Springer Texts in Statistics

Andrzej Gałecki
Tomasz Burzykowski

Linear Mixed-Effects Models Using R

A Step-by-Step Approach

Springer

Springer Texts in Statistics

Series Editors:
G. Casella
S.E. Fienberg
I. Olkin

For further volumes:
http://www.springer.com/series/417

Andrzej Gałecki • Tomasz Burzykowski

Linear Mixed-Effects Models Using R

A Step-by-Step Approach

Springer

Andrzej Gałecki
University of Michigan
300 North Ingalls Building
Ann Arbor
Michigan
USA

Tomasz Burzykowski
Center for Statistics
Hasselt University
Agoralaan D
Diepenbeek
Belgium

ISSN 1431-875X
ISBN 978-1-4614-3899-1 ISBN 978-1-4614-3900-4 (eBook)
DOI 10.1007/978-1-4614-3900-4
Springer New York Heidelberg Dordrecht London

Library of Congress Control Number: 2012941857

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Bliskim mojemu sercu
Oli i Łukaszowi
Rodzinie i Nauczycielom
Dekadentom
– A.T.G.

Moim najbliższym i przyjaciołom
– T.B.

In memory of Tom Ten Have

Preface

Linear mixed-effects models (LMMs) are powerful modeling tools that allow for the analysis of datasets with complex, hierarchical structures. Intensive research during the past decade has led to a better understanding of their properties. The growing body of literature, including recent monographs, has considerably increased their popularity among applied researchers. There are several statistical software packages containing routines for LMMs. These include, for instance, SAS, SPSS, STATA, S+, and R. The major advantage of R is that it is a freely available, dynamically developing, open-source environment for statistical computing and graphics.

The goal of our book is to provide a description of tools available for fitting LMMs in R. The description is accompanied by a presentation of the most important theoretical concepts of LMMs. Additionally, examples of applications from various research areas illustrate the main features of both theory and software. The presented material should allow readers to obtain a basic understanding of LMMs and to apply them in practice. In particular, we elected to present several theoretical concepts and their practical implementation in R in the context of simpler, more familiar classes of models such as e.g., the classical linear regression model. Based on these concepts, more advanced classes of models, such as models with heterogenous variance and correlated residual errors, along with related concepts are introduced. In this way, we incrementally set the stage for LMMs, so that the exposition of the theory and R tools for these models becomes simpler and clearer. This structure naturally corresponds to the object-oriented programming concept, according to which R functions/methods for simpler models are also applicable to the more complex ones.

We assume that readers are familiar with intermediate linear algebra, calculus, and the basic theory of statistical inference and linear modeling. Thus, the intended audience for this book is graduate students of statistics and applied researchers in other fields.

Our exposition of the theory of various classes of models presented in the book focuses on concepts, which are implemented in the functions available in R. Readers

interested in a more detailed description of the theory are referred to appropriate theoretical monograph books, which we indicate in the text.

There are a large number of R packages that can be used to fit LMMs. Rather than attempting to describe all of these packages, we focus mainly on two of them, namely, **nlme** and **lme4.0**. In this way, we can provide a more detailed account of the tools offered by the two packages, which include a wide variety of functions for model fitting, diagnostics, inference, etc.

The package **nlme** includes functions, which allow fitting of a wide range of linear models and LMMs. Moreover, it has been available for many years and its code has been stable for some time now. Thus, it is a well-established R tool.

In turn, **lme4.0** is a developmental branch version of the **lme4** package. The latter has been under development for several years. Both packages offer an efficient computational implementation and an enhanced syntax, though at the cost of a more restricted choice of LMMs, as compared to the **nlme** package. At the time of writing of our book, the implementation of LMMs in **lme4** has undergone major changes in terms of internal representation of the objects representing fitted models. Consequently, at the beginning of 2012, a snapshot version of **lme4** has been made available to the R users under the name of **lme4.0**. As we anticipate that **lme4.0** will not undergo any major changes, we decided to present it in more detail in our book. We would like to underscore, however, that the major part of the syntax, presented in the book, will be applicable both to **lme4** and **lme4.0**.

All classes of linear models presented in the book are illustrated using data from a particular dataset. In this way, the differences between the various classes of models, as well as differences in the R software, can be clearly delineated. LMMs, which are the main focus of the book, are also illustrated using three additional datasets, which extend the presentation of various aspects of the models and R functions. We have decided to include the direct output of R commands in the text. In this way, readers who would like to repeat the analyses conducted in the book can directly check their own output. However, in order to avoid the risk of incompatibility with updated versions of the software, the results of the analyses have also been summarized in the form of edited tables.

To further support those readers who are interested in actively using the material presented in the book, we have developed the package **nlmeU**. It contains all the datasets and R code used in the book. The package is downloadable at `http://www-personal.umich.edu/~agalecki/`.

We hope that our book, which aims to provide a state-of-the-art description of the details of implementing of LMMs in R, will support a widespread use of the models by applied researchers in a variety of fields including biostatistics, public health, psychometrics, educational measurement, and sociology.

When working on the text, we received considerable assistance and valuable comments from many people. We would like to acknowledge Geert Molenberghs (Hasselt University and the Catholic University of Leuven), Geert Verbeke (Catholic University of Leuven), José Pinheiro (Novartis AG), Paul Murrell (Auckland University), Przemysław Biecek (Warsaw University), Fabian Scheipl (Ludwig Maximilian University of Munich), Joshua Wiley (University of California, Los

Angeles), Tim Harrold (NSW Ministry of Health), Jeffrey Halter (University of Michigan), Shu Chen (University of Michigan), Marta Gałecka (Weill Cornell Medical College), anonymous reviewers and members of the R-sig-ME discussion group led by Douglas Bates (University of Wisconsin-Madison), and Ben Bolker (McMaster University) for their comments and discussions at various stages during the preparation of the book. We also acknowledge a formidable effort on the part of the developers of the **nlme** and **lme4** packages. Without them this book would not have been written. In particular, Ben Bolker's contribution was invaluable to ensure that the majority of the **lme4.0** syntax used in the text can also be used with the **lme4** package. We are grateful to John Kimmel for encouraging us to consider writing the book and to Marc Strauss, Hannah Bracken, and Brian Halm from Springer for their editorial assistance and patience. Finally, we gratefully acknowledge financial support from the Claude Pepper Center grants AG08808 and AG024824 from the National Institute of Aging and from the IAP Research Network P7/06 of the Belgian Government (Belgian Science Policy).

Ann Arbor, MI, USA — Andrzej Gałecki
Diepenbeek, Belgium, and Warszawa, Poland — Tomasz Burzykowski

Contents

Part I
Introduction

Chapter 1
Introduction

1.1 The Aim of the Book

Linear mixed-effects models (LMMs) are an important class of statistical models that can be used to analyze correlated data. Such data include *clustered observations*, *repeated measurements*, *longitudinal measurements*, *multivariate observations*, etc.

The aim of our book is to help readers in fitting LMMs using R software. R (www.r-project.org) is a language and an environment aimed at facilitating implementation of statistical methodology and graphics. It is an open-source software, which can be freely downloaded and used under the GNU General Public License. In particular, users can define and share their own functions, which implement various methods and extend the functionality of R. This feature makes R a very useful platform for propagating the knowledge and use of statistical methods.

We believe that, by describing selected tools available in R for fitting LMMs, we can promote the broader application of the models. To help readers less familiar with this class of linear models (LMs), we include in our book a description of the most important theoretical concepts and features of LMMs. Moreover, we present examples of applications of the models to real-life datasets from various areas to illustrate the main features of both theory and software.

1.2 Implementation of Linear Mixed-Effects Models in R

There are many packages in R, which contain functions that allow fitting various forms of LMMs. The list includes, but is not limited to, packages **amer**, **arm**, **gamm**, **gamm4**, **GLMMarp**, **glmmAK**, **glmmBUGS**, **heavy**, **HGLMMM**, **lme4.0**, **lmec**, **lmm**, **longRPart**, **MASS**, **MCMCglmm**, **nlme**, **PSM**, and **pedigreemm**. On the one hand, it would seem that the list is rich enough to allow for

A. Gałecki and T. Burzykowski, *Linear Mixed-Effects Models Using R: A Step-by-Step Approach*, Springer Texts in Statistics, DOI 10.1007/978-1-4614-3900-4_1,

a widespread use of LMMs. On the other hand, the number of available packages leads to difficulty in evaluating their relative merits and making the most suitable choice.

It is virtually impossible to describe the contents of all of the packages mentioned above. To facilitate and promote the use of LMMs in practice, it might be more useful to provide details for a few of them, so that they could be used as a starting point. Therefore, we decided to focus on the packages **nlme** and **lme4.0**, for several reasons. First, they contain the functions `lme()` and `lmer()`, respectively, which are specifically designed for fitting a broad range of LMMs. Second, they include many tools useful for applications such as model diagnostics. Finally, many other packages, which add new LMM classes or functionalities, depend on and are built around **nlme** and/or **lme4.0**. Examples include, but are not limited to, packages **amer**, **gamm**, **gamm4**, or **RLRsim**.

The reader may note that we focus more on the package **nlme** than on **lme4.0**. The main reason is that the former has already been around for some time. Thus, its code is stable. On the other hand, the package **lme4.0** is a development version of **lme4** made available at the beginning of 2012. At that time **lme4**'s code underwent major changes in terms of internal representation of the objects representing fitted models. Hence, the developers of **lme4** decided to make available the snapshot version of **lme4**, under the name of **lme4.0**, containing the functionalities preceding the changes. It is these dynamics of the development of the code of **lme4** and **lme4.0** which prompted us to focus more on **nlme**. However, it is expected that **lme4.0** will not undergo any major modifications, either. Given that it offers interesting tools for fitting LMMs, we decided to include a presentation of it in our book. The presentation should also be of help for **lme4** users. In particular, the major part of the **lme4.0** syntax used in the book should also be applicable to **lme4**.

An important feature that distinguishes R from many other existing statistical software packages implementing LMMs is that it incorporates several concepts of an *object-oriented* (O-O) programming, such as *classes* of *objects* and *methods* operating on those classes. There are two O-O systems that have been implemented in R, namely, S3 and S4. They incorporate the O-O concepts to a different degree, with S3 being a less formal and S4 being a more stringent implementation. In both systems, the O-O concepts are implemented by defining special type of functions called *generic* functions. When such a function is applied to an object, it dispatches an appropriate *method* based on object's class. The system S3 has been used in the package **nlme**, while S4 has been used in the package **lme4.0**.

The O-O programming approach is very attractive in the context of statistical modeling because models can often be broken down into separable (autonomous) components such as data, mean structure, variance function, etc. Moreover, components defined for one type of model can also be used as building blocks for a different type of model.

1.3 The Structure of the Book

As it was mentioned in the previous section, an inherent feature of the O-O programming approach is that concepts and methods used for simpler objects or models are applicable to the more complex ones. For this reason, in our book we opted for an incremental build-up of the knowledge about the implementation of LMMs in the functions from packages **nlme** and **lme4.0**. In particular, in the first step, we decided to introduce theoretical concepts and their practical implementation in the R code in the context of simpler classes of LMs, like the classical linear regression model. The concepts are then carried over to more advanced classes of models, including LMMs. This step-by-step approach offers a couple of advantages. First, we believe that it makes the exposition of the theory and R tools for LMMs simpler and clearer. In particular, the presentation of the key concepts in the context of a simpler model makes them easier to explain and become familiar with. Second, the step-by-step approach is helpful in the use of other R packages, which rely on classes of objects defined in the **nlme** and/or **lme4.0** packages.

As a result of this conceptual approach, we divided our book into four parts. Part I contains the introduction to the datasets used in the book. Parts II, III, and IV focus on different classes of LMs of increasing complexity. The structure of the three parts is, to a large extent, similar. First, a review of the main concepts and theory of a particular class of models is presented. Special attention is paid to the presentation of the link between similar concepts used for different classes. Then, the details of how to implement the particular class of models in the packages **nlme** and/or **lme4.0** are described. The idea is to present the key concepts in the context of simpler models, in order to enhance the understanding of them and facilitate their use for the more complex models. Finally, in each part, the particular class of LMs and the corresponding R tools are illustrated by analyzing real-life datasets.

In a bit more detail, the contents of the four parts are as follows:

Chapter 2 of Part I contains a description of four case studies, which are used to illustrate various classes of LMs and of the corresponding R tools. Chapter 3 contains results of exploratory analyses of the datasets. The results are used in later chapters to support model-based analyses. Note that one of the case studies, the Age-Related Macular Degeneration (ARMD) clinical trial, is used repeatedly for the illustration of all classes of LMs. We believe that in this way the differences between the models concerning, e.g., the underlying assumptions, may become easier to appreciate.

Part II focuses on LMs for independent observations. In Chap. 4, we recall the main concepts of the theory of the classical LMs with homoscedastic residual errors. Then, in Chap. 5, we present the tools available in R to fit such models. This allows us to present the fundamental concepts used in R for statistical model building, like *model formula*, *model frame*, etc. The concepts are briefly illustrated in Chap. 6 using the data from the ARMD trial.

Subsequently, we turn our attention to models with heteroscedastic residual errors. In Chap. 7, we review the basic elements of the theory. Chapter 8 presents the function `gls()` from the package **nlme**, which can be used to fit the models. In particular, the important concept of the *variance function* is introduced in the chapter. The use of the function `gls()` is illustrated using data from the ARMD trial in Chap. 9.

In Part III, we consider general LMs, i.e., LMs for correlated observations. In Chap. 10, we recall the basic elements of the theory of the models. In particular, we explain how the concepts used in the theory of the LMs with heteroscedastic residual errors for independent observations, presented in Chap. 7, are extended to the case of models for correlated observations. In Chap. 11, we describe additional features of the function `gls()`, which allow its use for fitting general LMs. In particular, we introduce the key concept of the *correlation structure*. The use of the function `gls()` is illustrated in Chap. 12 using the data from the ARMD trial.

Finally, Part IV is devoted to LMMs. Chapter 13 reviews the fundamental elements of the theory of LMMs. In the presentation, we demonstrate the links between the concepts used in the theory of LMMs with those developed in the theory of general LMs (Chap. 10). We believe that, by pointing to the links, the exposition of the fundamentals of the LMM theory becomes more transparent and easier to follow.

In Chap. 14, we describe the features of the function `lme()` from the package **nlme**. This function is the primary tool in the package used to fit LMMs. In particular, we describe in detail the representation of positive-definite matrices, which are instrumental in the implementation of the routines that allow fitting LMMs. Note that the concepts of the variance function and correlation structure, introduced in Chaps. 8 and 11, respectively, are also important for the understanding of the use of the function `lme()`.

In Chap. 15, we present the capabilities of the function `lmer()` from the package **lme4.0**. In many aspects, the function is used similarly to `lme()`, but there are important differences, which we discuss. The basic capabilities of both of the functions are illustrated by application of LMMs to the analysis of the ARMD trial data in Chap. 16. More details on the use of the function `lme()` are provided in Chaps. 17, 18, and 19, in which we apply LMMs to analyze the data from the progressive resistance training (PRT) study, the study of instructional improvement (SII), and the Flemish Community Attainment-Targets (FCAT) study, respectively. Finally, in Chap. 20, we present somewhat more advanced material on the additional R tools for LMMs, including the methods for power calculations, influence diagnostics, and a new class of positive-definite matrices. The latter can be used to construct LMMs with random effects having a variance–covariance matrix defined as a Kronecker product of two or more matrices. Note that the newly defined class is used in the analysis presented in Chap. 17.

Table 1.1 summarizes the successive classes of LMs, described in our book, together with the concepts introduced in the context of the particular class. The classes are identified by the assumptions made about the random part of the model.

Our book contains 67 figures, 46 tables, and 187 panels with R code.

Table 1.1 Classes of linear models with the corresponding components (building blocks) presented in the book. The R classes refer to the package **nlme**

Linear model			Model component	
Class (residual errors)	Theory	Syntax	Name	R class
Homoscedastic, indep.	Ch. 4	Ch. 5	Data	*data.frame*
			Mean structure	*formula*
Heteroscedastic, indep.	Ch. 7	Ch. 8	Variance structure	*varFunc*
Correlated	Ch. 10	Ch. 11	Correlation structure	*corStruct*
Mixed effects (LMM)	Ch. 13	Ch. 14	Random-effects structure	*reStruct*

Finally, we would like to outline the scope of the contents of the book:

- The book is aimed primarily at providing explanations and help with respect to the tools available in R for fitting LMMs. Thus, we do not provide a comprehensive account of the methodology of LMMs. Instead, we limit ourselves to the main concepts and techniques, which have been implemented in the functions `lme()` and `lmer()` from the packages **nlme** and **lme4.0**, respectively, and which are important to the understanding of the use of the functions. A detailed exposition of the methodology of LMMs can be found in books by, e.g., Searle et al. (1992), Davidian and Giltinan (1995), Vonesh and Chinchilli (1997), Pinheiro and Bates (2000), Verbeke and Molenberghs (2000), Demidenko (2004), Fitzmaurice et al. (2004), or West et al. (2007).
- In our exposition of methodology, we focus on the likelihood-based estimation methods, as they are primarily used in `lme()` and `lmer()`. Thus, we do not discuss, e.g., Bayesian approaches to the estimation of LMMs.
- We describe the use of various functions, which are available in the packages **nlme** and **lme4.0**, in sufficient detail. In our presentation, we focus on the main, or most often used, arguments of the functions. For a detailed description of all of the arguments, we refer the readers to R's help system.
- It is worth keeping in mind that, in many instances, the same task can be performed in R in several different ways. To some extent, the choice between the different methods is a matter of individual preference. In our description of the R code, we present methods, which we find to be the most useful. If alternative solutions are possible, we may mention them, but we are not aiming to be exhaustive.
- The analyses of the case studies aim principally at illustrating various linear models and the possibility of fitting the models in R. While we try to conduct as meaningful analyses as possible, they are not necessarily performed in the most optimal way with respect to, e.g., the model-building strategy. Thus, their results should not be treated as our contribution to the subject-matter discussion related to the examples. However, whenever possible or useful, we make an attempt to provide quantitative and/or qualitative interpretation of the results. We also try to formulate practical recommendations or guidance regarding model-building strategies, model diagnostics, etc. As mentioned earlier, however, the book is not meant to serve as a complete monograph on statistical modeling. Thus, we limit ourselves to providing recommendations or guidance for the topics which appear to be of interest in the context of the analyzed case studies.

1.4 Technical Notes

The book is aimed at helping readers in fitting LMMs in R. We do assume that the reader has a basic knowledge of R. An introduction to R can be found in the book by Dalgaard (2008). A more advanced exposition is presented by Venables and Ripley (2010).

To allow readers to apply the R code presented in the book, we have created the R package **nlmeU**. The package contains all the datasets and the code that we used in the text. It also includes additional R functions, which we have developed.

We tried to use short lines of the R code to keep matters simple, transparent, and easy to generalize. To facilitate locating the code, we placed it in *panels*. The panels are numbered consecutively in each chapter and referred to, e.g., as R2.3, where "2" gives the number of the chapter and "3" is the consecutive number of the panel within the chapter. Each panel was given a caption explaining the contents. In some cases, the contents of a panel were logically split into different subpanels. The subpanels are then marked by consecutive letters and referred to by adding the appropriate letter to panel's number, e.g., R2.3a or R2.3b. Tables and figures are numbered in a similar fashion.

Only in rare instances were a few lines of R code introduced directly into the text. In all these cases (as in the examples given later in this section), the code was written using the `true type` font and placed in separate lines marked with ">", mimicking R's command-window style.

To limit the volume of the output presented in the panels, in some cases we skipped a part of it. These interventions are indicated by the "... [*snip*]" string. Also, long lines in the output were truncated and extra characters were replaced with the "..." string.

The R functions are referred to in the text as `function()`, e.g., `lme()`. Functions' arguments and objects are marked using the same font, e.g., `argument` and `object`. For the R classes, we use italic, e.g., the *lme* class.

For the proper execution of the R code used in the book, the following packages are required: **lattice**, **lme4.0**, **nlme**, **Matrix**, **plyr**, **reshape**, **RLRsim**, **splines**, and **WWGbook**. Additionally, **nlmeU** is needed. Packages **lattice**, **nlme**, **Matrix**, and **splines** come with basic distribution of R and do not need to be installed. The remaining packages can be installed using the following code:

```
> pckgs  <-
+  c("lme4.0", "nlmeU", "plyr", "reshape", "RLRsim", "WWGbook",
+    "ellipse")
> install.packages(pckgs)
```

There are additional utility functions, namely, `Sweave()` (Leisch, 2002) and `xtable()` in **utils** and **xtable** (Dahl, 2009) packages, respectively, which are not needed to execute the code presented in the book, but which were extensively used by us when preparing this manuscript.

It is worth noting that there are functions that bear the same name in the packages **nlme** and **lme4.0**, but which have different definitions. To avoid unintentional masking of the functions, the packages should not be attached simultaneously. Instead, it is recommended to switch between the packages. For example, when using **nlme** in a hypothetical R session, we attach the package by using the `library()` or `require()` functions and execute statements as needed. Then, before switching to **lme4.0**, it is mandatory to detach the **nlme** package by using the `detach()` function. We also note that the `conflicts()` function, included for illustration below, is very useful to identify names' conflicts:

```
> library(nlme)                 # Attach package
> conflicts(detail = TRUE)      # Identifies names' conflicts
 ... statements omitted
> detach(package:nlme)          # Detach package
```

A similar approach should be applied when using the package **lme4.0**:

```
> library(lme4.0)
 ... statements omitted
> detach(package:lme4.0)
> detach(package:Matrix)      # Recommended
```

Note that detaching **Matrix** is less critical, but recommended.

In the examples presented above, we refer to the packages **nlme** and **lme4.0**. However, to avoid unintentional masking of objects, the same strategy may also be necessary for other packages, which may cause function names' conflicts.

When creating figures, we used `"CMRoman"` and `"CMSans"` Computer Modern font families available in **cmrutils** package. These fonts are based on the CM-Super and CMSYASE fonts (Murrell and Ripley, 2006). The full syntax needed to create figures presented in the book is often extensive. In many cases, we decided to present a shortened version of the code. A full version is available in the **nlmeU** package.

Finally, the R scripts in our book were executed by using `R version 2.15.0 (2012-03-30)` under the Windows 7 operating system. We used the following global options:

```
> options(width = 65, digits = 5, show.signif.stars = FALSE)
```

Chapter 2
Case Studies

2.1 Introduction

In this chapter, we introduce the case studies that will be used to illustrate the models and R code described in the book.

The case studies come from different application domains; however, they share a few features. For instance, in all of them the study and/or sampling design generates the observations that are *grouped* according to the levels of one or more *grouping* factors. More specifically, the levels of grouping factors, i.e., subjects, schools, etc., are assumed to be randomly selected from a population being studied. This means that observations within a particular group are likely to be correlated. The correlation should be taken into account in the analysis. Also, in each case there is one (or more) continuous measurement, which is treated as the dependent variable in the models considered in this book.

In particular, we consider the following datasets:

- *Age-Related Macular Degeneration (ARMD) Trial*: A clinical trial comparing several doses of interferon-α and placebo in patients with ARMD. Visual acuity of patients participating in the trial was measured at baseline and at four post-randomization timepoints. The resulting data are an example of *longitudinal data* with observations grouped by subjects. We describe the related datasets in more detail in Sect. 2.2.
- *Progressive Resistance Training (PRT) Trial*: A clinical trial comparing low- and high-intensity training for improving the muscle power in elderly people. For each participant, characteristics of two types of muscle fibers were measured at two occasions, pre- and post-training. The resulting data are an example of *clustered* data, with observations grouped by subjects. We present more detailed information about the dataset in Sect. 2.3.
- *Study of Instructional Improvement (SII)*: An educational study aimed at assessing improvement in mathematics grades of first-grade pupils, as compared to their kindergarten achievements. It included pupils from randomly selected

A. Gałecki and T. Burzykowski, *Linear Mixed-Effects Models Using R: A Step-by-Step Approach*, Springer Texts in Statistics, DOI 10.1007/978-1-4614-3900-4_2,

classes in randomly selected elementary schools. The dataset is an example of *hierarchical* data, with observations (pupils' scores) grouped within classes, which are themselves grouped in schools. We refer to Sect. 2.4 for more details about the data.

- *Flemish Community Attainment-Targets (FCAT) Study*: An educational study, in which elementary school graduates were evaluated with respect to reading comprehension in Dutch. Pupils from randomly selected schools were assessed for a set of nine attainment targets. The dataset is an example of *grouped* data, for which the grouping factors are *crossed*. We describe the dataset in more detail in Sect. 2.5.

The data from the ARMD study will be used throughout the book to illustrate various classes of LMs and corresponding R tools. The remaining case studies will be used in Part IV only, to illustrate R functions for fitting LMMs.

For each of the aforementioned case studies there is one or more datasets included into the package **nlmeU**, which accompanies this book. In the next sections of this chapter, we use the R syntax to describe the contents of these datasets. Results of exploratory analyses of the case studies are presented in Chap. 3. Note that, unlike in the other parts of the book, we are not discussing the code in much detail, as the data-processing functionalities are not the main focus of our book. The readers interested in the functionalities are referred to the monograph by Dalgaard (2008).

The R language is not particularly suited for data entry. Typically, researchers use raw data created using other software. Data are then stored in external files, e.g., in the .csv format, read into R, and prepared for the analysis. To emulate this situation, we assume, for the purpose of this chapter, that the data are stored in a .csv-format file in the "C:\temp" directory.

2.2 Age-Related Macular Degeneration Trial

The ARMD data arise from a randomized multi-center clinical trial comparing an experimental treatment (interferon-α) *versus* placebo for patients diagnosed with ARMD. The full results of this trial have been reported by Pharmacological Therapy for Macular Degeneration Study Group (1997). We focus on the comparison between placebo and the highest dose (6 million units daily) of interferon-α.

Patients with macular degeneration progressively lose vision. In the trial, visual acuity of each of 240 patients was assessed at baseline and at four post-randomization timepoints, i.e., at 4, 12, 24, and 52 weeks. Visual acuity was evaluated based on patient's ability to read lines of letters on standardized vision charts. The charts display lines of five letters of decreasing size, which the patient must read from top (largest letters) to bottom (smallest letters). Each line with at least four letters correctly read is called one "line of vision." In our analyses, we will focus on the visual acuity defined as the total number of *letters* correctly read.

Another possible approach would be to consider visual acuity measured by the number of *lines* correctly read. Note that the two approaches are closely linked, as each line of vision contains five letters.

It follows that, for each of 240 patients, we have *longitudinal data* in the form of up to five visual acuity measurements collected at different, but common to all patients, timepoints. These data will be useful to illustrate the use of LMMs for continuous, longitudinal data. We will also use them to present other classes of LMs considered in our book.

2.2.1 Raw Data

We assume that the raw ARMD data are stored in the "C:\temp" directory in a .csv-format file named `armd240.data.csv`. In what follows, we also assume that our goal is to verify the contents of the data and prepare them for analysis in R.

In Panel R2.1, the data are loaded into R using the `read.csv()` function and are stored in the data frame object `armd240.data`. Note that this data frame is not included in the **nlmeU** package.

The number of rows (records) and columns (variables) in the object `armd240.data` is obtained using the function `dim()`. The data frame contains 240 observations and 9 variables. The names of the variables are displayed using the `names()` function. All the variables are of class *integer*. By applying the function `str()`, we get a summary description of variables in the `armd240.data` data. In particular, for each variable, we get its class and a listing of the first few values.

The variable `subject` contains patients' identifiers. Treatment identifiers are contained in the variable `treat`. Variables `visual0`, `visual4`, `visual12`, `visual24`, and `visual52` store visual acuity measurements obtained at baseline and week 4, 12, 24, and 52, respectively. Variables `lesion` and `line0` contain additional information, which will not be used for analysis in our book.

Finally, at the bottom of Panel R2.1, we list the first three rows of the data frame `armd240.data` with the help of the `head()` function. To avoid splitting lines of the output and to make the latter more transparent, we shorten variables' names using the `abbreviate()` function. After printing the contents of the first three rows and before proceeding further, we reinstate the original names. Note that we apply a similar sequence of R commands in many other R panels across the book to simplify the displayed output.

Based on the output, we note that the data frame contains one record for each patient. The record includes all information obtained for the patient. In particular, each record contains five variables with visual acuity measurements, which are, essentially, of the same format. This type of data storage, with one record per subject, is called the "wide" format. An alternative is the "long" format with multiple records per subject. We will discuss the formats in the next section.

R2.1 *ARMD Trial*: Loading raw data from a .csv-format file into the `armd240.data` object and checking their contents

```
> dataDir <- file.path("C:", "temp")       # Data directory
> fp <-                                    # File path
+   file.path(dataDir, "armd240.data.csv")
> armd240.data <-                          # Read data
+   read.csv(fp, header = TRUE)
> dim(armd240.data)                        # No. of rows and cols
  [1] 240   9
> (nms <- names(armd240.data))             # Variables' names
  [1] "subject"  "treat"    "lesion"   "line0"    "visual0"
  [6] "visual4"  "visual12" "visual24" "visual52"
> unique(sapply(armd240.data, class))      # Variables' classes
  [1] "integer"
> str(armd240.data)                        # Data structure
  'data.frame':   240 obs. of  9 variables:
   $ subject : int  1 2 3 4 5 6 7 8 9 10 ...
   $ treat   : int  2 2 1 1 2 2 1 1 2 1 ...
   $ lesion  : int  3 1 4 2 1 3 1 3 2 1 ...
   $ line0   : int  12 13 8 13 14 12 13 8 12 10 ...
   $ visual0 : int  59 65 40 67 70 59 64 39 59 49 ...
   $ visual4 : int  55 70 40 64 NA 53 68 37 58 51 ...
   $ visual12: int  45 65 37 64 NA 52 74 43 49 71 ...
   $ visual24: int  NA 65 17 64 NA 53 72 37 54 71 ...
   $ visual52: int  NA 55 NA 68 NA 42 65 37 58 NA ...
> names(armd240.data) <- abbreviate(nms)   # Variables' names shortened
> head(armd240.data, 3)                    # First 3 records
    sbjc tret lesn lin0 vsl0 vsl4 vs12 vs24 vs52
  1    1    2    3   12   59   55   45   NA   NA
  2    2    2    1   13   65   70   65   65   55
  3    3    1    4    8   40   40   37   17   NA
> names(armd240.data) <- nms               # Variables' names reinstated
```

2.2.2 *Data for Analysis*

In this section, we describe auxiliary data frames, namely, `armd.wide`, `armd0`, and `armd`, which were derived from `armd240.data` for the purpose of analyses of the ARMD data that will be presented later in the book. The data frames are included in the package **nlmeU**. In what follows, we present the structure, contents, and for illustration purposes, how the data were created.

2.2.2.1 *Data in the "Wide" Format: The Data Frame* `armd.wide`

Panel R2.2 presents the structure and the contents of the `armd.wide` data frame.

Note that the data are loaded into R using the `data()` function, without the need for attaching the package **nlmeU**. The data frame contains 10 variables. In particular, it includes variables `visual0`, `visual4`, `visual12`, `visual24`, `visual52`, `lesion`, and `line0`, which are exactly the same as those in the `armd240.data`. In contrast to the `armd240.data` data frame, it contains three factors: `subject`, `treat.f`, and `miss.pat`. The first two contain patient's identifier and treatment. They are constructed from the corresponding numeric variables available in `armd240.data`. The factor `miss.pat` is a new variable and contains a missing-pattern identifier, i.e., a character string that indicates which of the four post-randomization measurements of visual acuity are missing for a particular patient. The missing values are marked by `X`. Thus, for instance, for the patient with the `subject` identifier equal to 1, the pattern is equal to `--XX`, because there is no information about visual acuity at weeks 24 and 52. On the other hand, for the patient with the `subject` identifier equal to 6, there are no missing visual acuity

R2.2 *ARMD Trial*: The structure and contents of data frame `armd.wide` stored in the "wide" format

```
> data(armd.wide, package = "nlmeU")          # armd.wide loaded
> str(armd.wide)                              # Structure of data
  'data.frame':   240 obs. of  10 variables:
   $ subject : Factor w/ 240 levels "1","2","3","4",..: 1 2 3 4 5 6 ...
  ...     [snip]
   $ treat.f : Factor w/ 2 levels "Placebo","Active": 2 2 1 1 2 2 1 ...
   $ miss.pat: Factor w/ 9 levels "----","---X",..: 4 1 2 1 9 1 1 1 ...
> head(armd.wide)                             # First few records
    subject lesion line0 visual0 visual4 visual12 visual24
  1       1      3    12      59      55       45       NA
  ...     [snip]
  6       6      3    12      59      53       52       53
    visual52 treat.f miss.pat
  1       NA  Active     --XX
  ...     [snip]
  6       42  Active     ----
> (facs <- sapply(armd.wide, is.factor))      # Factors indicated
   subject   lesion    line0  visual0  visual4 visual12 visual24
      TRUE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE
  visual52  treat.f miss.pat
     FALSE     TRUE     TRUE
> names(facs[facs == TRUE])                   # Factor names displayed
  [1] "subject"  "treat.f"  "miss.pat"
```

measurements, and hence the value of the `miss.pat` factor is equal to `----`. At the bottom of Panel R2.2, we demonstrate how to extract the names of the factors from a data frame.

Panel R2.3 presents the syntax used to create factors `treat.f` and `miss.pat` in the `armd.wide` data frame. The former is constructed in Panel R2.3a from the variable `treat` from the data frame `armd240.data` using the function `factor()`. The factor `treat.f` has two levels, `Placebo` and `Active`, which correspond to the values of 1 and 2, respectively, of `treat`.

The factor `miss.pat` is constructed in Panel R2.3b with the help of the function `missPat()` included in the **nlmeU** package. The function returns a character vector of length equal to the number of rows of the matrix created by column-wise concatenation of the vectors given as arguments to the function. The elements of the resulting vector indicate the occurrence of missing values in the rows of the matrix. In particular, the elements are character strings of the length equal to the number of the columns (vectors). As shown in Panel R2.2, the strings contain characters "`-`" and "`X`", where the former indicates a nonmissing value in the corresponding column of the matrix, while the latter indicates a missing value. Thus, application

R2.3 *ARMD Trial*: Construction of factors `treat.f` and `miss.pat` in the data frame `armd.wide`. The data frame `armd240.data` was created in Panel R2.1

(a) *Factor `treat.f`*

```
> attach(armd240.data)                    # Attach data
> treat.f <-                              # Factor created
+   factor(treat, labels = c("Placebo", "Active"))
> levels(treat.f)                         # (1) Placebo, (2) Active
  [1] "Placebo" "Active"
> str(treat.f)
   Factor w/ 2 levels "Placebo","Active": 2 2 1 1 2 2 1 1 2 1 ...
```

(b) *Factor `misspat`*

```
> miss.pat <-                             # Missing patterns
+   nlmeU:::missPat(visual4, visual12, visual24, visual52)
> length(miss.pat)                        # Vector length
  [1] 240
> mode(miss.pat)                          # Vector mode
  [1] "character"
> miss.pat                                # Vector contents
    [1] "--XX" "----" "---X" "----" "XXXX" "----" "----" "----"
  ...     [snip]
  [233] "----" "----" "----" "----" "----" "----" "----" "----"
> detach(armd240.data)                   # Detach armd240.data
```

of the function to variables visual4, visual12, visual24, and visual52 from the data frame armd240.data results in a character vector of length 240 with strings containing four characters as the elements. The elements of the resulting miss.pat vector indicate that, for instance, for the first patient in the data frame armd240.data visual acuity measurements at week 24 and 52 were missing, while for the fifth patient, no visual acuity measurements were obtained at any post-randomization visit.

Note that we used the nlmeU:::missPat() syntax, which allowed us to invoke the missPat() function without attaching the **nlmeU** package.

2.2.2.2 Data in the "Long" Format: The Data Frame armd0

In addition to the armd.wide data stored in the "wide" format, we will need data in the "longitudinal" (or "long") format. In the latter format, for each patient, there are multiple records containing visual acuity measurements for separate visits. An example of data in "long" format is stored in the data frame armd0. It was obtained from the armd.wide data using functions melt() and cast() from the package **reshape** (Wickham, 2007).

Panel R2.4 presents the contents and structure of the data frame armd0. The data frame includes eight variables and 1,107 records. The contents of variables subject, treat.f, and miss.pat are the same as in armd.wide, while visual0 contains the value of the visual acuity measurement at baseline. Note that the values of these four variables are repeated across the multiple records corresponding to a particular patient. On the other hand, the records differ with respect to the values of variables time.f, time, tp, and visual. The first three of those four variables are different forms of an indicator of the visit time, while visual contains the value of the visual acuity measurement at the particular visit. We note that having three variables representing time visits is not mandatory, but we created them to simplify the syntax used for analyses in later chapters.

The numerical variable time provides the actual week, at which a particular visual acuity measurement was taken. The variable time.f is a corresponding ordered factor, with levels Baseline, 4wks, 12wks, 24wks, and 52wks. Finally, tp is a numerical variable, which indicates the position of the particular measurement visit in the sequence of the five possible measurements. Thus, for instance, tp=0 for the baseline measurement and tp=4 for the fourth post-randomization measurement at week 52.

Interestingly enough, visual acuity measures taken at baseline are stored both in visual0 and in selected rows of the visual variables. This structure will prove useful when creating the armd data frame containing rows with post-randomization visual acuity measures, while keeping baseline values.

The "long" format is preferable for storing longitudinal data over the "wide" format. We note that storing of the visual acuity measurements in the data frame armd.wide requires the use of six variables, i.e., subject and the five variables containing the values of the measurements. On the other hand, storing the same

R2.4 *ARMD Trial*: The structure and contents of the data frame `armd0` stored in the "long" format

```
> data(armd0, package = "nlmeU")        # From nlmeU package
> dim(armd0)                            # No. of rows and cols
  [1] 1107    8
> head(armd0)                           # First six records
    subject treat.f visual0 miss.pat   time.f time visual tp
  1       1  Active      59     --XX Baseline    0     59  0
  2       1  Active      59     --XX     4wks    4     55  1
  3       1  Active      59     --XX    12wks   12     45  2
  4       2  Active      65     ---- Baseline    0     65  0
  5       2  Active      65     ----     4wks    4     70  1
  6       2  Active      65     ----    12wks   12     65  2
> names(armd0)                          # Variables' names
  [1] "subject"  "treat.f"  "visual0"  "miss.pat" "time.f"
  [6] "time"     "visual"   "tp"
> str(armd0)                            # Data structure
  'data.frame': 1107 obs. of  8 variables:
   $ subject : Factor w/ 240 levels "1","2","3","4",..: 1 1 1 2 2 2 ...
   $ treat.f : Factor w/ 2 levels "Placebo","Active": 2 2 2 2 2 2 ...
   $ visual0 : int  59 59 59 65 65 65 65 65 40 40 ...
   $ miss.pat: Factor w/ 9 levels "----","---X",..: 4 4 4 1 1 1 1 1 ...
   $ time.f  : Ord.factor w/ 5 levels "Baseline"<"4wks"<..: 1 2 3 1 ...
   $ time    : num  0 4 12 0 4 12 24 52 0 4 ...
   $ visual  : int  59 55 45 65 70 65 65 55 40 40 ...
   $ tp      : num  0 1 2 0 1 2 3 4 0 1 ...
```

information in the data frame `armd0` requires only three variables, i.e., `subject`, `time`, and `visual`. Of course, this is achieved at the cost of including more rows in the `armd0` data frame, i.e., 1,107, as compared to 240 records in `armd.wide`.

We also note that variables, with values invariant within subjects, such as `treat.f`, `visual0`, are referred to as *time-fixed*. In contrast, `time`, `tp`, and `visual` are called *time-varying*. This distinction will have important implications for the specification of the models and interpretation of the results.

2.2.2.3 Subsetting Data in the "Long" Format: The Data Frame `armd`

Data frame `armd` was also stored in a "long" format and was created from the `armd0` data frame by omitting records corresponding to the baseline visual acuity measurements.

Panel R2.5 presents the syntax used to create the data frame `armd`. In particular, the function `subset()` is used to remove the baseline measurements, by selecting

R2.5 *ARMD Trial*: Creation of the data frame `armd` from `armd0`

```
> auxDt <- subset(armd0, time > 0)          # Post-baseline measures
> dim(auxDt)                                # No. of rows & cols
  [1] 867   8
> levels(auxDt$time.f)                      # Levels of treat.f
  [1] "Baseline" "4wks"     "12wks"    "24wks"    "52wks"
> armd <- droplevels(auxDt)                 # Drop unused levels
> levels(armd$time.f)                       # Baseline level dropped
  [1] "4wks"  "12wks" "24wks" "52wks"
> armd <-                                   # Data modified
+   within(armd,
+          {
+            contrasts(time.f) <-           # Contrasts assigned
+              contr.poly(4, scores = c(4, 12, 24, 52))
+          })
> head(armd)                                # First six records
    subject treat.f visual0 miss.pat time.f time visual tp
  2       1  Active      59      --XX   4wks    4     55  1
  3       1  Active      59      --XX  12wks   12     45  2
  5       2  Active      65      ----   4wks    4     70  1
  6       2  Active      65      ----  12wks   12     65  2
  7       2  Active      65      ----  24wks   24     65  3
  8       2  Active      65      ----  52wks   52     55  4
```

only the records, for which `time>0`, from the object `armd0`. By removing the baseline measurements, we reduce the number of records from 1,107 (see Panel R2.4) to 867.

While subsetting the data, care needs to be taken regarding the levels of the `time.f` and, potentially, other factors. In the data frame `armd0`, the factor had five levels. In Panel R2.5, we extract the factor `time.f` from the auxiliary data frame `auxDt`. Note that, in the data frame, the level `Baseline` is not used in any of the rows. For many functions in R it would not be a problem, but sometimes the presence of an unused level in the definition of a factor may lead to unexpected results. Therefore, it is prudent to drop the unused level from the definition of the `time.f` factor, by applying the function `droplevels()`. It is worth noting that, using the `droplevels()` function, the number of levels of the factors `subject` and `miss.pat` is also affected (not shown).

After modifying the aforementioned factors, we store the resulting data in the data frame `armd`. We also assign orthogonal polynomial contrasts to the factor `time.f` using syntax of the form "`contrasts(`*factor*`)<-`*contr.function*". We will revisit the issue of assigning contrasts to a factor in Panel R5.9 (Sect. 5.3.2).

The display of the first six records of `armd` in Panel R2.5 confirms that the data do not include the records corresponding to the baseline measurements of visual acuity.

Of course, the information about the values of the measurements is still available in the variable `visual0`.

Both data frames `armd0` and `armd`, introduced in this section, are stored in "long" format. The `armd0` will be primarily used for exploratory data analyses (Sect. 3.2). On the other hand, `armd` will be the primary data frame used for the analyses throughout the entire book.

2.3 Progressive Resistance Training Study

The PRT data originate from a randomized trial aimed for devising evidence-based methods for improving and measuring the mobility and muscle power of elderly men and women in the 70+ age category (Claflin *et al.*, 2011). The working hypothesis was that a 12-week program of PRT would increase: (a) the power output of the overall musculature associated with movements of the ankles, knees, and hips; (b) the cross-sectional area and the force and power of permeabilized single fibers obtained from the *vastus lateralis* muscle; and (c) the ability of young and elderly men and women to safely arrest standardized falls. The training consisted of repeated leg extensions by shortening contractions of the leg extensor muscles against a resistance that was increased as the subject trained using a specially designed apparatus.

In the trial, healthy young (21–30 years) and older (65–80 years) male and female subjects were randomized between a "high" and "low" intensity of a 12-week PRT intervention. Randomization was stratified by age group (young or old) and sex. In total, the dataset used in our book includes 63 subjects.

For each subject, multiple measurements characterizing two types of muscle fibers were obtained before and after the 12-week PRT. The resulting data are thus an example of *clustered* data. In particular, the measurements for a given characteristic of muscle fibers for each subject correspond to a 2×2 factorial design, with fiber type (1, 2) and occasion (pre-training, post-training) as the two design factors, which has important implications for the data analysis (Chap. 17).

2.3.1 Raw Data

We assume that subjects' characteristics and experimental measurements are contained in external files named `prt.subjects.data.csv` and `prt.fiber.data.csv`, respectively.

In Panel R2.6, we present the syntax for loading and inspecting the two datasets. As can be seen from the output presented in Panel R2.6a, the file `prt.subjects.data.csv` contains information about 63 subjects, with one record per subject. It includes one character variable and five numeric variables, three of which are integer-valued. The variable `id` contains subjects' identifiers, `gender`

2.5.2 *Data for Analysis*

In the analyses presented later in the book, we will be using the data frame `fcat`, which is constructed based on the data frame `crossreg.data`. In Panel R2.16, we present the syntax used to create the `fcat` data and to investigate data grouping structure. First, in Panel R2.16a, we replace the variables `id` and `target` by corresponding factors. For the factor `target`, the labels given in parentheses indicate the number of items for a particular target.

In Panel R2.16b, we cross-tabulate the factors `id` and `target` and store the resulting table in the object `tab1`. Given the large number of levels of the factor `id`, it is difficult to verify the values of the counts for all cells of the table. By applying the function `all()` to the result of the evaluation of expression `tab1>0`, we check that all counts of the table are nonzero. On the other hand, with the help of the `range()` function, we verify that all the counts are equal to 1. This indicates that, in the data frame `fcat`, the levels of the factor `target` are crossed with the levels of the factor `id`. Moreover, the data are balanced, in the sense that there is the same number of observations, namely, one observation for each combination of the levels of the two factors. Because all counts in the table are greater than zero, we can say that the factors are *fully crossed*.

2.6 Chapter Summary

In this chapter, we introduced four case studies, which will be used for illustration of LMs described in our book.

We started the presentation of each case study by describing study design and considering that raw data are stored in a .csv file. We chose this approach in an attempt to emulate a common situation of using external data files when analyzing data using R. In the next step, we prepared the data for analysis by creating the necessary variables and, in particular, factors. Including factors as part of data is a feature fairly unique to R. It affects how a given variable is treated by graphical and modeling functions. This approach is recommended, but not obligatory. In particular, creating factors can be deferred to a later time, when, e.g., *model formula* is specified. We will revisit this issue in Chap. 5.

The data frames, corresponding to the four case studies, are included in the package **nlmeU**. As with other packages, the list of datasets available in the package can be obtained by using the `data(package = "nlmeU")` command. For the reader's convenience, the datasets are summarized in Table 2.2. The table includes the information about the R-session panels, which present the syntax used to create the data frames, grouping factors, and number of rows and variables.

The four case studies introduced in this chapter are conducted by employing different study designs. All of them lead to grouped data defined by one or more nested or crossed grouping factors. The preferable way of storing this type of data

R2.16 *FCAT Study*: Construction and inspection of the contents of the data frame `fcat`. The data frame `crossreg.data` was created in Panel R2.14

(a) *Construction of the data frame* `fcat`

```
> nItms <- c(4, 6, 8, 5, 9, 6, 8, 6, 5)        # See Table 2.1
> (lbls <- paste("T", 1:9, "(", nItms, ")", sep = ""))
 [1] "T1(4)" "T2(6)" "T3(8)" "T4(5)" "T5(9)" "T6(6)" "T7(8)"
 [8] "T8(6)" "T9(5)"
> fcat <-
+   within(crossreg.data,
+          {
+           id <- factor(id)
+           target <- factor(target, labels = lbls)
+          })
> str(fcat)
 'data.frame':   4851 obs. of  3 variables:
  $ target: Factor w/ 9 levels "T1(4)","T2(6)",..: 1 2 3 4 5 6 7 8 ...
  $ id    : Factor w/ 539 levels "1","2","3","4",..: 1 1 1 1 1 1 1 ...
  $ scorec: int  4 6 4 1 7 6 6 5 5 3 ...
```

(b) *Investigation of the data grouping structure*

```
> (tab1 <- xtabs(~ id + target, data = fcat))   # id by target table
     target
id    T1(4) T2(6) T3(8) T4(5) T5(9) T6(6) T7(8) T8(6) T9(5)
  1       1     1     1     1     1     1     1     1     1
  2       1     1     1     1     1     1     1     1     1
...     [snip]
  539     1     1     1     1     1     1     1     1     1
> all(tab1 > 0)                                 # All counts > 0?
 [1] TRUE
> range(tab1)                                   # Range of counts
 [1] 1 1
```

is to use the "long" format with multiple records per subject. Although this term is borrowed from the literature pertaining to longitudinal data, it is also used in the context of other grouped data. Below, we describe the key features of the data in each study.

In the ARMD trial, the `armd.wide` data frame stores data in the "wide" format. Data frames `armd` and `armd0` store data in the "long" format and reflect the hierarchical data structure defined by a single grouping factor, namely, `subject`. For this reason, and following the naming convention used in the **nlme** package, we will refer to the data structure in our book as data with a *single level of grouping*. Note that, more traditionally, these data are referred to as *two-level data* (West et al., 2007).

Table 2.2 Data frames available in the **nlmeU** package

Study	Data frame	R-panel	Grouping factors	Rows × vars
ARMD Trial	`armd.wide`	R2.2	*None*	240 × 10
	`armd0`	R2.4	`subject`	1,107 ×8
	`armd`	R2.5	`subject`	867 × 8
PRT Trial	`prt.subjects`	R2.7a	*None*	63 × 5
	`prt.fiber`	R2.7b	`id`	2,471 ×5
	`prt`	R2.8	`id`	2,471 ×9
SII Project	`SIIdata`	R2.10	`classid` nested ...	1,190 ×12
			... in `schoolid`	
FCAT Study	`fcat`	R2.16	`id` crossed ...	4,851 ×3
			... with `target`	

The hierarchical structure of data contained in the data frame `SIIdata` is defined by two (nested) grouping factors, namely, `schoolid` and `classid`. Thus, in our book, this data structure will be referred to as data with *two levels of grouping*.

This naming convention works well for hierarchical data, i.e., for data with nested grouping factors. It is more problematic for structures with crossed factors. This is the case for the FCAT study, in which the data structure is defined by two crossed grouping factors, thus without a particular hierarchy.

As a result of data grouping, variables can be roughly divided into group- and measurement-specific categories. In the context of longitudinal data they are referred to as time-fixed and time-varying variables. The classification of the variables has important implications for the model specification.

To our knowledge, the *groupedData* class, defined in the **nlme** package, appears to be the only attempt to directly associate a hierarchical structure of the data with objects of the *data.frame* class. We do not describe this class in more detail, however, because it has some limitations. Also, its initial importance has diminished substantially over time. In fact, the data hierarchy is most often reflected indirectly by specifying the structure of the model fitted to the data. We will revisit this issue in Parts III and IV of our book.

When introducing the SII case study, we noted that the nested data structure can be specified by using two different approaches, namely, explicit and implicit nesting, depending on the coding of the levels of grouping factors. The choice of the approach is left to the researcher's discretion. The issue has important implications for the specification of LMMs, though, and it will be discussed in Chap. 15.

The different data structures of the cases studies presented in this chapter will allow us to present various aspects of LMMs in Part IV of the book. Additionally, the ARMD dataset will be used in the other parts to illustrate other classes of LMs and related R tools.

The main focus of this chapter was on the presentation of the data frames related to the case studies. In the presentation, we also introduced selected concepts related

to grouped data and R functions, which are useful for data transformation and inspection of the contents of datasets. By necessity, our introduction was very brief and fragmentary; a more in-depth discussion of those and other functions is beyond the scope of our book. The interested readers are referred to, e.g., the book by Dalgaard (2008) for a more thorough explanation of the subject.

Chapter 3
Data Exploration

3.1 Introduction

In this chapter, we present the results of exploratory analyses of the case studies introduced in Chap. 2. The results will serve as a basis for building LMs for the data in the following parts of the book.

While exploring the case-study data, we also illustrate the use of selected functions and graphical tools which are commonly used to perform these tasks. Note, however, that, unlike in the other parts of the book, we are not discussing the functions and tools in much detail. The readers interested in the functionalities are referred to the monograph by Venables and Ripley (2010).

3.2 ARMD Trial: Visual Acuity

In the ARMD data, we are mainly interested in the effect of treatment on the visual acuity measurements. Thus, in Fig. 3.1, we first take a look at the measurements by plotting them against time for several selected patients from both treatment groups. More specifically, we selected every 10th patient from each group.

Based on the plots shown in Fig. 3.1, several observations can be made:

- In general, visual acuity tends to decrease in time. This is in agreement with the remark made in Sect. 2.2 that patients with ARMD progressively lose vision.
- For some patients, a linear decrease of visual acuity over time can be observed, but there are also patients for whom individual profiles strongly deviate from a linear trend.
- Visual acuity measurements adjacent in time are fairly well correlated, with the correlation decreasing with an increasing distance in time.
- Visual acuity at baseline seems to, at least partially, determine the overall level of the post-randomization measurements.
- There are patients for whom several measurements are missing.

A. Gałecki and T. Burzykowski, *Linear Mixed-Effects Models Using R: A Step-by-Step Approach*, Springer Texts in Statistics, DOI 10.1007/978-1-4614-3900-4_3,

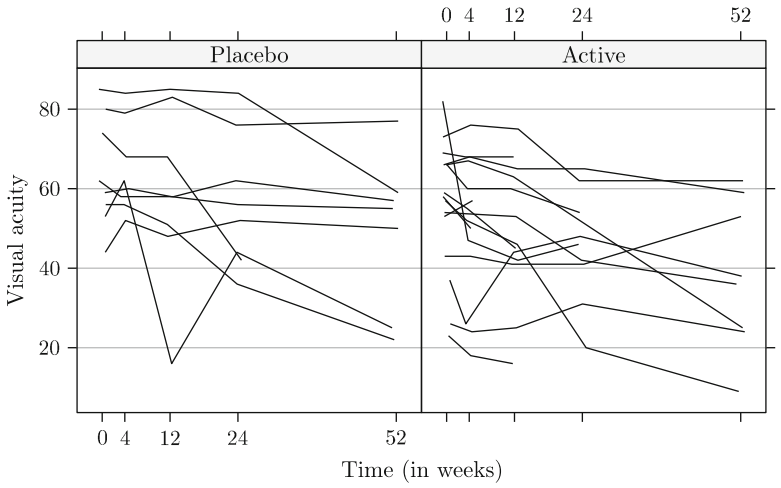

Fig. 3.1 *ARMD Trial*: Visual-acuity profiles for selected patients ("spaghetti plot")

These observations will be taken into account when constructing models for the data.

R3.1 *ARMD Trial*: Syntax for the plot of visual acuity profiles for selected patients in Fig. 3.1

```
> data(armd.wide, armd0, package = "nlmeU")       # Data loaded
> library(lattice)
> armd0.subset <-                                 # Subset
+   subset(armd0, as.numeric(subject) %in% seq(1, 240, 10))
> xy1 <-                                          # Draft plot
+   xyplot(visual ~ jitter(time) | treat.f,
+          groups = subject,
+          data = armd0.subset,
+          type = "l", lty = 1)
> update(xy1,                                     # Fig. 3.1
+        xlab = "Time (in weeks)",
+        ylab = "Visual acuity",
+        grid = "h")
> detach(package:lattice)
```

The syntax used to create Fig. 3.1 is shown in Panel R3.1. First, we load data to be used for exploration from the **nlmeU** package. Note that the code used to create figure employs the function `xyplot()` from the package **lattice** (Sarkar, 2008). The function is applied to the subset of the data frame `armd0` (Sect. 2.2.2). The formula used in the syntax indicates that the variables `visual` and `time` are to be used on the

y- and *x*-axis, respectively. These variables are plotted against each other in separate panels for different values of the `treat.f` factor. Within each panel, data points are grouped for each subject and connected using solid lines. The function `jitter()` is used to add a small amount of noise to the variable `time`, thereby reducing the number of overlapping points.

In the next sections, we explore particular features of the ARMD data in more detail.

3.2.1 Patterns of Missing Data

First, we check the number and patterns of missing visual acuity measurements. Toward this end, we use the data frame `armd.wide`. As mentioned in Sect. 2.2.2, the data frame contains the factor `miss.pat` that indicates which of the four post-randomization measurements are missing for a particular patient. For example, the pattern `--X-` indicates that the only missing measurement was at the third post-randomization timepoint, i.e., at 24 weeks.

In Panel R3.2, we use three different methods to tabulate the number of patients with different levels of the factor `miss.pat`. From the displayed results, we can conclude that, for instance, there were 188 patients for whom all four post-randomization visual acuity measurements were obtained. On the other hand, there were six patients for whom the four measurements were missing.

R3.2 *ARMD Trial*: Inspecting missing-data patterns in the `armd.wide` data for the post-randomization visual acuity measurements using three different methods

```
> table(armd.wide$miss.pat)
  ---- ---X --X- --XX -XX- -XXX X--- X-XX XXXX
   188   24    4    8    1    6    2    1    6
> with(armd.wide, table(miss.pat))
  miss.pat
  ---- ---X --X- --XX -XX- -XXX X--- X-XX XXXX
   188   24    4    8    1    6    2    1    6
> xtabs(~miss.pat, armd.wide)
  miss.pat
  ---- ---X --X- --XX -XX- -XXX X--- X-XX XXXX
   188   24    4    8    1    6    2    1    6
```

It is also worth noting that there are eight $(= 4 + 1 + 2 + 1)$ patients with four different nonmonotone missing-data patterns, i.e., with intermittent missing visual acuity measurements. When modeling data with such patterns, extra care is needed when specifying variance–covariance structures. We will come back to this issue in Sect. 11.4.2.

3.2.2 Mean-Value Profiles

In this section, we investigate the number of missing values and calculate the sample means of visual acuity measurements for different visits and treatment groups. Toward this end, in Panel R3.3, we use the "long"-format data frame `armd0`, which was described in Sect. 2.2.2.

R3.3 *ARMD Trial*: Sample means and medians for visual acuity by time and treatment

(a) *Counts of nonmissing visual acuity measurements*

```
> attach(armd0)
> flst <- list(time.f, treat.f)                    # "By" factors
> (tN <-                                           # Counts
+   tapply(visual, flst,
+          FUN = function(x) length(x[!is.na(x)])))
           Placebo Active
  Baseline     119    121
  4wks         117    114
  12wks        117    110
  24wks        112    102
  52wks        105     90
```

(b) *Sample means and medians of visual acuity measurements*

```
> tMn <- tapply(visual, flst, FUN = mean)          # Sample means
> tMd <- tapply(visual, flst, FUN = median)        # Sample medians
> colnames(res  <- cbind(tN, tMn, tMd))            # Column names
  [1] "Placebo" "Active"  "Placebo" "Active"  "Placebo" "Active"
> nms1 <- rep(c("P", "A"), 3)
> nms2 <- rep(c("n", "Mean", "Mdn"), rep(2, 3))
> colnames(res) <- paste(nms1, nms2, sep = ":")    # New column names
> res
           P:n A:n P:Mean A:Mean P:Mdn A:Mdn
  Baseline 119 121 55.336 54.579  56.0  57.0
  4wks     117 114 53.966 50.912  54.0  52.0
  12wks    117 110 52.872 48.673  53.0  49.5
  24wks    112 102 49.330 45.461  50.5  45.0
  52wks    105  90 44.438 39.100  44.0  37.0
> detach(armd0)
```

To calculate counts of missing values in Panel R3.3a, we use the function `tapply()`. In general, this function is used to apply a selected function to each (nonempty) group of values defined by a unique combination of the levels of one or more factors. In our case, the selected function, specified in the `FUN` argument,

checks the length of the vector created by selecting nonmissing values from the vector passed as an argument to the function. Using the `tapply()` function, we apply it to the variable `visual` within the groups defined by combinations of the levels of factors `time.f` and `treat.f`. As a result, we obtain a matrix with the number of nonmissing visual acuity measurements for each visit and each treatment group. We store the matrix in the object `tN` for further use. The display of the matrix indicates that there were no missing measurements at baseline. On the other hand, at week 4, for instance, there were two and seven missing measurements in the placebo and active-treatment arms, respectively. In general, there are more missing measurements in the active-treatment group.

In Panel R3.3b, we use the function `tapply()` twice to compute the sample means and sample medians of visual acuity measurements for each combination of the levels of factors `time.f` and `treat.f`. We store the results in matrices `tMn` and `tMd`, respectively. We then create the matrix `res` by combining matrices `tN`, `tMn`, and `tMn` by columns. Finally, to improve the legibility of displays, we modify the names of the columns of `res`.

From the display of the matrix `res`, we conclude that, on average, there was very little difference in visual acuity between the two treatment groups at baseline. This is expected in a randomized study. During the course of the study, the mean visual acuity decreased with time in both arms, which confirms the observation made based on the individual profiles presented in Fig. 3.1. It is worth noting that the mean value is consistently higher in the placebo group, which suggests lack of effect of interferon-α.

Figure 3.2 presents box-and-whiskers plots of visual acuity for the five timepoints and the two treatment arms. The syntax to create the figure is shown in Panel R3.4. It uses the function `bwplot()` from the package **lattice**. Note that we first create a draft of the plot, which we subsequently enhance by providing labels for the horizontal axis. In contrast to Fig. 3.1, measurements for all subjects at all timepoints are plotted. A disadvantage of the plot is that it does not reflect the longitudinal structure of the data.

R3.4 *ARMD Trial*: Syntax for the box-and-whiskers plots in Fig. 3.2

```
> library(lattice)
> bw1 <-                                                  # Draft plot
+   bwplot(visual ~ time.f | treat.f,
+          data = armd0)
> xlims <- c("Base", "4\nwks", "12\nwks", "24\nwks", "52\nwks")
> update(bw1, xlim = xlims, pch = "|")                    # Final plot
> detach(package:lattice)
```

The box-and-whiskers plots illustrate the patterns implied by the sample means and medians, presented in Panel R3.3b. The decrease of the mean values in time is clearly seen for both treatment groups. It is more pronounced for the active-treatment arm. As there was a slightly higher dropout in that arm, a possible

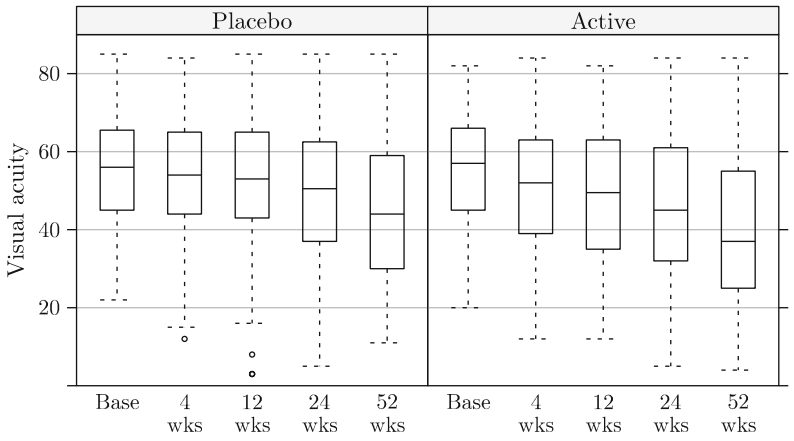

Fig. 3.2 *ARMD Trial*: Box-and-whiskers plots for visual acuity by treatment and time

explanation could be that patients whose visual acuity improved dropped out of the study. In such case, a faster progression of the disease in that treatment arm would be observed.

To check this possibility, we take a look at Fig. 3.3. It shows the mean values of visual acuity for patients with different monotone missing-data patterns. In addition, the number of subjects for each pattern is also given. We note that the number of subjects for the patterns with a larger number of missing values tends to be smaller. Note that, to save space, we do not present the syntax used to create the figure, as it is fairly complex.

The mean profiles, shown in Fig. 3.3, consistently decrease for the majority of the patterns. In general, they do not suggest an improvement in visual acuity before the drop off. Thus, they do not support the aforementioned explanation of a faster decrease of the mean visual acuity in the active-treatment arm.

In Panel R3.5, we present the syntax to investigate the number and form of monotone missing-data patterns for visual acuity. In particular, in Panel R3.5a, we create the data frame `armd.wide.mnt`, which contains data only for patients with monotone patterns. There are 232 such patients in total. Note that, despite the fact that some patterns are not present in the data frame `armd.wide.mnt`, they are still recognized as valid levels of the factor `miss.pat`. This might cause problems when using some R functions. Similarly to Panel R2.5, we could use the `droplevels()` function to remove the unused levels of the `miss.pat` variable. Instead, in Panel R3.5b, we modify the levels of the factor `miss.pat` in the `armd.wide.mnt` data with the help of the function `factor()`. Note that, instead of using the `levels` argument of the function, we could have used the argument `exclude` while indicating the levels to be excluded from the definition of the `miss.pat` factor.

Finally, in Panel R3.5c, we use the function `tapply()` to obtain a matrix containing the number of patients for each monotone missing-data pattern and for

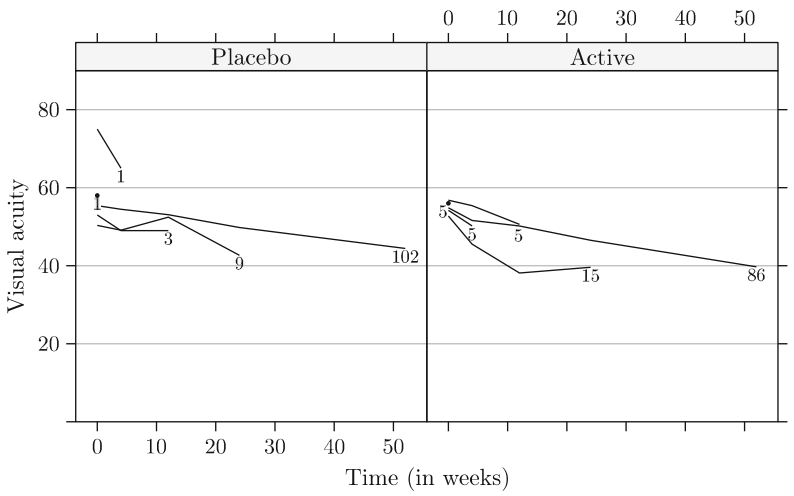

Fig. 3.3 *ARMD Trial*: Mean visual acuity profiles by missing pattern and treatment (monotone missing-data patterns only)

each treatment arm. The displayed results indicate that the mean-value profiles for missing-data patterns with a larger number of missing values, shown in Fig. 3.3, are based on measurements for a small number of patients. Thus, the variability of these profiles is larger than for the patterns with a smaller number of missing values. Therefore, Fig. 3.3 should be interpreted with caution.

3.2.3 Sample Variances and Correlations of Visual Acuity Measurements

Figure 3.4 shows a scatterplot matrix for the visual acuity measurements for those patients, for whom all post-randomization measurements are available. Scatterplots for corresponding pairs of variables are given below the diagonal. The size of the font for correlation coefficients reported above the diagonal is proportional to its value. We do not present the syntax for constructing the figure, as it is fairly complex. It can be observed that the measurements adjacent in time are strongly correlated. The correlation decreases with an increasing time gap. Worth noting is the fact that there is a substantial positive correlation between visual acuity at baseline and at the other post-randomization measurements. Thus, baseline values might be used to explain the overall variability of the post-randomization observations. This agrees with the observation made based on Fig. 3.1. It is worth noting that a scatterplot matrix of the type shown in Fig. 3.4 may not work well for longitudinal data with irregular time intervals.

R3.5 *ARMD Trial*: The number of patients by treatment and missing-data pattern (monotone patterns only)

(a) *Subset of the data with monotone missing-data patterns*

```
> mnt.pat<-                                    # Monotone patterns
+   c("----", "---X", "--XX", "-XXX", "XXXX")
> armd.wide.mnt <-                             # Data subset
+   subset(armd.wide, miss.pat %in% mnt.pat)
> dim(armd.wide.mnt)                           # Number of rows and cols
  [1] 232  10
> levels(armd.wide.mnt$miss.pat)               # Some levels not needed
  [1] "----" "---X" "--X-" "--XX" "-XX-" "-XXX" "X---" "X-XX"
  [9] "XXXX"
```

(b) *Removing unused levels from the* `miss.pat` *factor*

```
> armd.wide.mnt1 <-
+   within(armd.wide.mnt,
+          {
+            miss.pat <- factor(miss.pat, levels=mnt.pat)
+          })
> levels(armd.wide.mnt1$miss.pat)
  [1] "----" "---X" "--XX" "-XXX" "XXXX"
```

(c) *The number of patients with different monotone missing-data patterns*

```
> with(armd.wide.mnt1,
+      {
+        fl  <- list(treat.f, miss.pat)       # List of "by" factors
+        tapply(subject, fl, FUN=function(x) length(x[!is.na(x)]))
+      })
          ---- ---X --XX -XXX XXXX
  Placebo  102    9    3    1    1
  Active    86   15    5    5    5
```

In Panel R3.6, we provide the estimates of the variance–covariance and correlation matrices for visual acuity measurements. Toward this end, we create the data frame `visual.x` from `armd.wide` by selecting only the five variables containing the measurements. We then apply functions `var()` and `cor()` to estimate the variance–covariance matrix and the correlation matrix, respectively. Note that, for both functions, we specify the argument `use = "complete.obs"`, which selects only those rows of the data frame `visual.x` that do not contain any missing values. In this way, the estimated matrices are assured to be

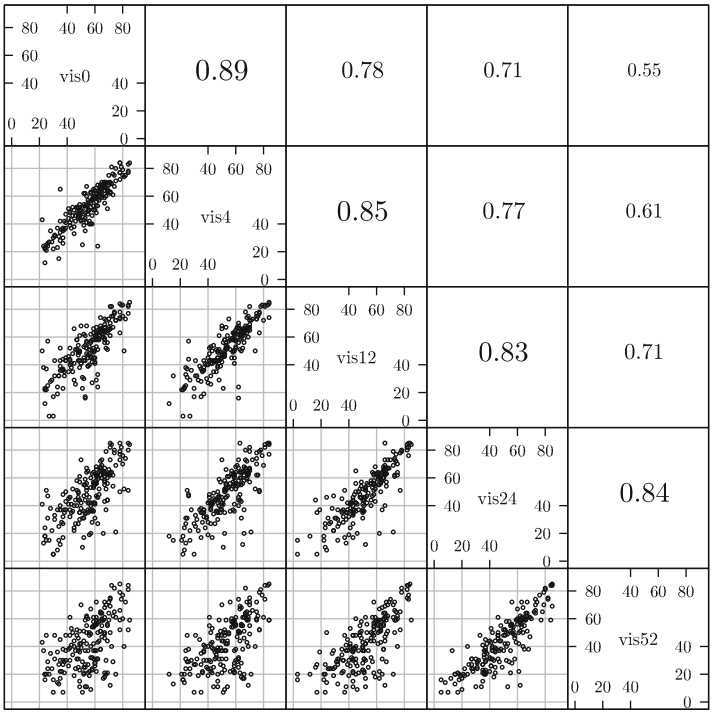

Fig. 3.4 *ARMD Trial*: Scatterplot matrix for visual acuity measurements. Scatterplots (*below diagonal*) and correlation coefficients (*above diagonal*) for complete cases only ($n = 188$)

positive semidefinite. An alternative (not shown) would be to specify `use = "pairwise.complete.obs"`. In that case, the elements of the matrices would be estimated using data for all patients with complete observations for the particular pair of visual acuity measurements. This could result in estimates of variance–covariance or correlation matrices, which might not be positive semidefinite.

The variance–covariance matrix for visual acuity measurements is stored in the `varx` matrix. It indicates an increase of the variance of visual acuity measurements obtained at later timepoints. The estimated correlation matrix suggests a moderate to strong correlation of the measurements. We also observe that the correlation clearly decreases with the time gap, as already concluded from Fig. 3.4.

At the bottom of Panel R3.6, we demonstrate how to extract the diagonal elements of the matrix `varx` using the `diag()` function. We also present the use of the function `cov2cor()` to compute a correlation matrix corresponding to the variance–covariance. Note that we do not display the result of the use of the function, as it is exactly the same as the one obtained for the function `cor()`, already shown in Panel R3.6.

R3.6 *ARMD Trial*: Variance–covariance and correlation matrices for visual acuity measurements for complete cases only ($n = 188$)

```
> visual.x <- subset(armd.wide, select = c(visual0:visual52))
> (varx <- var(visual.x, use = "complete.obs")) # Var-cov mtx
           visual0 visual4 visual12 visual24 visual52
  visual0   220.31  206.71   196.24   193.31   152.71
  visual4   206.71  246.22   224.79   221.27   179.23
  visual12  196.24  224.79   286.21   257.77   222.68
  visual24  193.31  221.27   257.77   334.45   285.23
  visual52  152.71  179.23   222.68   285.23   347.43
> print(cor(visual.x, use = "complete.obs"),    # Corr mtx
+       digits = 2)
           visual0 visual4 visual12 visual24 visual52
  visual0     1.00    0.89     0.78     0.71     0.55
  visual4     0.89    1.00     0.85     0.77     0.61
  visual12    0.78    0.85     1.00     0.83     0.71
  visual24    0.71    0.77     0.83     1.00     0.84
  visual52    0.55    0.61     0.71     0.84     1.00
> diag(varx)                  # Var-cov diagonal elements
   visual0  visual4 visual12 visual24 visual52
    220.31   246.22   286.21   334.45   347.43
> cov2cor(varx)               # Corr mtx (alternative way)
  ...      [snip]
```

3.3 PRT Study: Muscle Fiber Specific Force

In the PRT study, we are primarily interested in the effect of the intensity of the training on the muscle fiber specific force, measurements of which are contained in the variable `spec.fo` of the `prt` data frame (Sect. 2.3.2). In some analyses, we will also investigate the effect on the measurements of the isometric force, which are stored in the variable `iso.fo`.

First, however, we take a look at the information about subjects' characteristics, stored in the data frame `prt.subjects` (see Sect. 2.3.2). In Panel R3.7, we use the function `tapply()` to obtain summary statistics for the variable `bmi` for separate levels of the `prt.f` factor. The statistics are computed with the help of the `summary()` function. The displayed values of the statistics do not indicate any substantial differences in the distribution of BMI between subjects assigned to the low- or high-intensity training. Given that the assignment was randomized, this result is anticipated.

For illustration purposes, we also obtain summary statistics for all variables in the `prt.subjects` data frame, except for `id`, with the help of the function `by()`. The function splits the data frame according to the levels of the factor `prt.f` and applies the function `summary()` to the two data frames resulting from the split. As a result, we obtain summary statistics for variables `prt.f`, `age.f`, `sex.f`, and `bmi`

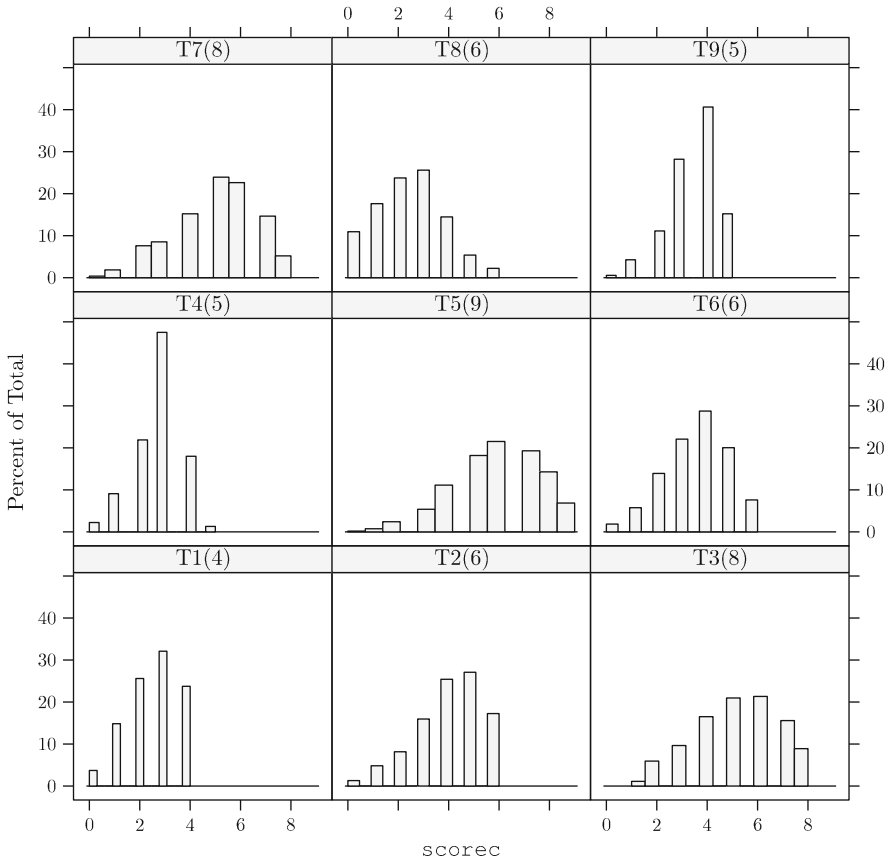

Fig. 3.11 Histograms of individual total scores for different attainment targets

3.6 Chapter Summary

In this chapter, we presented exploratory analyses of the four case studies introduced in Chap. 2. The results of the analyses will be used in the next parts of our book to build models for the case studies.

In parallel to the presentation of the results of the exploratory analyses, we introduced a range of R tools, which are useful for such analyses. For instance, functions `cast()` and `melt()` from the package **reshape** are very useful in transforming data involving aggregated summaries. The importance of using graphical displays is also worth highlighting. Toward this aim, the tools available in packages **graphics** (R Development Core Team, 2010) and **lattice** (Sarkar, 2008) are very helpful. The former package implements traditional graphical displays, whereas the latter offers displays based on a grid-graphics system (Murrell, 2005).

Due to space limitations, our presentation of the tools was neither exhaustive nor detailed. However, we hope that the syntax and its short description, which were provided in the chapter, can help the reader in finding appropriate methods applicable to the particular problem at hand.

Part IV
Linear Mixed-Effects Models

Chapter 13
Linear Mixed-Effects Model

13.1 Introduction

In Chap. 10, we presented models with fixed effects for correlated data. They are examples of population-averaged models, because their mean-structure parameters can be interpreted as effects of covariates on the mean value of the dependent variable in the entire population. The association between the observations in a dataset was a result of a grouping of the observations sharing the same level of a grouping factor(s). An example of grouped data is longitudinal data, with multiple measurements collected over time for an individual. This is an example of data with a single level of grouping (Sect. 10.2): measurements are grouped at the level of an individual. Such a hierarchy is present in the ARMD data (Sect. 2.2 and Chaps. 6, 9, and 12), with multiple visual acuity measurements available for individual patients. Another example of data with a single level of grouping is meta-analysis data, with patients grouped within clinical trials.

An example of data with a multilevel hierarchy is student's scores. Scores, e.g., for the same course across several years, are grouped for a student, students are grouped into classes, classes into schools, schools within districts, etc. Consequently, the total variability of the scores can be seen as resulting from their variability within students, between students, between classes within the same school, between schools in the same district, etc. Such a structure is present in the data for the Instructional Improvement Study, presented in Sect. 2.4.

Note that, because of grouping, a complex association structure of the observed data can be anticipated. For instance, we can expect correlation not only between the scores for an individual student, but also between scores from different students from the same class or between scores for different students from the same school. As argued in Part III of the book, the correlations should be taken into account in the analysis of the data.

In this chapter, we consider the analysis of continuous, hierarchical data using a different class of models, namely, LMMs. They allow taking into account the correlation of observations contained in a dataset. Moreover, they allow us to

A. Gałecki and T. Burzykowski, *Linear Mixed-Effects Models Using R: A Step-by-Step Approach*, Springer Texts in Statistics, DOI 10.1007/978-1-4614-3900-4_13,

effectively partition overall variation of the dependent variable into components corresponding to different levels of data hierarchy. The models are examples of *subject-specific* models, because they include subject-specific coefficients.

In this chapter, we describe the specification of LMMs for hierarchical data. We build upon the concepts introduced in Parts II and III of the book. We provide only essential theoretical information, linked to the concepts and methods used in R. For a more detailed exposition of the theory of LMMs, the reader is referred to the monographs by, e.g., Searle et al. (1992), Davidian and Giltinan (1995), Vonesh and Chinchilli (1997), Pinheiro and Bates (2000), Verbeke and Molenberghs (2000), Demidenko (2004), Fitzmaurice et al. (2004), or West et al. (2007).

This chapter is structured as follows. In Sects. 13.2–13.4, we describe the formulation of the model. Sections 13.5–13.7 are devoted to, respectively, the estimation approaches, diagnostic tools, and inferential methods used for the LMMs, in which the (conditional) residual variance-covariance matrix is independent of the mean value. This is the most common type of LMMs used in practice. In Sect. 13.8, we focus on the LMMs, in which the (conditional) residual variance-covariance matrix depends of the mean value. Section 13.9 summarizes the contents of this chapter and offers some general concluding comments.

In our presentation, we focus on the formulation and methods for LMMs applicable to data with a single level of grouping, with N groups indexed by $i = 1, \ldots, N$, each containing n_i observations. The extension of the formulation to multilevel grouped data is presented at the end of Sect. 13.2 and in Sect. 13.8.

13.2 The Classical Linear Mixed-Effects Model

In this section, we describe specification of the classical LMMs in their general form. Essentially, the formulation corresponds to the one proposed in the classical paper by Laird and Ware (1982).

In particular, in Sect. 13.2.1, we provide model specification at a particular level of grouping factor, while in Sect. 13.2.2 we describe a specification for all data. Extension of the classical LMM is presented in Sect. 13.3. More detailed aspects of the classical and extended model specification are discussed in Sect. 13.4.

13.2.1 Specification at a Level of a Grouping Factor

For hierarchical data with a single level of grouping, we can formulate the classical LMM at a given level of a grouping factor as follows:

$$\mathbf{y}_i = \boldsymbol{X}_i\boldsymbol{\beta} + \boldsymbol{Z}_i\mathbf{b}_i + \boldsymbol{\varepsilon}_i, \tag{13.1}$$

where $\mathbf{y}_i$, $\boldsymbol{X}_i$, $\boldsymbol{\beta}$, and $\boldsymbol{\varepsilon}_i$ are the vector of continuous responses, the design matrix, and the vector of residual errors for group i, specified in (10.2) and (10.3), respectively, while $\boldsymbol{Z}_i$ and $\mathbf{b}_i$ are the matrix of covariates and the corresponding vector of random effects:

$$\boldsymbol{Z}_i \equiv \begin{pmatrix} z_{i1}^{(1)} & z_{i1}^{(2)} & \dots & z_{i1}^{(q)} \\ \vdots & \vdots & \ddots & \vdots \\ z_{in_i}^{(1)} & z_{in_i}^{(2)} & \dots & z_{in_i}^{(q)} \end{pmatrix} = \left(\mathbf{z}_i^{(1)} \; \mathbf{z}_i^{(2)} \; \dots \; \mathbf{z}_i^{(q)} \right), \quad \mathbf{b}_i \equiv \begin{pmatrix} b_{i1} \\ \vdots \\ b_{iq} \end{pmatrix}. \tag{13.2}$$

Similar to the design matrix $\boldsymbol{X}_i$, the matrix $\boldsymbol{Z}_i$ contains known values of q covariates, with corresponding unobservable effects $\mathbf{b}_i$. Moreover,

$$\mathbf{b}_i \sim \mathcal{N}_q(\mathbf{0}, \boldsymbol{\mathcal{D}}), \quad \boldsymbol{\varepsilon}_i \sim \mathcal{N}_{n_i}(\mathbf{0}, \boldsymbol{\mathcal{R}}_i), \quad \text{with} \quad \mathbf{b}_i \perp \boldsymbol{\varepsilon}_i, \tag{13.3}$$

i.e., the residual errors $\boldsymbol{\varepsilon}_i$ for the same group are independent of the random effects $\mathbf{b}_i$. This particular assumption plays the key role in distinguishing a classical LMM from an extended LMM. In addition, we assume that vectors of random effects and residual errors for different groups are independent of each other, i.e., $\mathbf{b}_i$ is independent of $\boldsymbol{\varepsilon}_{i'}$ for $i \neq i'$.

We also specify that

$$\boldsymbol{\mathcal{D}} = \sigma^2 \boldsymbol{D} \quad \text{and} \quad \boldsymbol{\mathcal{R}}_i = \sigma^2 \boldsymbol{R}_i, \tag{13.4}$$

where σ^2 is an unknown scale parameter. In general, we will assume that $\boldsymbol{\mathcal{D}}$ and $\boldsymbol{\mathcal{R}}_i$ are positive-definite, unless stated otherwise.

The representation (13.4), in its general form, is not unique. To make it identifiable, similar to the case of the LM for correlated data (Sect. 10.3), we will specify the structure of the matrix $\boldsymbol{R}_i$ in terms of a set of parameters for a variance function and a correlation matrix (Sect. 13.4.2). The specification will imply constraints on $\boldsymbol{R}_i$ making (13.4) identifiable.

In addition to the fixed-effects parameters $\boldsymbol{\beta}$ for the covariates used in constructing the design matrix $\boldsymbol{X}_i$, model (13.1) includes two random components: the within-group residual errors $\boldsymbol{\varepsilon}_i$ and the random effects $\mathbf{b}_i$ for the covariates included in the matrix $\boldsymbol{Z}_i$. The presence of fixed and random effects of known variables gives rise to the name of the model.

In many cases, the (random) effects included in $\mathbf{b}_i$ have corresponding (fixed) effects, contained in $\boldsymbol{\beta}$. Consequently, the matrix $\boldsymbol{Z}_i$ is often created by selecting a subset of appropriate columns of the matrix $\boldsymbol{X}_i$. In such a situation, it is said that the corresponding fixed and random effects are "coupled."

Model (13.1)–(13.4) is commonly referred to as a *two-stage* model (Davidian and Giltinan 1995) or *two-level* model (West et al. 2007). However, some authors, e.g., Pinheiro and Bates (2000), call it a *single-level* LMM, because it applies to a data hierarchy defined by a single level of grouping. In what follows, we will use the latter terminology, as it is reflected in the nomenclature used in R.

The classical LMM, defined in (13.1)–(13.4), can be adapted to multilevel grouped data. For instance, a model for data with two levels of grouping, with observations grouped into N first-level groups (indexed by $i = 1, \ldots, N$), each with n_i second-level (sub-)groups (indexed by $j = 1, \ldots, n_i$) containing n_{ij} observations, can be written as

$$\mathbf{y}_{ij} = \boldsymbol{X}_{ij}\boldsymbol{\beta} + \boldsymbol{Z}_{1,ij}\mathbf{b}_i + \boldsymbol{Z}_{2,ij}\mathbf{b}_{ij} + \boldsymbol{\varepsilon}_{ij}, \tag{13.5}$$

with

$$\mathbf{b}_i \sim \mathcal{N}_{q_1}(\mathbf{0}, \boldsymbol{\mathcal{D}}_1), \quad \mathbf{b}_{ij} \sim \mathcal{N}_{q_2}(\mathbf{0}, \boldsymbol{\mathcal{D}}_2), \quad \text{and} \quad \boldsymbol{\varepsilon}_{ij} \sim \mathcal{N}_{n_{ij}}(\mathbf{0}, \boldsymbol{\mathcal{R}}_{ij}),$$

where the random vectors $\mathbf{b}_i$, $\mathbf{b}_{ij}$, and $\boldsymbol{\varepsilon}_{ij}$ are independent of each other. In model (13.5), $\mathbf{b}_i$ are the random effects associated with the first-level groups, while $\mathbf{b}_{ij}$ are the random effects, independent of the first-level random effects, associated with the second-level groups. Design matrices $\boldsymbol{Z}_{1,ij}$ and $\boldsymbol{Z}_{2,ij}$ can, but do not have to be, identical. Following Pinheiro and Bates (2000), this model can be referred to as a *two-level* LMM.

13.2.2 Specification for All Data

In this section, we briefly describe the specification of the single-level LMM, given by (13.1)–(13.4), for all data. Generalization to multilevel LMMs is obvious, though notationally more complex.

Let $\mathbf{y} \equiv (\mathbf{y}_1', \mathbf{y}_2', \ldots, \mathbf{y}_N')'$ be the vector containing all $n = \sum_{i=1}^{N} n_i$ observed values of the dependent variable. Similarly, let $\mathbf{b} \equiv (\mathbf{b}_1', \mathbf{b}_2', \ldots, \mathbf{b}_N')'$ and $\boldsymbol{\varepsilon} \equiv (\boldsymbol{\varepsilon}_1', \boldsymbol{\varepsilon}_2', \ldots, \boldsymbol{\varepsilon}_N')'$ be the vectors containing all Nq random effects and n residual errors, respectively. Define matrices

$$\boldsymbol{X} \equiv \begin{bmatrix} \boldsymbol{X}_1 \\ \boldsymbol{X}_2 \\ \vdots \\ \boldsymbol{X}_N \end{bmatrix} \quad \text{and} \quad \boldsymbol{Z} \equiv \begin{bmatrix} \boldsymbol{Z}_1 & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{Z}_2 & \ldots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & \boldsymbol{Z}_N \end{bmatrix}, \tag{13.6}$$

where $\mathbf{0}$ denotes a matrix with all elements equal to 0. Note that, to simplify notation, we do not indicate the dimensions of the matrices, as they can be deduced from (13.2). Overall, $\boldsymbol{X}$ is of dimension $n \times p$, while $\boldsymbol{Z}$ is of dimension $n \times Nq$.

Models (13.1)–(13.4) can then be written for *all* data as follows:

$$\mathbf{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{Z}\mathbf{b} + \boldsymbol{\varepsilon}, \tag{13.7}$$

with

$$\mathbf{b} \sim \mathcal{N}_{Nq}(\mathbf{0}, \sigma^2\mathbf{D}) \quad \text{and} \quad \boldsymbol{\varepsilon} \sim \mathcal{N}_n(\mathbf{0}, \sigma^2\boldsymbol{R}), \tag{13.8}$$

where

$$\mathbf{D} \equiv \boldsymbol{I}_N \otimes \boldsymbol{D} = \begin{bmatrix} \boldsymbol{D} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{D} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \boldsymbol{D} \end{bmatrix}, \quad \boldsymbol{R} \equiv \begin{bmatrix} \boldsymbol{R}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \boldsymbol{R}_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \boldsymbol{R}_N \end{bmatrix}, \tag{13.9}$$

with $\otimes$ denoting the (right) Kronecker product.

It is worth noting that the particular, block-diagonal form of matrices $\boldsymbol{Z}$, $\mathbf{D}$, and $\boldsymbol{R}$, given in (13.6), (13.8), and (13.9), respectively, results from the fact that the single-level LMM, defined by (13.1)–(13.4), assumes a particular hierarchy of data and random effects, as explicitly shown in (13.3). In particular, the model assumes that random effects for different groups, defined by levels of a particular factor, are independent. Informally, we can describe the hierarchy as generated by grouping factors, with one being *nested* within the other.

It is possible, however, to formulate random-effects models by using the representation (13.7) with non-block-diagonal matrices $\boldsymbol{Z}$, $\mathbf{D}$, and $\boldsymbol{R}$. This is the case, for instance, of models with *crossed* random effects. We will describe this type of models in Chap. 15.

13.3 The Extended Linear Mixed-Effects Model

In some cases, the assumption that the residual errors $\boldsymbol{\varepsilon}_i$ are independent of the random effects $\mathbf{b}_i$, as specified in (13.3), may be too restrictive. For instance, as is done in the mean-variance models, we might postulate that the variance of the residual errors depends on the subject-specific mean value. If we relax the assumption, we obtain an extended LMM. The model is specified by using (13.1)–(13.2) and replacing (13.3) by

$$\mathbf{b}_i \sim \mathcal{N}_q(\mathbf{0}, \boldsymbol{\mathcal{D}}), \quad \text{and} \quad \boldsymbol{\varepsilon}_i \mid \mathbf{b}_i \sim \mathcal{N}_{n_i}(\mathbf{0}, \boldsymbol{\mathcal{R}}_i), \tag{13.10}$$

with $\boldsymbol{\mathcal{D}}$ and $\boldsymbol{\mathcal{R}}_i$ decomposed further as in (13.4). We will refer to the above specification as a *hierarchical* specification.

Note that, if we assume that $\boldsymbol{\varepsilon}_i$ in (13.10) is independent of the random effects $\mathbf{b}_i$, then we obtain the classical LMM, specified by (13.1)–(13.4). Thus, the extended LMM allows for a more general modeling approach, as compared to the classical LMM.

A hierarchical specification of a *two-level*, extended LMM, corresponding to (13.5), would amount to assuming that

$$\mathbf{b}_i \sim \mathcal{N}_{q_1}(\mathbf{0}, \boldsymbol{\mathcal{D}}_1), \quad \mathbf{b}_{ij} \mid \mathbf{b_i} \sim \mathcal{N}_{q_2}(\mathbf{0}, \boldsymbol{\mathcal{D}}_2), \quad \text{and} \quad \boldsymbol{\varepsilon}_{ij} \mid \mathbf{b}_i, \mathbf{b}_{ij} \sim \mathcal{N}_{n_{ij}}(\mathbf{0}, \boldsymbol{\mathcal{R}}_{ij}).$$

13.4 Distributions Defined by the y and b Random Variables

Both the classical (Sect. 13.2) and extended (Sect. 13.3) LMMs introduce two continuous random variables **b** and **y**. They are described by two probability density functions, which play essential role in defining LMMs. The first one is an unconditional distribution of (unobserved) random effects **b**, defined by (13.8). The second one is a conditional distribution of the (random) dependent variable **y**, assuming that random effects are known. In the next two sections, we provide a more detailed description of the two distributions, which completes the model specification for the classical and extended LMMs. In Sect. 13.4.3, we will introduce additional auxiliary distributions related to **y** and **b** random variables.

13.4.1 Unconditional Distribution of Random Effects

The unconditional distribution $f_b(\mathbf{b}_i)$ of the random effects $\mathbf{b}_i$, defined by (13.3), is a multivariate normal distribution with zero mean and variance-covariance matrix $\boldsymbol{\mathcal{D}}$. Taking into account (13.4), we write

$$\boldsymbol{\mathcal{D}}(\sigma^2, \boldsymbol{\theta}_D) = \sigma^2 \boldsymbol{D}(\boldsymbol{\theta}_D), \tag{13.11}$$

where $\boldsymbol{\theta}_D$ is a vector of parameters, which represent the (scaled by σ^2) variances and covariances of the elements of $\mathbf{b}_i$.

Note that, according to (13.11), the matrix $\boldsymbol{D}$, used to define the variance-covariance matrix of random effects $\mathbf{b}_i$, is parameterized using a vector of parameters $\boldsymbol{\theta}_D$. In many cases, it is assumed that any two elements of the vector $\mathbf{b}_i$ can be correlated and there are no restrictions imposed on the matrix $\boldsymbol{\mathcal{D}}$, except that it is positive-definite and symmetric. In this case, $\boldsymbol{\mathcal{D}}$ has a general structure of a positive-definite matrix, with $q(q+1)/2$ distinct elements corresponding to q variances and $q(q-1)/2$ covariances of the random effects included in $\mathbf{b}_i$. Consequently, $\boldsymbol{\theta}_D$ contains $q(q+1)/2$ distinct parameters. Although q is typically small, estimating all of the parameters may be difficult if, e.g., the sample size n is limited. In such a situation, a simplified structure of the matrix $\boldsymbol{\mathcal{D}}$ can be chosen. For instance, a diagonal form can be assumed, which is equivalent to assuming that all elements of the vector $\mathbf{b}_i$ are independent. Plausibility of the assumption will depend on the data at hand. In this case, $\boldsymbol{\theta}_D$ contains only q distinct parameters.

13.4.2 Conditional Distribution of **y** *Given the Random Effects*

Note that, from (13.1) to (13.4), it follows that, for the classical LMMs, the conditional distribution, $f_{y|b}(\mathbf{y}_i|\mathbf{b}_i)$, of $\mathbf{y}_i$ given $\mathbf{b}_i$ is multivariate normal, with the mean and variance defined as:

$$\mathrm{E}(\mathbf{y}_i|\mathbf{b}_i) \equiv \boldsymbol{\mu}_i = \boldsymbol{X}_i\boldsymbol{\beta} + \boldsymbol{Z}_i\mathbf{b}_i \tag{13.12}$$

$$\mathrm{Var}(\mathbf{y}_i|\mathbf{b}_i) = \sigma^2\boldsymbol{R}_i, \tag{13.13}$$

with $\boldsymbol{\mu}_i \equiv (\mu_{i1}, \ldots, \mu_{i,n_i})'$ and

$$\mathrm{E}(y_{ij}|\mathbf{b}_i) \equiv \mu_{ij} = \mathbf{x}'_{ij}\boldsymbol{\beta} + \mathbf{z}'_{ij}\mathbf{b}_i, \tag{13.14}$$

where $\mathbf{x}_{ij} \equiv (x_{ij}^{(1)}, \ldots, x_{ij}^{(p)})'$ and $\mathbf{z}_{ij} \equiv (z_{ij}^{(1)}, \ldots, z_{ij}^{(q)})'$ are column vectors, which contain the values of predictors X and Z for the j-th observation from the i-th group. Thus, *conditionally* on the (unknown) values of the random effects $\mathbf{b}_i$, the mean value of the dependent-variable vector $\mathbf{y}_i$ is defined by a linear combination of the vectors of the X- and Z-covariates included, as columns, in the group-specific design matrices $\boldsymbol{X}_i$ and $\boldsymbol{Z}_i$, corresponding to the fixed effects $\boldsymbol{\beta}$ and random effects $\mathbf{b}_i$, respectively. Moreover, the conditional variance-covariance matrix of $\mathbf{y}_i$ is equal to the variance-covariance matrix of the residual errors $\boldsymbol{\varepsilon}_i$.

In their most general form, LMMs are not identifiable, because of the nonuniqueness of the representation (13.4) and because they potentially contain too many unknown parameters (see similar comments in Sects. 7.2 and 10.3). To make them identifiable, similarly to the matrix $\mathcal{D}$, we can consider representing elements of $\mathcal{R}_i$ as functions of a limited set of parameters $\boldsymbol{\theta}_R$, distinct from $\boldsymbol{\theta}_D$.

For the matrix $\mathcal{R}_i$, similarly to the approach described in Sect. 10.3 and implemented in R, we could consider the decomposition, given by (10.10), and combine it with the use of variance functions (Sects. 7.2.2 and 7.3.1) and correlation structures (Sect. 10.3.2). Thus, $\mathcal{R}_i$ would become parsimoniously parameterized in terms of a set of parameters of a variance function and a correlation structure. In this way not only the number of parameters of the model would be reduced, but the representation (13.4) would become identifiable.

To allow for the use of variance functions from the <δ, μ>- and <μ>-groups (Sect. 7.3.1), we follow the hierarchical specification (13.10) and apply the decomposition (10.11) to the conditional distribution of $\boldsymbol{\varepsilon}_i$ given $\mathbf{b}_i$. Consequently, we can postulate that

$$\mathrm{Var}(\varepsilon_{ij} \mid \mathbf{b}_i) = \sigma^2\lambda^2(\mu_{ij}, \boldsymbol{\delta}; \mathbf{v}_{ij}), \tag{13.15}$$

with μ_{ij} defined in (13.14). It follows that, upon combining the use of the variance function with a correlation structure (Sect. 10.3), we can write that

$$\mathrm{Var}(\boldsymbol{\varepsilon}_i \mid \mathbf{b}_i) = \sigma^2\boldsymbol{R}_i(\boldsymbol{\mu}_i, \boldsymbol{\theta}_R; \mathbf{v}_i), \tag{13.16}$$

with $\boldsymbol{\theta}_R \equiv (\boldsymbol{\delta}, \boldsymbol{\varrho})$, where $\boldsymbol{\delta}$ is a vector of variance parameters employed by the variance function $\lambda(\cdot)$, $\boldsymbol{\varrho}$ is a vector of parameters related to the chosen correlation structure for the matrix $\boldsymbol{R}_i$, and $\mathbf{v}_i \equiv (\mathbf{v}'_{i1}, \ldots, \mathbf{v}'_{i,n_i})'$ is a vector of variance covariates for the observations from the ith group.

Equations (13.15) and (13.16) imply that, for models with mean-dependent variance functions from the <δ, μ>- and <μ>-groups (Sect. 7.3.1), $\boldsymbol{\varepsilon}_i$ depend on $\mathbf{b}_i$ through $\boldsymbol{\mu}_i$. This violates the assumption of the classical LMM and leads to the extended LMM.

Extended LMMs, defined with the use of variance functions from the <δ, μ>- and <μ>-groups, pose theoretical and computational difficulties. For this reason, in the current chapter we will mainly focus on the classical LMMs, defined by (13.1)–(13.4), i.e., for which the matrix $\boldsymbol{R}_i$ is specified with the use of a mean-independent variance function. Models defined with the use of mean-dependent variance functions, which we term mean-variance models (see also Sects. 7.8 and 10.7), will be treated separately in Sect. 13.8.

For the *mean-independent* functions, such as those from the <δ>-group (see Table 7.2), the definition (13.16) can be simplified as follows:

$$\mathrm{Var}(\boldsymbol{\varepsilon}_i \mid \mathbf{b}_i) = \mathrm{Var}(\boldsymbol{\varepsilon}_i) = \sigma^2 \boldsymbol{R}_i(\boldsymbol{\theta}_R; \mathbf{v}_i). \tag{13.17}$$

Note that (13.17) is concordant with the assumption that the residual errors $\boldsymbol{\varepsilon}_i$ are independent of the random effects $\mathbf{b}_i$. Consequently, the hierarchical model specification with mean-independent variance functions leads to the classical LMM, with $\boldsymbol{\mathcal{R}}_i = \sigma^2 \boldsymbol{R}_i(\boldsymbol{\theta}_R; \mathbf{v}_i)$. Essentially, this is the LMM formulation developed by Laird and Ware (1982).

It is worth noting that the choice of the structure of matrices $\boldsymbol{\mathcal{D}}$ and $\boldsymbol{\mathcal{R}}_i$ or, equivalently, $\boldsymbol{D}$ and $\boldsymbol{R}_i$ has consequences for the form of the *marginal* variance-covariance matrix of vector $\mathbf{y}_i$, implied by model (13.1)–(13.4). This form will be discussed in Sect. 13.5.1.

13.4.3 Additional Distributions Defined by y and b

In this section, we introduce additional auxiliary distributions related to LMMs. They build on distributions defined in Sects. 13.4.1 and 13.4.2 and play important role in the various aspects of model fitting and checking model assumptions.

13.4.3.1 Joint Distribution of y and b

The joint distribution $f_{y,b}(\mathbf{y}_i, \mathbf{b}_i)$ of $\mathbf{y}$ and $\mathbf{b}$ for the classical LMMs can be specified by taking the product of the unconditional distribution of the random effects $\mathbf{b}$ and the conditional distribution of $\mathbf{y}$ defined in Sects. 13.4.1 and 13.4.2:

$$f_{y,b}(\mathbf{y}_i, \mathbf{b}_i) = f_{y|b}(\mathbf{y}_i \mid \mathbf{b}_i) f_b(\mathbf{b}_i).$$

Given that the component distributions, $f_b(\mathbf{b})$ and $f_{y|b}(\mathbf{y} \mid \mathbf{b}) f_b(\mathbf{b})$, are multivariate normal, the joint distribution is also normal. We refer to the joint distribution in Sect. 13.5.3.

13.4.3.2 Marginal Distribution of y

The marginal distribution $f_y(\mathbf{y}_i)$ of $\mathbf{y}_i$ is obtained by "integrating out" the random effects $\mathbf{b}_i$ from the joint distribution of $\mathbf{y}_i$ and $\mathbf{b}_i$. More specifically, we calculate the density of the marginal distribution of $\mathbf{y}_i$ as

$$f_y(\mathbf{y}_i) = \int f_{y,b}(\mathbf{y}_i, \mathbf{b}_i)\, \mathrm{d}\mathbf{b}_i = \int f_{y|b}(\mathbf{y}_i \mid \mathbf{b}_i) f_b(\mathbf{b}_i)\, \mathrm{d}\mathbf{b}, \tag{13.18}$$

where $f_{y,b}$ is the density of the joint distribution of $\mathbf{y}_i$ and $\mathbf{b}_i$, $f_{y|b}$ is the conditional distribution of $\mathbf{y}_i$ given $\mathbf{b}_i$, and f_b is the density of the unconditional distribution of $\mathbf{b}_i$. Given that $f_{y,b}$ and f_b are densities of multivariate normal distributions, the marginal distribution of $\mathbf{y}$ is also multivariate normal and it can be derived analytically. In fact, it is given in (13.26).

13.4.3.3 Posterior Distribution of b Given y Is Known

The distribution $f_b(\mathbf{b}_i)$ of random effects $\mathbf{b}_i$ defined in (13.3) does *not* depend on the observed values of $\mathbf{y}_i$. Therefore, in the Bayesian setting, it is referred to as *prior* distribution of $\mathbf{b}_i$. Assuming that the observed values of $\mathbf{y}_i$ are equal to $\mathbf{y}_i^{(obs)}$, the so-called *posterior* distribution of $\mathbf{b}_i$ conditional on $\mathbf{y}_i^{(obs)}$ can be calculated using the following general formula:

$$f_{b|y}(\mathbf{b}_i|\mathbf{y}_i) \equiv f_{b|y}(\mathbf{b}_i|\mathbf{y}_i = \mathbf{y}_i^{(obs)}) = \frac{f_{y|b}(\mathbf{y}_i \mid \mathbf{b}_i) f_b(\mathbf{b}_i)}{\int f_{y|b}(\mathbf{y}_i \mid \mathbf{b}_i) f_b(\mathbf{b}_i)\, \mathrm{d}\mathbf{b}}. \tag{13.19}$$

Assuming that the parameters $\boldsymbol{\beta}, \boldsymbol{\theta}$ are known, the *posterior* distribution $f_{b|y}(\mathbf{b}_i|\mathbf{y}_i)$ for the classical LMMs is multivariate normal. Based on the observed data, we often estimate this distribution using its (*posterior*) mean:

$$\widehat{\mathbf{b}}_i(\boldsymbol{\beta}, \boldsymbol{\theta}) \equiv \widehat{\mathbf{b}}_i = \boldsymbol{D}\boldsymbol{Z}_i' \boldsymbol{V}_i^{-1} (\mathbf{y}_i^{(obs)} - \boldsymbol{X}_i \boldsymbol{\beta}). \tag{13.20}$$

Since the posterior mean is a linear function of $\mathbf{y}_i$, the variance-covariance matrix of the $\widehat{\mathbf{b}}_i$ estimator is equal to

$$\mathrm{Var}(\widehat{\mathbf{b}}_i) = \sigma^2 \boldsymbol{D}\boldsymbol{Z}_i' \left\{ \boldsymbol{V}_i^{-1} - \boldsymbol{V}_i^{-1} \boldsymbol{X}_i \left(\sum_{i=1}^{N} \boldsymbol{X}_i' \boldsymbol{V}_i^{-1} \boldsymbol{X}_i \right)^{-1} \boldsymbol{X}_i' \boldsymbol{V}_i^{-1} \right\} \boldsymbol{Z}_i \boldsymbol{D}. \tag{13.21}$$

To make inference about random effects, we are often interested in assessing the variability of the $\widehat{\mathbf{b}}_i - \mathbf{b}_i$ difference. The following formula can be used:

$$\mathrm{Var}(\widehat{\mathbf{b}}_i - \mathbf{b}_i) = \boldsymbol{\mathcal{D}} - \mathrm{Var}(\widehat{\mathbf{b}}_i). \tag{13.22}$$

It follows from the formula that, for any linear combination of random effects represented by the column vector $\boldsymbol{\lambda}$, the following inequality (see (7.7) in Verbeke and Molenberghs 2000) holds:

$$\mathrm{Var}(\boldsymbol{\lambda}'\widehat{\mathbf{b}}_i) \leq \mathrm{Var}(\boldsymbol{\lambda}'\mathbf{b}_i) = \boldsymbol{\lambda}'\boldsymbol{\mathcal{D}}\boldsymbol{\lambda}. \tag{13.23}$$

This inequality is one of many ways which illustrate "shrinkage" of the random effects toward the prior mean of $\mathbf{b}_i$, i.e., toward zero. We revisit this issue in Sect. 13.6.1. On a side, we note that, for the LMM defined by (13.1)–(13.4), the posterior mean (13.20) is also the mode of the density of the posterior distribution of $\mathbf{b}_i$, given $\mathbf{y}_i$. In fact, the use of the mode to predict the random effects can be applied to mixed-effects models in general, including GLMMs and NLMMs. However, for mixed-effects models other than the LMMs (13.1)–(13.4), the mode does not have, in general, to be equal to the posterior mean.

13.5 Estimation

In this section, we present methods to obtain a set of estimates of parameters $\boldsymbol{\beta}$, σ^2, $\boldsymbol{\theta}_D$, and $\boldsymbol{\theta}_R$ for the classical LMM, defined by (13.1)–(13.4). The case of the extended, mean-variance model will be discussed separately in Sect. 13.8.

In Sect. 13.5.1, we present the marginal model, implied by the classical LMM. The marginal model allows estimating the LMM using the methods presented in Sect. 10.4 for the LM with fixed effects and correlated residual errors. In Sect. 13.5.2, we briefly describe the necessary modifications of the methods. In particular, we focus on the approaches that are implemented in R. Section 13.5.4 briefly discusses the issue of the parameterization of the classical LMM, while in Sect. 13.5.5 we describe the methods to assess the uncertainty of the parameter estimates. To complete the description of the estimation approaches, in Sect. 13.5.6, we briefly discuss approaches alternative to those presented in Sect. 13.5.2.

13.5.1 The Marginal Model Implied by the Classical Linear Mixed-Effects Model

For the classical LMM, Equations (13.12)–(13.13) and (13.17) imply that the marginal mean and variance-covariance matrix of $\mathbf{y}_i$ are given as follows:

$$\mathrm{E}(\mathbf{y}_i) = \boldsymbol{X}_i\boldsymbol{\beta}, \tag{13.24}$$

$$\begin{aligned}\mathrm{Var}(\mathbf{y}_i) &\equiv \boldsymbol{\mathcal{V}}_i(\sigma^2, \boldsymbol{\theta}; \mathbf{v}_i) \\ &= \sigma^2\boldsymbol{V}_i(\boldsymbol{\theta}; \mathbf{v}_i) = \sigma^2[\boldsymbol{Z}_i\boldsymbol{D}(\boldsymbol{\theta}_D)\boldsymbol{Z}_i' + \boldsymbol{R}_i(\boldsymbol{\theta}_R; \mathbf{v}_i)],\end{aligned} \tag{13.25}$$

where $\boldsymbol{\theta}' \equiv (\boldsymbol{\theta}_D', \boldsymbol{\theta}_R')'$. Note that, to simplify the notation, from now on, we will, in general, suppress the use of $\boldsymbol{\theta}$ and $\mathbf{v}_i$ in the formulae, in line with conventions used in Parts II and III of the book.

From (13.24) and (13.25), it follows that, marginally,

$$\mathbf{y}_i \sim \mathcal{N}_{n_i}(\boldsymbol{X}_i\boldsymbol{\beta}, \sigma^2\boldsymbol{Z}_i\boldsymbol{D}\boldsymbol{Z}_i' + \sigma^2\boldsymbol{R}_i). \tag{13.26}$$

The marginal mean value of the dependent variable vector $\mathbf{y}_i$, similarly to the linear model (10.1)–(10.5), is defined by a linear combination of the vectors of covariates included, as columns, in the group-specific design matrix $\boldsymbol{X}_i$, with parameters $\boldsymbol{\beta}$. Moreover, the variance-covariance matrix of $\mathbf{y}_i$ consists of two components. The first one, $\sigma^2\boldsymbol{Z}_i\boldsymbol{D}\boldsymbol{Z}_i'$, is contributed by the random effects $\mathbf{b}_i$. The second one, $\sigma^2\boldsymbol{R}_i$, is related to the residual errors $\boldsymbol{\varepsilon}_i$. Hence, strictly speaking, the model employing random effects, specified in (13.1)–(13.4), implies a marginal normal distribution, defined by (13.26), which is similar to distributions considered in Chap. 10 in the context of LMs for correlated data, but with the variance-covariance matrix of $\mathbf{y}_i$ of a very specific parametric form, given by (13.25).

It is worth observing that the marginal model, defined by (13.24)–(13.26), does not involve the random effects $\mathbf{b}_i$. Thus, the matrix $\mathcal{D}$ does not have to be treated as a variance-covariance matrix. Consequently, it does not have to be positive-definite, as long as the matrix $\boldsymbol{\mathcal{V}}_i$ is positive-definite. The matrix $\mathcal{D}$ does need to be symmetric, though, to assure that the matrix $\boldsymbol{\mathcal{V}}_i$ is symmetric. It follows that, while every LMM of the form, specified in (13.1)–(13.4), implies a marginal model, defined by (13.26), not every model of the form (13.26) can be interpreted as resulting from an LMM. Thus, the two models are *not* equivalent.

From the above it follows that LMs with fixed effects and correlated residual errors, presented in Chap. 10, are less restrictive than LMMs. Thus, in this respect, the former are more flexible than the latter. On the other hand, in general, LMs with fixed effects and correlated residual errors do not allow making inference about the variability that may be related to different levels of the data hierarchy.

It is worth noting that the effects of the covariates, included in the design matrix $\boldsymbol{X}_i$, are quantified by the *same* parameters $\boldsymbol{\beta}$ in both the conditional (13.12) and unconditional (13.24) mean. Thus, although the parameters are defined in the context of the subject-specific model (13.1), they can also be interpreted as quantifying effects at the population level. This possibility of a dual interpretation of fixed-effects $\boldsymbol{\beta}$ is a unique feature of the classical LMM, given by (13.1)–(13.4). It does not hold, for instance, for GLMMs, not described in this book.

The fact that the classical LMM implies the marginal model (13.26) is also important from a practical point of view. This is because it allows the construction of effective estimation approaches for the LMM. This topic is discussed in the next section.

13.5.2 Maximum-Likelihood Estimation

In general, the ML estimation involves constructing the likelihood function based on appropriate probability distribution function for the *observed data*. The unconditional distribution of $\mathbf{b}_i$ and the conditional distribution of $\mathbf{y}_i$ given $\mathbf{b}_i$, which were defined in Sect. 13.4 for the classical LMM, are not suitable for constructing the likelihood function, because the random effects $\mathbf{b}_i$ are not observed. For a similar reason, the joint distribution of $\mathbf{y}_i$ and $\mathbf{b}_i$ cannot be used.

Instead, estimation of LMMs is based on the marginal distribution of $\mathbf{y}_i$.

In fact, it coincides with the distribution given in (13.26). For this reason, the estimation of parameters of the classical LMM can be accomplished by using the ML or REML estimation for the implied marginal model, along the lines similar to those described in Sect. 10.4.2.

In particular, the ML estimation is based on the marginal log-likelihood resulting from (13.26). Following (10.25), the log-likelihood can be expressed as follows:

$$\ell_{\text{Full}}(\boldsymbol{\beta}, \sigma^2, \boldsymbol{\theta}) \equiv -\frac{N}{2}\log(\sigma^2) - \frac{1}{2}\sum_{i=1}^{N}\log[\det(\boldsymbol{V}_i)] - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(\mathbf{y}_i - \boldsymbol{X}_i\boldsymbol{\beta})'\boldsymbol{V}_i^{-1}(\mathbf{y}_i - \boldsymbol{X}_i\boldsymbol{\beta}), \tag{13.27}$$

where $\boldsymbol{V}_i$, defined in (13.25), depends on $\boldsymbol{\theta}$.

Estimates of $\boldsymbol{\beta}$, σ^2, and $\boldsymbol{\theta}$ are usually obtained using a log-profile-likelihood for $\boldsymbol{\theta}$ (see Sect. 10.4.2). The log-profile-likelihood results from plugging into (13.27) the estimators of $\boldsymbol{\beta}$ and σ^2, given by

$$\widehat{\boldsymbol{\beta}}(\boldsymbol{\theta}) \equiv \left(\sum_{i=1}^{N}\boldsymbol{X}_i'\boldsymbol{V}_i^{-1}\boldsymbol{X}_i\right)^{-1}\sum_{i=1}^{N}\boldsymbol{X}_i'\boldsymbol{V}_i^{-1}\mathbf{y}_i, \tag{13.28}$$

$$\widehat{\sigma}^2_{\text{ML}}(\boldsymbol{\theta}) \equiv \sum_{i=1}^{N}\mathbf{r}_i'\boldsymbol{V}_i^{-1}\mathbf{r}_i/n, \tag{13.29}$$

where $\mathbf{r}_i \equiv \mathbf{r}_i(\boldsymbol{\theta}) = \mathbf{y}_i - \boldsymbol{X}_i\widehat{\boldsymbol{\beta}}(\boldsymbol{\theta})$. Note that the expressions correspond to (10.26) and (10.27), presented in Sect. 10.4.2. By maximizing the log-profile-likelihood function over $\boldsymbol{\theta}$, we obtain estimators of these parameters. Plugging $\widehat{\boldsymbol{\theta}}$ into (13.28) and (13.29) yields the corresponding estimators of $\boldsymbol{\beta}$ and σ^2, respectively.

As has been mentioned in Sects. 4.4.2, 7.4.2, and 10.4.2, the ML estimates of the variance-covariance parameters are biased. For this reason, the parameters are better estimated using the REML estimation. Toward this end, the log-restricted-likelihood function, corresponding to (10.30), is considered. From this function, the parameter σ^2 is profiled out by replacing it by the following estimator, corresponding to (10.31):

$$\widehat{\sigma}^2_{\text{REML}}(\boldsymbol{\theta}) \equiv \sum_{i=1}^{N} \mathbf{r}_i' \boldsymbol{V}_i^{-1} \mathbf{r}_i / (n-p), \tag{13.30}$$

with $\mathbf{r}_i$ defined as in (13.29). This leads to a log-profile-restricted-likelihood function, which only depends on $\boldsymbol{\theta}$:

$$\begin{aligned} \ell^*_{\text{REML}}(\boldsymbol{\theta}) \equiv & -\frac{n-p}{2} \log \left(\sum_{i=1}^{N} \mathbf{r}_i' \mathbf{r}_i \right) - \frac{1}{2} \sum_{i=1}^{N} \log[\det(\boldsymbol{V}_i)] \\ & -\frac{1}{2} \log \left[\det \left(\sum_{i=1}^{N} \boldsymbol{X}_i' \boldsymbol{V}_i^{-1} \boldsymbol{X}_i \right) \right]. \end{aligned} \tag{13.31}$$

Maximization of (13.31) yields an estimator of $\boldsymbol{\theta}$, which is then plugged into (13.28) and (13.30) to provide estimators of $\boldsymbol{\beta}$ and σ^2, respectively.

For the mean-variance model, i.e., when the conditional variance of random errors is defined with the use of a variance function (13.15) that does depend on μ_{ij} (Sect. 7.3.1), the estimates of the parameters $\boldsymbol{\beta}$, σ^2, and $\boldsymbol{\theta}$ can be obtained using GLS approaches similar to those described in Sects. 7.8.1.1 and 10.4.2. We discuss these approaches in Sect. 13.8.

13.5.3 Penalized Least Squares

In this section, we outline a slightly different approach to the estimation of parameters $\boldsymbol{\beta}$, σ^2, $\boldsymbol{\theta}$ for the classical LMM, defined by (13.1)–(13.4). Essentially, the approach is based on the log-profile-restricted-likelihood for θ, as defined in Sect. 13.5.2. However, the numerical algorithm based on sparse matrices allows for a numerically efficient implementation of this *penalized least squares* (PnLS) approach. In our presentation we follow Bates (2012) who describes in detail a more general version of this algorithm, namely, *penalized weighted least squares* (PWLS) used in the context of GLMMs and NLMMs and implemented in the package **lme4.0**. For the sake of future reference and simplicity, we briefly summarize the methodology. We consider a single-level LMM, specified for all data (Sect. 13.2.2). Moreover, we assume the conditional independence model and homogeneous residual-error variance, i.e., $\boldsymbol{R} \equiv \boldsymbol{I}_n$.

In the PnLS estimation approach, the starting point is the density of the joint distribution of $\mathbf{y}$ and random effects $\mathbf{b}$ introduced in general terms in Sect. 13.4.3. The logarithm of the density of the joint distribution of $\mathbf{y}$ and random effects $\mathbf{b}$ is given by

$$\begin{aligned} h_{\text{Joint}}(\mathbf{y}, \mathbf{b}; \boldsymbol{\beta}, \sigma^2, \boldsymbol{\theta}) \equiv & -\frac{n+Nq}{2} \log(\sigma^2) - \frac{1}{2} \log[\det(\mathbf{D})] \\ & -\frac{(\mathbf{y} - \boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{Z}\mathbf{b})'(\mathbf{y} - \boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{Z}\mathbf{b}) + \mathbf{b}'\mathbf{D}^{-1}\mathbf{b}}{2\sigma^2}, \end{aligned} \tag{13.32}$$

where $\boldsymbol{X}$ and $\boldsymbol{Z}$ were defined in (13.7), while $\mathbf{D}$ was specified in (13.9). Note that, given the assumption that $\boldsymbol{R} \equiv \boldsymbol{I}_n$, in (13.32) for the remainder of the section, we have $\boldsymbol{\theta} \equiv \boldsymbol{\theta}_D$.

Upon applying the following form of the Cholesky representation:

$$\mathbf{D} = \boldsymbol{TSST}', \tag{13.33}$$

where $\boldsymbol{T}$ is a lower-triangular matrix with all diagonal elements equal to 1 and $\boldsymbol{S}$ is a diagonal matrix with nonnegative diagonal elements, we can express $\mathbf{b}$ as follows:

$$\mathbf{b} = \boldsymbol{TS}\mathbf{u}, \quad \text{with} \quad \mathbf{u} \sim \mathcal{N}_{Nq}(\mathbf{0}, \sigma^2 \mathbf{I}_{Nq}).$$

By allowing for zero elements on the diagonal matrix $\boldsymbol{S}$ used in (13.33), we consider a general case with a potentially singular (positive semi-definite) matrix $\mathbf{D}$.

It follows that, conditionally on $\mathbf{u}$, $\mathbf{y}$ is normally distributed with

$$\begin{aligned} \mathrm{E}(\mathbf{y} \mid \mathbf{u}) &= \boldsymbol{X\beta} + \boldsymbol{ZTS}\mathbf{u} \equiv \boldsymbol{X\beta} + \boldsymbol{A}'\mathbf{u}, \\ \mathrm{Var}(\mathbf{y} \mid \mathbf{u}) &= \sigma^2 \boldsymbol{I}_n, \end{aligned} \tag{13.34}$$

while the marginal mean and variance of $\mathbf{y}$ can be expressed as

$$\begin{aligned} \mathrm{E}(\mathbf{y}) &= \boldsymbol{X\beta}, \\ \mathrm{Var}(\mathbf{y}) &= \sigma^2 (\boldsymbol{A}'\boldsymbol{A} + \boldsymbol{I}_n). \end{aligned} \tag{13.35}$$

Using the representation introduced above, (13.32) can be written as follows:

$$\begin{aligned} h_{\text{PnLS}}(\mathbf{y}, \mathbf{b}; \boldsymbol{\beta}, \sigma^2, \boldsymbol{\theta}) &\equiv -\frac{n+Nq}{2} \log(\sigma^2) \\ &\quad - \frac{(\mathbf{y} - \boldsymbol{X\beta} - \boldsymbol{A}'\mathbf{u})'(\mathbf{y} - \boldsymbol{X\beta} - \boldsymbol{A}'\mathbf{u}) + \mathbf{u}'\mathbf{u}}{2\sigma^2} \\ &\equiv -\frac{n+Nq}{2} \log(\sigma^2) - \frac{d(\boldsymbol{\beta}, \boldsymbol{\theta})}{2\sigma^2}. \end{aligned} \tag{13.36}$$

Note that term $d(\boldsymbol{\beta}, \boldsymbol{\theta})$ in (13.36) resembles a penalized sum of squares. In fact, it can be seen as a residual sum of squares in a linear regression model

$$\mathrm{E}\begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix} = \begin{bmatrix} \boldsymbol{A}' & \boldsymbol{X} \\ \mathbf{I}_{Nq} & \mathbf{0} \end{bmatrix} \begin{pmatrix} \mathbf{u} \\ \boldsymbol{\beta} \end{pmatrix} \equiv \boldsymbol{X}^* \begin{pmatrix} \mathbf{u} \\ \boldsymbol{\beta} \end{pmatrix}.$$

The solution $(\widetilde{\mathbf{u}}', \widetilde{\boldsymbol{\beta}}')'$ for the linear regression problem satisfies

$$(\boldsymbol{X}^*)'\boldsymbol{X}^* \begin{pmatrix} \widetilde{\mathbf{u}} \\ \widetilde{\boldsymbol{\beta}} \end{pmatrix} = (\boldsymbol{X}^*)' \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix},$$

which can be explicitly written as

$$\begin{bmatrix} \boldsymbol{A}\boldsymbol{A}' + \mathbf{I}_{Nq} & \boldsymbol{A}\boldsymbol{X} \\ \boldsymbol{X}'\boldsymbol{A}' & \boldsymbol{X}'\boldsymbol{X} \end{bmatrix} \begin{pmatrix} \widetilde{\mathbf{u}} \\ \widetilde{\boldsymbol{\beta}} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}\mathbf{y} \\ \boldsymbol{X}'\mathbf{y} \end{pmatrix}. \tag{13.37}$$

It is worth noting that (13.37) corresponds to a general form of the LMM equations considered by Henderson (1984), which allow for a singular estimate of $\mathbf{D}$.

To reduce the storage space requirements and numerical complexity, it is advantageous to introduce a sparse lower-triangular Cholesky decomposition matrix

$$\boldsymbol{L} = \begin{bmatrix} \boldsymbol{L}_Z & \mathbf{0} \\ \boldsymbol{L}_{ZX} & \boldsymbol{L}_X \end{bmatrix},$$

which satisfies

$$\boldsymbol{L}\boldsymbol{L}' = \boldsymbol{P}(\boldsymbol{X}^*)'\boldsymbol{X}^*\boldsymbol{P}', \tag{13.38}$$

where the orthogonal matrix $\boldsymbol{P}$ is a "fill-reducing" permutation matrix, determined from the pattern of nonzero elements in $\boldsymbol{Z}$. The matrix reduces the number of nonzero elements in $\boldsymbol{L}$ and hence has a large impact on the storage space required for $\boldsymbol{L}$. It is important to stress that, although this has not been explicitly indicated in (13.38), $\boldsymbol{L}$ depends on $\boldsymbol{\theta}$.

If we assume that the matrix $\boldsymbol{P}$ is of a block-diagonal form

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{P}_Z & \mathbf{0} \\ \mathbf{0} & \boldsymbol{P}_X \end{bmatrix},$$

then we get

$$\begin{bmatrix} \boldsymbol{P}_Z'\boldsymbol{L}_Z & \mathbf{0} \\ \boldsymbol{P}_X'\boldsymbol{L}_{ZX} & \boldsymbol{P}_X'\boldsymbol{L}_X \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_Z'\boldsymbol{P}_Z & \boldsymbol{L}_{ZX}'\boldsymbol{P}_X \\ \mathbf{0} & \boldsymbol{L}_X'\boldsymbol{P}_X \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}\boldsymbol{A}' + \mathbf{I}_{Nq} & \boldsymbol{A}\boldsymbol{X} \\ \boldsymbol{X}'\boldsymbol{A}' & \boldsymbol{X}'\boldsymbol{X} \end{bmatrix}. \tag{13.39}$$

Consequently, we can rewrite (13.36) as follows:

$$\begin{aligned} h_{\text{PnLS}}(\mathbf{y}, \mathbf{b}; \boldsymbol{\beta}, \sigma^2, \boldsymbol{\theta}) = &-\frac{n+Nq}{2}\log(\sigma^2) - \frac{\widetilde{d}(\boldsymbol{\theta})}{2\sigma^2} \\ &-\frac{1}{2\sigma^2}\begin{pmatrix} \boldsymbol{P}_Z(\mathbf{u}-\widetilde{\mathbf{u}}) \\ \boldsymbol{P}_X(\boldsymbol{\beta}-\widetilde{\boldsymbol{\beta}}) \end{pmatrix}' \boldsymbol{L}\boldsymbol{L}' \begin{pmatrix} \boldsymbol{P}_Z(\mathbf{u}-\widetilde{\mathbf{u}}) \\ \boldsymbol{P}_X(\boldsymbol{\beta}-\widetilde{\boldsymbol{\beta}}) \end{pmatrix}, \end{aligned} \tag{13.40}$$

where $\widetilde{d}(\boldsymbol{\theta})$ is the value of penalized sum of squares $d(\boldsymbol{\beta}, \boldsymbol{\theta})$, defined in (13.36), computed at solution $(\widetilde{\mathbf{u}}', \widetilde{\boldsymbol{\beta}}')'$ of system of (13.37). Thus, $\widetilde{d}(\boldsymbol{\theta})$ is the minimum value of penalized sum of squares, assuming $\boldsymbol{\theta}$ is known.

The marginal log-likelihood, corresponding to (13.40), is given by

$$\ell_{\text{ML}}(\boldsymbol{\beta}, \sigma^2, \boldsymbol{\theta}) \equiv -\frac{n}{2}\log(\sigma^2) - \frac{1}{2}\log\{[\det(\boldsymbol{L}_Z)]^2]\} - \frac{\widetilde{d}(\boldsymbol{\theta})}{2\sigma^2} - \frac{1}{2\sigma^2}\left[\boldsymbol{L}_X'\boldsymbol{P}_X(\boldsymbol{\beta} - \widetilde{\boldsymbol{\beta}})\right]' \boldsymbol{L}_X'\boldsymbol{P}_X(\boldsymbol{\beta} - \widetilde{\boldsymbol{\beta}}). \tag{13.41}$$

Essentially, it is a re-parameterized form of (13.27).

Given $\boldsymbol{\theta}$, the resulting estimator of $\boldsymbol{\beta}$ is $\widetilde{\boldsymbol{\beta}}$, defined in (13.37), while for σ^2 the estimator is given by

$$\widetilde{\sigma}^2_{\text{ML}} \equiv \frac{\widetilde{d}(\boldsymbol{\theta})}{n}. \tag{13.42}$$

By plugging $\widetilde{\boldsymbol{\beta}}$ and $\widetilde{\sigma}^2$ into (13.41), we obtain the log-profile-likelihood for $\boldsymbol{\theta}$:

$$\ell^*_{\text{ML}}(\boldsymbol{\theta}) \equiv -\frac{1}{2}\log\{[\det(\boldsymbol{L}_Z)]^2\} - \frac{n}{2}\log[\widetilde{d}(\boldsymbol{\theta})]. \tag{13.43}$$

The log-profile-likelihood is a re-parameterized version of the function obtained from plugging the estimators (13.28) and (13.29) into (13.27) (see Sect. 13.5.2).

Maximization of (13.43) over $\boldsymbol{\theta}$ yields the ML estimator of the parameter vector. The estimator is then used to obtain the ML estimators of σ^2 and $\boldsymbol{\beta}$ from (13.42) and (13.37), respectively. The estimators correspond to those given in (13.28) and (13.29).

The REML estimator of $\boldsymbol{\theta}$ is obtained by maximizing the log-profile-restricted-likelihood:

$$\ell^*_{\text{REML}}(\boldsymbol{\theta}) \equiv -\frac{1}{2}\log\{[\det(\boldsymbol{L}_Z)\det(\boldsymbol{L}_X)]^2\} - \frac{n-p}{2}\log[\widetilde{d}(\boldsymbol{\theta})]. \tag{13.44}$$

The function is, essentially, a re-parameterized form of (13.31). The resulting estimator is then used to obtain the estimator of σ^2, which corresponds to the one given in (13.30):

$$\widetilde{\sigma}^2_{\text{REML}} \equiv \frac{\widetilde{d}(\boldsymbol{\theta})}{n-p}. \tag{13.45}$$

An estimate of $\boldsymbol{\beta}$ is computed from (13.37).

As mentioned at the beginning of this section, the PnLS approach, described above, has been implemented in the package **lme4.0**, a developmental branch version of **lme4**. In the latter package, the implementation has been modified in several ways (Bates et al. 2012). First, the decomposition (13.33) of the matrix $\mathbf{D}$ has been replaced by the classical Cholesky decomposition $\mathbf{D} = \mathbf{Q}\mathbf{Q}'$. Additionally, the lower-triangular matrix on the right-hand side of (13.39) has been assumed to take the following form:

$$\begin{bmatrix} \boldsymbol{P}_Z'\boldsymbol{T}_Z & \mathbf{0} \\ \boldsymbol{T}_{ZX}' & \boldsymbol{T}_X' \end{bmatrix},$$

where matrices $\boldsymbol{T}_Z$ (lower-triangular), $\boldsymbol{P}_Z$ (permutation), $\boldsymbol{T}_X$, and $\boldsymbol{T}_{ZX}$ (upper-triangular) are defined by the following relationships:

$$\begin{aligned} \boldsymbol{T}_Z\boldsymbol{T}_Z' &\equiv \boldsymbol{P}_Z(\boldsymbol{A}\boldsymbol{A}' + \mathbf{I}_{Nq})\boldsymbol{P}_Z', \\ \boldsymbol{T}_Z\boldsymbol{T}_{ZX} &\equiv \boldsymbol{P}_Z\boldsymbol{A}\boldsymbol{X}, \\ \boldsymbol{T}_{ZX}'\boldsymbol{T}_{ZX} &\equiv \boldsymbol{X}'\boldsymbol{X} - \boldsymbol{T}_X'\boldsymbol{T}_X, \end{aligned}$$

with $\boldsymbol{A} \equiv \boldsymbol{\Omega}'\boldsymbol{Z}'$. By using the resulting decomposition, we obtain formulae for the log-profile-likelihood and log-profile-restricted-likelihood similar to (13.43) and (13.44), respectively, but with $\det(\boldsymbol{L}_Z)$ and $\det(\boldsymbol{L}_X)$ replaced, respectively, by $\det(\boldsymbol{T}_Z)$ and $\det(\boldsymbol{T}_X)$. Moreover, Equation (13.37), defining the PnLS estimates $\widetilde{\mathbf{u}}$ and $\widetilde{\boldsymbol{\beta}}$, can now be equivalently expressed as

$$\begin{aligned} \boldsymbol{T}_X\widetilde{\boldsymbol{\beta}} &= \mathbf{c}_\beta, \\ \boldsymbol{T}_Z'\boldsymbol{P}_Z\widetilde{\mathbf{u}} &= \mathbf{c}_u - \boldsymbol{T}_{ZX}\widetilde{\boldsymbol{\beta}}, \end{aligned}$$

where the vectors $\mathbf{c}_\beta$ and $\mathbf{c}_u$ are defined by

$$\begin{aligned} \boldsymbol{T}_Z\mathbf{c}_u &= \boldsymbol{P}_Z\boldsymbol{A}\mathbf{y}, \\ \boldsymbol{T}_X'\mathbf{c}_\beta &= \boldsymbol{X}'\mathbf{y} - \boldsymbol{T}_{ZX}\mathbf{c}_u. \end{aligned}$$

13.5.4 Constrained Versus Unconstrained Parameterization of the Variance-Covariance Matrix

Solving of maximization problems, necessary to obtain the estimators described in Sects. 13.5.2 and 13.5.3, is difficult from a numerical point of view. This is because the solution should lead to symmetric and positive-definite variance-covariance matrices $\boldsymbol{\mathcal{D}}$ and $\boldsymbol{\mathcal{R}}_i$, defined in (13.3). A possible solution is to parameterize the matrices in such a way that the optimization problem becomes unconstrained.

Toward this end, for matrix $\boldsymbol{\mathcal{R}}_i$, one can use representation (13.4) and parameterize $\boldsymbol{R}_i$ by the parameterizations described in Sect. 10.4.3. For matrix $\boldsymbol{\mathcal{D}}$, several solutions are possible (Pinheiro and Bates 1996).

For instance, we could consider parameterizing $\boldsymbol{\mathcal{D}}$ in terms of variances and correlations. By using the log-transformation for the variances and Fisher's z-transform, defined in (10.33), for correlations, we would obtain a set of unconstrained parameters. This parameterization would reflect the individual constraints, i.e., that variances need to be positive and correlation coefficients are constrained

to lie within the $[0,1]$ interval. However, in general, it would not reflect the *joint* restriction, i.e., that the set of back-transformed parameters has to define a positive-definite matrix. Thus, while this parameterization could be used for positive-definite matrices of some particular structure like, e.g., the compound-symmetry structure, i.e., with equal diagonal elements and equal off-diagonal elements (see Sect. 11.4.1), it is not suitable for the numerical optimization purposes in general. However, it is useful for the construction of confidence intervals for the elements of matrix $\boldsymbol{\mathcal{D}}$, as it will be explained in Sect. 13.7.3.

An alternative, which addresses the issue, uses (13.4) and considers the representation of $\boldsymbol{D}$ in terms of the elements of its Cholesky decomposition, i.e., in terms of the elements of the upper-triangular matrix $\boldsymbol{U}$, where $\boldsymbol{D} = \boldsymbol{U}'\boldsymbol{U}$. The main advantage of this approach is that it is computationally simple and stable. However, one of its disadvantages is that the resulting parameterization is not unique. This problem is removed by requiring that the diagonal elements of $\boldsymbol{U}$ are positive. In that case, an unconstrained parameterization of $\boldsymbol{D}$ is obtained using the logarithms of the diagonal elements of $\boldsymbol{U}$ together with the off-diagonal elements of $\boldsymbol{U}$. Pinheiro and Bates (1996) call this parameterization a "log-Cholesky parameterization". Another disadvantage of this approach, however, is that there is no straightforward relationship between the elements of $\boldsymbol{D}$ and $\boldsymbol{U}$, except for the fact that $\mid \boldsymbol{U}_{1,1} \mid = \sqrt{\boldsymbol{D}_{1,1}}$. This latter relationship does allow deriving confidence intervals for the diagonal elements of $\boldsymbol{D}$, i.e., variances, but not for the off-diagonal elements, i.e., covariances.

Another approach to obtaining an unconstrained parameterization of $\boldsymbol{D}$ is to use the *matrix logarithm* (Pinheiro and Bates (1996, 2000), pp. 78–79). In particular, $\boldsymbol{D}$ is expressed using its *singular value decomposition* (SVD):

$$\boldsymbol{D} = \boldsymbol{Q}\boldsymbol{T}\boldsymbol{Q}', \tag{13.46}$$

where $\boldsymbol{T}$ is a diagonal matrix with all diagonal elements positive, and $\boldsymbol{Q}$ is an orthogonal matrix. Let us denote by $\log(\boldsymbol{T})$ a diagonal matrix with the diagonal elements equal to the logarithms of diagonal elements of $\boldsymbol{T}$. Next, define

$$\boldsymbol{D}^* \equiv \boldsymbol{Q}\log(\boldsymbol{T})\boldsymbol{Q}'. \tag{13.47}$$

The $\boldsymbol{D}^*$ matrix is logarithm of $\boldsymbol{D}$, i.e., $\boldsymbol{D} = \exp(\boldsymbol{D}^*)$, where

$$\exp(\boldsymbol{D}^*) \equiv \sum_{k=0}^{\infty} \frac{(\boldsymbol{D}^*)^k}{k!}. \tag{13.48}$$

The relationship $\boldsymbol{D} = \exp(\boldsymbol{D}^*)$ allows expressing the parameters $\boldsymbol{\theta}_D$ (Sect. 13.4), which define the matrix, as a function of the elements of the upper triangle of the matrix $\boldsymbol{D}^*$. The latter elements form a set of unconstrained parameters that can be used for numerical optimization purposes. However, there is no straightforward relation between the elements of $\boldsymbol{D}$ and $\boldsymbol{D}^*$. Thus, the matrix-logarithm parameterization is not suitable for the construction of confidence intervals for the elements of $\boldsymbol{D}$.

In some situations, it may not be possible to find a solution of the optimization problem that would lead to a positive-definite $\boldsymbol{D}$. This may happen if, e.g., the assumed form of the LMM, defined by (13.1)–(13.4), is not correct. In this case, a possible alternative is to consider the implied marginal model (13.26). As was mentioned in Sect. 13.5.1, in the marginal model, the important constraint is that the marginal variance-covariance matrix $\boldsymbol{\mathcal{V}}_i$, given in (13.25), is positive-definite; the positive-definiteness of $\boldsymbol{D}$ is not a necessary condition. Thus, in principle, one could consider fitting the LMM with the only constraint that $\boldsymbol{\mathcal{V}}_i$ is positive-definite. The resulting solution, if feasible, may lead, however, to a non-positive-definite $\boldsymbol{D}$, which would violate the interpretation of the model as a hierarchical one (Sect. 13.5.1), but would lead to a valid marginal model nevertheless. Such an option is not routinely available in functions used for fitting LMMs in R.

13.5.5 Uncertainty in Parameter Estimation

Similar to the case of the LM for correlated data (Sect. 10.4.4), the variance-covariance matrix of $\widehat{\boldsymbol{\beta}}$ is estimated by

$$\widehat{\mathrm{Var}}(\widehat{\boldsymbol{\beta}}) \equiv \widehat{\sigma}^2 \left(\sum_{i=1}^{N} \boldsymbol{X}_i' \widehat{\boldsymbol{V}}_i^{-1} \boldsymbol{X}_i \right)^{-1}, \tag{13.49}$$

where $\widehat{\sigma}^2$ and $\widehat{\boldsymbol{V}}_i$ are estimated by one of the methods described in Sect. 13.5.2. Note that, in the computation of $\widehat{\mathrm{Var}}(\widehat{\boldsymbol{\beta}})$, given in (13.49), the extra variability resulting from the use of the estimate $\widehat{\boldsymbol{V}}_i$ is not accounted for. For this reason, the computed variance underestimates the true variability of $\widehat{\boldsymbol{\beta}}$.

The variance-covariance matrix of $\widehat{\sigma}^2$ and $\widehat{\boldsymbol{\theta}}$ can be estimated in various ways. A possible solution, implemented in the `lme()` function of the **nlme** package in R, is to use the inverse of the negative Hessian of the log-restricted-likelihood (see (10.30) and Sect. 13.5.2), evaluated at the estimated values of σ^2 and $\boldsymbol{\theta}$. An alternative is to consider the inverse of the negative Hessian of the log-profile-likelihood, which results from replacing $\boldsymbol{\beta}$ in (13.27) with the estimator given by (13.28). Obviously, the validity of these computations depends on the validity of the likelihood functions, i.e., on the correct specification of the model.

It is worth mentioning that the variance-covariance matrix of $\widehat{\boldsymbol{\beta}}$ can be expressed in a form similar to (10.45), with $\boldsymbol{\mathcal{R}}_i$ replaced by the model-based marginal variance-covariance $\boldsymbol{\mathcal{V}}_i$, given in (13.25) (see, e.g., Equation (6.2) in Verbeke and Molenberghs (2000)). Consequently, similarly to the situation described Sect. 10.7 for LMs for correlated data, if $\boldsymbol{\mathcal{V}}_i \not\equiv \mathrm{Var}(\mathbf{y}_i)$, i.e., if the LMM is not correctly specified, (13.49) will result in a biased (underestimated) assessment of the variability of $\widehat{\boldsymbol{\beta}}$.

13.5.6 Alternative Estimation Approaches

Although the ML- and REML-based approaches, described in Sect. 13.5.2, are the most popular estimation methods for LMMs, other approaches are also possible. Among them one discerns, for instance, a Bayesian approach, a noniterative minimum variance quadratic unbiased estimation (MIVQUE), and the EM-algorithm. Neither the Bayesian approach nor MIVQUE is implemented in the most popular packages used to fit LMMs in R. The EM-algorithm is used in the function `lme()` from the package **nlme** only to refine the initial values of the parameters $\boldsymbol{\theta}_D$ in the first iterations of the optimization routine. For these reasons, we will not provide a more detailed description of these approaches here. The interested reader is referred to the monographs by Davidian and Giltinan (1995), Gelman et al. (1995), or Verbeke and Molenberghs (2000).

13.6 Model Diagnostics

In analogy with other types of LMs (see Sects. 4.5, 7.5, and 10.5), after fitting an LMM, and before making any inferences based on it, it is important to check whether the model assumptions are fulfilled. The two main distributional assumptions pertain to the normality of the random effects $\mathbf{b}_i$ and of the residual errors $\boldsymbol{\varepsilon}_i$. Evaluation of the influence of individual observations on the model fit (Sect. 4.5.3) may also be of importance. These topics are discussed in this section. Note that, as in Sect. 13.5, we focus on the classical LMM, defined by (13.1)–(13.4), in which the matrix $\boldsymbol{R}_i$ is specified with the use of a variance function, which does not depend on the mean values μ_{ij} (Table 7.2).

13.6.1 Normality of Random Effects

In the LMM defined by (13.1)–(13.4), it is assumed that the random effects $\mathbf{b}_i$ are normally distributed with the mean zero and the variance-covariance matrix $\sigma^2\boldsymbol{D}$. To check the assumption, some "estimates" of the random effects $\mathbf{b}_i$ are needed. Toward this end, usually the conditional expectations of the random effects, given the observed responses of $\mathbf{y}_i$, are used:

$$\widehat{\mathbf{b}}_i \equiv \widehat{\boldsymbol{D}}\boldsymbol{Z}_i'\widehat{\boldsymbol{V}}_i^{-1}(\mathbf{y}_i^{(obs)} - \boldsymbol{X}_i\widehat{\boldsymbol{\beta}}). \tag{13.50}$$

The conditional expectations are often called *empirical Bayes* (EB) estimates, because they are obtained by using the estimated values of the fixed parameters $\boldsymbol{\beta}$ and variance-covariance parameters $\boldsymbol{\theta}$ in (13.20). Note that, strictly speaking, the random effects $\mathbf{b}_i$ are not parameters, so that rather than estimating their values,

we are predicting them. Following this convention, the conditional expectations (13.50) might be called "predictors." In fact, they are often referred to as *best linear unbiased predictors* (BLUPs) or *empirical BLUPs* (EBLUPs). This term follows from the fact that it can be shown that the conditional expectations are BLUPs of $\mathbf{b}_i$ in the sense that they are unbiased and have minimum variance among all unbiased estimators, which are linear combinations of $\mathbf{y}_i$ (see, e.g., Verbeke and Molenberghs (2000, Sect. 7.4)). In what follows, we will be referring to the random-effects predictors, given by (13.50), as EBLUPs.

Similarly to (13.23), shrinkage of EBLUPs can be illustrated by noting that the following inequality,

$$\text{var}(\boldsymbol{\lambda}'\widehat{\mathbf{b}}_i) \leq \boldsymbol{\lambda}'\widehat{\mathcal{D}}\boldsymbol{\lambda}, \tag{13.51}$$

is true for any linear transformation $\boldsymbol{\lambda}$. We refer to this inequality in Fig. 17.1 and Panel R19.7.

It appears that using histograms or Q-Q plots of the predicted random errors for the purpose of checking their normality is of limited value. That is because the observed distribution of $\widehat{\mathbf{b}}_i$ does not necessarily reflect the true distribution of $\mathbf{b}_i$ (Verbeke and Molenberghs 2000, Sec.7.8.1). However, the plots of the conditional modes can be used to detect, e.g., outlying values that might warrant further inspection. Also, if the histogram is, e.g., bimodal, it may indicate that a covariate has been omitted from the $\boldsymbol{Z}_i$ matrix.

In practice, checking the normality assumption for $\mathbf{b}_i$ should be based on the comparison of the results obtained for a LMM with and without assuming the normality (Verbeke and Molenberghs 2000, Sec.7.8.4). This requires software for fitting LMMs with relaxed distributional assumptions about the random effects. Such an approach will not be presented in our book.

It is worth noting, however, that if the inferential goal focuses on the marginal model (13.26), and especially on the fixed effects $\boldsymbol{\beta}$, valid inference can be obtained even if the random effects do not follow a normal distribution (Verbeke and Molenberghs 2000, Sec.7.8.4).

13.6.2 Residual Diagnostics

The main tools for checking the assumption of the normality of residual errors $\boldsymbol{\varepsilon}_i$ are based on residuals. Note that, given the structure of the classical LMM, defined in (13.1)–(13.4), various types of raw residuals can be defined.

One set is the *conditional residuals*, which follow from the conditional mean representation (13.12), and are defined as

$$\widehat{\boldsymbol{\varepsilon}}_{(c)i} \equiv \mathbf{y}_i - \boldsymbol{X}_i\widehat{\boldsymbol{\beta}} - \boldsymbol{Z}_i\widehat{\mathbf{b}}_i, \tag{13.52}$$

where the formula for $\widehat{\mathbf{b}}_i$ is given in (13.50).

Another set is the *marginal residuals*, resulting from the marginal mean representation, given by (13.24). The marginal residuals are defined as

$$\widehat{\boldsymbol{\varepsilon}}_{(m)i} \equiv \mathbf{y}_i - \boldsymbol{X}_i\widehat{\boldsymbol{\beta}}. \tag{13.53}$$

The raw residuals are useful to check heterogeneity of the conditional or marginal variance. They are less recommended, however, for checking normality assumptions and/or detecting outlying observations. This is because, usually, raw residuals will be correlated and their variances will differ. Therefore, studentized and Pearson residuals are more often used (see Sects. 4.5.1 and 7.5). However, as in the case of the LM for correlated data (see Sect. 10.5), even the scaled residuals are not appropriate for, e.g., checking the normality of the residual errors. This is because the model (13.1)–(13.4) allows for a correlation between the errors. An approximate solution is to consider the transformation of the raw conditional or marginal residuals, which were defined in (13.52) and (13.53), respectively, based on the Cholesky decomposition of the (estimate of) residual variance-covariance matrix $\sigma^2\boldsymbol{R}_i$ or the marginal variance-covariance matrix $\sigma^2\boldsymbol{V}_i$, respectively (see Sects. 4.5.1 and 10.5). That is, to define

$$\widehat{\boldsymbol{\varepsilon}}^*_{(c)i} \equiv (\widehat{\sigma}\widehat{\boldsymbol{U}}'_{(c)i})^{-1}\widehat{\boldsymbol{\varepsilon}}_{(c)i}, \tag{13.54}$$

or

$$\widehat{\boldsymbol{\varepsilon}}^*_{(m)i} \equiv (\widehat{\sigma}\widehat{\boldsymbol{U}}'_{(m)i})^{-1}\widehat{\boldsymbol{\varepsilon}}_{(m)i}, \tag{13.55}$$

where the upper-triangular matrices $\widehat{\boldsymbol{U}}_{(c)i}$ and $\widehat{\boldsymbol{U}}_{(m)i}$ are defined by $\widehat{\boldsymbol{U}}'_{(c)i}\widehat{\boldsymbol{U}}_{(c)i} = \widehat{\boldsymbol{R}}_i$ and $\widehat{\boldsymbol{U}}'_{(m)i}\widehat{\boldsymbol{U}}_{(m)i} = \widehat{\boldsymbol{V}}_i$, respectively. Then, $\widehat{\boldsymbol{\varepsilon}}^*_{(c)i}$ (Pinheiro and Bates 2000, pp. 239) and $\widehat{\boldsymbol{\varepsilon}}^*_{(m)i}$ (Schabenberger 2004) should be approximately normally distributed with mean zero and variance-covariance matrix equal to an identity matrix. Thus, e.g., the normal Q-Q plot of the residuals should show approximately a straight line. Also, the scatterplot of the residuals against the estimated marginal mean values can be used to detect patterns suggesting a possible problem in the specification of the mean structure of the data or to check for outliers. Note that in R, in the **nlme** package, the transformed conditional residuals (13.54) are available.

Santos Nobre and da Motta Singer (2007) argue that the marginal residuals are *pure*, in the sense that the residuals are a function of only the marginal errors $\boldsymbol{\varepsilon}_{(m)i} \equiv \mathbf{y}_i - \boldsymbol{X}_i\boldsymbol{\beta}$, which they are supposed to estimate. On the other hand, the conditional residuals, which estimate the residual errors $\boldsymbol{\varepsilon}_i$, are *confounded* with the random effects $\mathbf{b}_i$, because the residuals are a function of $\mathbf{b}_i$ *and* $\boldsymbol{\varepsilon}_i$. For this reason, they suggest that the conditional residuals may not be suitable for checking, e.g., the normality of $\boldsymbol{\varepsilon}_i$. In particular, Santos Nobre and da Motta Singer (2007) recommend to use the plots of marginal residuals against covariates to check the linearity assumption for the covariates. On the other hand, the plots of the conditional residuals against the estimated conditional means $\widehat{\boldsymbol{\mu}}_i$ can be used to detect outlying observations or heteroscedasticity of the residual errors.

13.6.3 Influence Diagnostics

The basic tool to investigate the influence of a given observation on the estimates of $\boldsymbol{\beta}$, $\boldsymbol{\theta}$, and σ^2 is the likelihood displacement. It was introduced in the context of the classical LM in Sect. 4.5.3. Recall that the likelihood displacement, LD_i, as in (4.27), is defined as the change between the maximum log-likelihood computed when using all data and when excluding the i-th observation. For the LMM, given by (13.1)–(13.4), the likelihood-displacement definition (4.27) is modified by specifying $\widehat{\Theta} \equiv (\widehat{\boldsymbol{\beta}}', \widehat{\boldsymbol{\theta}}', \widehat{\sigma}^2)'$ and using the log-likelihood (13.27).

13.7 Inference and Model Selection

The inference for the classical LMM, specified by (13.1)–(13.4), focuses on the fixed-effect parameters $\boldsymbol{\beta}$ and/or the variance-covariance parameters $\boldsymbol{\theta}$. For these models, as described in Sect. 13.5.2, the estimation of the parameters uses the methods based on the marginal log-likelihood (13.27). Consequently, the inferential tools are very similar to those used for the LMs for correlated data. Thus, in what follows, we will frequently refer to the material contained in Sect. 10.6.

In Sect. 13.7.1, we describe statistical significance tests for the fixed effects, while in Sect. 13.7.2 we discuss the tests for variance-covariance parameters. Section 13.7.3 briefly discusses the construction of confidence intervals for the parameters of the model (13.1)–(13.4).

13.7.1 Testing Hypotheses About the Fixed Effects

Hypotheses about the parameters $\boldsymbol{\beta}$ are tested using the same methods that are applied for LMs for correlated data (see Sect. 10.6). In particular, linear hypotheses may be tested using the F-test, given by (4.36). The issue related to the computation of the degrees of freedom for the approximation of the distribution of the F-statistic by a central F distribution applies here as well. In the package **nlme**, this issue is ignored and the functions, available for fitting LMMs, and the null distribution of the F-statistic is crudely approximated with $\text{rank}(\boldsymbol{L})$ numerator and $n - p$ denominator degrees of freedom. In the package **lme4.0**, the issue is addressed using a Bayesian approach (see, e.g., Davidian and Giltinan (1995), Sect. 3.2.3) and by applying the Markov chain Monte Carlo (MCMC) technique to sample from the posterior distribution of the parameters (Baayen et al. 2008).

An alternative is to use ML-based LR tests (Sect. 7.6.1). Pinheiro and Bates (2000, Sect. 2.4.2) argue that the LR tests for hypotheses about $\boldsymbol{\beta}$ can be "anti-conservative", i.e., yield p-values smaller than those resulting from the postulated χ^2 distribution. For this reason they suggest to *condition* on the estimates of variance-covariance parameters $\boldsymbol{\theta}$ and use the F-tests. Based on the example they provided,

it appears, though, that the problem is more pronounced in the case when several fixed effects are tested at once and sample size is relatively small. Therefore, for some models considered in Chap. 18, fitted to a large sample size data, we in fact used the LR test to test the selected hypotheses about fixed effects.

Note that, instead of using a χ^2 distribution for an LR test, one could use an empirical distribution of the test statistic, obtained by fitting the alternative and null models to multiple datasets simulated under the null model (Pinheiro and Bates 2000, Sect. 2.4.1).

Finally, if the hypothesis about the parameters $\boldsymbol{\beta}$ cannot be expressed in a way that it would lead to alternative and null models, we can apply information criteria, like AIC or BIC (Sect. 4.7.2), to select the model that seems to best fit the data. Of course, strictly speaking, this is not a formal statistical testing approach. In this respect, it is also worth mentioning that the use of the log-restricted-likelihood-based criteria for LMMs with different mean structures is generally not advocated (see, e.g., Verbeke and Molenberghs (2000, Sect. 6.4)). For such cases, the use of the ML-based criteria is recommended. However, Gurka (2006) provides empirical arguments that this may not be necessarily a general recommendation. Thus, the issue is still debatable.

13.7.2 Testing Hypotheses About the Variance-Covariance Parameters

Similarly to the case of testing hypotheses about the parameters $\boldsymbol{\beta}$ (Sect. 13.7.1), inference about $\boldsymbol{\theta}$ uses the methods, which are applied for LMs for correlated data (Sect. 10.6). In particular, LR tests (Sect. 4.6.1) and information criteria (Sect. 4.7.2) are used for this purpose. The comments related to the need of the use of the REML-based tests apply to the LMMs as well. However, for the latter models, several additional issues need to be mentioned.

One issue concerns the distribution of the LR tests for testing null hypotheses about parameters $\boldsymbol{\theta}_D$, related to the matrix $\boldsymbol{D}$. The distribution depends on the type of the null hypothesis. In this respect, two cases can be considered. The first one pertains to the situation when the values of the variance-covariance parameters, compatible with the null hypothesis, do not lie on the boundary of the parameter space. This is, e.g., the case when we test a hypothesis that a correlation coefficient is equal to 0. In this case, the null distribution of the LR test is a χ^2 distribution with the number of degrees of freedom equal to the difference in the number of variance-covariance parameters between the null and alternative models.

The second case pertains to the situation when the values of the variance-covariance parameters, compatible with the null hypothesis, do lie on the boundary of the parameter space. In such situations, the null distribution of the LR test is not a χ^2 distribution. In certain cases (see, e.g., Verbeke and Molenberghs 2000, Sect. 6.3.4), it is possible to show that the null distribution is a mixture of several χ^2 distributions. As an example, consider the case of the model (13.1)–(13.4) with

only the group-level random effects, i.e., random intercepts. In this case, the vectors $\mathbf{b}_i \equiv b_i$ and $\boldsymbol{\theta}_D \equiv \theta_D$ are unidimensional (scalars), and $\boldsymbol{D} \equiv \theta_D$. Now, let us consider the null hypothesis, which specifies that no group-level random effects are needed. We can express the null hypothesis as $H_0 : \theta_D = 0$. The alternative is that a random effect is required, expressed as $H_A : \theta_D > 0$. Clearly, H_0 specifies a value of the variance parameter θ_D on the boundary of the parameter space, as variance cannot be negative. In this case, the results developed by Self and Liang (1987), Stram and Lee (1994), and Liang and Self (1996), suggested that the null distribution of the LR test statistic is a 50:50 mixture of χ^2_0 and χ^2_1 distributions. However, Crainiceanu and Ruppert (2004) show that the mixture is actually a conservative approximation to the finite-sample distribution of the LR test, which they derive.

It is important to note that the issue of testing a hypothesis on the boundary of the parameter space applies only when a fully hierarchical view of the classical LMM, specified by (13.1), (13.2), (13.10), and (13.4), is taken. When a purely marginal view, corresponding to (13.26), is adopted, the issue does not apply. Indeed, in the latter case, as it was argued in Sect. 13.5.4, $\boldsymbol{D}$ does not have to be positive-definite, as long as $\boldsymbol{\mathcal{V}}_i$, given in (13.25), is positive definite. This means that, in our example, values of $\theta_d \leq 0$ are possible for the alternative hypothesis as well. More details on this issue can be found in, e.g., Verbeke and Molenberghs (2003) and Molenberghs and Verbeke (2007).

If the null distribution of the LR test cannot be derived analytically, a potential solution is to use an empirical distribution obtained by fitting the alternative and null model to multiple datasets, with the dependent variable simulated under the null model.

Another issue is related to the approach based on the information criteria. The approach is used when the hypothesis about $\boldsymbol{\theta}$ cannot be expressed in the way that it would lead to alternative and null models. In this case, we can apply information criteria, like AIC or BIC (Sect. 4.7.2), to select the model that seems to best fit the data. Strictly speaking, this is not a formal statistical testing approach. Also, recent work (Gurka 2006) suggests that none of the information criteria is optimal to select LMMs, and that more work is still needed to understand the role that information criteria play in the selection of LMMs.

Obviously, irrespectively of the approach selected, before conducting any statistical significance tests, the fit of the chosen final model should be formally checked using the residual diagnostic methods described in Sect. 13.6.

13.7.3 Confidence Intervals for Parameters

Confidence intervals for the individual components of the parameter vector $\boldsymbol{\beta}$ can be constructed based on the t-distribution, used as an approximate distribution for the t-test statistic (see Sects. 4.6.2, 10.6, and 13.7.1). On the other hand, confidence intervals for the parameters $\boldsymbol{\theta}_R$, related to the matrix $\boldsymbol{R}_i$, and for σ can be obtained in the same way as it was described for the case of LMs for correlated data (Sect. 10.6).

Confidence intervals for parameters describing the structure of the matrix $\mathcal{D}$ can be obtained by considering a representation of the matrix in terms of variances (or standard deviations) and correlations. As explained in Sect. 13.5.4, application of the logarithmic transformation to the variances and Fisher's z-transform to the correlations yields a set of unconstrained parameters. After fitting an LMM, confidence intervals for the transformed parameters can be constructed using the normal approximation to the distribution of the ML or REML estimators (Sect. 10.6). The confidence intervals can then be back-transformed (see (7.33) in Sect. 7.6.2 and (10.41) in Sect. 10.6) to yield the corresponding intervals for variances (or standard deviations) and correlations.

13.8 Mean-Variance Models

In this section, we discuss the estimation approaches and inferential issues related to the use of the extended, mean-variance LMM.

As mentioned in Sect. 13.2, to define the model, we specify the conditional variance of residual errors with the help of a mean-dependent variance function, defined in (13.15), i.e., a function from the $<\delta, \mu>$- or $<\mu>$-group of variance functions (Sect. 7.3.1). The use of a mean-dependent variance function implies that, in the hierarchical model, defined by (13.1), (13.2), (13.10), and (13.4), residual errors and random effects for the same group are no longer independent. This violates the assumption of the classical LMM (13.1)–(13.4) and raises theoretical and computational issues. In this section, we describe the issues and possible solutions.

In particular, in Sect. 13.8.1, we focus on the single-level mean-variance LMM. Section 13.8.2 briefly describes the formulation of multilevel mean-variance LMMs. In Sect. 13.8.3, issues related to the inference, model diagnostics, and other aspects of the use of the mean-variance LMMs are summarized.

13.8.1 Single-Level Mean-Variance Linear Mixed-Effects Models

In Sect. 13.4, we pointed out that the marginal distribution for the classical LMM was a normal distribution of the form given in (13.26). Thus, the estimation of the model could be based on the use of the likelihood functions derived from the normal distribution (Sect. 13.5.2).

However, for the mean-variance LMM, defined by (13.1), (13.2), (13.10), (13.4), and (13.15), the marginal distribution does not have a closed form expression. In fact, it is not obvious that it is a normal distribution. Thus, the estimation approaches, described in Sect. 13.5.2, cannot be applied.

To estimate mean-variance LMMs, we might use the fact that, following (13.10) and (13.16),

$$\text{Var}(\boldsymbol{\varepsilon}_i) = \text{E}\left[\sigma^2 \boldsymbol{R}_i(\boldsymbol{\mu}_i, \boldsymbol{\theta}_R; \mathbf{v}_i)\right], \tag{13.56}$$

where the expected value is taken with respect to the distribution of the random effects $\mathbf{b}_i$, indicated in (13.10). Thus, the unconditional variance (13.56) does not depend on the random effects, which could, in principle, simplify the estimation of the model. However, analytical computation of the unconditional variance is, generally, not feasible, because the variance function (13.15) is usually nonlinear in $\mathbf{b}_i$.

A computationally feasible alternative is to estimate (13.56) by plugging-in suitable predictors of $\mathbf{b}_i$. Then, mean-variance LMMs can be estimated by algorithms similar to those presented in Sects. 7.8.1 and 10.7.

In particular, consider an LMM, defined by (13.1), (13.2), (13.10), (13.4), and (13.15), with a variance function from the $<\delta, \mu>$-group (see Table 7.3). Then, the following algorithm, similar to the one described in Sects. 7.8.1 and 10.7, can be used:

1. Assume an initial value $\widehat{\boldsymbol{\beta}}^{(0)}$ of $\boldsymbol{\beta}$, $\widehat{\boldsymbol{\theta}}^{(0)}$ of $\boldsymbol{\theta}$, $\widehat{\mathbf{b}}_i^{(0)}$ of $\mathbf{b}_i$, and set the iteration counter $k = 0$.
2. Increase k by 1.
3. *Use* $\widehat{\boldsymbol{\beta}}^{(k-1)}$ *to compute* $\widehat{\boldsymbol{\mu}}_i^{(k)}$ *and (re)define the matrix function* $\boldsymbol{V}_i^{(k)}(\boldsymbol{\theta})$.
   Calculate $\widehat{\boldsymbol{\mu}}_i^{(k)} \equiv \mathbf{x}_i\widehat{\boldsymbol{\beta}}^{(k-1)} + \mathbf{z}_i\widehat{\mathbf{b}}_i^{(k-1)}$ (see Sect. 13.6.1).
   (Re)define the variance function $\lambda^{(k)}(\boldsymbol{\delta}; \widehat{\mu}_{ij}^{(k)}) \equiv \lambda(\widehat{\mu}_{ij}^{(k)}, \boldsymbol{\delta})$.
   (Re)define diagonal elements of matrix function $\boldsymbol{\Lambda}^{(k)}(\boldsymbol{\delta}; \widehat{\boldsymbol{\mu}}_i^{(k)})$.
   (Re)define the matrix function $\boldsymbol{R}_i^{(k)}(\boldsymbol{\theta}_R) \equiv \boldsymbol{\Lambda}^{(k)}(\boldsymbol{\delta}; \widehat{\boldsymbol{\mu}}_i^{(k)})\mathbf{C}(\varrho)\boldsymbol{\Lambda}^{(k)}(\boldsymbol{\delta}; \widehat{\boldsymbol{\mu}}_i^{(k)})$, and $\boldsymbol{V}_i^{(k)}(\boldsymbol{\theta}) \equiv \boldsymbol{Z}_i\boldsymbol{D}(\boldsymbol{\theta}_D)\boldsymbol{Z}_i' + \boldsymbol{R}_i^{(k)}(\boldsymbol{\theta}_R)$.
4. *Keep* $\widehat{\boldsymbol{\beta}}^{(k-1)}$ *fixed and compute* $\widehat{\boldsymbol{\theta}}^{(k)}$.
   While keeping the value $\widehat{\boldsymbol{\beta}}^{(k-1)}$ of $\boldsymbol{\beta}$ fixed and using the matrix functions $\boldsymbol{V}_i^{(k)}(\boldsymbol{\theta})$, compute an estimate $\widehat{\boldsymbol{\theta}}^{(k)}$ of $\boldsymbol{\theta}$ by maximizing an appropriate log-profile-likelihood.
5. *Keep* $\widehat{\boldsymbol{\theta}}^{(k)}$ *fixed. Compute* $\boldsymbol{\beta}^{(k)}$ *and* $\widehat{\mathbf{b}}_i^{(k)}$.
   Based on the formula in step 3, compute the matrices $\boldsymbol{V}_i^{(k)}(\widehat{\boldsymbol{\theta}}^{(k)})$ and use them to obtain estimates $\widehat{\boldsymbol{\beta}}^{(k)}$ of $\boldsymbol{\beta}$ from (13.28) and $\widehat{\mathbf{b}}_i^{(k)}$ of $\mathbf{b}_i$ from (13.50).
6. Iterate between steps 2 and 5 until convergence or until a predetermined number of iterations k.
7. Compute the estimate of σ^2 from (13.29) using the estimates of $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$.

An ML-based version of the algorithm is obtained by using, in step 4, the fixed value $\widehat{\boldsymbol{\beta}}^{(k-1)}$ of $\boldsymbol{\beta}$ and the redefined form of the matrices $\boldsymbol{V}_i^{(k)}(\boldsymbol{\theta})$ to express σ^2 as in (13.29). The resulting expression for σ^2 is then plugged in (13.27), and the estimate $\widehat{\boldsymbol{\theta}}^{(k)}$ of $\boldsymbol{\theta}$ is computed by maximizing the so-obtained log-profile-likelihood.

A REML-based version of the algorithm results from using, in step 4, the log-profile-restricted-likelihood (13.31) to compute the estimate $\widehat{\boldsymbol{\theta}}^{(k)}$ of $\boldsymbol{\theta}$ (Sects. 7.8.1 and 10.7). Then, in step 7, σ^2 is computed from (13.30).

Note that the algorithm involves two iterative loops: the "external" one, related to the computation of the values $\widehat{\boldsymbol{\beta}}^{(k)}$ of $\boldsymbol{\beta}$ and $\widehat{\boldsymbol{\mu}}_i^{(k)}$ of $\widehat{\boldsymbol{\mu}}_i$, and the "internal" one, related to the computation (in step 4) of the value $\widehat{\boldsymbol{\theta}}^{(k)}$ of $\boldsymbol{\theta}$. It is also worth noting that, in step 3, the redefined variance function $\lambda^{(k)}(\cdot)$ does not depend on μ_{ij}, and therefore it belongs to the $<\delta>$-group. This allows for the use of the marginal-model-based log-(profile)-likelihood functions, like (13.31), in step 4.

LMMs, defined by (13.1), (13.2), (13.4)–(13.10), and (13.15), with a variance function that depends only on μ_{ij}, i.e., which belongs to the $<\mu>$-group (see Table 7.4), can be estimated by using an IRLS procedure similar to the one described in Sect. 7.8.1, with obvious modifications.

Algorithms equivalent to the PL-GLS and IRLS procedures, described above, can be formulated for the PnLS estimation technique (Sect. 13.5.3), resulting in a *penalized, iteratively re-weighted, least squares* (PnIRLS) approach (Bates 2012).

13.8.2 Multilevel Hierarchies

To extend the mean-variance formulation to, e.g., two-level LMMs, which were defined in (13.5), the definition of the variance function, given in (13.15), needs to be modified. In particular, we can assume that

$$\mathrm{Var}(\varepsilon_{ijk} \mid \mathbf{b}_i, \mathbf{b}_{ij}) = \sigma^2 \lambda^2(\mu_{ijk}, \boldsymbol{\delta}; \mathbf{v}_{ijk}), \tag{13.57}$$

where

$$\mathrm{E}(y_{ijk} \mid \mathbf{b}_i, \mathbf{b}_{ij}) \equiv \mu_{ijk} = \mathbf{x}'_{ijk}\boldsymbol{\beta} + \mathbf{z}'_{1,ijk}\mathbf{b}_i + \mathbf{z}'_{2,ijk}\mathbf{b}_{ij}, \tag{13.58}$$

where $\mathbf{x}_{ijk}$, $\mathbf{z}_{1,ijk}$, and $\mathbf{z}_{2,ijk}$ are column vectors containing the values of the x-, z_1-, and z_2-covariates for the k-th observation from the j-th subgroup of the i-th group.

The estimation algorithms, described in Sect. 13.8.1 for a single-level LMM, can be adapted to the two-level model case. Needless to say, they become more involved numerically.

13.8.3 Inference

For the mean-variance LMMs, estimates of $\boldsymbol{\beta}$ are asymptotically approximately normally distributed with a variance-covariance matrix, which can be estimated as in (13.49). Consequently, tests for linear hypotheses and CIs for the elements of $\boldsymbol{\beta}$ can be obtained along the lines described in Sect. 13.7.1. Note, however, that

the use of LR tests is problematic given the fact that the algorithms described in Sect. 13.8.1 are not likelihood-based. It should also be born in mind that, especially for small sample sizes, standard errors computed from (13.49) may be too small, as the precision of estimation of $\boldsymbol{\beta}$ is influenced by the precision of the estimation of $\boldsymbol{\theta}$ (see Sects. 7.8.2 and 10.7).

Inference on the parameters $\boldsymbol{\theta}$ and σ^2 is complicated by problems similar to those described in Sects. 7.8.2 and 10.7. Thus, we do not discuss it further here.

13.9 Chapter Summary

In this chapter, we reviewed the essential concepts and methods underlying the formulation of an LMM for hierarchical data. In our presentation, we were focusing on the theoretical constructions, which are linked to the implementation of LMMs in R. Readers interested in a more detailed account of the theory of LMMs are referred to the monographs mentioned in the introduction to this chapter.

In Sect. 13.2, we described the formulation of the classical LMM, while in Sect. 13.3 we discussed the formulation of the extended model. Details of the formulation of both types of LMMs were discussed in Sect. 13.4. In the formulation, the concepts of variance function and correlation structure, developed in Chaps. 7 and 10, respectively, were used. An additional, novel element was the introduction of the random effects in the mean structure of the model. The use of the random effects allows to directly address the hierarchical structure of the data.

Sections 13.5, 13.6, and 13.7 were devoted to, respectively, the estimation approaches, diagnostic tools, and inferential methods used for the classical LMM. This type of LMMs is most commonly used in practice. In Sect. 13.8, we described the estimation and inferential techniques used for the extended LMM defined using a mean-dependent variance function.

To the extent possible, we used in our presentation the concepts and theory introduced in Chaps. 4, 7, and 10. Especially relevant was the material from Chap. 10, because, as mentioned in Sect. 13.5, the estimation of classical LMM is based primarily on the marginal likelihood or restricted likelihood functions, which are special cases of the likelihood functions presented in Sect. 10.4.2. We note that, if the missing at random (MAR) assumption about missing data mechanism is tenable, the ML estimation for the classical linear mixed-effect models yields valid estimates. Thorough discussion of this important topic can be found in Verbeke and Molenberghs (2000).

As compared to LMs for correlated data, described in Part III of the book, LMMs address directly the hierarchy present in grouped data. They allow drawing conclusions about the partition of the total variability of observations between the different levels of the hierarchy. This additional insight can be considered as an advantage of LMMs. On the other hand, as mentioned in Sect. 13.5.1, LMs for correlated errors are more flexible than LMMs. Thus, the choice between them depends on the goals of a particular analysis.

In the next two chapters, we describe the tools available for fitting LMMs in R.

Chapter 14
Fitting Linear Mixed-Effects Models: The `lme()` Function

14.1 Introduction

In Chap. 13, we summarized the main theoretical concepts underlying the construction of LMMs. Compared to the LMs introduced in Chaps. 4, 7, and 10, LMMs allow taking the hierarchical structure of data into account in the analysis. This is achieved by introducing, in addition to the mean (fixed-effects) structure, a random-effects structure.

There are several packages in R, which contain tools for fitting LMMs, like, e.g., **nlme**, **lme4.0**, or **MCMCglmm**. In the current chapter, we describe the use of the popular and well-established package **nlme**. The primary tool to fit LMMs in this package is the function `lme()`. In the next chapter, we will describe the use of the package **lme4.0**. Note that both packages allow to fit GLMMs and NLMMs, but these models are outside of the scope of this book.

The chapter is organized as follows. In Sect. 14.2, we describe objects of class *pdMat*, which represent positive-definite matrices. In particular, the class is used to represent variance–covariance matrices of random effects. In Sect. 14.3, we describe the class *reStruct*, used to represent the random-effects structure of an LMM. The random part of an LMM is represented using the *lmeStruct* class described in Sect. 14.4. All the aforementioned classes are related to the function `lme()`, which is the key function in the **nlme** package to fit LMMs. The use of the function is reviewed in Sect. 14.5. On the other hand, in Sect. 14.6, we summarize the methods, which allow extracting information from model-fit objects of class *lme*. Section 14.7 is devoted to the implementation of inferential tools for LMMs. A summary of the chapter is provided in Sect. 14.8.

As the basic example, we use the classical, single-level LMM, defined by (13.1)–(13.4). However, to illustrate the features of the syntax, we refer, in a few instances, to the two-level LMM, specified in (13.5).

A. Gałecki and T. Burzykowski, *Linear Mixed-Effects Models Using R: A Step-by-Step Approach*, Springer Texts in Statistics, DOI 10.1007/978-1-4614-3900-4_14,

14.2 Representation of a Positive-Definite Matrix: The *pdMat* Class

As compared to the LMs, introduced in Chaps. 4, 7, and 10, an important, new component of LMMs is the random-effects structure. By the random-effects structure, we mean the levels of the model hierarchy, the $\mathbf{Z}_i$ design matrices for the random effects $\mathbf{b}_i$, and the parameterized form of the matrix $\boldsymbol{D}$, as defined in Equations (13.1)–(13.4) for the classical, single-level LMM. In this section, we provide details of the implementation of various forms of positive-definite matrices available in the package **nlme**. The forms are used for specifying the matrix $\boldsymbol{D}$.

In particular, in Sect. 14.2.1, we describe the classes of such matrices and the corresponding constructor functions, while in Sect. 14.2.2, we discuss the methods to extract information from the objects constructed with the help of the functions.

14.2.1 Constructor Functions for the pdMat Class

Positive-definite matrices are represented in the package **nlme** by objects inheriting from the *pdMat* class. By issuing the `?pdClasses` command at the command prompt we obtain a list of standard classes of *pdMat* structures. We itemize them below for the reader's reference:

pdIdent	a multiple of identity
pdDiag	a diagonal matrix
pdCompSymm	compound symmetry
pdLogChol	a general positive-definite matrix using the log-Cholesky parameterization
pdSymm	a general positive-definite matrix with a parameterization based on SVD
pdNatural	a general-positive-definite matrix with the "natural" parameterization, i.e., in terms of standard deviations and correlations
pdBlocked	a blocked-diagonal matrix, with blocks defined by structures/classes defined above

The classes listed above are ordered roughly according to the increasing order of complexity of the represented matrix structures. The main difference between the *pdLogChol*, *pdSymm*, and *pdNatural* classes, which all represent a general variance–covariance matrix, lies in the used parameterization (Sect. 13.5.4) and will be illustrated in Sect. 14.2.2.

The constructor function, used to create or to modify objects that inherit from a particular class, is named after the corresponding class. For example, the `pdDiag()` function creates/modifies an object of class *pdDiag*. Note that the created object inherits also from the *pdMat* class. The *pdMat* constructors are primarily used in the specification of the random-effects structure of an LMM, with the help of the `random` argument of the model-fitting function `lme()`.

In the next section, we describe the arguments of the constructor functions.

14.2.1.1 Arguments of the Constructor Functions

The arguments of the *pdMat* constructor functions are `value`, `form`, `nam`, and `data`. We will focus on the `value` and `form` arguments. Arguments `data` and `nam` are merely used to assign names to the rows and columns of a positive-definite matrix. As such, they are less important, so we do not describe them. A description of all the arguments for a specific *pdMat* class is obtained by issuing a command like, e.g., `?pdSymm`.

The argument `form` is simply an optional one-sided formula. When used together with the `data` argument, the formula is evaluated and, subsequently, the appropriate names are assigned to the rows/columns of the matrix represented by the object. By default, the value of the `form` argument is `NULL`. If the `value` argument contains a one-sided formula, the argument `form` is ignored.

Although `value` is the main argument of a *pdMat* constructor function, we describe it as the last, because it can be used to specify the components of *pdMat* objects, defined by the arguments described above. The main role of this argument is to assign coefficients to *pdMat* objects by supplying a positive-definite matrix or a numeric vector. Other possible values of the `value` argument include: a *pdMat*-class object, a one-sided linear formula, or a vector of character strings. By default, its value is `numeric(0)`, which results in an uninitialized object.

The code in Panel R14.1 presents examples of application of constructor functions `pdCompSymm()` and `pdSymm()` to create objects of class *pdCompSymm* and *pdSymm*, respectively, which inherit from the *pdMat* class.

In Panel R14.1a, the `pdCompSymm()` function applies the argument `value` in the form of a one-sided formula `~agex`. Thus, it does not assign any numeric values. Consequently, the resulting object `pdCS0` of class *pdCompSymm*, representing a compound-symmetry matrix with constant diagonal and off-diagonal elements, is uninitialized.

In Panel R14.1b, the object `mtxUN` is a positive-definite, 2×2 matrix, while `dt1` is an auxiliary data frame with a single numeric variable `agex` with four observations. The `pdSymm()` function uses the matrix `mtxUN` as the `value` argument. Additionally, it specifies the one-sided formula `~agex` as the `form` argument and uses the data frame `dt1` to evaluate the variable `agex`. The resulting object `pdSm` of class *pdSymm*, which represents a general positive-definite matrix, is initialized.

To explain the names of the rows and columns of the matrix, contained in the object `pdSm`, we note that the formula `~agex`, used in Panel R14.1b, assumes the presence of an intercept and is equivalent to the formula `~1 + agex`. It follows that object `pdSm` can be interpreted as the variance–covariance matrix of a vector of random effects generated by the formula. Given that the variable `agex` is numeric (continuous), the formula implies the use of random intercepts and of random slopes (for `agex`). Hence, the use of the names `(Intercept)` and `agex` for the rows and columns of the matrix, contained in the object `pdSm`.

In Panel R14.1c, we illustrate how to construct an initialized object of class *pdCompSymm*. The object `mtxCS` is a positive-definite, 3×3 compound-symmetry

R14.1 *R syntax*: Creating objects inheriting from the *pdMat* class

(a) *Uninitialized object of class pdCompSymm*

```
> library(nlme)
> (pdCS0    <- pdCompSymm(~agex))
  Uninitialized positive definite matrix structure of class pdCompSymm
> isInitialized(pdCS0)                                  # Not initialized
  [1] FALSE
```

(b) *Initialized object of class pdSymm*

```
> mtxUN    <- matrix(c(4, 1, 1, 9), nrow = 2)         # pdSymm matrix
> dt1      <- data.frame(agex = c(15, 45, 71, 82))    # Numeric age
> (pdSm    <- pdSymm(mtxUN, ~agex, data = dt1))
  Positive definite matrix structure of class pdSymm representing
              (Intercept) agex
  (Intercept)           4    1
  agex                  1    9
> isInitialized(pdSm)                                 # Initialized
  [1] TRUE
```

(c) *Initialized object of class pdCompSymm*

```
> mtxCS    <- matrix(4 * diag(3) + 1, nrow = 3)       # CompSymm matrix
> dt2      <- data.frame(agef=c("Y", "M", "O", "O")) # Factor age
> (pdCSf   <- pdCompSymm(mtxCS, ~-1 + agef, data = dt2))
  Positive definite matrix structure of class pdCompSymm representing
        agefM agefO agefY
  agefM     5     1     1
  agefO     1     5     1
  agefY     1     1     5
```

matrix, with all diagonal elements equal to 5 and all off-diagonal elements equal to 1. The auxiliary object `dt2` is a data frame with a single-variable `agef`, which is a factor with three levels and four observations. The `pdCompSymm()` constructor function uses the object `mtxCS` as the `value` argument. Additionally, it specifies the one-sided formula `~-1+agef` in the argument `form` and uses the data frame `dt2` to evaluate the variable `agef`. The resulting object `pdCsf` of class *pdCompSymm*, which represents a compound-symmetry matrix, is initialized. Note that its row and column names are defined by the levels of the factor `agef`. This is because the formula, used in the `form` argument, does not use an intercept and includes the factor `agef` (Sect. 5.2.1). Thus, the formula implies the use of three random effects, associated with the levels of the factor. Hence, the use of the factor-level names for

the rows and columns of `pdCSf`. Note that the names are obtained by referring to the data frame `dt2`, which was provided in the argument `data`. Had the formula been changed to, e.g., `~agef`, it would have implied the use of an intercept, and the name of the first row and column of the object `pdSm` would have been changed to `(Intercept)`. The names of the remaining two rows and columns would remain unchanged.

In many cases, e.g., when specifying an LMM, it is sufficient to work with uninitialized *pdMat* objects, such as the one defined in Panel R14.1a. Initialized objects, i.e., objects with defined, known numerical values, such as those shown in Panels R14.1b and R14.1c, can be useful if initial values for coefficients of a positive-definite matrix need to be specified for a model-fitting routine.

14.2.2 Inspecting and Modifying Objects of Class pdMat

A list of methods available for probing and modifying objects of class *pdMat* is obtained by issuing the command `methods(class="pdMat")`. In Panel R14.2, we present examples of the use of selected methods, which can be applied to extract information from such objects.

More specifically, in Panel R14.2a, we present methods for an object of class *pdSymm*. As an example, we use the object `pdSm` defined in Panel R14.1. The `summary()` function displays the standard deviations and correlations associated with the positive-definite matrix represented by the object. Specific attributes of the object can be displayed using appropriate functions like, e.g., `formula()`, `Names()`, or `Dim()`. The function `logDet()` prints out the value of the logarithm of the determinant of the Cholesky factor of the positive-definite matrix represented by the *pdSymm*-class object.

In Panel R14.2b, we show the results of the application of selected methods to the object `pdCSf` of class *pdCompSymm*. Essentially, all the methods used in Panel R14.2a could be applied to the object as well.

Panel R14.3 is devoted entirely to the use of the `coef()` method for extracting coefficients of an initialized *pdMat*-class object. The method returns a vector with coefficients associated with the object. It allows for an optional logical argument `unconstrained`. If `unconstrained=FALSE`, a vector of constrained coefficients is returned (Sect. 13.5.4). Depending on the class of the object, the vector may contain upper-triangular elements of the positive-definite matrix represented by a *pdSymm*-class object or the standard deviation and correlation coefficient corresponding to a compound-symmetry matrix from a *pdCompSymm*-class object. If `unconstrained=TRUE`, the coefficients are returned in unconstrained form, suitable for the optimization purposes. By default, `unconstrained=TRUE`.

In Panel R14.3a, we extract coefficients from the object `pdSm`. This is an object of class *pdSymm*, i.e., it represents a positive-definite matrix of a general form. First, by specifying the argument `unconstrained=FALSE`, we obtain the upper-triangular elements of the matrix. Subsequently, using the default value of

R14.2 *R syntax*: Probing objects inheriting from the *pdMat* class. Objects pdSm and pdCSf were created in Panel R14.1

(a) *Extracting information from the object pdSm of class pdSymm*

```
> summary(pdSm)                # Summary
   Formula: ~agex
   Structure: General positive-definite
              StdDev Corr
  (Intercept) 2      (Intr)
  agex        3      0.167
> formula(pdSm)                # Formula
  ~agex
> Names(pdSm)                  # Row/col names
  [1] "(Intercept)" "agex"
> (Dmtx <- as.matrix(pdSm))    # D matrix
              (Intercept) agex
  (Intercept)           4    1
  agex                  1    9
> Dim(pdSm)                    # Dimensions of D
  [1] 2 2
> logDet(pdSm)                 # log|D^1/2|
  [1] 1.7777
> # VarCorr(pdSm)              # Variances, correlation coefficients
> # corMatrix(pdSm)            # Corr(D)
```

(b) *Extracting information from the object pdCSf of class pdCompSymm*

```
> Names(pdCSf)                  # Row/col names
  [1] "agefM" "agef0" "agefY"
> as.matrix(pdCSf)              # D matrix
        agefM agef0 agefY
  agefM     5     1     1
  agef0     1     5     1
  agefY     1     1     5
```

the argument, we obtain the unconstrained coefficients, which result from applying the matrix-logarithm transformation (Sect. 13.5.4). We will explain the computation of the unconstrained coefficients in Panel R14.4.

Panel R14.3b illustrates the method of obtaining coefficients from the object pdCSf. The object is of class *pdCompSymm* and represents a compound-symmetry matrix. First, by setting the argument unconstrained=FALSE, we obtain the standard deviation and the correlation coefficient, which define the compound-symmetry

R14.3 *R syntax*: Extracting coefficients from an object inheriting from a *pdMat* class. Objects `pdSm` and `pdCSf` were created in Panel R14.1

(a) *Extracting coefficients from the object `pdSm` of class pdSymm*

```
> coef(pdSm, unconstrained = FALSE)   # Constrained coefficients
       var((Intercept)) cov(agex,(Intercept))
                      4                      1
              var(agex)
                      9
> coef(pdSm)                          # Unconstrained coefficients
  [1] 0.68424 0.08184 1.09344
```

(b) *Extracting coefficients from the object `pdCSf` of class pdCompSymm*

```
> coef(pdCSf, unconstrained = FALSE)  # Constrained coefficients
  std. dev    corr.
    2.2361   0.2000
> coef(pdCSf)                         # Unconstrained coefficients
  [1]  0.80472 -0.13353
> log(5)/2                            # First coefficient verified
  [1] 0.80472
> rho <- 0.2                          # ϱ
> nc  <- 3                            # No. of columns
> aux <- (rho + 1/(nc - 1))/(1 - rho) # Modified Fisher's z: (10.35)
> log(aux)                            # Second coefficient verified
  [1] -0.13353
```

structure. The use of the default value of the argument, `TRUE`, results in two coefficients. The first one is the logarithm of the standard deviation. The second one is the modified Fisher's z-transform (10.35) of the correlation coefficient. The computations of the values of the two unconstrained coefficients are verified at the end of Panel R14.3b.

In Panel R14.4, we illustrate different parameterizations of a general positive-definite matrix, represented by different *pdMat* classes. The parameterizations were described in Sect. 13.5.4.

First, in Panel R14.4a, we show explicitly the link between the unconstrained coefficients of an object of class *pdSymm* and the logarithm of a positive-definite matrix (Sect. 13.5.4). Toward this end, we create the object `pdSm0` of class *pdSymm* from the matrix `mtxUN` and we list the unconstrained coefficients by applying the `coef()` method to the object. Next, by applying the function `pdMatrix()`, we obtain the positive-definite matrix, represented by the object, and store it in the object `Dmtx`. With the help of the `chol()` function, we compute the Cholesky

R14.4 *R syntax*: Various unconstrained parameterizations of a general positive-definite (variance–covariance) matrix. The matrix `mtxUN` was created in Panel R14.1

(a) *The matrix-logarithm parameterization – pdSymm class*

```
> pdSm0 <- pdSymm(mtxUN)
> coef(pdSm0)                        # Unconstrained θ_D
  [1] 0.68424 0.08184 1.09344
> Dmtx <- pdMatrix(pdSm0)            # Matrix D
> CholD <- chol(Dmtx)                # Cholesky factor U of D: D= U'U
> vd <- svd(CholD, nu=0)             # SVD of U: (13.46)
> vd$v %*% (log(vd$d) * t(vd$v))     # (13.47)
          [,1]    [,2]
  [1,] 0.68424 0.08184
  [2,] 0.08184 1.09344
```

(b) *The log-Cholesky parameterization – pdLogChol class*

```
> pdLCh <- pdLogChol(mtxUN)
> coef(pdLCh)                        # Unconstrained coefficients θ_D
  [1] 0.69315 1.08453 0.50000
> LChD  <- CholD                     # U
> diag(LChD) <- log(diag(LChD))      # diag(U) log-transformed
> LChD
          [,1]   [,2]
  [1,] 0.69315 0.5000
  [2,] 0.00000 1.0845
```

(c) *The "natural" parameterization – pdNatural class*

```
> pdNat <- pdNatural(mtxUN)
> coef(pdNat)                        # Unconstrained θ_D
  [1] 0.69315 1.09861 0.33647
> log(sqrt(diag(Dmtx)))              # log(SDs)
  [1] 0.69315 1.09861
> corD  <- cov2cor(Dmtx)             # Corr(D)
> rho   <- corD[upper.tri(corD)]     # ϱ_ij (for i < j)
> log((1+rho)/(1-rho))               # Fisher's z: (10.33)
  [1] 0.33647
```

decomposition of the matrix `Dmtx` and store the resulting Cholesky factor in the object `CholD`. Then, we apply the function `svd()` to compute SVD of `CholD`. The components of the decomposition are stored in the object `vd`. By extracting the components `vd$v` and `vd$d`, we compute the logarithm of the matrix `CholD`.

The upper-triangular elements of the resulting matrix-logarithm correspond to the unconstrained coefficients obtained by applying the `coef()` method to the object `pdSm0`. It is worth mentioning that the matrix logarithm of the matrix `Dmtx` can be obtained by simply doubling the elements of the matrix logarithm of the matrix `CholD`.

In Panel R14.4b, we show the *pdLogChol* representation of the matrix `mtxUN`. Toward this end, we apply the `pdLogChol()` constructor function to `mtxUN` and display the resulting unconstrained coefficients using the `coef()` method. The representation is based on the Cholesky decomposition of the matrix, obtained with the requirement that the diagonal elements of the resulting Cholesky factor are positive (Sect. 13.5.4). The coefficients are obtained from the elements of the Cholesky factor matrix, but with the diagonal elements replaced by their logarithms. To illustrate the computation of the coefficients explicitly, we reuse the matrix `CholD`, created in Panel R14.4a. Then, we replace the diagonal elements of the resulting matrix, `LChD`, by their logarithms. The upper-triangular elements of the so-obtained matrix correspond to the unconstrained coefficients of the *pdLogChol* representation.

Finally, in Panel R14.4c, we present the *pdNatural* representation of the matrix `mtxUN`. Toward this end, we apply the `pdNatural()` constructor function. The representation is based on the use of standard deviations and correlation coefficients, which correspond to `mtxUN`. The coefficients are obtained by log-transforming the standard deviations and by applying Fisher's z-transform to the correlation coefficients (Sect. 13.5.4). The transformations are shown explicitly in this subpanel. In the process, the function `cov2cor()` is used to compute the correlation matrix, corresponding to `mtxUN` (Sect. 12.3.2), while the function `upper.tri()` is applied to define the correlation coefficients as the upper-triangular elements of the computed correlation matrix.

Following the discussion, presented in Sect. 13.5.4, and the description given above, it is clear that the *pdLogChol*- and *pdSymm*-class representations are suitable for the numerical optimization purposes. On the other hand, the representation used in the *pdNatural* class does not guarantee that the represented matrix is positive definite. Thus, it should not be used in numerical optimization. However, it is suitable for the construction of the confidence intervals for the elements of the matrix, as explained in Sect. 13.7.3.

14.3 Random-Effects Structure Representation: The *reStruct* class

As mentioned in Sect. 14.2, the random-effects structure of an LMM includes the information about the levels of the model hierarchy, the $\mathbf{Z}_i$ design matrices, and the parameterized form of the matrix (or matrices) $\boldsymbol{D}$.

In the package **nlme**, the structure is represented by specialized list-objects of class *reStruct*. Every component of the list is in itself an object of class *pdMat*, corresponding to an appropriate level of model hierarchy.

14.3.1 Constructor Function for the reStruct Class

The function `reStruct()` is a constructor function for an object of class *reStruct*. The arguments of the function include `object`, `pdClass`, `REML`, `data`, `x`, `sigma`, `reEstimates`, and `verbose`. Description of these arguments can be obtained by issuing the command `?reStruct`.

The argument `object` is the most important one. We will describe its use in more detail, because the syntax is very similar to that of the `random` argument of the `lme()` function, which is the key function to fit LMMs in the package **nlme**. The syntax of the `lme()` function will be described in Sect. 14.5.

The essential role of the `object` argument is to pass the information necessary for the specification of the random-effects structure. In particular, the argument is used to provide the information about the model hierarchy and about the formulae associated with the `pdMat` objects, which are later used to create the design matrices $\boldsymbol{Z}_i$. In addition, the argument can be used to specify the information about the structure of the matrix (or matrices) $\boldsymbol{D}$, including the values of their elements.

In Table 14.1, we provide examples of four forms of the syntax that can be used for the argument `object` of the `reStruct()` constructor function. To maintain generality of the presentation, the examples are given for a hypothetical, two-level LMM, as defined in (13.5). We assume that the two levels of grouping are defined by grouping factors `g1` and `g2`. The variables `z1` and `z2`, together with random intercepts, are used as random-effects covariates at the grouping levels defined by `g1` and `g2`, respectively.

All forms of the syntax, shown in Table 14.1, allow a direct specification of the hierarchical structure of the model using grouping factors, such as `g1` and `g2` in our example. However, they differ in the flexibility of specifying other components of the random-effects structure. To illustrate the differences, we consider the use of the variables `z1` and `z2` to introduce random effects associated with covariates.

In Table 14.2, we point to the limitations of the different forms of the syntax, which were presented in Table 14.1. In part (a) of the table, we present an example of syntax for a single-level LMM, with grouping defined by the factor `g1` and a single random-effects covariate `z1`. In part (b) of the table, we show the four forms of the syntax for the same setting as in Table 14.1, i.e., for a two-level LMM.

The syntax (a) is the most flexible. It essentially allows incorporating the information about all components of the random-effects structure, which are supported by the `lme()` function. In particular, for a two-level LMM (see (13.5)), it allows specifying different structures of the $\boldsymbol{D}$ matrices at different levels of the model hierarchy. In the example presented in Table 14.1, the different matrix structures are represented by objects of classes *pdSymm* and *pdDiag*. That is, the matrix $\boldsymbol{D}_1$ is assumed to have a general form, while the matrix $\boldsymbol{D}_2$ is assumed to be diagonal.

Table 14.1 *R syntax*: Syntax[a] for the argument `object` of the `reStruct()` constructor function

Syntax form	Description
(a)	List with named components of class *pdMat* and with grouping factors[b,c] used as names of the components, e.g., `list(g1 = pdSymm(~z1), g2 = pdDiag(~z2))`
(b)	Unnamed list of one-sided formulae with | operator, e.g., `list(~z1 | g1, ~z2 | g2)`
(c)	Named list of one-sided formulae without | operator, with grouping factors used as names of the components, e.g., `list(g1 = ~z1, g2 = ~z2)`
(d)	One-sided formula with | operator, e.g., `~z1 | g1/g2`

[a] The examples of the syntax are given for a hypothetical two-level model (13.5)
[b] Variables `z1` and `z2` are used as the random-effects covariates.
[c] Variables `g1` and `g2` are considered grouping factors

Table 14.2 *R syntax*: Limitations of the different forms of the syntax for the `object` argument of the `reStruct()` function

(**a**) A single-level LMM. Grouping factor: `g1`. *Z*-covariate: `z1`		
Form	Syntax of the argument	Limitation
(a)	`list(g1 = pdSymm(~z1))`	Most flexible
(b)	`list(~z1 | g1)`	No structure for $\boldsymbol{D}$; *pdLogChol* class by default[a]
(c)	`list(g1 = ~z1)`	*Same as above*
(d)	`~z1 | g1`	*Same as above*
(b) A two-level LMM. Grouping factors: `g1`, `g2`. *Z*-covariates: `z1`, `z2`		
Form	Syntax of the argument	Limitation
(a)	`list(g1 = pdSymm(~z1), g2 = pdDiag(~z2))`	Most flexible
(b)	`list(~z1 | g1, ~z2 | g2)`	The same $\boldsymbol{D}$ structure (*pdLogChol* class) used for both grouping factors[a]
(c)	`list(g1 = ~z1, g2 = ~z2)`	*Same as above*
(d)	`~z1 | g1/g2`	*Same as above* Additionally, the same *Z*-covariate(s) for both levels.

[a] The default value of the second argument, `pdClass = "pdLogChol"`, is assumed

The remaining forms of the syntax, (b)–(d), are notationally simpler, but also less flexible, as compared to (a). One complication is that the structure of the matrix (matrices) $\boldsymbol{D}$ has to be determined from the value of another argument of the function `reStruct()`, namely, `pdClass`. By default, the argument specifies the *pdLogChol* class, which results in a general positive-definite matrix. To change this default choice, the argument `pdClass` needs to be specified explicitly, and the call to the `reStruct()` function has to assume the form `reStruct(`*`object`*`,`*`pdClass`*`)`.

For LMMs for data with two or more levels of grouping, an additional limitation of the forms (b)–(d) of the syntax is that the structures of matrices $\boldsymbol{D}$ at different levels of grouping are forced to be *the same*.

A specific limitation of the syntax (d) for multilevel LMMs is that it also requires that the random-effects covariates are assumed to be the same at different grouping levels. For some models, this limitation is irrelevant; however, this is the case for, e.g., LMMs with random intercepts only.

It is worth mentioning that, regardless of the form of the syntax used, the order of specifying the grouping factors is important. More specifically, even if the grouping factors are coded as crossed with each other, they are effectively treated as nested, with the nesting order corresponding to the order, in which the factors are specified in the syntax. In particular, the grouping factors specified later in the syntax are nested within the factors specified earlier. For example, according to the syntax (a) in Table 14.1, the factor `g2` would be treated as nested within the factor `g1`.

14.3.2 Inspecting and Modifying Objects of Class reStruct

In Panel R14.5, we demonstrate how to create and extract information from objects of class *reStruct*. We use the syntax form (a) (Table 14.1) to create the *reStruct*-class object `reSt`. The object is constructed for a hypothetical two-level LMM, as defined in (13.5). We assume that the two levels of grouping are defined by the grouping factors `g1` and `g2`. The structures of the variance–covariance matrices $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ of random effects at the two levels of grouping are defined by the objects `pdSm` of class *pdSymm* and `pdCSf` of class *pdCompSymm*, respectively.

Using the function `isInitialized()`, we verify whether the object `reSt` is initialized. Given that both `pdSm` and `pdCSf` were initialized objects that inherited from the *pdMat* class (see Panel R14.1), the resulting *reStruct*-class object is also initialized. By applying the function `names()`, we get the names of the components of the list, contained in `reSt`, i.e., the names of factors `g1` and `g2`. The function `formula()` extracts the formula from each of the components. The displayed formulae correspond to those used in the definition of the objects `pdSymm` and `pdDCf` in Panel R14.1.

The function `getGroupsFormula()` provides information about the grouping of the data, used in the definition of the *reStruct*-class object. It refers to the conditioning expression, i.e., the expression used after the `|` operator in the formula(e) defining the object (see the syntax forms shown in Table 14.2). In our example, the structure is defined by the factors `g1` and `g2`, with levels of `g2` nested within the levels of `g1`. Note that the function `getGroupsFormula()` allows two optional arguments, `asList` and `sep`. Information about the use of these arguments can be obtained by issuing the command `?getGroupsFormula`.

In Panel R14.5, we also apply the function `Names()` to the object `reSt`. The function returns the names of rows/columns for the matrices, represented by the *pdMat*-class objects, which define the *reStruct*-class object (see also Panel R14.2).

R14.5 *R syntax*: Creating an object of class *reStruct*, representing a two-level LMM for data with two levels of grouping, and extracting information from the object. Auxiliary objects `pdSm` and `pdCSf`, which inherit from the *pdMat* class, were created in Panel R14.1

```
> reSt <- reStruct(list(g1=pdSm,    # D1
+                       g2=pdCSf)) # D2
> isInitialized(reSt)
  [1] TRUE
> names(reSt)                     # Note: order g1, g2 reversed
  [1] "g2" "g1"
> formula(reSt)                   # Formulae for pdMat components
  $g2
  ~-1 + agef

  $g1
  ~agex
> getGroupsFormula(reSt)          # Model hierarchy
  ~g1/g2
  <environment: 0x0000000003d6efd8>
> Names(reSt)                     # Row/col names for pdMat components
  $g2
  [1] "agefM" "agef0" "agefY"

  $g1
  [1] "(Intercept)" "agex"
```

In Panel R14.6, we show the methods of extracting information about the matrices corresponding to the *pdMat*-class objects, which define a *reStruct*-class object. As an example, we use the object `reSt`, which was created in Panel R14.5. The function `as.matrix()` used in Panel R14.6a displays the positive-definite matrices, corresponding to the two variance–covariance matrices of random effects at the two levels of grouping. The displayed matrices are, obviously, equivalent to those stored in the objects `pdSm` and `pdCSf`, which were used to define the object `reSt` (see Panel R14.1). By applying the function `coef()`, we list the unconstrained coefficients corresponding to the matrices. They correspond to the values displayed in Panel R14.3.

The individual *pdMat*-class objects, defining the *reStruct*-class object, can be obtained by extracting the appropriate components of the list, which is contained in the latter object. One possible way to achieve that goal is illustrated in Panel R14.6b. Additionally, using the `all.equal()` function, we confirm that the object, extracted as the `g2` component of `reSt`, is equivalent to the *pdMat*-class object `pdCsf`.

R14.6 *R syntax*: Extracting information about *pdMat*-class objects directly from an object of class *reStruct*, representing a two-level LMM for data with two-levels of grouping. The object reSt, which inherits from the *reStruct* class, was created in Panel R14.5

(a) *Listing information about positive-definite matrices from a reStruct object*

```
> as.matrix(reSt)                    # D1, D2
  $g1
              (Intercept) agex
  (Intercept)           4    1
  agex                  1    9

  $g2
        agefM agef0 agefY
  agefM     5     1     1
  agef0     1     5     1
  agefY     1     1     5
> coef(reSt)                         # Unconstrained coeff. for D2, D1
       g21       g22       g11       g12       g13
   0.80472 -0.13353   0.68424   0.08184   1.09344
```

(b) *Extracting individual pdMat-class components from a reStruct object*

```
> reSt[["g1"]]                       # See pdSm in Panel R14.1b
  Positive definite matrix structure of class pdSymm representing
              (Intercept) agex
  (Intercept)           4    1
  agex                  1    9
> g2.pdMat  <- reSt[["g2"]]          # See pdCSf in Panel R14.1c
> all.equal(pdCSf, g2.pdMat)         # g2.pdMat and pdCSf are equal
  [1] TRUE
```

Panel R14.7 demonstrates how to evaluate an object of class *reStruct* in the context of a dataset. Toward this end, we use data frames dt1 and dt2, which were created in Panel R14.1, together with the object reSt, which was created in Panel R14.5.

In Panel R14.7, we first apply the default method of the generic model.matrix() function (Sect. 5.3.2) to formulae extracted from the *pdMat*-class objects pdSm and pdCSf. The formulae are evaluated using the data stored in data frames dt1 and dt2, respectively. The created random-effects design matrices, $\boldsymbol{Z}_1$ and $\boldsymbol{Z}_2$, are stored in the objects Zmtx1 and Zmtx2, respectively, and displayed with the help of the matrix-printing function prmatrix().

Next, we create the random-effects design matrix $\boldsymbol{Z}$ corresponding to the object reSt. Toward this end, we first create the data frame dtz by merging the data frames dt1 and dt2. Then, we apply the function model.matrix() with

R14.7 *R syntax*: Creation of the design matrix $\mathbf{Z}$ by evaluating an object of class *reStruct* for (hypothetical) data containing random-effects covariates. Objects `dt1`, `dt2`, `pdSm`, and `pdCSf` were created in Panel R14.1. The object `reSt` was created in Panel R14.5

```
> Zmtx1 <- model.matrix(formula(pdSm), dt1)
> prmatrix(Zmtx1)                  # Design matrix Z_1 for pdSm
    (Intercept) agex
  1           1   15
  2           1   45
  3           1   71
  4           1   82
> Zmtx2 <- model.matrix(formula(pdCSf),dt2)
> prmatrix(Zmtx2)                  # Design matrix Z_2 for pdCSf
    agefM agef0 agefY
  1     0     0     1
  2     1     0     0
  3     0     1     0
  4     0     1     0
> dtz  <- data.frame(dt1,dt2)      # Data frame to evaluate reSt
> Zmtx <- model.matrix(reSt, dtz) # Design matrix Z for reSt
> prmatrix(Zmtx)                   # Matrix Z w/out attributes
    g2.agefM g2.agef0 g2.agefY g1.(Intercept) g1.agex
  1        0        0        1              1      15
  2        1        0        0              1      45
  3        0        1        0              1      71
  4        0        1        0              1      82
```

arguments `object=reSt` and `data=dtz`. Note that, because the object `reSt` is of class *reStruct*, the generic function `model.matrix()` does *not* dispatch its default method, but the `model.matrix.reStruct()` method from the **nlme** package. As a result, the random-effects design matrices for the objects `pdSm` and `pdCSf`, which define the object `reSt`, are created and merged. The outcome is stored in the matrix-object `Zmtx`, which is displayed with the use of the function `prmatrix()`. Note that, in `Zmtx`, the three first columns come from the design matrix corresponding to the object `pdSm`, which was used to define the variance–covariance matrix of random effects present at the level of grouping corresponding to the factor `g2`. When defining the object `reSt`, the factor was specified as the second one, after the factor `g1` (see Panel R14.5).

It is worth noting that, as compared to the default method of the function `model.matrix()`, the `model.matrix.reStruct()` method also allows for an optional argument `contrast`. The argument can be used to provide a named list of the contrasts, which should be used to decode the factors present in the definition of the *reStruct*-class object. Unless the argument is explicitly used, the default contrast specification is applied (see Sect. 5.3.2).

Table 14.3 *R syntax*: Extracting results from a hypothetical object `reSt` of class *reStruct*

Random- effects structure component to be extracted	Syntax
Summary	`summary(reSt)`
The *reStruct* formula	`formula(reSt)`
Groups formula	`getGroupsFormula(reSt)`
Constrained coefficients	`coef(reSt, unconstrained=FALSE)`
Unconstrained coefficients	`coef(reSt)`
List of $\boldsymbol{D}$ matrices	`as.matrix(reSt)`
	`pdMatrix(reSt)`
Log-determinants of $\boldsymbol{D}^{1/2}$ matrices	`logDet(reSt)`

For the reader's convenience, in Table 14.3, we summarize the methods used to extract the information about the components of an *reStruct*-class object.

14.4 The Random Part of the Model Representation: The *lmeStruct* Class

The *lmeStruct* class is an auxiliary class, which allows us to compactly store the information about the random part of an LMM, including the random effects structure, correlation structure, and variance function. Objects of this class are created using the `lmeStruct()` function with three arguments: `reStruct`, `corStruct`, and `varStruct`. The arguments are given as objects of class *reStruct*, *corStruct*, and *varFunc*, respectively. The classes were described in Sects. 14.3, 11.2, and 8.2, respectively.

The argument `reStruct` is mandatory, while `corStruct` and `varStruct` are optional, with the default value equal to `NULL`.

The function `lmeStruct()` returns a list determining the model components. The list contains at least one component, namely, `reStruct`.

When specifying an LMM with the help of the `lme()` function, the use of an *lmeStruct*-class object is not needed. Such an object is nevertheless created very early during the execution of the `lme()`-function call. The importance of the *lmeStruct* class will become more apparent in Sect. 14.6, where we demonstrate how to extract results from an object containing a fit of an LMM.

In Panel R14.8, we demonstrate how to create and extract information from an object of class *lmeStruct*.

First, we create an object of class *reStruct*. Toward this end, we use the `reStruct()` constructor function (Sect. 14.3.1). The created object, `reSt`, is the same as the one constructed in Panel R14.5. It defines the random-effects structure of a two-level LMM, with grouping specified by factors `g1` and `g2` (Sect. 14.3.2). The variance–covariance matrices of random effects at the two levels of grouping are defined by the objects `pdSm` of class *pdSymm* (a general positive-definite matrix) and `pdCSf` of class *pdCompSymm* (a compound-symmetry matrix), respectively.

R14.8 *R syntax*: Creating and probing objects of class *lmeStruct*. Objects `pdSm` and `pdCSf`, which inherit from the *pdMat* class, were created in Panel R14.1

```
> reSt  <- reStruct(list(g1=pdSm, g2=pdCSf))          # reStruct class
> corSt <- corExp(c(0.3,0.1), form=~tx, nugget=TRUE) # corStruct class
> vF    <- varExp(0.8, form=~agex)                    # varFunc class
> (lmeSt<- lmeStruct(reStruct=reSt, corStruct=corSt, # lmeStruct class
+           varStruct = vF))                          # ... created.
  reStruct  parameters:
       g21       g22       g11       g12       g13
   0.80472 -0.13353   0.68424   0.08184   1.09344
  corStruct  parameters:
   range nugget
     0.3    0.1
  varStruct  parameters:
  expon
    0.8
> coefs <- coef(lmeSt,unconstrained=FALSE)# Constrained coefficients...
> (as.matrix(coefs))                        # ... printed more compactly
                                      [,1]
  reStruct.g2.std. dev              2.23607
  reStruct.g2.corr.                 0.20000
  reStruct.g1.var((Intercept))      4.00000
  reStruct.g1.cov(agex,(Intercept)) 1.00000
  reStruct.g1.var(agex)             9.00000
  corStruct.range                   1.34986
  corStruct.nugget                  0.52498
  varStruct.expon                   0.80000
```

In the next step, we create an object of class *corStruct* (Sect. 11.2). As an example, we consider the *corExp* class, which represents the exponential correlation structure (Sect. 11.4.3). The object `corSt` corresponds to an exponential structure with the range parameter $\varrho = 0.3$ and the nugget equal to 0.1 (see Sects. 10.3.2 and 11.2.1 and Panel R11.1).

Finally, we specify the object `vF` of class *varFunc* (Sect. 8.2). As an example, we consider the *varExp* class, which represents a variance structure defined by an exponential function of the covariate `agex` (see Table 7.2 in Sect. 7.3.1).

Using the objects `reSt`, `corSt`, and `vF` as the arguments `reStruct`, `corStruct`, and `varStruct`, respectively, of the `lmeStruct()` function, we create the object `lmeSt` of class *lmeStruct*. With the help of the `coef()` function, combined with the use of the `unconstrained=FALSE` argument, we display the coefficients, defining the various components of the *lmeStruct*-class object, in the constrained form (see Sects. 8.3, 11.3.1, and 14.2.2).

14.5 Using the Function lme() to Specify and Fit Linear Mixed-Effects Models

The generic lme() function is the most frequently used function to fit LMMs in R. It allows to specify and fit models described in Sect. 13.2 with nested random effects and correlated and/or heteroscedastic within-group residual errors. In general, to define the LMM in full, we need at least to specify the mean structure and the random-effects structure, including the grouping factors defining model hierarchy. In addition, the correlation structure, variance function, and model frame need to be defined.

In Table 14.4, we summarize selected arguments used by the function lme(), together with a reference to the section describing the appropriate syntax and an indication of the implied LMM components. In what follows, we briefly summarize the use of the arguments.

The principal argument fixed is primarily used to define the mean structure of an LMM. The argument can accept objects of classes *formula*, *groupedData*, or *lmList*. Depending on the class of the object, the corresponding method of the lme() function, i.e., lme.formula(), lme.groupedData(), or lme.lmList(), is used.

The most common choice for the fixed argument is a two-sided formula (Sect. 5.2).

The argument can be specified using an object of class *groupedData*. This way allows providing the information about the mean structure and about the model hierarchy defined by (nested) grouping factors. An important limitation of this form of specification of the fixed argument is that it allows only for mean structures with one (primary) covariate.

The argument can also be specified by providing an *lmList*-class object, i.e., a list of *lm*-class linear-model-fit objects (Sect. 5.5) for all levels of a grouping factor. This method is rarely used and we do not present it here.

Table 14.4 *R syntax*: Selected arguments of the function lme() used to specify a linear mixed-effects model defined in Sect. 13.2

	Argument		
Name	Class	Syntax	Component(s) created/defined
fixed	*formula*	Sect. 5.2	Mean structure
	groupedData	Sect. 2.6	Mean structure; grouping factors
	lmList	–	–
random	*reStruct*[a]	Sect. 14.3	Random-effects structure; grouping factors (optionally)
correlation	*corStruct*[a]	Sect. 11.2	Correlation structure
weights	*varFunc*[a]	Sect. 8.2	Variance function
data	*data.frame*	Sect. 5.4	Data
	groupedData	Sect. 2.6	Data; grouping factors
method		Sect. 5.4	Estimation method

[a]Other choices of the class for the corresponding argument are possible but not listed.

The `random` argument is the primary argument used to define the random-effects structure. In the argument, the syntax forms (a)–(d), shown in Table 14.1, can be used. They allow specifying all aspects of the random-effects structure, *including* the model hierarchy defined by grouping factors.

The specification of the `random` argument can be simplified by omitting the reference to grouping factors in the forms (a)–(c) of syntax from Table 14.1. For example, in the syntax (a), the names of the list components `g1` and `g2` can be omitted. That is, a list with *unnamed* components, i.e., `list(pdSymm(~z1), pdDiag(~z2))`, can be used. The simplified syntax has disadvantages. For instance, it does not include the information about the grouping factors defining the model hierarchy. The information needs to be supplemented using a *groupedData*-class object in the `fixed` or `data` argument.

If the `random` argument is not specified, then, by default, it is assumed that the design matrices for the fixed and random effects are equal ($\boldsymbol{X}_i \equiv \boldsymbol{Z}_i$) and that the variance–covariance matrix for the random effects is defined by an object of class *pdSymm*. That is, a general variance–covariance matrix is assumed. Also in this case, the information about the model hierarchy needs to be supplemented using a *groupedData*-class object in the `fixed` or `data` argument.

The syntax of the arguments `weights` and `correlation` is the same as for the corresponding arguments of the `gls()` function (Sects. 8.4 and 11.5). The two arguments allow to specify the residual variance–covariance matrix $\mathbf{R}_i$, as defined in (13.4), using the decomposition given by (10.8). The default values for the `weights` and `correlation` arguments imply independent and homoscedastic conditional residual errors.

The `data` argument is used to provide the raw data and, optionally, the information about the data hierarchy. Similarly to other model-fitting functions, additional arguments `subset` and `na.action` can be used together with `data` to define the model frame (Sect. 5.3.1). For the definition of these arguments, we refer to Sect. 5.3.1.

The default value of the `method` argument is `method="REML"`. That is, the model parameters are estimated using the restricted likelihood (Sect. 13.5.2). An alternative is `method="ML"`. It is worth mentioning that the initial values for the $\boldsymbol{\theta}_D$ parameters are refined using an EM-based algorithm (Sect. 13.5.6).

14.6 Extracting Information from a Model-Fit Object of Class *lme*

In Table 14.5, we present several methods to extract information from a model-fit object of class *lme*. We assume that an object `lme.fit` is available, which contains the results of fitting a single-level LMM, defined in (13.1)–(13.4).

Table 14.5 *R syntax*: Extracting results from a hypothetical object `lme.fit` of class *lme*, which represents a single-level linear mixed-effects model fitted using the `lme()` function

Model-fit component to be extracted	Function: `lme()` Package: **nlme** Object: `lme.fit` Class: *lme*
Summary	`(summ <- summary(lme.fit))`
Estimation method	`lme.fit$method`
$\widehat{\boldsymbol{\beta}}$	`fixef(lme.fit)`
$\widehat{\boldsymbol{\beta}}$, se$(\widehat{\boldsymbol{\beta}})$, t-test	`summ$tTable`
$\widehat{\text{Var}}(\widehat{\boldsymbol{\beta}})$	`vcov(lme.fit)`
95% CI for $\boldsymbol{\beta}$	`intervals(lme.fit, which="fixed")`
$\widehat{\sigma}$	`summ$sigma`
95% CI for $\boldsymbol{\theta}, \sigma$	`intervals(lme.fit, which="var-cov")`
95% CI for $\boldsymbol{\theta}_D$	`intervals(lme.fit, which="var-cov")$reStruct`
$\widehat{\mathbf{b}}_i$	`ranef(lme.fit)`
$\widehat{\boldsymbol{\beta}}$ + "coupled" $\widehat{\mathbf{b}}_i$	`coef(lme.fit)` `coef(summ)`
$\widehat{\mathcal{D}}$	`getVarCov(lme.fit)`
$\widehat{\mathcal{D}}$ and $\widehat{\sigma}$	`VarCorr(lme.fit)`
$\widehat{\mathcal{R}}_i$	`getVarCov(lme.fit, type="conditional")`
$\widehat{\mathcal{V}}_i$	`getVarCov(lme.fit, type="marginal")`
ML value	`logLik(lme.fit, REML = FALSE)`
REML value	`logLik(lme.fit, REML = TRUE)`
AIC	`AIC(lme.fit)`
BIC	`BIC(lme.fit)`
Fitted values:	
- conditional, (13.12)	`fitted(lme.fit)`
- marginal, (13.24)	`fitted(lme.fit, level=0)`
Raw residuals:	
- conditional, (13.52)	`resid(lme.fit, type="response")`
- marginal, (13.53)	`resid(lme.fit, type="response", level=0)`
Normalized residuals:	
- conditional, (13.54)	`resid(lme.fit, type="normalized")`
- marginal, (13.55)	–
Pearson residuals, Sect. 7.5.1	`resid(lme.fit, type="pearson")`
Predicted values:	
- conditional	`predict(lme.fit, newdata)`
- marginal	`predict(lme.fit, newdata, level= 0)`

Using the generic function `summary()` allows us to obtain general information about the fitted form of the model, including the information about the estimated values of the fixed effects, the fitted random-effects structure, and the estimated residual variance–covariance matrix.

If information about only a specific aspect of the fitted model is needed, it can be obtained by extracting a specific component of the model-fit object or by applying

a special function to the object. For instance, the estimation method, used to fit the model, is displayed by extracting the `lme.fit$method` component. Estimates of the fixed effects $\boldsymbol{\beta}$ are displayed using the function `fixef()`, while the estimated variance–covariance of the estimates (Sect. 13.5.5) is obtained using the function `vcov()`.

Confidence intervals for the model parameters (see Sect. 13.7.3) are obtained by applying the generic function `intervals()`. Intervals for a specific subgroup of the parameters are selected using the argument `which`. For instance, `which="fixed"` provides CIs for the fixed effects, while `which="var-cov"` yields the intervals for all variance–covariance parameters. By default, `which="all"`, i.e., CIs for all model parameters are provided. The confidence level can be chosen with the help of the `level` argument. By default, `level=0.95`. The result of the application of the function `intervals()` to a model-fit object of class *lme* is a list with named components. Each of the components is a data frame, with rows corresponding to the parameters of the model, and columns representing the estimated values and the confidence limits for the parameters. The possible components are the following: `fixed` (fixed effects), `reStruct` (parameters of the variance-covariance matrices of the random effects), `corStruct` (residual correlation-structure parameters), `varFunc` (residual variance-function parameters), and `sigma` (scale parameter). In Table 14.5, we present how to display CIs only for the parameters of the variance–covariance matrices of the random effects by extracting the `reStruct` component of the object resulting from the `intervals()` function call.

By applying the function `ranef()` to a model-fit object of class *lme*, the estimated random effects are displayed. By default, the effects at all levels of grouping are displayed. The levels can be selected with the help of the `level` argument. Information about other arguments, available for the function `ranef()`, can be obtained by issuing the command `?ranef`.

The function `coef()`, applied to an *lme*-class model-fit object, displays the estimated coefficients for a particular (or all) levels of grouping. The coefficients are obtained by summing the fixed effects and, if appropriate, the "coupled" random effects (Sect. 13.2.1). The levels can be selected with the help of the `level` argument. Information about other arguments of the function `coef()` can be obtained by issuing command `?coef.lme`.

Estimates of the variance–covariance matrices of the random effects and residual errors, as well as the marginal variance–covariance matrix, are obtained using the function `getVarCov()`. The argument `type` allows choosing the matrix to be displayed. In particular, `type="random.effects"` (the default) prints out the estimates of the variance–covariance matrices of random effects, `type="conditional"` prints out the estimate of the residual variance–covariance matrix, and `type="marginal"` provides the estimate of the marginal variance-covariance matrix. With the help of the `individuals` argument, it is possible to select the group(s) of observations, for which the function `getVarCov()` should display the (residual or marginal) variance–covariance matrices.

An alternative method to extract the variance–covariance matrix of the random effects is to use the function `VarCorr()`. When applied to an *lme*-class model-

fit object, the function extracts the estimated variances, standard deviations, and correlations of the random effects. Additionally, it provides the estimates of σ^2 and σ. The function uses three arguments: `x`, `sigma`, and `rdig`. The first one specifies the model-fit object; the second one is an optional numeric value that indicates a multiplier for the standard deviations and assumes the value of 1 as default; and the last one is an optional integer value, which indicates the number of digits (by default, 3) that are to be used to represent the correlation estimates.

Fitted values, residuals, and predicted values are obtained by applying the functions `fitted()`, `resid()`, and `predict()`, respectively. All the functions allow for an optional argument `level`, in the form of a vector of integers, which indicates the level(s) of grouping, for which the values are to be extracted. The levels increase from 0, i.e., the population level, to the highest level of grouping, i.e., the level corresponding to the grouping factor, which is nested within all the other factors. Thus, `level=0` yields the estimates of the marginal mean values or marginal residuals, while for nonzero levels, the conditional mean values or conditional residuals are provided. In particular, the conditional mean values at a particular level, k say, are obtained by adding the marginal mean values and the predictors of the random effects at the grouping levels lower or equal to k. The conditional residuals at level k are obtained by subtracting the conditional mean values at that level from the dependent-variable vector. By default, `level` specifies the highest level of grouping.

An important argument of the function `resid()` is `type`. It indicates the type of residuals to be computed (Sect. 13.6.2). The possible choices are `type="response"` (raw residuals), `type="pearson"` (Pearson residuals), and `type="normalized"` (normalized residuals). It is worth mentioning that the Pearson/normalized residuals are standardized/transformed based on the elements of the residual variance–covariance matrix $\widehat{\mathcal{R}}_i$, and not on the marginal variance–covariance matrix $\widehat{\mathcal{V}}_i$. Hence, the use of arguments `type="pearson"` or `type="normalized"` in combination with a nondefault value of `level` argument is not meaningful. This remark applies, for example, to marginal residuals obtained using `level=0`.

The function `predict()` allows for an optional argument `newdata`. It indicates a data frame for which the predictions are to be calculated. The data frame should contain all variables that were used to specify the fixed effects and the random effects of the fitted LMM, as well as the grouping factors. If the argument is missing, the function will employ data used to fit the model. Consequently, it returns the fitted values corresponding to the level specified in the `level` argument.

In Table 14.6, we present methods of extracting the details about the `lme()`-function call which was used to create a model-fit object of class *lme*. The methods are similar to those presented in Tables 5.5, 8.2, and 11.1 for a *gls*-class model-fit object. Note that the function `model.matrix()` (Sect. 5.3.2) provides the design matrix for the mean structure, i.e., the matrix $\boldsymbol{X}_i$. Extracting the design matrix for the random effects, $\boldsymbol{Z}_i$, from an *lme*-class model-fit object is difficult. As it requires extra programming, we do not present the required code.

Table 14.6 *R syntax*: Extracting components of the `lme()`-function call from a hypothetical object `lme.fit`, which represents a fitted single-level linear mixed-effect model

R call component	Syntax
R call	`(cl <- getCall(lme.fit))`
Formula for the mean structure	`(form <- formula(lme.fit))`
Argument `random`	`cl$random`
Argument `correlation`	`cl$correlation`
Argument `weights`	`cl$weights`
Data name	`(df.name <- cl$data)`
Data frame	`(df <- eval(df.name))`
Model frame	`(mf <- model.frame(form, df))`
Design matrix	`model.matrix(form, mf)`

Table 14.7 *R syntax*: Extracting information about the random components from a hypothetical object `lme.fit`, which represents a fitted single-level linear mixed-effect model

Auxiliary objects to be extracted (R class)	Syntax
Random part of the model (*lmeStruct*) *see* Section 14.4	`lmeSt <- lme.fit$modelStruct`
Random-effects structure (*reStruct*) *see* Section 14.3.2	`reSt <- lmeSt$reStruct`
Variance-function structure (*varFunc*) *see* Table 8.2b	`vF <- lmeSt$varStruct`
Correlation structure (*corStruct*) *see* Table 11.1b	`cSt <- lmeSt$corStruct`

In Table 14.7, we summarize methods to extract information about the random components of a fitted LMM. As mentioned in Sect. 14.4, the random part of the model, which includes the random effects structure, the residual correlation structure, and the residual variance function, is represented by an object of class *lmeStruct*. The object can be accessed by referring to the `modelStruct` component of the *lme*-class model-fit object. The random effects structure, as described in Sect. 14.3, is represented by an object of class *reStruct*, which is stored as the `reStruct` component of the *lmeStruct*-class object. If a correlation structure and/or a variance function were used in defining the residual variance–covariance matrix of the LMM, they are represented by objects of classes *corStruct* and *varFunc*, respectively, which are stored as components `corStruct` and `varStruct`, respectively, of the *lmeStruct*-class object.

14.7 Tests of Hypotheses About the Model Parameters

As was the case for LMs for independent, heteroscedastic observations (Sect. 8.5) or fixed-effects LMs for correlated data (Sect. 11.6), results of the F-tests for linear hypotheses about the fixed effects (Sect. 13.7.1), based on a fitted LMM, are accessed by applying the `anova()` method to the model-fit object of class *lme*. By default, the sequential-approach tests are obtained (Sect. 4.7.1). To obtain the

marginal-approach tests, the argument `type="marginal"` should be used. *F*-tests for individual/multiple terms can be obtained by applying the argument `Terms` in the form of an integer vector or a character vector that specifies the terms in the model that should be jointly tested. If a character vector is used, it should contain the names of the terms used in the model formula. If an integer vector is used, its elements should correspond to the order in which terms are included in the model formula. Additional arguments that can be used in the `anova()` method for *lme*-class model-fit objects include `test`, `adjustSigma`, `L`, and `verbose`. The information about the use of these arguments can be obtained by issuing the command `?anova.lme`.

The *t*-tests for individual coefficients are provided by applying, e.g., the `summary()` method to the model-fit object. Note that, in this case, the marginal-approach tests are obtained.

As was discussed in Sect. 13.7.1, neither the *p* values for the *F*-tests, nor the ones for the *t*-tests, adjust for the fact that the null distribution of the test statistics is only approximated by *F*- or *t*-distributions, respectively. Thus, the degrees of freedom for the tests are computed as in a balanced, multilevel ANOVA design (Schluchter and Elashoff 1990; Pinheiro and Bates 2000). In particular, assuming G levels of grouping, the number of denominator degrees of freedom ddf_g for the tests of fixed effects at level g ($g = 1, \ldots, G+1$) is equal to

$$ddf_g = N_g - (N_{g-1} + p_g), \tag{14.1}$$

where N_g is the number of groups at the g-th grouping level and p_g is the number of fixed-effects coefficients estimated at that level. The latter is the number of coefficients related to the variables whose values change across the values of the grouping factor(s) at the grouping level g, but do not change across the values of the grouping factor(s) at the level $g-1$. Note that the intercept, if present in the model, is treated as being estimated at the level $g = 0$, but its denominator degrees of freedom are calculated from the level $G+1$, i.e., at the level of observations. An example of the calculation of the denominator degrees of freedom is presented in Sect. 16.7.1.

When the function `anova()` is applied to two or more objects of class *lme*, it provides LR-test statistics, calculated based on pairs of the LMMs represented by the consecutive objects. If the models are nested, have the same structure of random effects and of residual variance–covariance matrix, and are fitted using ML, the results of the LR tests, reported by the function, provide valid tests for hypotheses about the fixed effects (Sect. 7.6.1). On the other hand, if the nested models are fitted using REML and have the same mean structure, but different random structures, the reported LR tests are valid tests of hypotheses about the parameters defining random-effects structure.

In Sect. 13.7.1 it was mentioned that, instead of using a χ^2 distribution for an LR test of a hypothesis about fixed effects, one could use an empirical distribution of the test statistic, obtained by fitting the alternative and null models to multiple datasets simulated under the null model. A similar comment was made in Sect. 13.7.2 for

the LR test of hypotheses about random effects, when the values of the variance–covariance parameters, compatible with the null hypothesis, lie on the boundary of the parameter space. To address this issue, the function `simulate()` from the package **nlme** can be used. It computes the ML- and/or REML-based log-likelihood values for multiple datasets simulated from the null and alternative LMMs. This allows the calculation of the empirical distribution of the LR-test statistic.

The function admits the following arguments: `object`, `m2`, `nsim`, `method`, `seed`, `niterEM`, and `useGen`. The first four are the most important ones and we describe them in more detail below. A description of all of the arguments can be obtained by issuing the command `?simulate`.

The argument `object` defines the null model. The argument can be provided either as an object of class *lme*, which represents a fitted LMM, or as a named list with components `fixed`, `data`, and `random`, which should define a valid call of the function `lme()` to fit the LMM (Sect. 14.5). The argument `m2` defines the alternative model and can be specified in a similar way as the `object` argument. If it is specified as a list, only those components that change between the null and alternative models need to be specified. The argument `nsim` is a positive integer which indicates the number of simulations to be performed. By default, `nsim=1`. Thus, although the arguments is optional, in practice, it should be always specified.

Finally, the argument `method`, which is an optional character array, allows choosing the form of the likelihood on which the LR-test statistic is to be based. By default, `method=c("REML", "ML")`, i.e., both ML- and REML-based LR-test statistics are used.

The function returns an object of class *simulate.lme*, which is a named list with two components: `null` and `alt`. Each of them has components `ML` and/or `REML`, which are matrices. The matrices contain, in particular, the column `logLik` which provides the ML- or REML-based log-likelihood value for each of the `nsim` simulations. Additional attributes of the *simulate.lme*-class object include, among others, `seed` and `df`. The former gives the random seed used in the random number generator, while the latter gives the difference in the number of parameters between the null and alternative models.

One way to present the result of the `simulate()`-function call is to plot the empirical and nominal p values. The former are obtained from the empirical distribution of the LR-test statistic values corresponding to the simulated values of the ML- or REML-based log-likelihood, while the latter are computed from applying a χ^2 distribution or a mixture of χ^2 distributions to the simulated values of the LR-test statistic. The plot can be obtained by a call like `plot(object,df)`, where *object* is a *simulate.lme*-class object, while *df* is a vector of integers, which defines the degrees of freedom of a χ^2 distribution to be used to compute the nominal p values. If the vector contains more than one integer, multiple plots of nominal *versus* empirical p values are created by computing the nominal p values from the χ^2 distribution with the number of degrees of freedom equal to each of the integers. Additionally, a plot for an equal-weight mixture of the χ^2 distributions is created.

An example of the use of the function `simulate()` is provided in Sect. 16.6.2.

For the case of using the REML-based LR test for testing a hypothesis about the random-effects structure, a possible alternative is the function `exactRLRT()` from the package **RLRsim**. The function simulates values of the REML-based LR-test statistic for testing the null hypothesis that the variance of a random effect is 0 in an LMM with a known correlation structure of the tested random effect and independent and identically distributed random errors. The simulations are based on the finite-sample distribution of the test statistic (Sect. 13.7.2) which was derived by Crainiceanu and Ruppert (2004). The performance of the simulations was studied by Scheipl (2010).

The main arguments of the function `exactRLRT()` are `m`, `m0`, and `mA`. The first one is a model-fit object of class *lme* or *lmer*. For LMMs with a single variance component (random effect), it provides the fitted model under the alternative hypothesis. For models with multiple variance components, it should provide the model containing only the random effect to be tested. Arguments `mA` and `m0` apply only to models with multiple variance components. The former specifies the model fitted under the alternative hypothesis, while the latter gives the model fitted under the null hypothesis. Additional arguments include `nsim`, which is used to specify the number of values of the test statistic to be simulated. By default, `nsim=10000`. The list of all arguments of the function `exactRLRT()` can be obtained by issuing the command `?exactRLRT` (after attaching the package **RLRsim**). An example of the use of the function `exactRLRT()` is provided in Sect. 16.6.1.

It is worth noting that the functions `simulate()` and `exactRLRT()` have important limitations. For instance, they both only apply to conditional independence LMMs (Sect. 13.4). Additionally, the function `exactRLRT()` allows only for independent random effects; `simulate()` can accommodate correlated random effects, i.e., LMMs with nondiagonal variance–covariance matrices of random effects $\boldsymbol{D}$.

Finally, it is worth mentioning that the function `anova()`, when applied to two or more objects of class *lme*, also provides the information criteria (Sect. 4.7.2) that can be used to choose the best-fitting models from a set of nonnested models with different mean and/or variance–covariance structures. The AIC and BIC can also be obtained using the functions `AIC()` and `BIC()`, respectively (see Table 14.5).

14.8 Chapter Summary

In this chapter, we presented the tools available for fitting LMMs in the R package **nlme**. In particular, we focused on the function `lme()`.

The use of the function involves the concepts of model formula, grouped data, variance function, and correlation structure. The concepts were introduced in the previous chapters in the context of simpler LMs to facilitate their description and explanation. A new, important component was the random-effects structure.

We described several tools related to it, including the *pdMat* class for representing positive-definite matrices (Sect. 14.2) and the *reStruct* class for representing the random-effects structure (Sect. 14.3). In Sect. 14.4, we explained the representation of the random part of an LMM in the form of objects of class *lmeStruct*. The objects are created when the function `lme()` is used to fit an LMM (Sect. 14.5). In Sect. 14.6, we described how to extract information about the estimated model from an *lme*-class model-fit object. Finally, in Sect. 14.7, we briefly reviewed the tools available for inference based on a fitted LMM.

The use of the R tools presented in this chapter will be illustrated in Chap. 16, where the application of LMMs to the analysis of the ARMD case study will be described.

It is worth noting that, as was mentioned in Sect. 14.3.1, when a *reStruct*-class object is created, the grouping factors are effectively treated as nested. This means that the function `lme()` is not particularly suitable to fit LMMs with, e.g., crossed random effects. Such models can be easily fitted by applying the function `lmer()` from the package **lme4.0**, which we present in the next chapter.

Chapter 16
ARMD Trial: Modeling Visual Acuity

16.1 Introduction

In Chap. 12, we presented an analysis of the age-related macular degeneration (ARMD) data using LM with fixed effects for correlated data. The analysis took into account the correlation between visual acuity measurements obtained for the same patient. To apply the models, we used the function `gls()` from the R package **nlme**. Note that the models can be seen as population-averaged (marginal) models.

An alternative approach to the analysis of the ARMD data, which allows taking into account the correlation between the measurements, is to use linear mixed-effects models (LMMs). In this approach, the hierarchical structure of the data is directly addressed, with random effects that describe the contribution of the variability at different levels of the hierarchy to the total variability of the observations.

In this chapter, we fit several LMMs to the ARMD data. We primarily use the function `lme()` from the package **nlme**. For illustration purposes, several models are refitted by applying the function `lmer()` from the package **lme4.0**.

In particular, in Sect. 16.2, we consider a random-intercept model with homoscedastic residual errors, while in Sect. 16.3, we present a random-intercept model with heteroscedastic errors, with residual variance specified as a power function of time. A series of models with random intercepts and random slopes for time and with heteroscedastic residual errors is described in Sects. 16.4 and 16.5. In Sect. 16.6, we look at the issue of testing hypotheses about the random effects. In Sect. 16.7, we repeat the analysis for selected models using the function `lmer()` from the package **lme4.0**. A summary of the analyses is provided in Sect. 16.8.

16.2 A Model with Random Intercepts and Homogeneous Residual Variance

We start with a simple model, which we label **M16.1**. It contains subject-specific random intercepts and homoscedastic residual errors. Consequently, observations for the same individual, which share the random intercept, are correlated with

A. Gałecki and T. Burzykowski, *Linear Mixed-Effects Models Using R: A Step-by-Step Approach*, Springer Texts in Statistics, DOI 10.1007/978-1-4614-3900-4_16,

a constant (positive) correlation coefficient. Marginally, this corresponds to a compound-symmetry correlation structure with a correlation parameter greater than zero. A compound-symmetry marginal model was fitted to the ARMD data as model **M12.1** in Sect. 12.3. As was mentioned in that section, the compound-symmetry structure is too simple to describe the variance-covariance structure of the visual acuity measurements. Thus, we present model **M16.1** mainly to illustrate the fundamental steps in specifying and fitting an LMM.

16.2.1 Model Specification

Model **M16.1** is specified as follows:

$$\begin{aligned} \text{VISUAL}_{it} = {} & \beta_0 + \beta_1 \times \text{VISUAL0}_i + \beta_2 \times \text{TIME}_{it} + \beta_3 \times \text{TREAT}_i \\ & + \beta_4 \times \text{TREAT}_i \times \text{TIME}_{it} \\ & + b_{0i} + \varepsilon_{it}. \end{aligned} \tag{16.1}$$

The term VISUAL_{it} in (16.1) denotes the value of visual acuity measured for patient i ($i = 1, \ldots, 240$) at time t ($t = 1, 2, 3, 4$, corresponding to values of 4, 12, 24, and 52 weeks, respectively). In the fixed-effects part of the model, given by the first two lines of (16.1), VISUAL0_i is the value of visual acuity measured at baseline; TIME_{it} is the time of the measurement, corresponding to t; TREAT_i is the treatment indicator, equal 1 for the active group and 0 otherwise; and $\text{TREAT}_i \times \text{TIME}_{it}$ is their interaction. The parameter β_0 is an overall intercept, β_1 describes the change in the mean visual acuity due to a unit increase in visual acuity at baseline, β_2 describes the change due to a one week change in time, β_3 gives an overall treatment effect, and β_4 describes the additional change due to a one week change in time for patients treated with the active treatment. Note that we use a linear time effect, following the findings based on the final marginal model **M12.3**. However, contrary to these findings, we include the interaction between time and treatment in (16.1), to "enrich" the fixed part of the mean structure. Also, as it will become clear shortly, we simplify the variance-covariance structure.

In the random-effects part of the model, given by the last line of (16.1), b_{0i} is a patient-specific random intercept, assumed to be normally distributed with mean 0 and variance d_{11}, while ε_{it} is a residual random error assumed to be normally distributed with mean 0 and variance σ^2. Note that, formally speaking, the random intercept b_{0i} is a subject-specific *deviation* from the fixed intercept β_0. It is, however, customary to call b_{0i} a subject-specific random intercept, despite the fact that, actually, β_0 *and* b_{0i} are "coupled" (Sect. 13.2.1) and they *both* contribute to the subject-specific intercept. Typically, this convention does not lead to any misunderstanding.

The fixed part of model **M16.1** assumes that the average profile is linear in time, with different intercepts and slopes for the placebo and active treatment groups. The subject-specific profiles are assumed to also be linear in time, with subject-specific (random) intercepts that shift the individual profiles from the average linear trend.

In matrix notation, the model for the subject i with a complete set of four visual acuity measurements is expressed as follows:

$$\begin{pmatrix} \text{VISUAL}_{i1} \\ \text{VISUAL}_{i2} \\ \text{VISUAL}_{i3} \\ \text{VISUAL}_{i4} \end{pmatrix} = \left(\begin{array}{c|c|c|c|c} 1 & \text{VISUAL0}_i & 4 & \text{TREAT}_i & 4\cdot\text{TREAT}_i \\ 1 & \text{VISUAL0}_i & 12 & \text{TREAT}_i & 12\cdot\text{TREAT}_i \\ 1 & \text{VISUAL0}_i & 24 & \text{TREAT}_i & 24\cdot\text{TREAT}_i \\ 1 & \text{VISUAL0}_i & 52 & \text{TREAT}_i & 52\cdot\text{TREAT}_i \end{array}\right) \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{pmatrix}$$

$$+ \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \times b_{0i} + \begin{pmatrix} \varepsilon_{i1} \\ \varepsilon_{i2} \\ \varepsilon_{i3} \\ \varepsilon_{i4} \end{pmatrix}. \tag{16.2}$$

Solid vertical lines in (16.2) are used to separate the columns in the subject-specific design matrix $\boldsymbol{X}_i$ for the subject i.

Note that (16.2) can easily be written in the form of (13.1)–(13.3), upon defining

$$\mathbf{y}_i \equiv \begin{pmatrix} \text{VISUAL}_{i1} \\ \text{VISUAL}_{i2} \\ \text{VISUAL}_{i3} \\ \text{VISUAL}_{i4} \end{pmatrix}, \tag{16.3}$$

$$\boldsymbol{X}_i \equiv \left(\begin{array}{c|c|c|c|c} 1 & \text{VISUAL0}_i & 4 & \text{TREAT}_i & 4\cdot\text{TREAT}_i \\ 1 & \text{VISUAL0}_i & 12 & \text{TREAT}_i & 12\cdot\text{TREAT}_i \\ 1 & \text{VISUAL0}_i & 24 & \text{TREAT}_i & 24\cdot\text{TREAT}_i \\ 1 & \text{VISUAL0}_i & 52 & \text{TREAT}_i & 52\cdot\text{TREAT}_i \end{array}\right), \boldsymbol{Z}_i \equiv \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \tag{16.4}$$

$$\boldsymbol{\varepsilon}_i \equiv \begin{pmatrix} \varepsilon_{i1} \\ \varepsilon_{i2} \\ \varepsilon_{i3} \\ \varepsilon_{i4} \end{pmatrix}, \quad \boldsymbol{\beta} \equiv \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{pmatrix}, \quad \text{and} \quad \mathbf{b}_i \equiv b_{0i}, \tag{16.5}$$

with

$$\boldsymbol{\mathcal{D}} \equiv d_{11}, \quad \text{and} \quad \boldsymbol{\mathcal{R}}_i \equiv \sigma^2 \boldsymbol{I}_4, \tag{16.6}$$

where $\boldsymbol{I}_4$ is the 4×4 identity matrix.

The random part of model **M16.1**, specified by (16.6), leads, according to (13.25), to the following marginal variance-covariance matrix for the subject i with four observations:

$$\mathcal{V}_i \equiv \mathbf{Z}_i \mathcal{D} \mathbf{Z}_i' + \sigma^2 \mathbf{I}_4 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} d_{11} \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix} + \begin{pmatrix} \sigma^2 & 0 & 0 & 0 \\ 0 & \sigma^2 & 0 & 0 \\ 0 & 0 & \sigma^2 & 0 \\ 0 & 0 & 0 & \sigma^2 \end{pmatrix}$$

$$= \begin{pmatrix} \sigma^2 + d_{11} & d_{11} & d_{11} & d_{11} \\ d_{11} & \sigma^2 + d_{11} & d_{11} & d_{11} \\ d_{11} & d_{11} & \sigma^2 + d_{11} & d_{11} \\ d_{11} & d_{11} & d_{11} & \sigma^2 + d_{11} \end{pmatrix}. \qquad (16.7)$$

Consequently, the implied marginal variance-covariance structure is that of compound symmetry with a common correlation equal to $\varrho = d_{11}/(\sigma^2 + d_{11})$. Note that, because the variance component d_{11} is constrained to be nonnegative, ϱ is also forced to be nonnegative.

16.2.2 R *Syntax and Results*

In Panel R16.1, we use the function `lme()` to fit model **M16.1**, specified by (16.1)–(16.6).

The formula `lm2.form`, used in Panel R16.1, defines the fixed part of the model, as specified in (16.1), including an interaction between `time` and treatment. The factor `treat.f` is parameterized with "Placebo" as the reference level. The argument `random=~1|subject` specifies random subject-specific intercepts. By default, `lme()` assumes independent residual errors with a constant variance, σ^2. Also, because there is no `method` argument in the `lme()` function call, the default REML estimation is used. To change it to the ML estimation, we should add the `method="ML"` argument to the function call.

In addition to the model specification, in Panel R16.1, we also display the results of the fit of model **M16.1**. Note that the model formula is explicitly displayed in the printout. This was achieved by applying, before printing the results, the function `update()` to the object `fm16.1` to evaluate the formula `lm2.form` with the help of the function `eval()`. To simplify the code, this step is *not* shown in Panel R16.1.

Additionally, we print out the estimated fixed-effects table using the `printCoefmat()` function. The argument `has.Pvalue=TRUE` specifies that the last column of the table contains p-values which should be printed (`P.values=TRUE`). A description of all of the arguments of the function `printCoefmat()` can be obtained by issuing the command `?printCoefmat`.

R16.1 *ARMD Trial*: Model **M16.1** fitted using the function `lme()`

```
> lm2.form <-                                  # (16.1)
+    formula(visual ~ visual0 + time + treat.f + treat.f:time)
> (fm16.1 <-                                   # M16.1
+     lme(lm2.form,
+     random = ~1|subject, data = armd))       # b_0i:(16.5)
  Linear mixed-effects model fit by REML
    Data: armd
    Log-restricted-likelihood: -3289
    Fixed: visual ~ visual0 + time + treat.f + time:treat.f
         (Intercept)            visual0               time
           9.288078           0.826440          -0.212216
       treat.fActive time:treat.fActive
           -2.422000          -0.049591

  Random effects:
   Formula: ~1 | subject
          (Intercept) Residual
  StdDev:      8.9782   8.6275

  Number of Observations: 867
  Number of Groups: 234
> printCoefmat(summary(fm16.1)$tTable, # Print fixed-effects, etc.
+         has.Pvalue = TRUE, P.values = TRUE) # ... with p-values
                        Value Std.Error       DF t-value p-value
  (Intercept)          9.2881    2.6819 631.0000    3.46 0.00057
  visual0              0.8264    0.0447 231.0000   18.50 < 2e-16
  time                -0.2122    0.0229 631.0000   -9.26 < 2e-16
  treat.fActive       -2.4220    1.5000 231.0000   -1.61 0.10774
  time:treat.fActive  -0.0496    0.0336 631.0000   -1.48 0.14002
```

Results presented in Panel R16.1 indicate that the standard deviation $\sqrt{d_{11}}$ of the random intercepts, as specified in (16.6), is estimated to be equal to 8.98, while the residual standard deviation, σ, is estimated to be equal to 8.63. Note that, as was mentioned in Sect. 14.7, the p-values, corresponding to the t-test statistics for the fixed-effects coefficients, are for the marginal-approach tests. A summary of the REML-based estimates for model **M16.1** is also given in Table 16.1.

In Panel R16.2, we demonstrate how to extract information about the grouping of data or, equivalently, about the data hierarchy implied by the fitted model. By using the function `getGroupsFormula()` (see Panel R14.5), we obtain the conditioning expression used in the specification of the `random` argument. It indicates a single level of grouping, defined by the levels of the factor `subject`. By applying the function `getGroups()` to the model-fit object, we extract the grouping factor and store it in the object `grpF`. With the help of the function `str()`, we display the structure of the object. The printout implies that the grouping factor had 234 levels (subjects). Moreover, we can conclude that, e.g., for the first subject we had

Table 16.1 *ARMD Trial*: REML-based parameter estimates[a] for models **M16.1** and **M16.2** with subject-specific random intercepts

	Parameter	fm16.1	fm16.2
Model label		**M16.1**	**M16.2**
Log-REML value		−3288.99	−3260.56
Fixed effects			
Intercept	β_0	9.29(2.68)	7.07(2.30)
Visual acuity at t=0	β_1	0.83(0.04)	0.87(0.04)
Time (in weeks)	β_2	−0.21(0.02)	−0.21(0.03)
Trt(Actv *vs.* Plcb)	β_3	−2.42(1.50)	−2.31(1.24)
Tm × Treat(Actv)	β_4	−0.05(0.03)	−0.05(0.04)
reStruct(`subject`)			
SD(b_{i0})	$\sqrt{d_{11}}$	8.98(7.99,10.09)	7.71(6.83,8.69)
Variance function			
Power (TIME$^{\delta}$)	δ		0.31(0.23,0.39)
Scale	σ	8.63(8.16,9.12)	3.61(2.87,4.54)

[a]Approximate SE for fixed effects and 95% CI for covariance parameters are included in parentheses

R16.2 *ARMD Trial*: Data grouping/hierarchy implied by model **M16.1**. The model-fit object `fm16.1` was created in Panel R16.1

```
> getGroupsFormula(fm16.1)           # Grouping formula
  ~subject
  <environment: 0x000000001a310670>
> str(grpF <- getGroups(fm16.1))     # Grouping factor
   Factor w/ 234 levels "1","2","3","4",..: 1 1 2 2 2 2 3 3 3 4...
   - attr(*, "label")= chr "subject"
> grpF[1:17]
   [1] 1 1 2 2 2 2 3 3 3 4 4 4 4 6 6 6 6
  234 Levels: 1 2 3 4 6 7 8 9 10 11 12 13 14 15 16 17 18 19... 240
> levels(grpF)[1:5]
  [1] "1" "2" "3" "4" "6"
> range(xtabs(~grpF))              # Min, Max no. of observations
  [1] 1 4
```

two observations, for the second subject we had four observations, etc. Similar information is obtained by listing a subset of elements of the grouping factor. The minimum and maximum number of observations across all subjects are obtained by applying the function `range()` to the result of a cross tabulation of the levels of the factor `grpF`, provided by the function `xtabs()`.

To get more insight into the estimated variance-covariance structure of model **M16.1**, we use the `getVarCov()` and `VarCorr()` functions, as shown in Panel R16.3.

R16.3 *ARMD Trial*: The estimated variance-covariance matrices for random effects ($\mathcal{D}$) and residual errors ($\mathcal{R}_i$) for model **M16.1**. The model-fit object `fm16.1` was created in Panel R16.1

(a) *The $\mathcal{D}$-matrix estimate*

```
> getVarCov(fm16.1, individual = "2")   # d̂11:(16.6)
  Random effects variance covariance matrix
              (Intercept)
  (Intercept)      80.608
    Standard Deviations: 8.9782
> VarCorr(fm16.1)                       # d̂11, σ̂2
  subject = pdLogChol(1)
              Variance StdDev
  (Intercept) 80.608   8.9782
  Residual    74.434   8.6275
```

(b) *The $\mathcal{R}_i$-matrix estimate*

```
> getVarCov(fm16.1,
+           type = "conditional",       # R̂i:(16.6)
+           individual = "2")
  subject 2
  Conditional variance covariance matrix
        1      2      3      4
  1 74.434  0.000  0.000  0.000
  2  0.000 74.434  0.000  0.000
  3  0.000  0.000 74.434  0.000
  4  0.000  0.000  0.000 74.434
    Standard Deviations: 8.6275 8.6275 8.6275 8.6275
```

The `getVarCov()`-function call, used in Panel R16.3a, does not include the `type` argument (see Sect. 14.6 and Table 14.5). This means that the default value of the argument, i.e., `type="random.effect"`, is employed. As a result, the function provides the estimated variance-covariance matrix $\mathcal{D}$ of the random effects. In the case of model **M16.1**, it gives the estimated variance and standard deviation of the subject-specific random intercepts. The argument `individual="2"`, used in the `getVarCov()`-function call, requests the random effects variance-covariance matrix for the second individual, i.e., `subject==2`, in the analyzed dataset. In fact, in our case, the subject number is not of importance, as the variance-covariance structure of random effects is assumed to be the same for all individuals.

In Panel R16.3a, we also illustrate how to extract estimates of the $\mathcal{D}$ matrix elements using the function `VarCorr()` (see Sect. 14.6 and Table 14.5).

In Panel R16.3b, we specify the `type="conditional"` and `individual="2"` arguments in a call to the `getVarCov()` function. As a result, we obtain the

estimated variance-covariance matrix $\mathcal{R}_i$ of the residual random errors for the second subject. As noted previously, this subject has all four post-randomization visual acuity measurements, so a 4×4 matrix is reported. Because model **M16.1** assumes independent residual errors with the same variance at all measurement times, a diagonal matrix $\widehat{\mathcal{R}}_i = \widehat{\sigma}^2 \boldsymbol{I}_4 = 74.434 \times \boldsymbol{I}_4$ is displayed, as specified in (16.6).

Finally, in Panel R16.4, we obtain the estimated marginal variance-covariance matrix, defined in (16.7), by applying the function `getVarCov()` with the `type="marginal"` argument. The result, for `individual="2"`, is stored in the object `fm16.1cov` and displayed. The marginal variance is estimated by the sum of the estimated residual variance $\widehat{\sigma}^2 = 74.434$ and the variance of the random intercepts $\widehat{d}_{11} = 80.608$. The latter variance component becomes the covariance, as seen from (16.7).

The resulting marginal correlation matrix is obtained by applying the `cov2cor()` function (see Panel R14.4) to the first component of the list-object `fm16.1cov`, which contains the estimated marginal variance-covariance matrix. As noted earlier, the estimated marginal correlation matrix implies a constant, positive correlation coefficient equal to 0.52 for any two visual acuity measurements obtained for the same patient at different timepoints.

16.3 A Model with Random Intercepts and the varPower$(\cdot)$ Residual Variance-Function

As noted in the exploratory analysis (Sect. 3.2) and, e.g., in Chap. 12, the variability of visual acuity measurements increases in time. Therefore, we consider a model with variance of random errors expressed as a power function of the TIME covariate.

16.3.1 Model Specification

To specify the new model, labeled **M16.2**, we use the same fixed-effects part as in model **M16.1**. However, we modify the variance-covariance structure of residual random errors, specified in (16.6). More specifically, following the results obtained in Chaps. 9 and 12, we consider the use of the varPower$(\cdot)$ variance function, introduced in Sect. 7.3.1. Thus, we assume that

$$\mathcal{R}_i = \sigma^2 \begin{pmatrix} (\text{TIME}_{i1})^{2\delta} & 0 & 0 & 0 \\ 0 & (\text{TIME}_{i2})^{2\delta} & 0 & 0 \\ 0 & 0 & (\text{TIME}_{i3})^{2\delta} & 0 \\ 0 & 0 & 0 & (\text{TIME}_{i4})^{2\delta} \end{pmatrix}. \qquad (16.8)$$

R16.4 *ARMD Trial*: The estimated marginal variance-covariance matrix and the corresponding correlation matrix for model **M16.1**. The model-fit object `fm16.1` was created in Panel R16.1

```
> (fm16.1cov <-
+     getVarCov(fm16.1,
+               type = "marginal",            # V̂_i:(16.7)
+               individual = "2"))
  subject 2
  Marginal variance covariance matrix
          1       2       3       4
  1 155.040  80.608  80.608  80.608
  2  80.608 155.040  80.608  80.608
  3  80.608  80.608 155.040  80.608
  4  80.608  80.608  80.608 155.040
    Standard Deviations: 12.452 12.452 12.452 12.452
> (cov2cor(fm16.1cov[[1]]))                   # Corr(V̂_i)
          1       2       3       4
  1 1.00000 0.51991 0.51991 0.51991
  2 0.51991 1.00000 0.51991 0.51991
  3 0.51991 0.51991 1.00000 0.51991
  4 0.51991 0.51991 0.51991 1.00000
```

Note that $\mathcal{R}_i$, defined in (16.8), can be decomposed as $\mathcal{R}_i = \sigma^2 \Lambda_i C_i \Lambda_i$ using Λ_i, given in (12.3), and by setting $C_i = I_4$. It should be stressed here that the parameter σ^2, used in (16.8), can only be interpreted as a (unknown) scale parameter. This is in contrast to (16.6), where it could be interpreted as the variance of residual errors.

The matrix $\mathcal{R}_i$, given in (16.8), is diagonal with unequal elements defined by the varPower(·) function. Consequently, as compared to model **M16.1**, the marginal variance-covariance and correlation matrices of model **M16.2** have different structures. In particular, the marginal variance-covariance matrix becomes equal to

$$\mathcal{V}_i = \begin{pmatrix} \sigma_1^2 + d_{11} & d_{11} & d_{11} & d_{11} \\ d_{11} & \sigma_2^2 + d_{11} & d_{11} & d_{11} \\ d_{11} & d_{11} & \sigma_3^2 + d_{11} & d_{11} \\ d_{11} & d_{11} & d_{11} & \sigma_4^2 + d_{11} \end{pmatrix}, \tag{16.9}$$

where

$$\sigma_t^2 = \sigma^2 (\text{TIME}_{it})^{2\delta}.$$

It is worth observing that, because the variance changes with time, the marginal correlation coefficients between observations made at different times are no longer equal.

R16.5 *ARMD Trial*: Model **M16.2** fitted using the function `lme()`. The model-fit object `fm16.1` was created in Panel R16.1

```
> (fm16.2 <-                                          # M16.2 ← M16.1
+    update(fm16.1,
+           weights = varPower(form = ~ time),    # (9.4)
+           data = armd))
  Linear mixed-effects model fit by REML
    Data: armd
    Log-restricted-likelihood: -3260.6
    Fixed: visual ~ visual0 + time + treat.f + time:treat.f
         (Intercept)            visual0               time
           7.066881           0.866544          -0.212627
       treat.fActive time:treat.fActive
           -2.305034          -0.050888

  Random effects:
   Formula: ~1 | subject
          (Intercept) Residual
  StdDev:      7.7056   3.6067

  Variance function:
   Structure: Power of variance covariate
   Formula: ~time
   Parameter estimates:
    power
  0.31441
  Number of Observations: 867
  Number of Groups: 234
```

16.3.2 R *Syntax and Results*

In Panel R16.5, we fit model **M16.2**. More specifically, we update the object `fm16.1`, representing the fitted model **M16.1**, using the `weights = varPower(form = ~time)` argument in a call to the `update()` function. Note the use of the `varPower()` variance-function constructor (see Sect. 8.2) in the `weights` argument (see Sect. 14.5). Results of fitting model **M16.2** using REML are stored in the object `fm16.2`. Panel R16.5 presents a summary of the estimates of the model parameters. More detailed results are shown in Table 16.1.

The results, presented in Panel R16.5, indicate that the scale parameter σ is estimated to be equal to 3.607. The power coefficient δ of the `varPower()` variance function, as specified in (9.4), is estimated to be equal to 0.314. The estimate of the standard deviation of the random intercepts equals 7.706.

Panel R16.6 presents the estimates of the variance-covariance matrices associated with model **M16.2**. To obtain the estimate of the $\mathcal{D}$ matrix, we apply the

R16.6 *ARMD Trial*: The estimated $\mathcal{D}$, $\mathcal{R}_i$, and $\mathcal{V}_i$ matrices for model **M16.2**. The model-fit object `fm16.2` was created in Panel R16.5

```
> VarCorr(fm16.2)                                  # d̂11: (16.6), σ̂²
  subject = pdLogChol(1)
              Variance StdDev
  (Intercept) 59.376   7.7056
  Residual    13.008   3.6067
> getVarCov(fm16.2,                                # R̂i: (16.8)
+           type = "conditional",
+           individual = "2")
  subject 2
  Conditional variance covariance matrix
         1      2      3      4
  1 31.103  0.000  0.000   0.00
  2  0.000 62.062  0.000   0.00
  3  0.000  0.000 95.966   0.00
  4  0.000  0.000  0.000 156.05
    Standard Deviations: 5.577 7.8779 9.7962 12.492
> (fm16.2cov <-                                    # V̂i: (16.9)
+    getVarCov(fm16.2,
+              type = "marginal",
+              individual = "2"))
  subject 2
  Marginal variance covariance matrix
         1       2       3       4
  1 90.479  59.376  59.376  59.376
  2 59.376 121.440  59.376  59.376
  3 59.376  59.376 155.340  59.376
  4 59.376  59.376  59.376 215.430
    Standard Deviations: 9.512 11.02 12.464 14.677
> cov2cor(fm16.2cov[[1]])                          # Corr(V̂i)
          1       2       3       4
  1 1.00000 0.56645 0.50083 0.42529
  2 0.56645 1.00000 0.43230 0.36710
  3 0.50083 0.43230 1.00000 0.32457
  4 0.42529 0.36710 0.32457 1.00000
```

`VarCorr()` function. The estimated variance of random intercepts is equal to 59.376. Note that it is smaller than the value of 80.608, obtained for model **M16.1** (see Panel R16.3). This is expected, because, by allowing for heteroscedastic residual random errors, a larger part of the total variability is explained by the residual variances. The estimated variance-covariance matrix of the residual errors $\mathcal{R}_i$ is obtained using the `getVarCov()` function with the `type="conditional"`

Table 16.2 *ARMD Trial*: REML-based estimates[a] for linear mixed-effects models[b] with random intercepts and time slopes

	Parameter	fm16.3	fm16.4	fm16.5
Model label		**M16.3**	**M16.4**	**M16.5**
Log-REML value		−3215.30	−3215.90	−3214.47
Fixed effects				
Intercept	β_0	4.74(2.26)	5.26(2.27)	5.44(2.26)
Visual acuity at t=0	β_1	0.91(0.04)	0.90(0.04)	0.90(0.04)
Time (in weeks)	β_2	−0.22(0.03)	−0.22(0.03)	−0.24(0.02)
Trt(Actv *vs.* Plcb)	β_3	−2.26(1.15)	−2.28(1.17)	−2.66(1.13)
Tm × Treat(Actv)	β_4	−0.06(0.05)	−0.06(0.05)	
reStruct(`subject`)				
SD(b_{i0})	$\sqrt{d_{11}}$	6.98(5.99,8.13)	7.23(6.33,8.26)	7.24(6.33,8.27)
SD(b_{i1})	$\sqrt{d_{22}}$	0.27(0.23,0.32)	0.28(0.24,0.33)	0.28(0.24,0.33)
cor((Intercept),time)	ϱ_{12}	0.14(−0.13,0.38)		
Variance function				
Power (TIME$^{\delta}$)	δ	0.11(0.02,0.20)	0.11(0.01,0.21)	0.11(0.02,0.21)
Scale	σ_1	5.12(4.00,6.56)	5.03(3.90,6.49)	5.04(3.92,6.48)

[a]Approximate SE for fixed effects and 95% CI for covariance parameters are included in parentheses

[b]The variance function `varPower()` of the `time` covariate was used in all three models

argument. It corresponds to the matrix specified in (16.8). Thus, for instance, the first diagonal element of the $\widehat{\mathcal{R}}_i$ matrix is equal to $\hat{\sigma}^2 \cdot 4^{2\hat{\delta}} = 3.6067^2 \cdot 4^{2 \cdot 0.3144} = 31.103$.

The estimated marginal variance-covariance matrix, shown in Panel R16.6, corresponds to the matrix $\mathcal{V}_i$, given in (16.9). It is obtained by applying the `getVarCov()` function with the `type="marginal"` argument to the `fm16.2` model-fit object. The corresponding estimated marginal correlation matrix indicates a decreasing correlation between visual acuity measurements made at more distant timepoints. This agrees with the conclusion drawn for the final marginal model **M12.3**, defined by (12.3), (12.6), and (12.9), for which results are displayed in Table 12.2 and Panel R12.12. Note, however, that the direct comparison of the marginal variance-covariance matrices for models **M12.3** and **M16.2** is not appropriate. This is because the marginal variance-covariance matrix of model **M16.2**, displayed in Panel R16.6, is much more structured than that of model **M12.3**, printed in Panel R12.12. On the other hand, they both allow for marginal correlation coefficients, which depend on the time "distances", or "positions", of visual-acuity measurements.

To summarize the results of analyses presented in the current and the previous section, Table 16.1 displays REML-based parameter estimates for models **M16.1** and **M16.2**.

R16.7 *ARMD Trial*: Residual plots for model **M16.2**. The model-fit object `fm16.2` was created in Panel R16.5

(a) *Default residual plot of conditional Pearson residuals*

```
> plot(fm16.2)                          # Fig. 16.1
```

(b) *Plots (and boxplots) of Pearson residuals per time and treatment*

```
> plot(fm16.2,                          # Figure not shown
+      resid(., type = "pearson") ~ time | treat.f,
+      id = 0.05)
> bwplot(resid(fm16.2, type = "p") ~ time.f | treat.f, # Fig. 16.2
+        panel = panel.bwxplot2,        # User-defined panel (not shown)
+        data = armd)
```

(c) *Normal Q-Q plots of Pearson residuals and predicted random effects*

```
> qqnorm(fm16.2, ~resid(.) | time.f)    # Fig. 16.3
> qqnorm(fm16.2, ~ranef(.))             # Fig. 16.4
```

16.3.3 Diagnostic Plots

At this point, we might want to take a look at the goodness of fit of model **M16.2**. The fitted model is represented by the object `fm16.2`. The syntax for several residual plots is given in Panel R16.7.

The default residual plot for the object is obtained using the `plot()` command in Panel R16.7a and presented in Fig. 16.1. The plot displays the conditional Pearson residuals (Sect. 13.6.2) *versus* fitted values. As such, the plot is not very informative, because it pools all the residuals together, despite the fact that residuals obtained from the same individual are potentially correlated. However, it can serve for detecting, e.g., outliers. In Fig. 16.1, a group of such residuals can be seen in at the bottom and the top of the central part of the scatterplot.

A modified plot of the residuals for each timepoint and treatment group might be more helpful. Toward this end, we use the form of the `plot()`-function call shown in Panel R16.7b. Note that, in the plot formula, we apply the `type="pearson"` argument in the `resid()` function, which indicates the use of the Pearson residuals. Moreover, in the formula, we use the term `~time|treat` to obtain plots per treatment group over time in separate panels. Additionally, by applying the argument `id=0.05` to the `plot()` statement, we label the residuals larger, in absolute value, than the 97.5th percentile of the standard normal distribution by the number of the corresponding observation from the `armd` data frame.

Note that we do not present the resulting plot. Instead, in Fig. 16.2, we present its enhanced version, with box-and-whiskers plots superimposed over a stripplot of the

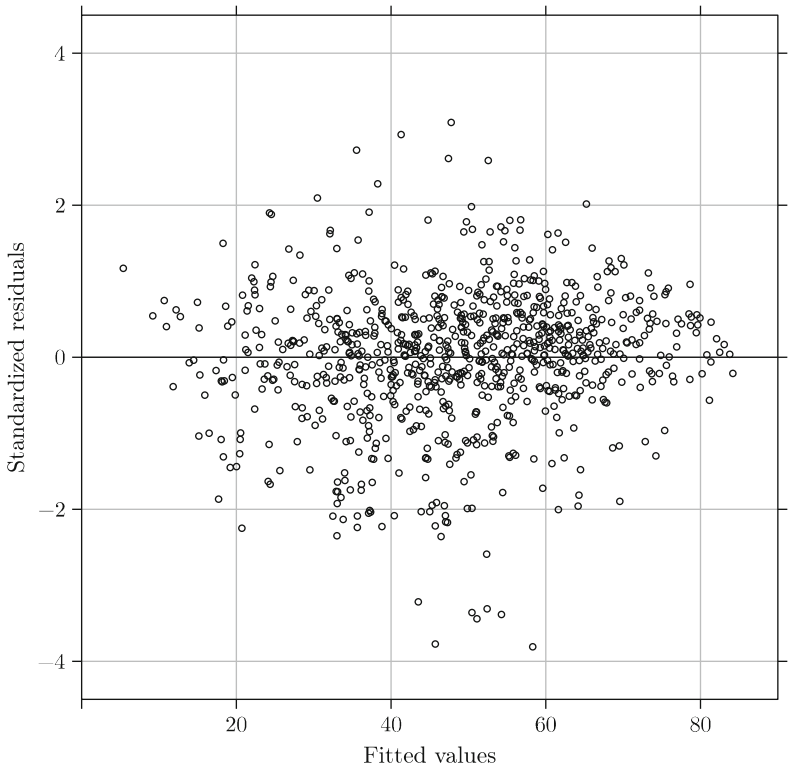

Fig. 16.1 *ARMD Trial*: Scatterplot of the conditional Pearson residuals for model **M16.2**

residuals for each timepoint and treatment group. Toward this end, in Panel R16.7b, we use the function `bwplot()` from the package **lattice** (Sect. 3.2.2). In the first argument of `bwplot()`, we use a formula requesting a plot of the Pearson residuals *versus* the levels of the `time.f` factor, separately for the levels of the `treat.f` factor. The residuals are extracted from the model-fit object `fm16.2` by applying the `resid()` function (Sect. 5.5). The key component of the `bwplot()`-function call is an auxiliary panel-function `panel.bwxplot2`. Due to the complexity of the R code used to create the panel function, we do not present it; however, the code is available in the package **nlmeU** containing the supplementary materials for the book.

Figure 16.2 allows for an evaluation of the distribution of the conditional Pearson residuals for each timepoint and treatment group. Despite standardization, the variability of the residuals seems to vary. The plot reveals also a number of outliers, i.e., residuals larger, in absolute value, than the 97.5th percentile of the standard normal distribution (they have been labeled in the plot by the corresponding observation number). However, given the large number of observations, one might

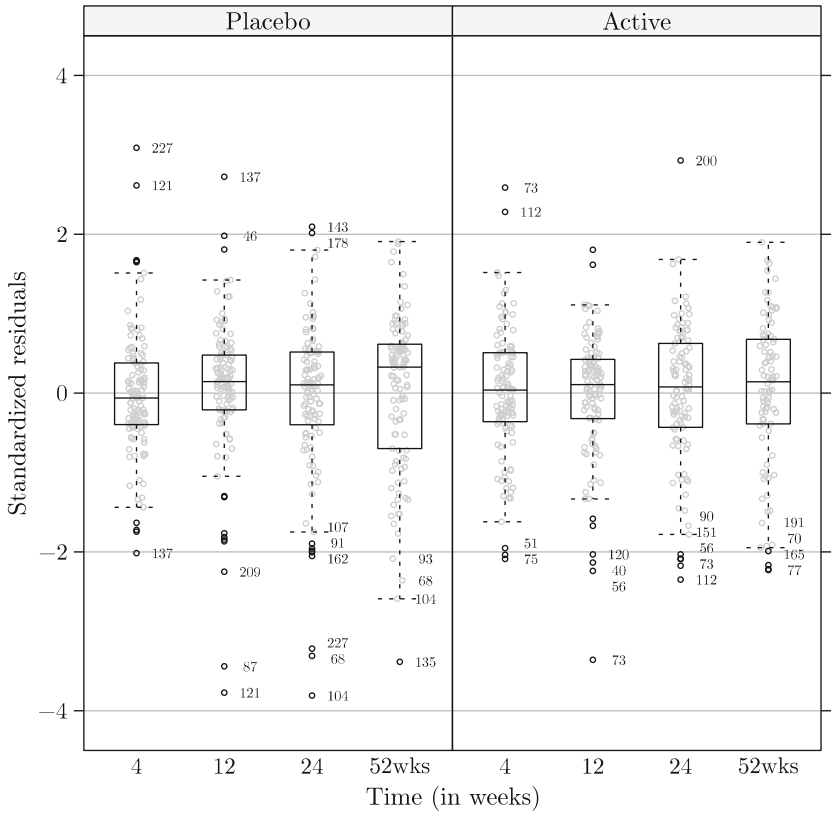

Fig. 16.2 *ARMD Trial*: Stripplots (and box-and-whiskers plots) of the conditional Pearson residuals for each timepoint and treatment group for model **M16.2**

expect a group of outlying values. It is worth noting that the outliers are present in all treatment groups and at all timepoints.

Panel R16.8 lists the subjects for whom outlying residuals were labeled in Fig. 16.2. Toward this end, the conditional Pearson residuals are extracted from the model-fit object `fm16.2` and stored in the vector `resid.p`. Indices for the residuals larger, in absolute value, than the 97.5th percentile of the standard normal distribution are stored in the logical vector `idx`. The data frame `outliers.idx` contains selected variables from the `armd` dataset together with the residuals and the logical index vector. The data frame `outliers` is a subset of `outliers.idx` and contains observations for which the value of the variable `idx`, given as the second argument of the function `subset()`, is equal to 1. There are 38 such observations, for which the value of the `subject` number is printed out. Note that, for several subjects, there is more than one outlying residual, because there is more than one visual acuity measurement possible per subject.

R16.8 *ARMD Trial*: The list of outlying conditional Pearson residuals for model **M16.2**. The model-fit object `fm16.2` was created in Panel R16.5

```
> id <- 0.05                                    # Argument for qnorm()
> outliers.idx <-
+  within(armd,
+         {
+         resid.p <- resid(fm16.2, type = "pearson") # Pearson resids.
+         idx <- abs(resid.p) > -qnorm(id/2)         # Indicator vector
+         })
> outliers  <- subset(outliers.idx, idx)     # Data with outliers
> nrow(outliers)                             # Number of outliers
  [1] 38
> outliers$subject                           # IDs of outliers
   [1] 40  46  51  56  56  68  68  70  73  73  73  75  77  87  90
  [16] 91  93  104 104 107 112 112 120 121 121 135 137 137 143 151
  [31] 162 165 178 191 200 209 227 227
  234 Levels: 1 2 3 4 6 7 8 9 10 11 12 13 14 15 16 17 18 19... 240
```

Figure 16.3 shows the normal Q-Q plot of the conditional Pearson residuals per timepoint. The plot was obtained using the first `qqnorm()`-function call shown in Panel R16.7c. The patterns do show some deviations from a linear trend.

We can also look at the normal Q-Q plot of the predicted random effects (random intercepts). The effects are estimated by EBLUPs (Sect. 13.6.1). They can be extracted from the `fm16.2` model-fit object using the function `ranef()` (see Sect. 14.6 and Table 14.5), as shown in the second `qqnorm()`-function call shown in Panel R16.7c. The resulting Q-Q plot is shown in Fig. 16.4 and is slightly curvilinear. This could be taken as an indication of nonnormality of the random effects. However, as mentioned in Sect. 13.6.1, such a plot may not necessarily reflect the true distribution of the random effects. Hence, it should be interpreted with caution.

An important diagnostic plot is presented in Fig. 16.5. It shows the observed and predicted values of the visual acuity measurements for selected patients. Panel R16.9 demonstrates how to generate the object containing the data necessary for constructing the figure using the `augPred()` function.

The function `augPred()` allows obtaining predicted values for the object specified as the first argument. The object can be of class *lmList* (14.5), *gls* (11.6), and *lme* (14.6). If the object has a grouping structure, the predicted values are obtained for each group. Conveniently, the function adds the original observations to the returned object, which is a data frame with four columns containing the values of the primary covariate, the groups, the predicted or observed values, and the indicator of the type of the value from the third column.

The optional arguments of the function `augPred()` include: `primary`, `level`, `length.out`, `minimum`, and `maximum`. The argument `primary` is a one-sided formula indicating the covariate at which values the predicted values should be

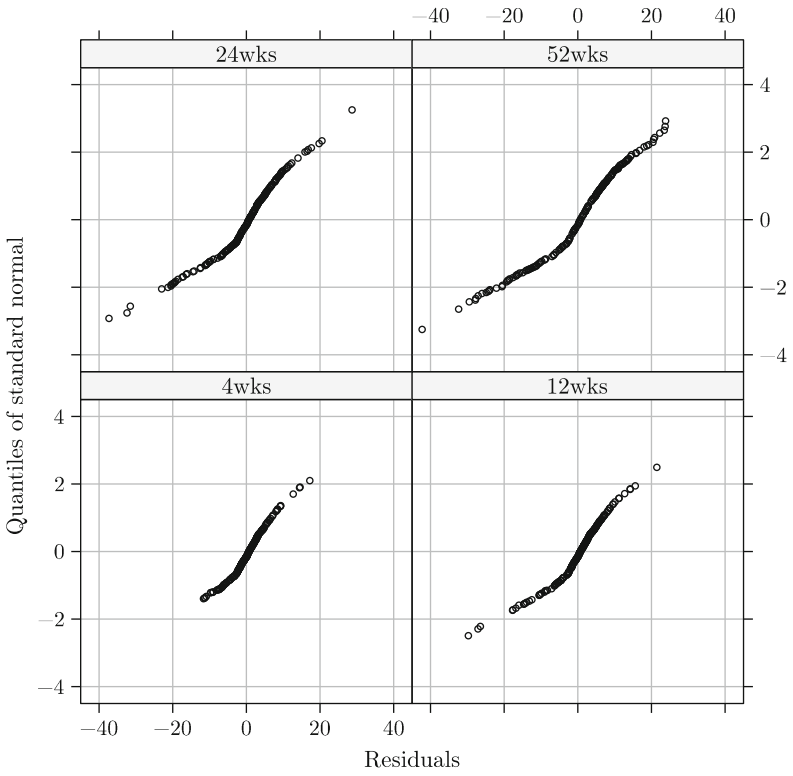

Fig. 16.3 *ARMD Trial*: Normal Q-Q plots of the conditional Pearson residuals for each timepoint for model **M16.2**

computed. In the call presented in Panel R16.9, we indicate that the predicted values should be computed at the values of the variable `time`.

The arguments `minimum` and `maximum` allow for providing the lower and upper limit, respectively, for the values of the primary covariate at which the predicted values are to be computed. By default, the arguments become equal to, respectively, the minimum and maximum of the values of the covariate. In the call presented in Panel R16.9, we use the default values of the arguments, i.e., the minimum and maximum values of the `time` variable, which are equal to, respectively, 4 and 52 weeks.

The argument `level` of the function `augPred()` is an integer vector specifying the grouping levels for which the predicted values are to be computed. Its interpretation is the same as for the function `predict()` (Sect. 14.6). In the `augPred`-function call shown in Panel R16.9, we use `level=0:1`, which amounts to specifying that the predicted values should be computed at the level 0, i.e., the population level, and at the level 1, i.e., the subject level.

Finally, the `length.out` argument is an integer indicating the number of values of the primary covariate at which the predictions should be evaluated. By default,

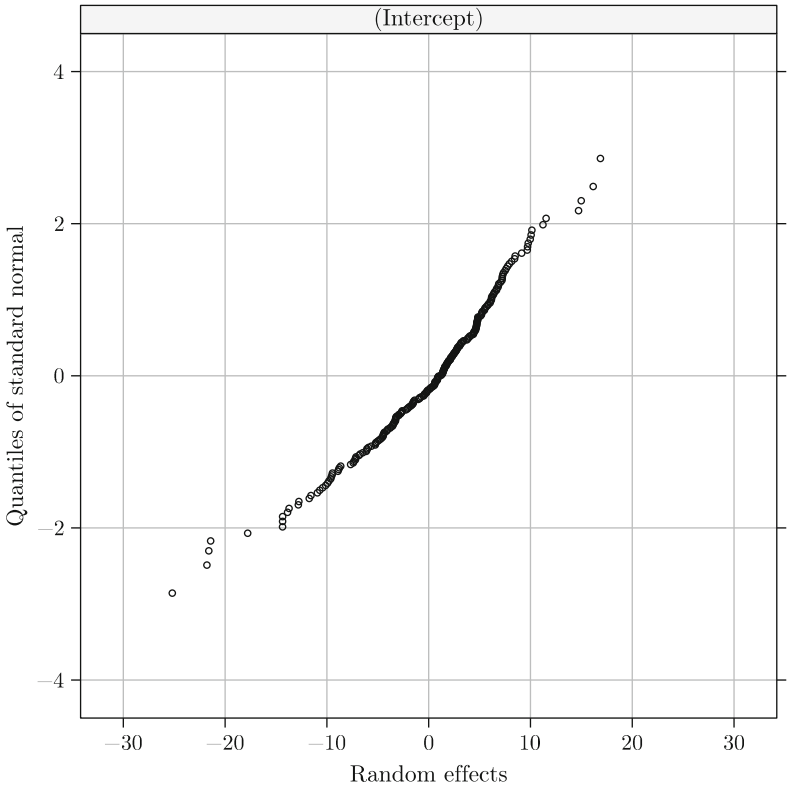

Fig. 16.4 *ARMD Trial*: The normal Q-Q plot of the predicted random intercepts for model **M16.2**

R16.9 *ARMD Trial*: Predicted visual acuity values for model **M16.2**. The model-fit object `fm16.2` was created in Panel R16.5

```
> aug.Pred <-                      # augPred for M16.2
+    augPred(fm16.2,
+            primary = ~time,      # Primary covariate
+            level = 0:1,          # Marginal(0) and subj.-spec.(1)
+            length.out = 2)
> plot(aug.Pred, layout = c(4, 4, 1), # Fig. 16.5
+      key = list(lines = list(lty = c(1,2)),
+                 text = list(c("Marginal", "Subject-specific")),
+                 columns = 2))
```

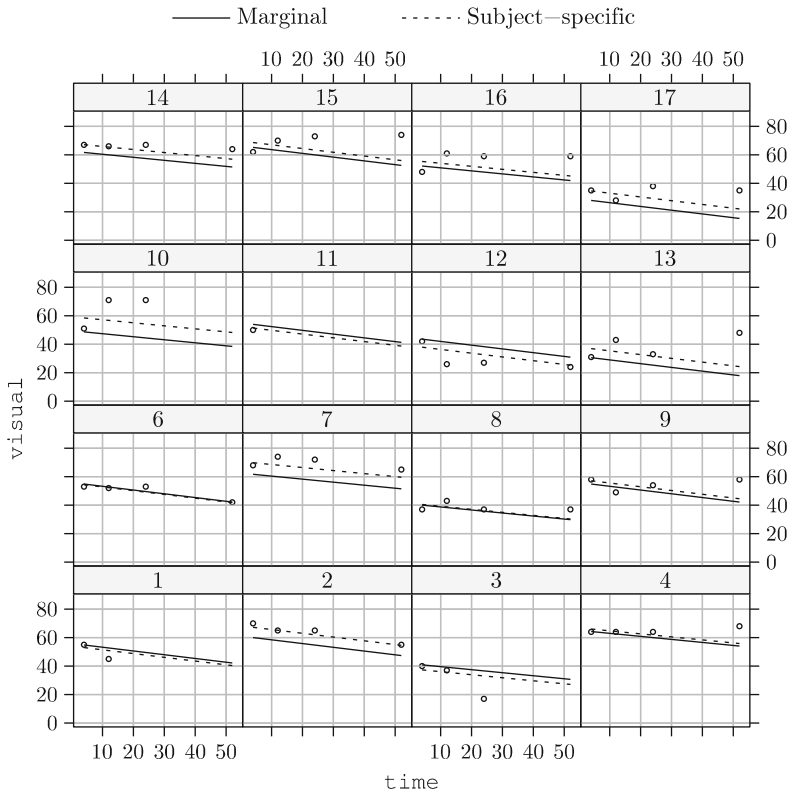

Fig. 16.5 *ARMD Trial*: Observed and predicted values of visual acuity for selected patients for model **M16.2**

it assumes the value 51. In Panel R16.9, we set `length.out=2`, i.e., the predicted values are obtained at two values `time`, i.e., at the minimum (4 weeks) and the maximum (52 weeks). The two predicted values are sufficient to describe the (population- and subject-specific) linear trend in (continuous) time, implied by the fitted form of model **M16.2**.

By applying, in Panel R16.9, the `plot()` function to the object `aug.Pred` with the `level=0:1` argument, a plot of the population-level and within-subject predicted values is obtained. The argument `layout=c(4,4,1)` requests one page of plots, arranged in four rows with four plots each. Each plot corresponds to a single subject; thus, the predictions for the first 16 subjects are plotted. Finally, the `key` argument allows specifying the legend, which is placed at the top of the graph. We refer the reader to the R help system for the `xyplot()` function from the **lattice** package for a detailed description of other available arguments.

The resulting plot is shown in Fig. 16.5. The predicted population means, shown in the plot, decrease linearly in time. This is consistent with the trend observed in Fig. 3.2. According to the assumed structure of the model, the population means are shifted for individual patients by subject-specific random intercepts.

Note that, as a result, the slopes of the individual profiles are the same for all subjects. Consequently, all subject-specific lines are parallel to lines representing the population means. For some patients, the so-obtained predicted individual profiles strongly deviate from the observed ones. For instance, for the subjects 4 and 15, the predicted individual patterns suggest a decrease of visual acuity over time, while the observed values actually increase over time.

A possible way to improve the individual predictions is to allow not only for patient-specific random intercepts, but also for patient-specific random slopes. We focus on this issue in the next section.

16.4 Models with Random Intercepts and Slopes and the varPower$(\cdot)$ Residual Variance-Function

In this section, we consider a model with two subject-specific random effects: a random intercept and a random slope for time. We use two variance-covariance structures $\mathcal{D}$ for the random effects, namely, a general one and a diagonal one. By using the varPower$(\cdot)$ variance function, the residual variances are allowed to differ between different timepoints.

16.4.1 Model with a General Matrix $\mathcal{D}$

To specify model **M16.3** with a general variance-covariance matrix $\mathcal{D}$, we modify the model equation (16.1) as follows:

$$\begin{aligned} \text{VISUAL}_{it} &= \beta_0 + \beta_1 \times \text{VISUAL0}_i + \beta_2 \times \text{TIME}_{it} + \beta_3 \times \text{TREAT}_i \\ &\quad + \beta_4 \times \text{TREAT}_i \times \text{TIME}_{it} \\ &\quad + b_{0i} + b_{2i} \times \text{TIME}_{it} + \varepsilon_{it}. \end{aligned} \tag{16.10}$$

The equation (16.10) can be written in the form of (13.1)–(13.3), upon defining $\mathbf{y}_i$, $\boldsymbol{X}_i$, $\boldsymbol{\varepsilon}_i$, and $\boldsymbol{\beta}$ as in (16.3)–(16.5), but with

$$\boldsymbol{Z}_i = \begin{pmatrix} 1 & 4 \\ 1 & 12 \\ 1 & 24 \\ 1 & 52 \end{pmatrix}, \quad \mathbf{b}_i = \begin{pmatrix} b_{0i} \\ b_{2i} \end{pmatrix}, \tag{16.11}$$

and with the variance-covariance structure of the random effects given by

$$\mathbf{b}_i \sim \mathcal{N}(\mathbf{0}, \mathcal{D}) \quad \text{and} \quad \boldsymbol{\varepsilon}_i \sim \mathcal{N}(\mathbf{0}, \mathcal{R}_i), \tag{16.12}$$

where

$$\mathcal{D} = \begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{pmatrix} \tag{16.13}$$

and $\mathcal{R}_i$ is given by (16.8).

Note that the assumed form of $\mathcal{D}$ implies that the random intercepts and slopes are correlated. For instance, a positive correlation between b_{0i} and b_{2i} means that, for individuals with a higher initial value of visual acuity, the post-randomization measurements will increase more rapidly or decrease more slowly than for patients with a lower initial value.

It is worth reflecting on the marginal variance-covariance structure implied by model **M16.3**. According to this model, the marginal covariance between visual acuity measurements for the subject i at times t_1 and t_2 ($t_1, t_2 = 1, 2, 3, 4$) can be written as follows:

$$\begin{aligned} \text{Cov}(y_{it_1}, y_{it_2}) &= \begin{pmatrix} 1 & \text{TIME}_{it_1} \end{pmatrix} \mathcal{D} \begin{pmatrix} 1 \\ \text{TIME}_{it_2} \end{pmatrix} + \text{I}(t_1 = t_2)\sigma^2 (\text{TIME}_{it_1})^{2\delta} \\ &= d_{11} + d_{12}(\text{TIME}_{it_1} + \text{TIME}_{it_2}) + d_{22}\text{TIME}_{it_1}\text{TIME}_{it_2} \\ &\quad + \text{I}(t_1 = t_2)\sigma^2 (\text{TIME}_{it})^{2\delta}, \end{aligned} \tag{16.14}$$

where $\text{I}(A)$ is the indicator function for condition A. Hence, the marginal variance of visual acuity measurements for the subject i at the time t can be expressed as

$$\text{Var}(y_{it}) = d_{11} + 2d_{12}\text{TIME}_{it} + d_{22}\text{TIME}_{it}^2 + \sigma^2 (\text{TIME}_{it})^{2\delta}. \tag{16.15}$$

Thus, the variance becomes a power function, including a quadratic component, of the measurement time.

In Panel R16.10, we fit model **M16.3**, defined by (16.10)–(16.13), by updating the object `fm16.2`. Specifically, we use the syntax `random = ~ 1 + time | subject` to specify the random-effects structure (Sect. 14.3.1). By applying this particular formula in the `random` argument, we imply that, for each level of the `subject` grouping variable, a random intercept and a random slope for time are to be considered, with a (default) general variance-covariance matrix $\mathcal{D}$ represented by an object of class *pdLogChol*.

The basic results of fitting model **M16.3** are displayed in Panel R16.10. More details are shown in Table 16.2. In Panel R16.10, we also present the 95% CIs for the variance-function and correlation-structure parameters. They are computed using the methods described in Sect. 13.7.3. The results show a low estimated value of the correlation coefficient for the random effects b_{0i} and b_{2i}, equal to 0.138. The confidence interval for the correlation coefficient suggests that, in fact, the two random effects can be uncorrelated. Therefore, in the next section, we consider a simplified form of the $\mathcal{D}$ matrix.

R16.10 *ARMD Trial*: The estimated $\widehat{\mathcal{D}}$ matrix and confidence intervals for the $\boldsymbol{\theta}_D$ parameters for model **M16.3**. The model-fit object `fm16.2` was created in Panel R16.5

```
> fm16.3 <-                                  # M16.3 ← M16.2
+    update(fm16.2,
+           random = ~1 + time | subject,
+           data = armd)
> getVarCov(fm16.3, individual = "2")      # D̂: (16.16)
  Random effects variance covariance matrix
              (Intercept)    time
  (Intercept)    48.70500 0.26266
  time            0.26266 0.07412
    Standard Deviations: 6.9789 0.27225
> intervals(fm16.3, which = "var-cov")    # 95% CI for θ_D, δ: (16.8), σ
  Approximate 95% confidence intervals

   Random Effects:
    Level: subject
                            lower    est.   upper
  sd((Intercept))        5.99019 6.97891 8.13082
  sd(time)               0.23009 0.27225 0.32213
  cor((Intercept),time) -0.12564 0.13824 0.38386

   Variance function:
           lower    est.  upper
  power 0.015191 0.10744 0.1997
  attr(,"label")
  [1] "Variance function:"

   Within-group standard error:
   lower   est.  upper
  3.9993 5.1222 6.5604
```

16.4.2 Model with a Diagonal Matrix $\mathcal{D}$

In this section, we consider model **M16.4**, which, similarly to model **M16.3**, is defined by (16.10), but for which we specify that

$$\mathcal{D} = \begin{pmatrix} d_{11} & 0 \\ 0 & d_{22} \end{pmatrix}. \tag{16.16}$$

Thus, we assume that random intercepts b_{0i} and random slopes b_{1i} have different variances and are uncorrelated.

R16.11 *ARMD Trial*: Confidence intervals for the parameters of model **M16.4**. The model-fit object `fm16.3` was created in Panel R16.10

```
> fm16.4 <-                                  # M16.4 ← M16.3
+     update(fm16.3,
+            random = list(subject = pdDiag(~time)), # Diagonal D
+            data = armd)
> intervals(fm16.4)                          # 95% CI for β, θ_D, δ, σ
  Approximate 95% confidence intervals

   Fixed effects:
                         lower      est.     upper
  (Intercept)          0.81277  5.262213  9.711655
  visual0              0.82464  0.899900  0.975157
  time                -0.27954 -0.215031 -0.150524
  treat.fActive       -4.58882 -2.278756  0.031308
  time:treat.fActive  -0.15055 -0.056451  0.037646
  attr(,"label")
  [1] "Fixed effects:"

   Random Effects:
    Level: subject
                     lower    est.   upper
  sd((Intercept)) 6.33067 7.23195 8.26153
  sd(time)        0.24108 0.28096 0.32744

   Variance function:
           lower    est.   upper
  power 0.014823 0.11108 0.20733
  attr(,"label")
  [1] "Variance function:"

   Within-group standard error:
   lower   est.  upper
  3.8979 5.0312 6.4939
```

To fit model **M16.4**, we use the constructor-function `pdDiag()`. The function creates an object of class *pdDiag*, representing a diagonal positive-definite matrix (Sect. 14.2.1). Thus, in Panel R16.11, we update the object `fm16.3`, which represents model **M16.3**, using the argument `random=pdDiag(~time)`. By specifying the argument, we imply a diagonal form of the variance-covariance matrix $\mathcal{D}$ of the random intercepts and slopes (Sect. 14.3.1).

Panel R16.11 presents the 95% CIs for all the parameters of model **M16.4**. They suggest that the mean structure could be simplified by removing the `time:treat.f` interaction. More detailed results for the model are provided in Table 16.2.

R16.12 *ARMD Trial*: Testing a null hypothesis about the $\boldsymbol{\theta}_D$ parameters for model **M16.4**. The model-fit object `fm16.3` was created in Panel R16.10

```
> anova(fm16.4, fm16.3)                # H0: d12=0 (M16.4 ⊂ M16.3)
        Model df    AIC    BIC  logLik   Test L.Ratio p-value
fm16.4      1  9 6449.8 6492.6 -3215.9
fm16.3      2 10 6450.6 6498.2 -3215.3 1 vs 2   1.194  0.2745
```

In Panel R16.12, we use the REML-based LR test (Sect. 13.7.2) to verify the null hypothesis that in the matrix $\mathcal{D}$, defined in (16.13), the element $d_{12} = 0$. Toward this end, we apply the `anova()` function to the objects `fm16.4` and `fm16.3`, which represent the fitted models **M16.4** (null) and **M16.3** (alternative), respectively. We note that both models have the same mean structure so that the use of the REML-based LR test is justified. In addition to information criteria and REML values for both models, the results of the LR test, which is based on models **M16.3** and **M16.4**, are displayed. Given that the null hypothesis specifies a value inside the parameter space, the asymptotic χ^2 distribution with one degree of freedom can be used to assess the outcome of the test (Sect. 13.7.2). The result is not statistically significant at the 5% significance level. It indicates that, by assuming a simpler, diagonal structure of the matrix $\mathcal{D}$, we do not worsen the fit of the model. This conclusion is in agreement with the computed values of AIC: the value of 6,450.6 for model **M16.3** is slightly larger than the value of 6,449.8 for model **M16.4**, which indicates a slightly better fit of the latter model.

Note that, according to model **M16.4** and (16.15), the marginal variance of visual acuity for the subject i at time t can be written as

$$\text{Var}(y_{it}) = d_{11} + d_{22}\text{TIME}_{it}^2 + \sigma^2(\text{TIME}_{it})^{2\delta}.$$

Consequently, given that $\hat{\delta} = 0.11$, the implied marginal variance function is predominantly a quadratic function over time. As d_{11}, d_{22}, and σ^2 are necessarily positive, the function increases with time, which is in agreement with the observation made in the exploratory analysis (see, e.g., Panel R3.6 in Sect. 3.2).

Figure 16.6 presents the conditional Pearson residuals for model **M16.4**. As compared to the similar plot for model **M16.2** (see Fig. 16.1), it shows fewer residuals with an absolute value larger than the 97.5th percentile of the standard normal distribution.

Figure 16.7 presents the normal Q-Q plot of the conditional Pearson residuals per timepoint for model **M16.4**. The plot looks comparable to the corresponding plot for model **M16.2** shown in Fig. 16.3.

Note that Figs. 16.6 and 16.7 were constructed using the syntax similar to the one presented in Panels R16.7b and R16.7c, respectively. Thus, we do not present the details of the syntax for the two figures.

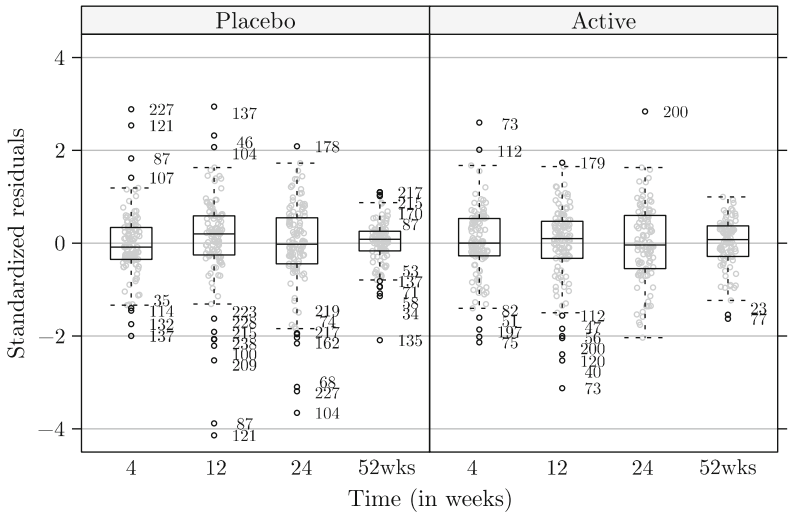

Fig. 16.6 *ARMD Trial*: Stripplots (and box-and-whiskers plots) of the conditional Pearson residuals for each timepoint and treatment group for model **M16.4**

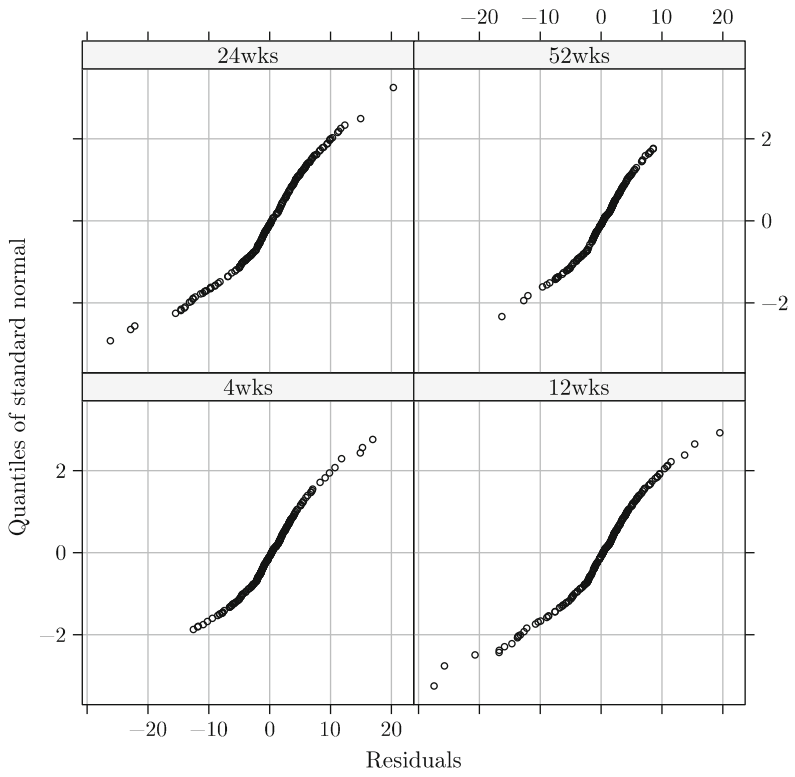

Fig. 16.7 *ARMD Trial*: Normal Q-Q plots of the conditional Pearson residuals for each timepoint for model **M16.4**

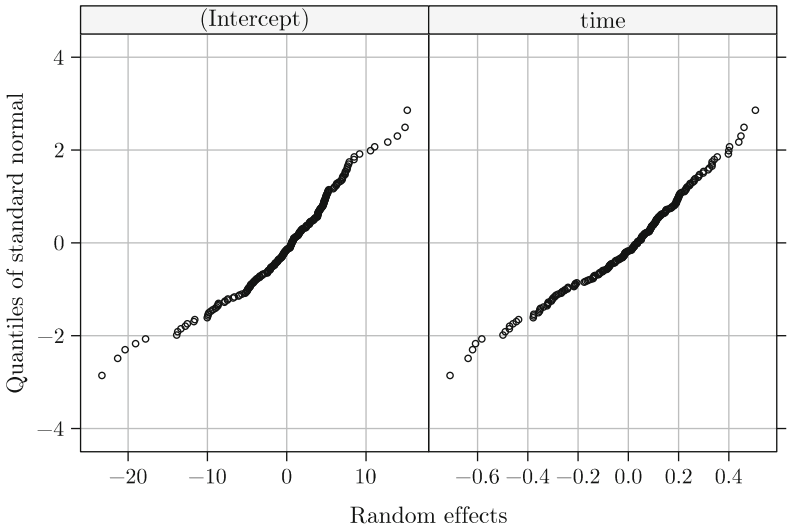

Fig. 16.8 *ARMD Trial*: Normal Q-Q plots of the predicted random effects for model **M16.4**

Figure 16.8 presents the normal Q-Q plots of the predicted random effects for model **M16.4**. The plots were obtained by using the following form of the `qqnorm()`-function call:

```
> qqnorm(fm16.4, ~ranef(.))          # Fig. 16.8
```

As a result, two plots are produced: one for the random intercepts and one for the random slopes. The latter is slightly closer to a straight line than the former. It is worth noting that the plot for the random intercepts resembles the one obtained for model **M16.2** (see Fig. 16.4). Recall that, as mentioned in Sect. 13.6.1, we should interpret the graphs with caution, because they may not necessarily reflect the correct distribution of the random effects.

Finally, Fig. 16.9 presents the predicted marginal and subject-specific values for model **M16.4**. Recall that, for model **M16.2**, a similar plot (see Fig. 16.5) showed a decreasing slope of the individual profiles, the same for all subjects. As a consequence, for some patients, e.g., no. 4 and 15, the so-obtained predicted individual profiles strongly deviated from the observed ones. This is not the case of the profiles shown in Fig. 16.9, for which the slopes vary. As a result, the predicted individual profiles follow more closely the observed values and capture, e.g., increasing trends in time. This illustrates that model **M16.4** offers a better fit to the data than model **M16.2**.

Given the satisfactory fit of model **M16.4**, in the next section, we focus on the inference about the mean structure of the model.

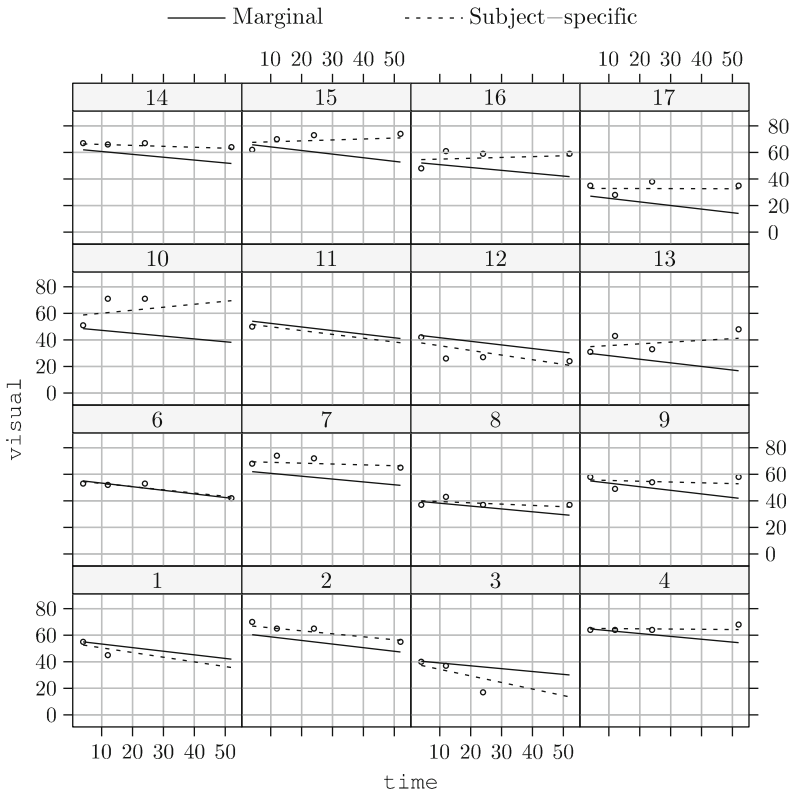

Fig. 16.9 *ARMD Trial*: Observed and predicted values of visual acuity for selected patients for model **M16.4**

16.4.3 Model with a Diagonal Matrix $\mathcal{D}$ and a Constant Treatment Effect

As mentioned in Sect. 16.4.2, the mean structure of model **M16.4** could be simplified by removing the $\text{TREAT}_i \times \text{TIME}_{it}$ interaction (see Panel R16.11). Toward this end, we specify model **M16.5** by modifying (16.10) as follows:

$$\begin{aligned}\text{VISUAL}_{it} = {} & \beta_0 + \beta_1 \times \text{VISUAL0}_i + \beta_2 \times \text{TIME}_{it} + \beta_3 \times \text{TREAT}_i \\ & + b_{0i} + b_{2i} \times \text{TIME}_{it} + \varepsilon_{it}. \end{aligned} \tag{16.17}$$

As compared to (16.10), (16.17) does not contain the $\beta_4 \times \text{TREAT}_i \times \text{TIME}_{it}$ interaction term in the fixed-effects part. Note that we keep all other elements of the model specification as for model **M16.4**. In particular, the variance-covariance matrix $\mathcal{D}$ is given by (16.16).

To fit model **M16.5**, we modify the LM formula and update the object `fm16.4` using the new formula object. The syntax is presented in Panel R16.13. The panel

R16.13 *ARMD Trial*: Fixed-effects estimates, their approximate standard errors, and 95% confidence intervals for the variance-covariance parameters of model **M16.5**. The model-fit object `fm16.4` was created in Panel R16.11

```
> lm3.form <- formula(visual ~ visual0 + time + treat.f) # (12.9)
> fm16.5  <-                                  # M16.5 ← M16.4
+    update(fm16.4,
+           lm3.form, data = armd)
> summary(fm16.5)$tTable                      # β̂, se(β̂), t-test
                   Value Std.Error  DF  t-value    p-value
  (Intercept)    5.44156  2.261866 632   2.4058 1.6424e-02
  visual0        0.89983  0.038215 231  23.5464 2.5503e-63
  time          -0.24156  0.023917 632 -10.0997 2.4641e-22
  treat.fActive -2.65528  1.128683 231  -2.3525 1.9485e-02
> intervals(fm16.5, which = "var-cov")        # 95% CI for θ_D, δ, σ
  Approximate 95% confidence intervals

   Random Effects:
    Level: subject
                     lower    est.  upper
  sd((Intercept)) 6.33448 7.23570 8.2651
  sd(time)        0.24121 0.28102 0.3274

   Variance function:
           lower    est.   upper
  power 0.015687 0.11052 0.20535
  attr(,"label")
  [1] "Variance function:"

   Within-group standard error:
   lower   est.  upper
  3.9177 5.0391 6.4815
```

also shows the results of the *t*-tests for the fixed effects (Sect. 13.7.1). Note that these are the marginal-approach tests (Sect. 5.6). Thus, the effect of each covariate is tested under the assumption that all other covariates are included in the model as well. The result of the test for the `treat.f` factor is statistically significant at the 5% significance level. It suggests a time-independent, negative average effect of the active treatment. This finding is in agreement with the results of the exploratory analysis (Sect. 3.2) and of the previous analysis using an LM with fixed effects for correlated data (Chap. 12). Note that the point estimates of the fixed effects, shown in Panel R16.13, are close to the corresponding estimates obtained for the final model **M12.3** for correlated data (see Table 12.2).

Panel R16.13 also presents the 95% CIs for all the variance-covariance parameters of model **M16.5**. The point estimates and intervals are very close to

R16.14 *ARMD Trial*: The estimates of matrices $\mathcal{D}$, $\mathcal{R}_i$, and $\mathcal{V}_i$ for model **M16.5**. The model-fit object `fm16.5` was created in Panel R16.13

```
> VarCorr(fm16.5)                                  # D̂: (16.16), σ̂
  subject = pdDiag(time)
              Variance  StdDev
  (Intercept) 52.355293 7.23570
  time         0.078974 0.28102
  Residual    25.392868 5.03913
> getVarCov(fm16.5,                                # R̂_i: (16.8)
+           type = "conditional", individual = "2")
  subject 2
  Conditional variance covariance matrix
         1     2      3      4
  1 34.498  0.00  0.000  0.000
  2  0.000 43.98  0.000  0.000
  3  0.000  0.00 51.262  0.000
  4  0.000  0.00  0.000 60.816
    Standard Deviations: 5.8735 6.6317 7.1597 7.7984
> (fm16.5cov <-                                    # V̂_i: (16.9)
+    getVarCov(fm16.5,
+              type = "marginal",
+              individual = "2"))
  subject 2
  Marginal variance covariance matrix
         1       2       3       4
  1 88.117  56.146  59.937  68.782
  2 56.146 107.710  75.100 101.640
  3 59.937  75.100 149.110 150.920
  4 68.782 101.640 150.920 326.720
    Standard Deviations: 9.387 10.378 12.211 18.075
> cov2cor(fm16.5cov[[1]])                          # Corr(V̂_i)
          1       2       3       4
  1 1.00000 0.57633 0.52290 0.40538
  2 0.57633 1.00000 0.59261 0.54180
  3 0.52290 0.59261 1.00000 0.68375
  4 0.40538 0.54180 0.68375 1.00000
```

those displayed in Panel R16.11 for model **M16.4**. This is not surprising, given that the two models differ only slightly with respect to their mean structure.

Another summary of estimates of the parameters of model **M16.5** is given in Table 16.2, which also contains estimated parameters of models **M16.3** and **M16.4**.

Panel R16.14 displays the estimated forms of matrices $\mathcal{D}$, $\mathcal{R}_i$, and $\mathcal{V}_i$ for model **M16.5**. The estimated marginal variance-covariance matrix $\widehat{\mathcal{V}}_i$ indicates an increasing trend of variances of visual acuity measurements over time, while the

corresponding correlation matrix suggests a decreasing correlation between the measurements obtained at more distant timepoints. These findings are in agreement with the results of the exploratory analysis (Sect. 3.2) and with the results obtained for model **M12.3** for correlated data (Table 12.2). Note, however, that a direct comparison of the estimated marginal matrices to their counterparts obtained for model **M12.3** is not appropriate, because the matrices for model **M16.5** are much more structured than those of model **M12.3** (see a similar comment in Sect. 16.3.2).

16.5 An Alternative Residual Variance Function: varIdent(·)

The LMMs, presented in Sects. 16.3 and 16.4, were specified with the use of the varPower(·) variance function (see the definition of the matrix $\boldsymbol{\Lambda}_i$ in (16.8)). This may be an overly constrained function, because it assumes that the variances of the visual acuity measurements change as a power function of the measurement time. The choice was motivated by the results obtained in Chaps. 9 and 12, where models, defined with the use of the varPower(·) variance function, fitted the ARMD data better than models with unconstrained variances, specified with the use of the varIdent(·) function (see, e.g., Sect. 12.5.2). However, it is possible that, in the framework of LMMs, a more general variance function might allow obtaining a better fit than the power function.

To verify this hypothesis, we will use the LR test constructed based on models **M16.3** and **M16.6**. Both of the models have the same fixed- and random effects structure, given by (16.10). They differ with respect to $\boldsymbol{\mathcal{R}}_i$ matrix specification. Specifically, in the former model, the matrix $\boldsymbol{\mathcal{R}}_i$ is defined using power function. In contrast, in the latter model the matrix $\boldsymbol{\mathcal{R}}_i$ is defined as follows:

$$\boldsymbol{\mathcal{R}}_i = \sigma_1^2 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sigma_2^2}{\sigma_1^2} & 0 & 0 \\ 0 & 0 & \frac{\sigma_3^2}{\sigma_1^2} & 0 \\ 0 & 0 & 0 & \frac{\sigma_4^2}{\sigma_1^2} \end{pmatrix} \equiv \sigma^2 \begin{pmatrix} \delta_1^2 & 0 & 0 & 0 \\ 0 & \delta_2^2 & 0 & 0 \\ 0 & 0 & \delta_3^2 & 0 \\ 0 & 0 & 0 & \delta_4^2 \end{pmatrix}, \tag{16.18}$$

where $\delta_t \equiv \sigma_t/\sigma_1$ $(t = 1, \ldots, 4)$ is the ratio of SD of the visual acuity measurements at occasion t relative to SD of the measurements at the first occasion, and where $\sigma^2 \equiv \sigma_1^2$. This parameterization corresponds to a *varIdent*-class variance function (Sect. 7.3.1) and is specified in such a way that it allows identifying the variance-function parameters δ_t (Sect. 7.3.2).

To fit model **M16.6**, we update the object `fm16.3` using an appropriate form of the `varIdent()` constructor function in the `weights` argument of the `lme()` function. The suitable syntax and results of fitting of the model are displayed in Panel R16.15a. Additional results are provided in Table 16.3. Panel R16.15b also includes the result of the LR test obtained with the use of the `anova()` function,

Table 16.3 *ARMD Trial*: REML-based estimates[a] for linear mixed-effects models with random intercepts and slopes

	Parameter	fm16.6	fm16.7
Model label		**M16.6**	**M16.7**
Log-REML value		−3204.05[b]	−3218.57
Fixed effects			
Intercept	β_0	5.10(2.18)	5.35(2.33)
Visual acuity at t=0	β_1	0.90(0.04)	0.90(0.04)
Time (in weeks)	β_2	−0.21(0.03)	−0.22(0.03)
Trt(Actv *vs.* Plcb)	β_3	−2.18(1.12)	−2.31(1.21)
Tm × Treat(Actv)	β_4	−0.06(0.05)	−0.06(0.05)
reStruct(`subject`)			
SD(b_{i0})	$\sqrt{d_{11}}$		7.35(6.41,8.43)
SD(b_{i1})	$\sqrt{d_{22}}$		0.28(0.24,0.33)
Scale	σ_1		6.68(6.25,7.14)

[a]Approximate SE for fixed effects and 95% CI for covariance parameters are included in parentheses

[b]Likelihood optimization did not converge

which is based on the likelihoods of models **M16.6** and **M16.3**. Note that the latter (null) model is nested in the former. The outcome of the test is statistically significant at the 5% significance level and suggests that the use of the more general varIdent(·) variance function to define matrix $\mathcal{R}_i$, as in (16.18), gives a better fit than the use of the varPower(·) function.

We need to be careful before accepting this conclusion, though. A closer inspection of the results displayed in Panel R16.15 reveals that the estimated value of parameter δ_4 is extremely small and substantially differs from the estimated values of δ_2 and δ_3. This is surprising, because all previous analyses indicated that the variance of the last visual acuity measurement (at week 52) was the largest.

A signal of the problems with the estimation of model **M16.6** can be also obtained by, e.g., attempting to compute confidence intervals for the variance-covariance parameters. In particular, issuing the command

```
> intervals(fm16.6, which = "var-cov")
```

results in an error message indicating problems with estimating the variance-covariance matrix for the estimates of the parameters.

Finally, the problem with convergence of the estimation algorithm for model **M16.6** is also clearly reflected in the normal Q-Q plot of the conditional Pearson residuals, shown in Fig. 16.10 and obtained by issuing the command

```
> qqnorm(fm16.6, ~resid(.)|time.f)          # Fig. 16.10
```

Note that the residuals for week 52 are all equal to 0.

To investigate the source of the problem, we present, in Fig. 16.11, plots of the cross-sections of the restricted-likelihood surface for δ_2, δ_3, δ_4, and σ. For brevity, we do not show the R code used to create the figure. Each plot is obtained by

R16.15 *ARMD Trial*: Fitting model **M16.6** and testing its variance function using a REML-based likelihood-ratio test. The model-fit object `fm16.3` was created in Panel R16.10

(a) *Fitting of model* **M16.6**

```
> (fm16.6 <-                                        # M16.6 ← M16.3
+     update(fm16.3, weights = varIdent(form = ~1 | time.f)))
  Linear mixed-effects model fit by REML
    Data: armd
    Log-restricted-likelihood: -3204
    Fixed: visual ~ visual0 + time + treat.f + time:treat.f
         (Intercept)            visual0               time
             5.10354            0.90120           -0.21041
       treat.fActive time:treat.fActive
            -2.18434           -0.05931

  Random effects:
   Formula: ~1 + time | subject
   Structure: General positive-definite, Log-Cholesky parametrization
              StdDev  Corr
  (Intercept) 7.34621 (Intr)
  time        0.31104 -0.132
  Residual    4.62311

  Variance function:
   Structure: Different standard deviations per stratum
   Formula: ~1 | time.f
   Parameter estimates:
        4wks      12wks      24wks      52wks
  1.00000000 1.62525293 1.74357631 0.00051508
  Number of Observations: 867
  Number of Groups: 234
```

(b) *REML-based LR test for the variance function*

```
> anova(fm16.3, fm16.6)    # varPower (M16.3) ⊂ varIdent (M16.6)
         Model df    AIC    BIC  logLik   Test L.Ratio p-value
  fm16.3     1 10 6450.6 6498.2 -3215.3
  fm16.6     2 12 6432.1 6489.2 -3204.0 1 vs 2  22.499  <.0001
```

fixing the other parameters at the reported REML estimates. The panel for δ_4, the ratio of the residual SD of the visual acuity measurements at 52 weeks relative to week 4, shows an approximately flat horizontal line close to zero. More precisely, the line shows that the difference between the log-restricted-likelihood for the values of δ_4 within the interval presented in the plot and the reported log-REML value

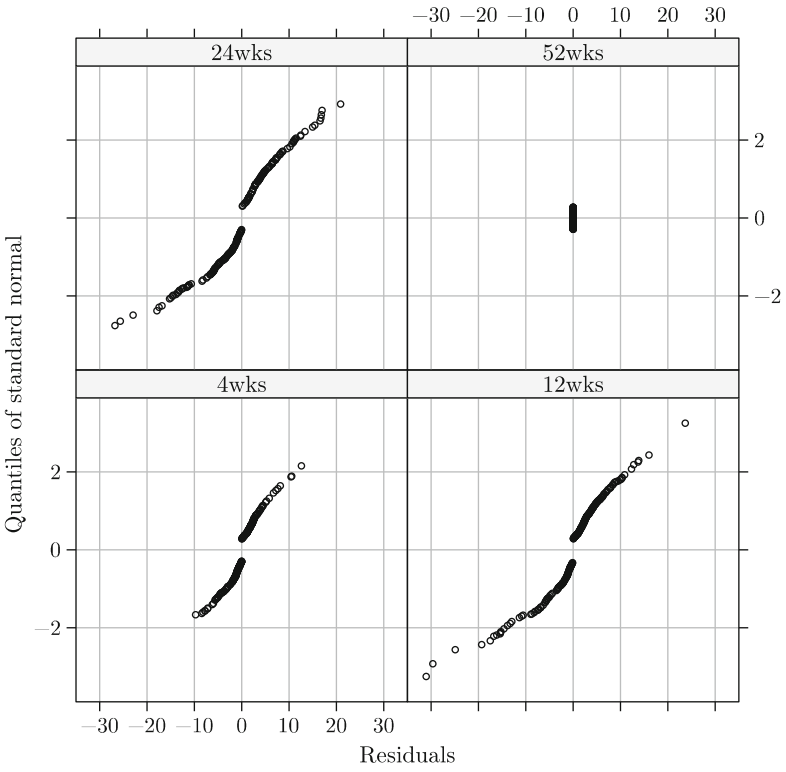

Fig. 16.10 *ARMD Trial*: The normal Q-Q plot of the conditional Pearson residuals for model **M16.6**. Panel for 52 weeks indicates the problem with model fit

of −3204.05 ranges between 2.4×10^{-7} and -4.0×10^{-7}. This indicates that, if we assume model **M16.6**, the data contain very little information about this particular parameter, because the log-restricted-likelihood function surface is virtually flat in the corresponding direction of the parameter space. Moreover, the plot for δ_4, unlike the other plots shown in Fig. 16.11, does not suggest any maximum of the likelihood function within the presented interval of δ_4 values. This means that the REML estimate, reported by the `lme()` function in Panel R16.14a, is not an optimum value.

Given the close similarity of the structure of models **M16.6** and **M16.3**, a question is: Why were there no apparent problems with fitting of the latter model? Although the models are similar, they differ with respect to the form of the marginal variance-covariance structure of visual acuity measurements. The form of the covariance of the measurements obtained for the subject i at different times, implied by model **M16.6**, is the same as the one resulting from model **M16.3** and is given by (16.14). However, the variance for a measurement obtained at the time t, implied by model **M16.6**, is

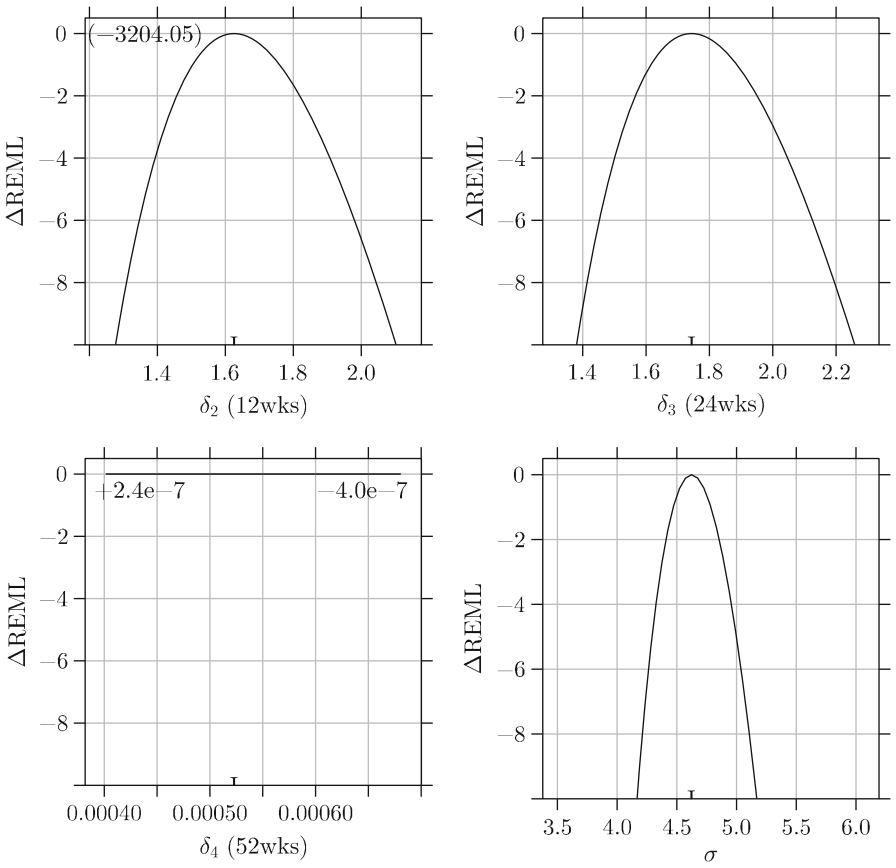

Fig. 16.11 *ARMD Trial*: Differences (ΔREML) between the values of the log-restricted-likelihood for model **M16.6** at the reported REML estimate and at different values of parameters δ_2, δ_3, δ_4, and σ. The value of 0 at the vertical axis corresponds to the reported log-REML value of −3204.05. Flat line in panel for δ_4 indicates problem with model fit

$$\text{Var}(y_{it}) = d_{11} + 2d_{12}\text{TIME}_{it} + d_{22}\text{TIME}_{it}^2 + \sigma^2\delta_t^2. \tag{16.19}$$

Equations (16.14) and (16.19) define the ten unique elements of the marginal variance-covariance matrix $\mathcal{V}_i$ for model **M16.6** as linear functions of seven parameters: d_{11}, d_{12}, d_{22}, σ^2, δ_2, δ_3, and δ_4. Given that the number of parameters is close to the number of equations, collinearity among the parameters may result, with consequences in the form of convergence problems of the estimation algorithm.

On the other hand, the right-hand side of (16.15) for model **M16.3** has fewer parameters and involves a power function of time, which is nonlinear in terms of the parameter δ. Hence, in this case, the collinearity is less likely to appear.

Thus, as compared to model **M16.6**, model **M16.3** imposes an additional restriction on the form of the marginal variance-covariance structure. The restriction limits the parameter space, in which it is possible to find an optimum solution, because the data become more informative.

It follows that, to use the Ident(·) function, some additional restrictions on the form of model **M16.6** would need to be introduced. However, we will not pursue this direction further but rather leave it as an exercise to the reader.

16.6 Testing Hypotheses About Random Effects

As mentioned in Sect. 13.7.2, formal tests of hypotheses about the variance-covariance structure can be performed using the LR test based on the restricted likelihood function. An important issue is the null distribution of the test statistics. In particular, when the values of the variance-covariance parameters, compatible with the null hypothesis, lie in the interior of the parameter space, the null distribution is a χ^2 distribution with the number of degrees of freedom equal to the difference in the number of (independent) variance-covariance parameters between the null and alternative models (Sect. 13.7.2). Examples of such tests were shown in Sects. 16.4.2 (Panel R16.11) and 16.5 (Panel R16.15).

However, when the values of the variance-covariance parameters, compatible with the null hypothesis, lie on the boundary of the parameter space, the exact form of the null distribution is difficult to obtain. As it was mentioned in Sect. 13.7.2, in certain cases (see, e.g., Verbeke and Molenberghs 2000, Sect. 6.3.4), the distribution is given by a mixture of several χ^2 distributions. Note that this result has been obtained by assuming that the residual errors are independent and homoscedastic. In other cases, the only practical alternative is to simulate the null distribution. In R this can be done using the `simulate()` function from the **nlme** package or using the function `exactRLRT()` from the package **RLRsim**.

The use of the functions was briefly reviewed in Sect. 14.7. It was noted there that both functions only allow for independent, homoscedastic residual errors. Moreover, the function `exactRLRT()` accommodates only independent random effects, while the function `simulate()` is not defined for model-fit objects of class *gls*.

These limitations preclude us from testing, e.g., whether inclusion of random intercepts is improving the fit of model **M16.2**, as compared to a model without random effects, but with residual variances expressed with the help of the Power(·) variance function. For the same reason, we cannot test the statistical significance of extending model **M16.2** by the inclusion of random slopes, which leads to model **M16.3**.

In these cases, the plausibility of the modifications of the random effects structure needs to be assessed using, e.g., residual diagnostics and/or by applying the information criteria (Sect. 13.7.2). Panel R16.16 presents the values of the AIC for models **M16.1**–**M16.5**. It can be seen that, e.g., the AIC for model **M16.2**,

R16.16 *ARMD Trial*: The values of Akaike's Information Criterion for models **M16.1**–**M16.5**

```
> AIC(fm16.1, fm16.2,                          # M16.1, M16.2
+     fm16.3, fm16.4)                          # M16.3, M16.4
         df    AIC
  fm16.1  7 6592.0
  fm16.2  8 6537.1
  fm16.3 10 6450.6
  fm16.4  9 6449.8
> fm16.4ml <- update(fm16.4, method = "ML")
> fm16.5ml <- update(fm16.5, method = "ML")
> anova(fm16.4ml, fm16.5ml)                    # M16.4 ⊂ M16.5
           Model df    AIC    BIC  logLik   Test L.Ratio p-value
  fm16.4ml     1  9 6438.0 6480.9 -3210.0
  fm16.5ml     2  8 6437.4 6475.5 -3210.7 1 vs 2  1.3972  0.2372
```

i.e., 6,537.1, is much larger than the value of 6,450.6 for model **M16.3**. This points to a better fit of the latter model. Also, as suggested by, e.g., Fig. 16.9, the predicted values obtained for model **M16.3** follow more closely the observed ones, as compared to model **M16.2** (see Fig. 16.5).

Note that the lowest value of the AIC is obtained for model **M16.5**, suggesting that the model provides the best overall fit to the data. This reflects the choices we made with respect to the random-effects structure in the process of arriving at the model.

In the remainder of this section, we illustrate the use of the analytic results and of the R simulation functions for testing hypotheses about the random effects structure with parameter values at the boundary of the parameter space. Toward this end, we consider several models for the ARMD data which assume homoscedasticity of the residual errors.

16.6.1 Test for Random Intercepts

Let us first consider model **M16.1** containing random intercept. To test whether subject-specific random intercepts are needed, we might use a REML-based LR test based on the alternative model **M16.1** and a null model that assumes homoscedastic residual errors and no random effects.

In Panel R16.17, we conduct the REML-based LRT by referring the LR-test statistic to a null distribution obtained using a mixture of χ^2 distributions or a simulation technique.

In particular, for the first approach, presented in Panel R16.17a, we create the object `vis.gls1a`, which represents the fit of the null model. The model does not

R16.17 *ARMD Trial*: The REML-based likelihood-ratio test for no random intercepts in model **M16.1**. The formula-object `lm2.form` and the model-fit object `fm16.1` were created in Panel R16.1

(a) *Using* $0.5\chi^2_0 + 0.5\chi^2_1$ *as the null distribution*

```
> vis.gls1a   <-                              # Null model
+    gls(lm2.form, data = armd)
> (anova.res  <- anova(vis.gls1a, fm16.1))  # Null vs. M16.1
           Model df    AIC    BIC logLik   Test L.Ratio p-value
  vis.gls1a     1  6 6839.9 6868.5  -3414
  fm16.1        2  7 6592.0 6625.3  -3289 1 vs 2  249.97  <.0001
> (anova.res[["p-value"]][2])/2              # 0.5χ²_0 + 0.5χ²_1
  [1] 0
```

(b) *Using the function* `exactRLRT()` *to simulate the null distribution*

```
> library(RLRsim)
> exactRLRT(fm16.1)                          # M16.1 (alternative)
      simulated finite sample distribution of RLRT.  (p-value
      based on 10000 simulated values)

  data:
  RLRT = 249.97, p-value < 2.2e-16
```

include any random intercepts and is defined by the formula `lm2.form`. Thus, it has the same mean structure as the alternative model **M16.1**, which is represented by the object `fm16.1`. Then, we apply the `anova()` to calculate value of the REML-based LR test statistics.

Note that we are testing the null hypothesis that the variance of the random intercept is zero, which is on the boundary of the parameter space. Thus, the p-value reported by `anova()` is computed by referring the value of the LR-test statistic to the incorrect χ^2_1 null distribution. In this case, the appropriate asymptotic distribution is a 50%–50% mixture of the χ^2_0 and χ^2_1 distributions (Sect. 13.7.2). To obtain the correct p-value, we divided the χ^2_1-based p-value, extracted from the object `anova.res` containing the results of the `anova()`-function call, by 2. Clearly, in the current case, the adjusted p-value indicates that the result of the test is statistically significant. It allows us to reject the null hypothesis that the variance of the distribution of random intercepts is equal to 0.

An alternative, shown in Panel R16.17b, is to use the empirical null distribution of the LR test, obtained with the help of the function `exactRLRT()` from the package **RLRsim** (Sect. 14.7). In the panel, we show the result of application of the function to the object `fm16.1`. Because we test a random effect in model **M16.1**, which contains only a single random effect, we use the abbreviated form of the

function call, with `m` as the only argument. The p-value of the REML-based LR test, estimated from 10,000 simulations (the default), clearly indicates that the result of the test is statistically significant. In this case, given the importance of including the random intercepts into the model, which are needed to adjust for the correlation between visual acuity measurements, there is not much difference with the p-value obtained using the asymptotic 50%–50% mixture of the χ^2_0 and χ^2_1 distributions.

To simulate the null distribution of the LRT, we could consider applying the `simulate()` function to objects `vis.gls1` (see Panel R6.3) and `fm16.1`. Unfortunately, the necessary `simulate.gls()` method is not developed for model-fit objects of class *gls*. In the next section, we will illustrate how to use the `simulate()` function to test for the need of random slope.

16.6.2 Test for Random Slopes

For illustrative purposes, we consider a model with uncorrelated subject-specific random intercepts and slopes and independent, homoscedastic residual errors. That is, we consider a model specified by (16.10)–(16.12), with $\mathcal{D}$ given by (16.16), and $\mathcal{R}_i = \sigma^2 \times \boldsymbol{I}_4$. We will refer to this newly defined model as **M16.7**. In this section, we will use the REML-based LR test to test whether random slopes are needed in model **M16.7**. The test involves comparison of two models, namely, **M16.1** (null) and **M16.7** (alternative).

In Panel R16.18, we introduce three approaches to perform the LR test for random slopes.

To begin, in Panel R16.18a, we fit model **M16.7**, which contains random slopes, by modifying model **M16.4**. More specifically, we assume a constant residual variance. The resulting model is stored in the model-fit object `fm16.7`. The results of fitting of the model are provided in Table 16.3.

In the first approach, shown in Panel R16.18b, we perform the REML-based LR test and explore the use of a 50%–50% mixture of the χ^2_1 and χ^2_2 distributions as the null distribution (see Verbeke and Molenberghs 2000, Sect. 6.3.4). To compute the corresponding p-value, we extract the LR-test statistic value from the object `an.res`, which contains the results of the `anova()`-function call, and we use it as an argument of the `pchisq()` function, which computes the upper tail probabilities of the χ^2 distributions with 1 and 2 degrees of freedom. Clearly, the adjusted p-value indicates that the result of the test is statistically significant. Thus, the test allows us to reject the null hypothesis that the variance of random slopes is equal to 0.

In Panels R16.18c and R16.18d, we consider simulating the null distribution of the REML-based LR-test statistic.

Toward this end, in Panel R16.18c, we use the `exactRLRT()` function. Note that, as it was mentioned earlier, the function allows only for independent random effects. This is the reason why we illustrate the use of the function for model **M16.7** with a diagonal matrix $\mathcal{D}$. Because we consider a model with two variance components, i.e., random intercepts and random slopes, we need to specify all three arguments

R16.18 *ARMD Trial*: The REML-based likelihood-ratio test for random slopes for model **M16.7**. The model-fit objects `fm16.1` and `fm16.4` were created in Panels R16.1 and R16.10, respectively

(a) *Fitting model* **M16.7**

```
> fm16.7 <-                                   # M16.7 ← M16.4
+    update(fm16.4, weights = NULL,           # Constant resid. variance
+           data = armd)
```

(b) *Using* $0.5\chi^2_1 + 0.5\chi^2_2$ *as the null distribution*

```
> (an.res <-                                  # M16.1 (null)
+     anova(fm16.1, fm16.7))                  # M16.7 (alternative)
         Model df    AIC    BIC  logLik   Test L.Ratio p-value
  fm16.1     1  7 6592.0 6625.3 -3289.0
  fm16.7     2  8 6453.1 6491.2 -3218.6 1 vs 2  140.83  <.0001
> (RLRT <- an.res[["L.Ratio"]][2])            # LR-test statistic
  [1] 140.83
> .5 * pchisq(RLRT, 1, lower.tail = FALSE) + # 0.5χ²₁ + 0.5χ²₂
+   .5 * pchisq(RLRT, 2, lower.tail = FALSE)
  [1] 1.3971e-31
```

(c) *Using the function* `exactRLRT()` *to simulate the null distribution*

```
> mAux <-                                     # Auxiliary model with ...
+  update(fm16.1, random = ~0 + time|subject,    # ... random slopes only.
+           data = armd)
> exactRLRT(m = mAux,                         # Auxiliary model
+           m0 = fm16.1,                      # M16.1 (null)
+           mA = fm16.7)                      # M16.7 (alternative)
      simulated finite sample distribution of RLRT.  (p-value
      based on 10000 simulated values)

  data:
  RLRT = 140.83, p-value < 2.2e-16
```

(d) *Using the function* `simulate()` *to simulate the null distribution*

```
> vis.lme2.sim <-                                 # M16.1 (null)
+    simulate(fm16.1, m2 = fm16.7, nsim = 10000) # M16.7 (alternative)
> plot(vis.lme2.sim, df = c(1, 2),                # Fig. 16.12
+      abline = c(0,1, lty=2))
```

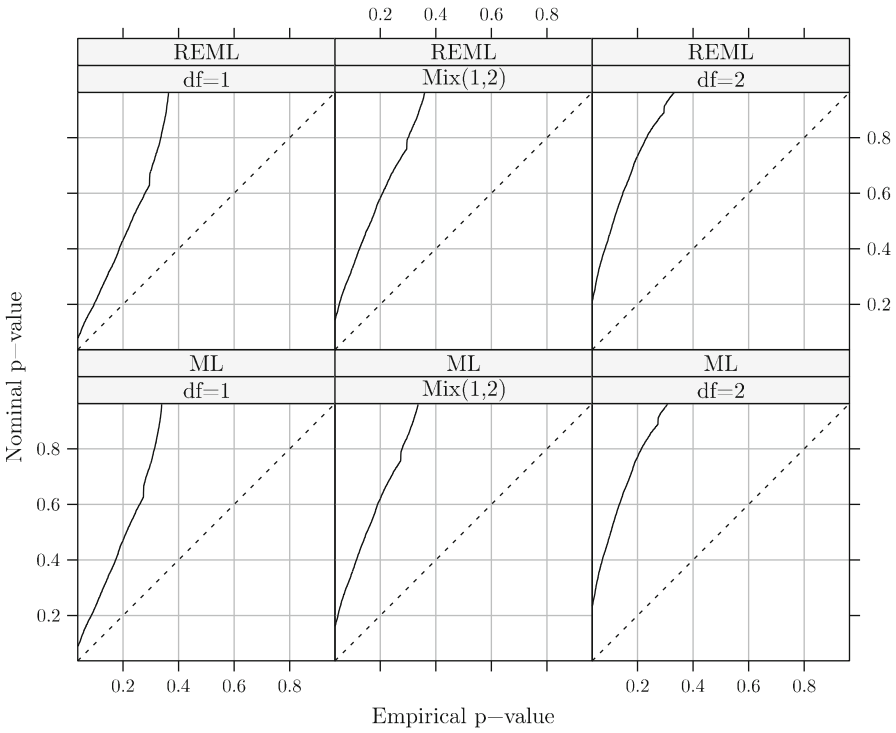

Fig. 16.12 *ARMD Trial*: Empirical and nominal *p*-values for testing the need of random slopes in model **M16.7**

`m`, `m0`, and `mA` of the function `exactRLRT()` (Sect. 14.7). The required form of the function call is shown in Panel R16.18b. The simulated *p*-value is essentially equal to 0, indicating that null hypothesis can be rejected.

Finally, in Panel R16.18d, the function `simulate()` is applied to obtain a plot of empirical and nominal *p*-values (Sect. 14.7). The former are generated by simulating the values of the REML-based LR-test statistic. The plot, in turn, can be used to choose the appropriate null distribution for the calculation of the *p*-value corresponding to the observed value of the test statistic.

More specifically, the function `simulate()` is applied to the objects `fm16.1` and `fm16.7`, with the former specified as the null model and the latter indicated, with the help of the argument `m2`, as the alternative model. The number of the simulated test-statistic values is set, with the help of the `nsim` argument, at 10,000.

The `plot()` statement creates a plot of the empirical and nominal *p*-values of the LR-test statistic. The nominal *p*-values are computed using three distributions: χ^2_1, χ^2_2, and a 50%–50% mixture of χ^2_1 and χ^2_2. The required degrees of freedom are passed to the `plot()` function using the argument `df` in the form of a numeric vector (Sect. 14.7). To include in the plot, e.g., a 65–35% mixture, the argument `weights=c(0.65,0.35)` should explicitly be used.

The resulting plot is shown in Fig. 16.12. Note that two rows of three panels are displayed: one row for the REML and one for the ML estimation. As was mentioned in Sect. 14.7, by default, the function `simulate.lme()` uses both forms of the LR test.

The plot shows that the nominal p-values, obtained using χ^2_1, χ^2_2, or a 50%–50% mixture of χ^2_1 and χ^2_2 distributions, are larger than the corresponding simulated values. This implies that the use of any of those distributions would result in a conservative test.

16.7 Analysis Using the Function `lmer()`

In this section, we refit models **M16.1** and **M16.7**, presented in Sects. 16.2.1 and 16.6.2, respectively, using the function `lmer()` from the package **lme4.0**. The choice of the models is dictated by the fact that, at the time of writing of this book, the function allows only for independent, homoscedastic residual errors. Note that the two models do not adequately describe the ARMD data, as can be concluded from the results of the analyses obtained with the help of the `lme()` function. Thus, the results presented in the current section should be treated mainly as the illustration of the use of the `lmer()` function.

16.7.1 Basic Results

In Panel R16.19, we demonstrate how to fit model **M16.1** with the help of the function `lmer()`. The model included random intercepts and assumed that residual variance was constant. It was fitted using the `lme()` function in Panel R16.1, with the fit stored in the object `fm16.1`.

In Panel R16.19a, we present the `lmer()`-function syntax for fitting model **M16.1**. Note the direct specification of the random-effects structure in the `formula` argument (Sect. 15.3.1). Also, it is worth noting that the argument `data` is provided with a data frame, and not with a grouped data object. In fact, in contrast to the `lme()` function, the use of grouped data objects is neither needed nor recommended (Sect. 15.3). The model is fitted using REML, which is the default estimation method.

The results of the fitted model are printed using the generic `print()` function. It is worth noting that the values of the t-test statistics for the fixed effects are provided without any p-values (Sect. 15.5). Methods to calculate p-values will be presented later in this section.

The `corr=FALSE` argument, used in the `print()`-function call, excludes the estimated correlation matrix of the fixed effects from the printout. This is because the names of the fixed effects are long and the printout of the matrix would not be legible. Instead, in Panel R16.19b, the variance-covariance matrix of the fixed

R16.26 *ARMD Trial*: Model **M16.7** fitted using the function `lmer()`

(a) *Fitting the model and extracting basic information*

```
> fm16.2mer  <-                                    # M16.7
+    lmer(visual ~ visual0 + time + treat.f + treat.f:time +
+           (1|subject) + (0 + time|subject),
+         data = armd)
> summ <- summary(fm16.2mer)
> coef(summ)                                       # t-Table
                     Estimate Std. Error t value
  (Intercept)         5.349030   2.332568  2.2932
  visual0             0.898460   0.039317 22.8519
  time               -0.215370   0.032266 -6.6749
  treat.fActive      -2.313752   1.209754 -1.9126
  time:treat.fActive -0.055059   0.047090 -1.1692
> unlist(VarCorr(fm16.2mer))                       # D̂. Short printout
    subject   subject
  54.071157  0.079359
> sigma(fm16.2mer)                                 # σ̂
  [1] 6.6834
```

(b) *Likelihood-ratio test for the* `treat.f:time` *interaction*

```
> fm16.2aux  <-                                    # Model M16.7 with ...
+    update(fm16.2mer, . ~ . - treat.f:time)      #... interaction omitted
> anova(fm16.2aux, fm16.2mer)
  Data: armd
  Models:
  fm16.2aux: visual ~ visual0 + time + treat.f +
  fm16.2aux:  (1 | subject) + (0 + time | subject)
  fm16.2mer: visual ~ visual0 + time + treat.f + treat.f:time +
  fm16.2mer:  (1 | subject) + (0 + time | subject)
            Df  AIC  BIC logLik Chisq Chi Df Pr(>Chisq)
  fm16.2aux  7 6441 6474  -3213
  fm16.2mer  8 6441 6480  -3213  1.38      1       0.24
```

16.8 Chapter Summary

In this chapter, we analyzed the ARMD data by applying LMMs. By using the models, the hierarchical structure of the data was directly addressed, which allowed taking into account the correlation between the visual acuity measurements obtained for the same individual.

Table 16.4 provides information about the models defined in this chapter.

The main tool that was used to fit the models in Sects. 16.2–16.6 was the function `lme()` from the package **nlme**. In Sect. 16.7, we refitted some of the models using the function `lmer()` from the package **lme4.0**. The latter function

Table 16.4 *ARMD Trial*: Summary of the models defined and fitted using the REML estimation, in Chap. 16
(**a**) Models fitted using the function `lme()` from the package **nlme**[a]

Model label	Section	Syntax	R object	Mean	Random effects (matrix $\mathcal{D}$)	Residual variance
M16.1	16.2	R16.1	`fm16.1`	(12.8)	Intercept (16.6)	Constant
M16.2	16.3	R16.5	`fm16.2`	(12.8)	Intercept (16.6)	*varPower*
M16.3	16.4.1	R16.10	`fm16.3`	(12.8)	Intercept, slope correlated (16.13)	*varPower*
M16.4	16.4.2	R16.11	`fm16.4`	(12.8)	Intercept, slope uncorrelated (16.16)	*varPower*
M16.5	16.4.3	R16.13	`fm16.5`	(12.9)	Intercept, slope uncorrelated (16.16)	*varPower*
M16.6	16.5	R16.15	`fm16.6`[b]	(12.8)	Intercept, slope correlated (16.13)	*varIdent*
M16.7	16.6.2	R16.18	`fm16.7`	(12.8)	Intercept, slope uncorrelated (16.16)	Constant
(**b**) Models fitted using the function `lmer()` from the package **lme4.0**						
Model label	Section	Syntax	R object	Mean	Random effects (matrix $\mathcal{D}$)	Residual variance
M16.1	16.2	R16.19	`fm16.1mer`	(12.8)	Intercept (16.6)	Constant
M16.7	16.6.2	R16.26	`fm16.7mer`	(12.8)	Intercept, slope uncorrelated (16.16)	Constant

[a] The mean structure, defined in (12.8), is represented by the formula `visual ~ visual0 + time + treat.f + time:treat.f`
[b] Optimization of the likelihood for **M16.6** did not converge

R16.27 *ARMD Trial*: The REML-based likelihood-ratio test for no random slopes in model **M16.7**. Model-fit objects `fm16.1mer` and `fm16.2mer` were created in Panels R16.19 and R16.26, respectively

(a) *Using* $0.5\chi^2_1 + 0.5\chi^2_2$ *as the null distribution*

```
> RML0 <- logLik(fm16.1mer)            # log-REML, M16.1 (null)
> RMLa <- logLik(fm16.2mer)            # log-REML, M16.7 (alternative)
> (RLRTstat <- -2 * as.numeric(RML0 - RMLa))
  [1] 140.83
> .5 * pchisq(RLRTstat, 1, lower.tail = FALSE) + # p-value
+   .5 * pchisq(RLRTstat, 2, lower.tail = FALSE)
  [1] 1.3971e-31
```

(b) *Using the function* `exactRLRT()` *to simulate the null distribution*

```
> require(RLRsim)
> mAux  <- lmer(visual ~               # Auxiliary model with ...
+            visual0 + time + treat.f + treat.f:time +
+            (0 + time| subject),      # ... random slopes only.
+          data = armd)
> exactRLRT(m = mAux,                  # Auxiliary model
+           m0= fm16.1mer,             # M16.1 (null)
+           mA= fm16.2mer)             # M16.7 (alternative)
      simulated finite sample distribution of RLRT.  (p-value
      based on 10000 simulated values)

  data:
  RLRT = 140.83, p-value < 2.2e-16
```

is especially suited for, e.g., LMMs with crossed random effects, but it can only deal with conditional-independence models with homoscedastic residual errors. In this respect, it offers a more limited choice of models than `lme()`. For this reason, in our presentation, we primarily focused on the use of `lme()`.

In the process of arriving at the form of the final model **M16.5**, we fixed the mean structure as in (16.1) and built a series of models (see Table 16.4) with various random structures: model **M16.1** with random intercepts and homoscedastic residual variances (Sect. 16.2); model **M16.2** with random intercepts and residual variances described by a variance function defined as a power of the measurement time (Sect. 16.3); model **M16.3** with correlated random intercepts and random slopes and the power-of-time residual variances (Sect. 16.4.1); and model **M16.4** with independent random intercepts and random slopes and the power-of-time

residual variances (Sect. 16.4.2). The last model gave a satisfactory fit to the data and allowed us to simplify the mean structure by adopting a constant treatment effect, as reflected in model **M16.5** in Sect. 16.4.3.

The presented approach was adopted mainly for illustrative purposes. In practice, we should start building the model using the most general fixed- and random-effects structures. Then, we might consider simplifying the random-effects structure while checking the fit of the simplified models using the LR test or information criteria (Sect. 13.7.2). When a more parsimonious structure with a satisfactory fit to the data has been found, we could consider in turn simplifying the mean structure. After arriving at a final model, we should check its fit by residual diagnostics (Sect. 13.6.2).

Thus, in the case of the visual acuity data, we might begin, for instance, from model **M16.3**, but with time included in the mean structure as a factor, and try to simplify the model by removing the random effects of time. We would most likely find that the simplification was worsening the fit of the model. Thus, we might settle for a model with random intercepts and time effects, and consider simplifying the mean structure by assuming, e.g., a continuous time effect and a constant treatment effect. This step would most likely lead us to model **M16.5** as the final model.

In Sect. 16.5, we additionally considered model **M16.6** with correlated random intercepts and random slopes and time-specific residual variances. As the model assumes a slightly more general residual-variance structure than model **M16.3**, it could offer a better fit. We discovered, however, that model **M16.6** could not be fitted to the data by the function `lme()`. From a practical point of view of using the function to fit LMMs, this example illustrates that the results of a model fit need always to be carefully checked for symptoms of nonconvergence. This is because the function may fail to report any apparent error messages that would indicate problems with convergence of the estimation algorithm.

In Sect. 16.6, we discussed the issue of testing hypotheses about the random-effects structure. This is a difficult issue, due to the problems with obtaining the null distribution of the LR-test statistic in situations when the null hypothesis involves values of parameters at the boundary of the parameter space. Exact analytical results are available only for a limited set of special cases. In practice, a simulation approach is often used. However, the R functions available for this purpose are also limited in their scope. For instance, they apply to models with homoscedastic residual errors. For this reason, their application to the models considered for the ARMD data, which specified the residual variances using the varPower$(\cdot)$ variance function, was not possible. In such a case, the choice of the random effects structure may need to be based on an informal comparison of the fit of the models based on residual diagnostics and/or the information criteria. To nevertheless illustrate the tools for testing hypotheses about the random-effects structure, we considered model **M16.7** with uncorrelated random intercepts and slopes and homoscedastic, independent residual errors.

As mentioned earlier, in Sect. 16.7, we refitted models **M16.1** and **M16.7** using the function `lmer()` from the package **lme4.0**. This allowed us to illustrate the differences in the use of the function, as compared to `lme()`. Important differences

include, e.g., the form of the model-defining formula and the methods to extract components from a model-fit object. Also, `lmer()` does not report p-values, which means that the user needs to know additional tools that allow to evaluate results of significance tests. We have presented such tools in Sects. 16.7.2–16.7.4.

In the next chapter, we further illustrate the use of the function `lme()` for fitting LMMs by applying the models in the analysis of the PRT study data.