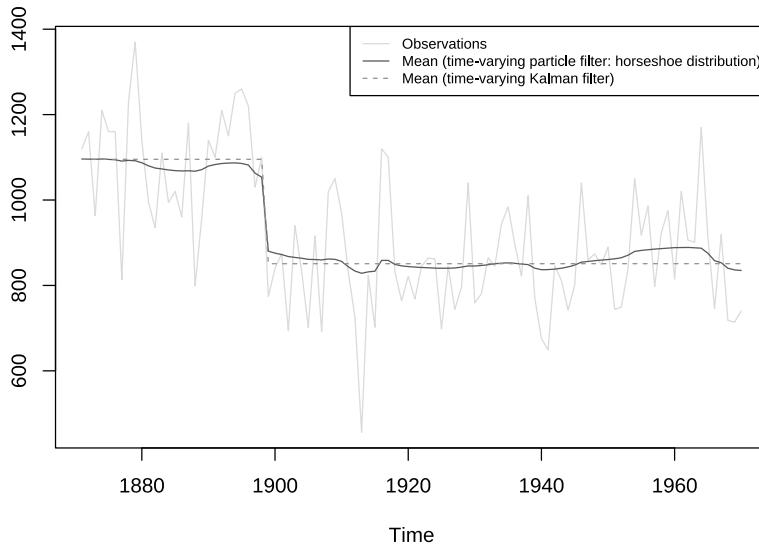


Fig. 12.12 Result by time-varying particle filter

12.5 Real-Time Detection for an Unknown Change Point

Thus far in this chapter, we have discussed methods for properly considering a known or unknown change point in structural change using several approaches, and fixed-interval smoothing has been assumed to unify the examination condition through them. This section extends the previous descriptions to detect unknown change points in real time. We have considered a general state-space model for considering an unknown change point. Thus, as a starting point for the examination, we have two choices: a solution using MCMC and a particle filter. As mentioned in Sect. 10.1, when we apply a solution using MCMC to sequentially obtained data, we must recalculate the MCMC iteration from the beginning each time new data are obtained. The MCMC-based solution is not well suited to this problem from a pragmatic point of view. Thus, this section decides to use a particle filter as the base for the solution. Regarding the criteria for detecting an unknown change point using the particle filter, we propose to judge that a structural change occurs when the time-varying scaling factor related to the variance of the state noise discussed in the previous section exceeds a specified threshold.

Regarding code implementation, we partially revise the code in the previous section. The study in this section requires real-time analysis; hence, the estimation type is limited to the filtering or fixed-lag smoothing. We have already obtained the filtering results through the previous code; hence, the modification is limited to the smoothing part. Specifically, we can achieve such a modification by changing the smoothing interval in the Kitagawa algorithm to that for fixed-lag smoothing, as shown in the following code.

Code 12.5

```

1 > # <<Fixed-lag smoothing in particle filter>>
2 >
3 > # User-defined function obtaining indices to reselect filtering particles
   ← considering future information
4 > smoothing_index <- function(t_current, lag_val){
5 +   # Index sequence at current time t_current
6 +   index <- 1:N
7 +
8 +   # Virtual repetition of resampling from t_current + 1 to t_current + lag_val
9 +   for (t in (t_current+1):ifelse(t_current + lag_val <= t_max,
10 +                                     t_current + lag_val, t_max)){
11 +     index <- index[k[, t]]
12 +   }
13 +
14 +   # Return the final reselecting index from virtual repetition of resampling
15 +   return(index)
16 + }
```

In the above code, we shade the difference from Code 11.3. While we use this function to obtain the smoothed time-varying factor, we omit the code display because it is almost the same as previously. Figure 12.13 shows the result.

This result plots the estimated mean in filtering and fixed-lag smoothing. Examining Fig. 12.13, we see that filtering does not yield a precise result. On the other

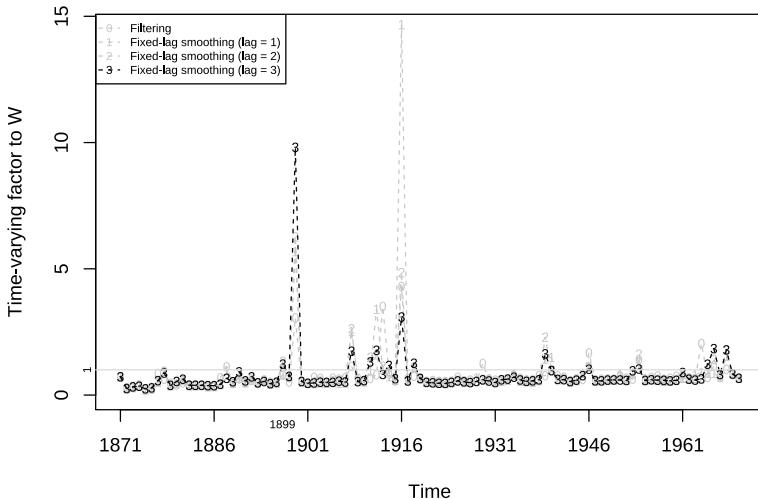


Fig. 12.13 Time-varying factor λ_t^2 to the W (mean of filtering and fixed-lag smoothing distributions)

hand, since fixed-lag smoothing uses more information from the relative future as the lag length is increased, it can enhance estimation accuracy; increasing the lag to three yields precise results. While the details of the optimal lag length and threshold depend on the problem, from the above result, this approach appears to capture an unknown change point in real time.

References

1. Adams, R.P., MacKay, D.J.C.: Bayesian online changepoint detection (2007)
2. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, Berlin (2006)
3. Boehmke, B., Gutierrez, R.: `anomalyDetection`: Implementation of Augmented Network Log Anomaly Detection Procedures (2017). <https://CRAN.R-project.org/package=anomalyDetection>. R package version 0.1.1
4. Carvalho, C.M., Johannes, M.S., Lopes, H.F., Polson, N.G.: Particle learning and smoothing. *Stat. Sci.* **25**(1), 88–106 (2010)
5. Carvalho, C.M., Polson, N.G., Scott, J.G.: Handling sparsity via the horseshoe. In: Proceedings of the 12th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 5, pp. 73–80 (2009)
6. Carvalho, C.M., Polson, N.G., Scott, J.G.: The horseshoe estimator for sparse signals. *Biometrika* **97**(2), 465–480 (2010)
7. Consider the distribution of shrinkage factor in the sparse model \sim introduction of horseshoe prior~. <http://statmodeling.hatenablog.com/entry/shrinkage-factor-and-horseshoe-prior>. Visited on 21 Apr 2017. [in Japanese]
8. Fong, W., Godsill, S.J., Doucet, A., West, M.: Monte Carlo smoothing with application to audio signal enhancement. *IEEE Trans. Signal Proces.* **50**(2), 438–449 (2002)
9. Ito, H.: A comparison of softwares for state space models. *Jpn. J. Ecol.* **66**(2), 361–374 (2016). [in Japanese]

10. Killick, R., Eckley, I.A.: changepoint: an R package for changepoint analysis. *J. Stat. Softw.* **58**(3), 1–19 (2014)
11. Kitagawa, G.: Introduction to Time Series Modeling. CRC Press, Boca Raton (2009)
12. Lindsten, F., Bunch, P., Särkkä, S., Schön, T.B., Godsill, S.J.: Rao-Blackwellized particle smoothers for conditionally linear Gaussian models. *IEEE J. Select. Top. Sig. Proces.* **10**(2), 353–365 (2016)
13. Matsura, K.: Bayesian Statistical Modeling using Stan and R. Kyoritsu Shuppan Co. Ltd., Tokyo (2016). [in Japanese]
14. Petris, G., Petrone, S., Campagnoli, P.: Dynamic Linear Model with R. Springer, Berlin (2009)
15. Prado, R., West, M.: Time Series: Modeling, Computation, and Inference. CRC Press, Boca Raton (2010)
16. Sanchez-Espigares, J.A., Lopez-Moreno, A.: MSwM: Fitting Markov Switching Models (2014). <https://CRAN.R-project.org/package=MSwM>. R package version 1.2
17. Särkkä, S.: Bayesian Filtering and Smoothing. Institute of Mathematical Statistics Textbooks. Cambridge University Press, Cambridge (2013)
18. Zeileis, A., Leisch, F., Hornik, K., Kleiber, C.: strucchange: An R package for testing for structural change in linear regression models. *J. Stat. Softw.* **7**(2), 1–38 (2002)

Appendix A

Library in R and External Software

A.1 dlm

dlm is a library in R [5].

This library can perform Bayesian estimation of dynamic linear model including the Kalman filter. The implementation of the Kalman filter is based on the algorithm [6] that uses singular value decomposition of the covariance matrix; therefore, the deterioration of numerical accuracy can be suppressed to the extent possible. Furthermore, the algorithm is executable even when the covariance of the state noise is singular. The library core is described in C language considering the execution performance. The well-known library **KFAS**, as mentioned in Chap.10, can also execute the Kalman filter in R; see [1–4] for a detailed explanation.

We can install the **dlm** from CRAN using function `install.packages()` similarly to other regular libraries.

A.2 Stan

Stan is a probabilistic programming language suitable for statistical modeling. It can also execute the MCMC.

Stan enables collaboration with multiple languages. The R library **rstan** is used for collaboration with R. The MCMC implementation is based on the hybrid Monte Carlo method (Hamiltonian Monte Carlo method) and generally has better sampling efficiency than other similar software. In addition, the source code is compiled using C++ language at runtime considering the execution performance.

Regarding the installation of **Stan**, follow the directions provided on <https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started>. The installation requires presetting the C++ compiler, such as **Rtools** on Windows OS. The support site for **Stan**, <http://mc-stan.org/>, contains various types of information.

A.3 pomp

pomp is a library in R.

This library can perform Bayesian estimation for the general state-space model including the particle filter. The typical algorithms in the particle filter are almost implemented. Furthermore, among the auxiliary particle filter, Rao–Blackwellization, and Liu and West filter, the functions `bsmc()` and `bsmc2()` implement the Liu and West filter. The library core is described in C language, and the source code can be compiled using C at runtime taking the execution performance into account.

We can install the **pomp** from CRAN with the function `install.packages()` similar to other regular libraries. For the compilation using C at runtime, we must preset a C compiler, such as **Rtools** on Windows OS. The support site for the **pomp**, <http://kingaa.github.io/pomp/>, contains various types of information.

A.4 NIMBLE

NIMBLE is a library in R. The library name is **nimble** in lower case.

This library is general software for Bayesian estimation similar to BUGS including the MCMC and the particle filter. The typical algorithms in the particle filter are almost implemented. Furthermore, among the auxiliary particle filter, Rao–Blackwellization, and Liu and West filter, the functions `buildAuxiliaryFilter()` and `buildLiuWestFilter()` implement the auxiliary particle filter and Liu and West filter, respectively. The source code is compiled using C++ language at runtime taking the execution performance into account.

We can install **NIMBLE** from CRAN using the function `install.packages()` similarly to other regular libraries. The installation requires presetting of the C++ compiler, such as **Rtools** on Windows OS. The support site for **NIMBLE**, <http://r-nimble.org/>, contains various types of information.

References

1. Baba, S.: Fundamentals of Time Series Analysis and State Space Models—Theory and Applications with R and Stan—. Pleiades Publishing Co., Ltd. (2018). [in Japanese]
2. Helske, J.: KFAS: Exponential family state space models in R. *Journal of Statistical Software* **78**(10), 1–39 (2017)
3. Iwanami Data Science Publication Committee (ed.): Iwanami Data Science, vol. 6. Iwanami Shoten, Publishers (2017). [in Japanese]
4. Nomura, S.: Kalman Filter—Time Series Prediction and State Space Model using R—. Kyoritsu Shuppan Co., Ltd. (2016). [in Japanese]
5. Petris, G., Petrone, S., Campagnoli, P.: Dynamic Linear Model with R. Springer (2009)
6. Wang, L., Libert, G., Manneback, P.: Kalman filter algorithm based on singular value decomposition. In: Proc. 31st IEEE Conference on Decision and Control (CDC), pp. 1224–1229. Tucson, U.S.A. (1992)

Appendix B

Library **dlm**

B.1 Handling of Model

The model in the library **dlm** is treated as a list that may have a class attribute `dlm`. The elements of the list are `m0`, `C0`, `FF`, `V`, `GG`, `W`, `JFF`, `JV`, `JGG`, and `JW`. The first half elements `m0`, `C0`, `FF`, `V`, `GG`, and `W` correspond to the parameters \mathbf{m}_0 , \mathbf{C}_0 , \mathbf{F}_t , \mathbf{V}_t , \mathbf{G}_t , and \mathbf{W}_t of the linear Gaussian state-space model in Eqs. (8.1) and (8.2), respectively. The second half elements with names starting with `J` are used to specify the time-varying model; for the time-invariant model, they do not exist or are set to `NULL`.

B.2 Setting of Time-Varying Model

We explain the setting of the time-varying model in the library **dlm** using the `$JFF` as an example.

First, the `$JFF` is a matrix whose dimension is same as that of `$FF`, and its elements can be 0 or a positive integer k . The values 0 and k indicate the part without and with temporal change, respectively, such as

$$\text{Value of } \mathbf{F}_t \text{ at time point } t = \begin{cases} \$FF[i, j] & \text{for } \$JFF[i, j] = 0, \\ \$X[t, k] & \text{for } \$JFF[i, j] = k. \end{cases}$$

Therefore, the values actually used for the part with temporal change are stored in the k -th column of the `$X`. Figure B.1 conceptually shows this relation.

While the above explanation regards the example of `$JFF`, the same concept is also applied to `$V`, `$GG`, and `$W` for `$JV`, `$JGG`, and `$JW`, respectively.

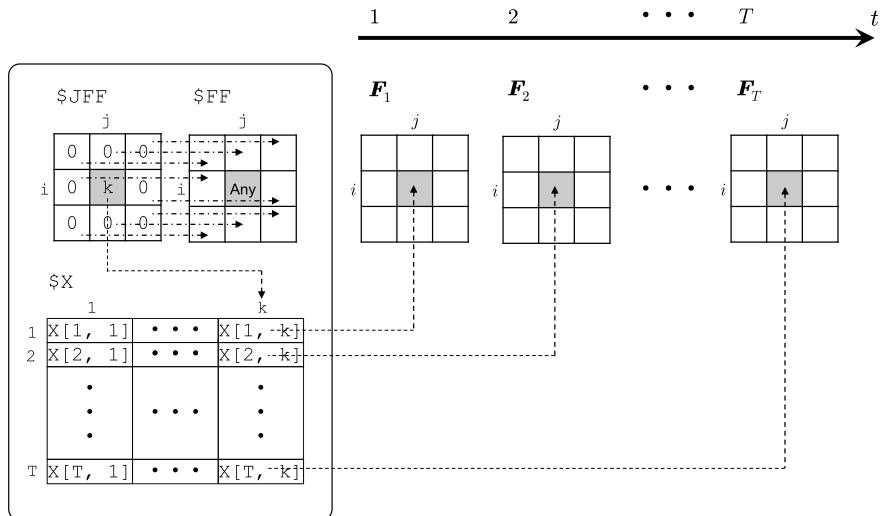


Fig. B.1 Setting the time-varying model in the library **dlm**

B.3 Square Root Algorithm

Library **dlm** implements a Kalman filter algorithm based on singular value decomposition for the covariance matrix to prevent the deterioration of numerical accuracy to the extent possible [2]. This is a kind of *square root Kalman filter* whose covariance matrix is updated in the form of the square root matrix. We can generally decompose a covariance matrix A as $A = \mathbf{U} \mathbf{D}^2 \mathbf{U}^\top$ using *singular value decomposition* [1]. The return values \mathbf{U} , \mathbf{S} , $\mathbf{U.C}$, and $\mathbf{U.R}$ from the various functions of the library **dlm** indicate “ \mathbf{U} ” in the singular value decomposition of the matrices \mathbf{S} , \mathbf{C} , and \mathbf{R} , respectively. Similarly, the return values $\mathbf{D.S}$, $\mathbf{D.C}$, and $\mathbf{D.R}$ indicate “ \mathbf{D} ” in the singular value decomposition of the matrices \mathbf{S} , \mathbf{C} , and \mathbf{R} , respectively.

B.4 Functions Primarily Used in This Book

This section provides an overview of the functions that are primarily used in this book. Detailed descriptions of the functions can be found in the library manual <https://cran.r-project.org/web/packages/dlm/dlm.pdf>.

B.4.1 *dlmFilter()*

This function performs Kalman filtering. Its processing is based on Algorithm 8.1, which specializes Eq. (6.18) for the linear Gaussian state-space model.

The arguments are y and mod . The argument y is the observation. The argument mod is a list representing the model.

The return value is a list with a class attribute `dlmFiltered`. The elements of the list are y , mod , m , $U.C$, $D.C$, a , $U.R$, $D.R$, and f . The elements y and mod are the observation and list representing a model, respectively. The element m is the mean of the filtering distribution. The elements $U.C$ and $D.C$ are the singular value decomposition for the covariance matrix of the filtering distribution. The element a is the mean of the one-step-ahead predictive distribution. The elements $U.R$ and $D.R$ are the singular value decomposition for the covariance matrix of the one-step-ahead predictive distribution. The element f is the mean of the one-step-ahead predictive likelihood. Among the above elements, the forefront of m , $U.C$, and $D.C$ is added with an object corresponding to the prior distribution; therefore, their length is one more than that for the observations.

B.4.2 *dlmForecast()*

This function performs Kalman prediction. Its processing is based on Algorithm 8.2 that specializes Eq. (6.22) for the linear Gaussian state-space model.

The arguments are mod , $nAhead$, and `sampleNew`. The argument mod is a list representing the model or an object of the `dlmFiltered` class. In case of predicting the future from the end time point of the observations, it is set to an object of the `dlmFiltered` class obtained through the Kalman filtering. The argument $nAhead$ is the maximum number k for the future time points in the prediction. While the argument `sampleNew` is set to the `FALSE` by default, if we set an integer, the samples for the number of trials are included in the return value.

The return value is a list. The elements of the list are a , R , f , Q , `newStates`, and `newObs`. The first four elements correspond to the mean and covariance matrix of the k -steps-ahead predictive distribution and the mean and covariance matrix of the k -steps-ahead predictive likelihood, respectively. The last two elements `newStates` and `newObs` contain samples for the future states and observations, respectively. They exist only when `sampleNew = an integer`.

B.4.3 *dlmSmooth()*

This function performs Kalman smoothing. Its processing is based on Algorithm 8.3, which specializes Eq. (6.24) for the linear Gaussian state-space model.

The arguments are y and mod . The argument y is the observation. The argument mod is a list representing the model. If we complete the Kalman filtering in advance, we can obtain the smoothing result by merely setting the object of the `dlmFiltered` class in the y .

The return value is a list. The elements of the list are s , U , S , and D . S . The element s is the mean of the smoothing distribution. The elements U , S and D , S are the singular value decomposition for the covariance matrix of the smoothing distribution. The foreparts of all the above elements are added with an object corresponding to the prior distribution; therefore, their length is one more than that for the observations.

B.4.4 `dlmBSample()`

This function performs the FFBS algorithm. Its processing is based on Algorithm 10.4.

The argument is the `modFilt`. We typically apply an object of the `dlmFiltered` class obtained through Kalman filtering.

The return value is the samples of the state (for one trial).

B.4.5 `dlmSvd2var()`

This function restores the original covariance matrix from its singular value decomposition.

The arguments are u and d . When we assume that the covariance matrix is decomposed as $A = UD^2U^\top$, the u and d are a list of the matrix U and a matrix storing the diagonal elements of the diagonal matrix D in the row direction, respectively.

The return value is a list of the restored covariance matrices.

B.4.6 `dlmLL()`

This function calculates the marginal log-likelihood of the model. Its processing is based on Eq. (8.6), which specializes Eq. (6.26) for the linear Gaussian state-space model.

The arguments are y and mod . The argument y is the observation. The argument mod is a list representing the model.

The return value is the “negative” marginal log-likelihood value.

B.4.7 ***dlmMLE()***

This function performs the maximum likelihood estimation for the parameters of the model. Its processing is based on Eq. (6.27).

The arguments are `y`, `parm`, and `build`. The argument `y` is the observation. The argument `parm` is the initial searching value for the parameter in the maximum likelihood estimation, and we can set any finite value. The argument `build` is a function that defines a model and returns a list representing the model. Note that the numerical algorithm for executing the maximum likelihood estimation uses the L-BFGS method by default, which is a version of the quasi-Newton method.

The return value is a list same as that returned from the R function `optim()`. In particular, the element `par` is the resulting maximum likelihood estimates, and the element `convergence` is a flag indicating the convergence status of the estimation: a value of 0 indicates that the convergence has been achieved.

B.4.8 ***dlmModPoly()***

This function sets up a polynomial model. Its setting is based on Eqs. (9.12) and (9.13).

The arguments are `order`, `dW`, `dV`, `m0`, and `C0`. The argument `order` is the order of the polynomial model. The arguments `dW` and `dV` are the variances W and V of the state and observation noises, respectively. The arguments `m0` and `C0` are the mean vector \mathbf{m}_0 and covariance \mathbf{C}_0 of the prior distribution, respectively.

The return value is a list representing the model with a class attribute `dlm`.

B.4.9 ***dlmModSeas()***

This function sets up the seasonal model (time-domain approach). Its setting is based on Eqs. (9.25) and (9.26).

The arguments are `frequency`, `dW`, `dV`, `m0`, and `C0`. The argument `frequency` is the cycle s . The arguments `dW` and `dV` are the diagonal elements $W, 0, \dots, 0$ in the covariance matrix of the state noise, and the variance V of the observation noise, respectively. The arguments `m0` and `C0` are the mean vector \mathbf{m}_0 and the covariance \mathbf{C}_0 of the prior distribution, respectively.

The return value is a list representing the model with a class attribute `dlm`.

B.4.10 `dlmModTrig()`

This function sets up the seasonal model (frequency-domain approach). Its setting is based on Eqs. (9.31) and (9.32) or Eqs. (9.33) and (9.34).

The arguments are s , q , $\text{d}W$, $\text{m}0$, $C0$, om , and $\tau\omega$. The argument s is an integer fundamental period. The argument q is the number of frequency components left in the order from the lowest frequency. The arguments $\text{d}W$ and $\text{d}V$ are the diagonal elements in the covariance of the state noise and the variance V of the observation noise, respectively. While this model can set different variances of the state noise, such as $W^{(1)}, \dots, W^{(N)}$ for each frequency component, if there is no particular prior information, we set the unique value W to them. The arguments $\text{m}0$ and $C0$ are the mean vector \mathbf{m}_0 and covariance \mathbf{C}_0 of the prior distribution, respectively. The arguments $\tau\omega$ and om are non-integer fundamental period and fundamental angular frequency, respectively; either of them is used only if the fundamental period is not an integer.

The return value is a list representing the model with a class attribute `dlm`.

B.4.11 `dlmModARMA()`

This function sets up the ARMA model. Its setting is based on Eqs. (9.36) and (9.37).

The arguments are `ar`, `ma`, `sigma2`, $\text{d}V$, $\text{m}0$, and $C0$. The argument `ar` is the AR coefficient; we set it to `NULL` if it is not used. The argument `ma` is the MA coefficient; we set it to `NULL` if it is not used. The argument `sigma2` is the variance of white noise. The argument $\text{d}V$ is the variance V of the observation noise. The arguments $\text{m}0$ and $C0$ are the mean vector \mathbf{m}_0 and covariance \mathbf{C}_0 of the prior distribution, respectively.

The return value is a list representing the model with a class attribute `dlm`.

B.4.12 `dlmModReg()`

This function sets up the regression model. Its setting is based on Eqs. (9.39) and (9.40).

The arguments are X , `addInt`, $\text{d}W$, $\text{d}V$, $\text{m}0$, and $C0$. The argument X is a vector or matrix of the explanatory variables. The argument `addInt` is an indicator of whether or not to consider the regression intercept, and the setting `TRUE` specifies that the regression intercept is included. The arguments $\text{d}W$ and $\text{d}V$ are the diagonal elements $W^{(\alpha)}, W^{(1)}, \dots, W^{(N)}$ in the covariance matrix of the state noise and variance V of the observation noise, respectively. The arguments $\text{m}0$ and $C0$ are the mean vector \mathbf{m}_0 and covariance \mathbf{C}_0 of the prior distribution, respectively.

The return value is a list representing the model with a class attribute `dlm`.

B.4.13 ARtransPars()

This function coerces the AR coefficients to their approximate values in the stationary region.

The argument is `xaw`, which is a vector of the original AR coefficients.

The return value is a vector whose length is the same as that of the argument and contains the AR coefficients approximated to the stationary region.

B.4.14 weighted.quantile()

This function calculates the weighted quantiles.

This is not a built-in function of the library but is a user-defined utility function originally provided on the library support site <http://definetti.uark.edu/> (dead link as of 2020-11-12). This book often applies this function to the results of the particle filter; we introduce the contents here. This code is as follows.

```

1 > # quantile function for weighted particle clouds
2 > weighted.quantile <- function(x, w, probs)
3 + {
4 +   ## Make sure 'w' is a probability vector
5 +   if ((s <- sum(w)) != 1)
6 +     w <- w / s
7 +   ## Sort 'x' values
8 +   ord <- order(x)
9 +   x <- x[ord]
10 +  w <- w[ord]
11 +  ## Evaluate cdf
12 +  W <- cumsum(w)
13 +  ## Invert cdf
14 +  tmp <- outer(probs, W, "<=")
15 +  n <- length(x)
16 +  quantInd <- apply(tmp, 1, function(x) (1 : n)[x][1])
17 +  ## Return
18 +  ret <- x[quantInd]
19 +  ret[is.na(ret)] <- x[n]
20 +  return(ret)
21 + }
```

The arguments are `x`, `w`, and `probs`, which are a vector of the realizations, a vector of the weights, and the quantiles of interest, respectively. This code first performs the sort according to the realization value and then derives the realization value at the required quantile point based on the accumulated weight.

The return value is the realization value at the quantile point of interest.

References

1. Petris, G., Petrone, S., Campagnoli, P.: Dynamic Linear Model with R. Springer (2009)
2. Wang, L., Libert, G., Manneback, P.: Kalman filter algorithm based on singular value decomposition. In: Proc. 31st IEEE Conference on Decision and Control (CDC), pp. 1224–1229. Tucson, U.S.A. (1992)

Appendix C

Supplement on Conditional Independence in the State-Space Model

C.1 Derivation of Eq.(5.3)

Equation (5.3) is derived as follows:

$$p(\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:T}) = p(\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t}, y_{t+1:T}) \quad (\text{C.1})$$

change the order of the last variables

$$= p(\mathbf{x}_t \mid y_{t+1:T}, \mathbf{x}_{t+1}, y_{1:t}) \quad (\text{C.2})$$

from the relation $p(a, b \mid c) = p(a \mid b, c)p(b \mid c)$ with considering \mathbf{x}_{t+1} and $y_{1:t}$ as a single block

$$= \frac{p(\mathbf{x}_t, y_{t+1:T} \mid \mathbf{x}_{t+1}, y_{1:t})}{p(y_{t+1:T} \mid \mathbf{x}_{t+1}, y_{1:t})} \quad (\text{C.3})$$

change the order of the first two variables in the numerator

$$= \frac{p(y_{t+1:T}, \mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t})}{p(y_{t+1:T} \mid \mathbf{x}_{t+1}, y_{1:t})} \quad (\text{C.4})$$

the numerator can be expanded from the relation $p(a, b \mid c) = p(a \mid b, c)p(b \mid c)$

$$= \frac{p(y_{t+1:T} \mid \mathbf{x}_t, \mathbf{x}_{t+1}, y_{1:t})p(\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t})}{p(y_{t+1:T} \mid \mathbf{x}_{t+1}, y_{1:t})} \quad (\text{C.5})$$

the first item in the numerator can be simplified based on the conditional independence suggested by Eq.(5.2)

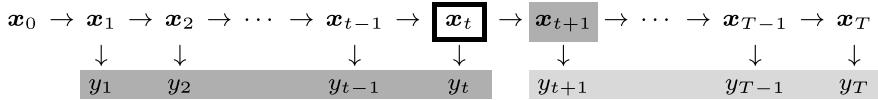


Fig. C.1 Conditional independence on state (when $\boxed{\text{■}}$ is given, $\boxed{\text{□}}$ is independent of $\boxed{\text{■}}$)

$$= \frac{p(y_{t+1:T} | \mathbf{x}_{t+1}, y_{1:t}) p(\mathbf{x}_t | \mathbf{x}_{t+1}, y_{1:t})}{p(y_{t+1:T} | \mathbf{x}_{t+1}, y_{1:t})} \quad (\text{C.6})$$

$$= p(\mathbf{x}_t | \mathbf{x}_{t+1}, y_{1:t}). \quad (\text{C.7})$$

Figure C.1 shows the DAG representation of the above relationship.

Incidentally, the following relation also holds through almost the same discussion:

$$p(\mathbf{x}_t | \mathbf{x}_{t+1:T}, y_{1:T}) = p(\mathbf{x}_t | \mathbf{x}_{t+1}, y_{1:t}). \quad (\text{C.8})$$

Appendix D

Symbol Assignment in the Linear Gaussian State-Space Model

As mentioned in Sect. 5.4, there is no rule regarding the assignment of the symbols in the linear Gaussian state-space model, and it varies depending on the literature. While this book is principally in accordance with [9–10] because of the consistency with the functions in the libraries **dlm** and **Stan**, we summarize the correspondences with the other significant literature in Table D.1 for information. Any decoration for symbols, such as bold font and subscripts, are ignored.

References

1. Anderson, B.D.O., Moore, J.B.: Optimal Filtering. Information and system sciences (Thomas Kailath editor). Prentice-Hall (1979)
2. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer (2006)
3. Commandeur, J.J., Koopman, S.J.: An Introduction to State Space Time Series Analysis. Oxford University Press (2007)
4. Durbin, J., Koopman, S.J.: Time Series Analysis by State Space Methods, 2nd edn. Oxford University Press (2012)
5. Harvey, A.C.: Forecasting, Structural Time Series Models and the Kalman Filter. Cambridge University Press (1989)
6. Kailath, T., Sayed, A.H., Hassibi, B.: Linear Estimation. Prentice Hall (2000)
7. Kitagawa, G.: Introduction to Time Series Modeling. CRC Press (2009)
8. Kitagawa, G., Gersch, W.: Smoothness Priors Analysis of Time Series. Springer (1996)
9. Petris, G., Petrone, S., Campagnoli, P.: Dynamic Linear Model with R. Springer (2009)
10. Prado, R., West, M.: Time Series: Modeling, Computation, and Inference. CRC Press (2010)
11. West, M., Harrison, J.: Bayesian Forecasting and Dynamic Models, 2nd edn. Springer-Verlag (1997)

Table D.1 Assignment of symbols in the linear Gaussian state-space model

	This book	[9]	[11]	[10]	[5]	[3]	[4]	[8]	[7]	[1]	[6]	[2]
State	x	θ	θ	θ	α	α	α	x	x	x	x	z
Observations	y	Y	Y	y	y	y	y	y	y	z	y	x
State transition matrix	G	G	G	G^T	T	T	T	F	F	F	F	A
Observation matrix	F	F	F	F^T	Z	Z^T	Z	H	H	H^T	H	C
State noise	w	w	ω	ω	$R\eta$	$R\eta$	$R\eta$	Gw	Gv	Gw	Gu	w
Observation noise	v	v	ν	ν	ϵ	ϵ	ϵ	w	v	v	v	v
Variance of state noise	W	W	W	W	RQR^T	RQR^T	RQR^T	GQG^T	GQG^T	GQG^T	GQG^T	R
Variance of observation noise	V	V	V	V	H	H	H	R	R	R	R	Σ
Dimension of state	p	p	n	p	m	m	m	m	k	n	n	K
Dimension of observations	m	m	r	r	N	p	p	l	l	p	p	R

Appendix E

Algorithm Derivation

E.1 Wiener Filter

E.1.1 Auxiliary Information for Derivation

E.1.1.1 Frequency Domain Representation

We first describe z -transform, a kind of general frequency domain transform for discrete data. The (two-sided) z -transform for discrete time series x_t is defined as

$$\mathcal{Z}[x_t] = \sum_{t=-\infty}^{\infty} x_t z^{-t}. \quad (\text{E.1})$$

Given the complex value $z = e^{i\omega}$, z -transform is equivalent to the discrete Fourier transform, where ω denotes the angular frequency.

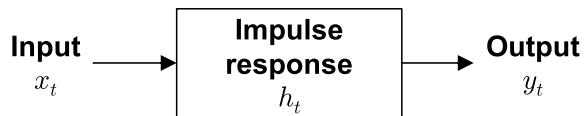
From the definition of z -transform, the shift (delay) in the time domain is represented by the product of the power of z in the frequency domain as

$$\mathcal{Z}[x_{t-k}] = z^{-k} \mathcal{Z}[x_t]. \quad (\text{E.2})$$

Regarding the power spectrum mentioned in the note of Sect. 4.2.4, it has been proved that the z -transform of correlation function $R(k)$ for the stationary process is equal to the *power spectrum* $S(z)$ as follows:

$$S(z) = \mathcal{Z}[R(k)]. \quad (\text{E.3})$$

Fig. E.1 Linear time-invariant system



The above relation is called the *Wiener–Khinchin theorem*. When specifying the target time series for the power spectrum, we add subscripts to the notation, such as $S_{xy}(z)$, as with the correlation function.

E.1.1.2 Linear Time-Invariant System

Suppose a system with input and output as shown in Fig. E.1.

In the above figure, the input time series is x_t , impulse response completely representing the property of the system is h_t , and output time series is y_t . If a system has a linear relation between its input and output and time shift applied to input is also held in the output, we call the system a *linear time-invariant system*. Note that Chap. 7 examines a linear time-invariant system.

From the above definition, the output of the linear time-invariant system is represented by the *convolution* of the input and impulse response [1, 2, 5] as follows:

$$y_t = \sum_{k=-\infty}^{\infty} x_k h_{t-k} = x_t \circledast h_t, \quad (\text{E.4})$$

where the symbol \circledast denotes the convolution operation. From the definition of the z -transform, the convolution in the time domain corresponds to a product in the frequency domain as follows:

$$\mathcal{Z}[y_t] = \mathcal{Z}[x_t \circledast h_t] = \mathcal{Z}[x_t] \mathcal{Z}[h_t] = H(z) \mathcal{Z}[x_t], \quad (\text{E.5})$$

where the function $\mathcal{Z}[h_t] = H(z)$ is referred to as the *transfer function*.

In addition, the following relation holds for the autocorrelation functions of the input and output in linear time-invariant systems [3, Proposition 2.4][4, Appendix B.14]:

$$\mathcal{Z}[R_{yy}(k)] = |H(z)|^2 \mathcal{Z}[R_{xx}(k)] \quad (\text{E.6})$$

$$S_{yy}(z) = |H(z)|^2 S_{xx}(z) \quad (\text{E.7})$$

$$= H(z) \overline{H(z)} S_{xx}(z) \quad (\text{E.8})$$

from the definition of the z -transform (when the impulse response h_t is real-valued)

$$= H(z) H(z^{-1}) S_{xx}(z). \quad (\text{E.9})$$

E.1.2 Wiener Smoothing

The transfer function $H(z) = \mathcal{Z}[h_t]$ of the Wiener filter (in case of smoothing) satisfies the following *Wiener–Hopf equation* [3, Theorem 4.2] [4, Chap. 3.1.1]:

$$S_{xy}(z) = H(z)S_{yy}(z). \quad (\text{E.10})$$

From the above equation, $H(z)$ is derived as follows:

$$H(z) = \frac{S_{xy}(z)}{S_{yy}(z)} \quad (\text{E.11})$$

from the assumption that x_t and v_t are independent of each other

$$= \frac{S_{xx}(z)}{S_{xx}(z) + S_{vv}(z)}. \quad (\text{E.12})$$

E.1.3 Derivation of Eq. (7.6)

First, to derive $S_{xx}(z)$, we transform Eq. (7.4) as follows:

$$w_t = x_t - \phi x_{t-1} \quad (\text{E.13})$$

apply the z -transform to both sides

$$\mathcal{Z}[w_t] = \mathcal{Z}[x_t] - \phi \mathcal{Z}[x_{t-1}] \quad (\text{E.14})$$

take the delay element out of the last term

$$= \mathcal{Z}[x_t] - \phi \mathcal{Z}[x_t] z^{-1} \quad (\text{E.15})$$

$$= (1 - \phi z^{-1}) \mathcal{Z}[x_t]. \quad (\text{E.16})$$

Through the above equation, we can regard the AR(1) model in Eq. (7.4) as a linear time-invariant system, wherein the input is $\mathcal{Z}[x_t]$, transfer function is $(1 - \phi z^{-1})$, and output is $\mathcal{Z}[w_t]$.

Such a filter that converts the input into white noise without losing information is called a *whitening filter*.

Based on this viewpoint and Eq. (E.9), we obtain the following equation:

$$S_{ww}(z) = (1 - \phi z^{-1})(1 - \phi z)S_{xx}(z). \quad (\text{E.17})$$

From the above equation, we derive $S_{xx}(z)$ as

$$S_{xx}(z) = \frac{1}{(1 - \phi z^{-1})(1 - \phi z)} S_{ww}(z) \quad (\text{E.18})$$

recall the w_t is white noise

$$= \frac{1}{(1 - \phi z^{-1})(1 - \phi z)} W. \quad (\text{E.19})$$

Next, $S_{vv}(z)$ becomes V from the definition of white noise.

Based on the above and Eq.(7.3), the transfer function of the Wiener filter is as follows:

$$H(z) = \frac{S_{xx}(z)}{S_{xx}(z) + S_{vv}(z)} \quad (\text{E.20})$$

$$= \frac{\frac{W}{(1 - \phi z^{-1})(1 - \phi z)}}{\frac{W}{(1 - \phi z^{-1})(1 - \phi z)} + V} \quad (\text{E.21})$$

$$= \frac{W}{W + V(1 - \phi z^{-1})(1 - \phi z)} \quad (\text{E.22})$$

define $r = V/W$

$$= \frac{1}{1 + r(1 - \phi z^{-1})(1 - \phi z)} \quad (\text{E.23})$$

$$= \frac{1}{1 + r(-\phi z + (1 + \phi^2) - \phi z^{-1})} \quad (\text{E.24})$$

$$= \frac{1}{1 - r\phi z^{-1} \left(z^2 - \frac{1+\phi^2}{\phi} z + 1 \right)} \quad (\text{E.25})$$

$$= \frac{1}{-r\phi z^{-1} \left(z^2 - \frac{r^{-1}+1+\phi^2}{\phi} z + 1 \right)} \quad (\text{E.26})$$

$$= \frac{1}{-r\phi z^{-1} \left(z^2 - \left(\frac{1}{r\phi} + \phi^{-1} + \phi \right) z + 1 \right)}. \quad (\text{E.27})$$

Herein, we assume the following relation:

$$\frac{1}{r\phi} + \phi^{-1} + \phi = \beta^{-1} + \beta. \quad (\text{E.28})$$

This assumption leads to the quadratic equation for β as $\beta^2 - \left(\frac{1}{r\phi} + \phi^{-1} + \phi\right)\beta + 1 = 0$, with the consequence that the following relations hold:

$$\frac{1}{r\phi}\beta = \beta^2 - (\phi^{-1} + \phi)\beta + 1 = (\beta - \phi^{-1})(\beta - \phi) = (\phi^{-1} - \beta)(\phi - \beta), \quad (\text{E.29})$$

$$\beta = \frac{\left(\frac{1}{r\phi} + \phi^{-1} + \phi\right) - \sqrt{\left(\frac{1}{r\phi} + \phi^{-1} + \phi\right)^2 - 4}}{2}. \quad (\text{E.30})$$

From Eqs. (E.28) and (E.29), Eq. (E.27) is further transformed as follows:

$$H(z) = \frac{(\phi^{-1} - \beta)(\phi - \beta)}{-\beta z^{-1}(z^2 - (\beta^{-1} + \beta)z + 1)} \quad (\text{E.31})$$

$$= \frac{(\phi^{-1} - \beta)(\phi - \beta)}{-\beta z^{-1}(z - \beta)(z - \beta^{-1})} \quad (\text{E.32})$$

$$= \frac{(\phi^{-1} - \beta)(\phi - \beta)}{-\beta(1 - \beta z^{-1})\beta^{-1}(\beta z - 1)} \quad (\text{E.33})$$

$$= (\phi^{-1} - \beta)(\phi - \beta) \frac{1}{(1 - \beta z^{-1})(1 - \beta z)} \quad (\text{E.34})$$

$$= (\phi^{-1} - \beta)(\phi - \beta) \frac{1}{1 - \beta^2} \left(\frac{1}{1 - \beta z^{-1}} + \frac{\beta z}{1 - \beta z} \right) \quad (\text{E.35})$$

if the range of $|\beta| < |z| < |\beta^{-1}|$ is satisfied, we can apply the Taylor expansion inside the last parentheses

$$= \frac{(\phi^{-1} - \beta)(\phi - \beta)}{1 - \beta^2} \left(\sum_{k=0}^{\infty} \beta^k z^{-k} + \sum_{k=0}^{\infty} \beta^{k+1} z^{k+1} \right) \quad (\text{E.36})$$

$$= \frac{(\phi^{-1} - \beta)(\phi - \beta)}{1 - \beta^2} \sum_{k=-\infty}^{\infty} \beta^{|k|} z^k. \quad (\text{E.37})$$

Recall that the multiplication of z^k in the frequency domain corresponds to a k -steps-ahead shift in the time domain, yielding h_t as follows:

$$h_t = \frac{(\phi^{-1} - \beta)(\phi - \beta)}{1 - \beta^2} \sum_{k=-\infty}^{\infty} \beta^{|k|} \delta_{t+k}, \quad (\text{E.38})$$

where δ_t is the Dirac delta function, i.e., a unit impulse function.

Based on Eq. (7.2), we finally obtain the desired signal d_t as follows:

$$d_t = h_t \circledast y_t \quad (\text{E.39})$$

$$= \frac{(\phi^{-1} - \beta)(\phi - \beta)}{1 - \beta^2} \sum_{k=-\infty}^{\infty} \beta^{|k|} y_{t+k}. \quad (\text{E.40})$$

E.2 Kalman Filter

E.2.1 Auxiliary Information for Derivation

The below description uses *law of iterated expectations* and *law of total variance*:

$$\text{law of iterated expectations: } \mathbb{E}[X | Y] = \mathbb{E}[\mathbb{E}[X | Y, Z] | Y], \quad (\text{E.41})$$

$$\begin{aligned} \text{law of total variance: } \text{Var}[X | Y] &= \mathbb{E}[\text{Var}[X | Y, Z] | Y] \\ &\quad + \text{Var}[\mathbb{E}[X | Y, Z] | Y]. \end{aligned} \quad (\text{E.42})$$

E.2.1.1 Matrix Inversion Lemmas

Although the *matrix inversion lemmas* have several forms, this book uses the following forms:

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(DA^{-1}B + C^{-1})^{-1}DA^{-1}, \quad (\text{E.43})$$

$$(A^{-1} + B^\top C^{-1}B)^{-1}B^\top C^{-1} = AB^\top(BAB^\top + C)^{-1}. \quad (\text{E.44})$$

E.2.1.2 Bayesian Estimation for Linear Gaussian Regression Model

Assume the following linear Gaussian regression model:

$$y = X\beta + \epsilon, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, V), \quad (\text{E.45})$$

where we assume that X and V are known.

Assuming that the β has a normal distribution with mean vector m_0 and covariance matrix C_0 , we have the Bayesian estimation for β given y as follows:

$$p(\beta | y) \propto \text{likelihood} \times \text{prior distribution} \quad (\text{E.46})$$

$$= p(y | \beta)p(\beta) \quad (\text{E.47})$$

$$= \mathcal{N}(X\beta, V)\mathcal{N}(\mathbf{m}_0, C_0). \quad (\text{E.48})$$

Furthermore, assuming $p(\beta | \mathbf{y}) = \mathcal{N}(\mathbf{m}, \mathbf{C})$, we can express \mathbf{C} and \mathbf{m} as follows similarly to Eqs. (6.8) and (6.9):

$$\mathbf{C}^{-1} = \mathbf{C}_0^{-1} + \mathbf{X}^\top \mathbf{V}^{-1} \mathbf{X}, \quad (\text{E.49})$$

$$\mathbf{m} = \mathbf{C} \mathbf{X}^\top \mathbf{V}^{-1} \mathbf{y} + \mathbf{C} \mathbf{C}_0^{-1} \mathbf{m}_0. \quad (\text{E.50})$$

The above equations can be further transformed. First, Eq. (E.49) can be expressed as

$$\mathbf{C} = (\mathbf{C}_0^{-1} + \mathbf{X}^\top \mathbf{V}^{-1} \mathbf{X})^{-1} \quad (\text{E.51})$$

from Eq. (E.43)

$$= \mathbf{C}_0 - \mathbf{C}_0 \mathbf{X}^\top (\mathbf{X} \mathbf{C}_0 \mathbf{X}^\top + \mathbf{V})^{-1} \mathbf{X} \mathbf{C}_0. \quad (\text{E.52})$$

In addition, Eq. (E.50) can be expressed as

$$\mathbf{m} = \mathbf{C} \mathbf{X}^\top \mathbf{V}^{-1} \mathbf{y} + \mathbf{C} \mathbf{C}_0^{-1} \mathbf{m}_0 \quad (\text{E.53})$$

apply Eq. (E.51) to the first term

$$= (\mathbf{C}_0^{-1} + \mathbf{X}^\top \mathbf{V}^{-1} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{V}^{-1} \mathbf{y} + \mathbf{C} \mathbf{C}_0^{-1} \mathbf{m}_0 \quad (\text{E.54})$$

apply Eq. (E.44) to the first term

$$= \mathbf{C}_0 \mathbf{X}^\top (\mathbf{X} \mathbf{C}_0 \mathbf{X}^\top + \mathbf{V})^{-1} \mathbf{y} + \mathbf{C} \mathbf{C}_0^{-1} \mathbf{m}_0 \quad (\text{E.55})$$

apply Eq. (E.52) to the second term

$$\begin{aligned} &= \mathbf{C}_0 \mathbf{X}^\top (\mathbf{X} \mathbf{C}_0 \mathbf{X}^\top + \mathbf{V})^{-1} \mathbf{y} \\ &\quad + (\mathbf{C}_0 - \mathbf{C}_0 \mathbf{X}^\top (\mathbf{X} \mathbf{C}_0 \mathbf{X}^\top + \mathbf{V})^{-1} \mathbf{X} \mathbf{C}_0) \mathbf{C}_0^{-1} \mathbf{m}_0 \end{aligned} \quad (\text{E.56})$$

$$\begin{aligned} &= \mathbf{C}_0 \mathbf{X}^\top (\mathbf{X} \mathbf{C}_0 \mathbf{X}^\top + \mathbf{V})^{-1} \mathbf{y} \\ &\quad + \mathbf{m}_0 - \mathbf{C}_0 \mathbf{X}^\top (\mathbf{X} \mathbf{C}_0 \mathbf{X}^\top + \mathbf{V})^{-1} \mathbf{X} \mathbf{m}_0 \end{aligned} \quad (\text{E.57})$$

$$= \mathbf{m}_0 + \mathbf{C}_0 \mathbf{X}^\top (\mathbf{X} \mathbf{C}_0 \mathbf{X}^\top + \mathbf{V})^{-1} (\mathbf{y} - \mathbf{X} \mathbf{m}_0). \quad (\text{E.58})$$

E.2.2 Kalman Filtering

E.2.2.1 One-Step-Ahead Predictive Distribution

First, the mean is derived based on Eq. (8.4) as follows:

$$\mathbf{a}_t = \text{E}[\mathbf{x}_t \mid y_{1:t-1}] \quad (\text{E.59})$$

from Eq.(E.41)

$$= \text{E}[\text{E}[\mathbf{x}_t \mid \mathbf{x}_{t-1}, y_{1:t-1}] \mid y_{1:t-1}] \quad (\text{E.60})$$

from Eq.(5.1)

$$= \text{E}[\text{E}[\mathbf{x}_t \mid \mathbf{x}_{t-1}] \mid y_{1:t-1}] \quad (\text{E.61})$$

from Eq.(5.10)

$$= \mathbf{G}_t \mathbf{x}_{t-1} \mid y_{1:t-1} \quad (\text{E.62})$$

from Eq.(8.3)

$$= \mathbf{G}_t \mathbf{m}_{t-1}. \quad (\text{E.63})$$

In addition, the variance is derived based on Eq.(8.4) as follows:

$$\mathbf{R}_t = \text{Var}[\mathbf{x}_t \mid y_{1:t-1}] \quad (\text{E.64})$$

from Eq.(E.42)

$$= \text{E}[\text{Var}[\mathbf{x}_t \mid \mathbf{x}_{t-1}, y_{1:t-1}] \mid y_{1:t-1}] + \text{Var}[\text{E}[\mathbf{x}_t \mid \mathbf{x}_{t-1}, y_{1:t-1}] \mid y_{1:t-1}] \quad (\text{E.65})$$

from Eq.(5.1)

$$= \text{E}[\text{Var}[\mathbf{x}_t \mid \mathbf{x}_{t-1}] \mid y_{1:t-1}] + \text{Var}[\text{E}[\mathbf{x}_t \mid \mathbf{x}_{t-1}] \mid y_{1:t-1}] \quad (\text{E.66})$$

from Eq.(5.10)

$$= \text{E}[\mathbf{W}_t \mid y_{1:t-1}] + \text{Var}[\mathbf{G}_t \mathbf{x}_{t-1} \mid y_{1:t-1}] \quad (\text{E.67})$$

from Eq.(8.3)

$$= \mathbf{W}_t + \mathbf{G}_t \mathbf{C}_{t-1} \mathbf{G}_t^\top. \quad (\text{E.68})$$

E.2.2.2 One-Step-Ahead Predictive Likelihood

First, the mean is derived based on Eq.(8.5) as follows:

$$f_t = \text{E}[y_t \mid y_{1:t-1}] \quad (\text{E.69})$$

from Eq.(E.41)

$$= \text{E}[\text{E}[y_t | \mathbf{x}_t, y_{1:t-1}] | y_{1:t-1}] \quad (\text{E.70})$$

from Eq.(5.2)

$$= \text{E}[\text{E}[y_t | \mathbf{x}_t] | y_{1:t-1}] \quad (\text{E.71})$$

from Eq.(5.11)

$$= \text{E}[\mathbf{F}_t \mathbf{x}_t | y_{1:t-1}] \quad (\text{E.72})$$

from Eq.(8.4)

$$= \mathbf{F}_t \mathbf{a}_t. \quad (\text{E.73})$$

In addition, the variance is derived based on Eq.(8.5) as follows:

$$\mathcal{Q}_t = \text{Var}[y_t | y_{1:t-1}] \quad (\text{E.74})$$

from Eq.(E.42)

$$= \text{E}[\text{Var}[y_t | \mathbf{x}_t, y_{1:t-1}] | y_{1:t-1}] + \text{Var}[\text{E}[y_t | \mathbf{x}_t, y_{1:t-1}] | y_{1:t-1}] \quad (\text{E.75})$$

from Eq.(5.2)

$$= \text{E}[\text{Var}[y_t | \mathbf{x}_t] | y_{1:t-1}] + \text{Var}[\text{E}[y_t | \mathbf{x}_t] | y_{1:t-1}] \quad (\text{E.76})$$

from Eq.(5.11)

$$= \text{E}[V_t | y_{1:t-1}] + \text{Var}[\mathbf{F}_t \mathbf{x}_t | y_{1:t-1}] \quad (\text{E.77})$$

from Eq.(8.4)

$$= V_t + \mathbf{F}_t \mathbf{R}_t \mathbf{F}_t^\top. \quad (\text{E.78})$$

E.2.2.3 Filtering Distribution

First, according to Eq.(6.18), this distribution is expressed as follows:

$$p(\mathbf{x}_t | y_{1:t}) \propto p(y_t | \mathbf{x}_t) p(\mathbf{x}_t | y_{1:t-1}) \quad (\text{E.79})$$

from Eqs.(5.11) and (8.4)

$$= \mathcal{N}(\mathbf{F}_t \mathbf{x}_t, V_t) \mathcal{N}(\mathbf{a}_t, \mathbf{R}_t). \quad (\text{E.80})$$

When comparing the above equation with Eq. (E.48), we see that the mean and variance of the filtering distribution can be obtained by replacing \mathbf{X} , \mathbf{V} , \mathbf{m}_0 , \mathbf{C}_0 , and \mathbf{y} in Eqs. (E.58) and (E.52) with \mathbf{F}_t , V_t , \mathbf{a}_t , \mathbf{R}_t , and y_t , respectively.

Specifically, the mean is derived based on Eq. (8.3) as follows:

$$\mathbf{m}_t = \mathbb{E}[\mathbf{x}_t \mid y_{1:t}] \quad (\text{E.81})$$

from Eq. (E.58)

$$= \mathbf{a}_t + \mathbf{R}_t \mathbf{F}_t^\top (\mathbf{F}_t \mathbf{R}_t \mathbf{F}_t^\top + V_t)^{-1} (y_t - \mathbf{F}_t \mathbf{a}_t) \quad (\text{E.82})$$

from Eqs. (E.78) and (E.73)

$$= \mathbf{a}_t + \mathbf{R}_t \mathbf{F}_t^\top Q_t^{-1} [y_t - f_t]. \quad (\text{E.83})$$

In addition, the variance is derived based on Eq. (8.3) as follows:

$$\mathbf{C}_t = \text{Var}[\mathbf{x}_t \mid y_{1:t}] \quad (\text{E.84})$$

from Eq. (E.52)

$$= \mathbf{R}_t - \mathbf{R}_t \mathbf{F}_t^\top (\mathbf{F}_t \mathbf{R}_t \mathbf{F}_t^\top + V_t)^{-1} \mathbf{F}_t \mathbf{R}_t \quad (\text{E.85})$$

from Eq. (E.78)

$$= \mathbf{R}_t - \mathbf{R}_t \mathbf{F}_t^\top Q_t^{-1} \mathbf{F}_t \mathbf{R}_t \quad (\text{E.86})$$

$$= [\mathbf{I} - \mathbf{R}_t \mathbf{F}_t^\top Q_t^{-1} \mathbf{F}_t] \mathbf{R}_t. \quad (\text{E.87})$$

E.2.3 Kalman Prediction

First, the mean is derived based on Eq. (8.7) as follows:

$$\mathbf{a}_t(k) = \mathbb{E}[\mathbf{x}_{t+k} \mid y_{1:t}] \quad (\text{E.88})$$

from Eq. (E.41)

$$= \mathbb{E}[\mathbb{E}[\mathbf{x}_{t+k} \mid \mathbf{x}_{t+k-1}, y_{1:t}] \mid y_{1:t}] \quad (\text{E.89})$$

from the conditional independence suggested by Eq. (5.1)

$$= \text{E}[\text{E}[\mathbf{x}_{t+k} \mid \mathbf{x}_{t+k-1}] \mid y_{1:t}] \quad (\text{E.90})$$

from Eq.(5.10)

$$= \text{E}[\mathbf{G}_{t+k} \mathbf{x}_{t+k-1} \mid y_{1:t}] \quad (\text{E.91})$$

from Eq.(8.7)

$$= \mathbf{G}_{t+k} \mathbf{a}_t(k-1). \quad (\text{E.92})$$

In addition, the variance is derived based on Eq.(8.7) as follows:

$$\mathbf{R}_t(k) = \text{Var}[\mathbf{x}_{t+k} \mid y_{1:t}] \quad (\text{E.93})$$

from Eq.(E.42)

$$= \text{E}[\text{Var}[\mathbf{x}_{t+k} \mid \mathbf{x}_{t+k-1}, y_{1:t}] \mid y_{1:t}] + \text{Var}[\text{E}[\mathbf{x}_{t+k} \mid \mathbf{x}_{t+k-1}, y_{1:t}] \mid y_{1:t}] \quad (\text{E.94})$$

from the conditional independence suggested by Eq.(5.1)

$$= \text{E}[\text{Var}[\mathbf{x}_{t+k} \mid \mathbf{x}_{t+k-1}] \mid y_{1:t}] + \text{Var}[\text{E}[\mathbf{x}_{t+k} \mid \mathbf{x}_{t+k-1}] \mid y_{1:t}] \quad (\text{E.95})$$

from Eq.(5.10)

$$= \text{E}[\mathbf{W}_{t+k} \mid y_{1:t}] + \text{Var}[\mathbf{G}_{t+k} \mathbf{x}_{t+k-1} \mid y_{1:t}] \quad (\text{E.96})$$

from Eq.(8.7)

$$= \mathbf{W}_{t+k} + \mathbf{G}_{t+k} \mathbf{R}_t(k-1) \mathbf{G}_{t+k}^\top. \quad (\text{E.97})$$

E.2.4 Kalman Smoothing

First, the mean is derived based on Eq.(8.8) as follows:

$$\mathbf{s}_t = \text{E}[\mathbf{x}_t \mid y_{1:T}] \quad (\text{E.98})$$

from Eq.(E.41)

$$= \text{E}[\text{E}[\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:T}] \mid y_{1:T}] \quad (\text{E.99})$$

from Eq.(5.3)

$$= \mathbb{E}[\mathbb{E}[\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t}] \mid y_{1:T}]. \quad (\text{E.100})$$

Now, to obtain $\mathbb{E}[\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t}]$ in the above equation, transform the distribution $p(\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t})$ as follows:

from the Bayes' theorem

$$p(\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t}) = \frac{p(\mathbf{x}_t, \mathbf{x}_{t+1} \mid y_{1:t})}{p(\mathbf{x}_{t+1} \mid y_{1:t})} \quad (\text{E.101})$$

$$\propto p(\mathbf{x}_{t+1}, \mathbf{x}_t \mid y_{1:t}) \quad (\text{E.102})$$

from the Bayes' theorem again

$$= p(\mathbf{x}_{t+1} \mid \mathbf{x}_t, y_{1:t}) p(\mathbf{x}_t \mid y_{1:t}) \quad (\text{E.103})$$

from the conditional independence suggested by Eq.(5.1)

$$= p(\mathbf{x}_{t+1} \mid \mathbf{x}_t) p(\mathbf{x}_t \mid y_{1:t}) \quad (\text{E.104})$$

from Eqs.(5.10) and (8.3)

$$= \mathcal{N}(\mathbf{G}_{t+1}\mathbf{x}_t, \mathbf{W}_{t+1})\mathcal{N}(\mathbf{m}_t, \mathbf{C}_t). \quad (\text{E.105})$$

When comparing the above equation with Eq.(E.48), we see that the mean of the above distribution can be obtained by replacing X , V , \mathbf{m}_0 , \mathbf{C}_0 , and y in Eq.(E.58) with \mathbf{G}_{t+1} , \mathbf{W}_{t+1} , \mathbf{m}_t , \mathbf{C}_t , and \mathbf{x}_{t+1} , respectively. Specifically, the mean of the above distribution is obtained as follows:

$$\mathbb{E}[\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t}] = \mathbf{m}_t + \mathbf{C}_t \mathbf{G}_{t+1}^\top (\mathbf{G}_{t+1} \mathbf{C}_t \mathbf{G}_{t+1}^\top + \mathbf{W}_{t+1})^{-1} (\mathbf{x}_{t+1} - \mathbf{G}_{t+1} \mathbf{m}_t) \quad (\text{E.106})$$

from Eqs.(E.68) and (E.63)

$$= \mathbf{m}_t + \mathbf{C}_t \mathbf{G}_{t+1}^\top \mathbf{R}_{t+1}^{-1} (\mathbf{x}_{t+1} - \mathbf{a}_{t+1}). \quad (\text{E.107})$$

Thus, the mean of the smoothing distribution is derived as follows:

$$\mathbf{s}_t = \mathbb{E}[\mathbb{E}[\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t}] \mid y_{1:T}] \quad (\text{E.108})$$

from Eq.(E.107)

$$= \mathbb{E}[\mathbf{m}_t + \mathbf{C}_t \mathbf{G}_{t+1}^\top \mathbf{R}_{t+1}^{-1} (\mathbf{x}_{t+1} - \mathbf{a}_{t+1}) \mid y_{1:T}] \quad (\text{E.109})$$

from Eq.(8.8)

$$= \mathbf{m}_t + \mathbf{C}_t \mathbf{G}_{t+1}^\top \mathbf{R}_{t+1}^{-1} [\mathbf{s}_{t+1} - \mathbf{a}_{t+1}]. \quad (\text{E.110})$$

In addition, the variance is derived based on Eq.(8.8) as follows:

$$\mathbf{S}_t = \text{Var}[\mathbf{x}_t \mid y_{1:T}] \quad (\text{E.111})$$

from Eq.(E.42)

$$= \text{E}[\text{Var}[\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:T}] \mid y_{1:T}] + \text{Var}[\text{E}[\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:T}] \mid y_{1:T}] \quad (\text{E.112})$$

from Eq.(5.3)

$$= \text{E}[\text{Var}[\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t}] \mid y_{1:T}] + \text{Var}[\text{E}[\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t}] \mid y_{1:T}] \quad (\text{E.113})$$

Herein, the $\text{Var}[\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t}]$ is obtained in the same manner as in the above discussion for the mean and using Eq.(E.52):

$$\text{Var}[\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t}] = \mathbf{C}_t - \mathbf{C}_t \mathbf{G}_{t+1}^\top (\mathbf{G}_{t+1} \mathbf{C}_t \mathbf{G}_{t+1}^\top + \mathbf{W}_{t+1})^{-1} \mathbf{G}_{t+1} \mathbf{C}_t \quad (\text{E.114})$$

from Eq.(E.68)

$$= \mathbf{C}_t - \mathbf{C}_t \mathbf{G}_{t+1}^\top \mathbf{R}_{t+1}^{-1} \mathbf{G}_{t+1} \mathbf{C}_t. \quad (\text{E.115})$$

Thus, the variance of the smoothing distribution is derived as follows:

$$\mathbf{S}_t = \text{E}[\text{Var}[\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t}] \mid y_{1:T}] + \text{Var}[\text{E}[\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t}] \mid y_{1:T}] \quad (\text{E.116})$$

from Eqs.(E.115) and (E.107)

$$\begin{aligned} &= \text{E}[\mathbf{C}_t - \mathbf{C}_t \mathbf{G}_{t+1}^\top \mathbf{R}_{t+1}^{-1} \mathbf{G}_{t+1} \mathbf{C}_t \mid y_{1:T}] \\ &\quad + \text{Var}[\mathbf{m}_t + \mathbf{C}_t \mathbf{G}_{t+1}^\top \mathbf{R}_{t+1}^{-1} (\mathbf{x}_{t+1} - \mathbf{a}_{t+1}) \mid y_{1:T}] \end{aligned} \quad (\text{E.117})$$

from Eq.(8.8)

$$= \mathbf{C}_t - \mathbf{C}_t \mathbf{G}_{t+1}^\top \mathbf{R}_{t+1}^{-1} \mathbf{G}_{t+1} \mathbf{C}_t + \mathbf{C}_t \mathbf{G}_{t+1}^\top \mathbf{R}_{t+1}^{-1} \mathbf{S}_{t+1} \mathbf{R}_{t+1}^{-1} \mathbf{G}_{t+1} \mathbf{C}_t \quad (\text{E.118})$$

$$= \mathbf{C}_t + \mathbf{C}_t \mathbf{G}_{t+1}^\top \mathbf{R}_{t+1}^{-1} [\mathbf{S}_{t+1} - \mathbf{R}_{t+1}] \mathbf{R}_{t+1}^{-1} \mathbf{G}_{t+1} \mathbf{C}_t. \quad (\text{E.119})$$

E.3 Solution Using MCMC

E.3.1 FFBS

The joint smoothing distribution $p(\mathbf{x}_{0:T} \mid y_{1:T})$ can be transformed as follows:

$$p(\mathbf{x}_{0:T} \mid y_{1:T}) = p(\mathbf{x}_0, \mathbf{x}_{1:T} \mid y_{1:T}) \quad (\text{E.120})$$

from the Bayes' theorem

$$= p(\mathbf{x}_0 \mid \mathbf{x}_{1:T}, y_{1:T}) p(\mathbf{x}_{1:T} \mid y_{1:T}) \quad (\text{E.121})$$

$$= p(\mathbf{x}_0 \mid \mathbf{x}_{1:T}, y_{1:T}) p(\mathbf{x}_1, \mathbf{x}_{2:T} \mid y_{1:T}) \quad (\text{E.122})$$

apply Bayes' theorem to the last term

$$= p(\mathbf{x}_0 \mid \mathbf{x}_{1:T}, y_{1:T}) p(\mathbf{x}_1 \mid \mathbf{x}_{2:T}, y_{1:T}) p(\mathbf{x}_{2:T} \mid y_{1:T}) \quad (\text{E.123})$$

apply Bayes' theorem to the last term repeatedly

$$= p(\mathbf{x}_T \mid y_{1:T}) \prod_{t=0}^{T-1} p(\mathbf{x}_t \mid \mathbf{x}_{t+1:T}, y_{1:t}) \quad (\text{E.124})$$

simplify the last term of the product from Eq. (C.8)

$$= p(\mathbf{x}_T \mid y_{1:T}) \prod_{t=T-1}^0 p(\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t}). \quad (\text{E.125})$$

The above equation implies that as we consider the term $p(\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t})$ in the time reverse direction starting from the filtering distribution at time point T , we obtain the joint smoothing distributions sequentially. In the linear Gaussian state-space model, all the terms in Eq. (E.125) follow a normal distribution. Specifically, the $p(\mathbf{x}_T \mid y_{1:T})$ becomes the normal distribution with mean \mathbf{m}_T and covariance \mathbf{C}_T , and $p(\mathbf{x}_t \mid \mathbf{x}_{t+1}, y_{1:t})$ also becomes the normal distribution with mean in Eq. (E.107) and covariance in Eq. (E.115).

E.4 Particle Filter

E.4.1 Particle Filtering

To derive a discrete approximation of the filtering distribution $p(\mathbf{x}_t \mid y_{1:t})$, we first obtain a discrete approximation of the joint posterior distribution $p(\mathbf{x}_{0:t} \mid y_{1:t})$ and then marginalize it. We obtain the discrete approximation of joint posterior distribution $p(\mathbf{x}_{0:t} \mid y_{1:t})$ using importance sampling with the proposal distribution $q(\mathbf{x}_{0:t} \mid y_{1:t})$. Herein, let the ratio of the target density to the proposal density be the weight ω_t , and transform it for sequential updating as follows:

$$\omega_t = \frac{p(\mathbf{x}_{0:t} \mid y_{1:t})}{q(\mathbf{x}_{0:t} \mid y_{1:t})} \quad (\text{E.126})$$

$$= \frac{p(\mathbf{x}_{0:t} \mid y_t, y_{1:t-1})}{q(\mathbf{x}_{0:t} \mid y_{1:t})} \quad (\text{E.127})$$

$$= \frac{p(\mathbf{x}_{0:t}, y_t \mid y_{1:t-1}) / p(y_t \mid y_{1:t-1})}{q(\mathbf{x}_{0:t} \mid y_{1:t})} \quad (\text{E.128})$$

$$\propto \frac{p(\mathbf{x}_{0:t}, y_t \mid y_{1:t-1})}{q(\mathbf{x}_{0:t} \mid y_{1:t})} \quad (\text{E.129})$$

$$= \frac{p(\mathbf{x}_t, \mathbf{x}_{0:t-1}, y_t \mid y_{1:t-1})}{q(\mathbf{x}_{0:t} \mid y_{1:t})} \quad (\text{E.130})$$

$$= \frac{p(\mathbf{x}_t, y_t \mid \mathbf{x}_{0:t-1}, y_{1:t-1}) p(\mathbf{x}_{0:t-1} \mid y_{1:t-1})}{q(\mathbf{x}_{0:t} \mid y_{1:t})} \quad (\text{E.131})$$

assuming the Markov property for the proposal distribution in the denominator, we decompose it as $q(\mathbf{x}_{0:t} \mid y_{1:t}) = q(\mathbf{x}_t \mid \mathbf{x}_{0:t-1}, y_{1:t})q(\mathbf{x}_{0:t-1} \mid y_{1:t-1})$

$$= \frac{p(\mathbf{x}_t, y_t \mid \mathbf{x}_{0:t-1}, y_{1:t-1}) p(\mathbf{x}_{0:t-1} \mid y_{1:t-1})}{q(\mathbf{x}_t \mid \mathbf{x}_{0:t-1}, y_{1:t})q(\mathbf{x}_{0:t-1} \mid y_{1:t-1})} \quad (\text{E.132})$$

$$= \frac{p(\mathbf{x}_t, y_t \mid \mathbf{x}_{0:t-1}, y_{1:t-1})}{q(\mathbf{x}_t \mid \mathbf{x}_{0:t-1}, y_{1:t})} \omega_{t-1} \quad (\text{E.133})$$

$$= \frac{p(y_t \mid \mathbf{x}_t, \mathbf{x}_{0:t-1}, y_{1:t-1}) p(\mathbf{x}_t \mid \mathbf{x}_{0:t-1}, y_{1:t-1})}{q(\mathbf{x}_t \mid \mathbf{x}_{0:t-1}, y_{1:t})} \omega_{t-1} \quad (\text{E.134})$$

from Eqs. (5.2) and (5.1)

$$= \frac{p(y_t \mid y_{1:t-1}, \mathbf{x}_t) p(\mathbf{x}_t \mid \mathbf{x}_{t-1})}{q(\mathbf{x}_t \mid \mathbf{x}_{0:t-1}, y_{1:t})} \omega_{t-1}. \quad (\text{E.135})$$

Adding the normalization of ω_t to the above equation completes the algorithm prototype. Based on this mechanism, by sequentially repeating the weight calculation for samples from the proposal distribution, we can obtain a discrete approxima-

tion of the joint posterior distribution $\hat{p}(\mathbf{x}_{0:t} | y_{1:t}) = \sum_{n=1}^N \omega_t^{(n)} \delta_{\mathbf{x}_{0:t}^{(n)}}$, where δ_\star is a unit probability mass at \star . Marginalizing this discrete approximation at time point t is easy because the Markov property is assumed for the proposal distribution: the weights and realizations at time point t simply result in representing the discrete approximation $\hat{p}(\mathbf{x}_t | y_{1:t}) = \sum_{n=1}^N \omega_t^{(n)} \delta_{\mathbf{x}_t^{(n)}}$ of the filtering distribution. Note that Algorithm 11.1 has additional resampling processes from the viewpoint of practical use.

E.4.2 Particle Prediction

As described in the text, the basic concept of particle prediction corresponds to the case wherein the observations do not exist in the particle filtering; hence, we ignore the details of derivation.

E.4.3 Particle Smoothing

E.4.3.1 Kitagawa Algorithm

As described in the text, the Kitagawa algorithm basically corresponds to the case wherein the state is augmented as including the former state in the particle filtering; hence, we ignore the details of derivation.

E.4.3.2 FFBSi Algorithm

The joint smoothing distribution is expressed based on Eq. (E.125) as follows:

$$p(\mathbf{x}_{0:T} | y_{1:T}) \quad (\text{E.136})$$

$$= p(\mathbf{x}_T | y_{1:T}) \prod_{t=T-1}^0 p(\mathbf{x}_t | \mathbf{x}_{t+1}, y_{1:t}) \quad (\text{E.137})$$

the term $p(\mathbf{x}_t | \mathbf{x}_{t+1}, y_{1:t})$ is given by Eq. (E.104)

$$\propto p(\mathbf{x}_T \mid y_{1:T}) \prod_{t=T-1}^0 p(\mathbf{x}_{t+1} \mid \mathbf{x}_t) p(\mathbf{x}_t \mid y_{1:t}) \quad (\text{E.138})$$

= (filtering distribution at time point T) \times

$$\prod_{t=T-1}^0 (\text{state equation}) \times (\text{filtering distribution}). \quad (\text{E.139})$$

The above equation implies that as we consider the state equation for the filtering distribution in the time reverse direction starting from the filtering distribution at time point T , we obtain the joint smoothing distributions sequentially. Using the weights of the joint smoothing distribution obtained in this manner, Algorithm 11.3 applies resampling to the realizations of the filtering distribution.

References

1. Hirota, K., Ikoma, N.: Mathematics of Stochastic Process. Asakura Publishing Co., Ltd. (2001). [in Japanese]
2. Kailath, T., Sayed, A.H., Hassibi, B.: Linear Estimation. Prentice Hall (2000)
3. Katayama, T.: (New Edition) Applied Kalman filter. Asakura Publishing Co., Ltd. (2000). [in Japanese]
4. Nishiyama, K.: Optimal Filtering. Baifukan Co., Ltd. (2001). [in Japanese]
5. Ozaki, T., Kitagawa, G. (eds.): Time Series Analysis Method. Asakura Publishing Co., Ltd. (1998). [in Japanese]

Appendix F

Execution of Particle Filtering with Library

F.1 Example: Artificial Local-Level Model

This chapter considers the libraries **pomp** and **NIMBLE** and describes their basic notation and behavior. For this purpose, we perform filtering for the linear Gaussian state-space model with known parameters and compare the results with those obtained using Kalman filtering. For the model and data, we use the artificial local-level model and data prepared with Code 9.1 in Sect. 9.2.

F.1.1 **pomp**

The library **pomp** describes the model directly within the R code. This code is as follows.

Code F.1

```

1 > # <<Particle filtering (pomp) for local-level model with known parameters>>
2 >
3 > # Preprocessing
4 > set.seed(4521)
5 > library(pomp)
6 >
7 > # Presetting of particle filter
8 > N <- 10000           # Number of particles
9 >
10 > # Load data on artificial local-level model
11 > load(file = "ArtificialLocalLevelModel.RData")
12 >
13 > # Model: specification
14 >
15 > # State equation (draw)
16 > state_draw <- function(X, W, delta.t, ...){
17 +   c(X = rnorm(1, mean = X, sd = sqrt(W)))
18 + }
19 >
20 > # Observation equation (draw)
21 > obs_draw <- function(X, V, ...){
22 +   c(Y = rnorm(1, mean = X, sd = sqrt(V)))
23 + }
24 >
25 > # Observation equation (evaluation)
26 > obs_eval <- function(Y, X, V, log, ...){
27 +   dnorm(Y, mean = X, sd = sqrt(V), log = log)
28 + }
29 >
30 > # Model: generation
31 > pomp_mod <- pomp(data = data.frame(time = seq_along(y), Y = y),
32 +                      times = seq_along(y), t0 = 0,
33 +                      rprocess = discrete_time(step.fun = state_draw, delta.t = 1),
34 +                      rmeasure = obs_draw, dmeasure = obs_eval
35 + )
36 >
37 > # Particle filtering: execution
38 > pomp_smc_out <- pfilter(data = pomp_mod, Np = N,
39 +                             params = c(W = mod$W, V = mod$V, X.0 = mod$m0),
40 +                             save.states = TRUE
41 + )
42 >
43 > # Find mean, 2.5%, and 97.5% values
44 > pomp_m      <- sapply(1:t_max, function(t){
45 +   mean(pomp_smc_out@saved.states[[t]])
46 + })
47 > pomp_m_quant <- sapply(1:t_max, function(t){
48 +   quantile(pomp_smc_out@saved.states[[t]], probs=c(0.025, 0.975))
49 + })
50 >
51 > # Ignore the display of following codes

```

The naming for variables used in the code is consistent with that used in the explanation thus far. First, we load the library **pomp**. We set the number N of particles to 10,000 and then load the data regarding the artificial local-level model prepared with Code 9.1. The description thus far is the preparation; the following describes the explicit processing of **pomp**. Then, we describe the state and observation equations for specifying the model. Those descriptions are based on the probability distribution expressions for the state and observation equations in Eqs. (5.10) and (5.11), respectively. While **pomp** can use both R and C as model description languages, here is an example in R. For the state equation, we define the user-defined function `state_draw()` for state drawing. For the observation equation, we define the user-defined functions for drawing and evaluating the observations as `obs_draw()` and `obs_eval()`, respectively. When executing the particle filter with **pomp**, we must prepare two types of definition for the observation equation. We then generate a model object with the function `pomp()` and save the result in `pomp_mod`. The arguments of this function are `data` for the data frame including observations, `times` for the name indicating the time in the data, `t0` for the time value corresponding to the prior distribution, `rprocess` for the user-defined function regarding the state equation (for drawing), and `rmeasure & dmeasure` for the user-defined functions regarding the observation equation for drawing & evaluation, respectively. We set the user-defined function to `rprocess` via the **pomp** function `discrete_time()`. We then perform particle filtering using the function `pfilter()` and save the result in `pomp_smc_out`. The arguments of this function are `data` for the generated model object, `Np` for the number of particles, `params` for the various settings to be passed to **pomp**, and `save.states` for the indicator of whether to include the state in the return value. The name `X.0` in `params` depends on the name of the state variable; it becomes `name of the state variable.0` in general. The element `X.0` is set to the state value for the prior distribution, and this example sets the mean of the prior distribution. Note that the **pomp** performs resampling at every time point; we need not use the threshold for adaptively determining the execution of resampling. Finally, we calculate the summary statistics for the state contained in the return value `pomp_smc_out`. Figure F.1 shows the comparison plots between the results obtained by particle and Kalman filtering.

As shown in Fig. F.1, both results are almost identical.

Through the above simple example, we have confirmed that **pomp** can accurately estimate the linear Gaussian state-space model.

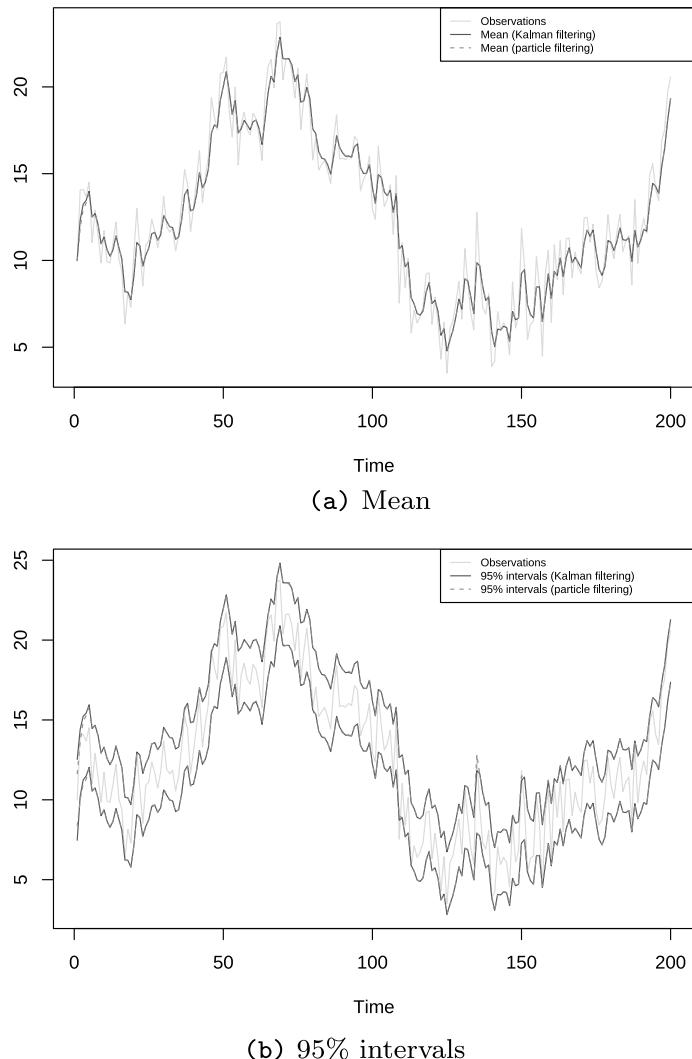


Fig. F.1 Particle filtering with **pomp** and Kalman filtering (linear Gaussian state-space model with known parameters)

F.1.2 NIMBLE

The library **NIMBLE** describes the model directly within the R code. This code is as follows.

Code F.2

```

1 > # <<Particle filtering (NIMBLE) for local-level model with known parameters>>
2 >
3 > # Preprocessing
4 > set.seed(4521)
5 > library(nimble)
6 >
7 > # Presetting of particle filter
8 > N <- 10000           # Number of particles
9 >
10 > # Load data on artificial local-level model
11 > load(file = "ArtifitialLocalLevelModel.RData")
12 >
13 > # *Note: Assuming that the time point of the prior distribution corresponds to one,
   ← we regard the shifted time points (from 2 to t_max+1) as the original ones (from
   ← 1 to t_max).
14 >
15 > # Data formatting (adding the forefront dummy corresponding to prior distribution)
16 > y <- c(NA_real_, y)
17 >
18 > # Model: specification
19 > nimble_mod_script <- nimbleCode({
20 +   # State equation
21 +   x[1] ~ dnorm(m0, 1/C0)
22 +   for (t in 2:(t_max+1)){
23 +     x[t] ~ dnorm(x[t-1], 1/W)
24 +   }
25 +
26 +   # Observation equation
27 +   # y[1] as dummy
28 +   for (t in 2:(t_max+1)){
29 +     y[t] ~ dnorm(x[t], 1/V)
30 +   }
31 + })
32 >
33 > # Model: generation
34 > nimble_mod <- nimbleModel(code = nimble_mod_script,
35 +                               data = list(y = y),
36 +                               constants = list(t_max = t_max,
37 +                                                 W = mod$W, V = mod$V,
38 +                                                 m0 = mod$m0, C0 = mod$C0)
39 + )
40 defining model...
41 building model...
42 setting data and initial values...
43 running calculate on model (any error reports that follow may simply reflect missing
   ← values in model variables) ...
44 checking model sizes and dimensions... This model is not fully initialized. This is
   ← not an error. To see which variables are not initialized, use
   ← model$initializeInfo(). For more information on model initialization, see
   ← help(modelInitialization).
45 model building finished.

```

```

46 >
47 > # Particle filtering: generation
48 > nimble_smc_out <- buildBootstrapFilter(model = nimble_mod,
49 +                                         nodes = "x",
50 +                                         control = list(thresh = 1.0,
51 +                                                       saveAll = TRUE)
52 + )
53 >
54 > # Model: compilation
55 > compiled_nimble_mod <- compileNimble(nimble_mod)
56 compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++
57 ← compilation details.
58 compilation finished.
59 >
60 > # Particle filtering: compilation
61 > compiled_nimble_smc_out <- compileNimble(nimble_smc_out, project = nimble_mod)
62 compiling... this may take a minute. Use 'showCompilerOutput = TRUE' to see C++
63 ← compilation details.
62 compilation finished.
63 >
64 > # Particle filtering: execution
65 > compiled_nimble_smc_out$run(m = N)      # Display likelihood in the case of
66 ← completion
66 [1] -420.5497
67 >
68 > # Retrieve state and weight, then format the result by removing the forefront
69 ← corresponding to the prior distribution, etc.
69 > nimble_x <- as.matrix(compiled_nimble_smc_out$mvWSamples, "x" )
70 > nimble_x <- t(   nimble_x[, -1])
71 > nimble_w <- as.matrix(compiled_nimble_smc_out$mvWSamples, "wts")
72 > nimble_w <- t(exp(nimble_w[, -1]))
73 > y <- y[-1]
74 >
75 > # Find mean, 2.5%, and 97.5% values
76 > nimble_m      <- sapply(1:t_max, function(t){
77 +                           weighted.mean(nimble_x[t, ], w = nimble_w[t, ])
78 +                         })
79 > nimble_m_quant <- lapply(c(0.025, 0.975), function(quant){
80 +                           sapply(1:t_max, function(t){
81 +                             weighted.quantile(nimble_x[t, ], w = nimble_w[t, ],
82 +                                               probs = quant)
83 +                           }))
84 + }
85 >
86 > # Ignore the display of following codes

```

The naming for variables used in the code is consistent with that used in the explanation thus far. First, we load the library **nimble**. We set the number N of particles to 10,000 and then load the data regarding the artificial local-level model prepared with Code 9.1. Assuming that the time point for the prior distribution corresponds to 1, we consider the shifted time points $(2, \dots, t_{\text{max}}+1)$ as the original ones $(1, \dots, t_{\text{max}})$. For easy handling in conjunction with such a shifted time point, we add one dummy object at the beginning of the observations. The description thus far is preparation for main processing; the following describes the specific processing of **NIMBLE**. We first create an object specifying a model using the function `nimbleCode()` and save the result in `nimble_mod_script`. In the argument of this function, we describe the state and observation equations based on their

probability distribution expressions in Eqs. (5.10) and (5.11), respectively. Since the notation of **NIMBLE** basically conforms to BUGS language, the variance of the normal distribution is set in the form of its inverse, i.e., the precision. The beginning of the observations is set to the dummy; we need not describe the observation equation for $y[1]$. We then generate the model object using the function `nimbleModel()` and save the result in `nimble_mod`. The arguments of this function are `code` for the model specifying object, `data` for a list containing the observations, and `constants` for the various settings to be passed to **NIMBLE**. We then create an object for particle filtering using the function `buildBootstrapFilter()`, and save the result in `nimble_smc_out`. The arguments of this function are `model` for the generated model object, `nodes` for the variable name to be estimated, and `control` for a list containing various settings. Regarding the elements in `control`, `thresh` and `saveAll` indicate the threshold for adaptively determining the execution of resampling and indicator of whether to include the state value in the return value, respectively. The library **NIMBLE** performs resampling when the effective number of particles decreases to below the number of particles \times `thresh`. Since we decide to perform resampling at every time point, `thresh` is set as 1.0. Next, we compile both the generated model object and object for particle filtering using the function `compileNimble()`. When compiling the object for particle filtering, we must specify the generated model object through the argument `project`. Finally, we execute particle filtering with `compiled_nimble_smc_out$run`. At this time, we set the number of particles to the argument `m`. When the execution is completed, the likelihood for all time points is displayed on the console, and the estimated particles, i.e., the state values before resampling and their log weights, are saved in `compiled_nimble_smc_out`. We extract the particle values from `compiled_nimble_smc_out` using the function `as.matrix()` and format them for easy handling. We then calculate the summary statistics. We use the utility function `weighted.quantile()` of the **dlm** introduced in Appendix B to obtain the quantile values. Figure F.2 shows the comparison plots between the results obtained using particle and Kalman filtering.

As shown in Fig. F.2, the two results are nearly identical; graph lines overlap and cannot be distinguished from each other.

Through the above simple example, we have confirmed that **NIMBLE** can estimate the linear Gaussian state-space model accurately.

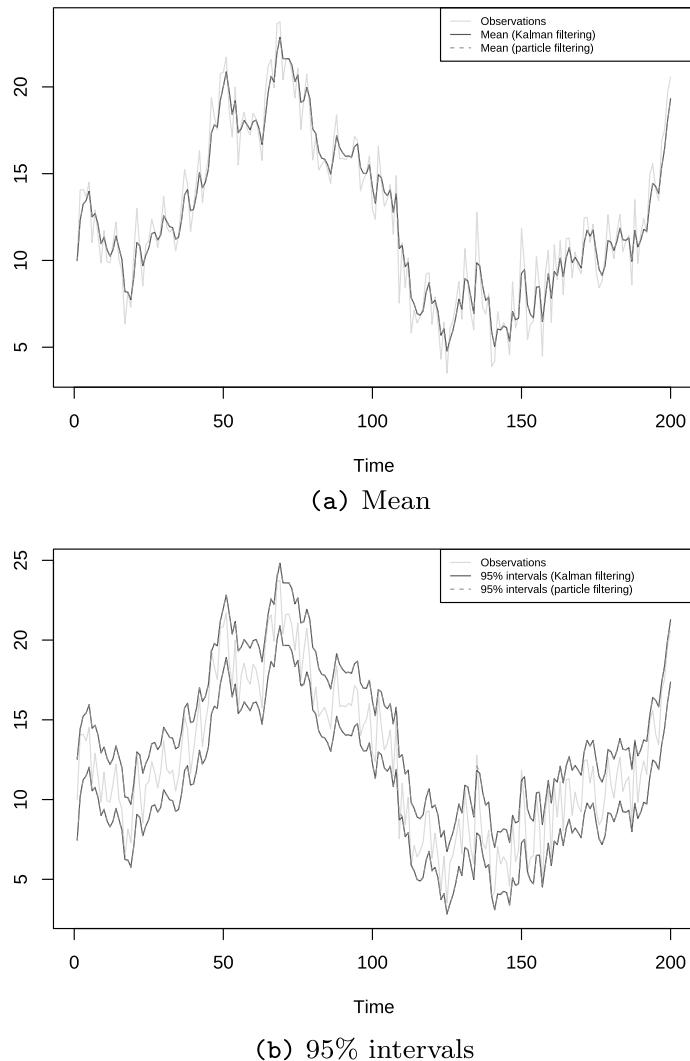


Fig. F.2 Particle filtering with **NIMBLE** and Kalman filtering (linear Gaussian state-space model with known parameters)

Index

Symbols

in code

+^{130, 142}
a^{116, 119, 307}
acf()³⁷
addInt^{163, 166, 310}
alpha⁴⁸
ar^{158, 310}
ARtransPars()^{158, 311}
as.matrix()³⁴¹
as.numeric^{117, 119, 121}
beta⁴⁸
bsmc()³⁰⁴
bsmc2()³⁰⁴
build^{113, 309}
buildAuxiliaryFilter()³⁰⁴
buildBootstrapFilter()³⁴¹
buildLiuWestFilter()³⁰⁴
C0^{110, 142, 145, 146, 158, 163, 166, 305, 309, 310, 312}
code³⁴¹
compileNimble()³⁴¹
constants³⁴¹
control³⁴¹
convergence^{113, 309}
d³⁰⁸
D.C^{116, 117, 306, 307}
D.R^{116, 306, 307}
D.S^{120, 306, 308}
data^{191, 196, 204, 337, 341}
data block^{188, 195, 202, 213, 288}
discrete_time()³³⁷
dlm^{110, 305, 309, 310, 312}
dlm_local_level_loglik()^{288, 290}
dlmBSample()^{201, 205, 206, 308}
dlmFilter()^{116, 205, 268, 307}

dlmFiltered^{116, 118, 120, 124, 125, 307, 308, 310}
dlmForecast()^{93, 118, 119, 131, 147, 307}
dlmLL()^{122, 308}
dlmMLE()^{113, 114, 122, 309}
dlmModARMA()^{158, 310}
dlmModPoly()^{93, 110, 142, 144, 146, 208, 282, 309}
dlmModReg()^{163, 166, 310}
dlmModSeas()^{142, 145, 208, 309}
dlmModTrig()^{145, 310}
dlmSmooth()^{93, 120, 307}
dlmSvd2var()^{117, 121, 308}
dmeasure³³⁷
dropFirst()^{117, 120, 121}
dV^{110, 142, 145, 146, 158, 163, 166, 309, 310, 312}
dW^{110, 142, 145, 146, 163, 166, 309, 310, 312}
End²⁴
end²⁴
extract()¹⁹²
f^{116, 119, 307}
FF^{110, 305}
fft()⁴¹
filter()⁹³
fitted⁴⁸
for^{189, 231, 233, 238, 257, 264, 271}
frequency^{24, 142, 309}
functions block²⁸⁸
gamma⁴⁸
gaussian_dlm_obs()^{200, 202, 204, 213, 286}
generated_quantities block²⁸⁶
GG^{110, 305}

hessian, 114
 hist(), 35
 HoltWinters(), 46, 48, 50, 51
 JFF, 110, 164, 305
 JGG, 110, 305
 jholidays(), 172
 JV, 110, 305
 JW, 110, 305
 log1p(), 242
 lp__, 191, 197
 m, 116, 117, 307, 341
 m0, 110, 142, 145, 146, 158, 163, 166,
 305, 309, 310, 312
 ma, 158, 310
 mod, 116, 118, 120, 122, 307, 308
 model, 195, 341
 model block, 189, 195, 202, 211, 213,
 288
 modFilt, 205, 308
 n.ahead, 50
 NA, 24, 25, 35, 36
 nAhead, 118, 307
 newObs, 307
 newStates, 307
 Nile, 1, 23–26, 30, 108, 111, 113, 116,
 120, 122
 nimbleCode(), 340
 nimbleModel(), 341
 nodes, 341
 Np, 337
 object, 124, 125, 191
 om, 145, 310
 optim(), 113, 114, 309
 order, 110, 309
 par, 113, 158, 163, 309
 parameters block, 188, 195, 202,
 211, 213, 288
 params, 337
 parm, 113, 309
 pars, 191, 196, 204
 pfilter(), 337
 plot(), 24, 32, 48, 51
 pomp(), 337
 predict(), 50, 51
 project, 341
 Q, 119, 307
 q, 145, 310
 qnorm(), 117, 119, 121
 qqline(), 125
 qqnorm(), 125
 R, 119, 307
 raw, 311
 read.csv(), 24
 residuals(), 49, 125
 rmeasure, 337
 rprocess, 337
 s, 120, 145, 308, 310
 sample(), 231, 242
 sampleNew, 118, 307
 sampling(), 191, 196, 204, 212, 290
 save.states, 337
 saveAll, 341
 sd, 125
 seed, 191
 seq(), 27
 shapiro.test(), 125
 sigma2, 158, 310
 stan_model(), 191
 Start, 24
 stepfun(), 244
 str(), 27
 summary(), 35
 t0, 337
 tau, 145, 310
 thresh, 341
 time(), 26
 times, 337
 traceplot(), 191
 transformed parameters block,
 213, 288
 ts(), 24
 ts.plot(), 25
 ts.union(), 25, 49, 147
 tsdiag(), 124
 tsp(), 26
 u, 308
 U.C, 116, 117, 306, 307
 U.R, 116, 306, 307
 U.S, 120, 306, 308
 UKDriverDeaths, 208
 UKgas, 30
 V, 110, 290, 298, 305
 value, 122
 W, 110, 305
 weekdays(), 27
 weighted.quantile(), 311, 341
 window(), 24
 X, 163, 164, 166, 290, 298, 305, 310
 y, 113, 116, 120, 122, 147, 307–311

A

additive, 45
 angular frequency, 39
 anomaly detection, 277
 AR coefficient, 65

- AR model, 18
 ARIMA model, 65, 150
 ARMA model, 65, 150
 autocorrelation coefficient, 16, 37, 54, 124
 autocovariance, 16
 autoregressive integrated moving average model *see* ARIMA model
 autoregressive model *see* AR model
 autoregressive moving average model *see* ARMA model
 auxiliary particle filter, 254, 262
 auxiliary variable, 254
- B**
 batch solution, 56
 Bayes' theorem, 13
 Bayesian estimation, 20, 85
 Bayesian updating, 14, 70
 black box approach, 66
 bootstrap filter, 223
- C**
 change point, 277
 component decomposition, 129
 conditional distribution, 13
 conditional independence, 62, 314
 conditional probability, 13
 confidence interval, 101
 conjugate prior distribution, 179
 convolution, 91, 318
 correlation coefficient, 15
 correlation function, 16
 covariance, 14
- D**
 DAG, 61, 62, 314
 Date class, 26
 decomposition, 43
 delta method, 114
 deterministic method, 3
 differences, 32
 directed acyclic graph *see* DAG
 DLM *see* linear Gaussian state-space model
 dynamic linear model *see* linear Gaussian state-space model
- E**
 effective sample size, 191, 255
 EWMA *see* exponentially weighted moving average
- expected value, 8
 explanatory variable, 160
 exploratory data analysis, 176
 exponential weight, 45, 261
 exponentially weighted moving average, 45, 74, 99
- F**
 fast Fourier transform, 41
 fat-tailed distribution, 284
 FFBS, 186, 201, 330
 FFBSi algorithm, 228, 238, 332
 FFT *see* fast Fourier transform
 filtering, 3, 69, 75, 98, 222
 filtering distribution, 70, 77, 98, 223
 five-number summary, 33
 fixed-interval smoothing, 3
 fixed-lag smoothing, 3, 299
 fixed-point smoothing, 3
 forecast, 3
 forward filtering backward sampling *see* FFBS
 Fourier coefficient, 39
 Fourier series, 38
 Fourier transform, 40
 frequency, 38
 frequency domain, 38
 frequency spectrum, 38
 full conditional distribution, 182, 185
- G**
 Gaussian distribution, 9
 general state-space model, 56, 66
 Gibbs method, 182, 185
- H**
 Hessian matrix, 114
 histogram, 33
 Holt–Winters method, 45
 horseshoe distribution, 284
 hyperparameters, 61
- I**
 importance sampling, 180, 221
 impulse response, 318
 independent, 14
 innovations, 102, 122
 integrated random walk, 135
 intervention variable, 160

J

joint distribution, 13, 63
 joint posterior distribution of the state, 69
 joint probability, 13

K

Kalman filter, 56, 97
 Kalman gain, 99
 kernel smoothing, 260
 Kitagawa algorithm, 227, 235, 299, 332

L

lag, 16
 law of iterated expectations, 322
 law of total variance, 322
 level component, 45
 level + trend + season, 43
 likelihood, 14, 70, 83, 101, 122, 189, 195, 259
 linear Gaussian state-space model, 56, 67
 linear growth model, 133
 linear time-invariant system, 318
 Liu and West filter, 259, 262
 local-level model, 99, 130
 local-trend model, 133
 log-likelihood, 20
 log marginal likelihood, 84
 logarithmic transformation, 32, 153, 169, 208
 logsumexp, 241

M

M–H method *see* Metropolis–Hastings method
 MA coefficient, 65
 MAP *see* maximum a posteriori probability
 MAPE, 55, 149
 marginal distribution, 13
 marginal likelihood, 84
 marginal posterior distribution of the state, 70
 marginal probability, 13
 marginalization, 13, 70, 192, 228
 marginalized particle filter, 266
 Markov chain Monte Carlo *see* MCMC
 Markov property, 60
 matrix inversion lemmas, 322
 maximum a posteriori probability, 86
 maximum likelihood estimation, 20
 maximum likelihood estimator, 85
 maximum likelihood method, 20, 85, 111

MCMC, 56, 180

mean absolute percentage error *see* MAPE
 mean squared error, 55
 mean value, 8
 measurement equation *see* observation equation
 measurement matrix *see* observation matrix
 measurement noise *see* observation noise
 Metropolis method, 181
 Metropolis–Hastings method, 182
 missing observations, 24, 35, 80, 103
 mixture Kalman filter, 266
 Monte Carlo filter, 223
 multidimensional normal distribution, 10
 multinomial resampling, 242
 multiple regression, 160
 multivariate, 3

N

noninformative prior distribution, 189
 nonlinear non-Gaussian state-space model, 67
 nonstationary, 18
 normal distribution, 9
 normalized innovations, 123

O

observable canonical form, 150
 observation equation, 63, 67
 observation matrix, 67
 observation noise, 63, 67
 one-step-ahead predictive distribution, 76, 78, 98, 223
 one-step-ahead predictive likelihood, 76, 78, 98, 223
 order of AR, 65
 order of MA, 65
 outliers, 35, 279

P

parameter, 7, 84, 110, 194, 259
 particle degeneracy, 224
 particle filter, 56, 220
 particles, 220
 periodical model, 135
 polynomial model, 134
 posterior distribution, 14, 69, 70, 189, 195
 posterior distribution of the state, 69
 power spectrum, 40, 91, 317
 precision, 8, 73
 prediction, 3, 69, 79, 102, 225

prediction error, 99, 102, 123
 predictive distribution, 70, 79, 103, 226
 prior distribution, 14, 70, 189, 195
 probability density function, 7
 probability distribution, 7
 process noise *see* state noise
 product rule of probability, 13
 proposal distribution, 180, 221

R

random walk, 109
 random walk plus noise model, 109, 130
 Rao–Blackwellization, 266, 270
 realization, 7, 184, 220
 regression coefficient, 160
 regression intercept, 160
 regression model, 160
 reproducibility, 10
 resampling, 223
 residuals, 43, 102, 123
 residuals resampling, 243
 RTS algorithm, 106

S

seasonal component, 45
 seasonal model, 135
 self-organizing, 259
 sequential Monte Carlo method, 220
 sequential solution, 56
 simulation smoothing, 186
 single regression, 160
 singular value decomposition, 306
 SIR method, 220
 smoothing, 3, 69, 81, 90, 105, 180, 226, 227
 smoothing distribution, 70, 82, 105, 228
 sparsity, 285
 square root Kalman filter, 306
 Stan, 187
 standard deviation, 8
 standard error, 114
 state, 60, 69
 state equation, 63, 67
 state evolution matrix *see* state transition matrix
 state noise, 63, 67

state transition matrix, 67
 state-space model, 59, 66
 stationary, 18
 stationary Kalman filter, 99
 stochastic method, 4
 stochastic process, 14
 stratified resampling, 243
 strictly stationary, 18
 structural change, 277
 system equation *see* state equation
 system matrix *see* state transition matrix
 system noise *see* state noise
 systematic resampling, 243

T

time domain, 38
 time-invariant, 7, 278
 time-varying, 7, 164, 278, 305
 time-varying Kalman filter, 280
 trace plot, 184
 transfer function, 91, 318
 trend component, 45
 ts class, 23

U

univariate, 3

V

variance, 8

W

weakly stationary, 18
 white box approach, 66
 white noise, 18, 63
 whitening filter, 319
 Wiener filter, 56, 89
 Wiener–Hopf equation, 319
 Wiener–Khinchin theorem, 318

Z

z-transform, 91, 317