Use R!

Patrick Mair

Modern Psychometrics with R

Springer

Use R!

Series Editors

Robert Gentleman Kurt Hornik Giovanni Parmigiani

More information about this series at http://www.springer.com/series/6991

Patrick Mair

Modern Psychometrics with R

Springer

Patrick Mair
Department of Psychology
Harvard University
Cambridge, MA, USA

ISSN 2197-5736 ISSN 2197-5744 (electronic)
Use R!
ISBN 978-3-319-93175-3 ISBN 978-3-319-93177-7 (eBook)
https://doi.org/10.1007/978-3-319-93177-7

Library of Congress Control Number: 2018946544

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

We are living in exciting times when it comes to statistical applications in psychology. The way statistics is used (and maybe even perceived) in psychology has drastically changed over the last few years – computationally as well as methodologically. From a computational perspective, up to some years ago many psychologists considered R as an analysis tool used by the nerdiest quants only. In recent years, R has taken the field of psychology by storm, to the point that it can now safely be considered the lingua franca for statistical data analysis in psychology. R is open source, makes statistical analyses reproducible, and comes with a vast amount of statistical methodology implemented in a myriad of packages. Speaking of packages, in the past decade a flourishing psychometric developer community has evolved; their work has made R a powerful engine for performing all sorts of psychometric analyses.

From a methodological perspective, the latent continuum of statistical methods in modern psychology ranges from old school classical test theory approaches to funky statistical and machine learning techniques. This book is an attempt to pay tribute to this expansive methodology spectrum by broadening the standard "psychological measurement" definition of psychometrics toward the statistical needs of a modern psychologist. An important aspect of this book is that in several places uncommon, maybe even exotic techniques are presented. From my own experience consulting and teaching at a psychology department, these techniques turned out to be very useful to the students and faculty when analyzing their data.

The broad methodological range covered in this book comes at a price: It is not possible to provide in-depth narrative details and methodological elaborations on various methods. Indeed, an entire book could probably be written about each single chapter. At the end of the day, the goal of this book is to give the reader a starting point when analyzing data using a particular method (including fairly advanced versions for some of them), and to hopefully motivate him/her to delve deeper into a method based on the additional literature provided in the respective sections, if needed. In the spirit of Springer's useR! series, each method is presented according to the following triplet: (1) when to apply it and what is it doing, (2) how to compute it in R, and (3) how to present, visualize, and interpret the results.

Throughout the book an effort has been made to use modern, sparkling, real-life psychological datasets instead of standard textbook data. This implies that results do not always turn out perfectly (e.g., assumptions are not always perfectly fulfilled, models may not always fit perfectly, etc.), but it should make it easier for the reader to adapt these techniques to his/her own datasets. All datasets are included in the **MPsychoR** package available on CRAN. The R code for reproducing the entire book is available from the book website.

The target audience of this manuscript is primarily graduate students and postdocs in psychology, thus the language is kept fairly casual. It is assumed that the reader is familiar with the R syntax and has a solid understanding of basic statistical concepts and linear models and preferably also generalized linear models and mixed-effects models for some sections. In order to be able to follow some of the chapters, especially those on multivariate exploratory techniques, basic matrix algebra skills are needed. In addition, some sections cover Bayesian modeling approaches. For these parts, the reader needs to be familiar with basic Bayesian principles. Note that no previous exposure to psychometric techniques is required. However, if needed, the reader may consult more narrative texts on various topics to gain a deeper understanding, since explanations in this book are kept fairly crisp.

The book is organized as follows. It starts with the simplest and probably oldest psychometric model formulation: the true score model within the context of classical test theory, with special emphasis placed on reliability. The simple reliability concept is then extended to a more general framework called generalizability theory. Chapter 2 deals with factor analytic techniques, exploratory as well as confirmatory. The sections that cover confirmatory factor analysis (CFA) focus on some more advanced techniques such as CFA with covariates, longitudinal, multilevel, and multigroup CFA. As a special exploratory flavor, Bayesian factor analysis is introduced as well. The CFA sections lay the groundwork for Chap. 4 on structural equation models (SEM). SEM is introduced from a general path modeling perspective including combined moderator-mediator models. Modern applications such as latent growth models are described as well.

Chapter 4 covers what is often called modern test theory. The first four sections cover classical item response theory (IRT) methods for dichotomous and polytomous items. The remaining sections deal with some more advanced flavors such as modern DIF (differential item functioning) techniques, multidimensional IRT, longitudinal IRT, and Bayesian IRT. Chapter 5, the last parametric chapter for a while, tackles a special type of input data: preference data such as paired comparisons, rankings, and ratings.

Then, a sequence of multivariate exploratory methods chapters begins. Chapter 6 is all about principal component analysis (PCA), including extensions such as three-way PCA and independent component analysis (ICA) with application on EEG (electroencephalography) data. What follows is a chapter on correspondence analysis which, in the subsequent section, is extended to a more general framework called Gifi. An important incarnation of Gifi is the possibility to fit a PCA on ordinal data or, more generally, mixed input scale levels. Chapter 9 covers in detail multidimensional scaling (MDS). Apart from standard exploratory MDS, various

sections illustrate extensions such as confirmatory MDS, unfolding, individual differences scaling, and Procrustes. Chapter 10 explores techniques for advanced visualization techniques by means of biplots, which can be applied to each of the preceding exploratory methods.

The remaining chapters of the book cover various special techniques useful for modern psychological applications. Chapter 11 introduces correlation networks, a hot topic in clinical psychology at the moment. These simple network approaches are extended to various advanced techniques involving latent networks and Bayesian networks, the latter based on the concept of directed acyclic graphs.

Chapter 12 is all about sophisticated parametric clustering techniques. By way of the concept of mixture distributions, various extensions of the well-known psychometric technique called latent class analysis are presented. This mixture distribution idea is then integrated into a regression framework, which leads to mixture regression and Dirichlet process regression. Since text data are of increasing importance in psychology, a special clustering approach called topic models is included as well.

Many modern psychological experiments involve longitudinal or time series measurements on a single participant. Chapter 13 offers some possibilities for how to model such data. It starts with Hidden Markov models, yet another approach from the parametric clustering family. The subsequent section presents time series analysis (ARIMA models), where the main goal is to forecast future observations. Finally, functional data analysis is introduced, a modeling framework for data settings where each participant is represented by a function.

The final chapter is somewhat special since it is completely driven by a particular type of input data: functional magnetic resonance imaging (fMRI) data. The aim of this chapter is to give the reader more insight into what is going on in standard fMRI modeling techniques. He/she will, for instance, learn how to establish statistical parametric maps and how to deal with the heavy duty multiple comparison problem. In addition, in this chapter various techniques from previous chapters such as ICA and latent networks are applied to fMRI data.

That should do it.

I would like to thank a few people who had a vital impact on this book. First of all, a resounding “thank you” goes out to all the folks at the Harvard Psychology Department for sharing datasets and for allowing me to make them publicly available, such that the entire book can be reproduced. This includes Mahzarin Banaji, Robin Bergh, Tessa Charlesworth, Peter Franz, Benedek Kurdi, Richard McNally, Ken Nakayama, Erik Nook, Hrag Pailian, Maryam Pashkam, Leah Somerville, Rachel Vaughn-Coaxum, and John Weisz. Researchers from other institutions who generously shared their datasets are Alexandra Grand, Pascal Haegeli, Christine Hooker, Ingrid Koller, Kimberley Lakes, Daniel Levitin, Matea Paškvan, Horst Treiblmaier, and Jeremy Wilmer. I also forced several experts to volunteer to read various chapters or sections, whom I would like to thank explicitly: Ingwer Borg, Regina Dittrich, Pieter Kroonenberg, Thomas Rusch, Jim Sidanius, Mark Thornton, Remi Piatek, and Achim Zeileis. Deep gratitude goes out to my psychometric and statistical mentors: Peter Bentler, Jan de Leeuw, Anton Formann

(deceased on July 12, 2010), Wilfried Grossmann, Reinhold Hatzinger (deceased on July 17, 2012), Kurt Hornik, and Alexander von Eye.

In addition, I would like to thank all the graduate students who attended my advanced classes on psychometrics, statistical learning, and Bayesian data analysis. They have been a great source of inspiration and pushed me to get the whole thing done. Also, thanks to all the psychometric R package developers who invested countless night shifts to make R such a powerful engine for psychometric modeling.

Last but not least I would like to thank my family for their support, especially my wife Haley for keeping me balanced during the writing process and for proofreading.

Cambridge, MA, USA Patrick Mair
January, 2018

Contents

Chapter 1
Classical Test Theory

1.1 Classical True Score Model

It is always good to start a book with a formula right away:

$$X = T + E. \tag{1.1}$$

X is the *observed score* in a test, T is the *true score* (unknown), and E is the *error* (unknown). This equation is called the *true score model* and is at the core of *classical test theory* (CTT). Before we dissect this equation with two unknowns and perform various computations, let us elaborate on some general measurement issues related to classical test theory.

Psychological measurement can be a difficult task. We aim to measure not directly observable variables like cognitive abilities, personality traits, motivation, quality of life, diseases in psychopathology, etc. Measuring such *latent* constructs is much more challenging than determining body height or weight (for which we have a measuring tape and a scale as measurement instruments). Obviously we cannot simply ask: "How depressed are you?" or "How intelligent are you?". We need a measurement instrument such as a test or questionnaire to assess a participant's location on the underlying latent variable. There are at least two problems associated with this task:

- Precision of a test: No test measures a latent variable perfectly; we have to deal with some degree of *measurement error*.
- Instability over time: If we were to perform repeated measurements on a single person over time (under the same external conditions), we cannot expect that the results will be identical. This repeated measurement concept (without necessarily being able to obtain multiple measurements per participant) plays an important role when assessing the quality or consistency of a test.

P. Mair, *Modern Psychometrics with R*, Use R!,
https://doi.org/10.1007/978-3-319-93177-7_1

CTT is an old approach that attempts to formalize a statistical theory of (psychological) measurement and, eventually, allows us to make statements about the quality of a scale. What Eq. (1.1) basically tells us is that an individual has a true value T on the underlying latent variable which cannot be directly observed. We can only observe X resulting from the test we constructed. In practice, X is often a composite score based on k items: $X = \sum_{i=1}^{k} X_i$. Equation (1.1) implies that X can deviate from T, that is, our measurement X is associated with an error E.

Let us embed Eq. (1.1) into a statistical sampling context. In general, persons are drawn from a *population*, and for each person we have (in theory) an infinite sequence of repeated measurements. Items are drawn from a *universe* of items supposed to measure the underlying latent variable. Depending on the sampling level we consider for both items and persons, Eq. (1.1) can come in different flavors. As usual, we use capital letters for random variables and lower case notation for non-stochastic variables:

- $X = T + E$: randomly drawn persons from a population (repeated measurements), randomly drawn test/items from a universe.
- $X_i = T_i + E_i$: randomly drawn persons from a population (repeated measurements), fixed item i.
- $X_v = t_v + E_v$: fixed person v (repeated measurements), randomly drawn test/items from a universe.
- $X_{vi} = t_{vi} + E_{vi}$: fixed person v (repeated measurements), fixed item i.
- $x_{vil} = t_{vi} + e_{vil}$: fixed person v (fixed measurement l), fixed item i.

The first formulation is the most general population/universe model, whereas the second equation "zooms in" on a particular item. If we formulate the true score model for an individual person v (third variant), the true score t_v is not random anymore since it remains constant across multiple measurements. The same applies to the fourth variant at an item level for each individual. The last variant is at a sample level. What we observe in practice is one particular realization of the random variable X_{vi} on measurement occasion l. Thus, x_{vil} represents the observed sample value of an item (e.g., a 0 or 1 score of person v on item i).

Regardless which representation we use, certain properties need to hold, here illustrated using the variant involving X_{vi}. If we were to present the same item i to an individual v many times, the true score is the average of the observed scores. At a population level, this means that the expected value $E(X_{vi}) = t_{vi}$. It follows that the expected value $E(E_{vi}) = 0$, that is, in the long run the error is 0 on average, just as in regression. In addition, again similar to regression, we assume that the error is normally distributed and uncorrelated with the true score.

A crucial point in CTT is the magnitude of the error variance. The smaller the error variance, the more accurately the true score is reflected by our observed scores (across multiple individuals). In a perfect world, all error values (across individuals) are 0. That is, each participant scores exactly his/her true score. This is not realistic, of course. Thus, we have some variance in the errors. The corresponding standard deviation of the errors has its own name: the *standard error of measurement*, denoted by σ_E. In the subsequent section, we will illustrate how to get an estimate for σ_E.

1.2 Reliability

As we have seen so far, CTT is concerned about relationships among X, T, and E. Our ultimate goal is to make statements about the quality of the scale. Quality within the context of CTT means that we are able to replicate the results if the individuals were tested multiple times. As Hoyt and Melby (1999) put it, the concept referring to the tendency toward *consistency* among repeated attempts to measure the same thing is called *reliability*. In other words, reliability represents the accuracy with which a test can measure true scores (Coaley, 2014):

- If the reliability is large, σ_E is small: X has little measurement error and will be close to T.
- If the reliability is small, σ_E is large: X has large measurement errors and will deviate from T.

Formally, reliability is defined as

$$\text{reliability} = \frac{\sigma_T^2}{\sigma_X^2} = \frac{\sigma_T^2}{\sigma_T^2 + \sigma_E^2} = \rho_{XT}^2. \tag{1.2}$$

It is the proportion of variance of T in X which is the same as the squared correlation between X and T (see, e.g., Revelle (2015) for how this formal relationship can be derived). Thus, if the (squared) correlation between the observed scores and the true scores is high, the test has a high reliability. In this case the standard error of measurement is small.

The reliability ρ_{XT}^2 is bounded between [0; 1]. It becomes 1 if $\sigma_T^2 = \sigma_X^2$, that is, our instrument measures T perfectly. It approaches 0 as the error variance σ_E^2 increases: the observed scores are scattered broadly around T. Still, the problem is that we cannot directly compute ρ_{XT}^2 since we do not know σ_T^2.

1.2.1 Cronbach's α

A trick that makes it possible to get an estimate of reliability is to establish a *parallel form* of the original test X (with standard deviation σ_X). Let us denote this parallel form by X' (with standard deviation $\sigma_{X'}$). A parallel test is a second version of the original test with the same true score and the same error variance. Let $\sigma_{XX'}$ denote the covariance between these two tests which can be transformed into a squared correlation (i.e., a reliability measure). Equation (1.2) becomes

$$\rho_{XX'} = \frac{\sigma_{XX'}}{\sigma_X \sigma_{X'}} = \frac{\sigma_T^2}{\sigma_X^2} = \rho_{XT}^2. \tag{1.3}$$

From this equation we can derive an expression for the standard error of measurement:

$$\sigma_E = \sigma_X \sqrt{1 - \rho_{XT}^2}. \tag{1.4}$$

In practice, establishing a good parallel test is very hard to come by. One option to obtain a parallel test is to split the original test in half and treat the two halves as parallel tests (*split-half reliability*). Another option, widely used in practice, was proposed by Cronbach (1951). The basic idea is that the total score is made up of the k individual item scores ($X = \sum_{i=1}^{k} X_i$). Thus, each item is considered as a single test, and we have, at least conceptually, constructed k parallel tests. This allows us to compute a lower bound for the reliability, which is known as *Cronbach's* α:

$$\text{reliability} \geq \alpha = \frac{k}{k-1}\left(1 - \frac{\sum_{i=1}^{k} \sigma_{X_i}^2}{\sigma_X^2}\right) \tag{1.5}$$

It is a lower bound since we cannot assume that the composites are strictly parallel. The amazing news at this point is—and we cannot take this for granted in CTT—that we can actually compute something.

Let us illustrate the computation of Cronbach's α by means of a simple example. The dataset we use is based on a larger study presented in Mair et al. (2015) on motivational structures (intrinsic, extrinsic, hybrid) of R package authors. The authors were interested in exploring the question: What motivates package developers to contribute to the R environment? Here we focus on hybrid motivation only and select 19 dichotomous items associated with this latent variable. After removing missing values, 777 package authors are left in the sample.

```
library("MPsychoR")
library("psych")
data("Rmotivation")
ind <- grep("hyb", colnames(Rmotivation))
HybMotivation <- na.omit(Rmotivation[,ind]) ## item selection
k <- ncol(HybMotivation)                    ## number of items
```

Computing Cronbach's α by hand is fairly simple. The first step is to calculate the variance-covariance (VC) matrix of the items (i.e., a matrix with variances in the main diagonal and covariances in the off-diagonals).

```
vcmat <- cov(HybMotivation)
```

From this matrix we extract the individual item variances $\sigma^2_{X_i}$ from the main diagonal and take the sum, as suggested by Eq. (1.5). The sum of the main diagonal elements of a matrix is called *trace*:

```
sigma2_Xi <- tr(vcmat)
```

The total variance consists of the sum of the variances and the sum of the covariances on both sides of the diagonal.[1] We can simply say:

```
sigma2_X <- sum(vcmat)
```

Now we are all set to compute α according to Eq. (1.5):

```
cronalpha <- k/(k-1)*(1-sigma2_Xi/sigma2_X)
round(cronalpha, 2)
## [1] 0.82
```

Using Eq. (1.4), we can compute the standard error of measurement according to $\sigma_E = \sigma_X\sqrt{1-\alpha}$:

```
sqrt(sigma2_X)*sqrt(1-cronalpha)
## [1] 1.710338
```

As a shortcut for Cronbach's α computation, we can use the `alpha` function included in the **psych** package (Revelle, 2017). This functions also computes a confidence interval (CI) for α (including options for bootstrapping), determines changes in α if a particular item would be eliminated, and returns other descriptive item statistics.

```
alpha.hyb <- psych::alpha(HybMotivation)
round(alpha.hyb$total[1], 2)          ## Cronbach's alpha
##  raw_alpha
##       0.82
```

[1] This is based on an extension of the simple variance sum law: $Var(X+Y) = Var(X)+Var(Y)+2Cov(X,Y)$.

In practice, we aim for an α in the area of 0.8–0.9. Values of $\alpha > 0.9$ may reflect a scale burdened by question redundancy (see, e.g., Streiner, 2003) and will generally have a lower correlation with external variables (which is an indication of low *validity*). In such cases it is suggested to do a closer item inspection in order to determine if, in fact, wording redundancy is the problem. Note that in the case of a high α value, we cannot conclude that the scale is unidimensional (i.e., all items measure the same construct), as Ten Berge and Sočan (2004) show. Dimensionality assessment will be discussed in Sect. 4.1.2 within the context of item response models.

Cronbach's α became the biggest rock star of all reliability measures and is still widely applied. However, it has been criticized repeatedly in the literature. For instance, a thorough discussion on the limited usefulness of α is given in Sijtmsa (2009). Therefore, the next section presents some alternative coefficients for reliability assessment.

1.2.2 Other Reliability Coefficients

Sijtmsa (2009) advocates a reliability measure called the *greatest lower bound* (*glb*; Jackson and Agunwamba, 1977). Cronbach's α is a reasonable lower bound for reliability if the items in a test are *essentially* τ*-equivalent*. This implies that each item measures the same latent variable but with possibly different degrees of precision and that the inter-item covariances are constant across item pairs (see Graham, 2006, for details). This is not a very realistic assumption in practice. If violated, α underestimates the reliability, whereas the *glb* provides a better reliability approximation. It holds that $glb \geq \alpha$.

In our hybrid motivation example, we can compute the *glb* using the following function call:

```
glb(HybMotivation)
```

We get a *glb* of 0.89 which, as expected, is larger than Cronbach's α from above.

Other coefficients for reliability assessment are McDonald's ω_t and ω_h (McDonald, 1999; Revelle and Zinbarg, 2009). Their computation is based on factor analysis which we will discuss in Chap. 2. The ω_h coefficient gives the proportion of variance in observed scores accounted for by a single *general* factor (i.e., a single latent variable). Note that the general factor concept is weaker than unidimensionality: it tells us whether our scale has a dominating single factor and, possibly, some additional, "weaker" subfactors. The ω_t does not care too much about the general factor issue. Rather, it represents the proportion of test variance due to all common factors. Details on their computation can be found in Revelle and Zinbarg (2009).

Using our hybrid motivation example we can apply the following **psych** function call to estimate these ω-coefficients:

```
omega(HybMotivation)
```

We get an ω_t of 0.85 which is close to Cronbach's α. Note that in case of unidimensionality $\omega_t = \alpha$. For ω_h we get a value 0.58 which tells us that 58% of the variance in observed scores is due to the general factor and the remaining portion is due to possible subfactors of hybrid motivation. As Revelle (2015) points out, ω_h is particularly relevant when evaluating the importance and reliability of the general factor of a test. Its upper limit is not 1 (as opposed to ω_t).

Other reliability measures are Guttman's λ-coefficients (where $\lambda_3 = \alpha$). They can be computed using the `guttman` function in **psych**. Other helpful R packages within this context are **CTT** (Willse, 2014) and **psychometric** (Fletcher, 2010). The **cocron** package (Diedenhofen, 2016) offers functions to statistically compare two or more α coefficients based on either dependent or independent groups of individuals. The **CMC** package (Cameletti and Caviezel, 2012) calculates and plots the Cronbach-Mesbach curve, a method based on Cronbach's α for checking unidimensionality.

This concludes our introductory elaborations on CTT and reliability. More comprehensive treatments can be found in the following publications. An excellent, nontechnical introduction to this topic is presented in Coaley (2014); more technical beef is given in Crocker and Algina (1986), Algina and Penfield (2009), and Revelle (2015), the latter showing additional options implemented in **psych**.

1.3 Generalizability Theory

The classical true score model in Eq. (1.1) has only one source of error: E. Cronbach (1972) extended the reliability concept by combining the true score model with ANOVA techniques in order to account for multiple sources of measurement errors. This framework is called *generalizability theory* (G-theory). In G-theory slang, these multiple error sources are called *facets*. Examples of facets are items, raters, measurement occasions, etc. Excellent G-theory introductions within a psychological context can be found in Hoyt and Melby (1999) and Lakes and Hoyt (2009). The G-theory bible is Brennan (2001). Shavelson and Webb (1991) provide a more lightweight introduction to the topic.

1.3.1 Reliability and Generalizability

We start our elaborations with a single facet example using the motivation data from above. Note that the following explanations are designed to illustrate basic G-theory concepts based on classical reliability and ANOVA. We will show alternative ways to compute Cronbach's α and introduce the idea of variance components. In practice, G-theory is applied to multiple facets, as shown in Sect. 1.3.2. The true score concept from CTT is replaced by the notion of the *universe score*.

In Sect. 1.2.1, using the hybrid motivation data, we obtained a Cronbach's α of 0.82. This coefficient can be also computed through ANOVA (see Algina and Penfield, 2009, p. 101) with main effects for persons and items. Before fitting the fixed-effects ANOVA, we need to convert the data into a long format.

```
library("reshape2")
Hyb1 <- data.frame(HybMotivation,
                   person = 1:nrow(HybMotivation))
Hyblong <- melt(Hyb1, id.vars = c("person"),
                variable.name = "item")
Hyblong$person <- as.factor(Hyblong$person)
summary(aov(value ~ person + item, data = Hyblong))
##                Df Sum Sq Mean Sq F value Pr(>F)
## person        776  663.0    0.85   5.549 <2e-16 ***
## item           18  573.8   31.88 207.048 <2e-16 ***
## Residuals   13968 2150.5    0.15
```

An approximation of Cronbach's α can be obtained via the person mean squares MS_p and residual mean squares MS_e: $(MS_p - MS_e)/MS_p$.

```
round((0.85-0.15)/0.85, 2)
## [1] 0.82
```

Note such an ANOVA strategy is also pursued when computing the *intraclass correlation coefficient* (ICC; Shrout and Fleiss, 1979), often applied in the medical area to determine the reliability of ratings (e.g., participants being rated by judges). In our hybrid motivation example, the items play the role of raters. The following function from the **psych** package computes a sequence of ICCs[2]:

```
icchyb <- ICC(HybMotivation)
```

[2]Details on different types of ICCs can be found in Shrout and Fleiss (1979).

For our purposes $ICC(3, k) = 0.82$ is the relevant one (average fixed raters). We see that in our simple one-facet example, the ICC is the same as Cronbach's α.

Instead of computing a fixed-effects ANOVA, we can also fit a random-effects ANOVA which gives us variance components. As we will see, variance components play a crucial role in G-theory. They can be derived from the fixed-effects computation involving the mean squares (persons p, items i, residuals e) according to $\sigma_p^2 = (MS_p - MS_e)/n_i$ and $\sigma_i^2 = (MS_i - MS_e)/n_p$, with n_i and n_p as the number of items and persons, respectively. Expressed as standard deviations, they are

```
sqrt((0.85-0.15)/19)
## [1] 0.191943
sqrt((31.88-0.15)/777)
## [1] 0.2020806
```

A more efficient way is to compute them directly through a random-effects ANOVA using the **lme4** package (Bates et al., 2015):

```
library("lme4")
VarCorr(lmer(value ~ (1|person) + (1|item), data = Hyblong))
##  Groups   Name        Std.Dev.
##  person   (Intercept) 0.19200
##  item     (Intercept) 0.20206
##  Residual             0.39238
```

In G-theory we use these variance components to compute the *generalizability coefficient*. Formal expressions are given further below. For the moment it is only important to know that the generalizability coefficient is a reliability coefficient which, in our simple one-facet example, is equivalent to Cronbach's α. We can use the **gtheory** package (Moore, 2016) which performs the **lme4** call from above internally, and, based on these results, it computes the generalizability coefficient.

```
library("gtheory")
gfit <- gstudy(data = Hyblong,
               formula = value ~ (1|person) + (1|item))
dfit <- dstudy(gfit, colname.objects = "person",
               colname.scores = "value", data = Hyblong)
round(dfit$generalizability, 3)
## [1] 0.82
```

We see that once more we obtained Cronbach's α. Now let us move on with multiple facets and explain in more detail what the last code chunk is actually doing.

1.3.2 Multiple Sources of Error

In the computations above, the items were the only facet since persons are typically not considered as facets. G-theory can be seen as an extension of both ICC (which in its classical definition has a single rater facet only) and CTT, involving multiple sources of errors by using random-effects ANOVA techniques.

The dataset we use to illustrate a multi-facet G-theory application is taken from Lakes and Hoyt (2009). The authors used the RCS (response to challenge scale; see Lakes and Hoyt, 2004; Lakes, 2012) in order to assess children's self-regulation in response to a physically challenging situation. The scale consists of three domains: cognitive, affective/motivational, and physical.

Here we focus on the physical domain only. Each of the 194 children in the sample is rated by five raters on three items on his/her self-regulatory ability. The ratings are on a scale from 1 to 7. Let us start with the data preparation:

```
data("Lakes")
phydat <- subset(Lakes, subtest == "physical")
phydat$item <- droplevels(phydat$item)
head(phydat)
##       personID raterID item score  subtest
## 12611        1       7 phy1     5 physical
## 12612        1       1 phy1     5 physical
## 12613        1       3 phy1     5 physical
## 12614        1       8 phy1     4 physical
## 12615        1       5 phy1     6 physical
## 12616        2       3 phy1     5 physical
```

We consider two facets: items and raters. Note that in case of multiple measurement occasions, time could be included as third facet.

The starting point of a generalizability analysis is to conduct a so-called *G-study*. This implies that we fit a random-effects ANOVA model of the form:

$$X_{pir} = \mu + \nu_p + \nu_i + \nu_r + \nu_{pi} + \nu_{pr} + \nu_{ir} + \nu_{pir,e}. \tag{1.6}$$

The indices are p for the persons, i for the items, and r for the raters. The grand population mean is μ; the remaining ν-parameters are the main and interaction effects. Note that the last effect is actually a residual effect involving the three-way interaction and all other sources of error not captured by the specification. The `gstudy` function in the **gtheory** package fits a random-effects ANOVA (via **lme4**)

and extracts the variance components. Based on Eq. (1.6), the total variance can be decomposed as follows (see Shavelson and Webb, 1991):

$$\sigma^2(X_{pir}) = \sigma_p^2 + \sigma_i^2 + \sigma_r^2 + \sigma_{pi}^2 + \sigma_{pr}^2 + \sigma_{ir}^2 + \sigma_{pir,e}^2. \tag{1.7}$$

Here, σ_p^2 indicates how much the children differ in their self-regulation; σ_i^2 shows how homogeneous the items are; σ_r^2 tells us whether some raters are more lenient than others in their scoring; σ_{pi}^2 denotes relative differences in relative self-regulation across items, averaged over raters; σ_{pr}^2 denotes the relative differences self-regulation across raters, averaged over raters; σ_{ir}^2 shows the inconsistency of raters' average ratings of children from one item to the next; $\sigma_{pir,e}^2$ captures the three-way interaction plus error (i.e., the systematic variation due to sources not controlled for in this design and unsystematic variation due to sources that cannot be controlled). We get the following result:

```
formula <- score ~ (1|personID) + (1|raterID) + (1|item) +
  (1|personID:raterID) + (1|personID:item) + (1|raterID:item)
gfit <- gstudy(formula = formula, data = phydat)
gfit
## $components
##             source        var percent n
## 1 personID:raterID 0.12729986    10.7 1
## 2    personID:item 0.15122340    12.7 1
## 3         personID 0.45876447    38.5 1
## 4     raterID:item 0.00880244     0.7 1
## 5          raterID 0.13906434    11.7 1
## 6             item 0.03339230     2.8 1
## 7         Residual 0.27270445    22.9 1
##
## attr(,"class")
## [1] "gstudy" "list"
```

We see that most of the variance in the data is explained by the differences in the children ($\hat{\sigma}_p$) and that there is some amount of unexplained variance left. This concludes the G-study.

The second part of a G-theory analysis consists of a *D-study*. According to Shavelson and Webb (1991, p.12), the purpose of a G-study is to anticipate the multiple uses of a measurement and to provide as much information as possible about the sources of variation in the measurement. A D-study makes use of the information provided by the G-study to design the best possible application of the measurement. For instance, we might be interested in whether fewer raters are sufficient or more raters are needed or whether fewer items are sufficient or more items are needed. Thus, a D-study helps us to set up a final design for our measurement tasks.

Let n_r denote the number of raters and n_i the number of items. We first print out the D-study variance components. These variance components are for person mean scores over three items and five raters and result from dividing the G-study variance components by the corresponding n (last column; $n_i = 3$, $n_r = 5$, $n_r n_i = 15$; see, e.g., Brennan, 2001, p. 11):

```
dfit <- dstudy(gfit, colname.objects = "personID",
               colname.scores = "score", data = phydat)
dfit$components
##              source          var percent  n
## 1 personID:raterID 0.0254599720     4.3  5
## 2    personID:item 0.0504077998     8.5  3
## 3         personID 0.4587644723    77.4  1
## 4     raterID:item 0.0005868294     0.1 15
## 5          raterID 0.0278128690     4.7  5
## 6             item 0.0111307655     1.9  3
## 7         Residual 0.0181802970     3.1 15
```

In this example we used the same n_r and n_i as in the G-study; this is not necessary however.[3] Based on these new variances, we can compute various measures. First, we focus on the error variance. The *absolute error variance* is simply the difference between a person's observed and universe scores, that is, the sum of all the variance components except the one for the persons:

$$\sigma^2(\Delta) = \sigma_i^2 + \sigma_r^2 + \sigma_{pi}^2 + \sigma_{pr}^2 + \sigma_{ir}^2 + \sigma_{pir,e}^2. \tag{1.8}$$

```
dfit$var.error.abs
## [1] 0.1335785
```

The square root of $\sigma^2(\Delta)$ is interpretable as an *absolute standard error of measurement*, which can be used to construct a confidence interval (CI) for the observed scores:

```
dfit$sem.abs
## [1] 0.365484
```

[3] At the point this book was written, the package did not provide options to specify different n's explicitly.

If this standard error is too large for our purposes, we can change n_i and n_r accordingly in order to get a smaller CI. The *relative error variance* is defined as the difference between a person's observed deviation score and the universe score. Equation (1.8) simplifies to

$$\sigma^2(\delta) = \sigma_i^2 + \sigma_r^2 + \sigma_{ir}^2. \tag{1.9}$$

The square root of $\sigma^2(\delta)$ is the *relative standard error of measurement*.

```
dfit$var.error.rel
## [1] 0.09404807
dfit$sem.rel
## [1] 0.3066726
```

At this point we are able to compute the following two important coefficients. Both are based on the variance of universe scores (which in this design is simply σ_p^2). The first one is called *dependability coefficient* and involves the absolute error variance:

$$\Phi = \frac{\sigma_p^2}{\sigma_p^2 + \sigma^2(\Delta)}. \tag{1.10}$$

```
dfit$dependability
## [1] 0.7744912
```

The second one is the *generalizability coefficient* which involves the relative error variance:

$$\rho^2 = \frac{\sigma_p^2}{\sigma_p^2 + \sigma^2(\delta)}. \tag{1.11}$$

```
dfit$generalizability
## [1] 0.8298735
```

We see that both equations are strikingly similar to the reliability formulation in Eq. (1.2). The main difference is that the error variance is more complex. Equation (1.10) is based on the absolute error variance, whereas Eq. (1.11) is based on the relative error variance. Φ can be used if it makes sense to interpret the observed scores in an absolute way. In general, more attention is paid to ρ^2: it is the

analogue of a reliability coefficient in CTT. In our application, the generalizability coefficient is fairly high. In the D-study, it can be examined how it would change if we would have, for instance, fewer raters. Conversely, if ρ^2 is too low, the user can study how it can be increased by modifying n_i and n_r. Corresponding examples can be found in Brennan (2001, Chapter 4).

In this section we just scratched on the surface of G-theory using a fully crossed two-facet design. Extensions include three or more facets, hierarchical designs (e.g., raters are nested within items), and multivariate G-theory by defining strata (e.g., for multiple subtests). In order to compute such extensions using **gtheory**, all that needs to be changed is the **lme4** formula specification. Detailed descriptions of these extensions can be found in Brennan (2001).

References

Algina, J., & Penfield, R. D. (2009). Classical test theory. In: R. E. Millsap & A. Maydeu-Olivares (Eds.), *The sage handbook of quantitative methods in psychology* (pp. 93–122). Thousand Oaks: Sage.

Bates, D., Mächler, M., Bolker, B., & Walker, S. (2015). Fitting linear mixed-effects models using **lme4**. *Journal of Statistical Software, 67*(1), 1–48. https://doi.org/10.18637/jss.v067.i01.

Brennan, R. L. (2001). *Generalizability theory*. New York: Springer.

Cameletti, M., & Caviezel, V. (2012). **CMC**: Cronbach-Mesbah curve. R package version 1.0. https://CRAN.R-project.org/package=CMC

Coaley, K. (2014). *An introduction to psychological assessment & psychometrics* (2nd ed.). London: Sage.

Crocker, L. M., & Algina, J. (1986). *Introduction to modern and classical test theory*. Belmont: Wadsworth Group/Thomson Learning.

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika, 16*, 297–334.

Cronbach, L. J. (1972). *The dependability of behavioral measurements*. New York: Wiley.

Diedenhofen, B. (2016). **cocron**: Statistical comparisons of two or more alpha coefficients. R package version 1.0-1. https://CRAN.R-project.org/package=cocron

Fletcher, T. D. (2010). **psychometric**: Applied psychometric theory. R package version 2.2. https://CRAN.R-project.org/package=psychometric

Graham, J. M. (2006). Congeneric and (essentially) tau-equivalent estimates of score reliability: What they are and how to use them. *Educational and Psychological Measurement, 66*, 930–944.

Hoyt, W. T., & Melby, J. N. (1999). Dependability of measurement in counseling psychology: An introduction to generalizability theory. *The Counseling Psychologist, 27*, 325–352.

Jackson, P., & Agunwamba, C. (1977). Lower bounds for the reliability of the total score on a test composed of nonhomogeneous items: I: Algebraic lower bounds. *Psychometrika, 42*, 567–578.

Lakes, K. D. (2012). The response to challenge scale (RCS): The development and construct validity of an observer-rated measure of children's self-regulation. *The International Journal of Educational and Psychological Assessment, 10*, 83–96.

Lakes, K. D., & Hoyt, W. T. (2004). Promoting self-regulation through school-based martial arts training. *Journal of Applied Developmental Psychology, 25*, 283–302.

Lakes, K. D., & Hoyt, W. T. (2009). Applications of generalizability theory to clinical child and adolescent psychology research. *Journal of Clinical Child & Adolescent Psychology, 38*, 144–165.

Mair, P., Hofmann, E., Gruber, K., Zeileis, A., & Hornik, K. (2015). Motivation, values, and work design as drivers of participation in the R open source project for statistical computing. *Proceedings of the National Academy of Sciences of the United States of America, 112*, 14788–14792.

McDonald, R. P. (1999). *Test theory: A unified treatment*. Hillsdale: Erlbaum.

Moore, C. T. (2016). **gtheory**: Apply generalizability theory with R. R package version 0.1.2. https://CRAN.R-project.org/package=gtheory

Revelle, W. (2015). An introduction to psychometric theory with applications in R. Freely available online, http://www.personality-project.org/r/book/

Revelle, W. (2017). **psych**: Procedures for psychological, psychometric, and personality research. R package version 1.7.8. http://CRAN.R-project.org/package=psych

Revelle, W., & Zinbarg, R. E. (2009). Coefficients alpha, beta, omega, and the glb: Comments on Sijtsma. *Psychometrika, 74*, 145–154.

Shavelson, R. J., & Webb, N. M. (1991). *Generalizability theory: A primer*. Newbury Park: Sage.

Shrout, P. E., & Fleiss, J. L. (1979). Intraclass correlations: Uses in assessing rater reliability. *Psychological Bulletin, 86*, 420–428.

Sijtmsa, K. (2009). On the use, the misuse, and the very limited usefulness of Cronbach's alpha. *Psychometrika, 74*, 107–120.

Streiner, D. L. (2003). Starting at the beginning: An introduction to coefficient alpha and internal consistency. *Journal of Personality Assessment, 80*, 99–103.

Ten Berge, J. M. F., & Sočan, G. (2004). The greatest lower bound to the reliability of a test and the hypothesis of unidimensionality. *Psychometrika, 69*, 613–625.

Willse, J. T. (2014). **CTT**: Classical test theory functions. R package version 2.1. https://CRAN.R-project.org/package=CTT

Chapter 2
Factor Analysis

2.1 Correlation Coefficients

Before we dive into factor analytic techniques, it is worthwhile to have a closer look at correlation coefficients, since the techniques presented in the next two chapters are based on input correlations or covariances. The most well-known correlation coefficient is, of course, the *Pearson product-moment correlation*. Let **x** and **y** be two metric variables in a dataset ($i = 1, \ldots, n$ observations in total).

$$r_{\mathbf{xy}} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}. \tag{2.1}$$

The formula divides the covariance between **x** and **y** by the product of the standard deviations such that the correlation coefficient is normalized to a $[-1, 1]$ interval. It is important to point out that the Pearson correlation is suited for two metric variables, accounts for bivariate linear relationships only, and does not allow us to make any causal statements.

Another classical correlation coefficient is the *Spearman correlation*, which simply applies Pearson's formula to the ranks of the data. Thus, it takes the data on an ordinal scale level and is able to detect monotone relationships between **x** and **y**. There should not be too many ties in the data when using Spearman correlation. Other versions of Pearson's correlation formula include the *point biserial correlation* (one variable dichotomous, one variable metric) and the ϕ-coefficient (both variables dichotomous).

In psychology, we often have to deal with ordinal data, being it at a dichotomous level (0/1 values) as well as at a polytomous level (e.g., Likert scales). Several correlation-based latent variable methods such as structural equation models or some variants of factor analysis are based on a (multivariate) normality assumption

P. Mair, *Modern Psychometrics with R*, Use R!,
https://doi.org/10.1007/978-3-319-93177-7_2

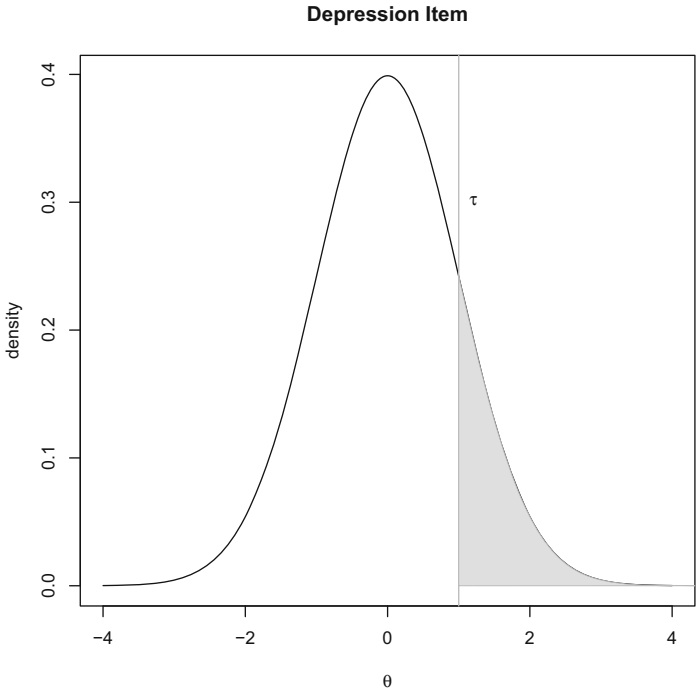

Fig. 2.1 Threshold visualization for a single dichotomous item measuring depression

on the input data. Assuming normality makes sense for metric variables only; for ordinal variables the normal distribution is conceptually the wrong distribution.

We can apply a little trick, however, by assuming that an ordinal response is generated by an underlying normal distribution. Let us consider the following simple example. We have a single dichotomous item which is supposed to measure the latent variable "depression." Depression is on a metric continuum denoted by θ, ranging from (in theory) $-\infty$ (infinitely non-depressed) to ∞ (infinitely depressed). We assume a normal distribution for this latent variable. Highly depressed individuals will score 1 on this item, whereas individuals with weak or no depression will score 0. We can imagine the item being a rater that separates highly depressed from weakly depressed persons. The question arising at this point is: What is the cut point (i.e., a *threshold* τ) for the depression degree, above which an individual scores 1 on this item and below which an individual scores 0. Figure 2.1 illustrates this setup for a fixed τ.

Since correlation coefficients are based on two input variables, let us extend this setup to two items, both measuring depression. The two items switch from 0 to 1 at two different degrees of depression τ_1 and τ_2. If we look at this problem jointly, we look at a bivariate normal distribution. Figure 2.2 illustrates this concept. Our two items slice the distribution into four quadrants.

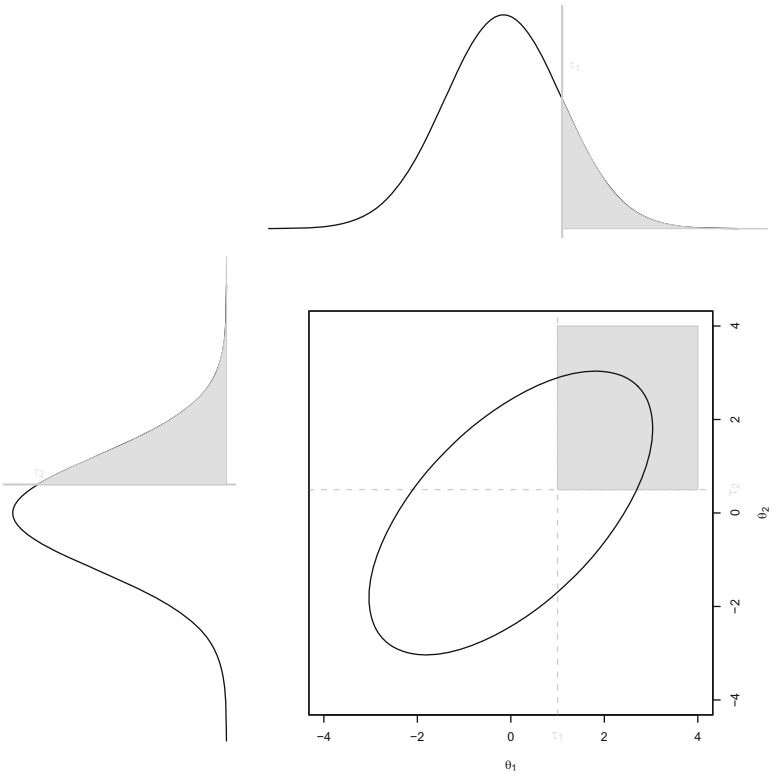

Fig. 2.2 Tetrachoric correlation involving two dichotomous items. The shaded area to the upper right reflects $X_1 > \tau_1$ and $X_2 > \tau_2$

Let us use a real-life dataset from Vaughn-Coaxum et al. (2016) who used the children's depression inventory (CDI) that rates the severity of symptoms related to depression children and adolescents. We pick two items: "I am sad all the time" (item 1) and "I look ugly" (item 2). The items are on a three-point scale with values 0, 1, and 2. For simplicity, we dichotomize the items by merging categories 1 and 2 into a single category. The following code chunk performs the data manipulation and organizes the scores as 2×2 table.

```
library("MPsychoR")
data("YouthDep")
item1 <- YouthDep[, 1]
levels(item1) <- c("0", "1", "1")
item2 <- YouthDep[, 14]
levels(item2) <- c("0", "1", "1")
```

(continued)

```
table(item1, item2)
##       item2
## item1    0    1
##     0 1353  656
##     1  115  166
```

We see that item 2 switches to a score of 1 at a lower part of the depression continuum, compared to item 1. For instance, someone with a moderate depression may score 1 on item 2 but 0 on item 1. Thus, ultimately, τ_2 should be lower than τ_1.

A correlation coefficient suited for dichotomous data and based on this underlying normal strategy is the *tetrachoric correlation*. It gives us a single number describing the degree of dependence in the table above with the extreme values of 1 if the off-diagonals are 0 and -1 if the diagonals are 0. In addition, we get estimates for the thresholds τ_1 and τ_2. In order to compute the tetrachoric correlation, we can use the corresponding function from the **psych** package (Revelle, 2017).

```
library("psych")
tetcor <- tetrachoric(cbind(item1, item2))
tetcor
## Call: tetrachoric(x = cbind(item1, item2))
## tetrachoric correlation
##       item1 item2
## item1 1.00
## item2 0.35  1.00
##
##  with tau of
## item1 item2
##  1.16  0.36
```

We get a tetrachoric correlation coefficient of 0.35, and the threshold parameters are $\tau_1 = 1.162$ and $\tau_2 = 0.361$. Details on the computation of a tetrachoric correlation are given in Kirk (1973). The `draw.tetra()` function in the **psych** package provides some helpful illustrations. An approach to test assumptions underlying tetrachoric correlations is presented in Muthén and Hofacker (1988).

The concept of underlying normal traits as used in tetrachoric correlation for dichotomous items can be generalized to settings with two polytomous items. In this case, the coefficient is called *polychoric correlation*. For each item we get multiple threshold parameters (number of categories -1). It can be visualized in the same way as we did in Fig. 2.2. To illustrate, we use the same two items as above; but this time we keep them on the original three-point scale. Thus, we get two threshold parameters for each item denoting the switching points for the categories on the underlying depression trait.

```
item1 <- YouthDep[, 1]
item2 <- YouthDep[, 14]
polcor <- polychoric(cbind(item1, item2))
polcor
## Call: polychoric(x = cbind(item1, item2))
## Polychoric correlations
##       item1 item2
## item1 1.00
## item2 0.33  1.00
##
##  with tau of
##          1   2
## item1 1.16 2.3
## item2 0.36 1.2
```

We get a polychoric correlation coefficient of 0.333 and two threshold parameters for each item.

Another variant of this correlation concept is the *polyserial correlation*, where one of the two variables is metric and the other one is ordinal (see `polyserial` function in **psych**). Similar to tetrachoric correlations, the computation of polychoric/polyserial correlations is rather technical. Details can be found in Drasgow (1986).

A final note concerns two warnings we sometimes get when applying the functions `tetrachoric` and `polychoric`, which we should not blindly ignore. The first one says that some "cells were adjusted for 0 values using the correction for continuity." As we have seen, tetrachoric/polychoric correlations are based on frequency tables. Sometimes, especially having small samples and/or many categories, some cells become 0. In such cases a *continuity correction* is applied (by default the 0's are replaced by 0.5; see the `correct` argument). If there are many 0's, the results may become unstable, and different corrections lead to heavily different results (see Savalei, 2011). Let us look at an example using all the CDI items of the youth depression dataset:

```
DepItems <- YouthDep[,1:26]
Depnum <- data.matrix(DepItems) - 1  ## convert to numeric
Rdep <- polychoric(Depnum)
## 7 cells were adjusted for 0 values using the correction for
## continuity. Examine your data carefully.
```

Note that the dataset is fairly large ($n = 2290$) and only seven cells were 0. In such a case, we can assume that the solution is fairly stable.

Note that as the number of response categories increases, one can use the Pearson correlation instead of polychoric since both correlation coefficients lead to the

same results. A detailed study of this phenomenon within the context of structural equation models can be found in Rhemtulla et al. (2012). The authors suggest that in the area of 6, 7, or more categories, differences between Pearson and polychoric become negligible.

A second warning we can get is that the matrix was not positive definite and that some smoothing was done. A positive definite matrix is a matrix whose eigenvalues are positive.[1] Correlation matrices have to be positive semidefinite, that is, eigenvalues need to be larger than 0. In the case of tetrachoric/polychoric correlation computations, we can end up in a situation where one or more eigenvalues are negative. Here is an example using data taken from Mair et al. (2015) who examined motivational structures of R package developers. In Sect. 1.2 we used the hybrid motivation items. Here we use the intrinsic and extrinsic motivation items (17 items in total) and compute the correlation matrix. Since the items are dichotomous, we use the tetrachoric correlation.

```
data("Rmotivation")
vind <- grep("ext|int", colnames(Rmotivation))
Rmotivation1 <- Rmotivation[, vind]
Rmot1 <- tetrachoric(Rmotivation1, smooth = FALSE)
tail(round(eigen(Rmot1$rho)$values, 3))
## [1]  0.384  0.268  0.176  0.114  0.062 -0.035
```

We print out the last six eigenvalues and see that the last eigenvalue is negative. Thus, this matrix does not fulfill the properties of a correlation matrix. The trick is now to apply some smoothing on the correlations. The **psych** package provides the `cor.smooth` function which we can evoke in the `tetrachoric` call by setting the `smooth` argument to `TRUE` (default[2]).

```
Rmot <- tetrachoric(Rmotivation1)
tail(round(eigen(Rmot$rho)$values, 3))
## [1] 0.383 0.268 0.176 0.114 0.062 0.000
```

We see that the eigenvalues are all non-negative which implies that we have obtained a proper correlation matrix. In general, in such situations we have to consider how many eigenvalues are negative and what their magnitude is. In this example we only had one eigenvalue slightly below 0, and thus it is fairly safe to proceed with this smoothed correlation matrix. Details on how the smoothing is done can be found in the `cor.smooth` help file.

[1] Eigenvalues will be introduced in Sect. 6.1.1.

[2] This call gives a warning that the matrix is not positive definite.

2.2 Exploratory Factor Analysis

Exploratory factor analysis (EFA) is one of the very classical latent variable techniques. The basic idea is to find latent variables (*factors*) based on the correlation structure of the manifest input variables (*indicators*).

First, EFA needs to be clearly distinguished from confirmatory factor analysis (CFA; introduced in Sect. 2.4). As we will see, in EFA each indicator loads on each factor, and the factors are generally uncorrelated. In CFA the user determines which indicator is associated with which factor, based on an underlying substantive theory. The factors are generally correlated. Second, we should also distinguish EFA from principal component analysis (PCA; see Chap. 6), a dimension reduction technique for metric data. Corresponding differences between the two techniques will be outlined at the end of Sect. 6.1.2. Third, factor analysis was originally designed for metric input variables. Having ordinal data, tetrachoric/polychoric correlations can be used. This strategy is sometimes called the *underlying variable approach* since we assume that the ordinal variable comes from an underlying normal distribution. An alternative, modern way of performing factor analysis on categorical data that avoids the computation of correlations entirely is called *item factor analysis*. This approach will be described in Sect. 4.7 within the context of item response theory. Further details on relationships between these techniques can be found in Bartholomew and Knott (1999) and Bartholomew et al. (2008).

2.2.1 EFA Model Formulation and Computation

EFA operates on a multivariate dataset with m manifest variables (indicators). Let $\mathbf{X}$ denote the $n \times m$ data matrix, with n being the sample size. In EFA applications, some sets of variables are typically related to each other and therefore show a particular correlation structure, captured by an $m \times m$ correlation matrix $\mathbf{R}$.

In EFA each set (or block) of variables reflects a latent variable. In other words, EFA tries to find p latent variables on the basis of the correlation structure of the m manifest variables. These underlying latent variables influence the manifest variables. Mathematically, the EFA problem can be formulated as follows:

$$\begin{aligned} x_1 &= \lambda_{11}\xi_1 + \lambda_{12}\xi_2 + \cdots + \lambda_{1p}\xi_p + \varepsilon_1 \\ x_2 &= \lambda_{21}\xi_1 + \lambda_{22}\xi_2 + \cdots + \lambda_{2p}\xi_p + \varepsilon_2 \\ &\vdots \\ x_m &= \lambda_{m1}\xi_1 + \lambda_{m2}\xi_2 + \cdots + \lambda_{mp}\xi_p + \varepsilon_m \end{aligned} \tag{2.2}$$

This looks suspiciously like a collection of regression equations—which it actually is but with one problem: the right-hand side of the equation is fully unknown. It

also shows similarities with the true score model in Eq. (1.1). While G-theory (see Sect. 1.3) is concerned with a complex error structure, factor analysis deals with a more complex latent variable (i.e., true score) structure.

On the left-hand side of Eq. (2.2), we have our indicators $x_1, \ldots, x_m$. The ξ's on the right-hand side are the factors. They are sometimes also called *common factors* since they influence all our manifest variables. The λ's determine how strong the influence of each common factor on the respective manifest variable is. They are typically referred to as *factor loadings*. The ε's are the *unique factors* since each manifest variable gets its own unique ε. The number of factors p is typically much smaller than m. Using matrix notation, Eq. (2.2) becomes

$$\mathbf{x} = \boldsymbol{\Lambda}\boldsymbol{\xi} + \boldsymbol{\varepsilon}. \tag{2.3}$$

$\boldsymbol{\Lambda}$ is a $m \times p$ matrix of loadings, and $\mathbf{x}$ and $\boldsymbol{\xi}$ are random vectors of length m containing indicators and the factors. In this form, since there are so many unknowns, the model is *indeterminate* (i.e., we cannot estimate it in a simple way).

However, the problem can be reformulated using either the model variance-covariance (VC) matrix or the correlation matrix (see, e.g., MacCallum, 2009). Here we show the equation for the model correlation matrix:

$$\mathbf{P} = \boldsymbol{\Lambda}\boldsymbol{\Phi}\boldsymbol{\Lambda}' + \boldsymbol{\Psi}, \tag{2.4}$$

which is the *fundamental equation in factor analysis*. $\mathbf{P}$ is the $m \times m$ implied model correlation matrix, $\boldsymbol{\Psi}$ is an $m \times m$ diagonal matrix containing the unique factor variances, and $\boldsymbol{\Phi}$ is the correlation matrix of the factors. If we assume that the factors are independent from each other, $\boldsymbol{\Phi}$ is an identity matrix and Eq. (2.4) simplifies to

$$\mathbf{P} = \boldsymbol{\Lambda}\boldsymbol{\Lambda}' + \boldsymbol{\Psi}. \tag{2.5}$$

We see that the variances of the manifest variables are partitioned into a portion accounted for by the common factor effects (i.e., $\boldsymbol{\Lambda}\boldsymbol{\Lambda}'$ called *communalities*) and a portion accounted for by the unique factors given in $\boldsymbol{\Psi}$.

To summarize, the two main constituents of factor analysis are the loadings and the communalities; both of them are unknown. The problem is that in order to estimate the communalities, we need the loadings. Conversely, in order to estimate the loadings, we need the communalities. In EFA, this dilemma is referred to as the *communality problem*. Over the years several estimation approaches have been proposed in order to be able to fit the EFA model. The most relevant ones are maximum likelihood (ML) and least squares approaches that minimize various versions of residuals.[3] Differences among these methods are described in Revelle (2015). Regardless which algorithm we use (unless we do it in a Bayesian way as presented in Sect. 2.3), we need to fix the number of factors p a priori.

[3] In EFA, residuals are defined by $(\mathbf{R} - \hat{\mathbf{P}})$, where $\mathbf{R}$ is the sample correlation matrix and $\hat{\mathbf{P}}$ the estimated model correlation matrix.

R has a basic EFA implementation by means of the function `factanal` that uses ML. ML assumes that the data are multivariate normally distributed. The **psych** package offers a more comprehensive implementation (`fa` function) which we will use throughout this chapter. To illustrate, let us use the tetrachoric correlation matrix from Sect. 2.1 on the intrinsic/extrinsic motivation items from Mair et al. (2015) and fit an EFA. We fix the number of factors to $p = 2$, use ML estimation, and print out the loadings matrix $\boldsymbol{\Lambda}$. Note that we blank out loadings <0.2 while keeping in mind that each indicator gets a loading on each factor according to Eq. (2.2).

```
motFA <- fa(Rmot$rho, nfactors = 2, rotate = "none", fm = "ml")
print(motFA$loadings, cutoff = 0.2)
##
## Loadings:
##       ML1    ML2
## ext1          0.282
## ext2          0.218
## ext3   0.288
## ext4   0.237  0.626
## ext5
## ext6
## ext7   0.306  0.430
## ext8          0.909
## ext9   0.361  0.430
## ext10         0.593
## ext11         0.582
## ext12         0.765
## int1   0.817
## int2   0.721
## int3   0.804
## int4   0.736
## int5   0.881
##
##                  ML1   ML2
## SS loadings    3.667 3.105
## Proportion Var 0.216 0.183
## Cumulative Var 0.216 0.398
```

Note that the `fa` function takes as input either a correlation matrix or the raw data. By providing the raw data, the type of correlation can be specified via the `cor` argument. The communalities can be extracted as follows:

```
round(motFA$communality, 2)
##  ext1  ext2  ext3  ext4  ext5  ext6  ext7  ext8  ext9 ext10
##  0.11  0.08  0.09  0.45  0.00  0.00  0.28  0.83  0.31  0.37
## ext11 ext12  int1  int2  int3  int4  int5
##  0.37  0.61  0.70  0.54  0.68  0.55  0.80
```

These communalities are simply the sum of the squared loadings and represent the proportion of variance (i.e., squared multiple correlations) explained by the common factors. The intrinsic items have fairly high communalities (the higher, the better), whereas some of the extrinsic items have low values. In order to fully interpret the solution, there are still two more aspects that need to be addressed: factor rotation and how to determine the number of factors.

2.2.2 Factor Rotation and Interpretation

After fitting an EFA, we typically want to interpret (i.e., label or name) the factors based on the loading patterns. By looking at the loadings matrix in the example above, we see, for instance, that the intrinsic motivation items load highly on factor 1. Their loadings on the second factor are small. Thus, factor 1 can be interpreted as "intrinsic motivation." However, in order to get an even clearer picture, in EFA we typically apply a rotation on the loadings matrix. Such a rotation does not change the fit of the model; it is only done for interpretation purposes by transforming the loadings. We distinguish between two basic types of rotations: *orthogonal* and *non-orthogonal rotation*.[4]

Let us start with orthogonal rotation techniques. Using linear algebra, rotations are achieved by multiplying a matrix with a rotation matrix $\mathbf{T}$. In EFA geometry, the loadings represent vector coordinates. Thus, a rotation is carried out on the basis of the estimated $m \times p$ loadings matrix $\hat{\boldsymbol{\Lambda}}$:

$$\hat{\boldsymbol{\Lambda}}_r = \hat{\boldsymbol{\Lambda}}\mathbf{T}. \tag{2.6}$$

$\hat{\boldsymbol{\Lambda}}_r$ is the $m \times p$ matrix of rotated loadings, and $\mathbf{T}$ is the $p \times p$ rotation matrix. In orthogonal rotation we impose the following restriction on the rotation matrix: $\mathbf{T}\mathbf{T}' = \mathbf{I}$ with $\mathbf{I}$ as identity matrix. This keeps the orthogonal factor structure intact, and the fundamental equation from Eq. (2.5) holds:

$$\hat{\mathbf{P}} = \hat{\boldsymbol{\Lambda}}_r \hat{\boldsymbol{\Lambda}}_r' + \hat{\boldsymbol{\Psi}} = \hat{\boldsymbol{\Lambda}} \hat{\boldsymbol{\Lambda}}' + \hat{\boldsymbol{\Psi}}, \tag{2.7}$$

Thus, the loadings are altered but the fit remains unchanged.

Still, we need to find $\mathbf{T}$. This problem was successfully solved by Kaiser (1958) who found $\mathbf{T}$ by maximizing the sum of the variances of the squared factor loadings on the separate factors. This strategy is called *varimax* rotation and is the most popular orthogonal rotation technique. Alternatives are *quartimax* (should be used if a researcher suspects a single dominating general factor), and *equimax* as a

[4] An overview of rotation techniques and corresponding comparisons can be found in Browne (2001).

compromise between *varimax* and *quartimax*. Let us apply a varimax rotation on the motivation EFA solution from above:

```
motFA2 <- fa(Rmot$rho, nfactors = 2, rotate = "varimax",
             fm = "ml")
```

Since this is a two-dimensional solution, we can represent the loadings by means of a 2D loadings plot. Figure 2.3 shows the effect of the rotation which, in this example, was only minor. This is due to the fact that the unrotated solution already gave us a fairly good loadings picture. Note that the communalities remain unchanged with respect to the unrotated solution. We see that the items with low communalities are close to the origin and items with high communalities go clearly into either factor 1 direction or factor 2 direction. In terms of interpretation, it is safe to name factor 1 as "intrinsic motivation" and factor 2 as "extrinsic motivation." The two factors are independent from each other.

For $p > 2$, a loadings plot matrix can be obtained using the `factor.plot` function in **psych**. Alternatively, we can look at the loadings matrix and look for corresponding loading patterns in order to achieve a reasonable factor interpretation.

Let us proceed with non-orthogonal rotation which abandons the $\mathbf{TT}' = \mathbf{I}$ restriction. As a consequence, the factors are not independent from each other anymore. Such rotations are also called *oblique rotations*. Popular oblique rotation approaches are *oblimin* and *promax* (Hendrickson and White, 1964). In practice, EFA with oblique rotation is often used prior to a CFA in order to explore whether the underlying latent structure theory is reflected by the data. Most often, theories do not assume that the underlying factors are independent. In such cases non-orthogonal rotation is a very attractive instrument.

We consider once more the R motivation example from above. In addition to extrinsic and intrinsic motivation items, we include the items on hybrid motivation and fit a three-factor solution. It would not be wise to assume independence between the three factors since we can expect that hybrid motivation is not completely independent from neither intrinsic nor extrinsic motivation.

```
Rmot2 <- tetrachoric(Rmotivation[,1:36])
motFA3 <- fa(Rmot2$rho, nfactors = 3, rotate = "oblimin",
             fm = "ml")
motFA3$loadings
```

From the loadings matrix (not shown here), we see that the first factor can be interpreted as intrinsic motivation, the second factor as extrinsic motivation, and the third factor as hybrid motivation. Note that some of the hybrid motivation items have

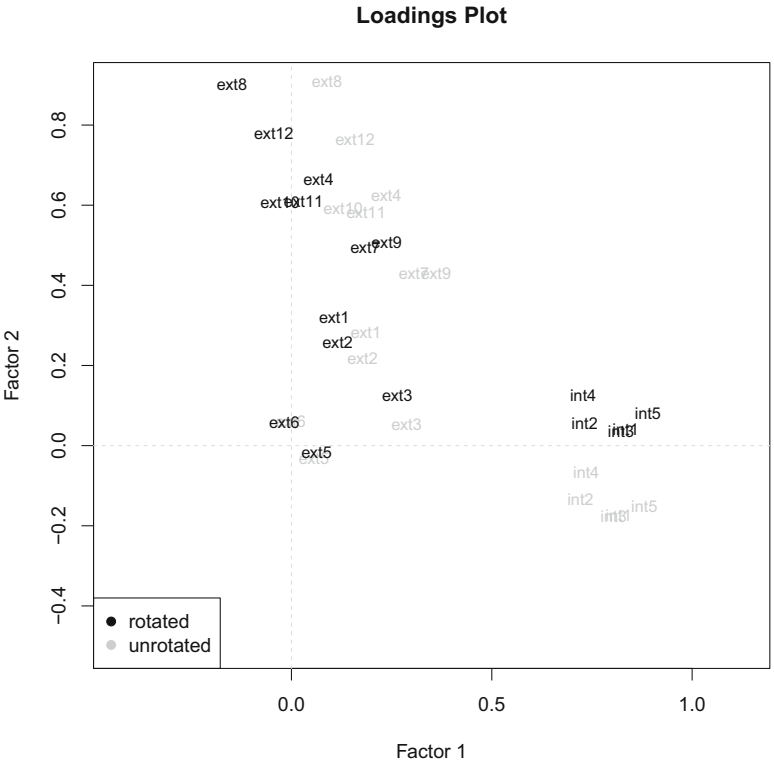

Fig. 2.3 Unrotated (gray labels) and rotated (black labels) solution for the extrinsic/intrinsic motivation items

considerably high loadings on factor 1, not so much on factor 2. Since the factors are correlated, we can also request the factor correlation matrix:

```
round(motFA3$Phi, 3)
##       ML1   ML2   ML3
## ML1 1.000 0.119 0.505
## ML2 0.119 1.000 0.090
## ML3 0.505 0.090 1.000
```

The correlation among the first two factors (intrinsic vs. extrinsic) is quite low. The third factor (hybrid motivation) shows a fairly high correlation with intrinsic motivation.

2.2.3 Factor Scores

Before we discuss how to determine the number of factors, let us have a quick look at *factor scores*. Factors represent "new" variables in our dataset. That is, each of the n individuals gets a score on each of the p factors (called *factor score*). Unlike PCA, which estimates the scores directly, in EFA the factor scores are computed post hoc. The most common approach to obtain factor scores is via a regression approach. First, a factor score coefficient matrix $\hat{\mathbf{B}}$ is estimated from the sample correlation and loadings matrix according to

$$\hat{\mathbf{B}} = \mathbf{R}^{-1}\hat{\boldsymbol{\Lambda}}. \tag{2.8}$$

This matrix is subsequently subject to a regression fit using the standardized version of the input data matrix $\mathbf{X}$, here denoted by $\mathbf{Z}$:

$$\hat{\mathbf{F}} = \mathbf{Z}\hat{\mathbf{B}}. \tag{2.9}$$

The $n \times p$ matrix $\hat{\mathbf{F}}$ contains the factor scores.

In order to get the factor scores in our example, we need to re-fit the model using the raw data matrix as input. Note that our data contain some missing values. The **psych** package provides some simple imputation mechanisms (here we use the median). We focus again on the orthogonal (varimax rotated) two-factor solution involving intrinsic and extrinsic motivation items, for which the factor scores can be computed as follows:

```
motFA2 <- fa(Rmotivation1, nfactors = 2, rotate = "varimax",
             cor = "tet", fm = "ml", scores = "regression",
             missing = TRUE, impute = "median")
dim(motFA2$scores)
## [1] 852   2
```

For each person we get a score on the intrinsic and the extrinsic factor. Care is advised with using the factor scores for further analyses since the factor scores are not unique (due to *factor indeterminacy*, i.e., the factors cannot be determined exactly from the manifest variables). Details can be found in McDonald and Mulaik (1979). Other algorithms to compute the factor scores can be specified via the `scores` argument.

2.2.4 Determining the Number of Factors

There is one final ingredient missing in our EFA toolbox: How many factors should we extract? Several statistical criteria for the choice of p have been proposed in the literature which we will discuss in this section. However, it is important to point out that we should not rely on a single criterion but rather consider multiple criteria in conjunction with the interpretability of the solution. We group the criteria as follows: ad hoc criteria, parallel analysis and very simple structure, and statistical goodness-of-fit indices. Throughout this section we use the CDI dataset from Vaughn-Coaxum et al. (2016), introduced in Sect. 2.1. From the theory we can expect a single depression factor. EFA helps us to assess whether our data are in line with this single factor theory or whether a multiple-factor solution is needed.

Before fitting an EFA, we can look at some ad hoc criteria. One of them is to carry out an *eigenvalue decomposition* of the input correlation matrix. As we will see in Chap. 6, this means that we are fitting a PCA. An eigenvalue tells us how much variance is explained by each factor (or component, to be precise). Thus, the higher an eigenvalue of a particular factor/component, the more variance this factor/component explains with respect to our original data.

Since we have 26 variables in our CDI dataset, we can extract 26 eigenvalues from **R** (polychoric, in our example). Note that eigenvalues are always of decreasing order. We can plot the number of components on the x-axis and the eigenvalues on the y-axis. This strategy is called a *scree plot* (Cattell, 1966) and shown in Fig. 2.4.

```
Rdep <- polychoric(Depnum)$rho
evals <- eigen(Rdep)$values
scree(Rdep, factors = FALSE)
```

In this plot we should look at two things. First, we hope to see a clear cut point which separates the systematic structure (*rock*) from the random structure (*scree*). This point is called the *elbow* and we retain the number of factors from the elbow to the left. In our example we have a strongly dominating first factor. Second, Kaiser (1960) suggests that eigenvalues should be larger than 1. The rationale behind it is that if a factor is associated with an eigenvalue smaller than 1, it accounts for less variability than a single variable does. In our example we would retain four factors. Both rules have been criticized in the literature for subjectivity (elbow criterion) and overestimation of the factor number (eigenvalues larger 1). Nevertheless, the scree plot gives us a good first picture of data reduction possibilities.

Note that since eigenvalues tell us how much variance is explained by each factor, we can easily compute the proportion of explained variance (in %) of, for instance, the first two factors:

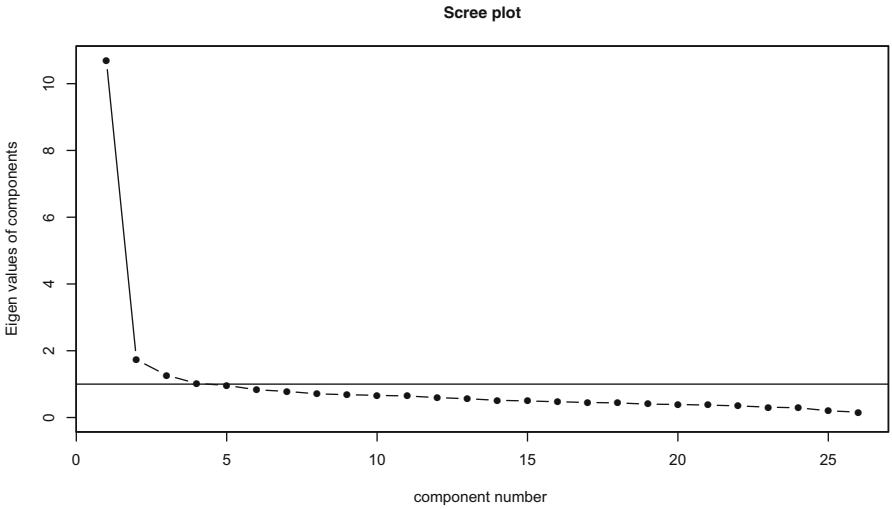

Fig. 2.4 Scree plot for CDI data. The horizontal line denotes an eigenvalue of 1

```
(evals/sum(evals)*100)[1:2]
## [1] 41.143134  6.655083
```

The first factor already explains around 41.1% of the variance in the data, the second factor additional 6.7%.

A scree plot is also the main output of *parallel analysis* (Horn, 1965). Parallel analysis performs a full model fit (i.e., $p = m$) on the original dataset, on resampled data (bootstrap) as well as on random, uncorrelated data matrices drawn from a normal distribution. Based on the eigenvalues, three scree plots are produced as shown in Fig. 2.5.

```
set.seed(123)
resPA <- fa.parallel(Depnum, fa = "pc", cor = "poly",fm = "ml")
```

In parallel analysis, the criterion to be used in order to determine the number of factors is the following: A factor is considered as "significant" if its eigenvalue is larger than the 95% quantile (red line) of those obtained from random or resampled data. In our example we would pick two or three factors. Note that as Revelle (2015, p. 176) points out, parallel analysis is partially sensitive to sample size in that for large samples (as in our example), the eigenvalues of random factors will tend to be very small and thus the number of components or factors will tend to be more than using other rules.

Another criterion is called *very simple structure* (VSS; Revelle and Rocklin, 1979) analysis and is based on the following idea (see Revelle, 2015, p. 177). When it comes to factor interpretation, we often set a cutoff point (e.g., 0.2 as in Sect. 2.2.1 or even higher) for blanking out loadings. That is, we interpret a loadings matrix

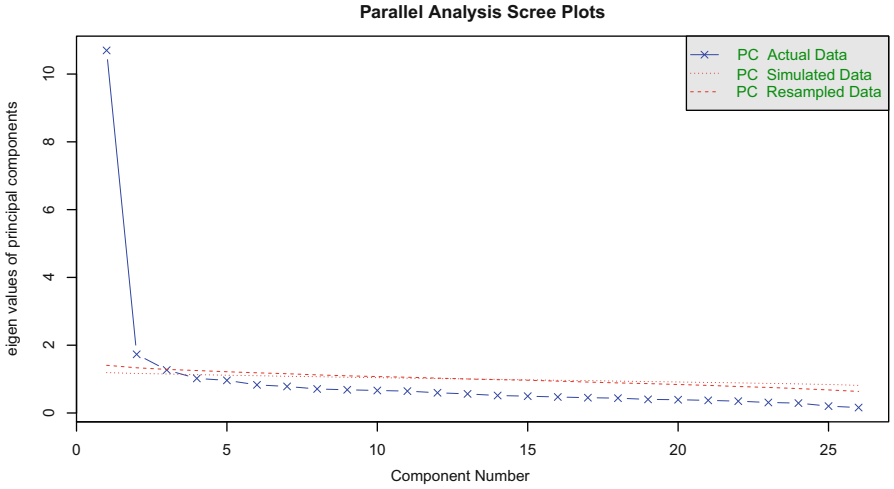

Fig. 2.5 Parallel analysis scree plot: The solid line is simply the scree plot based on observed data (cf. Fig. 2.4). The red lines reflects the 95% quantiles of the bootstrapped data (resampled; dotted) and random data (generated from a normal distribution; dashed)

which is much simpler than the one actually fitted. Such simple loadings matrices are at the core of VSS. VSS fits factor models by varying p systematically. We define a number c denoting the amount of non-zero loadings of a variable in the model fit. This is called *complexity*. For instance, $c = 2$ means that we only consider two non-zero loadings per variable. The two largest loadings (absolute values) go into a loadings matrix $\hat{\boldsymbol{\Lambda}}_{pc}$; the remaining elements are 0. Thus, $\hat{\boldsymbol{\Lambda}}_{pc}$ represents a simplified version of the original loadings matrix. The corresponding communalities are $\hat{\boldsymbol{\Lambda}}_{pc}\hat{\boldsymbol{\Lambda}}'_{pc}$. How well these communalities approximate our observed correlation matrix $\mathbf{R}$ can be quantified by the residual matrix $\mathbf{S}_{pc} = \mathbf{R} - \hat{\boldsymbol{\Lambda}}_{pc}\hat{\boldsymbol{\Lambda}}'_{pc}$. The VSS criterion is defined as follows:

$$VSS_{pc} = 1 - \frac{MS_{S_{pc}}}{MS_R}. \tag{2.10}$$

The higher this value, the better the simple loadings structure represents our data. For a given c, VSS_{pc} is computed systematically for varying p with $p \geq c$. In our example we get the following solution:

```
resvss <- vss(Rdep, fm = "ml", n.obs = nrow(Depnum), plot = FALSE)
resvss
##
## Very Simple Structure
## Call: vss(x = Rdep, fm = "ml", n.obs = nrow(Depnum))
## VSS complexity 1 achieves a maximimum of 0.9  with  1  factors
## VSS complexity 2 achieves a maximimum of 0.92  with  2  factors
##
## The Velicer MAP achieves a minimum of 0.01  with  3  factors
```

By default the `vss` function prints out the maximum VSS values for $c = 1$ and $c = 2$. For $c = 1$ it suggests a one-factor solution, whereas for $c = 2$ a two-factor solution is a good choice. Note that for $c = 2$, it is not possible to obtain a one-factor solution since p cannot be smaller than c.

The last line reports a criterion called *minimum average partial* (MAP; Velicer, 1976). The basic idea is to partial out the factors/components from the input correlation matrix $\mathbf{R}$. This leads to a partial correlation matrix $\mathbf{R}^*$ (see Velicer, 1976; Revelle, 2015, for details). The MAP criterion can be computed by the sum-of-squares of the lower triangular part of $\mathbf{R}^*$. The smaller the partial correlations (and therefore the sum-of-squares), the smaller the MAP and the better the fit. In our example, the MAP criterion suggests three factors.

Finally, we look at some parametric goodness-of-fit indices based on the sample correlation matrix $\mathbf{R}$ and the implied (fitted) correlation matrix $\hat{\mathbf{P}}$. The closer these two matrices are to each other, the better the model fit. The indices presented below will be especially important in CFA and structural equation modeling (see Chap. 3). The first criterion is the *root mean squared residual* (RMSR):

$$RMSR = \sqrt{\frac{\sum_j \sum_{k<j} (r_{jk} - \hat{r}_{jk})^2}{m(m-1)/2}} \tag{2.11}$$

Here, r_{jk} denotes a single element in $\mathbf{R}$ and $\hat{r}_{jk}$ a single element in $\hat{\mathbf{P}}$. The smaller the RMSR, the better the fit.

Another useful measure is the *root mean squared error of approximation* (RMSEA) which is based on the discrepancy function of the ML estimation (see, e.g., MacCallum, 2009, for technical details). The classical guidelines for RMSEA evaluations are given by Browne and Cudeck (1993): values smaller than 0.05 indicate good fit, 0.6–0.8 fair fit, and values larger than 0.10 poor fit. One should also look at the corresponding confidence interval (CI; typically a 90% CI is reported).

Another popular measure is the *Tucker-Lewis index* (TLI; Tucker and Lewis, 1973). The idea is again very simple: it compares a worst-case model Q_0 (i.e., a zero-factor model) and a best-case model with our fitted model Q_p.

$$TLI_p = \frac{Q_0 - Q_p}{Q_0 - 1}. \tag{2.12}$$

TLI_p is in the [0, 1] interval; the larger its value the better the fit. Hu and Bentler (1999) consider values above 0.90 as "good" fit.

Let us fit a one-dimensional solution on our CDI data and print out the indices covered so far:

```
fadep <- fa(Depnum, 1, cor = "poly", fm = "ml")
summary(fadep)
## The root mean square of the residuals (RMSA) is  0.06
## The df corrected root mean square of the residuals is  0.07
##
## Tucker Lewis Index of factoring reliability =  0.76
## RMSEA index =  0.099
##  and the 10 % confidence intervals are  0.097 0.101
```

We see that the TLI is considerably low, and the RMSEA is barely within an acceptable range.

Other options for determining the number of factors in R are the following. The **psych** package offers the convenience function `nfactors`, which fits a sequence of EFA models with varying p and prints out several criteria. Below we specify a maximum of eight factors (output not shown here).

```
resnf <- nfactors(Depnum, n = 8, fm = "ml", cor = "poly")
resnf
```

The **nFactors** package (Raiche and Magis, 2011) offers additional possibilities for assessing the factor number.

Last but not least, the final criterion is the interpretability. Of course, there is always some subjectivity involved when it comes to using interpretability as criterion. In our example we achieve the best interpretability with one factor since the scale is supposed to measure a single underlying latent variable, that is, depression. Some of the statistical criteria support this assertion; some others suggest to extract more factors. At the end it also comes down how strict we want to be. If we want to construct a scale, we should be strict but then item response theory is a more attractive modeling framework. If we merely want to explore the factor structure of a well-established scale, as in our example, we see that the CDI gives a highly dominating general factor of depression.

2.3 Bayesian Exploratory Factor Analysis

This is the first Bayesian section in this book and at the same time the most difficult one. It is assumed that the reader has some basic knowledge of Bayesian techniques. If not, accessible introductory texts are Kruschke (2014) and Kaplan (2014).

Conti et al. (2014) proposed Bayesian version of EFA (BEFA), implemented in the **BayesFM** package (Piatek, 2017). Technical BEFA details are beyond the scope of this book and can be found in the corresponding publication. Here we focus on applied BEFA aspects, that is, we show how to specify the priors, how to fit a BEFA in R using a real-life dataset, and how to interpret the results.[5]

The dataset we use is from Treiblmaier (2006, see also Treiblmaier et al. 2011). It contains items measuring various advantages and disadvantages online users perceive when providing personal information on the Internet. The items are based on 25 qualitative interviews and represent opinions of both organizations and individuals. Advantages of providing personal information online include support for purchasing decisions, increased satisfaction, targeted communication, participation in raffles, time savings, and interesting content. Disadvantages include

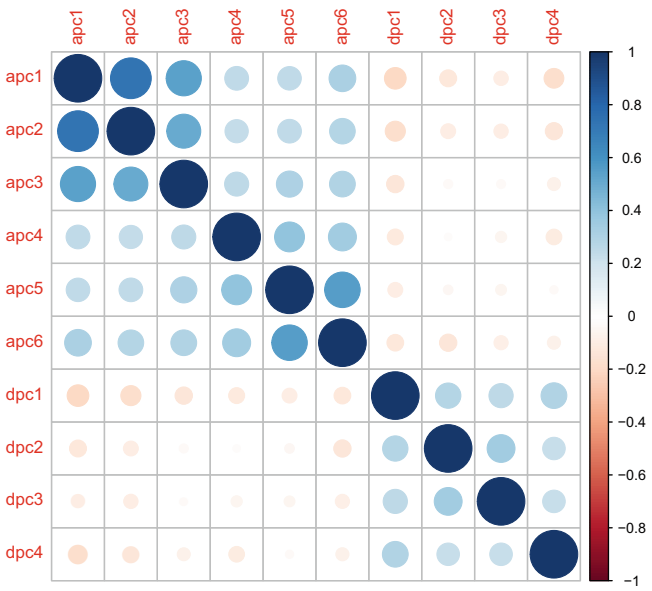

Fig. 2.6 Correlation structure in the Privacy dataset

unsolicited advertising, excessive data collection, lack of information about data usage, and decreasing service quality. In this analysis we include six items related

[5]Thanks to Rémi Piatek and Sylvia Frühwirth-Schnatter for their support with this application.

to advantages of personal communication (variable names starting with `apc`) and four items related to disadvantages of personal communication (`dpc`). Each item was scored on a slide bar from 1 to 100.

We start the analysis with using the **corrplot** package (Wei and Simko, 2016) to display the entire correlation matrix (see Fig. 2.6). Note that for BEFA it is recommended to standardize the variables.

```
library("MPsychoR")
library("corrplot")
library("BayesFM")
data("Privacy")
Privstd <- scale(Privacy)
corrplot(cor(Privstd))
```

The plot shows two separate sets of variables (advantage items vs. disadvantage items) with high correlations among the items within each set. By having a closer look, we see that the advantage itemset could potentially be split up into another two subsets.

BEFA uses the same basic equations as EFA, and it generates correlated factors. However, the way the problem is technically solved is completely different from standard EFA since it uses *Markov Chain Monte Carlo* (MCMC) to get the posterior distributions of the parameters. Major selling points of this approach are that it allows the factors to be correlated and that it estimates the number of factors p, instead of fixing them a priori. In BEFA, the rotation problem is solved by specifying a *dedicated structure* for the factor loadings matrix: factors are either associated with least 2–3 measurements, or not associated with any measurements, in which case they are discarded from the model (Conti et al., 2014, p. 8).

In order to fit a BEFA in R, we first need to set an upper limit for the number of factors. This is a necessary identification constraint in the model. To do so we first need fix the minimum number of manifest variables per factor (here we set it to 2).[6] Based on this value, the maximum number of factors can be computed as follows:

```
Nid <- 2            ## minimum number of variables per factor
pmax <- trunc(ncol(Privstd)/Nid) ## maximum number of factors
pmax
## [1] 5
```

[6] In their original paper, Conti et al. (2014) use the more restrictive assumption of at least three manifest variables per active factor, to rule out potential identification problems due to extreme cases with zero correlation between some factors. With correlated factors, however, the weaker assumption of two manifest variables per factor is sufficient for identification.

The prior setup in BEFA is a bit tricky. Let us start with the correlation (or covariance) matrix for which the package uses an inverse Wishart distribution with parameters ν_0 (degrees of freedom) and S_0 (scale parameter). The ν_0 parameter is of critical importance since it steers the correlation among the factors. The larger ν_0, the larger the a priori factor correlation, and, consequently, the larger the number of factors p since *factor splitting* can occur. In order to determine a feasible prior distribution for the correlation matrix, we vary ν_0 systematically from 6 to 15. We keep S_0 at a default value of 1. For each parameter setting, the function below simulates 10,000 matrices and computes the maximum absolute correlation and the minimum eigenvalue of each correlation matrix. Each of these measures summarizes the overall factor correlation structure by a single number.

```
set.seed(123)
Rsim <- simul.R.prior(pmax, nu0 = pmax + c(1, 2, 5, 7, 10))
plot(Rsim)
```

In Fig. 2.7 we look for a distribution based on ν_0 which bounds the prior sufficiently away from extreme regions (i.e., regions where we run into identifiability issues). In our example a value of $\nu_0 = 10$ leads to a good prior that avoids extreme cases. If we would pick a ν_0 of 6 or 7, most likely we would run into identifiability issues since these priors allow for multiple highly correlated factors (*factor splitting problem*). Picking values in the area of $\nu_0 = 12$–15 would imply low factor correlations. Consequently, the number of factors tends to be lower.

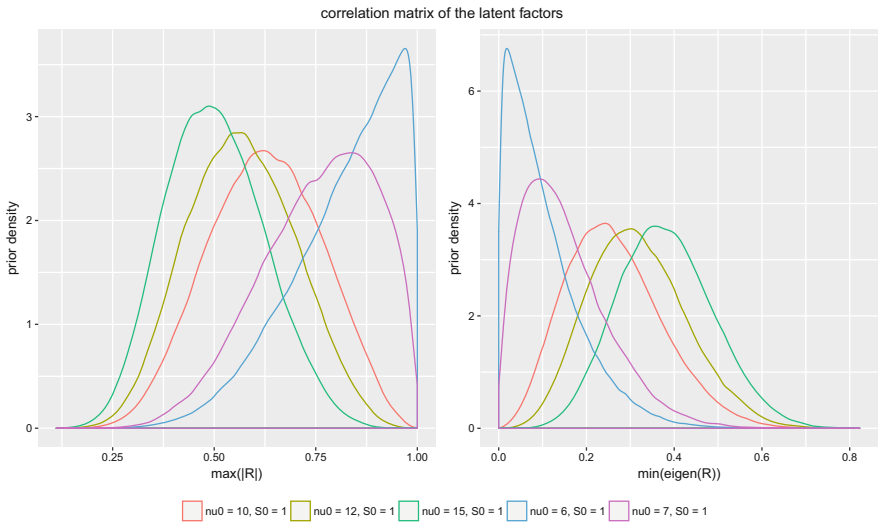

Fig. 2.7 Prior parameterizations for correlation matrix **R**. Left panel: absolute maximum correlation. Right panel: smallest eigenvalue

Next, we need a prior for the number of factors, since they will be estimated by the model as well. BEFA uses a Dirichlet prior with concentration parameter κ. Again, we perform a quick simulation for a fixed sequence of κ-parameters.

```
Ksim <- simul.nfac.prior(nvar = ncol(Privstd), Nid = Nid,
                         Kmax = pmax, kappa = c(.1, .2, .5, 1))
plot(Ksim)
```

Figure 2.8 shows the prior probabilities of the different numbers of factors for different values of κ (all bars of the same color sum up to 1). We pick a value of $\kappa = 0.2$ in order to put more prior weight on a two-factor solution. Note that we could also pick $\kappa = 0.5$ if we want a three-factor solution to be almost equally likely than a two-factor solution. Of course, other values between 0.2 and 0.5 can be examined as well using this simulation function. At the end of the day, it is a matter of Bayesian taste which value we pick.

Next we need a prior for the probability that a manifest variable does not load on any factor τ_0. The package uses a beta prior with shape parameters κ_0 and ξ_0. We set both of them to 0.1, which implies a U-shaped informative prior. That is, we believe that τ_0 is either close to 0 or to 1. The default settings of the `befa` function are $\kappa_0 = \xi_0 = 1$, reflecting a uniform prior. The remaining prior parameters we keep at the uninformative default settings.

Now we are ready to fit the model. We use a burn-in of 5,000 samples and subsequently draw 50,000 MCMC samples. The last two lines post-process the solution in terms of restoring the identification of the model and the consistency of the signs of the factors loadings (see Conti et al., 2014, for details). It is necessary to perform these two operations after fitting the BEFA model; otherwise the solution cannot be interpreted.

```
set.seed(222)
fitbefa <- befa(Privstd, Nid = 2, Kmax = pmax, nu0 = 10,
                kappa = 0.2, kappa0 = 0.1, xi0 = 0.1,
                burnin = 5000, iter = 50000)
fitbefa <- post.column.switch(fitbefa)   ## column reordering
fitbefa <- post.sign.switch(fitbefa)     ## sign switching
sumbefa <- summary(fitbefa)
```

The `summary(fitbefa)` call gives a detailed output of the results. Let us first have a look at the number of factors p proposed by the BEFA fit. For $p = 3$ factors, we get a posterior probability of 0.98. Thus, we have strong evidence that a three-factor solution fits best.

Using the `plot(fitbefa)` call, various plots can be produced. Figure 2.9 shows heatmaps for the loadings (those with highest posterior probabilities) and the

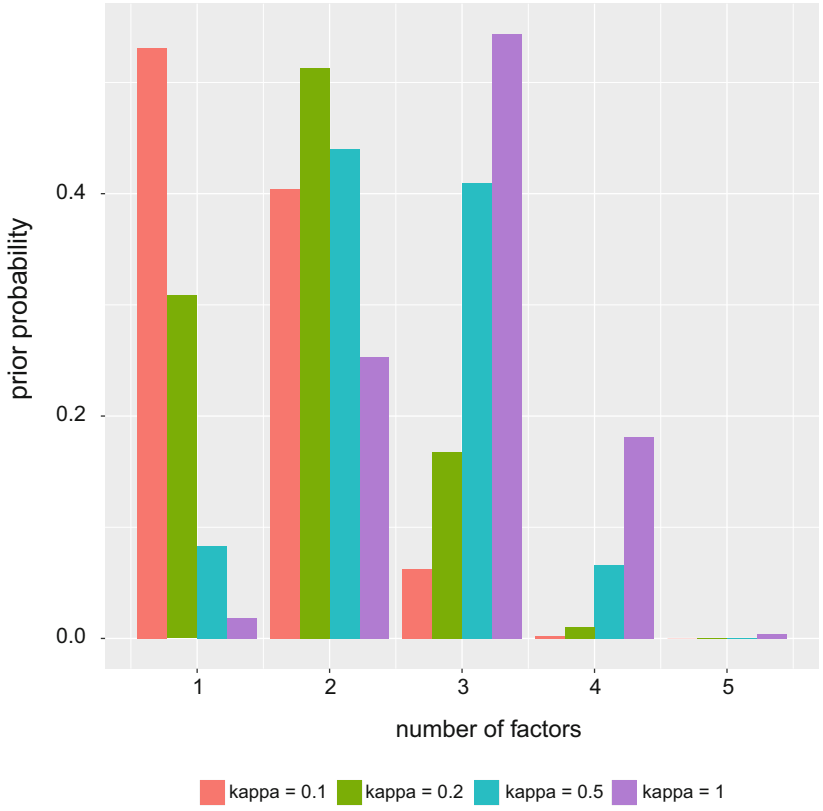

Fig. 2.8 Dirichlet prior parameterizations for number of factors with concentration parameter κ

factor correlations. For the `apc` variables, we get two factors, moderately correlated in positive direction. The `dpc` variables load on a third factor, which has a smaller negative correlation with the remaining two factors.

The **BayesFM** package also provides options for the inclusion of covariates into the factor model. Further details can be found in Conti et al. (2014).

2.4 Confirmatory Factor Analysis

Confirmatory factor analysis (CFA) is conceptually different from EFA. In EFA, the number of factors is determined in an exploratory manner according to the strategies presented in Sect. 2.2.4. In addition, each indicator loads on each factor. A CFA is specified according to an underlying theory concerning the number of factors and a particular loadings structure. Factors are generally allowed to be correlated. CFA is often embedded into a larger path modeling framework called *structural equation modeling* (SEM) which will be introduced in the next chapter.

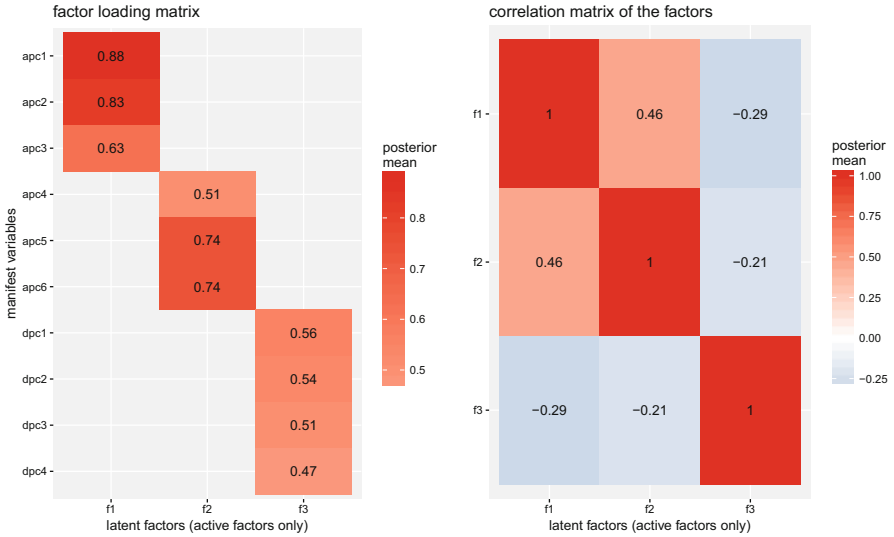

Fig. 2.9 Left panel: Maximum posterior factor loadings. Right panel: Maximum posterior factor correlation matrix

2.4.1 CFA Model Formulation and Computation

We start with the basic CFA model formulation. Let $\mathbf{Y}$ be the $n \times m$ data matrix containing the indicators.[7] Represented as random vector $\mathbf{y}$ of scores on the m variables, the CFA model (*measurement model*) with p factors can be formulated as follows:

$$\mathbf{y} = \boldsymbol{\Lambda}\boldsymbol{\eta} + \boldsymbol{\varepsilon}, \tag{2.13}$$

with $\boldsymbol{\Lambda}$ as the $m \times p$ matrix containing the loadings, $\boldsymbol{\eta}$ as the $p \times 1$ latent variable vector, and $\boldsymbol{\varepsilon}$ as the $m \times 1$ vector of errors associated with the latent variables.

The implied covariance matrix is expressed through the fundamental equation in factor analysis (cf. Eq. (2.4) where we expressed it in terms of the correlation matrix)[8]:

$$\boldsymbol{\Sigma} = \boldsymbol{\Lambda}\boldsymbol{\Psi}\boldsymbol{\Lambda}' + \boldsymbol{\Theta}, \tag{2.14}$$

[7] We switch the notation for the input data ($\mathbf{Y}$ instead of $\mathbf{X}$) in order to be consistent with the standard SEM model formulation presented in the next chapter.

[8] Note that compared to Eq. (2.4) we slightly change the notation (i.e., $\boldsymbol{\Psi}$ instead of $\boldsymbol{\Phi}$, and $\boldsymbol{\Theta}$ instead of $\boldsymbol{\Psi}$) in order to be consistent with the names of the output objects in the **lavaan** package (Rosseel, 2012), which is used throughout this chapter.

with $\boldsymbol{\Psi}$ as the $p \times p$ latent variable covariance matrix and $\boldsymbol{\Theta}$ as the $m \times m$ covariance matrix of the errors. In general we assume multivariate normally distributed errors $\boldsymbol{\varepsilon} \sim N(\mathbf{0}, \boldsymbol{\Theta})$.

EFA and CFA are mathematically very similar, since we have the same fundamental equation in both cases. In EFA we assumed uncorrelated factors by setting $\boldsymbol{\Psi} = \mathbf{I}$. In CFA we do not include such a restriction since we typically allow for correlated factors. By hypothesizing which indicators load on which factor(s), we put restrictions on the loadings matrix $\boldsymbol{\Lambda}$: the elements related to indicators that do not load on a particular factor are set to 0.

Several estimation approaches have been developed in the CFA/SEM literature. Most software implementations use maximum likelihood (ML) by default, which assumes multivariate normality of the manifest variables, an assumption often not fulfilled in practice. In such cases robust ML approaches can be used. A detailed discussion on normality assumptions in CFA/SEM can be found in Finney and DiStefano (2013).

As a first, simple CFA example, we use the R motivation dataset (Mair et al., 2015) from Sect. 2.2.1. We focus on intrinsic and extrinsic motivation items presented to R package developers. We use tetrachoric correlations since the indicators are binary. When we fitted an EFA on these data, each item loaded on each factor. Here we force the intrinsic items (according to an underlying theory on motivation) to load on one factor ("intrinsic motivation") and the extrinsic items to load on a second factor ("extrinsic motivation"). The two factors are allowed to be correlated.

In **lavaan**, in order to use the tetrachoric correlation, we can either define the corresponding variables in the input data frame as ordinal factors or set the `ordered` argument accordingly (see below). In case of mixed scale levels, a corresponding subset of variables can be defined as ordinal. Even though the package is able to handle missing values, we use the full responses only. In **lavaan** model syntax, we specify the CFA model in a regression-like way. We need two equations: one for the extrinsic factor and one for the intrinsic factor. The syntax uses the =~ symbol which can be read as "is manifested by."

```
library("MPsychoR")
library("lavaan")
data("Rmotivation")
vind <- grep("ext|int", colnames(Rmotivation)) ## ext/int items
Rmot <- na.omit(Rmotivation[, vind])
mot_model <- '
  extrinsic  =~ ext1 + ext2 + ext3 + ext4 + ext5 + ext6 +
                ext7 + ext8 + ext9 + ext10 + ext11 + ext12
  intrinsic =~  int1 + int2 + int3 + int4 + int5'
fitMot <- lavaan::cfa(mot_model, data = Rmot,
                      ordered = names(Rmot))
```

The corresponding CFA path diagram (see Fig. 2.10) with unstandardized estimates can be produced using the **semPlot** package (Epskamp, 2015):

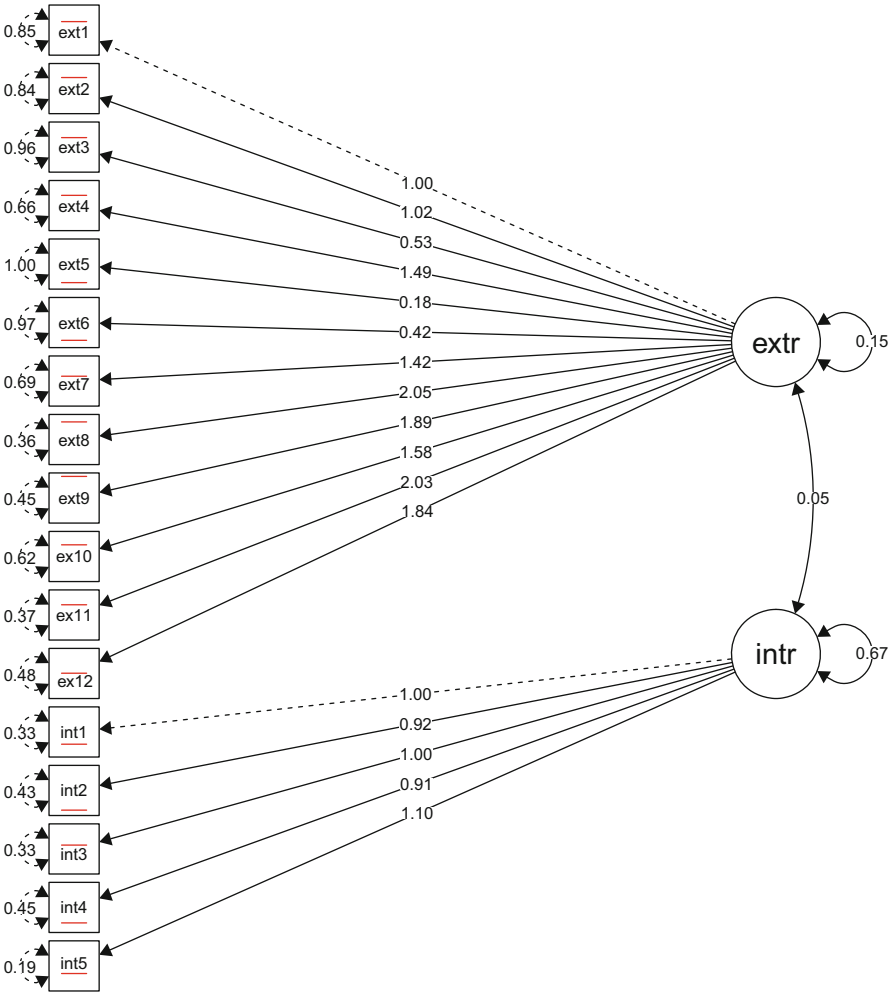

Fig. 2.10 Path diagram with unstandardized parameters of the intrinsic/extrinsic motivation CFA model. Manifest variables are presented as squares with their associated error variances, and latent variables as circles with their associated variances and covariance (edge). The values on the arrows are the loadings. The red lines in the squares denote the thresholds of the tetrachoric correlations

```
library("semPlot")
semPaths(fitMot, what = "est", edge.label.cex = 0.7,
  edge.color = 1, esize = 1, sizeMan = 4.5, asize = 2.5,
  intercepts = FALSE, rotation = 4, thresholdColor = "red",
  mar = c(1, 5, 1.5, 5), fade = FALSE, nCharNodes = 4)
```

For standardized estimates in the diagram, the user can set `what="std"`. The dashed arrows show which loadings were fixed to a value of 1 for identifiability reasons. The manifest variables (indicators) are plotted as squares and the latent variables as circles. Each indicator is associated with an error variance ($\boldsymbol{\Theta}$ in Eq. (2.14)), which can be obtained as follows (due to space restrictions, we omit showing the outputs).

```
inspect(fitMot, what = "est")$theta
```

The loadings ($\boldsymbol{\Lambda}$) and the corresponding standardized versions can be extracted as follows:

```
inspect(fitMot, what = "est")$lambda
inspect(fitMot, what = "std")$lambda
```

The final term in Eq. (2.14) is the latent variable covariance matrix $\boldsymbol{\Psi}$:

```
inspect(fitMot, what = "est")$psi
inspect(fitMot, what = "std")$psi
```

The second call returns the standardized version, that is, the correlation matrix. Explicit tests whether loadings (and other model parameters) differ from 0 including 95% CIs can be requested via:

```
parameterEstimates(fitMot, standardized = TRUE)
```

The significance tests are carried out on the unstandardized parameters. The `standardized=TRUE` argument adds the columns `std.lv` and `std.all`, denoting standardized estimates when only the latent variables are standardized, and standardized estimates when latent and observed variables are standardized, respectively.

To get the full model output including goodness-of-fit measures, we can say:

```
summary(fitMot, standardized = TRUE, fit.measures = TRUE)
```

The standard is to report the following measures (including generally accepted fit rules from Hu and Bentler 1999):

- χ^2-statistic including the degrees of freedom (df) and the p-value: a nonsignificant result suggests that the model fits. However, we do not have to put too much emphasis on this output since for reasonably large samples, the statistic becomes most likely significant anyway, and for small samples it has low power.
- Comparative fit index (CFI): should be ≥ 0.95.
- RMSEA including 90% CI: RMSEA should be ≤ 0.05, upper CI bound ≤ 0.10.
- Standardized root mean square residual (SRMR): should be ≤ 0.08.

Discussions related to these rules as well as additional fit measures can be found in Kline (2016, Chapter 12).

In our example we get a CFI of 0.913, a RMSEA of 0.063 with a corresponding 90% CI of [0.058, 0.069], and an SRMR of 0.119. The χ^2-statistic is 492.422 (df = 118, $p = 0$).

How can we improve the model fit? From the `parameterEstimates()` output, we see that item `ext5` ("R packages are a by-product of my empirical research") has a nonsignificant loading:

```
parameterEstimates(fitMot)[5,]
##         lhs op  rhs   est    se     z pvalue ci.lower ci.upper
## 5 extrinsic =~ ext5 0.181 0.175 1.034  0.301   -0.162    0.523
```

We can eliminate this item and re-fit the model:

```
mot_model2 <- '
  extrinsic  =~ ext1 + ext2 + ext3 + ext4 + ext6 + ext7 +
                ext8 + ext9 + ext10 + ext11 + ext12
  intrinsic =~  int1 + int2 + int3 + int4 + int5'
fitMot2 <- lavaan::cfa(mot_model2, data = Rmot,
                       ordered = names(Rmot)[-5])
```

At this point all loadings are significant. We get a CFI of 0.944, a RMSEA of 0.054 with a corresponding 90% CI of [0.047, 0.06], and an SRMR of 0.109. The model fit has improved.

It is also possible to work with modification indices in order to improve the model fit. The `modindices` function can be used, and the `mi` value tells us how much the χ^2-statistic would improve by freeing fixed parameters. However, the user has to keep in mind that this is an exploratory, data-driven re-specification of the model which contradicts the confirmatory nature of CFA.

The estimated values of the latent variables (*factor scores*) can be extracted via `lavPredict(fitMot2)`. They can be used for further statistical analyses as they are, since they do not suffer from the factor indeterminacy problem as the ones from EFA (see Sect. 2.2.3).

2.4.2 Higher-Order CFA Models

Higher-order CFA models, sometimes also called *hierarchical CFA*, add additional layers of latent variables that are associated with other latent variables in a hierarchical manner. Let us use the R motivation dataset once more. In addition to intrinsic and extrinsic motivation, we consider a hybrid motivation factor as well. In order to keep the path diagram fairly simple, we use the first four items of each subscale only.

```
vind <- c(1:4, 13:16, 32:35)
Rmot2 <- na.omit(Rmotivation[, vind])
```

In the **lavaan** model syntax, we begin with specifying the first-order factor structure involving the indicators. Subsequently, we add one more line that defines the second-order layer in terms of merging extrinsic, hybrid, and intrinsic motivation to a general motivation factor.

```
mot_model3 <- '
  extrinsic  =~ ext1 + ext2 + ext3 + ext4
  hybrid =~ hyb1 + hyb2 + hyb3 + hyb4
  intrinsic =~  int1 + int2 + int3 + int4
  motivation =~ extrinsic + hybrid + intrinsic'
fitMot3 <- lavaan::cfa(mot_model3, data = Rmot2,
                       ordered = names(Rmot2))
```

The corresponding path diagram, this time with standardized parameters, is given in Fig. 2.11. Again, the model output can be investigated by saying:

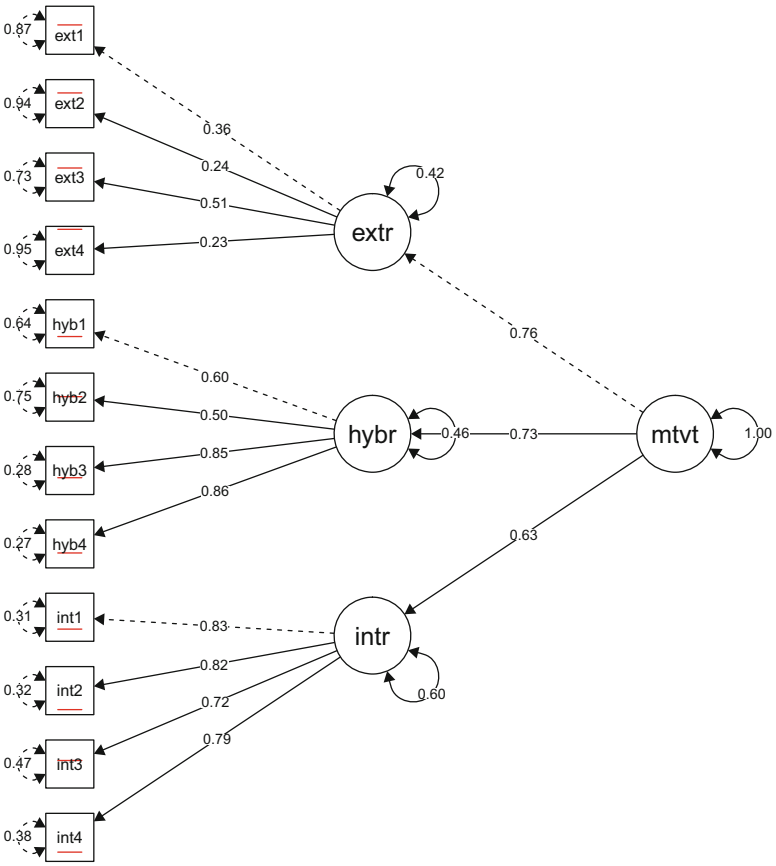

Fig. 2.11 Path diagram of second-order R motivation CFA model (standardized parameters): a general motivation factor causes extrinsic, intrinsic, and hybrid motivation

```
summary(fitMot3, standardized = TRUE, fit.measures = TRUE)
```

The model fits fairly well. We obtain a CFI of 0.955, a RMSEA of 0.052 with a corresponding 90% CI of [0.043, 0.061], and an SRMR of 0.088.

Note that if we would do a second-order specification for the intrinsic/extrinsic CFA model from the section above, **lavaan** would scream. This is due to the fact that we only have two first-order factors loading to the general motivation construct. In that case the model is not identifiable, and we would have to work with restrictions (either fixing both loadings to 1 or restricting the two loadings to be equal in addition to a variance-1 restriction for the general motivation factor). As a general rule, in CFA/SEM we need to have at least two indicators per factor in order to have the

model mathematically identified. But this does not necessarily guarantee that we obtain reasonable estimates. For instance, we could get negative variances. Such weird outcomes are called *Heywood cases*. As Kenny (1979) puts it (cited from Kline, 2016, p. 195): "Two might be fine, three is better, four is best, and anything else is gravy."

2.4.3 CFA with Covariates: MIMIC

MIMIC stands for *multiple indicators multiple independent causes* (Jöreskog and Goldberger, 1975) and is a general structural latent variable concept where CFA is extended in terms of linking covariates with latent variables. MIMIC models can be used to control for sociodemographic or other types of covariates in CFA and more general SEM specifications.

Let us extend the second-order CFA model from above in terms of two covariates: number of R packages (`npkgs`; count variable) and the academic degree (`phd`; dichotomous variable). Compared to the second-order CFA, we need to add one line at the bottom of the model syntax that specifies the regression of the general motivation factor on the covariates. The **lavaan** syntax uses the ~ symbol which means "is regressed on."

```
vind <- c(1:4, 13:16, 32:35, 39:41)
Rmot3 <- na.omit(Rmotivation[, vind])
mot_model4 <- '
  extrinsic  =~ ext1 + ext2 + ext3 + ext4
  hybrid =~ hyb1 + hyb2 + hyb3 + hyb4
  intrinsic =~  int1 + int2 + int3 + int4
  motivation =~ extrinsic + hybrid + intrinsic
  motivation ~ npkgs + phd'
fitMot4 <- lavaan::cfa(mot_model4, data = Rmot3,
                       ordered = names(Rmot3[1:12]))
```

The path diagram is shown in Fig. 2.12. The parameter estimates for the covariates are:

```
parameterEstimates(fitMot4)[16:17,]
##           lhs op   rhs    est    se      z pvalue ci.lower ci.upper
## 16 motivation  ~ npkgs -0.003 0.004 -0.715  0.474    -0.01    0.005
## 17 motivation  ~   phd  0.025 0.028  0.893  0.372    -0.03    0.080
```

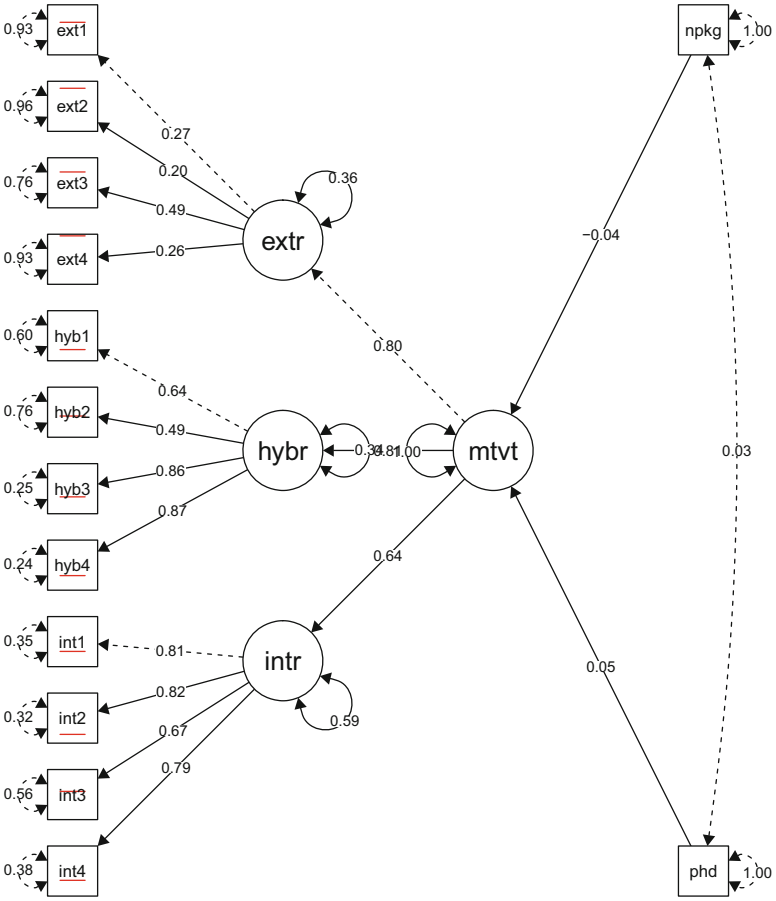

Fig. 2.12 Path diagram of CFA model with covariates (standardized parameters)

We see that these covariates barely have an influence on motivation. Therefore, compared to the model in Fig. 2.11, there were only slight changes in the measurement model parameters.

2.4.4 Multigroup CFA

An important CFA extension is *multigroup CFA* where we are interested in testing parameter differences across groups. One of the key concepts in multigroup CFA are *mean structures* which, simply speaking, extend the t-test/ANOVA idea to latent variables. Including the means of latent variables in a CFA model implies that we need to consider intercepts in the CFA model specification. That is, we extend

Eq. (2.13) in terms of an $m \times 1$ intercept vector $\boldsymbol{\nu}$:

$$\mathbf{y} = \boldsymbol{\nu} + \boldsymbol{\Lambda}\boldsymbol{\eta} + \boldsymbol{\varepsilon}. \tag{2.15}$$

In multiple group settings, we can test hypotheses on group differences by incorporating various restriction patterns on the loadings and the intercepts (*measurement invariance*). The three most important measurement invariance structures are the following:

- *Configural invariance*: unrestricted loadings and intercepts across groups.
- *Weak invariance*: the factor loadings are constrained to be equal across groups; the intercepts are free.
- *Strong invariance*: the factor loadings and indicator intercepts are restricted to be equal across groups.

For each invariance type, we impose the same factor structure on all groups. In addition to these generic invariance structures, specific user-defined loadings/intercepts restrictions can be considered, as we will show below.

In R, the `measurementInvariance` function from **semTools** (semTools Contributors, 2016) can be used to fit the standard invariance sequence from above. We illustrate this approach using a dataset from Bergh et al. (2016). Part of their analysis consisted of a multigroup CFA, where ethnic prejudice (EP), sexism (SP), sexual prejudice against gays and lesbians (HP), and prejudice toward mentally people with disabilities (DP) were modeled as indicators of a generalized prejudice factor (GP). The four manifest variables are composite scores, determined by averaging across the underlying sets of items. The multigroup aspect comes in by exploring differences in CFA parameters across gender. We use a robust ML estimator since some of the prejudice measures deviate from normality.

```
library("semTools")
data("Bergh")
GP_model <- 'GP =~ EP + HP + DP + SP'
minvfit <- measurementInvariance(GP_model, data = Bergh,
              group = "gender", estimator = "MLR")
##
##                Df    AIC    BIC    Chisq Chisq diff Df diff Pr(>Chisq)
## fit.configural  4 7376.1 7490.2   1.6552
## fit.loadings    7 7382.9 7482.8  14.4960     11.372     3   0.009876 **
## fit.intercepts 10 7417.7 7503.3  55.2875     47.707     3  2.459e-10 ***
## fit.means      11 7469.3 7550.2 108.9365     43.818     1  3.603e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This code chunk fits four invariance models: configural, weak, strong, and a fourth model where, in addition to the indicator intercepts, the means of the latent variables are restricted to be equal as well. Each model is tested against the previous one, and a significant result indicates that the higher restricted model is significantly worse (!) than the previous model.

In our example we would go with the configural model, for which the output can be requested as follows:

```
summary(minvfit$fit.configural, standardized = TRUE,
        fit.measures = TRUE)
```

Instead of using these generic invariance sequences of models, we can approach multigroup CFA from a more hypothesis-driven direction. In the first model we are going to fit, the loadings for ethnic minorities (EP), gays/lesbians (HP), and people with disabilities (DP) are held equal across men and women. We keep the sexism (SP) loadings free. This gives the factor a similar meaning for men and women (i.e., as much measurement invariance as possible while testing our hypothesis about one particular loading). Note that all intercepts are free to vary.

In **lavaan**, this can be achieved by using one of the following two equivalent model formulations. In the first variant, we explicitly incorporate the loadings restrictions in the model formulation. We specify a vector of length 2, the first element denoting the loading for the first gender category and the second element denoting the loading for the second category. Since we use the same loadings symbols for both categories, they are restricted to be equal.

```
GP_model <-'GP =~ c(v1,v1)*EP + c(v2,v2)*HP + c(v3,v3)*DP + SP'
fitBase <-lavaan::cfa(GP_model, data = Bergh, group = "gender",
               estimator = "MLR")
```

In the second variant, we restrict all the loadings to be equal across groups using the `group.equal` argument. We then free up the SP loading through the `group.partial` argument.

```
GP_model <- 'GP =~ EP + HP + DP + SP'
fitBase <- lavaan::cfa(GP_model,data = Bergh, group = "gender",
               group.equal = c("loadings"),
               group.partial = c("GP=~ SP"), estimator = "MLR")
```

Both variants lead to the same results. Using the `fitMeasures(fitBase)` call, we get a χ^2-value of 7.321 (df = 6, $p = 0.292$), a RMSEA of 0.023 with a 90% CI of [0, 0.069], a CFI of 0.998, and an SRMR of 0.028. The model fits well.

Since the difference in intercepts between men and women is negligible for the EP indicator, we can force the intercepts to be equal for this indicator. Let us build on the second specification variant from above, in order to set up this model. Through `group.equal` we constrain all loadings and intercepts to be equal across groups

and, subsequently, free up the SP loading as well as the intercepts for DP, HP, and SP using `group.partial`:

```
fitBase1 <- lavaan::cfa(GP_model, data = Bergh,
  group = "gender", group.equal = c("loadings", "intercepts"),
  group.partial = c("GP=~SP", "DP~1", "HP~1", "SP~1"),
  estimator = "MLR")
```

Again, this model fits well. We now use this model as baseline model and compare it to two additional models. First, we constrain the SP loading to be 0 for the women (ingroup-outgroup model). We can specify this restriction directly in the model specification through `c(NA,0)`. `NA` means free to vary across men (first group), whereas the second element fixes the parameter to 0 for the women. Note that this is quite a restrictive assumption.

```
GP_model2 <- 'GP =~ c(v1,v1)*EP + c(v2,v2)*HP + c(v3,v3)*DP +
                    c(NA, 0)*SP'
fitIO <- lavaan::cfa(GP_model2, data = Bergh, group = "gender",
             group.equal = c("intercepts"),
             group.partial = c("DP~1", "HP~1", "SP~1"),
             estimator = "MLR")
```

It gives a χ^2-value of 233.756 (df = 7, $p = 0$), a RMSEA of 0.274 with a 90% CI of [0.245, 0.305], a CFI of 0.678, and an SRMR of 0.148. Bad fit.

In the next model, we restrict all the loadings to be equal (marginalization model). The intercepts have the same constraints as above.

```
fitMarg <- lavaan::cfa(GP_model, data = Bergh,group = "gender",
                group.equal = c("loadings", "intercepts"),
                group.partial = c("DP~1", "HP~1", "SP~1"),
                estimator = "MLR")
```

We get a χ^2-value of 14.496 ($df = 7$, $p = 0.043$), a RMSEA of 0.05 with a 90% CI of [0.008, 0.086], a CFI of 0.989, and an SRMR of 0.036. In terms of goodness-of-fit statistics, we do not see much of a difference compared to the baseline model.

Let us do some model comparison. Since the marginalization model is nested in the base model, we can apply the following χ^2-difference test. Note that a significant result suggests that the higher parametrized model fits significantly worse.

```
anova(fitMarg, fitBase1)
## Scaled Chi Square Difference Test (method = "satorra.bentler.2001")
##
##          Df    AIC    BIC   Chisq Chisq diff Df diff Pr(>Chisq)
## fitBase1  6 7377.7 7482.4  7.3214
## fitMarg   7 7382.9 7482.8 14.4960      5.974       1    0.01452 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Even though significant, the AIC/BIC barely differ across the two groups, and from the fit indices above, we learned that both models fit well. Thus, the marginalization model may be an attractive option to go with.

2.4.5 *Longitudinal CFA*

CFA can also be extended to longitudinal settings, where indicators are presented to the same individual at multiple points in time t (Tisak and Meredith, 1990). Equations (2.13) and (2.15) need to be extended accordingly. Here we extend Eq. (2.15) in order to point out that we can fit combined longitudinal-multigroup CFA models:

$$\mathbf{y}_t = \boldsymbol{\nu}_t + \boldsymbol{\Lambda}_t \boldsymbol{\eta}_t + \boldsymbol{\varepsilon}_t. \tag{2.16}$$

This CFA version extends the repeated measurement ANOVA concept to latent variables, and we are interested in differences over time. Across the time points, we can consider the same measurement invariance principles as in multigroup CFA (configural, weak, and strong invariance). However, in longitudinal CFA we need to account for correlated residuals since we cannot assume that independence holds across time points.

To illustrate longitudinal CFA, we use a dataset on *social dominance orientation* (SDO; Sidanius and Pratto, 2001). SDO is assessed on the same individuals from 1996 to 2000 (Sidanius et al., 2010), involving the following four items, scored on a 7-point scale:

- It is probably a good thing that certain groups are at the top and other groups are at the bottom (`I1`).
- Inferior groups should stay in their place (`I2`).
- We should do what we can to equalize conditions for different groups (`I3`, reversed).
- Increased social equality is beneficial to society (`I4`, reversed).

For the moment, let us consider a simple latent variable structure where all four items load on a general SDO dimension. We pick 2 years only: 1996 vs. 1998. The most relevant model within this context is the strong invariance model, which

allows us to compare the latent means across the two time points. We restrict the factor loadings as well as the corresponding intercepts of each item to be equal in both measurement occasions. Also, we need to allow for residual covariances (using the ~~ symbol in the syntax) for each item across time points.

```
library("MPsychoR")
library("lavaan")
data("SDOwave")
model_sdo1 <- '
  SDO1996 =~ 1*I1.1996 + a2*I2.1996 + a3*I3.1996 + a4*I4.1996
  SDO1998 =~ 1*I1.1998 + a2*I2.1998 + a3*I3.1998 + a4*I4.1998
  SDO1996 ~~ SDO1998

  ## intercepts
  I1.1996 ~ int1*1; I1.1998 ~ int1*1
  I2.1996 ~ int2*1; I2.1998 ~ int2*1
  I3.1996 ~ int3*1; I3.1998 ~ int3*1
  I4.1996 ~ int4*1; I4.1998 ~ int4*1

  ## residual covariances
  I1.1996 ~~ I1.1998
  I2.1996 ~~ I2.1998
  I3.1996 ~~ I3.1998
  I4.1996 ~~ I4.1998

  ## latent means: 1996 as baseline
  SDO1996 ~ 0*1
  SDO1998 ~ 1'
fitsdo1 <- cfa(model_sdo1, data = SDOwave, estimator = "MLR")
```

The parameters of main interest are the latent variable means:

```
parameterEstimates(fitsdo1)[22:23,]
##        lhs op rhs label    est    se      z pvalue ci.lower ci.upper
## 22 SDO1996 ~1            0.000 0.000     NA     NA    0.000    0.000
## 23 SDO1998 ~1           -0.047 0.019 -2.432  0.015   -0.085   -0.009
```

We see that the mean for 1996 was fixed to 0 and is therefore used as the reference year. The second line suggests that there is a significant decrease in SDO from 1996 to 1998.

Note that a weak invariance version of this model can be fitted by restricting both latent means to 0 and freeing up the intercepts. For a configural invariance version, the loadings need to be freed up as well (first two syntax lines). If we do not want to do this by hand, the `longInvariance` function from **semTools** can be used to establish such a generic sequence. Using the `anova` function, the models can be again tested against each other.

By examining the goodness-of-fit statistics of the model fitted above, we see that it does not fit particularly well (e.g., RMSEA = 0.151). This is not too surprising since research has shown that there are two subdimensions of SDO (Ho et al., 2015): anti-egalitarianism and dominance. According to the theory, the first two items are supposed to load on the dominance dimension (SDO-D), whereas the remaining two items load on anti-egalitarianism (SDO-E). Using the second-order CFA concept, we specify a general SDO factor which influences SDO-D and SDO-E. To make the model specification a bit more challenging, we also add a third year (1999 in addition to 1998 and 1996). Again, we restrict the parameters according to the strong invariance principle.

```
model_sdo2 <- '
  ## 1st CFA level, constant loadings across time
  SDOD1996 =~ 1*I1.1996 + d1*I2.1996
  SDOD1998 =~ 1*I1.1998 + d1*I2.1998
  SDOD1999 =~ 1*I1.1999 + d1*I2.1999
  SDOE1996 =~ 1*I3.1996 + a1*I4.1996
  SDOE1998 =~ 1*I3.1998 + a1*I4.1998
  SDOE1999 =~ 1*I3.1999 + a1*I4.1999

  ## 2nd CFA level, constant loadings across time
  SDO1996 =~ 1*SDOD1996 + sd1*SDOE1996
  SDO1998 =~ 1*SDOD1998 + sd1*SDOE1998
  SDO1999 =~ 1*SDOD1999 + sd1*SDOE1999

  ## Constant 1st level intercepts
  I1.1996 ~ iI1*1; I1.1998 ~ iI1*1; I1.1999 ~ iI1*1
  I2.1996 ~ iI2*1; I2.1998 ~ iI2*1; I2.1999 ~ iI2*1
  I3.1996 ~ iI3*1; I3.1998 ~ iI3*1; I3.1999 ~ iI3*1
  I4.1996 ~ iI4*1; I4.1998 ~ iI4*1; I4.1999 ~ iI4*1

  ## residual covariances:
  I1.1999 ~~ I1.1998; I1.1996 ~~ I1.1998; I1.1999 ~~ I1.1996
  I2.1999 ~~ I2.1998; I2.1996 ~~ I2.1998; I2.1999 ~~ I2.1996
  I3.1999 ~~ I3.1998; I3.1996 ~~ I3.1998; I3.1999 ~~ I3.1996
  I4.1999 ~~ I4.1998; I4.1996 ~~ I4.1998; I4.1999 ~~ I4.1996

  ## latent means
  SDO1996 ~ 0*1    ## 1996 baseline year
  SDO1998 ~ 1      ## 1998 vs. 1996
  SDO1999 ~ 1      ## 1999 vs. 1996
'
fitsdo2 <- cfa(model_sdo2, data = SDOwave, estimator = "MLR")
```

Note that in this model we have two indicators per construct only but we did not get any weird estimates (Heywood cases). The path diagram is given in Fig. 2.13. We plot the unstandardized estimates such that the imposed parameter restrictions are nicely reflected.

The goodness-of-fit measures suggest that the model fits fairly well (e.g., RMSEA = 0.061). Let us extract the latent means reflecting changes in relation to the baseline year 1996:

```
parameterEstimates(fitsdo2)[43:45,]
##        lhs op rhs label    est    se      z pvalue ci.lower ci.upper
## 43 SDO1996 ~1           0.000 0.000     NA     NA    0.000    0.000
## 44 SDO1998 ~1          -0.057 0.024 -2.331   0.02   -0.104   -0.009
## 45 SDO1999 ~1          -0.033 0.025 -1.311   0.19   -0.082    0.016
```

We see that, as above, we have a significant change from 1996 to 1998, but no significant change from 1996 to 1999 (again, 1996 is used as baseline year).

We can now think of several extensions of longitudinal CFA. First, we can combine longitudinal CFA with multigroup models (*multigroup-longitudinal CFA*). Second, instead of estimating covariances between the latent variables across time, we can specify directed paths which allow us to study trends in a more sophisticated way (instead of ANOVA-type comparisons). Such models are called *CFA panel models*. Details on these extensions can be found in Little (2013). Another alternative to study longitudinal developments of latent variables are *latent growth models* which we introduce in Sect. 3.4 within a larger SEM context.

2.4.6 Multilevel CFA

Recent **lavaan** developments (v0.6-0 and higher) extend the syntax and estimation engine to *multilevel CFA* modeling. Multilevel models are a general modeling framework where individuals are nested within aggregate units (see, e.g., Hox, 2010). To illustrate multilevel CFA, we use a dataset from Hox (2010) involving six intelligence measures: word list, cards, matrices, figures, animals, and occupations. We have a two-level data structure: children (first level) are nested *within* families (second level). The model we are going to present here involves a two-factor solution at an individual level ("within"; factors are "numeric" and "perception") and a one-factor solution at a family level ("within"; g-factor).

In **lavaan** syntax we need to specify the factor structure for each level separately (see `level`). The `cluster` argument denotes the clustering variable.[9]

[9] At the time this book was written, **lavaan** allows for two-level structures only. Also, thanks to Yves Rosseel for sharing the code.

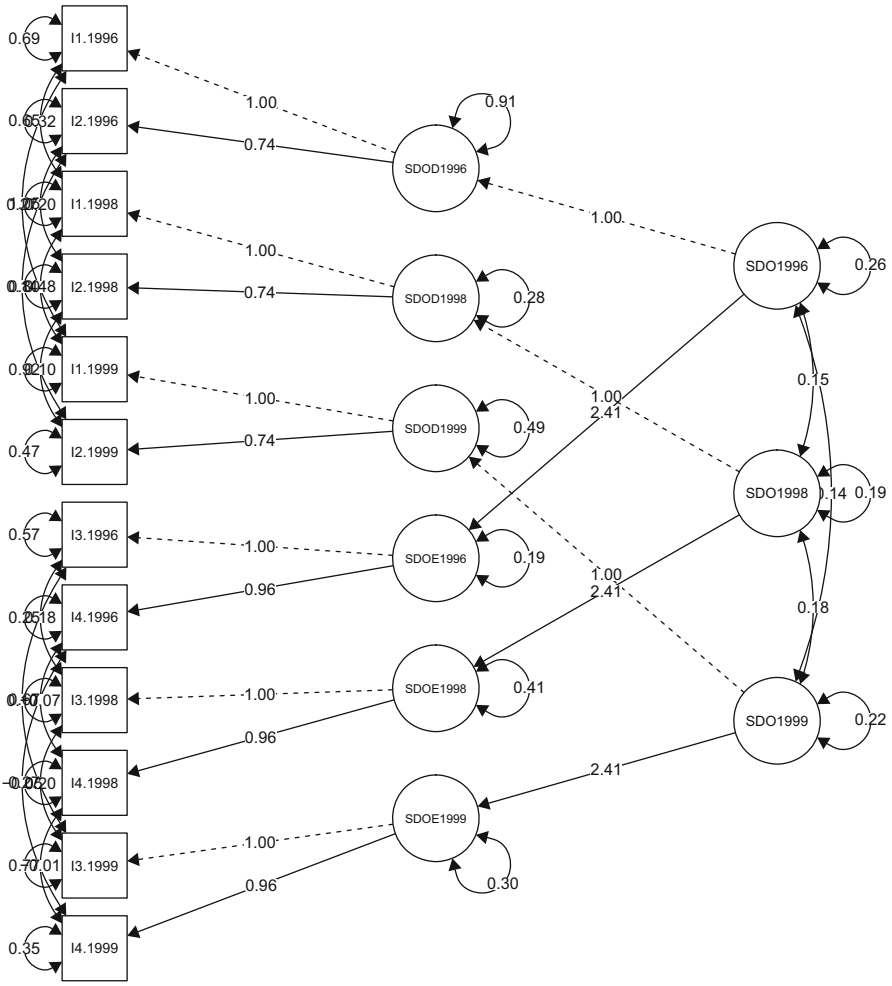

Fig. 2.13 Path diagram for longitudinal CFA model on SDO data (unstandardized parameters). The parameters are restricted according to the strong invariance principle

```
data("FamilyIQ")
modelIQ <- '
 level: 1
  numeric =~ wordlist + cards + matrices
  perception =~ figures + animals + occupation
 level: 2
  general =~ wordlist + cards +  matrices + figures + animals +
             occupation'
```

(continued)

```
fitIQ <- cfa(modelIQ, data = FamilyIQ, cluster = "family",
             std.lv = TRUE)
fitIQ
## lavaan (0.6-1.1179) converged normally after  64 iterations
##
##   Number of observations                           399
##   Number of clusters [family]                       60
##
##   Estimator                                         ML
##   Model Fit Test Statistic                      11.927
##   Degrees of freedom                                17
##   P-value (Chi-square)                           0.805
```

Unfortunately, the classical goodness-of-fit indices (CFI, RMSEA, etc.) are not sensitive to level-2 misspecifications (Hsu et al., 2015) and therefore only of limited use. This leaves us with the χ^2-statistic which, in our example, tells us that the model fits. Hox (2010) interprets the fact that we need two factors at an individual level in the tradition of Cattell's theory of fluid and crystallized intelligence: As a result of influences at an individual level (social/physical environment, education), the g-factor crystallizes into specific individual competencies. At the family level, where shared genetic and environmental influences play a role, a single g-factor is sufficient.

2.5 Bayesian Confirmatory Factor Analysis

Thanks to the development of the **blavaan** package (Merkle and Rosseel, 2018), Bayesian versions of CFA models can be fitted easily. Basically, all we have to do is to replace the `cfa` call by `bcfa` and we are all set; the syntax is exactly the same. However, we have the opportunity to specify additional MCMC-related arguments. In terms of priors, the default setting is to use uninformative priors for $\boldsymbol{\Lambda}$, $\boldsymbol{\Theta}$, and $\boldsymbol{\Psi}$ (cf. Eq. (2.14)).

```
library("blavaan")
dpriors()[c("lambda", "itheta", "ipsi")]
##          lambda          itheta            ipsi
## "dnorm(0,1e-2)"  "dgamma(1,.5)"  "dgamma(1,.5)"
```

The user can specify his/her own more informative priors, if desired. When fitting a model using one of the **blavaan** functions, the package establishes a JAGS file internally, does the MCMC sampling using **runjags** (Denwood, 2016), and organizes the results such that they look as **lavaan**-like as possible. As always in Bayesian analysis, we aim for a low autocorrelation of the samples.

The number of MCMC iterations needs to be considerably high to get reasonably low autocorrelation patterns. Thus, some patience is required when estimating the model.

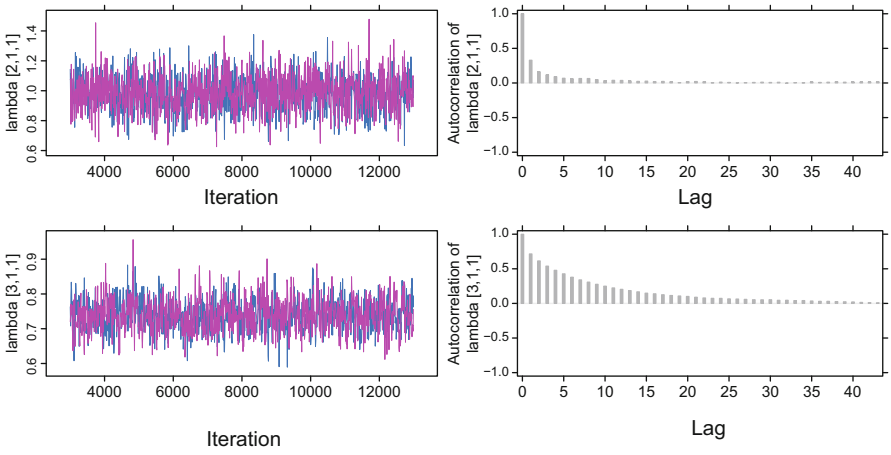

Fig. 2.14 Left panel: MCMC trace plots for the first two loading parameters. Right panel: autocorrelation plots for the first two loading parameters

To illustrate how to use the package, we use the data from Bergh et al. (2016) once more and fit a simple one-factor CFA model. This is similar to what we did in Sect. 2.4.4, but here we ignore the multigroup aspect. We set the number burn-in samples to 2,000, we draw 10,000 MCMC samples, and we run 2 chains.

```
library("MPsychoR")
data("Bergh")
GP_model <- 'GP =~ EP + HP + DP + SP'
set.seed(123)
fitBCFA <- bcfa(GP_model, data = Bergh, burnin = 2000,
                sample = 10000, n.chains = 2,
                jagcontrol = list(method = "rjparallel"))
```

Before we look at the posterior distributions of the parameters, let us check whether we did a good MCMC job. Autocorrelation and trace plots can be produced as follows (we show the first two loading parameters only; see Fig. 2.14):

```
plot(fitBCFA, pars = 1:2, plot.type = "trace")
plot(fitBCFA, pars = 1:2, plot.type = "autocorr")
```

The two MCMC chains seem to be fairly well mixed. We see that especially for the second parameter, there is still some autocorrelation present. Increasing the number MCMC samples may lead to an improvement. A full model output can be requested as follows:

```
summary(fitBCFA)
##
##   Number of observations                           861
##
##   Number of missing patterns                         1
##
##   Statistic                                 MargLogLik         PPP
##   Value                                      -3803.682       0.577
##
## Parameter Estimates:
##
##
## Latent Variables:
##                    Estimate  Post.SD  HPD.025  HPD.975     PSRF
##   GP =~
##     EP                1.000
##     HP                0.985    0.119    0.762    1.228    1.000
##     DP                0.739    0.045    0.651    0.828    1.000
##     SP                0.946    0.058    0.837    1.064    1.000
##
## Intercepts:
##                    Estimate  Post.SD  HPD.025  HPD.975     PSRF
##    .EP                1.990    0.024    1.941    2.036    1.001
##    .HP                1.217    0.053    1.111    1.318    1.000
##    .DP                2.059    0.018    2.022    2.094    1.001
##    .SP                2.115    0.023     2.07     2.16    1.001
##     GP                0.000
##
## Variances:
##                    Estimate  Post.SD  HPD.025  HPD.975     PSRF
##    .EP                0.232    0.018    0.197    0.266    1.000
##    .HP                2.175    0.108     1.97    2.391    1.001
##    .DP                0.138    0.010    0.118    0.157    1.000
##    .SP                0.226    0.016    0.195     0.26    1.000
##     GP                0.274    0.026    0.224    0.325    1.001
```

The first important output is the *Bayesian posterior predictive p-value* (PPP) which uses the difference between observed χ^2-values and χ^2-values based on internally replicated data. A value close to 0.5 suggests that the model fits well, which is the case in our example. For each (freely estimated) parameter, we get the posterior mean, the posterior standard deviation, and a 95% *highest posterior density* (HPD) interval. The *potential scale reduction factor* (PSRF) is another convergence diagnostic. Values should be smaller than 1.2.

Additional model information can be extracted via `blavInspect`. Having multiple competing CFA models, a Bayesian model comparison can be carried out using the `blavCompare` function. Details on these statistics, estimation, as well as additional modeling options can be found in Merkle and Rosseel (2018).

References

Bartholomew, D. J., & Knott, M. (1999). *Latent variable models and factor analysis* (2nd ed.). London: Hodder Arnold.

Bartholomew, D. J., Steele, F., Moustaki, I., & Galbraith, J. I. (2008). *Analysis of multivariate social science data* (2nd ed.). Boca Raton: CRC Press.

Bergh, R., Akrami, N., Sidanius, J., & Sibley, C. (2016). Is group membership necessary for understanding prejudice? A re-evaluation of generalized prejudice and its personality correlates. *Journal of Personality and Social Psychology, 111*, 367–395.

Browne, M. W. (2001). An overview of analytic rotation in exploratory factor analysis. *Multivariate Behavioral Research, 36*, 111–150.

Browne, M. W., & Cudeck, R. (1993). Alternative ways of assessing model fit. In: K. A. Bollen & J. S. Long (Eds.), *Testing structural equation models* (pp. 136–162). Beverly Hills: Sage.

Cattell, R. B. (1966). The scree test for the number of factors. *Multivariate Behavioral Research, 1*, 245–276.

Conti, G., Frühwirth-Schnatter, S., Heckman, J. J., & Piatek, R. (2014). Bayesian exploratory factor analysis. *Journal of Econometrics, 183*, 31–57.

Denwood, M. J. (2016). **runjags**: An R package providing interface utilities, model templates, parallel computing methods and additional distributions for MCMC models in JAGS. *Journal of Statistical Software, 71*(9), 1–25. https://www.jstatsoft.org/article/view/v071i09

Drasgow, F. (1986). Polychoric and polyserial correlations. In: S. Kotz & N. L. Johnson (Eds.), *Encyclopedia of statistical sciences* (Vol. 7, pp. 68–74). New York: Wiley.

Epskamp, S. (2015). **semPlot**: Unified visualizations of structural equation models. *Structural Equation Modeling: A Multidisciplinary Journal, 22*, 474–483.

Finney, S. J., & DiStefano, C. (2013). Nonnormal and categorical data in structural equation modeling. In: G. R. Hancock & R. O. Mueller (Eds.), *Structural equation modeling: A second course* (2nd ed., pp. 439–492). Charlotte: Information Age Publishing.

Hendrickson, A. E., & White, P. O. (1964). PROMAX: A quick method for rotation to oblique simple structure. *British Journal of Mathematical and Statistical Psychology, 17*, 65–70.

Ho, A. K., Sidanius, J., Kteily, N., Sheehy-Skeffington, J., Pratto, F., Henkel, K. E., Foels, R., & Stewart, A. L. (2015). The nature of social dominance orientation: Theorizing and measuring preferences for intergroup inequality using the new SDO_7 scale. *Journal of Personality and Social Psychology, 109*, 1003–1028.

Horn, J. L. (1965). A rationale and test for the number of factors in factor analysis. *Psychometrika, 30*, 179–185.

Hox, J. J. (2010). *Multilevel analysis: Techniques and applications* (2nd ed.). New York: Routledge.

Hsu, H. Y., Kwok, O. M., Lin, J. H., & Acosta, S. (2015). Detecting misspecified multilevel structural equation models with common fit indices: A Monte Carlo study. *Multivariate Behavioral Research, 50*, 197–215.

Hu, L., & Bentler, P. M. (1999). Cutoff criteria for fit indexes in covariance structure analysis: Conventional criteria versus new alternatives. *Structural Equation Modeling, 6*, 1–55.

Jöreskog, K. G., & Goldberger, A. S. (1975). Estimation of a model with multiple indicators and multiple causes of a single latent variable. *Journal of the American Statistical Association, 70*, 631–639.

Kaiser, H. F. (1958). The varimax criterion for analytic rotation in factor analysis. *Psychometrika, 23*, 187–200.

Kaiser, H. F. (1960). The application of electronic computers to factor analysis. *Educational and Psychological Measurement, 20*, 141–151.

Kaplan, D. (2014). *Bayesian statistics for the social sciences*. New York: Guilford.

Kenny, D. A. (1979). *Correlation and causality*. New York: Wiley.

Kirk, D. B. (1973). On the numerical approximation of the bivariate normal (tetrachoric) correlation coefficient. *Psychometrika, 38*, 259–268.

Kline, R. B. (2016). *Principles and practice of structural equation modeling* (4th ed.). New York: Guilford Press.

Kruschke, J. (2014). *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan* (2nd ed.). Cambridge: Academic.

Little, T. D. (2013). *Longitudinal structural equation modeling*. New York: Guilford.

MacCallum, R. C. (2009). Factor analysis. In: R. E. Millsap, & A. Maydeu-Olivares (Eds.), *The Sage handbook of quantitative methods in psychology* (pp. 123–177) London: Sage.

Mair, P., Hofmann, E., Gruber, K., Zeileis, A., & Hornik, K. (2015). Motivation, values, and work design as drivers of participation in the R open source project for statistical computing. *Proceedings of the National Academy of Sciences of the United States of America 112*, 14788–14792.

McDonald, R., & Mulaik, S. A. (1979). Determinacy of common factors: A nontechnical review. *Psychological Bulletin, 86*, 430–445.

Merkle, E. C., & Rosseel, Y. (2018). **blavaan**: Bayesian structural equation models via parameter expansion. *Journal of Statistical Software*, *85*(4), 1–30.

Muthén, B. O., & Hofacker, C. (1988). Testing the assumptions underlying tetrachoric correlations. *Psychometrika, 83*, 563–578.

Piatek, R. (2017). **BayesFM**: Bayesian inference for factor modeling. R package version 0.1.2. https://CRAN.R-project.org/package=BayesFM

Raiche, G., & Magis, D. (2011). **nFactors:**An R package for parallel analysis and non graphical solutions to the Cattell scree test. R package version 2.3.3. http://CRAN.R-project.org/package=nFactors

Revelle, W. (2015). An introduction to psychometric theory with applications in R. Freely available online. http://www.personality-project.org/r/book/

Revelle, W. (2017). **psych**: Procedures for psychological, psychometric, and personality research. R package version 1.7.8. http://CRAN.R-project.org/package=psych

Revelle, W., & Rocklin, T. (1979). Very simple structure: An alternative procedure for estimating the optimal number of interpretable factors. *Multivariate Behavioral Research, 14*, 403–414.

Rhemtulla, M., Brosseau-Liard, P. E., & Savalei, V. (2012). When can categorical variables be treated as continuous? A comparison of robust continuous and categorical SEM estimation methods under suboptimal conditions. *Psychological Methods, 17*, 354–373.

Rosseel, Y. (2012). **lavaan**: An R package for structural equation modeling. *Journal of Statistical Software, 48*(2), 1–36. http://www.jstatsoft.org/v48/i02/

Savalei, V. (2011). What to do about zero frequency cells when estimating polychoric correlations. *Structural Equation Modeling: A Multidisciplinary Journal, 18*, 253–273.

semTools Contributors. (2016). **semTools**: Useful tools for structural equation modeling. R package version 0.4-14. https://CRAN.R-project.org/package=semTools

Sidanius, J., & Pratto, F. (2001). *Social dominance: An intergroup theory of social hierarchy and oppression*. Cambridge: Cambridge University Press.

Sidanius, J., Levin, S., van Laar, C., & Sears, D. O. (2010). *The diversity challenge: Social identity and intergroup relations on the college campus*. New York: The Russell Sage Foundation.

Tisak, J., & Meredith, W. (1990). Longitudinal factor analysis. In: A. von Eye (Ed.), *Statistical methods in longitudinal research* (Vol. 1, pp. 125–149). San Diego: Academic.

Treiblmaier, H. (2006). Datenqualität und individualisierte Kommunikation [Data Quality and Individualized Communication]. Wiesbaden: DUV Gabler Edition Wissenschaft.

Treiblmaier, H., Bentler, P. M., & Mair, P. (2011). Formative constructs implemented via common factors. *Structural Equation Modeling: A Multidisciplinary Journal 18*, 1–17.

Tucker, L. R., & Lewis, C. (1973). A reliability coefficient for maximum likelihood factor analysis. *Psychometrika, 38*, 1–10.

Vaughn-Coaxum, R., Mair, P., & Weisz, J. R. (2016) Racial/ethnic differences in youth depression indicators: An item response theory analysis of symptoms reported by White, Black, Asian, and Latino youths. *Clinical Psychological Science 4*, 239–253.

Velicer, W. F. (1976). Determining the number of components from the matrix of partial correlations. *Psychometrika, 41*, 321–327.

Wei, T., & Simko, V. (2016) **corrplot**: Visualization of a correlation matrix. R package version 0.77. https://CRAN.R-project.org/package=corrplot

Chapter 3
Path Analysis and Structural Equation Models

3.1 Multivariate Regression as Path Model

The simplest path model we can think of is a *multiple linear regression* with m predictors and a single response Y. It can be fully estimated through a path model framework, but there is not really a statistical reason to do so. Therefore, in this section we start our elaborations with *multivariate regression* path models. Note that multivariate regression should not be confounded with multiple regression. In multivariate regression we have, in general, multiple predictors and multiple, potentially correlated responses. Let us collect these response variables in an $n \times q$ matrix $\mathbf{Y}$ with n as the sample size and q as the number of response variables. The multivariate regression model can be written as

$$\mathbf{Y} = \mathbf{XB} + \mathbf{E}. \tag{3.1}$$

The design matrix $\mathbf{X}$ is of dimension $n \times (m+1)$ (since we have a unit vector for the intercept), the parameter matrix $\mathbf{B}$ is correspondingly $(m+1) \times q$, and the matrix of error terms $\mathbf{E}$ is $n \times q$.

There are several ways of computing multivariate regression models: One approach is to stay within a classical linear model context, estimate a multiple regression model for each response separately, and then compute various measures known from MANOVA (multivariate analysis of variance), subject to inference. A second approach is to fit the model through a path-analytic framework. A third approach is to use mixed-effects models and specify a random effect for the response variables. In this section we focus on the first two approaches: classical MANOVA approach and path modeling. Details regarding the third approach can be found in Berridge and Crouchley (2011).

Let us start with MANOVA. We use the generalized prejudice dataset from Bergh et al. (2016), already presented in Sect. 2.4.4. This time we consider two indicators from the agreeableness scale (`A1`, `A2`) and two indicators from the openness scale

P. Mair, *Modern Psychometrics with R*, Use R!,
https://doi.org/10.1007/978-3-319-93177-7_3

(O1, O2) as predictors, as well as two responses related to prejudices: ethnic prejudice (EP) and prejudice toward mentally people with disabilities (DP). All variables represent composite scores, based on averaging underlying items. Note that there are no latent variables involved. The two regression models look as follows ($i = 1, \ldots, n$):

$$Y_{i1} = \beta_{10} + \beta_{11}X_{i1} + \beta_{12}X_{i2} + \beta_{13}X_{i3} + \beta_{14}X_{i4} + \varepsilon_{i1},$$
$$Y_{i2} = \beta_{20} + \beta_{21}X_{i1} + \beta_{21}X_{i2} + \beta_{21}X_{i3} + \beta_{21}X_{i4} + \varepsilon_{i2}.$$

Let us fit these two multiple regression models using cbind on the left-hand side of the R formula specification in the lm call.

```
library("MPsychoR")
data("Bergh")
fitmvreg <- lm(cbind(EP, DP) ~ A1 + A2 + O1 + O2, data = Bergh)
```

Note that we would get the same results with two separate lm calls. The multivariate aspect comes in by evoking the Manova function from the **car** package (Fox and Weisberg, 2010):

```
library("car")
Manova(fitmvreg)
##
## Type II MANOVA Tests: Pillai test statistic
##    Df test stat approx F num Df den Df    Pr(>F)
## A1  1  0.001205    0.516      2    855    0.5971
## A2  1  0.103297   49.247      2    855 < 2.2e-16 ***
## O1  1  0.023774   10.411      2    855 3.411e-05 ***
## O2  1  0.026217   11.510      2    855 1.168e-05 ***
## ---
```

By default, it prints out *Pillai's trace*. Other popular measures such as Wilks' lambda and Roy's largest eigenvalue can be obtained using summary. These measures are based on a decomposition of the *total sum-of-squares cross-products matrix* into regression and residual sum-of-squares matrices. From the output we see that all variables, except the first agreeableness item A1, have a significant influence on both prejudice responses jointly. This is the statement we can make from this

output; additional MANOVA illustrations can be found in an Appendix chapter of Fox and Weisberg (2010).[1]

Let us now consider a second strategy for multivariate regression by specifying a path model using **lavaan** (Rosseel, 2012) which, as we will see, gives us a more detailed insight into how each predictor influences each response variable. This approach uses the same idea as factor analysis, meaning that it aims to fit a model covariance matrix which is as close as possible to the empirical covariance matrix. Since only observed variables are involved in the model, we use the ~ symbol ("is regressed on") in the syntax.

```
library("lavaan")
mvreg.model <- '
  EP ~ b11*A1 + b12*A2 + b13*O1 + b14*O2
  DP ~ b21*A1 + b22*A2 + b23*O1 + b24*O2'
fitmvreg2 <- sem(mvreg.model, data = Bergh)
```

The corresponding path representation using the **semPlot** package (Epskamp, 2015) can be produced as follows (see Fig. 3.1):

```
library("semPlot")
semPaths(fitmvreg2, what = "est", edge.label.cex = 1,
  layout = "tree", residuals = FALSE, edge.color = 1,
  esize = 1, rotation = 3, sizeMan = 8, asize = 2.5,
  fade = FALSE, optimizeLatRes = TRUE)
```

Let us have a look at some parameters of interest:

```
parameterEstimates(fitmvreg2)[c(1:8, 11),]
##    lhs op rhs label    est    se      z pvalue ci.lower ci.upper
## 1   EP  ~  A1   b11 -0.005 0.058 -0.079  0.937   -0.118    0.109
## 2   EP  ~  A2   b12 -0.455 0.064 -7.133  0.000   -0.580   -0.330
## 3   EP  ~  O1   b13 -0.259 0.059 -4.430  0.000   -0.374   -0.145
## 4   EP  ~  O2   b14 -0.248 0.061 -4.061  0.000   -0.368   -0.128
## 5   DP  ~  A1   b21 -0.041 0.043 -0.971  0.332   -0.125    0.042
## 6   DP  ~  A2   b22 -0.428 0.047 -9.132  0.000   -0.519   -0.336
## 7   DP  ~  O1   b23 -0.118 0.043 -2.748  0.006   -0.202   -0.034
## 8   DP  ~  O2   b24 -0.176 0.045 -3.930  0.000   -0.264   -0.088
## 11  EP ~~  DP        0.106 0.010 10.382  0.000    0.086    0.127
```

[1] See https://socserv.socsci.mcmaster.ca/jfox/Books/Companion/appendix/Appendix-Multivariate-Linear-Models.pdf

For each predictor we get regression coefficients on each response while taking into account that the response variables are correlated. We see that `A1` does not have a significant influence on neither `EP` nor `DP`; all other predictors are significant. The estimate for the covariance between the two responses is given in the last line.

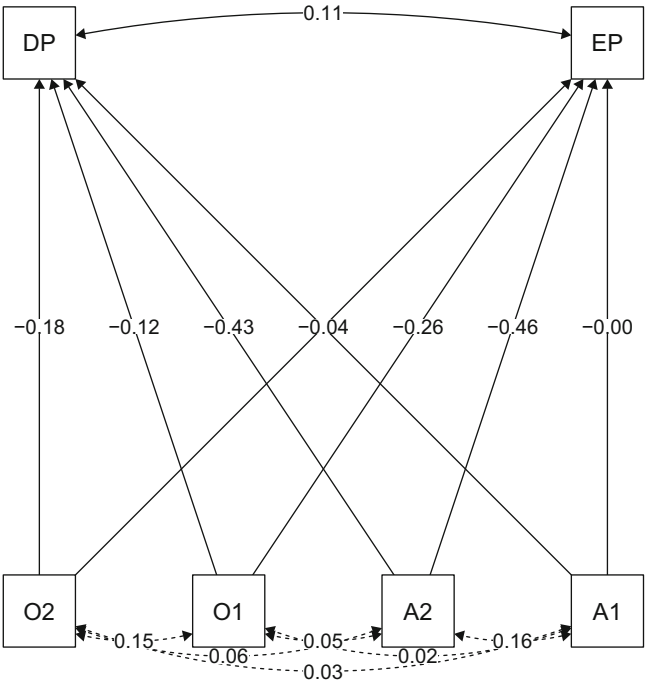

Fig. 3.1 Multivariate regression as path model (unstandardized parameters). Agreeableness/openness predictors: A1, A2, O1, O2. Prejudice responses: DP, EP

3.2 Moderator and Mediator Models

Moderator and mediator models are regression models with more complex dependency structures and can be estimated through a path-analytic framework. Baron and Kenny (1986) give the following definitions of moderators and mediators: "A *moderator* is a qualitative (e.g., sex, race, class) or quantitative (e.g., level of reward) variable that affects the direction and/or strength of the relation between an independent variable and a dependent variable" (p. 1174). "A given variable may be said to function as a *mediator* to the extent that it accounts for the relation between the independent variable and the dependent variable. Mediators explain how external physical events take on internal psychological significance" (p. 1176).

We see that in both cases we have a predictor X and a response Y. The relationship between X and Y is influenced by a third variable (moderator Z or mediator M).

- A moderator interacts directly with X (e.g., X = alcohol consumption, Z = social context, Y = social acceptance).
- A mediator acts "in between" X and Y (e.g., X = alcohol consumption, M = driving back home, Y = waking up in jail).

In the following sections, we elaborate on various moderator, mediator, and simple combined moderation-mediation strategies. An excellent, applied textbook on this topic is Hayes (2013) which illustrates many additional flavors of this modeling framework.

3.2.1 Moderator Models

Statistically speaking, moderation is expressed as interaction: in order to account for the moderation effect of Z in a regression model, we need to allow for an interaction between the predictors X and Z. The corresponding regression model (with the usual assumptions) looks as follows[2]:

$$Y_i = \text{int} + a_1 X_i + a_2 Z_i + a_3 X_i Z_i + \varepsilon_i . \tag{3.2}$$

The crucial point in moderator analysis is to look at the interaction: if a_3 is significant, we say that Z has a moderating effect on the relationship $X \to Y$.

Note that centering for multicollinearity purposes has been debunked as myth and is therefore not necessary (see, e.g., Kromrey and Foster-Johnson, 1998; Dalal and Zickar, 2012; Hayes, 2013). However, in terms of interpretation, centering Z and/or X can be helpful since parameters can be interpreted relative to the mean levels of the other variables. If X or Z are categorical, the interpretation of the effects depends on the coding scheme used in the model. Note that centering does not affect the model fit.

Let X^* and Z^* denote the centered versions of X and Z. Equation (3.2) can be rewritten as

$$Y_i = \text{int} + a_1 X_i^* + a_2 Z_i^* + a_3 X_i^* Z_i^* + \varepsilon_i , \tag{3.3}$$

To illustrate a simple moderator model, let us consider the following example. Paškvan et al. (2016) present a model which explores the effects of work intensification on various outcomes. Here we focus on the effect of work intensification X on

[2]We use a slightly nonstandard notation for the regression parameters in order to make it consistent with the symbols used in the moderator/mediator literature.

cognitive appraisal Y (i.e., how an individual views a situation) and explore whether participative climate Z (i.e., provision of information, participation in decision-making) has a moderating effect on this relationship. Figure 3.2 depicts the basic moderator relationship. Note that this is a simple conceptual representation of the moderator setup and does not represent a statistical path model.

Before we fit the moderator model, let us check the effects of the centered predictors X^* and Z^* on Y by means of two simple regressions.

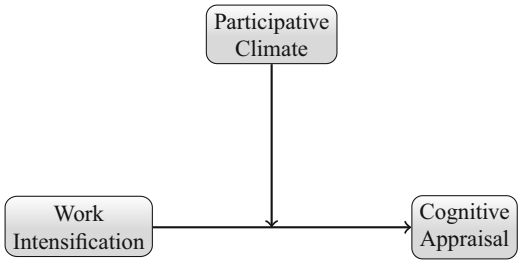

Fig. 3.2 Moderating effect of participative climate on the relationship between work intensification and cognitive appraisal (conceptual representation)

```
library("MPsychoR")
data("Paskvan")
wintense.c <- scale(Paskvan$wintense, scale = FALSE)  ## center
fit.YX <- lm(cogapp ~ wintense.c, data = Paskvan)  ## Y on X
round(summary(fit.YX)$coefficients, 4)
##             Estimate Std. Error  t value Pr(>|t|)
## (Intercept)   3.7265     0.0227 164.4470        0
## wintense.c    0.5458     0.0237  22.9954        0
pclimate.c <- scale(Paskvan$pclimate, scale = FALSE)  ## center
fit.YZ <- lm(cogapp ~ pclimate.c, data = Paskvan)  ## Y on Z
round(summary(fit.YZ)$coefficients, 4)
##             Estimate Std. Error  t value Pr(>|t|)
## (Intercept)   3.7265     0.0272 136.8341        0
## pclimate.c   -0.3324     0.0304 -10.9408        0
```

Both variables have a significant effect on Y, and the R^2 values are 0.398 and 0.13, respectively.

At this point we have two options: either use the `lm` function and fit the model according to Eq. (3.3) or we use the `moderate.lm` function from the **QuantPsyc** package (Fletcher, 2012) which, by default, centers X and Z. Let us explore the latter option.

```
library("QuantPsyc")
fit.mod <- moderate.lm(x = wintense, z = pclimate, y = cogapp,
                       data = Paskvan)
round(summary(fit.mod)$coefficients, 4)
##             Estimate Std. Error  t value Pr(>|t|)
## (Intercept)   3.7097     0.0227 163.0954   0.0000
## mcx           0.5003     0.0241  20.7653   0.0000
## mcz          -0.1790     0.0257  -6.9765   0.0000
## mcx:mcz      -0.0663     0.0236  -2.8094   0.0051
```

We see that the interaction between the centered work intensification (`mcx`) and the centered participative climate (`mcz`) is significant. Thus, there is evidence that participative climate moderates the relationship between work intensification and cognitive appraisal. We get an $R^2 = 0.436$, which is slightly larger than the one in first regression model without moderator. As mentioned above, due to centering parameters can be interpreted relative to the mean levels of the other variables. For instance, a_1 denotes the change in cognitive appraisal Y by a 1-unit change in work intensification X for individuals with average participate climate Z (as opposed to $Z = 0$ in the uncentered case).

In order to get a more detailed insight into the interaction, we can perform a *simple slope analysis*. In case of a metric moderator, this strategy splits the moderator into a set of levels and reports the regression parameters for each of these levels. By default, the `sim.slopes` function categorizes the centered moderator Z^* as follows: "low" for observations smaller than $-sd(Z^*)$, "high" for observations larger than $sd(Z^*)$, and "medium" for the observations within the $\pm sd(Z^*)$ range.

```
fit.ss <- sim.slopes(fit.mod, Paskvan$pclimate)
round(fit.ss, 4)
##             INT  Slope     SE    LCL    UCL
## at zHigh 3.5492 0.4407 0.0313 0.3793 0.5022
## at zMean 3.7097 0.5003 0.0241 0.4530 0.5475
## at zLow  3.8703 0.5598 0.0328 0.4953 0.6242
```

A simple slopes plot can be produced using the `graph.mod` function. We see that none of the confidence intervals (CIs) for the slope parameters includes 0. Thus, moderation happens at each level of participative climate.

As we have seen, from a statistical perspective, a moderator model is nothing else than a regression model with an interaction. This model can be extended in several directions. First, we can include additional covariates into the model. Second, we can have scenarios where the predictor and/or the moderator are categorical (see Baron and Kenny, 1986; Aiken and West, 1991). Third, we can extend this concept to arbitrary generalized linear model (GLM) settings such as, for instance,

a moderated logistic regression in case of a binary response. Fourth, moderator models can also be fitted within a repeated measurement design context as shown in Judd et al. (2001).

3.2.2 Mediator Models

The simplest mediator model involves a predictor X, a response Y, and a mediator M which influences the effect of X on Y but in a different way than in moderator models. Figure 3.3 shows such a simple mediation relationship using the dataset from above. Here we focus on the relationship between work intensification X and emotional exhaustion Y, with cognitive appraisal acting as mediator M between the two.

Readers who have some familiarity with mediator models may have heard about the *causal steps approach* to mediation laid out by Baron and Kenny (1986). For many years this has been the standard strategy to assess *partial* or *full mediation*. In recent years, there has been a growing consensus in the community to disregard this strategy since it is not based on a quantification of an indirect effect of X on Y (through M). Another reason for abandoning the distinction between partial and full mediation is that they are too sample size dependent. Details can be found in Hayes (2013, Chapter 6).

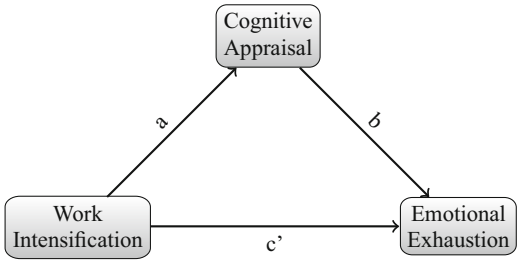

Fig. 3.3 Relationship between work intensification and emotional exhaustion mediated by cognitive appraisal

A simple mediator model consists of the following set of regression equations:

$$\begin{aligned} M_i &= \text{int}_1 + aX_i + \varepsilon_{i1}, \\ Y_i &= \text{int}_2 + c'X_i + bM_i + \varepsilon_{i2}. \end{aligned} \tag{3.4}$$

We use c' for the effect of X on Y, a for the effect of X on M, and b for the effect of M on Y. The intercepts are denoted by int_1 and int_2.

The *indirect effect* of X on Y (through M) is simply ab. This indirect effect is of key importance since it determines the strength of mediation. It has been found (Preacher and Hayes, 2004) that the sampling distribution of ab deviates from normality (as assumed by the classical *Sobel test*). The gold standard nowadays is

to perform a bootstrap and look at the corresponding bootstrap confidence interval (CI) for assessing significance.

In R we have at least two options to compute mediator models including bootstrapped CIs. We can either use the **mediation** package (Tingley et al., 2014), or we specify a mediator path model in **lavaan**. Below we show both strategies using the work intensification dataset (missings eliminated[3]). Let us start with the **mediation** package. First we need to compute the following two regressions according to Eq. (3.4):

```
library("mediation")
fit.MX <- lm(cogapp ~ wintense, data = Paskvan)
fit.YXM <- lm(emotion ~ wintense + cogapp, data = Paskvan)
```

Now we feed these objects into the `mediate` function. Through the `treat` argument, we specify X, and through the `mediator` argument M. The remaining arguments set up the bootstrap.

```
set.seed(123)
fitmed <- mediation::mediate(fit.MX, fit.YXM,
  treat = "wintense", mediator = "cogapp",
  sims = 999, boot = TRUE, boot.ci.type = "bca")
summary(fitmed)
##
## Causal Mediation Analysis
##
## Nonparametric Bootstrap Confidence Intervals with the BCa Method
##
##                Estimate 95% CI Lower 95% CI Upper p-value
## ACME              0.334        0.263         0.42  <2e-16 ***
## ADE               0.249        0.138         0.35  <2e-16 ***
## Total Effect      0.583        0.503         0.66  <2e-16 ***
## Prop. Mediated    0.572        0.438         0.73  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Sample Size Used: 803
##
##
## Simulations: 999
```

In this output the ACME (*average causal mediation effect*) represents the indirect effect ab, including the 95% bootstrap CI. Since ACME = 0 (null hypothesis) is not contained in the CI, we conclude that cognitive appraisal M mediates the relationship between work intensification X and emotional exhaustion Y.

ADE stands for *average direct effect* which, in our model, is the coefficient c'. The *total effect* is simply $ab + c'$. The final line reports the proportion of

[3]The packages **mediation** and **lavaan** treat missing values differently.

the mediated effect which is the indirect effect divided by the total effect. In this example 57.6% of the total effect of work intensification (X) on emotional exhaustion (Y) is mediated by cognitive appraisal (M). This measure can be used as effect size; additional effect sizes are outlined in Hayes (2013, Chapter 6) and can be easily computed from the `mediate` output objects.

Now let us fit the same model using **lavaan**. In the syntax, we first specify the two regressions,[4] then the indirect effect, and, if needed, the total effect and the mediated proportion as well. For each parameter we get a 95% bootstrap CI.

```
library("lavaan")
med.model <- '
  emotion ~ c*wintense + b*cogapp
  cogapp ~ a*wintense
  ind := a*b
  tot := ind+c
  prop := ind/tot'
set.seed(123)
fitmedsem <- lavaan::sem(med.model, Paskvan, se = "bootstrap",
                         bootstrap = 999)
parameterEstimates(fitmedsem, zstat = FALSE, pvalue = FALSE,
  boot.ci.type = "bca.simple")[c(7,1,8,9),]
##       lhs op      rhs label   est    se ci.lower ci.upper
## 7     ind :=      a*b   ind 0.334 0.038    0.262    0.414
## 1 emotion  ~ wintense     c 0.249 0.057    0.136    0.359
## 8     tot :=    ind+c   tot 0.583 0.043    0.504    0.676
## 9    prop :=  ind/tot  prop 0.572 0.079    0.431    0.740
```

The last line extracts the effects in the same order as in the `mediate` output: indirect effect ab, direct effect c', the total effect, and the mediation proportion. For all of them, we get bootstrapped CIs. These outcomes match the results from the **mediation** package, apart from slight deviations due to the stochastic nature of the bootstrap. The model can be visualized using `semPaths(fitmedsem)` from **semPlot** which results in Fig. 3.3.

Which package should we use? The **mediation** package provides a large amount of modeling possibilities. First, multiple mediator models (see Hayes, 2013, Chapter 5) can be fitted easily. Second, the underlying regression specifications can be of any GLM type (e.g., logistic mediator regression if Y is binary, Poisson mediator regression in case of counts). In addition, ordinal/multinomial mediator models, mixed-effects mediator models, as well as nonlinear spline regressions can be computed. Tingley et al. (2014) illustrates how to fit such extensions. The advantage of **lavaan** is that it allows to specify more complex path structures (including options for visualization via **semPlot**). In the next section, we will look at such a structure.

[4] Note that in the syntax we use the symbol `c` instead of c' as in Eq. (3.4).

3.2.3 Combined Moderator-Mediator Models

At this point we can think of combining moderator and mediator strategies into a single model. Simple setups are often called *moderated mediation* or *mediated moderation* (see Muller et al., 2005, for details on this distinction). Here we use the more general term *condition process analysis*, where the goal is to model the conditional mechanisms by which a variable transmits its effects on another (Hayes, 2013).

A simple example is given in Fig. 3.4, where we combine the moderator and mediator models from above into a single model. The indirect effect of work intensification (X) on emotional exhaustion (Y) through cognitive appraisal (mediator M) is contingent on participative climate (Z). The latter moderates the relationship between X and M and represents a *conditional indirect effect* where the magnitude of the effect depends on the levels of the moderator. As pointed out in the moderation section, variables can be centered for interpretability reasons. The same applies to moderator relations in conditional process analysis. In the example below, we will not center in order to accentuate that doing so is not necessary.

The set of regression equations in this example is a combination of Eqs. (3.2) and (3.4):

$$\begin{aligned} M_i &= \text{int}_1 + a_1 X_i + a_2 Z_i + a_3 X_i Z_i + \varepsilon_{i1}, \\ Y_i &= \text{int}_2 + c' X_i + b M_i + \varepsilon_{i2}. \end{aligned} \tag{3.5}$$

The conditional indirect effect is $(a_1 + a_3 Z_i)b$. This implies that the indirect effect needs to be evaluated for different values of Z. In case of categorical moderators, we can use corresponding factor levels of interest. For metric moderators, as in our example, we can use quantiles or any other moderator value of interest. For

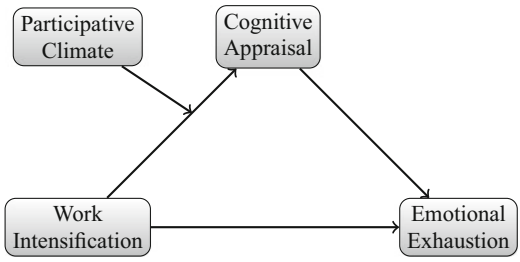

Fig. 3.4 A simple conditional process model (moderated mediation) for work intensification, participative climate, cognitive appraisal, and emotional exhaustion (conceptual diagram)

more complex models, more than two equations may be required. Examples of such structures are presented in Hayes (2013, Chapter 10), including the corresponding regression equations and expressions for the conditional indirect effects.

The **mediation** package is able to fit some moderator-mediator structures. However, we cannot fit the structure given in Fig. 3.4 unless we would include the moderator in both model parts (think of an arrow going from participative climate to $X \rightarrow Y$). In this case the second equation changes accordingly in order to account for XZ interaction. The package also includes a test (see `test.modmed`) on pairwise differences of indirect effects between two moderator strata. Details can be found in Tingley et al. (2014).

Here we use the **lavaan** package and evaluate the conditional indirect effect at the quartiles. In the **lavaan** syntax, we first specify the regressions according to Eq. (3.5). Subsequently, the conditional indirect effects are computed for given quartiles of participative climate. Note that we could also include equations for direct and total effects. As above, we bootstrap the CIs.

```
quantile(Paskvan$pclimate)
##   0%  25%  50%  75% 100%
##  1.0  2.0  3.0  3.5  5.0
medmod.model <- '
## set of regressions
cogapp ~ a1*wintense + a2*pclimate + a3*wintense:pclimate
emotion ~ c*wintense + b*cogapp

## conditional indirect effects
cie.q1 := (a1 + a3*2)*b      ## first quartile
cie.q2 := (a1 + a3*3)*b      ## median
cie.q3 := (a1 + a3*3.5)*b    ## third quartile
'
set.seed(123)
fitmedmod <- lavaan::sem(medmod.model, data = Paskvan,
                 se = "bootstrap", bootstrap = 999)
```

The corresponding path model can be plotted as follows using the **semPlot** package (see Fig. 3.5):

```
semPaths(fitmedmod, layout = "spring", asize = 2.5,
  sizeMan = 10, residuals = FALSE, nCharNodes = 7,
  edge.label.cex = 1)
```

Now we extract the moderator interaction, the direct effect, and the conditional indirect effects, evaluated at the three quartiles.

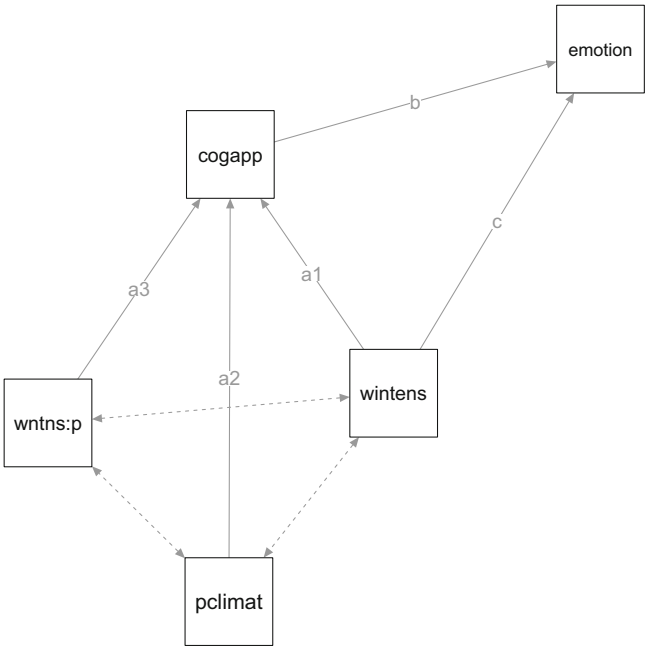

Fig. 3.5 Moderated mediation path model for work intensification (X), emotional exhaustion (Y), cognitive appraisal (M), and participative climate (Z). Node names are abbreviated (`wntns:p` denotes the interaction between X and Z)

```
parameterEstimates(fitmedmod, zstat = FALSE, pvalue = FALSE,
  boot.ci.type = "bca.simple")[c(3, 4, 14:16),]
##        lhs op               rhs  label    est    se ci.lower ci.upper
## 3   cogapp  ~ wintense:pclimate     a3 -0.066 0.030   -0.130   -0.009
## 4  emotion  ~          wintense      c  0.249 0.057    0.136    0.359
## 14  cie.q1 :=       (a1+a3*2)*b cie.q1  0.338 0.034    0.280    0.417
## 15  cie.q2 :=       (a1+a3*3)*b cie.q2  0.297 0.036    0.231    0.373
## 16  cie.q3 :=     (a1+a3*3.5)*b cie.q3  0.277 0.040    0.201    0.360
```

The interaction parameter a_3 is significant (note that the upper CI bound is close to 0) which suggests that participative climate moderates the relationship between work intensification and cognitive appraisal. The direct effect of work intensification on emotional exhaustion (c' in Eq. (3.5)) is significant as well. All three indirect effects are significant, meaning that for given levels of participative climate, cognitive appraisal mediates the relationship between work intensification and emotional exhaustion. This mediated relationship is weaker for employees working in a favorable participative climate (i.e., the effect is getting smaller for higher quantiles).

Covariates such as age and gender can be included as well by simply adding them to both regression equations. For this dataset this extension is presented in Paškvan et al. (2016).[5]

3.3 Structural Equation Models

3.3.1 SEM Model Formulation and Computation

Structural equation models (SEM) integrate *confirmatory factor analysis* (CFA) into a larger path analytic framework. Formally, we extend the basic CFA expression (*measurement model*) as given in Eq. (2.13), or, more generally, the multigroup CFA expression in Eq. (2.15), by an additional linear specification reflecting dependencies among the latent variables (*structural model*). This leads to the following system of equations, involving m observed variables (indicators) and p latent variables[6]:

$$\mathbf{y} = \boldsymbol{\nu} + \boldsymbol{\Lambda}\boldsymbol{\eta} + \boldsymbol{\varepsilon} \tag{3.6}$$

$$\boldsymbol{\eta} = \boldsymbol{\alpha} + \boldsymbol{B}\boldsymbol{\eta} + \boldsymbol{\zeta}. \tag{3.7}$$

As in CFA, $\mathbf{y}$ is the random vector containing the m observed variables, $\boldsymbol{\nu}$ the $m \times 1$ intercept vector, $\boldsymbol{\eta}$ the $p \times 1$ latent variable vector, $\boldsymbol{\Lambda}$ the $m \times p$ matrix containing the loadings, and $\boldsymbol{\varepsilon}$ the $m \times 1$ vector of errors associated with the latent variables. In the second equation, $\boldsymbol{\alpha}$ is the $p \times 1$ latent variable intercept vector, $\boldsymbol{B}$ the $p \times p$ matrix of directed path coefficients, and $\boldsymbol{\zeta}$ a $p \times 1$ vector of errors associated with the latent variables. The conventional assumptions for the errors are $\boldsymbol{\varepsilon} \sim N(\mathbf{0}, \boldsymbol{\Theta})$ and $\boldsymbol{\zeta} \sim N(\mathbf{0}, \boldsymbol{\Psi})$, which can be relaxed by corresponding robust estimation approaches.

To demonstrate how to fit a simple SEM using **lavaan**, we use again the prejudice dataset from Bergh et al. (2016). This time we consider three latent variables (factors): The first two factors (i.e., agreeableness and openness) pertain to personality traits, each of them is measured by three indicators. The third latent variable is a generalized prejudice (GP) factor which is associated with ethnic prejudice (EP), sexism (SP), sexual prejudice against gays and lesbians (HP), and prejudice toward mentally people with disabilities (DP). The structural model connects these three measurement models specifying paths from both agreeableness and openness to GP. The syntax is as follows:

[5]Note that the authors obtained slightly different results since, apart from including covariates, they had special missing value treatments and did centering.

[6]This notation is called "LISREL all-y" notation and is used by **lavaan** internally. There are several other options for SEM formulation (see Bollen, 1989; Kline, 2016).

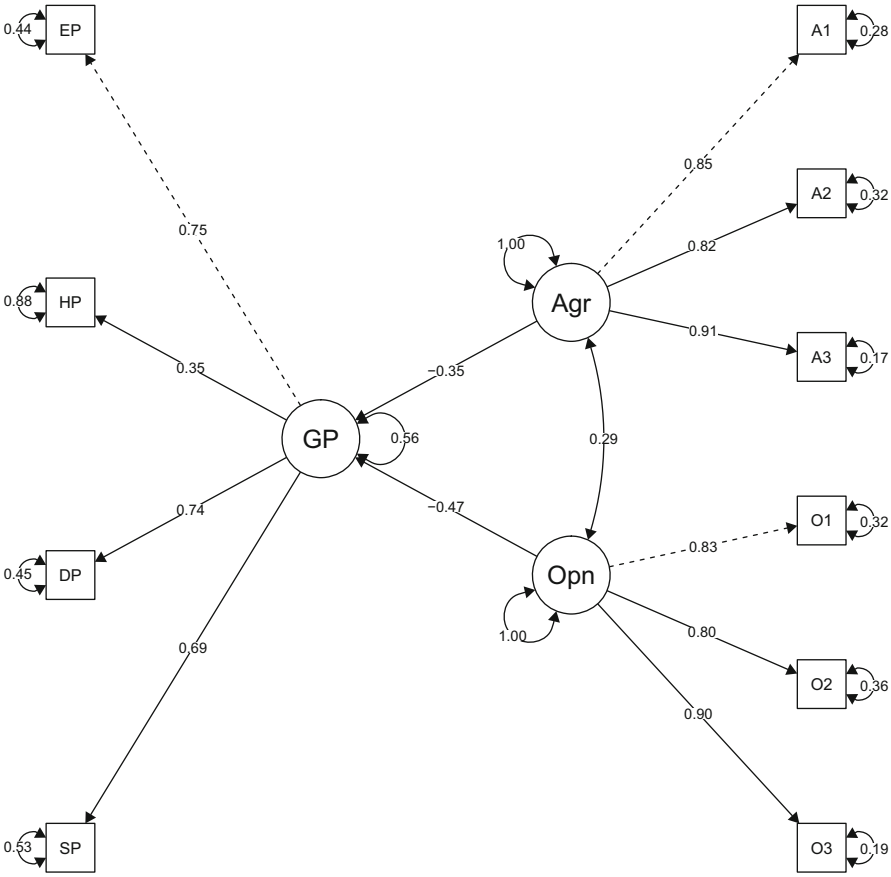

Fig. 3.6 Diagram of the generalized prejudice SEM (standardized parameters)

```
library("MPsychoR")
library("lavaan")
data("Bergh")
Bergh.model <- 'GP =~ EP + HP + DP + SP
                Agree =~ A1 + A2 + A3
                Open =~ O1 + O2 + O3
                GP ~ Agree + Open'
fitGP <- sem(Bergh.model, data = Bergh, estimator = "MLR")
```

The path diagram with standardized parameters can be produced as follows and is given in Fig. 3.6.

```
semPaths(fitGP, what = "std", edge.label.cex = 0.7, esize = 1,
  intercepts = FALSE,rotation = 4, edge.color = 1, asize = 2.5,
  sizeMan = 5, mar = c(1, 1.5, 1.5, 3), fade = FALSE)
```

We get a CFI of 0.963, a RMSEA of 0.075 with a corresponding 90% CI of [0.065,0.085], and an SRMR of 0.054. The χ^2-statistic is 186.62 (df = 32, $p = 0$). Note that here the same rules of thumb apply as in CFA (see Sect. 2.4.1). The model fits fairly well.

A full model summary including standardized parameters and fit measures can be obtained as follows:

```
summary(fitGP, standardized = TRUE, fit.measures = TRUE)
```

Here, we only show the relevant structural model parameters since the measurement model parameters can be interpreted as in CFA.

```
## Regressions:
##                   Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
##   GP ~
##     Agree           -0.435    0.056   -7.773    0.000   -0.350   -0.350
##     Open            -0.626    0.057  -11.000    0.000   -0.473   -0.473
##
## Covariances:
##                   Estimate  Std.Err  z-value  P(>|z|)   Std.lv  Std.all
##   Agree ~~
##     Open             0.049    0.007    6.761    0.000    0.286    0.286
```

From the regressions we see that both agreeableness and openness have a significant influence on GP in negative direction. The correlation between agreeableness and openness is 0.286 (see `Std.lv` in covariances output).

Admittedly, this was a very simple example; structural relations can be way more complicated. In practice, it is suggested to build up a complex SEM step-by-step. First, get the measurement models right and make sure that they fit. Reliability indices as presented in Sect. 1.2 are helpful within this context. After that, establish the structural relations step-by-step and examine the goodness of fit for each submodel. At the end, modification indices can be considered for additional fit improvement. However, as mentioned in Sect. 2.4, they have to be used with caution since they are purely data-driven and the corresponding model re-specification might be difficult to justify from a substantive point of view.

3.3.2 Multigroup SEM

In terms of multigroup models, we already did all the hard work in Sect. 2.4.4. There is nothing really new here except that the constraints can be extended to structural model parameters. Let us continue with the generalized prejudice SEM from Fig. 3.6 and examine potential parameter differences across gender. We are going to fit a series of models which build on the multigroup CFA strategy we used in Sect. 2.4.4. We start with a first model M_1 where the loadings and the structural regression parameters are free to vary across gender. We force the EP intercept to be equal for men and women by first constraining all intercepts to be equal through `group.partial` and subsequently freeing up this restriction for DP, HP, and SP through the `group.partial` specification.

```
fit.free <- sem(Bergh.model, group = "gender",
                group.equal = c("intercepts"),
                group.partial = c("DP~1", "HP~1", "SP~1"),
                data = Bergh, estimator = "MLR")
```

This model gives a RMSEA of 0.077 with a 95% CI of [0.067,0.088], a CFI of 0.958, an SRMR of 0.054, and a χ^2-value of 241.266 (df = 68, $p = 0$).

In the second model M_2, we use the same intercept restrictions as in M_1. In addition, we constrain the loadings to be equal, except the SP loading on GP.

```
fit.load <- sem(Bergh.model, group = "gender",
      group.equal = c("loadings", "intercepts"),
      group.partial = c("GP=~SP", "DP~1", "HP~1", "SP~1"),
      data = Bergh, estimator = "MLR")
```

This model gives a RMSEA of 0.076 with a 95% CI of [0.066,0.086], a CFI of 0.956, an SRMR of 0.06, and a χ^2-value of 256.084 (df = 74, $p = 0$). Both models computed so far fit moderately well.

As a third model M_3, again, we use the same intercept restrictions as above and free up all loadings. This time, we restrict all structural path coefficients to be equal through `"regressions"` in `group.equal`.

```
fit.prestrict <- sem(Bergh.model, group = "gender",
      group.equal = c("intercepts", "regressions"),
      group.partial = c("DP~1", "HP~1", "SP~1"),
      data = Bergh, estimator = "MLR")
```

For this model we obtain a RMSEA of 0.076 with a 95% CI of [0.065,0.086], a CFI of 0.958, an SRMR of 0.055, and a χ^2-value of 242.295 (df = 70, $p = 0$).

Let us now perform some model comparison. M_3 is nested within the mode general model M_1. That is, all the parameters from M_3 are contained in M_1 since M_3 simply specifies additional parameter restrictions. For such nested model comparisons, we can apply the usual χ^2-difference test:

```
anova(fit.free, fit.prestrict)
## Scaled Chi Square Difference Test (method = "satorra.bentler.2001")
##
##               Df   AIC   BIC  Chisq Chisq diff Df diff Pr(>Chisq)
## fit.free      68 11308 11603 241.27
## fit.prestrict 70 11305 11590 242.29    0.89287       2     0.6399
```

We see that AIC and BIC are of similar magnitude across both models, and the p-value tells us that M_3 is not significantly worse than M_1.

If we want to test M_2 against M_3, we cannot apply `anova` since the two models are not nested: neither of the two is a restricted version of the other one. In such cases we have the following options. Either we just look at the AIC/BIC and pick the one with the (substantially) smallest AIC/BIC, or we apply a variant of the χ^2-difference test for non-nested models according to Vuong (1989). The **nonnest2** package (Merkle et al., 2016) provides a corresponding implementation which we are going to illustrate.

The `icci` function computes the AIC and BIC including CI for AIC/BIC differences. Vuong's test statistic can be computed using `vuongtest`. Both functions require the models to be fitted via maximum likelihood (ML); the `update` calls below refit the models accordingly. Note that this idea can be applied to CFA as well.

```
library("nonnest2")
fit.load1 <- update(fit.load, estimator = "ML")
fit.prestrict1 <- update(fit.prestrict, estimator = "ML")
compIC <- icci(fit.load1, fit.prestrict1)
compIC
```

```
##
## Model 1
##  Class: lavaan
##  Call: lavaan::lavaan(model = Bergh.model, data = Bergh,  ...
##  AIC: 11310.695
##  BIC: 11577.148
##
## Model 2
##  Class: lavaan
##  Call: lavaan::lavaan(model = Bergh.model, data = Bergh,  ...
```

(continued)

```
##  AIC: 11304.905
##  BIC: 11590.390
##
## 95% Confidence Interval of AIC difference (AICdiff = AIC1 - AIC2)
##   -10.544 < AICdiff < 22.123
##
## 95% Confidence Interval of BIC difference (BICdiff = BIC1 - BIC2)
##   -29.576 < BICdiff < 3.091
```

We see that both the AIC and the BIC differences CIs contain 0 which implies that the model fits are sufficiently close that neither can be preferred over the other.

Vuong's non-nested testing strategy gives the following results:

```
vuongtest(fit.load1, fit.prestrict1)
##
## Model 1
##  Class: lavaan
##  Call: lavaan::lavaan(model = Bergh.model, data = Bergh, ...
##
## Model 2
##  Class: lavaan
##  Call: lavaan::lavaan(model = Bergh.model, data = Bergh, ...
##
## Variance test
##   H0: Model 1 and Model 2 are indistinguishable
##   H1: Model 1 and Model 2 are distinguishable
##     w2 = 0.020,   p = 0.0533
##
## Non-nested likelihood ratio test
##   H0: Model fits are equal for the focal population
##   H1A: Model 1 fits better than Model 2
##     z = -1.655,   p = 0.951
##   H1B: Model 2 fits better than Model 1
##     z = -1.655,   p = 0.04899
```

First, we look at the variance test output which gives a nonsignificant result. This confirms what we have detected using AIC/BIC: the models are indistinguishable, and there is no need to proceed with the likelihood ratio (LR) test further below in the output. If the variance test was significant, we can compare the models via the non-nested LR test (Merkle et al., 2016).

3.3.3 Remarks on SEM Extensions

Before covering a somewhat special SEM topic in the next section (i.e., latent growth models), we give a few remarks on various SEM extensions and related path

modeling approaches. In Sect. 2.4.6 we illustrated multilevel CFA. Multilevel SEM models can be specified in an analogous way using **lavaan**. To date, the package allows for two-level specification only. Bayesian CFA was introduced in Sect. 2.5 using the `bcfa` function from **blavaan** (Merkle and Rosseel, 2017). In order to fit a Bayesian SEM, the `bsem` function from the same package can be used. The model syntax is the same as in **lavaan**.

Sample size planning in SEM is described in detail in Beaujean (2014, Chapter 8). The author shows how to perform corresponding Monte Carlo simulations in R. The **simsem** package (Pornprasertmanit et al., 2016) facilitates such sample size simulations as well as other SEM-related simulation studies.

SEM in relation to *causality* has been widely discussed in the literature (see, e.g., Pearl, 2012). Causal modeling/inference (Pearl, 2009) builds on concepts from graph theory, and corresponding path analysis/SEM approaches integrated into a larger causal framework are sometimes referred to as *structural causal models* (see Kline, 2016, Chapter 8).

Another approach to path modeling are *partial least squares* (PLS) models, which are popular in economics and marketing. PLS can handle latent variables with corresponding measurement models, in addition to the structural specification. The main difference is that SEM is covariance based, whereas PLS is variance based. An extensive treatment of PLS model in R can be found in Sanchez (2013) using the **plspm** package (Sanchez et al., 2015).

3.4 Latent Growth Models

3.4.1 Simple Latent Growth Modeling

In Sect. 2.4.5 we presented a longitudinal CFA approach which gave us the opportunity to test hypotheses of change in latent variables in an ANOVA-like manner. Here we present an approach called *latent growth model* (LGM) which, in its basic form, does not involve any latent variables that are based on a measurement model. However, we create artificial latent variables that allow us to study longitudinal changes. In the simplest form of an LGM, we specify two *growth factors*:

- *Latent intercept*: allows us to describe individual starting points of the trajectories (as opposed to each individual starting at the same consumption level).
- *Latent shape*: allows us to specify various shapes or trend patterns for the growth trajectories.

To illustrate these concepts, throughout this section we use a dataset from Duncan et al. (1998), where alcohol, cigarette, and marijuana consumption is studied on $n = 1204$ individuals at four points in time. The dataset is included in the **aspect** package (Mair and De Leeuw, 2010). For the moment, we focus on cigarette consumption only. We specify two latent variables: intercept and shape. In an LGM we typically fix the latent intercept loadings to 1. This is analogous to what we do in a regression model where the first column of the design matrix is a full 1-vector.

For the latent shape, we have several options. For the moment we do not specify any functional relationship (i.e., linear or quadratic). We simply fix the first loading to 0. This reflects out initial level at time 1; thus, the intercept factor will be based on the time 1 measurement. For identifiability reasons we need to restrict one more shape factor loading: we fix the second loading to 1 while keeping the remaining ones open. Since the sample is fairly large, we use a weighted least squares estimator, sometimes also called *asymptotically distribution free* (ADF), such that we do not have to worry about potential normality violations. Using the `growth` function from **lavaan**, the model can be specified and fitted as follows:

```
library("lavaan")
library("aspect")
data("duncan")
model_shape <- '
   inter =~ 1*CIG_T1 + 1*CIG_T2 + 1*CIG_T3 + 1*CIG_T4
   shape =~ 0*CIG_T1 + 1*CIG_T2 + CIG_T3 + CIG_T4'
fitCig1 <-growth(model_shape, data = duncan, estimator = "WLS")
```

Before interpreting the parameters, let us examine the goodness of fit using the usual CFA/SEM indices. We get a RMSEA of 0.064 with a 95% CI of [0.037,0.094], a CFI of 0.975, an SRMR of 0.025, and a χ^2-value of 17.644 (df = 3, $p = 0.001$). Good fit.

The path diagram with standardized estimates can be produced as follows (see Fig. 3.7).

```
semPaths(fitCig1, what = "std",edge.label.cex = 0.7, esize = 1,
  edge.color = 1, sizeMan = 6, asize = 2.5, intercepts = FALSE,
  rotation = 4, mar = c(3, 5, 3.5, 5), fade = FALSE)
```

Let us have a look at the parameter estimates:

```
summary(fitCig1, header = FALSE)
##
## Latent Variables:
##                   Estimate  Std.Err  z-value  P(>|z|)
##   inter =~
##     CIG_T1           1.000
##     CIG_T2           1.000
##     CIG_T3           1.000
##     CIG_T4           1.000
##   shape =~
##     CIG_T1           0.000
##     CIG_T2           1.000
##     CIG_T3           2.408    0.456    5.275    0.000
##     CIG_T4           5.571    1.258    4.428    0.000
```

(continued)

```
##
## Covariances:
##                    Estimate  Std.Err  z-value  P(>|z|)
##   inter ~~
##     shape            -0.038    0.013   -2.854    0.004
##
## Intercepts:
##                    Estimate  Std.Err  z-value  P(>|z|)
##    .CIG_T1            0.000
##    .CIG_T2            0.000
##    .CIG_T3            0.000
##    .CIG_T4            0.000
##     inter             1.981    0.035   57.210    0.000
##     shape             0.050    0.013    3.797    0.000
##
## Variances:
##                    Estimate  Std.Err  z-value  P(>|z|)
##    .CIG_T1            0.179    0.032    5.658    0.000
##    .CIG_T2            0.145    0.019    7.558    0.000
##    .CIG_T3            0.198    0.016   12.338    0.000
##    .CIG_T4            0.078    0.047    1.666    0.096
##     inter             1.308    0.065   20.255    0.000
##     shape             0.022    0.010    2.139    0.032
```

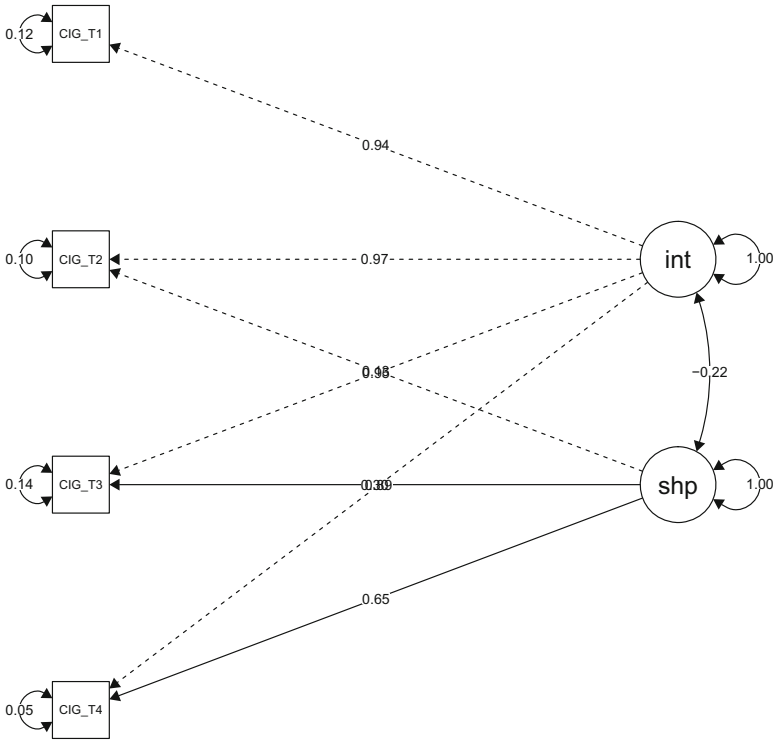

Fig. 3.7 Standardized parameter outputs of the cigarette consumption LGM (4 time points)

The first interesting output are the two free loadings for the shape factor. They determine the growth rate. Note that a linear growth loading sequence would be 0, 1, 2, and 3. We see that there is some accelerated growth on T4 (5.571 vs. 3 as expected with a linear growth). In a subsequent model, we will test whether a linear growth would fit equally well.

Furthermore, we get a significant intercept-shape covariance of −0.038 indicating that higher consumption levels at T1 predict lower rates of consumption after T1. Figure 3.7 shows the corresponding negative correlation (since we are plotting standardized parameters).

In the intercepts part of the output, the two non-zero estimates represent the fitted grand means for the latent intercept (at T1) and for the latent shape variable, respectively. The second estimate suggests an average consumption increase of 0.05 from T1 to T2. The remaining output refers to the error variances and the variances of the latent variables. Note that in this model we assume that the errors are uncorrelated. We will relax this assumption later.

Let us fit two additional models: In the first model, we specify a linear trend on the consumption, whereas in the second model, we use a quadratic trend. This quadratic trend results in a second latent shape construct. As mentioned above, a linear growth across the time points corresponds to a loadings sequence of 0, 1, 2, and 3. A quadratic trend corresponds to a loadings sequence of 0, 1, 4, and 9. Below we fix these loadings accordingly. After each model fit, we print out the trend parameters and some fit indices:

```
model_lin <- '
  inter =~ 1*CIG_T1 + 1*CIG_T2 + 1*CIG_T3 + 1*CIG_T4
  linear =~ 0*CIG_T1 + 1*CIG_T2 + 2*CIG_T3 + 3*CIG_T4'
fitCig2 <- growth(model_lin, data = duncan, estimator = "WLS")
parameterEstimates(fitCig2)[21, ]
##       lhs op rhs   est    se     z pvalue ci.lower ci.upper
## 21 linear ~1     0.086 0.009 9.598      0    0.069    0.104
round(fitMeasures(fitCig2)[c("rmsea", "cfi", "srmr")], 3)
## rmsea   cfi  srmr
## 0.072 0.948 0.027
model_quad <- '
  inter =~ 1*CIG_T1 + 1*CIG_T2 + 1*CIG_T3 + 1*CIG_T4
  linear =~ 0*CIG_T1 + 1*CIG_T2 + 2*CIG_T3 + 3*CIG_T4
  quad =~ 0*CIG_T1 + 1*CIG_T2 + 4*CIG_T3 + 9*CIG_T4'
fitCig3 <- growth(model_quad, data = duncan, estimator = "WLS")
parameterEstimates(fitCig3)[28:29,]
##       lhs op rhs   est    se    z pvalue ci.lower ci.upper
## 28 linear ~1     0.055 0.020 2.74  0.006    0.016    0.094
## 29   quad ~1     0.013 0.006 2.04  0.041    0.001    0.026
round(fitMeasures(fitCig3)[c("rmsea", "cfi", "srmr")], 3)
## rmsea   cfi  srmr
## 0.077 0.988 0.015
```

Both models fit fairly well. Of main interest are the trend parameters (intercepts). In the first model, there is a significant linear trend. The second model suggests a significant quadratic trend (note that the lower CI bound is very close to 0, however).

Let us now compare the linear trend model with the first model fitted above, where we did not specify a functional trend pattern:

```
anova(fitCig1, fitCig2)
## Chi Square Difference Test
##
##         Df AIC BIC  Chisq Chisq diff Df diff Pr(>Chisq)
## fitCig1  3         17.644
## fitCig2  5         35.947     18.303       2  0.0001061 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This test suggests that the model with the linear growth rate is significantly worse than the first model, where the growth parameters for T3 and T4 were freely estimated. This is due to the fact that we had an accelerated growth rate at T4. We could also test the linear model against the quadratic model in the same fashion. Comparing the quadratic model with the first model is somewhat trickier since they are non-nested. We cannot apply the `vuongtest` function because we used ADF estimation. We would have to refit the models using ML and then apply the same decision strategy as in the previous section.

3.4.2 Extended Latent Growth Modeling

In this section we extend the LGM from above into several directions. Note that the purpose of this section is to show various options for model specification using the **lavaan** syntax, rather than finding a best fitting model. In the first model fitted below, we treat the manifest variables as ordinal and use polychoric correlations; we keep the trend linear. We need to impose some restrictions on the threshold parameters in order to get a feasible solution. Below we set the first threshold parameters for the consumption at T1 and T2 to 0.

```
model_ord <- '
  inter =~ 1*CIG_T1 + 1*CIG_T2 + 1*CIG_T3 + 1*CIG_T4
  linear =~ 0*CIG_T1 + 1*CIG_T2 + 2*CIG_T3 + 3*CIG_T4
  CIG_T1 | 0*t1 + t2 + t3 + t4
  CIG_T2 | 0*t1 + t2 + t3 + t4'
fitCigord <- growth(model_ord, data = duncan,
                    ordered = names(duncan)[5:8])
```

The resulting parameter estimates can be interpreted in the same way as above.

In the second model, we treat the manifest variables as continuous (as in the previous section) and specify again a linear trend. However, this time we include a second variable: marijuana consumption. We impose a linear trend on this variable as well. Of course, we could easily try, for instance, a quadratic trend for the marijuana consumption, while keeping cigarette trend linear. To add an additional flavor to the model, we specify correlated errors across subsequent time points. Without any further restrictions, we would end up in a Heywood case. To avoid this, we restrict the error variances to be constant for the cigarette and marijuana indicators.

```
model_pc <- '
  cint =~ 1*CIG_T1 + 1*CIG_T2 + 1*CIG_T3 + 1*CIG_T4
  clin =~ 0*CIG_T1 + 1*CIG_T2 + 2*CIG_T3 + 3*CIG_T4
  pint =~ 1*POT_T1 + 1*POT_T2 + 1*POT_T3 + 1*POT_T4
  plin =~ 0*POT_T1 + 1*POT_T2 + 2*POT_T3 + 3*POT_T4

  ## correlated errors
  CIG_T1 ~~ CIG_T2; CIG_T2 ~~ CIG_T3; CIG_T3 ~~ CIG_T4
  POT_T1 ~~ POT_T2; POT_T2 ~~ POT_T3; POT_T3 ~~ POT_T4

  ## fix error variances
  CIG_T1 ~~ rc*CIG_T1
  CIG_T2 ~~ rc*CIG_T2
  CIG_T3 ~~ rc*CIG_T3
  CIG_T4 ~~ rc*CIG_T4
  POT_T1 ~~ rp*POT_T1
  POT_T2 ~~ rp*POT_T2
  POT_T3 ~~ rp*POT_T3
  POT_T4 ~~ rp*POT_T4'
fitPC1 <- growth(model_pc, data = duncan, estimator = "WLS")
round(fitMeasures(fitPC1)[c("rmsea", "cfi", "srmr")], 3)
## rmsea   cfi  srmr
## 0.055 0.923 0.046
```

The path diagram is shown in Fig. 3.8 and includes the unstandardized parameters in order to reflect the corresponding parameter restrictions. Again, we get a reasonably good fit. The linear trend parameters can be interpreted in the same way as above. Let us have a look at the correlation parameters of the latent variables:

```
inspect(fitPC1, "std")$psi
##       cint   clin   pint   plin
## cint  1.000
## clin -0.197  1.000
```

(continued)

```
## pint  0.609 -0.213  1.000
## plin -0.098  0.736 -0.182  1.000
```

We get a considerably high correlation in positive direction between the cigarette and marijuana intercepts (0.609). This indicates that at T1, the higher the cigarette consumption, the higher the marijuana consumption. The correlation between the linear constructs is −0.213. Hence, the higher the increase in cigarette consumption, the lower the increase in marijuana consumption. The remaining correlations can be interpreted in an analogous fashion.

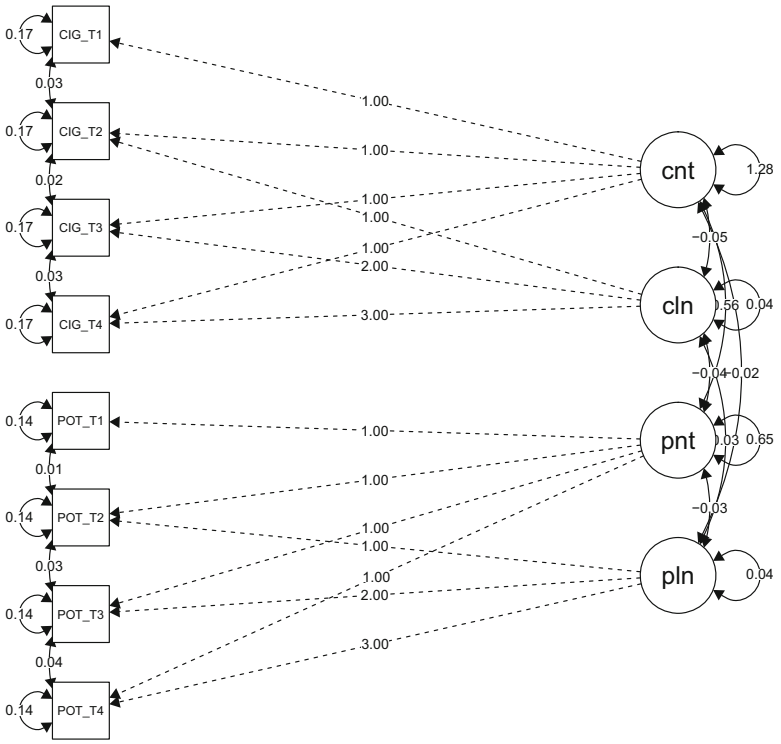

Fig. 3.8 LGM path diagram for cigarette and marijuana consumption (unstandardized parameters, correlated errors)

Finally, let us include a covariate. Covariates can be either *time-dependent* or *time-independent*. In this dataset there is an additional set of variables related to the alcohol consumption which we can use as covariates. In the first model, let us treat them as time-independent as we simply average the alcohol consumption across the four time points (i.e., we have one covariate only):

```
duncan$ALCavg <- rowMeans(duncan[, 9:12])
```

In order to keep the model specification somewhat slim, we remove the correlated errors from the specification above and get rid of the variance restrictions once more. We are interested in studying whether the average alcohol consumption influences the marijuana latent variables (i.e., intercept and trend). This time we use a robust ML estimator (since ADF screams and gives negative variances).

```
model_pca1 <- '
  cint =~ 1*CIG_T1 + 1*CIG_T2 + 1*CIG_T3 + 1*CIG_T4
  clin =~ 0*CIG_T1 + 1*CIG_T2 + 2*CIG_T3 + 3*CIG_T4
  pint =~ 1*POT_T1 + 1*POT_T2 + 1*POT_T3 + 1*POT_T4
  plin =~ 0*POT_T1 + 1*POT_T2 + 2*POT_T3 + 3*POT_T4

  ## effects of alcohol on marijuana
  pint ~ ALCavg
  plin ~ ALCavg'
fitPCA1 <- growth(model_pca1, data = duncan, estimator = "MLR")
round(fitMeasures(fitPCA1)[c("rmsea", "cfi", "srmr")], 3)
## rmsea   cfi  srmr
## 0.163 0.900 0.278
```

The fit is getting worse. Nevertheless, for illustration, let us pull out the path coefficients for alcohol on marijuana:

```
parameterEstimates(fitPCA1)[17:18,]
##     lhs op    rhs   est    se      z pvalue ci.lower ci.upper
## 17 pint  ~ ALCavg 0.316 0.026 12.213      0    0.265    0.366
## 18 plin  ~ ALCavg 0.048 0.008  5.686      0    0.032    0.065
```

We see that alcohol has a significant influence on the initial marijuana consumption as well as on the increase of marijuana consumption. Both effects point in a positive direction. The model structure including the unstandardized estimates is given in Fig. 3.9.

Finally, we climb the Mount Everest in this section and include alcohol as time-dependent covariate associated with marijuana consumption. The corresponding path diagram is given in Fig. 3.10.

```
model_pca2 <- '
```

(continued)

```
  cint =~ 1*CIG_T1 + 1*CIG_T2 + 1*CIG_T3 + 1*CIG_T4
  clin =~ 0*CIG_T1 + 1*CIG_T2 + 2*CIG_T3 + 3*CIG_T4
  pint =~ 1*POT_T1 + 1*POT_T2 + 1*POT_T3 + 1*POT_T4
  plin =~ 0*POT_T1 + 1*POT_T2 + 2*POT_T3 + 3*POT_T4

  ## effects of alcohol on marijuana
  POT_T1 ~ ALC_T1
  POT_T2 ~ ALC_T2
  POT_T3 ~ ALC_T3
  POT_T4 ~ ALC_T4'
fitPCA2 <- growth(model_pca2, data = duncan, estimator = "MLR")
round(fitMeasures(fitPCA2)[c("rmsea", "cfi", "srmr")], 3)
## rmsea   cfi  srmr
## 0.104 0.938 0.163
```

The fit has slightly improved compared to the time-independent model. Again, let us extract the path coefficients for alcohol on marijuana:

```
parameterEstimates(fitPCA2)[17:20,]
##       lhs op    rhs   est    se      z pvalue ci.lower ci.upper
## 17 POT_T1  ~ ALC_T1 0.159 0.018  8.642      0    0.123    0.195
## 18 POT_T2  ~ ALC_T2 0.163 0.015 10.552      0    0.133    0.193
## 19 POT_T3  ~ ALC_T3 0.179 0.015 11.779      0    0.149    0.208
## 20 POT_T4  ~ ALC_T4 0.182 0.018 10.302      0    0.148    0.217
```

We see that alcohol has a significant influence on marijuana consumption at each point in time.

This concludes the section on LGMs. The possibilities for specifying such growth models are vast. Some additional flavors of these models can be found in Beaujean (2014) and in Little (2013). They can be also integrated into a wider SEM framework where, instead of growth on manifest variables, we are interested in growth of latent variables. The indicators in the models above are then replaced by latent variables which, in turn, are measured by a CFA. Illustrations can be found in Little (2013, Chapter 8) who calls this type of model *multivariate growth curve model*.

Another option to specify growth curve models is through a mixed-effects framework (see, e.g., Mirman, 2014). Compared to this modeling framework, LGM offer more flexibility in terms of multivariate settings and the potential to embedding it into a larger latent variable framework. For LGM it is easy to obtain global fit statistics and parameter testing can be carried out in a very extensive manner. We also have the possibility to have detailed specification of covariance structures and perform corresponding inference on them. A shortcoming of LGM is that we need observations of each individual at each point in time. Thus, the design has to be balanced, apart from potential missing values that **lavaan** can handle. Using mixed-effects models, the data can be unbalanced (e.g., due to subject attrition). In addition,

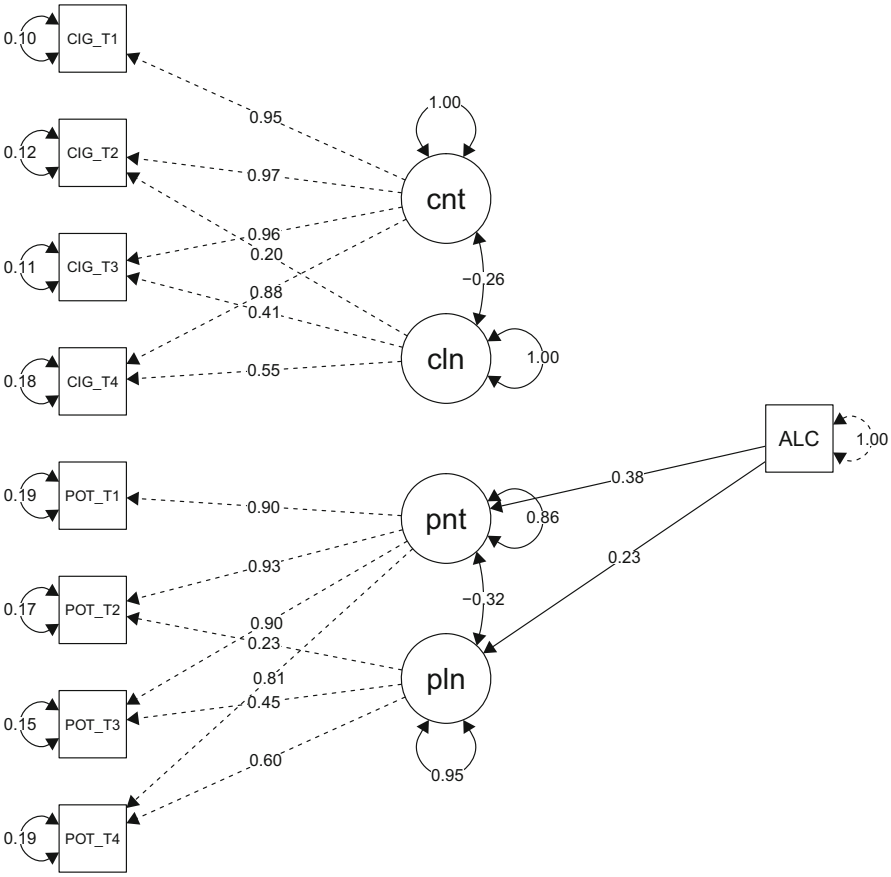

Fig. 3.9 LGM path diagram for cigarette and marijuana consumption with alcohol as time-independent covariate (unstandardized estimates)

mixed-effects models give us all the GLM modeling opportunities in terms of handling all sorts of response types. They make it easy to incorporate autocorrelation structures and other extensions from time series analysis. Elaborations on these two "competing" approaches can be found in Grimm et al. (2016).

References

Aiken, L. S., & West, S. G. (1991). *Multiple regression: Testing and interpreting interactions*. Newbury Park: Sage.

Baron, R. M., & Kenny, D. A. (1986). The moderator-mediator variable distinction in social psychological research: Conceptual, strategic, and statistical considerations. *Journal of Personality and Social Psychology, 51*, 1173–1182.

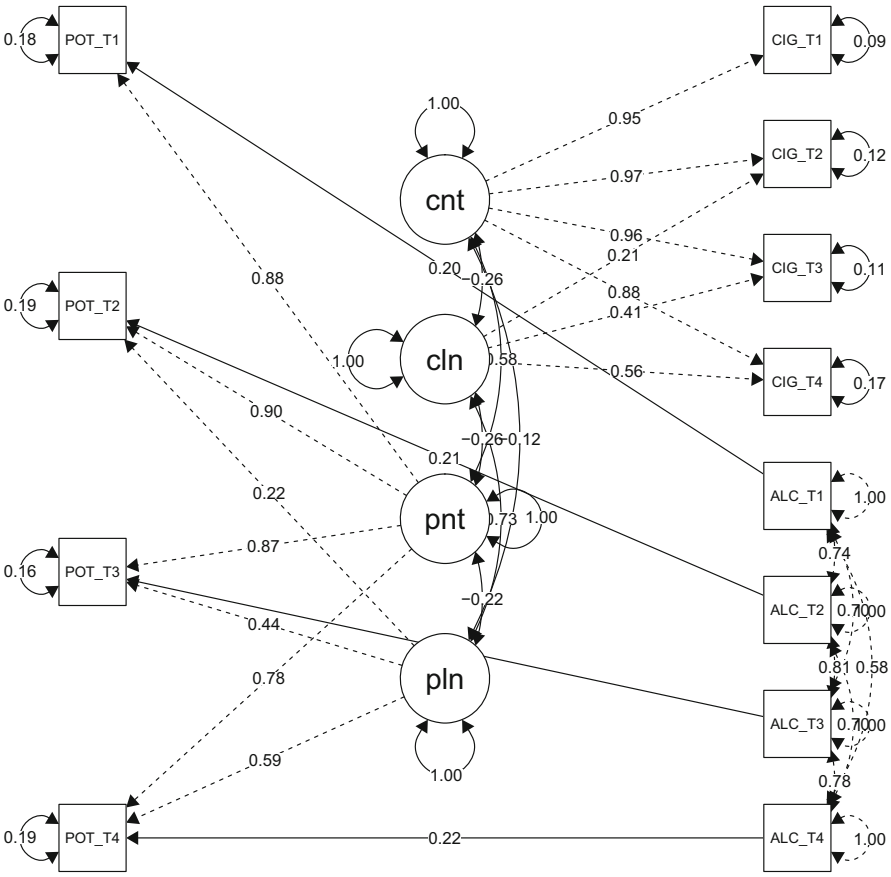

Fig. 3.10 LGM path diagram for cigarette and marijuana consumption with alcohol as time-dependent covariate (unstandardized estimates)

Beaujean, A. A. (2014). *Latent variable modeling using R: A step-by-step guide*. New York: Routledge.

Bergh, R., Akrami, N., Sidanius, J., & Sibley, C. (2016). Is group membership necessary for understanding prejudice? A re-evaluation of generalized prejudice and its personality correlates. *Journal of Personality and Social Psychology, 111*, 367–395.

Berridge, D. M., & Crouchley, R. (2011). *Multivariate generalized linear mixed models using R*. Boca Raton: CRC Press.

Bollen, K. A. (1989). *Structural equations with latent variables*. New York: Wiley.

Dalal, D. K., & Zickar, M. J. (2012). Some common myths about centering predictor variables in moderated multiple regression and polynomial regression. *Organizational Research Methods, 15*, 339–362.

Duncan, S. C., Duncan, T. E., & Hops, H. (1998). Progressions of alcohol, cigarette, and marijuana use in adolescence. *Journal of Bahavioral Medicine, 21*, 375–388.

Epskamp, S. (2015). **semPlot**: Unified visualizations of structural equation models. *Structural Equation Modeling: A Multidisciplinary Journal, 22*, 474–483.

Fletcher, T. D. (2012). **QuantPsyc**: Quantitative psychology tools. R package version 1.5. http://CRAN.R-project.org/package=QuantPsyc

Fox, J., & Weisberg, S. (2010). *An R companion to applied regression*. Thousand Oaks: Sage.

Grimm, K. J., Ram, N., & Estabrook, R. (2016). *Growth modeling: Structural equation and multilevel modeling approaches*. New York: Guilford.

Hayes, A. F. (2013). *Introduction to mediation, moderation, and conditional process analysis: A regression-based approach*. New York: Guilford.

Judd, C. M., Kenny, D. A., & McClelland, G. H. (2001). Estimating and testing mediation and moderation in within-participant designs. *Psychological Methods, 6*, 115–134.

Kline, R. B. (2016). *Principles and practice of structural equation modeling* (4th ed.). New York: Guilford Press.

Kromrey, J. D., & Foster-Johnson, L. (1998). Mean centering in moderated multiple regression: Much ado about nothing. *Educational and Psychological Measurement, 58*, 42–67.

Little, T. D. (2013). *Longitudinal structural equation modeling*. New York: Guilford.

Mair, P., & De Leeuw, J. (2010) A general framework for multivariate analysis with optimal scaling: The R package **aspect**. *Journal of Statistical Software, 32*(1), 1–23. https://www.jstatsoft.org/index.php/jss/article/view/v032i09

Merkle, E. C., & Rosseel, Y. (2017, Forthcoming). **blavaan**: Bayesian structural equation models via parameter expansion. *Journal of Statistical Software*. *85*(4), 1–30.

Merkle, E. C., You, D., & Preacher, K. J. (2016). Testing non-nested structural equation models. *Psychological Methods, 21*, 151–163.

Mirman, D. (2014). *Growth curve analysis and visualization using R*. Boca Raton: Chapman & Hall, CRC.

Muller, D., Judd, C. M., & Yzerbyt, V. Y. (2005). When moderation is mediated and mediation is moderated. *Journal of Personality and Social Psychology, 89*, 852–863.

Paškvan, M., Kubicek, B., Prem, R., & Korunka, C. (2016). Cognitive appraisal of work intensification. *International Journal of Stress Management, 23*, 124–146.

Pearl, J. (2009). *Causality: Models, reasoning, and inference*. New York: Cambridge University Press.

Pearl, J. (2012). The causal foundations of structural equation modeling. In: R. H. Hoyle (Ed.), *Handbook of structural equation modeling* (pp. 68–91). New York: Guilford.

Pornprasertmanit, S., Miller, P., & Schoemann, A. (2016). **simsem**: Simulated structural equation modeling. R package version 0.5-13. https://CRAN.R-project.org/package=simsem

Preacher, K. J., & Hayes, A. F. (2004). SPSS and SAS procedures for estimating indirect effects in simple mediation models. *Behavior Research Methods, Instruments, and Computers, 36*, 717–731.

Rosseel, Y. (2012). **lavaan**: An R package for structural equation modeling. *Journal of Statistical Software, 48*(2), 1–36. http://www.jstatsoft.org/v48/i02/

Sanchez, G. (2013). *PLS path modeling with R*. Berkeley: Trowchez Editions. http://www.gastonsanchez.com/PLS_Path_Modeling_with_R.pdf

Sanchez, G., Trinchera, L., & Russolillo, G. (2015). **plspm**: Tools for partial least squares path modeling (PLS-PM). R package version 0.4.7. https://CRAN.R-project.org/package=plspm

Tingley, D., Yamamoto, T., Hirose, K., Keele, L., & Imai, K. (2014). **mediation**: R package for causal mediation analysis. *Journal of Statistical Software, 59*, 1–38. http://www.jstatsoft.org/v59/i05/

Vuong, Q. H. (1989). Likelihood ratio tests for model selection and non-nested hypotheses. *Econometrica, 57*, 307–333.

Chapter 4
Item Response Theory

4.1 Introductory Remarks and Dimensionality Assessment

4.1.1 Classification of IRT Models

Item response theory (IRT) is often also referred to as *latent trait analysis* or *modern test theory*. It is a measuring paradigm, designed for categorical data, that is fundamentally different from classical test theory[1] (CTT; see Chap. 1). IRT models can be generally classified according to

- the nature of the input data (dichotomous vs. polytomous items),
- the dimensionality of the underlying latent trait (unidimensional vs. multidimensional).

Dimensionality assessment will be covered in the following subsection. Subsequently, we focus on various popular unidimensional models for dichotomous and polytomous items, before extending them to multidimensional traits.

4.1.2 Assessing Dimensionality

Prior to fitting an IRT model, the dimensionality of the scale should be assessed carefully. There are several ways to explore the dimensionality of a scale:

- categorical principal component analysis (*Princals*),
- *exploratory factor analysis* (EFA) on tetrachoric/polychoric correlations,
- *item factor analysis* (IFA).

[1] Detailed elaborations on differences between IRT, CTT, and factor analysis can be found in Rusch et al. (2017).

P. Mair, *Modern Psychometrics with R*, Use R!,
https://doi.org/10.1007/978-3-319-93177-7_4

Princals will be explained in detail in Sect. 8.2. For the moment, it is just important to know that it is a dimension reduction technique suited for categorical data, which gives us hints about the number of underlying dimensions. EFA was covered in Sect. 2.2. IFA will be discussed in Sect. 4.7.1. What we need to know for this section is that IFA is a variant of EFA suited for categorical data.

To illustrate dimensionality assessment using these approaches, we consider a dataset from Koller and Alexandrowicz (2010). They used the Neuropsychological Test Battery for Number Processing and Calculation in Children (ZAREKI-R; von Aster et al., 2006) for the assessment of dyscalculia in children. There are $n = 341$ children (2nd to 4th year of elementary school) in their sample, eight items on addition, eight items on subtraction, and two covariates (time needed for completion in minutes as well as grade). In this example we consider eight binary subtraction items only.

```
library("MPsychoR")
library("mirt")
data("zareki")
zarsub <- zareki[, grep("subtr", colnames(zareki))]
```

We are interested in whether the subtraction items measure a single latent trait or if multiple traits are needed. We start our dimensionality assessment with fitting a two-dimensional Princals solution using the **Gifi** package (Mair and De Leeuw, 2017) in order to get a picture of item associations in a 2D space:

```
library("Gifi")
prinzar <- princals(zarsub)
plot(prinzar, main = "Zareki Loadings")
```

Figure 4.1 shows the corresponding loadings plot. If the items were unidimensional, the arrows should approximately point in the same direction. We see that item `subtr5` is somewhat separated from the rest, whereas the remaining ones look fairly homogeneous. This plot suggested that unidimensionality might be violated due to `subtr5`.

As a second tool, we use an EFA on the tetrachoric correlation matrix and use criteria from Sect. 2.2.4 such as VSS, MAP, and BIC to assess dimensionality. We set the maximum number of factors to four and use the following function call from the **psych** package (Revelle, 2017):

The VSS (output not show here) suggests two factors, the MAP one factor, and the BIC three factors. As an additional diagnostic tool, a parallel analysis using the `fa.parallel` function can be performed as well. Note that if the input items are polytomous, setting `cor="poly"` does the job.

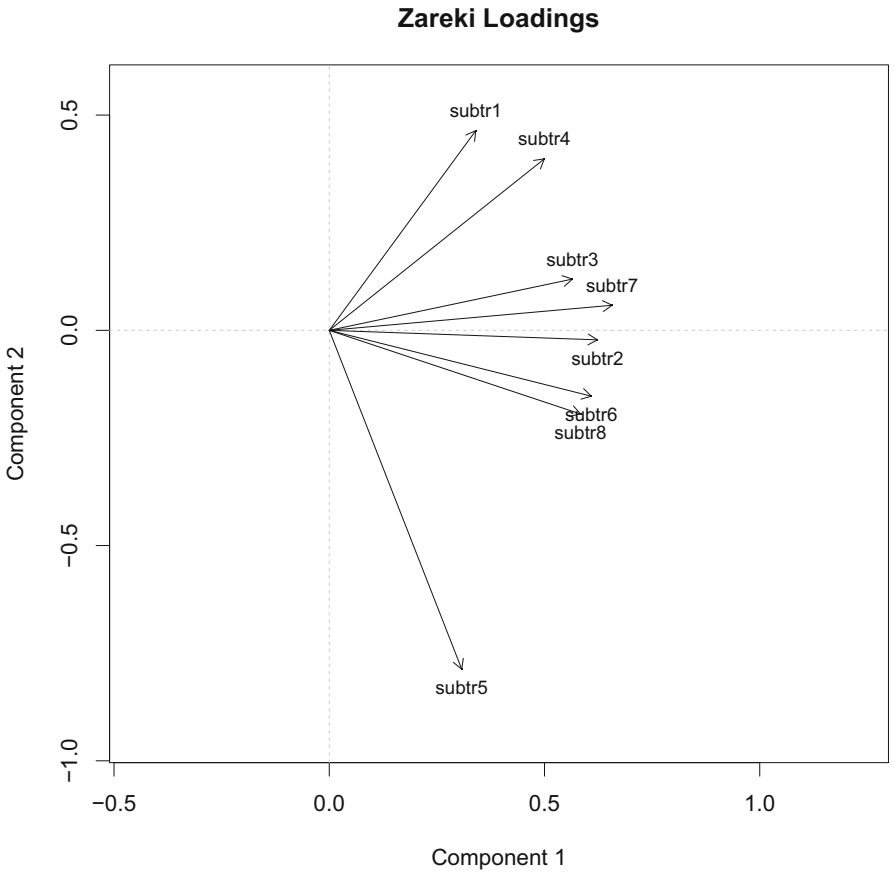

Fig. 4.1 Princals loadings plot for subtraction items in ZAREKI data

As a third tool, let us use IFA as implemented in the **mirt** package (Chalmers, 2012). We fit a one-factor model and a two-factor model, and we compare these nested fits via a likelihood-ratio (LR) test, in addition to the usual AIC/BIC criteria.

```
fitifa1 <- mirt(zarsub, 1, verbose = FALSE)
fitifa2 <- mirt(zarsub, 2, verbose = FALSE, TOL = 0.001)
anova(fitifa1, fitifa2, verbose = FALSE)
##        AIC     AICc    SABIC      BIC    logLik     X2  df
## 1 2558.405 2560.084 2568.959 2619.715 -1263.202    NaN NaN
## 2 2561.249 2564.732 2576.422 2649.383 -1257.625 11.155   7
##       p
## 1   NaN
## 2 0.132
```

The nonsignificant result of the LR-test suggests that the 2D fit is not superior to the 1D fit. AIC and BIC are (slightly) lower for the 1D solution.

Using all these tools in combination, we conclude that there is no drastic unidimensionality violation in these data. Still, we obtained some hints that it might be slightly violated. Princals gave a good indication that item 5 may not behave in the same way as the remaining items. We do not have to exclude any items at this point since, as we will see, for simple dichotomous IRT models, various tests are sensitive to unidimensionality violations. If we would have detected two clearly separated sets of items, we could fit two separate unidimensional IRT models, one for each itemset. Alternatively, as we will see later in this chapter, there are also options for fitting a two-dimensional IRT model on the entire set of items.

Other approaches for unidimensionality assessment in R are implemented in the **sirt** package (Robitzsch, 2017, see `expl.detect` and `unidim.test.csn`) as well as in **ltm** (Rizopoulos, 2006, see `unidimTest`).

Before we start elaborating on various IRT models, a few general remarks regarding fit assessment in IRT. IRT models can be used for several purposes. On the one extreme, we can use IRT for scale construction. That is, we want to find a set of "high-quality" items that measure an underlying construct as good as possible. On the other extreme, we use a (well-established) scale, and our main interest is to score the persons, and not so much to eliminate (slightly) misfitting items. Depending on the purpose of the IRT analysis, different criteria may be used for fit assessment and interpreted with various degrees of strictness: in a scale construction scenario, we want to be strict, whereas in a person scoring scenario, we can be more laid back in terms of misfit. The point is that users should not rely blindly on p-values spit out by various model tests but rather assess the fit in relation to the purpose the model is being used for (see Maydeu-Olivares, 2015). In any case, a good a priori dimensionality assessment is crucial.

4.2 Unidimensional Dichotomous IRT Models

4.2.1 The Rasch Model

The probabilistic model proposed by Rasch (1960) revolutionized psychometrics. Despite the fact that the model is mathematically embarrassingly simple, it is extremely profound from a measurement point of view (see Rasch, 1961; Fischer and Molenaar, 1995). The Rasch model can be used if we aim to construct a scale that fulfills highest measurement standards.

Let $\mathbf{X}$ be an $n \times m$ data matrix with n individuals in the rows ($v = 1, \ldots, n$) and m items in the columns ($i = 1, \ldots, m$). All items are scored dichotomously (i.e., 0 or 1). The Rasch model is formally defined as:

$$P(X_{vi} = 1) = \frac{\exp(\theta_v + \beta_i)}{1 + \exp(\theta_v + \beta_i)}. \tag{4.1}$$

We model the probability that person v scores 1 on item i. Each item gets its individual *item location parameter* β_i. Using the parameterization from Eq. (4.1), within the context of ability tests, β_i is often referred to as *item easiness parameter*.[2] Correspondingly, $-\beta_i$ is the *item difficulty parameter*. Each person gets a *person parameter* θ_v, often called *person ability parameter*, which is obtained in a second estimation step. Both the β's and the θ's are on an interval scale and can be mapped on the same latent trait (i.e., they are directly comparable). The Rasch model has three fundamental assumptions:

- unidimensionality of the latent trait,
- parallel *item characteristic curves* (ICCs),
- local independence.

Unidimensionality was covered in detail in the section above. ICCs in conjunction with parallel shifts will be explained further below, after fitting the Rasch model. Local independence means that given a person parameter, item responses become independent. This assumption is difficult to check and often omitted in practice. Below we present a nonparametric test which helps us to assess potential violations. A bootstrap-based approach is presented in Finch and French (2015, p. 238).

Let us fit a Rasch model on the ZAREKI-R subtraction items from above. If these data fit the Rasch model, all three assumptions can be considered as fulfilled. In R, Rasch models can be computed using the **eRm** package (Mair and Hatzinger, 2007b) which uses a conditional maximum likelihood (CML) approach, which has some advantages over other IRT estimation approaches (see Mair and Hatzinger, 2007a).

```
library("eRm")
fitrasch1 <- RM(zarsub)
fitrasch1
##
## Results of RM estimation:
##
## Call:  RM(X = zarsub)
##
## Conditional log-likelihood: -646.9202
## Number of iterations: 12
## Number of parameters: 7
##
## Item (Category) Difficulty Parameters (eta):
##              subtr2    subtr3     subtr4    subtr5
## Estimate -0.7552998 1.6808330 -0.4774069 -0.280543
## Std.Err   0.1619353 0.1310474  0.1515977  0.145557
```

(continued)

[2]We use the easiness parameterization in order to be consistent with the implementation in the **eRm** package.

```
##             subtr6    subtr7     subtr8
## Estimate 0.4163264 1.5508677 -0.1884142
## Std.Err  0.1316531 0.1296646  0.1430740
```

This model call fits the item parameters only. In the print output, the item parameters are called $\boldsymbol{\eta}$. There seems to be one parameter missing: the one for item 1. This is due to the fact that there is a restriction involved in the estimation, in order to make the model identifiable. The full vector of $\boldsymbol{\beta}$ parameters for all items can be obtained via a multiplication with a design matrix $\mathbf{W}$ which is constructed internally (i.e., $\boldsymbol{\beta} = \mathbf{W}\boldsymbol{\eta}$). Details can be found in Mair and Hatzinger (2007b). In order to extract the entire set of easiness parameters, we can say

```
round(fitrasch1$betapar, 3)
## beta subtr1 beta subtr2 beta subtr3 beta subtr4 beta subtr5
##       1.946       0.755      -1.681       0.477       0.281
## beta subtr6 beta subtr7 beta subtr8
##      -0.416      -1.551       0.188
```

The difficulty parameters, sorted from the easiest to the most difficult item, are

```
round(sort(-fitrasch1$betapar), 3)
## beta subtr1 beta subtr2 beta subtr4 beta subtr5 beta subtr8
##      -1.946      -0.755      -0.477      -0.281      -0.188
## beta subtr6 beta subtr7 beta subtr3
##       0.416       1.551       1.681
```

Still, we need to verify whether the model fits, otherwise the interpretation of these parameters is meaningless. A good strategy is to apply the LR-test proposed by Andersen (1973), which is based on the following idea. Rasch measurement implies that the item parameters have to be invariant across person subgroups (*measurement invariance*). Therefore, for the Rasch model to fit, the item parameters based on separate subgroup fits have to be approximately the same. This needs to hold for any subgroup split. For instance, we can split the sample according to a median or mean split based on the number of items solved or, even better, perform the split according to one or multiple binary covariates. Here we use time as external splitting variable (fast vs. slow according to a median split):

```
timecat <- factor(zareki$time <= median(zareki$time),
                  labels = c("fast", "slow"))
fitLR <- LRtest(fitrasch1, timecat)
fitLR
##
## Andersen LR-test:
## LR-value: 24.097
## Chi-square df: 7
## p-value:  0.001
```

It gives a significant result which implies that the likelihoods differ across the two groups. The model does not fit. Which item is responsible for the misfit? We can explore this question in detail using a Wald test with the same split criterion:

```
Waldtest(fitrasch1, timecat)
##
## Wald test on item level (z-values):
##
##             z-statistic p-value
## beta subtr1      -0.360   0.719
## beta subtr2       0.237   0.813
## beta subtr3      -2.342   0.019
## beta subtr4       0.730   0.465
## beta subtr5       4.199   0.000
## beta subtr6      -0.548   0.584
## beta subtr7      -1.529   0.126
## beta subtr8      -0.469   0.639
```

Here, we can use the magnitudes of the p-values to judge the degree of misfit and come to the conclusion that `subtr5`, already suspicious in the Princals solution above, is most responsible for the misfit. A graphical illustration reflecting this deviation can be produced as follows:

```
plotGOF(fitLR, ctrline = list(col = "gray"), conf = list())
```

The plot is given in Fig. 4.2. If the parameters were exactly the same across the two subsamples, they would lie on the diagonal, and the model would fit. The gray lines reflect the confidence bands around the diagonal. The 95% confidence ellipses for the item parameters are shown in red. We see that `subtr5` is clearly outside the confidence band, and we have good evidence for eliminating it. We are going to refit the model without this item, apply the LR-test once more, and check whether the Rasch model fits:

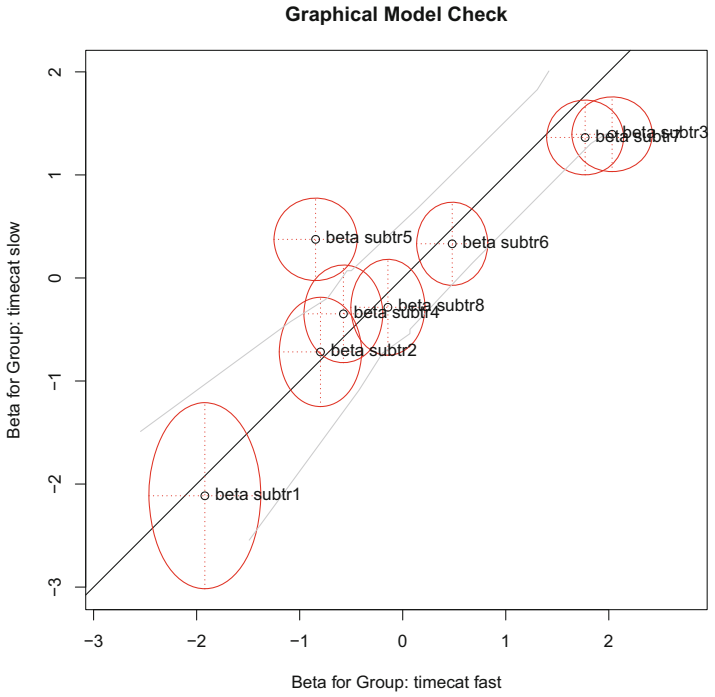

Fig. 4.2 Graphical Rasch model check based on time subgroup split, including 95% confidence ellipses (red) and diagonal confidence band (gray)

```
fitrasch2 <- RM(zarsub[, -5])
LRtest(fitrasch2, timecat)
##
## Andersen LR-test:
## LR-value: 5.715
## Chi-square df: 6
## p-value:  0.456
```

The nonsignificant p-value suggests that the model fits.

In practice, it is suggested to run these tests on multiple external binary covariates. It is important to eliminate only one item at a time and then refit the model which, of course, can be tedious in cases where many items need to be eliminated. The **eRm** package provides the convenience function `stepwiseIt` which eliminates items until a particular test of choice fits.

It has been shown in simulation studies (Suárez-Falcón and Glas, 2003) that the LR-test works well for detecting violations of unidimensionality and parallel ICC violations. Based on the nonsignificant result from above, we can conclude that these assumptions are not violated. If we want to test assumptions more explicitly,

the nonparametric testing framework proposed by Ponocny (2001) provides a large amount of testing possibilities. The **eRm** package uses the **RaschSampler** package (Verhelst et al., 2007) internally determine to compute the test statistics. As an example, we examine item-specific local independence (T1-test) and local independence at a global level (T11-test).

```
set.seed(123)
T1 <- NPtest(as.matrix(zarsub[, -5]), n = 1000, method = "T1")
T1
## Nonparametric RM model test: T1 (local dependence -
##     increased inter-item correlations)
##     (counting cases with equal responses on both items)
## Number of sampled matrices: 1000
## Number of Item-Pairs tested: 21
## Item-Pairs with one-sided p < 0.05
## none
T11 <- NPtest(as.matrix(zarsub[, -5]),n = 1000, method = "T11")
T11
## Nonparametric RM model test: T11 (global test - local
##     dependence)
## (sum of deviations between observed and expected
##     inter-item correlations)
## Number of sampled matrices: 1000
## one-sided p-value: 0.927
```

The first output suggests that none of the item pairs show significant local dependence. The second test output confirms this result by telling us that local independence holds at a global level.

Additional tests implemented in **eRm** are various itemfit statistics and indices (function `itemfit`, see Bond and Fox, 2015, for details) and the Martin-Löf LR-test (see `MLoef`).

At this point we can conclude that our data fit the Rasch model. This gives our scale the highest seal of approval. Thus, we can interpret the item parameters and, finally, elaborate on what (parallel) ICCs are. The sorted item difficulties are the following:

```
round(sort(-fitrasch2$betapar), 2)
## beta subtr1 beta subtr2 beta subtr4 beta subtr8 beta subtr6
##       -2.08       -0.83       -0.54       -0.24        0.39
## beta subtr7 beta subtr3
##        1.58        1.72
```

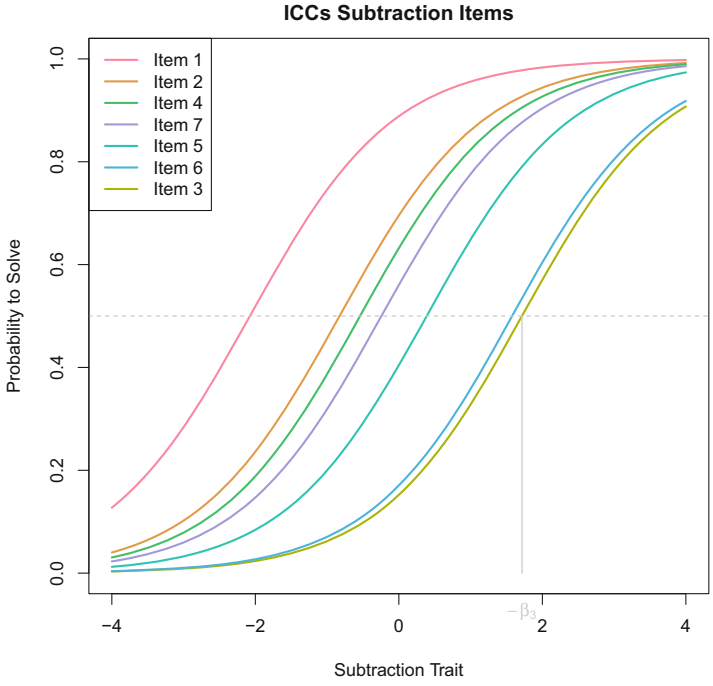

Fig. 4.3 ICCs for 7 ZAREKI-R subtraction items. The item difficulty parameter (i.e., negative easiness parameter from the **eRm** fit) is marked for item 3

ICCs, sometimes also called *item response functions*, determine the item behavior (i.e., endorsement probability) along the latent trait. Figure 4.3 plots the ICCs for all items in a single device.

```
plotjointICC(fitrasch2, xlab = "Subtraction Trait",
             main = "ICCs Subtraction Items")
```

First of all, we see that all ICCs are parallel, and all of them have a slope of 1. This is an implication of the Rasch model which will be relaxed in the next section. The plot shows that item 1 is the easiest item, since it is located furthest to the left on the trait. Item 3 is to the far right and consequently the most difficult one. In this plot, the item parameters are reflected the value on the x-axis for which the endorsement probability is exactly 0.5. In the plot, this is demonstrated for item 3.

Once the model fits, we can estimate the person parameter in a second step:

```
zarppar <- person.parameter(fitrasch2)
```

Each individual gets its own person parameter. Children who answered the same number of items correctly[3] will get the same person parameter. They are mapped on the same latent trait as the items. For instance, in Fig. 4.3, we could draw another vertical line at a particular $\hat{\theta}_v$ value, which would give us the response probabilities of person v for each item. If the person parameter and an item parameter coincide, this means that this person has a probability of 0.5 to score 1 on this particular item.

The person parameters can be added to the data matrix and used for further statistical analysis as metric variable. Here we show an ANOVA with school year as factor (three levels).

```
zareki$theta <- zarppar$theta.table[,1]
summary(aov(theta ~ class, data = zareki))
##              Df Sum Sq Mean Sq F value   Pr(>F)
## class         2  129.1   64.55    31.8 2.23e-13 ***
## Residuals   338  686.2    2.03
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The result suggests significant differences in ability across the 3 years. Still, the subtraction scale is fair since the Rasch model holds: fairness is a property of the item parameters, and not a person parameter property.

An important contribution to the Rasch ecosystem was the idea to embed Rasch models into a mixed-effects models framework. To be more precise, Rasch models are nothing else than mixed-effects logistic regressions with, in their most basic form, items as fixed effects and persons as random effects. The random intercepts correspond to the person parameters, the fixed effect parameters to the item location parameters. This idea of *explanatory item response models* adds many modeling possibilities (see De Boeck et al., 2011, for details).

4.2.2 Two-Parameter Logistic Model

The parallel ICC assumption involving constant slope parameters of 1 is a rather strict one. *One-parameter logistic model* versions of Rasch (called 1-PL or OPLM)

[3] In the Rasch model, it does not matter which particular items are solved correctly, as long as the sum scores are the same. This property is called *sufficiency*.

relax this assumption slightly by estimating a single *item discrimination parameter* α, generally different from 1. Still, the ICCs are parallel. This idea represents a different philosophical measurement perspective. Under the Rasch paradigm, we modify the data such that it fits the Rasch model. Under the 1-PL and the other models presented in this section, we try to find a model that fits our data.

In practice, the *two-parameter logistic model* (2-PL Birnbaum, 1968) is more relevant than the 1-PL. It estimates an *item discrimination parameter* α_i for each item individually, which allows the ICCs to cross. Equation (4.1) changes as follows[4]:

$$P(X_{vi} = 1) = \frac{\exp(\alpha_i(\theta_v - \beta_i))}{1 + \exp(\alpha_i(\theta_v - \beta_i))}. \tag{4.2}$$

Compared to the Rasch model, it relaxes the assumption of parallel ICCs through α_i, while unidimensionality and local independence still need to hold.

The dataset we use to illustrate the 2-PL is taken from Mair et al. (2015). In this study, the authors were interested in finding out why package developers contribute to R. Among other things, they presented three subscales of the Work Design Questionnaire (WDQ) by Morgeson and Humphrey (2006), in order to explore whether certain work design characteristics had an influence on the participation in package developments. In our little IRT application below, the main interest is not to remove many items in order to achieve a strong measurement instrument. Rather, the main objective is to score the developers and only remove items that are heavily misfitting. Hence, we use a more flexible 2-PL since a Rasch model might be too strict.

We consider a single WDQ subscale only: "knowledge characteristics" which includes 18 dichotomous items related to job complexity, information processing, problem solving, skill variety, and specialization.[5] Let us fit a 2-PL using the **ltm** package.

```
library("ltm")
data("RWDQ")
fit2pl1 <- ltm(RWDQ ~ z1)
```

Note that `z1`, on the right-hand side of the formula interface, is a generic placeholder for the single latent dimension. The item parameters, here shown for the first six items only, are the following:

[4]Note that in order to be consistent with the **ltm** package output, we state the model in terms of difficulty parameters, i.e., $(\theta_v - \beta_i)$.

[5]Due to space restrictions, we do not show the dimensionality assessment for this example. In practice, this should be done prior to fitting the 2-PL.

```
head(coef(fit2pl1))
##            Dffclt     Dscrmn
## wdq_22 -9.4899518  0.1235779
## wdq_23  1.0673713 -0.7324307
## wdq_24  0.3893959 -0.8157724
## wdq_25  1.4559338 -1.5614798
## wdq_26  2.1251683 -0.6637341
## wdq_27  0.5713899 -1.1531809
```

The first column contains the item difficulty parameters β_i. Since the construct is knowledge characteristics, speaking about difficulty does not make sense. A large β_i simply means that the item is located at the upper end of the knowledge characteristics continuum. That is, it allows us to discriminate among persons with high-knowledge characteristics. Conversely, a low item parameter measures knowledge characteristics at its lower end and, correspondingly, discriminates among persons with low knowledge characteristic. The second column shows the item discrimination parameters α_i which reflect the varying ICC slopes.

In the print output above, the item location parameter of item 22 (first row) is unreasonably low, compared to the other parameters. Reasons for such a heavily outlying parameter are that either the algorithm did not converge or the item violates an assumption. Plotting a Princals solution on this dataset (not shown here), suggests that item 22 deviates strongly from the remaining ones. Thus, let us eliminate `wdq_22` and refit the 2-PL model.

```
RWDQ1 <- RWDQ[,-1]
fit2pl2 <- ltm(RWDQ1 ~ z1)
head(coef(fit2pl2))
##            Dffclt    Dscrmn
## wdq_23 -1.0635633 0.7357065
## wdq_24 -0.3869728 0.8219136
## wdq_25 -1.4557177 1.5613090
## wdq_26 -2.1132467 0.6681285
## wdq_27 -0.5699980 1.1569626
## wdq_28 -1.1383853 0.8076892
```

All item parameters look reasonable now.[6] Let us do a final itemfit check by computing a χ^2-based itemfit statistic called Q_1, following Yen (1981). In general, a significant p-value suggests that the item does not fit. But based on the fact that the Q_1 statistic exhibits inflated Type I error rates and considering what we have said at the end of Sect. 4.1.2 on scale construction vs. evaluation, we do not have to use these p-values as our only criterion to keep or eliminate items.

[6] Again, due to space restrictions, we show the first six parameters only.

```
item.fit(fit2pl2)
```

The results (not shown here) suggest that the entire set of items fits; none of the p-values is significant. Thus, no further item elimination steps are needed. The ICCs for the first five items can be produced as follows and are given in Fig. 4.4.

```
plot(fit2pl2, item = 1:5, legend = TRUE)
```

Compared to the Rasch model, the striking difference is that the 2-PL ICCs are allowed to cross. Item locations can be displayed in the same way as in the Rasch model: think of a horizontal line at $p = 0.5$, and then drop vertical lines down to the x-axis from where it intersects with the ICC. We see that `wdq_25` has the largest discrimination parameter, leading to the steepest ICC slope:

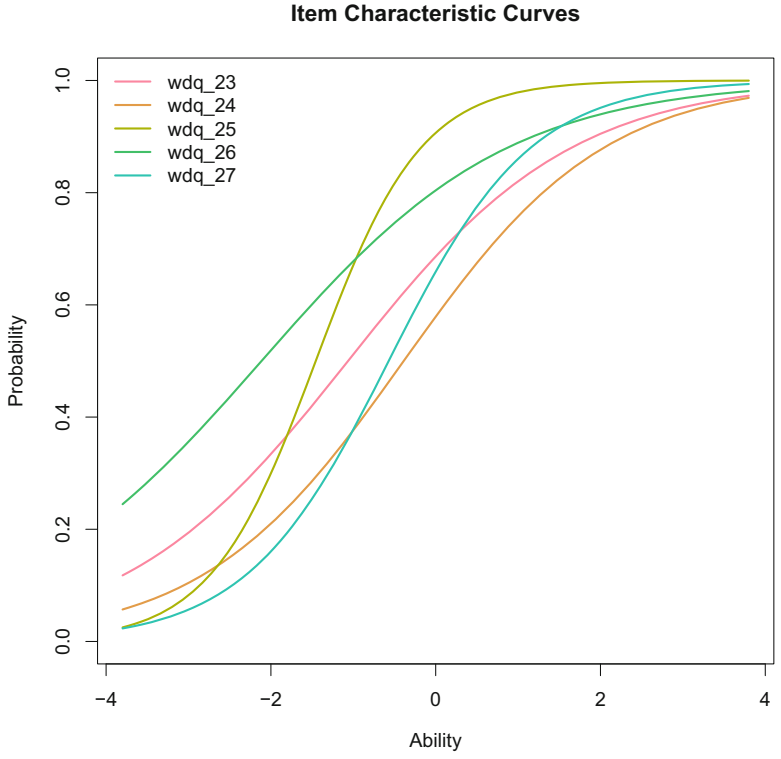

Fig. 4.4 ICCs for the 2-PL fit on WDQ data

```
round(coef(fit2pl2)[1:5, 2], 3)
## wdq_23 wdq_24 wdq_25 wdq_26 wdq_27
##  0.736  0.822  1.561  0.668  1.157
```

"Good" discrimination parameters are typically in the range from 0.8 to 2.5 (de Ayala, 2009). Outside this range, the ICCs are either too flat (such items do not discriminate well along the entire continuum) or too steep (such items discriminate only within a very narrow trait range).

Since the 2-PL fits, we can compute the person parameters:

```
ppars <- ltm::factor.scores(fit2pl2,
  resp.patterns = RWDQ1)$score.dat[, "z1"]
```

As in the Rasch model, these parameters can be used for further analyses. The reason why **ltm** calls the person parameter function `factor.scores` will be explained in Sect. 4.7.1.

4.2.3 Three-Parameter Logistic Model

Let us add another parameter to Eq. (4.2) which brings us to the *three-parameter logistic model* (3-PL; Birnbaum, 1968):

$$P(X_{vi} = 1) = \gamma_i + (1 - \gamma_i)\frac{\exp(\alpha_i(\theta_v - \beta_i))}{1 + \exp(\alpha_i(\theta_v - \beta_i))}. \tag{4.3}$$

The new parameter γ_i reflects the probability of a 1-response on item i due to chance alone and is often referred to as *pseudo-guessing* parameter. With respect to the ICCs, it reflects a lower asymptote parameter. That is, even someone with infinite low ability has a certain probability to score 1 on item i. These probabilities vary across items. With the 3-PL, we create a model that is even more flexible than the 2-PL, but still, unidimensionality and local independence need to be fulfilled.

To illustrate a 3-PL model fit, we reproduce some of the analyses carried out in Wilmer et al. (2012). Within the context of a face recognition experiment, they presented the Verbal Paired-Associates Memory Test (VPMT; Woolley et al., 2008) to the participants. The authors used a 3-PL since their main goal was to score the persons, rather than being strict on the item side in terms of eliminating items. Let us load the dataset and extract the 25 VPMT items:

```
data("Wilmer")
VPMT <- Wilmer[,3:27]
```

We fit a 3-PL using **ltm** and print the item parameter estimates (three per item) for the first six items.

```
fit3pl <- tpm(VPMT)
round(head(coef(fit3pl)), 3)
##       Gussng Dffclt Dscrmn
## vpmt1  0.071 -0.216  1.531
## vpmt2  0.280  1.020  1.736
## vpmt3  0.384  1.527  1.763
## vpmt4  0.103  0.470  1.264
## vpmt5  0.023 -0.299  1.247
## vpmt6  0.135  0.504  1.192
```

In addition to the item locations and the discrimination parameters, we get pseudo-guessing parameters. Again, we can examine the fit of the items via the `item.fit` function (not shown here). The ICCs for the first six items can be plotted as follows:

```
plot(fit3pl, item = 1:6, legend = TRUE)
```

Figure 4.5 shows the resulting ICCs. We see the effect of the γ-parameters in terms of lower ICC asymptotes. Item 3 has the largest pseudo-guessing parameter. As in the 2-PL, the ICCs are allowed to cross due to the α's in the model.

4.3 Unidimensional Polytomous IRT Models

In the previous section, the input data had to be dichotomous. Here we present popular IRT models for polytomously scored items. Again, for all models used in this section, unidimensionality can be assessed using the tools presented in Sect. 4.1.2 prior to fitting the model.

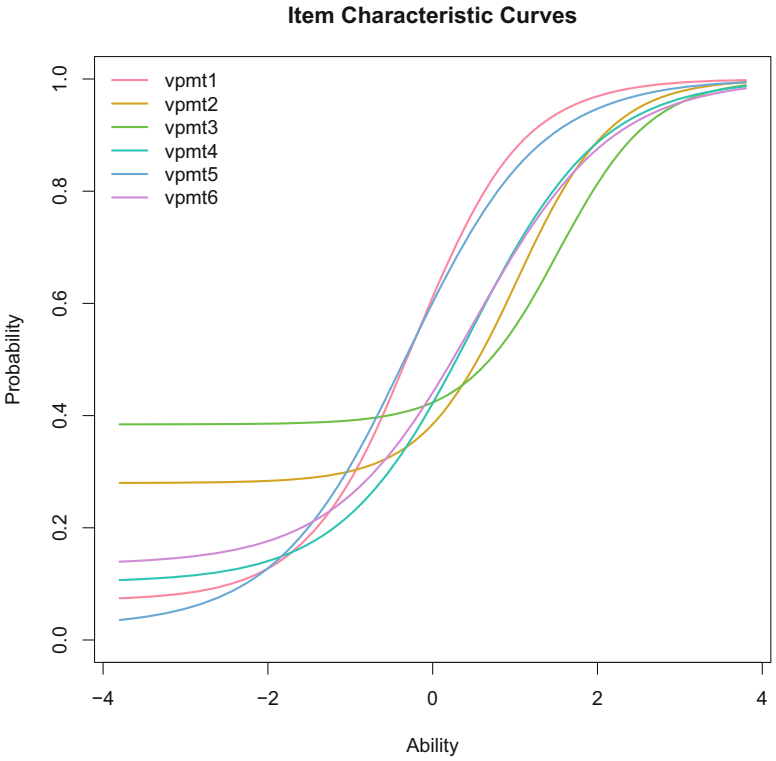

Fig. 4.5 ICCs for the 3-PL fit on VPMT data (first six items only)

4.3.1 Rating Scale Model

The rating scale model (RSM; Andrich, 1978) is one of the simplest IRT models for polytomous data. Let $\mathbf{X}$ be $n \times m$ data matrix with ordinal items. Each item has the same number of categories k ($h = 1, \ldots, k$). The RSM expresses the probability that a person v scores category h on item i as follows:

$$P(X_{vi} = h) = \frac{\exp(h(\theta_v + \beta_i) + \omega_h)}{\sum_{l=0}^{k} \exp(l(\theta_v + \beta_i) + \omega_l)}. \tag{4.4}$$

As in the Rasch model, θ_v denotes the person parameter and β_i the item location parameter, here again written as easiness parameter in order to be consistent with the specification used in the **eRm** package. Each category h gets a *category parameter* ω_h, constant across items. This means that item differences are solely reflected by the shifts in β_i across items. Belonging to the Rasch family, the RSM shares all the desirable Rasch measurement properties. The downside is that the model is pretty strict since the same assumptions need to be fulfilled as in the Rasch model.

To illustrate an RSM fit, we use data analyzed in Bond and Fox (2015). The Children's Empathic Attitudes Questionnaire (CEAQ; Funk et al., 2008) is a 16-item scale to measure empathy of late elementary and middle school-aged children. Each item has three ordered responses: "no" (1), "maybe" (2), and "yes" (3). The sample size is $n = 208$. Three covariates (age, gender, grade) were collected as well. Let us extract the items and set the lowest category to 0 in order to make it **eRm** compatible.

```
data("CEAQ")
itceaq <- CEAQ[,1:16] - 1
```

Now we fit the RSM, estimate the person parameters, and compute some χ^2-itemfit statistics (for which we need the person parameters).

```
fitrsm <- RSM(itceaq)
ppar <- person.parameter(fitrsm)
ifit0 <- eRm::itemfit(ppar)
ifit0
##
## Itemfit Statistics:
##          Chisq  df p-value Outfit MSQ Infit MSQ Outfit t
## ceaq1  210.938 205   0.373      1.024     0.918     0.20
## ceaq2  190.205 205   0.763      0.923     0.897    -0.50
## ceaq3  259.718 205   0.006      1.261     0.966     1.30
## ceaq4  222.146 205   0.196      1.078     0.995     0.72
## ceaq5  167.622 205   0.974      0.814     0.864    -1.58
## ceaq6  191.085 205   0.749      0.928     0.914    -0.76
## ceaq7  162.424 205   0.987      0.788     0.886    -1.51
## ceaq8  146.463 205   0.999      0.711     0.738    -3.41
## ceaq9  181.586 205   0.879      0.881     0.891    -1.31
## ceaq10 380.868 205   0.000      1.849     1.650     3.96
## ceaq11 149.579 205   0.999      0.726     0.753    -3.30
## ceaq12 192.132 205   0.731      0.933     0.980    -0.42
## ceaq13 188.073 205   0.796      0.913     0.924    -0.89
## ceaq14 232.512 205   0.091      1.129     1.206     0.75
## ceaq15 249.571 205   0.018      1.212     1.166     1.88
## ceaq16 184.072 205   0.850      0.894     0.920    -0.93
```

We show the output for the first five items only. We can compare the p-values across items since the degrees of freedom (df) are constant across the tests. Note that these χ^2-statistics exhibit inflated Type I error rates. Other measures reported here are the mean square fit (MSQ) statistics related to the amount of misfit in the original data. They should be within a [0.7, 1.3] interval. *Outfit* statistics result from dividing the χ^2-value by the corresponding df. Due to their sensitivity to outlying

scores, a modified statistic called *infit* is typically preferred. Both infit and outfit can be standardized t-values which should be between -2 and 2. Details on infit/outfit can be found in Bond and Fox (2015, Chapter 12).

For goodness-of-fit assessment, we should look at these measures in combination. In the output above, item 10 has the lowest p-value and high outfit/infit statistics. Let us eliminate this item and refit the model.

```
ind <- match("ceaq10", colnames(itceaq))
itceaq1 <- itceaq[,-ind]
fitrsm1 <- RSM(itceaq1)
ppar1 <- person.parameter(fitrsm1)
ifit1 <- eRm::itemfit(ppar1)
```

In the output table (not shown here), we get a significant p-value for item 15 and considerably high infit/outfit statistics. Let us eliminate this item as well and do another round of model fitting:

```
ind <- match("ceaq15", colnames(itceaq1))
itceaq2 <- itceaq1[, -ind]
fitrsm2 <- RSM(itceaq2)
ppar2 <- person.parameter(fitrsm2)
ifit2 <- eRm::itemfit(ppar2)
```

There is no item left which shows a suspicious combination of significant p-values and extreme infit/outfit values.

Let us double-check the model fit using Andersen's LR-test, which has better inferential properties than the itemfit statistics. We use grade as splitting criterion. Note that we have some missing grade values which we impute using the **mice** package (van Buuren and Groothuis-Oudshoorn, 2011), since we need to have full responses on the split criterion:

```
library("mice")
set.seed(222)
imp <- mice(CEAQ)
gradevec <- complete(imp)$grade
```

Now we binarize the grade variable and compute the LR-test:

```
levels(gradevec) <- c("grade56","grade56","grade78","grade78")
LRtest(fitrsm2, gradevec)
##
## Andersen LR-test:
## LR-value: 23.853
## Chi-square df: 14
## p-value:  0.048
```

Even though the p-value is slightly below 0.05, in conjunction with the itemfit outcomes, it is safe to assume that the data fit the RSM.

Note that another option for testing at an item-category level is to use the `Waldtest` function which computes a p-value for each item-category parameter. There is no clear rule for situations where, for instance, two item-category parameters are significant and two are not significant. Whether such an item should be eliminated or not depends on how strict we want to be when constructing the scale.

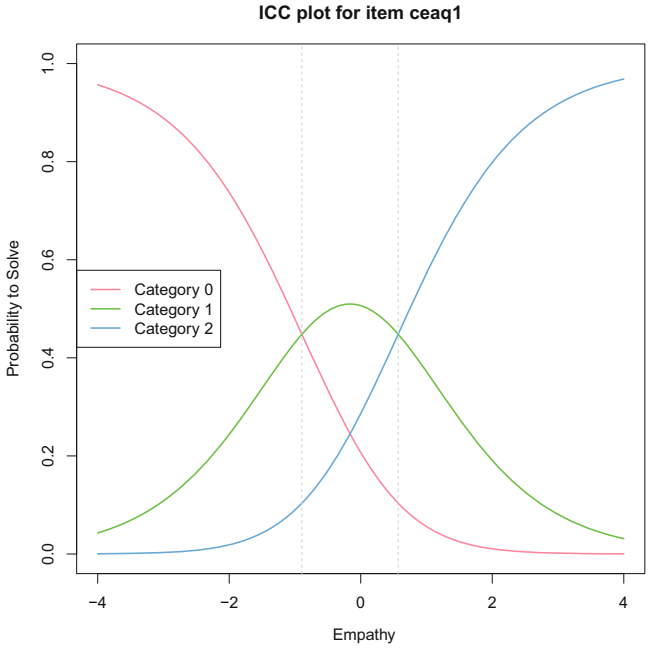

Fig. 4.6 Item-category characteristic curve for first CEAQ item. The vertical lines denote the threshold parameters

In this example, let us proceed with the `fitrsm2` model. The item parameters shown in the `print` or `summary` output are difficult to interpret. Let us convert them into *threshold parameters*:

```
thpar <- thresholds(fitrsm2)
thpar
##
## Design Matrix Block 1:
##        Location Threshold 1 Threshold 2
## ceaq1  -0.16252    -0.89441     0.56937
## ceaq2   0.07933    -0.65255     0.81122
## ceaq3  -0.58160    -1.31348     0.15029
## ceaq4   0.73797     0.00608     1.46986
## ceaq5   0.46689    -0.26500     1.19878
## ceaq6   2.04031     1.30842     2.77219
## ceaq7   0.04391    -0.68798     0.77579
## ceaq8   1.38216     0.65027     2.11405
## ceaq9   1.92245     1.19056     2.65434
## ceaq11  1.71188     0.97999     2.44377
## ceaq12  0.06169    -0.67020     0.79358
## ceaq13  2.26829     1.53640     3.00018
## ceaq14 -0.39327    -1.12516     0.33862
## ceaq16  0.66895    -0.06294     1.40084
```

We get a location parameter for each item and $k - 1$ threshold parameters. The meaning of these threshold parameters is displayed in Fig. 4.6 for item 1 (vertical lines). The threshold parameters reflect the point on the empathy trait continuum where a respondent would switch from scoring 0 to scoring 1 (first parameter) and would switch from scoring 1 to scoring 2 (second parameter). Using the `plotICC` function such curves can be produced for each item. Note that for polytomous models, we get a curve for each item category (*item-category characteristic curves*)[7].

Another plotting option, which summarizes nicely the entire set of parameter estimates, is the person-item map as illustrated in Fig. 4.7.

```
plotPImap(fitrsm2, latdim = "Empathy",
          main = "Person-Item Map CEAQ")
```

As this plot nicely illustrates, item differences occur due to location shifts only. That is, we fit one set of threshold parameters for all items, which is then shifted across the items via the location parameter. This is a main feature of the RSM and will be relaxed in the next section.

Muraki (1990) developed an generalized rating scale model (GRSM) which includes item discrimination parameters. This model can be fitted using the **mirt** package by saying `itemtype="grsm"` in the `mirt` function call.

[7] We continue to use ICC as abbreviation.

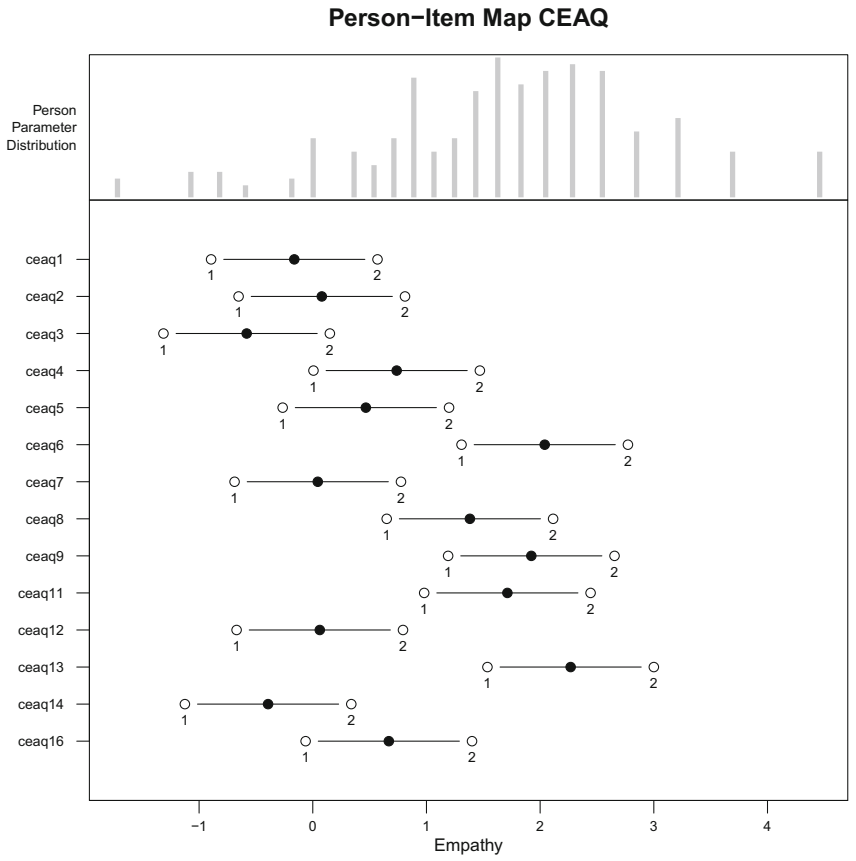

Fig. 4.7 Person-item map for RSM fit on CEAQ data. The top histogram shows the distribution of the person raw scores. The ticks below mark the estimated person parameters. The solid dots are the item location parameters and the hollow dots the threshold parameters

4.3.2 Partial Credit Model and Generalizations

The *partial credit model* (PCM; Masters, 1982) is a generalization of the RSM where we estimate specific item-category parameters for each item. Items do not need to have the same number of categories. Master's motivation was to develop a model suited for partial credit scenarios (e.g., 0 = "totally wrong," 1 = "partially correct," 2 = "almost correct," 3 = "correct"). As usual in IRT models, these scores are taken at an ordinal scale level. The PCM formulation, based on an $n \times m$ data matrix $\mathbf{X}$ where each item i has $h = 1, \ldots, k_i$ categories, looks as follows:

$$P(X_{vih} = 1) = \frac{\exp(h\theta_v + \beta_{ih})}{\sum_{l=0}^{k_i} \exp(l\theta_v + \beta_{ih})} \tag{4.5}$$

Apart from potentially differing numbers of categories, the most striking difference between the PCM and RSM is that each item-category gets its own *item-category parameter* β_{ih}. As the RSM, the PCM belongs to the Rasch family as well and the three Rasch assumptions need to be fulfilled. The PCM can be fitted using the **eRm** package, and itemfit can be assessed in the same way as we did for the RSM.

To illustrate the PCM, we use a dataset from Koller et al. (2017) who analyzed the Adult Self-Transcendence Inventory (ASTI; Levenson et al., 2005), a self-report scale measuring the target construct of wisdom. The ASTI has five subscales: self-knowledge and integration (SI), peace of mind (PM), non-attachment (NA), self-transcendence (ST), and presence in the here-and-now and growth (PG). Let us analyze the PG subscale by means of a PCM. This subscale has six items; four of them are on a 3-point scale and two of them on a 4-point scale.

We fit the PCM using the **eRm** package and, subsequently, convert these parameters into the thresholds for better interpretability.

```
data("ASTI")
PGitems <- ASTI[ ,c(11,14,15,17,18,23)]   ## extract PG items
fitpcm <- PCM(PGitems)
thresholds(fitpcm)
##
## Design Matrix Block 1:
##        Location Threshold 1 Threshold 2 Threshold 3
## ASTI11 -0.25342    -0.95748     0.45065          NA
## ASTI14  0.55114    -0.36856     0.14314     1.87882
## ASTI15 -0.23452    -1.10152     0.63248          NA
## ASTI17  0.36189    -0.03309     0.75686          NA
## ASTI18  0.60182    -0.60759     0.53785     1.87519
## ASTI23  0.26095    -0.36529     0.88720          NA
```

The four items with three response categories get two threshold parameters only. Goodness-of-fit evaluation is not shown here but can be performed in the same manner as for the RSM. Figure 4.8 shows the person-item map which can be produced as follows:

```
plotPImap(fitpcm, latdim = "Presence/Growth",
main = "Person-Item Map ASTI")
```

As opposed to the RSM, the distances between the category thresholds vary across items. Note that sometimes it can happen that two threshold parameters appear to be switched in the PI-map. In this case, a look at the corresponding ICC plot helps. For instance, if the threshold 2 is lower than the threshold 1, this means nowhere on the latent trait a category 2 has an endorsement probability higher than the remaining categories (see de Ayala, 2009, for some examples).

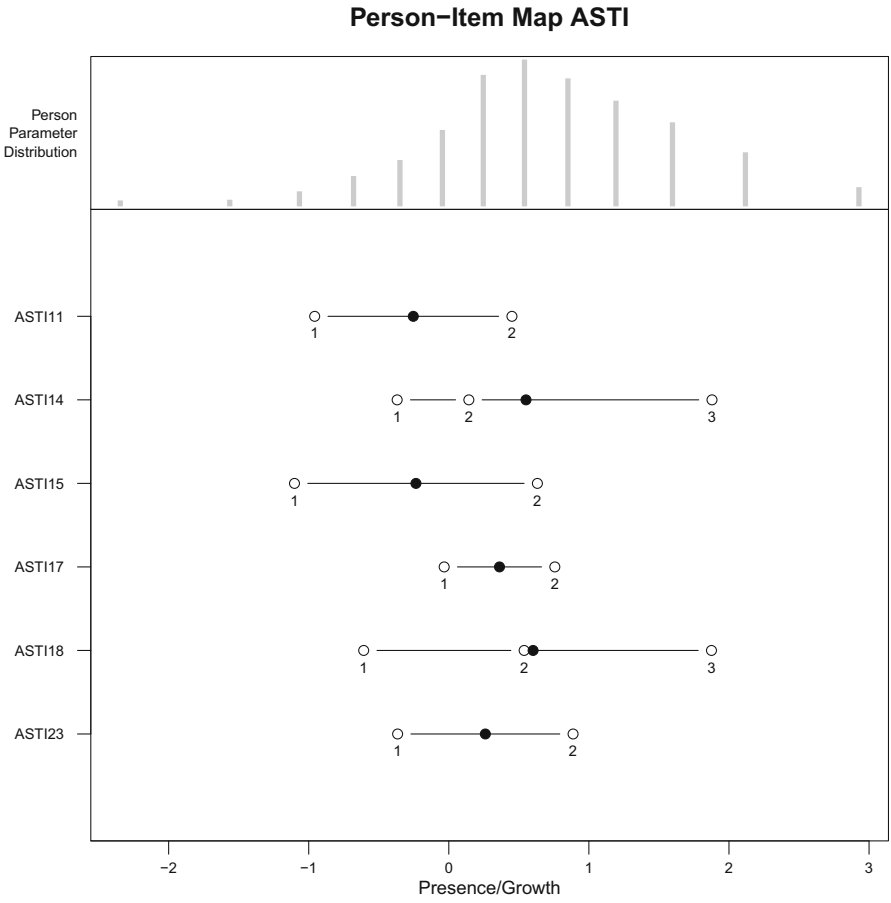

Fig. 4.8 Person-item map for PCM fit on PG subscale

An extension of the PCM was proposed by Muraki (1992): the *generalized partial credit model* (GPCM), which can be expressed by

$$P(X_{vih} = 1) = \frac{\exp(\alpha_i(h\theta_v + \beta_{ih}))}{\sum_{l=0}^{k_i} \exp(\alpha_i(l\theta_v + \beta_{ih}))}. \tag{4.6}$$

Similar two a 2-PL model, it adds item discrimination parameters α_i to the model. Whereas the PCM belongs to the Rasch family, the GPCM does not. The GPCM can be fitted using the **ltm** package via the `gpcm` function. This function allows us to put restrictions on the discrimination parameters. For instance, setting `constraint="rasch"`, we end up with a PCM. Thus, the PCM is nested within a GPCM. We can use this property for constructing an LR-test.

Let us illustrate this PCM/GPCM strategy using the ASTI data once more, but this time using a different subscale: self-transcendence (ST) with seven polytomous items.

```
data("ASTI")
STitems <- ASTI[ ,c(2,4,7,13,16,24,25)]       ## ST items
stpcm <- gpcm(STitems, constraint = "rasch")  ## PCM
stgpcm <- gpcm(STitems)                       ## GPCM
anova(stpcm, stgpcm)                          ## LR-test
##
##  Likelihood Ratio Table
##
##              AIC      BIC  log.Lik    LRT df p.value
## stpcm  18610.23 18705.78 -9286.11        19
## stgpcm 18190.04 18320.79 -9069.02 434.19 26  <0.001
```

We see that the GPCM fits significantly better than the PCM. AIC and BIC are clearly lower for the GPCM compared to the PCM.

The ICCs in Fig. 4.9, produced using `plot(stpcm)` and `plot(stgpcm)`, show the difference between the PCM and the GPCM. The top panels present the ICCs for the first two ST items based on a PCA fit. We see that the ICC slopes within and across items are constant. The ICCs for these two items look very similar, but they are not exactly the same. The bottom panels contain the ICCs from the GPCM fit. The slopes vary across items; within each item, they are the same. This variation is due to the α_i's added to the model.

4.3.3 Graded Response Model

Another popular polytomous IRT model, similar to the GPCM in terms of including discrimination parameters, is the *graded response model* (GRM; Samejima, 1969):

$$P(X_{vi} \geq h) = \frac{\exp(\alpha_i(\theta_v - \beta_{ih}))}{1 + \exp(\alpha_i(\theta_v - \beta_{ih}))} \tag{4.7}$$

By looking at the left-hand side of the equation, we see that the model is formulated in terms of cumulative logits, whereas the GPCM was formulated by means of adjacent logits.[8] Thus, the GRM estimates the probability for scoring category h or higher. The β_{ih} parameter is often referred to as *category boundary location*. As in the PCM/GPCM, the items can have different numbers of categories.

[8]For connections between the GRM and the GPCM, see Ostini and Nering (2005).

A GRM can be fitted using the **ltm** package. We illustrate the GRM on the self-transcendence (ST) subscale from the ASTI data, already used in the GPCM fit

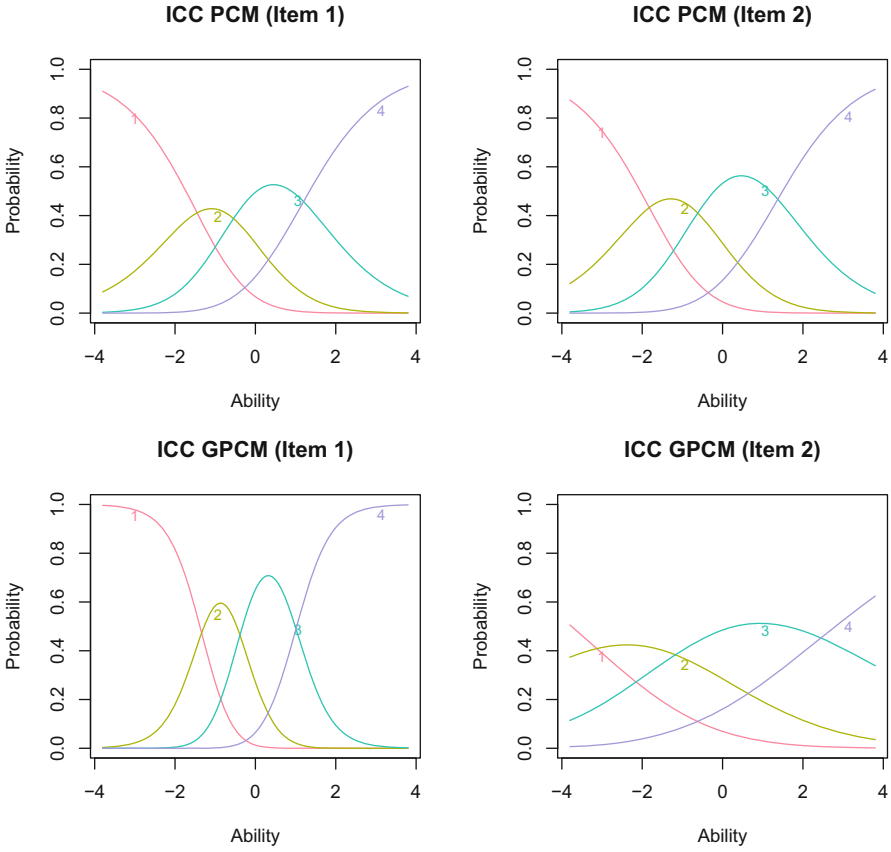

Fig. 4.9 Top panels: item-category curves for the first two items (PCM fit). Bottom panels: item-category curves for the same two items (GPCM fit)

above. The first line fits the model according to Eq. (4.7) with item discrimination and category boundary parameters. The second line estimates the person parameter.

```
fitgrm <- grm(STitems)
ppargrm <- ltm::factor.scores(fitgrm)
```

Figure 4.10 shows two plots, both related to the first item. The left panel, produced using `plot(fitgrm, type="OCCu")`, displays what is actually fitted: the cumulative logits (called *operation characteristic curves*) from Eq. (4.7).

The first curve (from left to right) reflects the probability for scoring 1 or higher, the second curve the probability for scoring 2 or higher, and the third curve the probability for scoring 3. The right-hand panel shows the ICC version of it, produced using `plot(fitgrm, type = "ICC")`.

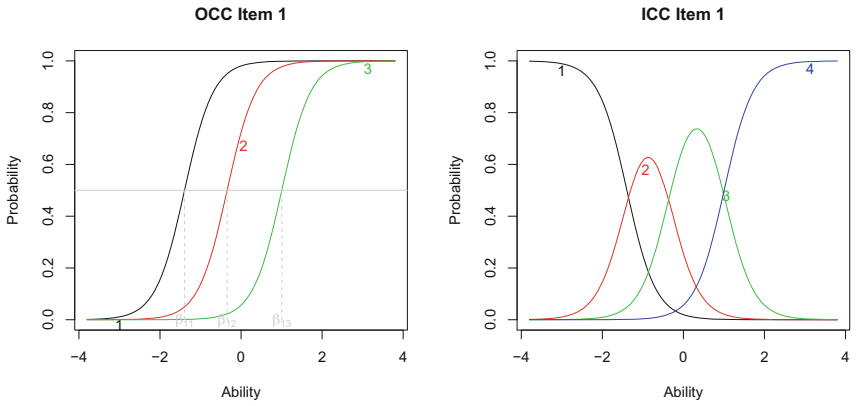

Fig. 4.10 Left panel: operation characteristic curves with corresponding GRM boundary parameter estimates (first item). Right panel: item-category curves resulting from GRM fit (first item)

4.3.4 Nominal Response Model

The last unidimensional polytomous model we consider here is the *nominal response model* (NRM; Bock, 1972). As the name already suggests, this model abandons the ordinality restriction on the item input categories. It is attractive to apply on multiple choice items or if there are response formats of the form "yes," "no," and "don't know." It is more general than the GRM/GPCM in the sense that each item-category gets its own individual discrimination parameter, in addition to the item-category location. The probability that person v scores category h on item i is expressed by

$$P(X_{vi} = h) = \frac{\exp(\alpha_{ih}(\theta_v - \beta_{ih}))}{1 + \exp(\alpha_{ih}(\theta_v - \beta_{ih}))}. \tag{4.8}$$

Here, α_{ih} is the *item-category discrimination parameter*. This implies that, neither within nor across items, the item-category characteristic curves have to be parallel.

In R, the NRM can be fitted using the **mirt** package. The dataset we use contains a modified version of the Wilson-Patterson conservatism scale (Wilson and Patterson, 1968). We use the first 15 "conservative" items. Each item has three response categories: 1 = "approve," 0 = "disapprove," and 2 = "don't know."

Unidimensionality assessment for nominal responses requires some adaption of the tools presented in Sect. 4.2. We can use a nominal Princals version which gives us an insight into the dimensionality structure. The top panel of Fig. 4.11 shows the Princals loadings plot. We eliminate the four items pointing to the bottom-left. On the remaining items, we fit a Homals solution. Simply speaking, Homals is a more general model than Princals where the focus is on scoring the categories, and it is described in detail in Sect. 8.3. Homals gives us insight into the category associations (see bottom panel of Fig. 4.11). The first dimension discriminates between responses 1 and 2; the second dimension is mostly determined by the 0 responses. Homals and Princals can be fitted as follows:

```
library("Gifi")
data("WilPat")
wpit15 <- WilPat[,1:15]
wpiprin <- princals(wpit15, ordinal = FALSE)
elim <- c("Nationalism", "Patriotism", "ChurchAuthority",
          "Obedience")
ind <- match(elim, colnames(wpit15))
wpitnew <- wpit15[, -ind]
wpihom <- homals(wpitnew)
```

At this point, we are ready to fit the NRM using the **mirt** package. The second argument in the `mirt` call refers to the dimensionality of the model.

```
library("mirt")
nrmwp <- mirt(wpitnew, 1, itemtype = "nominal")
```

The ICCs can be obtained using the `itemplot` function. Figure 4.12 presents ICCs for four interesting items. The further left a person is placed on the trait, the more conservative he/she is. Genetically modified foods are only approved by very conservative people. The remaining persons are most likely against it or say "don't know." For the capitalism, free market, and lower taxes items, the probability for scoring 0 ("disapprove") is low throughout the trait. Lower taxes are approved over a wide range of the continuum; very liberal persons most likely say "don't know." As we see, the NRM gives a highly detailed insight into the item-category/person behavior. Item-category discriminations can vary drastically within and across items, as in our example.

In terms of goodness-of-fit assessment, the **mirt** offers a highly attractive function. Here we demonstrate it for the NRM, but since **mirt** allows us to fit basically every model considered so far, these test criteria are generally applicable (also for multidimensional IRT models described further below).

```
M2(nrmwp)
##              M2 df         p      RMSEA RMSEA_5   RMSEA_95
## stats 24.97443 22 0.2982992 0.01297576       0 0.03322548
##              TLI       CFI
## stats 0.9895378 0.9947689
```

The first set of outputs is related to the M_2 statistic proposed by Maydeu-Olivares and Joe (2005). It is a limited information version of a χ^2-test involving observed probabilities and model probabilities. The nonsignificant p-value suggests that the model fits. The RMSEA (including 90% CI), TLI, and CFI are borrowed from confirmatory factor analysis (CFA) and structural equation models (SEM), respectively (see Sect. 2.4.1 for details). For the CFI, we can use the same 0.95 fit cutoff as in CFA/SEM. For the RMSEA, the CFA/SEM cutoff was 0.05 for a good fitting model. In IRT, it is suggested to use $0.05/k$ (with k being the number of categories per item) as fit cutoff. In our example $k = 3$, thus, the RMSEA should be smaller than 0.017. This is the case in our example. Overall we can conclude that the model fits. Further details on these statistics can be found in Maydeu-Olivares (2015).

4.4 Item and Test Information

In many applications, it is of interest in which area of the trait an item is particularly informative. In other words, what is the degree to which an item reduces the uncertainty in estimation of a person's trait value? This concept is called *item information*. All IRT packages used so far provide corresponding plot functions for item information. Here we sketch it for the NRM fit above using the **mirt** package. The following function call generates the item information curves for each of the 11 conservatism items (see Fig. 4.13):

```
plot(nrmwp, type = "infotrace", main = "Item Information")
```

From the information curve, we see that "privatization" is highly informative within a narrow range at the center of the trait (average conservative respondents). "Lower taxes" is not a very informative item; ultimately no one likes to pay taxes. The curve has a little bump located in the liberal respondents' area. The remaining curves can be interpreted in a similar fashion.

These item information curves can be aggregated. This leads us to the concept of *test information*.

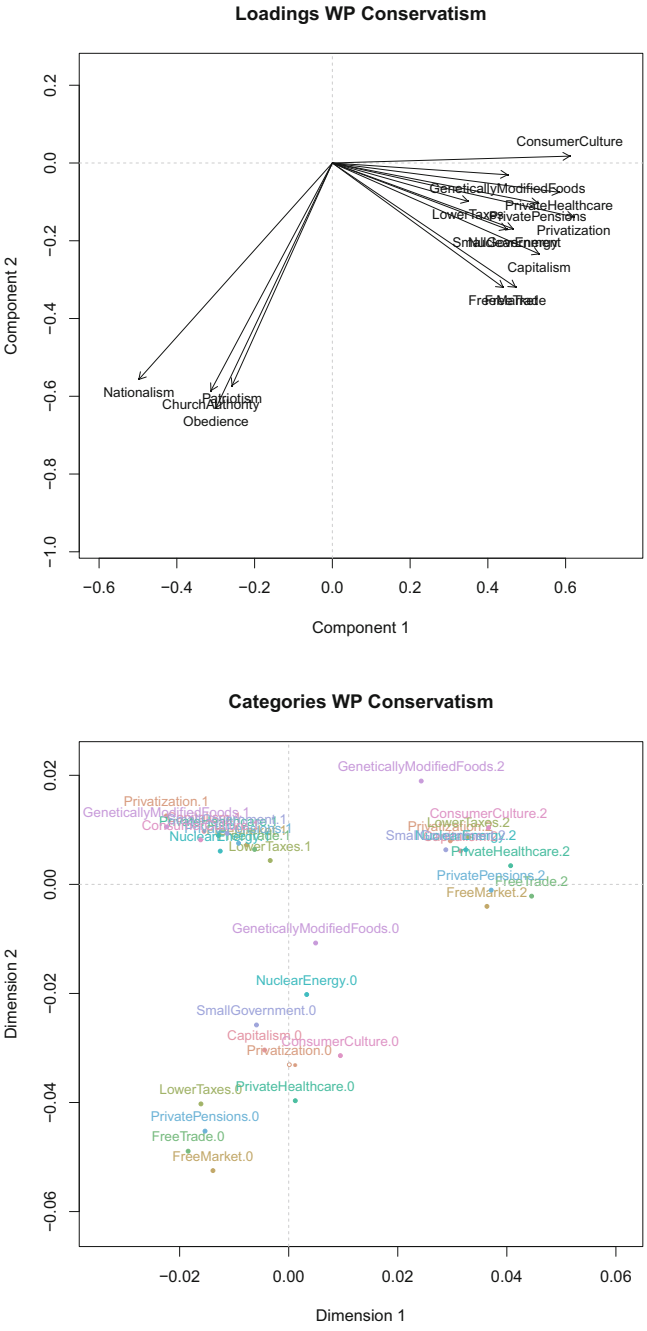

Fig. 4.11 Top panel: nominal Princals loadings plot. Bottom panel: 2D Homals joint category plot after eliminating four misfitting items through Princals

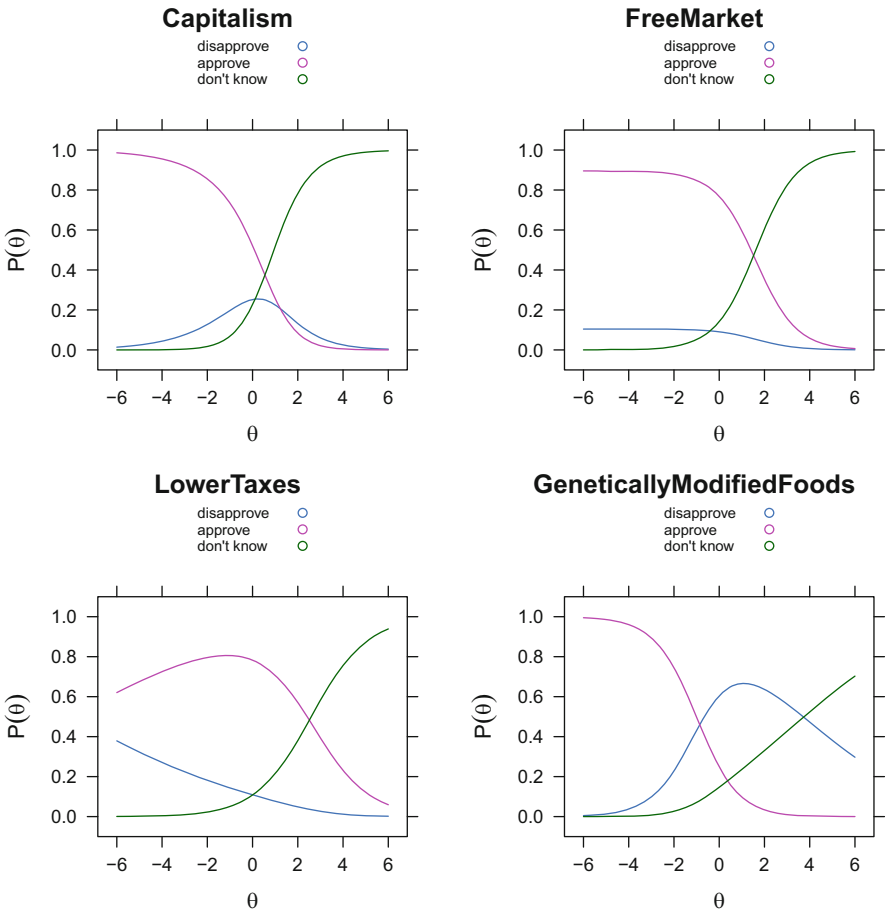

Fig. 4.12 NRM item-category characteristic curves for four selected items. The trait represents conservative (negative direction) and liberal (positive direction)

```
plot(nrmwp, type = "info")
```

This plot (not shown here) tells us in which trait area our entire scale is informative and thus able to assess a person's location on the conservatism trait with good precision.

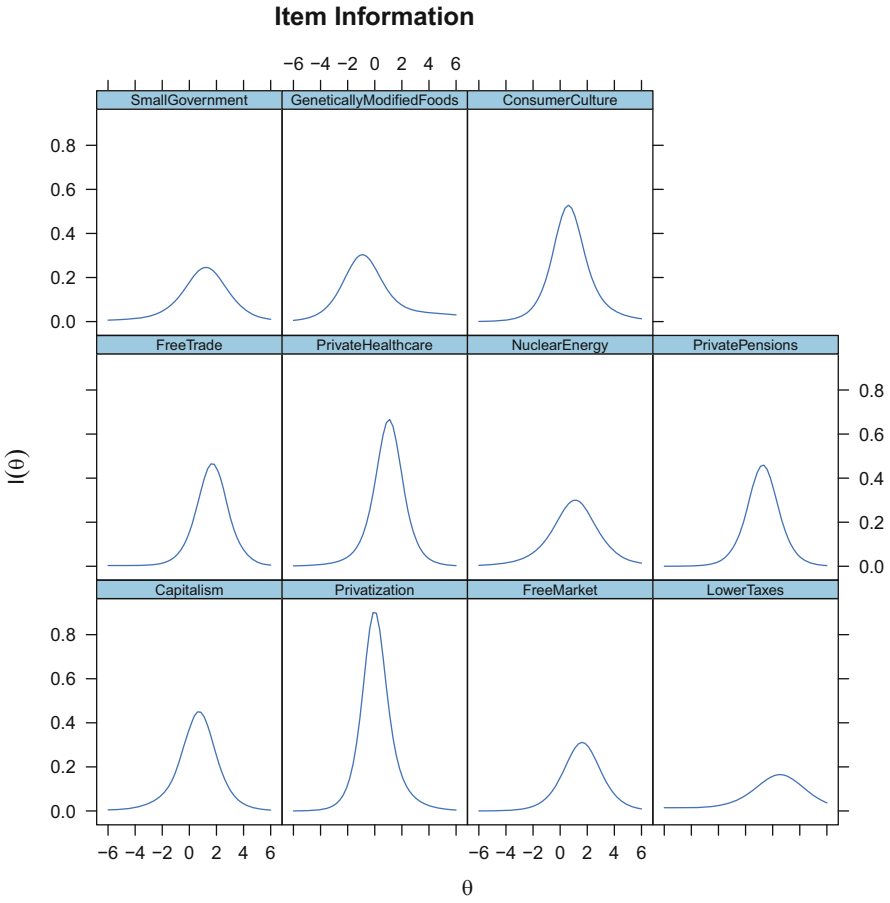

Fig. 4.13 Item information curves resulting from the NRM on Wilson-Patterson conservative items

4.5 IRT Sample Size Determination

How many persons do we need in an IRT analysis? While it is tempting to dig out some IRT sample size rules of thumb,[9] the best way to assess n is to set up a Monte Carlo simulation. Such a simulation starts with generating many data matrices based on a population parameter specification of a particular IRT model of interest. For each of these matrices, we fit the IRT model of choice and check how well the estimated parameters recover the population parameters. This is repeated for varying sample sizes n.

[9] In the book by de Ayala (2009), almost each chapter gives some guidelines of how large a calibration sample should be with respect to a particular IRT model.

A general purpose package which supports us to perform such simulation tasks is **SimDesign** (Chalmers, 2017). A Monte Carlo IRT sample size simulation involves the following four steps:

1. Specify the population parameters of an IRT model.
2. Simulate data according to the population parameterization ("generate" step).
3. Fit the model on the data ("analyze" step).
4. Determine how well the population parameters are recovered ("summarize" step).

Let us walk through this process step-by-step. Assume we have $m = 20$ binary items and our IRT model of choice is a 2-PL model. We vary the sample size n systematically from 50 to 300.

```
library("SimDesign")
m <- 20
n <- c(50, 75, 100, 150, 200, 300)
design <- as.data.frame(n)
set.seed(222)
poppars <- rbind(alpha = round(rlnorm(m, 0, 0.25), 2),
                 d = round(rnorm(m), 2))
```

The last line draws the population parameters. For the slopes α_i, the log-normal distribution is very attractive since this distribution does not allow for negative values. We fix the spread (sd on a log-scale) to a value of 0.25 which gives us reasonable α_i values from 0.8 to 2.5.

According to Eq. (4.2), the second parameter would be the difficulty. Since we are using the **mirt** package for the simulation, we need to consider that **mirt** parameterizes the model a bit differently: $d_i = -\alpha_i \beta_i$ (more details on this parameterization will follow in Sect. 4.7.1). Thus, we could draw the β_i's apply this conversion formula to get d_i, or we draw the d_i's right away. The latter is done above. For the d_i's, drawing from a standard normal distribution is a reasonable choice.

In the second step, we generate the data by means of the following function:

```
irtGenerate <- function(condition, fixed_objects = FALSE) {
  n <- condition$n
  a <- fixed_objects['alpha', ]
  d <- fixed_objects['d', ]
  dat <- simdata(a, d, n, itemtype = '2PL')
  return(dat)
}
```

Eventually, `condition` will be our design variable from above (i.e., it varies n), and the `fixed_objects` will include the population parameters. For generating 0/1 data matrices, we use the `simdata` function from **mirt**. We need to provide the parameter vectors and the sample size as input. The person parameter vector is drawn internally from $N(0, 1)$. At the end of this stage, we have a binary data matrix.

The next step is "analyze," that is, we fit the 2-PL on the data matrix generated above by means of the following function:

```
irtAnalyze <- function(condition, dat, fixed_objects = NULL) {
    mod <- mirt(dat, 1, itemtype = '2PL', verbose = FALSE)
    simpars <- coef(mod, simplify = TRUE, digits = Inf)$items
    irtpars <- c(a = simpars[,1], d = simpars[,2])
    return(irtpars)
}
```

The core of this function is the `mirt` call which fits the 2-PL. Whether the resulting parameter estimates recover the population parameters depends on the sample size. If n is too small, there will be substantial differences. The remaining lines in this function are just parameter cosmetics in order to get the object ready for the final step. We repeat these two steps L times for each given n value. This implies that we generate L replication datasets, and for each one, we fit the 2-PL. Here we use $L = 100$ in order to keep the running time low; in real-life applications, it is suggested to work with a larger L. Eventually for each item i, we get 100 α and 100 d parameters, for each given n.

At the end, this information needs to be summarized. In this last step, we compare the IRT estimates with the population parameters. How close they are to each other depends on n: the larger n, the closer they will be. A common measure to judge this parameter recovery is the *root mean squared error* (RSME). For instance, for the first discrimination parameter, the RMSE is

$$RMSE = \sqrt{\frac{\sum_{l=1}^{L}(\alpha_{1l} - \alpha_1^*)^2}{L}}, \tag{4.9}$$

with α_{1l} as the discrimination parameter for item 1 in replication l and α_1^* as the population parameter. We apply this formula to each parameter separately which gives us a RMSE for each α and for each d parameter. Obviously, the smaller the RMSE, the better the parameters are recovered. We repeat this process for each n.

```
irtSummarize <- function(condition, results,
                         fixed_objects = NULL) {
  apop <- fixed_objects['alpha', ]
  dpop <- fixed_objects['d', ]
  simrmse <- RMSE(results, c(apop, dpop))
  out <- c(RMSE = simrmse)
  return(out)
}
```

Other measures such as the *bias* (i.e., $\sum_{l=1}^{L}(\alpha_{1l} - \alpha_1^*)/L$) are often reported as well, for which the **mirt** package provides a corresponding `bias` function.

Now we are ready to actually run the simulation using the population parameters from above and the three functions: generate, analyze, and summarize.

```
set.seed(222)
simres <- runSimulation(design, replications = 100,
  parallel = TRUE,  generate = irtGenerate,
  analyse = irtAnalyze, summarise = irtSummarize,
  packages = c('mirt'), fixed_objects = poppars)
simres
```

This function call collects the results in a data frame, subject to further plotting. Let us pull out the RMSEs for the discrimination parameters and plot the parameter trajectories across varying sample sizes (see Fig. 4.14). Note that we use $\log(RMSE)$ on the y-axis for better visualization.

```
colind <- grep(".a.", colnames(simres))
sima <- as.data.frame(simres[, colind])
nvec <- as.numeric(levels(simres$n))
matplot(nvec, log(sima), type = "l", col = 1, lty = 1,
  ylab = "log(RMSE)", xlab = "sample size",
  main = "2-PL Monte Carlo", xaxt = "n")
axis(1, at = nvec)
```

The average RMSEs for each n are

```
meanRMSE <- rowMeans(sima)
names(meanRMSE) <- n
round(meanRMSE, 2)
##   50   75  100  150  200  300
## 0.84 0.49 0.39 0.31 0.25 0.20
```

By looking at these means in conjunction with the trajectories, we see that an n lower than 100 is certainly not a good idea. The RMSE becomes stable in the 150–200 area. This would be our n of choice. The same type of post simulation inspection can be done for the d_i parameters.

In this section, we focused on a simple 2-PL, but we can carry out this type of simulation for any other IRT model as well. For polytomous models, item-category population parameters need to be specified/drawn. The `mirt` call in the analyze step can be replaced by corresponding functions from **eRm** or **ltm** call. An example of a Monte Carlo sample size determination for a more complex model (i.e., a multidimensional GRM) can be found in Jiang et al. (2016).

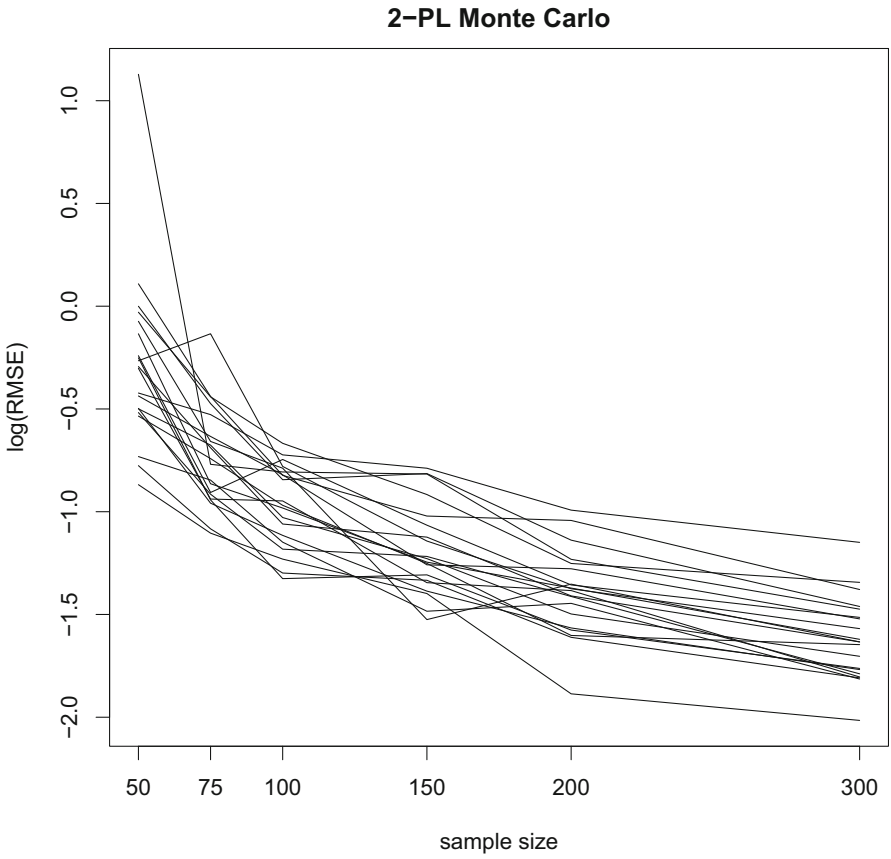

Fig. 4.14 Log-RMSE trajectories for 2-PL discrimination parameters (20 items) for varying sample sizes n. Each line represents a discrimination parameter trajectory across varying n

4.6 Differential Item Functioning

Differential item functioning (DIF; see Osterlind and Everson, 2009) is concerned with different item behavior across person subgroups. Corresponding differences in the item parameters are sometimes referred to as *item bias*. This bias is tied to the concept of fairness. The aim of DIF analysis is to identify (i.e., "flag") such biased items. DIF items can then be reconsidered by experts during the scale construction process and potentially reformulated or eliminated. If our aim is to score persons on a well-established scale, the person parameters for the flagged items can be adapted without removing these items. We can distinguish between two forms of DIF:

- *Uniform DIF*: the ICCs are shifted in location across subgroups, but they remain parallel (i.e., a group main effect).
- *Nonuniform DIF*: the ICCs across subgroups are shifted, and they cross (i.e., an interaction effect between group and the trait).

Basic DIF detection methods using the **difR** package (Magis et al., 2010) are illustrated in Finch and French (2015). Here we focus on two modern, flexible approaches: DIF detection using logistic regression and tree-based DIF assessment.

4.6.1 Logistic Regression DIF Detection

The idea of this approach (Zumbo, 1999) is to specify a set of logistic regression equations and predict the original item responses from the person parameters $\boldsymbol{\theta}$ and the external grouping variable $\mathbf{z}$. This method works for dichotomous as well as polytomous responses and allows for a single grouping variable with multiple categories. The following set of *proportional odds models* is formulated (Choi et al., 2011):

$$\begin{aligned} M_1: &\quad \text{logit}(P(\mathbf{x}_i)) = \tau_0 + \tau_1\boldsymbol{\theta} \\ M_2: &\quad \text{logit}(P(\mathbf{x}_i)) = \tau_0 + \tau_1\boldsymbol{\theta} + \tau_2\mathbf{z} \\ M_3: &\quad \text{logit}(P(\mathbf{x}_i)) = \tau_0 + \tau_1\boldsymbol{\theta} + \tau_2\mathbf{z} + \tau_3\boldsymbol{\theta}\mathbf{z} \end{aligned} \tag{4.10}$$

We are interested in comparing the following models via the LR-principle:

- M_2 vs. M_1: if significant, we have uniform DIF.
- M_3 vs. M_2: if significant, we have nonuniform DIF.

As an alternative to the LR-test, we can also look at differences in pseudo-R^2 values (e.g., McFadden's R^2) with corresponding effect sizes: < 0.13 "negligible," 0.13–0.26 "moderate," and > 0.26 "large" (Zumbo, 1999).

This approach, including Monte Carlo simulated empirical criteria, is implemented in the **lordif** package (Choi et al., 2011). Internally, this approach fits a

GRM (default) or a GPCM using **mirt**, uses Stocking-Lord equating for the item parameters, and estimates person parameters based on DIF and non-DIF items.

As an example, we use a dataset from Vaughn-Coaxum et al. (2016) on youth depression. The 26 items come from the Children's Depression Inventory (CDI); each item is scored on a scale from 0 to 2. The authors were interested in DIF analyses on an external race variable (four categories). Note that the aim was not to eliminate items from the CDI, which is a well-established scale. Rather, the authors wanted to identify DIF items (which already gives useful substantive information) and score all individuals in a "fair" way by means of group-specific person parameter estimates for items flagged as DIF.

```
library("lordif")
library("MPsychoR")
data("YouthDep")
cdi <- YouthDep[,1:26]        ## extract CDI items
cdiDIF <- lordif(cdi, YouthDep$race, criterion = "Chisqr")
```

In total, 20 out of 26 items are flagged as DIF. Let us print out the p-values of the LR-tests for the first three items:

```
cdiDIF$stats[1:3, 1:5]
##   item ncat chi12  chi13  chi23
## 1    1    2 0.352 0.3799 0.3718
## 2    2    2 0.000 0.0000 0.2084
## 3    3    2 0.000 0.0000 0.0022
```

We see that for the first item, none of the LR-χ^2 values is significant. In fact, item 1 was not flagged. For the second item, χ^2_{12} (i.e., M_2 vs. M_1) is significant, whereas χ^2_{23} (i.e., M_3 vs. M_2) is not significant. Thus, the second item has uniform DIF. For the third item, all p-values are significant; we have the case of nonuniform DIF.[10] Corresponding plots can be produced as follows:

```
plot(cdiDIF, labels = c("White", "Black", "Asian", "Latino"))
```

Figure 4.15 shows the *item true score functions* (i.e., the GRM model probabilities related to the original 0–2 scale) for the first two DIF items. For the items shown in Fig. 4.15, the true score functions result in simple ICCs since two categories

[10] χ^2_{13} is an additional LR-test for M_1 vs. M_3.

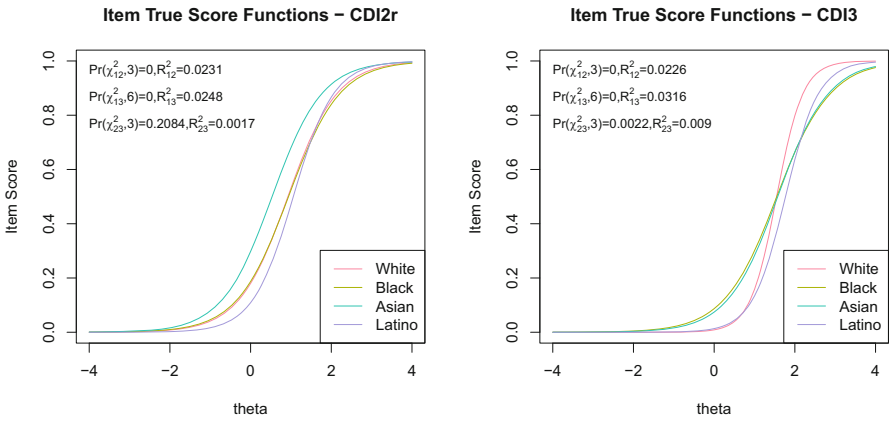

Fig. 4.15 Left panel: ICC for the first DIF item (uniform DIF). Right panel: ICC for the second item (nonuniform DIF). Each panel contains information on the p-values of the corresponding LR-tests according to the models in Eq. (4.10) as well as differences in McFadden's R^2

were collapsed by `lordif` due to an insufficient number of responses within each subgroup (details follow below). We see that `CDI2r` has uniform DIF, whereas `CDI3` has nonuniform DIF.

Let us print out the GRM parameters (discrimination, category boundaries) for the six non-DIF items (non-DIF) and the first DIF item (`I2`):

```
head(cdiDIF$ipar.sparse, 10)
##              a        cb1      cb2
## I1    2.055572 1.5197578       NA
## I9    1.942808 1.6169260       NA
## I12   1.224337 0.3175138 2.674436
## I15   1.424712 1.1699027 2.507799
## I19   2.107288 1.1153563       NA
## I21   1.129338 1.9135221       NA
## I2.1  1.638530 0.9251430       NA
## I2.2  1.564181 0.9395047       NA
## I2.3  1.611274 0.5323336       NA
## I2.4  1.987255 1.0579461       NA
```

Note that for some items, there is only one category boundary. This results from the fact that there were not enough observations in a particular category (here, category 2) for parameter estimation. For such cases, **lordif** collapses categories automatically. Item 2 was flagged as DIF. We get four sets of discrimination/boundary parameters, one of each race category.

The calibrated, group-specific person parameter vector can be extracted using

```
ppar <- cdiDIF$calib.sparse$theta
```

Based on the DIF subgroup structure, they are fairly scored, are on the same scale, and can be subject to further analysis (see Vaughn-Coaxum et al., 2016).

4.6.2 Tree-Based DIF Detection

Decision trees aim to classify/predict a response variable from a set of predictors. They find predictor splits in a recursive manner with the goal to maximize classification/prediction performance in the terminal nodes (see James et al., 2013, for an introduction). *Model-based partitioning trees* (Zeileis et al., 2008) follow the same concept with the difference that the "response" is an entire model. This leads to a set of splits with heterogeneous parameter estimates across the terminal nodes.

Strobl et al. (2015) adapted this idea to the Rasch world and implemented it in the **psychotree** package. Based on multiple external metric or categorical variables, the algorithm tries to find splits for which the item parameters differ (i.e., it looks for DIF). If no split is found, there is no DIF.

We explore this technique by means of a mathematics exam dataset (13 binary items). We use the following seven covariates, subject to DIF analysis: number of items solved (`nsolved`), number of online test exercises (`tests`), gender (`gender`), two types of degrees (`study`), semester enrolled (`semester`), number of times the exam has been attempted (`attempt`), and whether the students were in the first or second batch (`group`). The data are included in the **psychotools** package (Zeileis et al., 2016). Below we eliminate persons with 0/full responses and define the factor levels for the covariates properly. Note that the item responses, in order to be processed by the `raschtree` function, have to be provided as object of class `"itemresp"`.[11]

```
library("psychotree")
library("psychotools")
data("MathExam14W")
itmath <- as.list.data.frame(MathExam14W$solved)
```

(continued)

[11] The `"MathExam14W"` dataset from the package is already prepared that way. Here we bring it back to the standard data frame form and illustrate how to get them in shape for the tree function call.

```
covars <- MathExam14W[,3:9]
mex <- data.frame(solved = itemresp(itmath), covars)
mex <- subset(mex, nsolved > 0 & nsolved < 13)
mex$tests <- ordered(mex$tests)
mex$nsolved <- ordered(mex$nsolved)
mex$attempt <- ordered(mex$attempt)
```

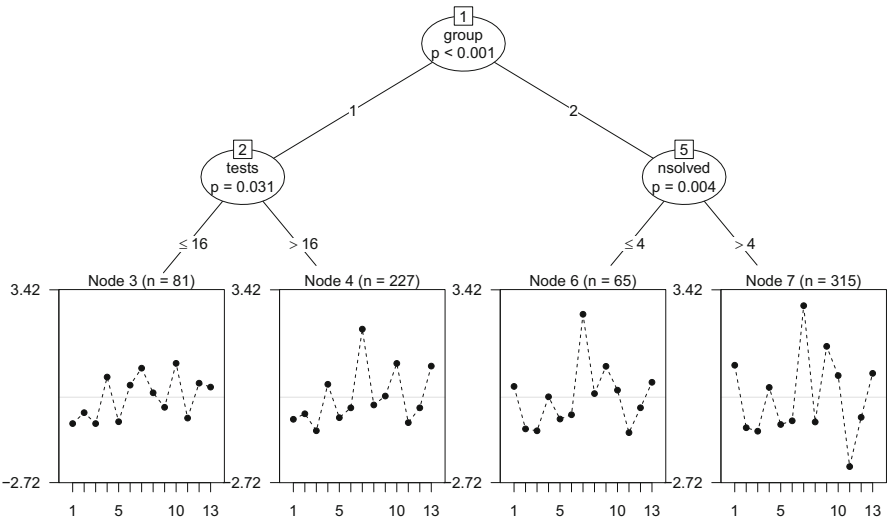

Fig. 4.16 Rasch partitioning tree for math exam dataset. The dot trajectories in the terminal nodes denote the 13-item parameters for the corresponding student subgroup of size *n*

In total, we have 688 students in our sample. We use all seven covariates and let the algorithm do the rest. The item responses are on the left-hand side of the formula and the covariates to the right.

```
set.seed(1)
mrt <- raschtree(solved ~ group + tests + nsolved + gender +
  attempt + study + semester, data = mex, vcov = "info",
  minsize = 50, ordinal = "l2", nrep = 1e5)
```

The fitted tree structure can be plotted as follows (see Fig. 4.16):

```
plot(mrt)
```

The first split is according to the student group variable (there were two batches of students). Students in the first batch were again split according to the test variable (i.e., the number of online test exercises solved correctly prior to the written exam). Note that the algorithm splits this metric variable at a value of 16. Depending on whether batch 1 students solved more or less than 16 exercises correctly, the item parameters differ. For the students in batch 2, a split according to the student's raw score was obtained (four or less items correct vs. more than four items correct). The item parameter profiles (items on the x-axis, difficulty parameters on the y-axis) are given at the bottom. We see clearly deviating patterns in the item parameters across the nodes.

Let us extract the item difficulty parameters for the first four items. The rows refer to the node in the plot and the columns to the corresponding item.

```
round(itempar(mrt)[,1:4], 2)
##   solvedquad solvedderiv solvedelasticity solvedintegral
## 3      -0.84       -0.49            -0.84           0.64
## 4      -0.70       -0.53            -1.07           0.41
## 6       0.34       -1.01            -1.07           0.01
## 7       1.02       -0.97            -1.08           0.31
```

This partitioning tree concept has been extended to the RSM and PCM as well (Komboz et al., 2018). Corresponding functions can be found in **psychotree**.

Another DIF package allowing for multiple covariates is **DIFlasso** (Tutz and Schauberger, 2015) which uses a *lasso* penalty approach on the covariates for DIF detection.

4.7 Multidimensional IRT Models

All IRT models considered so far were unidimensional, that is, the items were supposed to measure a single underlying latent trait. In this section, we relax this restriction by considering multiple traits and model the item responses via *multidimensional IRT models* (MIRT; see Reckase, 2009). Before presenting various specific modeling approaches, let us have a look at some relationships between IRT and factor analysis (FA).

4.7.1 IRT and Factor Analysis

EFA and CFA, as presented in Chap. 2, are designed for metric input variables (unless we use specific correlation coefficients suited for categorical data). The main outcomes are factor loadings and factor scores. IRT models are designed for categorical data with item-category parameters (location, discrimination) and person parameters as main outcomes. However, it has been found that there is a strong parametric relationship between FA and IRT (Takane and De Leeuw, 1986). Let us consider a simple unidimensional 2-PL, as given in Eq. (4.2), rewritten in logit form:

$$\text{logit}(P(X_{vi})) = \alpha_i(\theta_v - \beta_i)). \tag{4.11}$$

We can re-parameterize this model as

$$\text{logit}(P(X_{vi})) = \alpha_i\theta_v + d_i. \tag{4.12}$$

The second equation reflects a factor analytic intercept-slope representation of the 2-PL. The discrimination parameter α_i can be interpreted as loading (i.e., slope as reflected in the ICC plot). The parameter d_i is the *intercept*. From the first equation, it follows that $d_i = -\beta_i\alpha_i$. Note that both equations fit the same model, they are just parameterized differently. Consequently, the parameters θ_v are the same, regardless which expression we use. Within an IRT context (Eq. (4.11)), we call them "person parameters," whereas within a FA context (Eq. (4.12)), we call them "factor scores."

To illustrate, we repeat the 2-PL analysis from Sect. 4.2.2 which involved dichotomous items on knowledge characteristics from the WDQ (first item eliminated due to misfit). We use **ltm** package and request the two different parameterizations:

```
library("MPsychoR")
library("ltm")
data("RWDQ")
RWDQ1 <- RWDQ[,-1]  ## eliminate first item (misfit)
irtpar <- ltm(RWDQ1 ~ z1)
fapar <- ltm(RWDQ1 ~ z1, IRT.param = FALSE)
round(head(cbind(coef(irtpar), coef(fapar))), 3)
##         Dffclt Dscrmn (Intercept)    z1
## wdq_23 -1.064  0.736       0.782 0.736
## wdq_24 -0.387  0.822       0.318 0.822
## wdq_25 -1.456  1.561       2.273 1.561
## wdq_26 -2.113  0.668       1.412 0.668
## wdq_27 -0.570  1.157       0.659 1.157
## wdq_28 -1.138  0.808       0.919 0.808
```

The first two columns are based on the parameterization in Eq. (4.11), involving α_i and β_i. The last two columns correspond to d_i and α_i from Eq. (4.12). Let us compute the person parameters and confirm that they are the same for both models.

```
irtppar <- factor.scores(irtpar)$score.dat$z1
fappar <- factor.scores(fapar)$score.dat$z1
identical(irtppar, fappar)
## [1] TRUE
```

Note that all unidimensional IRT models presented in Sects. 4.2 and 4.3 can be reformulated using the intercept-slope parameterization.

Moving on to more general multidimensional representations, there are factor analytic techniques that make use of these connections and apply particular methods for parameter estimation, suited for categorical data. This variant is often referred to as *item factor analysis* (IFA; see Wirth and Edwards, 2007, for an overview). IFA marries IRT and FA so that we get the best of both worlds: On the one hand, we have all the possibilities exploratory/confirmatory model specification, good opportunities for goodness-of-fit assessment, and options for rotation from the FA world. On the other hand, we can make use of the wide range of developed IRT models and get a deep probabilistic insight into the behavior of items and persons.

Since FA uses aggregate measures (correlations, covariances), the corresponding estimation concept is called *limited information estimation*, as opposed to *full information estimation* which is based on the raw input data. IFA uses full information estimation; the **mirt** package (Chalmers, 2012) provides corresponding implementations. IFA can be estimated in an exploratory as well as in a confirmatory way, as illustrated in the following two subsections. Note that the terms "MIRT" and "IFA" are often used synonymously. Here, from now on, we use "MIRT."

4.7.2 Exploratory Multidimensional IRT

The **mirt** package allows us to fit all dichotomous and polytomous IRT models presented above in a multidimensional exploratory or confirmatory fashion. The main function is `mirt`. Through the `itemtype` argument, a particular IRT model can be specified. In this section, we focus on exploratory specifications, that is, each item is free to load on each factor.

MIRT models are typically formulated via the intercept-slope parameterization. For instance, a p-dimensional 2-PL can be written as

$$\text{logit}(P(X_{vi})) = \boldsymbol{\alpha}_i \boldsymbol{\theta}_v' + d_i. \tag{4.13}$$

In this equation, $\boldsymbol{\alpha}_i$ is a vector of length p containing the slope/discrimination parameters of item i on each dimension. Unfortunately, we cannot interpret d_i (which is a scalar) as item location parameter. However, the following transformation does the trick:

$$\beta_i = -\frac{d_i}{\sqrt{\boldsymbol{\alpha_i}'\boldsymbol{\alpha_i}}} \tag{4.14}$$

The denominator reflects the *multidimensional item discrimination*. Note that in MIRT models, we cannot obtain an item location for item i on each dimension. Rather, β_i denotes the *multidimensional item location*. As illustrated in the next section, in MIRT, unidimensional ICCs generalize to *item characteristic surfaces* (ICS) in the p-dimensional space. A β_i indicates the distance from the origin in the multidimensional trait space to the point of maximum slope of the surface. Corresponding illustrations can be found in de Ayala (2009, pp. 281–288) and Reckase (2009, pp. 86–91). Finally, $\boldsymbol{\theta}_v$ is the *multidimensional person parameter*. Each person gets a parameter (i.e., a factor score) on each dimension.

Let us illustrate an exploratory multidimensional 2-PL model fit using all binary ZAREKI addition and subtraction items described in Sect. 4.2.1. First, we compute an exploratory Princals in order to get an idea of the structure and dimensionality.

```
library("MPsychoR")
library("Gifi")
data("zareki")
itzareki <- zareki[, 1:16]
przar <- princals(itzareki)
plot(przar)
plot(przar, "screeplot")
```

The scree plot in Fig. 4.17 (right panel) suggests that two dimensions should be sufficient. From the loadings plot in the left panel, we see that the dimensions are not addition vs. subtraction, as we might have expected.

Let us fit two models: a unidimensional 2-PL and a two-dimensional 2-PL[12]:

```
zar1d <- mirt(itzareki, 1, itemtype = "2PL")
zar2d <- mirt(itzareki, 2, itemtype = "2PL")
```

[12]Note that we fitted these models already in Sect. 4.1.2 on dimensionality assessment.

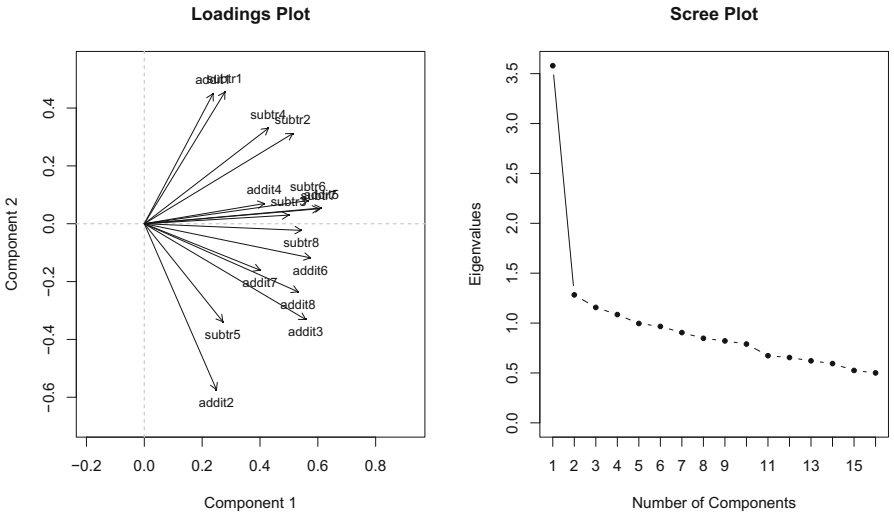

Fig. 4.17 Left panel: Princals loadings plot for addition/subtraction items in ZAREKI data. Right panel: Princals scree plot

We can apply an LR-test which suggests that the 2D-model fits significantly better than the 1D-model[13]:

```
anova(zar1d, zar2d)
##
## Model 1: mirt(data = itzareki, model = 1, itemtype = "2PL")
## Model 2: mirt(data = itzareki, model = 2, itemtype = "2PL")
##        AIC     AICc    SABIC      BIC    logLik    X2  df
## 1 4568.562 4575.419 4589.671 4691.182 -2252.281   NaN NaN
## 2 4564.952 4580.351 4595.957 4745.051 -2235.476 33.61  15
##       p
## 1   NaN
## 2 0.004
```

For further fit examination of the 2D model, let us compute the M_2 statistic including the CFA/SEM fit indices:

[13]There is some ambiguity in AIC/BIC in relation to the LR-test result. We could further explore the fit of the 1D solution via `M2(zar1d)` which suggests a slight misfit.

```
M2(zar2d)
##             M2 df         p RMSEA RMSEA_5   RMSEA_95
## stats 94.11394 89 0.3350678 0.013       0 0.03274862
##            SRMSR       TLI       CFI
## stats 0.04382372 0.9943675 0.9958225
```

We get a low RMSEA (here we use 0.05/2 as cutoff since we have binary items) and a high CFI, and the M_2 p-value is not significant. The 2D model fits well.

Itemfit statistics can be computed using the corresponding function in **mirt**. Here we print out the misfitting items only.

```
ifit2D2pl <- mirt::itemfit(zar2d)
ifit2D2pl[ifit2D2pl[, 4] < 0.05, ]  ## misfitting items
##      item   S_X2 df.S_X2 p.S_X2
## 2  addit2 15.986       8  0.043
## 12 subtr4 20.871       8  0.007
```

The `subtr4` item could be eliminated since it shows some misfit, and then the model needs to be refitted again. However, since the global model fit suggested a well-fitting solution, let us keep it.

The factor analytic parameterization can be obtained through the `summary` call. We can also request to apply an orthogonal or non-orthogonal rotation for better interpretability (output not shown here):

```
summary(zar2d, rotate = "varimax")
summary(zar2d, rotate = "oblimin")
```

The **mirt** package provides several plotting options for 2D models. Figure 4.18 shows the 2D-ICS for the third item:

```
itemplot(zar2d, 3, main = "ICS addit3",
         rot = list(xaxis = -70, yaxis = 50, zaxis = 10))
```

The intercept-slope IRT parameterization, as given in Eq. (4.13), can be obtained via `coef(zar2d)`. The multidimensional item location, according to Eq. (4.14), is (only first six items shown here)

```
head(MDIFF(zar2d))
##           MDIFF_1
## addit1 -3.7228677
## addit2 -2.7192539
## addit3 -1.4413654
## addit4 -2.3763393
## addit5 -1.0631562
## addit6 -0.5634567
```

Finally, we can compute the person parameters, one for each dimension (only first six persons shown here):

```
head(fscores(zar2d))
##              F1          F2
## [1,]  0.3162909 -0.08593608
## [2,] -0.1304509  0.26424651
## [3,] -0.3173961 -0.07663827
## [4,]  0.1257834 -0.45310387
## [5,] -0.3173961 -0.07663827
## [6,] -0.4318470  0.89689762
```

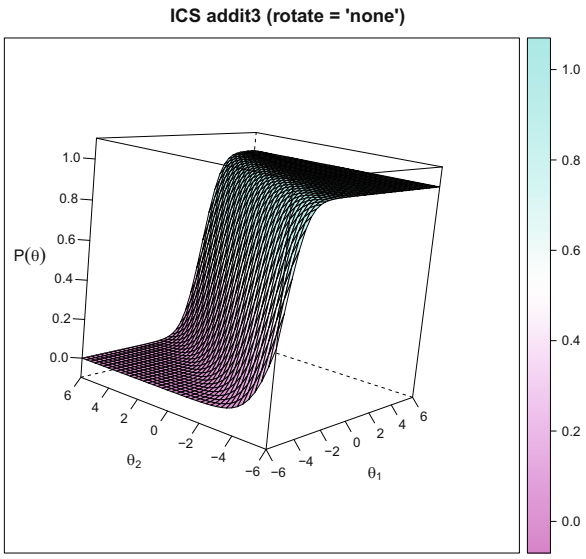

Fig. 4.18 Item characteristic surface (2D) for the third item

In this example, it is difficult to interpret the two dimensions without considering additional specific characteristics of the items itself since, as mentioned above, the dimensions are certainly not addition and subtraction.

In a second example, we illustrate how to fit an *exploratory multigroup MIRT* model on the same data which, within an IRT context, means nothing else than testing for DIF. We group the class variable into two categories and fit a multigroup 2D-2PL which results in separate sets of parameter estimates for both groups. We are interested in testing whether the discrimination parameters (on the first dimension) and the intercepts differ across the two groups. For testing parameter equality across the groups, we apply a Wald test and correct the p-values by means of the *false discovery rate* (FDR; see Sect. 14.3 for details).

```
class2 <- zareki$class
levels(class2) <- c("second", "thirdfourth", "thirdfourth")
modMG <- multipleGroup(itzareki, model = 2, group = class2,
                       SE = TRUE, verbose = FALSE)
astiDIF <- DIF(modMG, c('a1', 'd'), Wald = TRUE,
               p.adjust = 'fdr')
round(astiDIF$adj_pvals[astiDIF$adj_pvals < 0.05], 4)
## addit3 addit6 addit8 subtr3 subtr6 subtr7 subtr8
## 0.0145 0.0002 0.0005 0.0005 0.0000 0.0024 0.0007
```

The last row prints out the p-values for the items flagged for DIF (i.e., the corresponding p-values are significant). Note that strictly speaking, these results reflect a combination of DIF and latent trait distribution effects. The baseline model should contain a set of anchor items such that we properly equate the groups. Examples can be found in the `DIF` help file.

In general, the `multipleGroup` function is highly flexbile in terms of putting all sorts of parameter restrictions on multiple group models, similar to **lavaan** (Rosseel, 2012) in the CFA/SEM world.

4.7.3 Confirmatory Multidimensional IRT

In this section, we illustrate how MIRT models can be fitted in a confirmatory way. As in CFA, we assign items to load on a particular dimension. We use once more the polytomous ASTI data from Sect. 4.3.2 and assign the items to the following five factors, based on the underlying ASTI theory: self-knowledge and integration (`si`), peace of mind (`pm`), non-attachment (`na`), self-transcendence (`st`), and presence in the here-and-now and growth (`pg`), as described in Koller et al. (2017). We fit a multidimensional GRM (five dimensions).

Note that the **mirt** model specification syntax is a bit different from **lavaan**. Below we specify the item assignments to the factors and the corresponding covariance structure of the latent variables (we allow for a full correlation pattern).

```
data("ASTI")
itasti <- ASTI[, 1:25]
modASTI <- mirt.model('
  si = 10,19,20,21
  pm = 1,5,9,22
  na = 3,6,8,12
  st = 2,4,7,13,16,24,25
  pg = 11,14,15,17,18,23
  COV = si*pm*na*st*pg
')
asti5d <- mirt(itasti, model = modASTI, itemtype = 'graded',
  method = 'MHRM', SE.type = 'MHRM', verbose = FALSE)
```

Compared to the 2D-2PL from above, here we use a different estimation algorithm which is suggested to employ for higher-dimensional models.

First, we print out correlation matrix of the trait dimensions. Second, from the loadings in `astisum$rotF`, we see that item 18 ("I often lose myself in what I am doing") has a very small loading on the personal growth dimension and could therefore be subject to elimination.

```
astisum <- summary(asti5d, verbose = FALSE)
round(astisum$fcor, 3)
##       si    pm    na    st    pg
## si 1.000 0.695 0.217 0.101 0.836
## pm 0.695 1.000 0.546 0.159 0.735
## na 0.217 0.546 1.000 0.155 0.175
## st 0.101 0.159 0.155 1.000 0.345
## pg 0.836 0.735 0.175 0.345 1.000
round(astisum$rotF["ASTI18",], 4)
##      si      pm      na      st      pg
##  0.0000  0.0000  0.0000  0.0000 -0.0061
```

The global fit indices and model test suggest a poor model fit.

```
M2(asti5d, QMC = TRUE)
##             M2  df p      RMSEA    RMSEA_5   RMSEA_95
## stats 1399.775 228 0 0.06749943 0.06409668 0.07088774
##           SRMSR       TLI       CFI
## stats 0.1064949 0.6875048 0.7290916
```

For such complex scenarios, an exploratory version of the model with oblique rotation would give us the possibility to explore in more detail what is going on. Another good strategy, before even considering fitting a confirmatory MIRT model, is to compute unidimensional models for each subscale individually and eliminate misfitting items already at that level. The items kept in the model can be subsequently subject to a higher-dimensional IRT fit.

4.8 Longitudinal IRT Models

IRT models can be applied to longitudinal/repeated measurements designs as well. In this case, it is of interest to study changes over time. We start with presenting some approaches from the Rasch world, followed by specifications within an MIRT framework.

4.8.1 Linear Logistic Models for Measuring Change

Using a design matrix approach for Rasch model fitting, as the **eRm** package does, offers many possibilities to test specific hypotheses on the items. One option is to test whether change in item parameters occurs over time. As Fischer (1995) claims, any change in person parameters occurring between the testing occasions can be described as a change in the item parameters. For instance, an increase in ability is equivalent to a decrease in difficulty.

Models that allow us to test for such item changes are the *linear logistic test model* (LLTM), the *linear logistic model with relaxed assumptions* (LLRA), the *linear logistic rating scale model* (LRSM), and the *linear logistic partial credit model* (LPCM), all implemented in **eRm**. The LLTM is designed for dichotomous items, the LLRA for dichotomous and polytomous items, and LRSM/LPCM for polytomous items. Technical details on these models can be found in Fischer (1995) and Mair and Hatzinger (2007b), applied instructions in Glück and Spiel (1997). All of these models obey the Rasch measurement principle.

Let us start with an LLTM application using a dataset on social dominance orientation (SDO; Sidanius and Pratto, 2001). SDO items were assessed over 5 years on 612 persons. The four items are scored on a 7-point scale. Here we binarize them according to 1 vs. higher and use the first 3 years only (1996–1998). The input data matrix $\mathbf{X}$ is in wide format and of dimension $n \times (mT)$, where m denotes the number of items within each time point and T is the number of time points. The data preparation steps are the following:

```
library("MPsychoR")
data("SDOwave")
SDO3 <- SDOwave[,c(1:12)]
SDO3 <- sapply(SDO3, function(bin) ifelse(bin == 1, 0, 1))
```

In order to apply the LLTM, the data need to be unidimensional, and within each time point, the Rasch model should fit. This should be checked before fitting the LLTM, using the strategies presented in Sect. 4.2.1. Let us compute an LLTM using the **eRm** package:

```
library("eRm")
sdolltm1 <- LLTM(SDO3, mpoints = 3)
sdolltm1$W
##       eta 1 eta 2 eta 3 eta 4 eta 5
## I1 t1    -1    -1    -1     0     0
## I2 t1     1     0     0     0     0
## I3 t1     0     1     0     0     0
## I4 t1     0     0     1     0     0
## I1 t2    -1    -1    -1     1     0
## I2 t2     1     0     0     1     0
## I3 t2     0     1     0     1     0
## I4 t2     0     0     1     1     0
## I1 t3    -1    -1    -1     0     1
## I2 t3     1     0     0     0     1
## I3 t3     0     1     0     0     1
## I4 t3     0     0     1     0     1
```

By inspecting the design matrix, we see that we estimate $m - 1 = 3$ item parameters (η_1, η_2, and η_3). In addition, we get $T - 1 = 2$ dummy coded time contrasts (η_4 and η_5; 1996 as baseline year). The parameters associated with these contrasts are of crucial interest since they tell us whether a general change occurred in 1997 vs. 1996 and 1998 vs. 1996.

```
summary(sdolltm1)
##
## Basic Parameters eta with 0.95 CI:
##       Estimate Std. Error lower CI upper CI
## eta 1   -1.383      0.058   -1.498   -1.268
## eta 2    1.040      0.058    0.927    1.153
## eta 3    0.558      0.054    0.452    0.665
## eta 4   -0.145      0.076   -0.294    0.005
## eta 5    0.067      0.076   -0.083    0.216
```

The upper boundary of the η_4 CI is barely including 0. We get a hint that there might be an overall change in SDO (negative direction) from 1996 to 1997, but here it is not significant. There is no change from 1996 to 1998.

As an alternative testing strategy, we can apply an LR-test based on a more restrictive model where we delete both time contrasts from the design matrix (i.e., we eliminate all options for time change).

```
W0 <- sdolltm1$W[,-c(4:5)]
sdolltm0 <- LLTM(SDO3, W0)
anova(sdolltm0, sdolltm1)
## Analysis of Deviances Table
##
## Model 1: LLTM(X = SDO3, mpoints = 3)
## Model 2: LLTM(X = SDO3, W = W0)
##
##         cond. LL Deviance npar     LR df p-value
## Model 1  -2224.5   4449.1    5
## Model 2  -2228.6   4457.1    3 4.0347  2   0.133
```

This test suggests that there is no significant overall time effect in the item parameters which translates to no overall change in participants' SDO over these 3 years.

Another modeling option in LLTM is to incorporate a group contrasts. Since there are no person covariates in the dataset, let us, for illustration, create an artificial grouping variable, refit the model with time and group effect, and compare the two models via the LR-test. Not too surprisingly, it shows that there is no significant group effect.

```
group <- rep(c(1, 2), each = nrow(SDO3)/2)  ## fake covariate
sdolltm2 <- LLTM(SDO3, mpoints = 3, group = group)
anova(sdolltm2, sdolltm1)
## Analysis of Deviances Table
##
## Model 1: LLTM(X = SDO3, mpoints = 3, groupvec = group)
## Model 2: LLTM(X = SDO3, mpoints = 3)
##
##         cond. LL Deviance npar     LR df p-value
## Model 1  -2224.3   4448.5    6
## Model 2  -2224.5   4449.1    5 0.2783  1  0.5978
```

Note that the LLTM is pretty restrictive since we only obtain a single global time effect on all items jointly, rather than item-specific ones. That is, we assume that the change over time is the same across all items. The LLRA relaxes this assumption: each item gets its own dimension, and, consequently, for each item, we get an individual time effect. In **eRm**, an LLRA can be fitted as follows:

```
sdollra <- LLRA(SDO3, mpoints = 3)
summary(sdollra)
## Reference group:  CG
##             Estimate Std.Error lower.CI upper.CI
## trend.I1.t2    0.320     0.144    0.037    0.603
## trend.I2.t2   -0.109     0.156   -0.415    0.196
## trend.I3.t2   -0.416     0.162   -0.735   -0.098
## trend.I4.t2   -0.478     0.156   -0.784   -0.172
## trend.I1.t3    0.444     0.145    0.160    0.728
## trend.I2.t3    0.060     0.155   -0.243    0.363
## trend.I3.t3   -0.094     0.164   -0.414    0.227
## trend.I4.t3   -0.219     0.156   -0.526    0.087
```

We see that for items 3 and 4, there is a significant change in location (decreasing) from 1996 to 1997 (effects: `trend.I3.t2` and `trend.I4.t2`), whereas for item 1 there is a significant increase (`trend.I1.t2`) during that period. For the years 1998 vs. 1996, there was a significant increase in item 1 only (`trend.I1.t3`). This tells us that the participants change over time with respect to these particular SDO items.

The design matrix approach makes LLTM/LLRA models highly flexible. A manual-like instruction on LLRA model specification can be found in Hatzinger and Rusch (2009), who present additional LLRA modeling options, including LLRA group contrasts.

4.8.2 Two-Tier Approach to Longitudinal IRT

The *two-tier item factor* approach by Cai (2010) is a confirmatory multidimensional model which allows us to set specific structural restrictions. Here we focus on restrictions related to longitudinal IRT modeling. These models can be fitted using **mirt**'s bifactor model function (`bfactor`).

Bifactor models are a general structural concept for latent variable models. In a simple bifactor model, the indicators load on a *general factor* (also called *primary factor*) as well as on *specific factors* (i.e., specific to a particular indicator set). An overview can be found in Reise (2012), and an illustration on how to fit them within a classical FA context using **lavaan** is given in Beaujean (2014) and Finch and French (2015).

To illustrate the two-tier approach fitted through a bifactor model, let us use the SDO data from the previous section once more. Here we consider the first 2 years only and re-categorize the items by collapsing categories 5–7 into a single category 5 (i.e., we have 5-point responses):

```
SDO2 <- SDOwave[,1:8]
SDO2 <- sapply(SDO2, function(co) cut(co, c(0,1,2,3,4,7),
                                      labels = 1:5))
class(SDO2) <- "numeric"
```

The structure of the longitudinal two-tier bifactor model we are going to fit is given in Fig. 4.19. We need two primary factors which reflect the two time points: The first four items load on the 1996 factor; the remaining four items load on the 1997 factor. The pairwise residual correlations are captured by specific factors (S1 through S4).

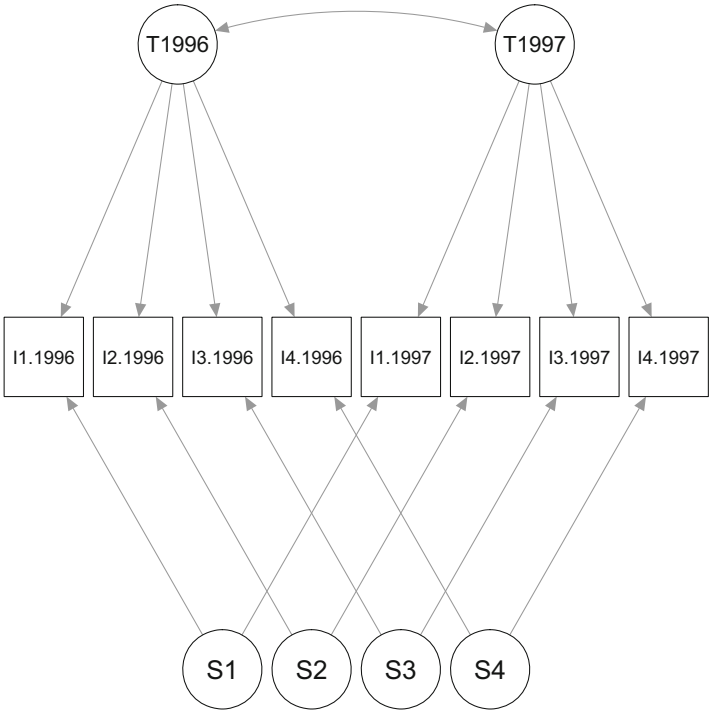

Fig. 4.19 Two-tier factor structure for SDO data (four items, 2 years). T1996 and T1997 are the primacy factors and S1, S2, S3, and S4 the specific factors

Let us specify this model in **mirt**.[14] Below, the first line determines the loading structure for the specific factors. We allow the time factors to be correlated, estimate a variance for `T1997` while fixing the `T1996` variance to 1, fix the mean for `T1996` to 0, and estimate the `T1997` in relation to it. This mean estimate will capture the degree of change over time.

```
iloads <- rep(1:4, 2)
ttmodel <- mirt.model('
  T1996 = 1-4
  T1997 = 5-8
  COV = T1996*T1997, T1997*T1997
  MEAN = T1997
  CONSTRAIN = (1, 5, d1), (2, 6, d1), (3, 7, d1), (4, 8, d1),
              (1, 5, d2), (2, 6, d2), (3, 7, d2), (4, 8, d2),
              (1, 5, d3), (2, 6, d3), (3, 7, d3), (4, 8, d3),
              (1, 5, d4), (2, 6, d4), (3, 7, d4), (4, 8, d4)')
fitSDO2 <- bfactor(SDO2, iloads, ttmodel, SE = TRUE)
```

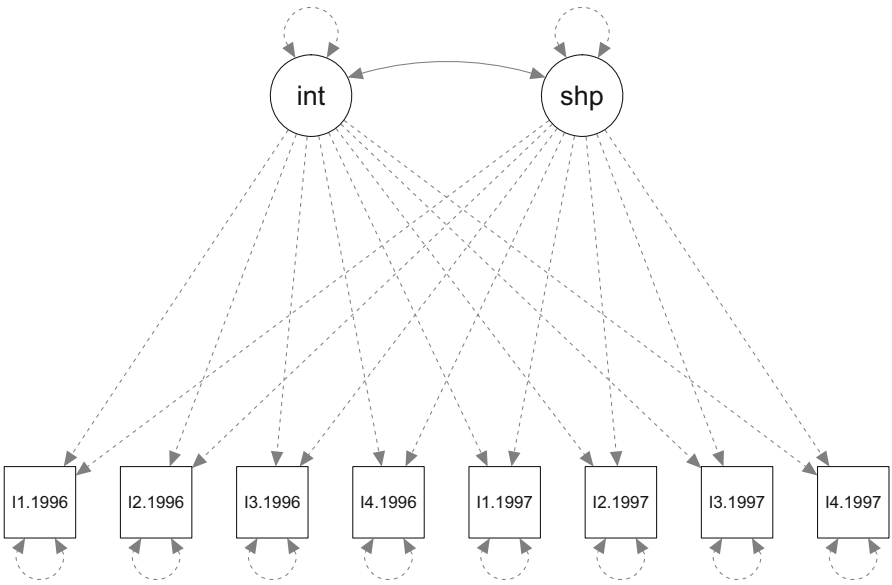

Fig. 4.20 IRT growth model for SDO data, two time points

[14] Note that there is also the option to specify it in **lavaan** and convert it into **mirt** syntax using the `lavaan2mirt` function in the **sirt** package. This can be especially useful for more complicated models with constraints.

Note that, by default, this function uses a GRM. Through the `itemtype` argument, other specifications can be set. By saying `summary(fitSDO2)`, we get the FA output. With `coef(fitSDO2, simplify=TRUE)`, we get the intercept-slope parameterization. If our focus is on the change parameter, we can pull it out as follows (including the 95% CI):

```
round(coef(fitSDO2)$GroupPars[,2], 4)
##     par  CI_2.5 CI_97.5
## -0.1235 -0.2209 -0.0261
```

It suggests a significant change in SDO (negative direction) from 1996 to 1997. Further options for two-tier specifications can be found in Cai (2010).

4.8.3 Latent Growth IRT Models

Another approach to assess longitudinal changes are *latent growth models* (LGMs), as introduced in Sect. 3.4, involving a latent intercept and a latent shape. For fitting an LGM within an IRT context, we can use the `bfactor` function in **mirt** once more. Compared to **lavaan**, it uses a different estimation algorithm suited for categorical data. In addition, we can request the IRT parameterization as output (by default it uses the FA parameterization).

Again, we use the SDO dataset (5-point response scale, two time points) from above. The structure of the latent growth model is given in Fig. 4.20 and can be specified as follows:

```
itloads <- rep(1:4, 2)
modgr <- mirt.model('
    Intercept = 1-8
    Slope = 1-8
    COV = Intercept*Slope, Intercept*Intercept, Slope*Slope
    MEAN = Intercept, Slope
    START = (1-8, a1, 1), (1-4, a2, 0), (5-8, a2, 1)
    FIXED = (1-8, a1), (1-4, a2), (5-8, a2)
    CONSTRAIN =(1, 5, d1), (2, 6, d1), (3, 7, d1), (4, 8, d1),
               (1, 5, d2), (2, 6, d2), (3, 7, d2), (4, 8, d2),
               (1, 5, d3), (2, 6, d3), (3, 7, d3), (4, 8, d3),
               (1, 5, d4), (2, 6, d4), (3, 7, d4), (4, 8, d4)')
fitGIRT <- bfactor(SDO2, itloads, modgr, SE = TRUE)
```

We print out the slope parameter and the covariance between the intercept and slope constructs (including 95% CI).

```
coef(fitGIRT)$GroupPars[, c(2,8)]
##             MEAN_2    COV_21
## par     -0.4569969 -2.124596
## CI_2.5  -0.7028949 -2.968603
## CI_97.5 -0.2110988 -1.280590
```

The mean tells us that, on the average, there was a significant overall decrease in SDO from 1996 to 1997. The significant negative covariance indicates that there is less SDO decrease in persons that had higher SDO in 1996 than those with lower SDO.

This concludes the section on longitudinal IRT models for the moment. In Sect. 4.9.2 below, we present another option for longitudinal IRT modeling by means of dynamic models fitted within a Bayesian framework. Dynamic models put emphasis on individual changes in the person parameters.

4.9 Bayesian IRT

In this last IRT section, we show how item response models can be fitted the Bayesian way. Bayesian IRT estimation is especially useful for hierarchical/multilevel settings and for incorporating response times. Detailed elaborations can be found in Fox (2010) who uses WinBUGS for Markov Chain Monte Carlo (MCMC) sampling. Here we use functions from **MCMCpack** (Martin et al., 2011) and, therefore, avoid to write our own BUGS, JAGS, or Stan code.[15] The package includes the following functions for dichotomous 2-PL IRT modeling:

- Unidimensional models: `MCMCirt1d` with `MCMCirtHier1d` as hierarchical extension.
- Multidimensional models: `MCMCirtKd` with `MCMCirtKdHet` as heteroscedastic variant and `MCMCirtKdRob` as robust variant.
- Dynamic IRT models: `MCMCdynamicIRT1d`.

4.9.1 Bayesian 2-PL Estimation

MCMCpack uses an intercept-slope IRT parameterization using the following symbols: $-\alpha_i + \beta_i\theta_v$. This is similar to Eq. (4.12), but here β_i is the discrimination parameter and $-\alpha_i$ is the negative intercept (which corresponds to d_i in Eq. (4.12)). Thus, we have to be careful when interpreting the output.

[15] See Luo and Jiao (2017) for how to specify IRT models in Stan.

By default, **MCMCpack** functions use standard normal priors for the person parameters and normal priors for the item parameters (mean 0 and precision, i.e., inverse variance, of 0.25). For many applications, this setting may not be the best choice, and the user should carefully consider the prior specification (details on IRT priors can be found in Natesan et al., 2016). Unfortunately, the options for customizing priors are very limited in **MCMCpack**.

To illustrate a simple Bayesian 2-PL fit, we use the eight items from the WDQ example from Sect. 4.2.2. First, we fit an ordinary 2-PL using the **ltm** package and extract the parameters which, in turn, will act as starting values for the MCMC sampler.

```
library("MPsychoR")
data("RWDQ")
RWDQ1 <- RWDQ[, 2:9]  ## select 8 items
freq2pl <- ltm(RWDQ1 ~ z1)
intstart <- -coef(freq2pl)[,1]
discstart <- coef(freq2pl)[,2]
```

Now we are going to fit three MCMC chains, each with a burn-in of 5000 and 50,000 Gibbs iterations. We lower the prior precision to `AB0=0.15` in order to keep the prior slightly more vague than the default setting.

```
library("MCMCpack")
chainWDQ1 <- MCMCirt1d(RWDQ1, burnin = 5000, mcmc = 50000,
  seed = 111, AB0 = 0.15, store.item = TRUE,
  store.ability = FALSE, verbose = TRUE,
  alpha.start = intstart, beta.start = discstart)
chainWDQ2 <- MCMCirt1d(RWDQ1, burnin = 5000, mcmc = 50000,
  seed = 222, AB0 = 0.15, store.item = TRUE,
  store.ability = FALSE, verbose = TRUE,
  alpha.start = intstart, beta.start = discstart)
chainWDQ3 <- MCMCirt1d(RWDQ1, burnin = 5000, mcmc = 50000,
  seed = 333, AB0 = 0.15, store.item = TRUE,
  store.ability = FALSE, verbose = TRUE,
  alpha.start = intstart, beta.start = discstart)
WDQlist <- mcmc.list(chainWDQ1, chainWDQ2, chainWDQ3)
```

By saying `summary(WDQlist)` the posterior means and corresponding quantiles are shown. Geweke's convergence diagnostics, for which the z-values should be within a $[-2, 2]$ interval, can be requested via `geweke.diag(WDQlist)`. Figure 4.21 shows the trace plots for the first two items and the posterior densities.

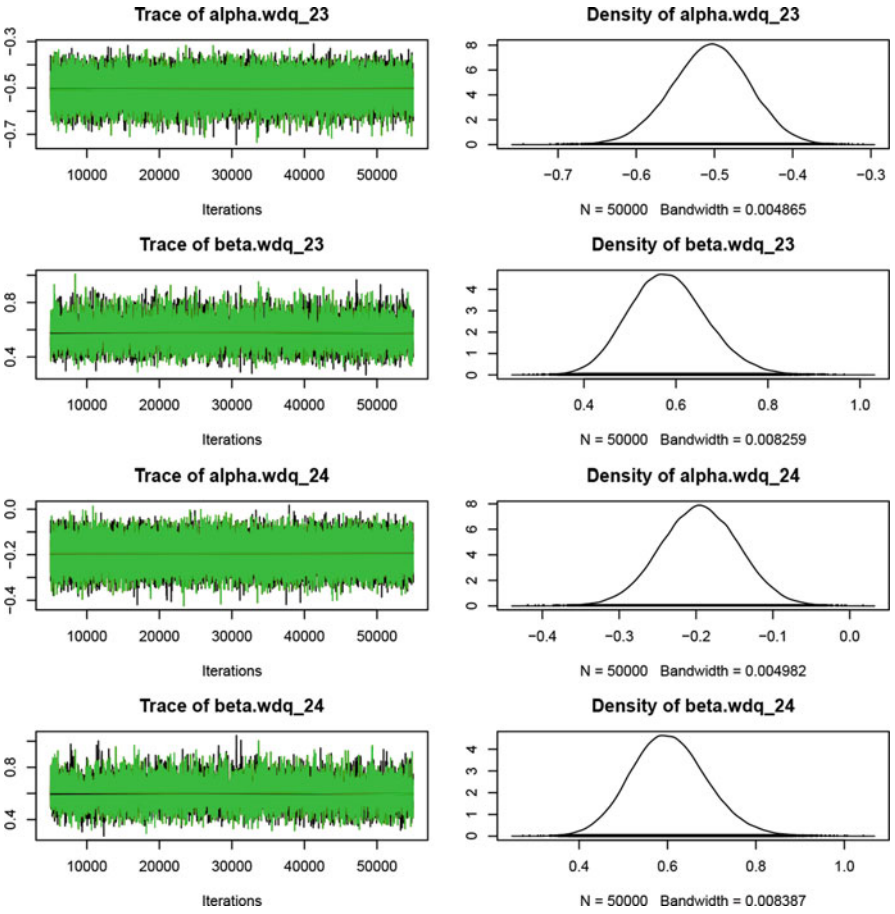

Fig. 4.21 Trace plots and posterior densities for the first two items: "alpha" represents the intercepts, "beta" the slope parameters

```
plot(WDQlist, auto.layout = FALSE, ask = FALSE)
```

The chains are fairly well mixed. If the posterior distributions for the person parameters are of interest, setting `store.ability=TRUE` in the `MCMCirt1d` call stores the corresponding posterior distributions.

4.9.2 Dynamic 2-PL Model

As a second Bayesian IRT example, we fit a dynamic 2-PL model. Dynamic IRT models (see, e.g., Wang et al., 2013) can be used to study individual learning effects or, more general, changes at an individual level. Thus, the main interest in the following analysis is in the person parameter trajectories across multiple measurements per individual.

We use data from a questionnaire assessing health risk behaviors,[16] including smoking, drinking, and marijuana consumption. The questionnaire was presented to teenagers at five points in time (from middle school to high school). The items are dichotomously scored. We only use teenagers who scored 1 on at least one item and create a time variable denoting the measurement point. We then fit a dynamic 2-PL using default prior settings and request to store the posteriors for the person parameters only. In order to keep the computing time considerably low, we only run a single, short MCMC chain.

```
data("HRB")
HRB1 <- HRB[rowSums(HRB) > 0, ]
rownames(HRB1) <- 1:nrow(HRB1)
time <- rep(1:5, each = 4)
fitdyn <- MCMCdynamicIRT1d(HRB1, item.time.map = time,
  mcmc = 20000, burnin = 5000,seed = 111, store.ability = TRUE,
  store.item = FALSE, verbose = TRUE)
dynsum <- summary(fitdyn)
```

Due to space restrictions, we omit to show the MCMC diagnostics here. Each person gets a posterior distribution (see `dynsum` object). Below we pull out the posterior means and organize the data for trajectory plotting as given in Fig. 4.22.

```
nt <- 5                              ## number of time points
postmean <- dynsum$statistics[,1]  ## posterior means
pertraj <- t(matrix(postmean[1:(nrow(HRB1)*nt)], nrow = nt))
colnames(pertraj) <- paste0("T", 1:5)
round(head(pertraj), 3)
##          T1     T2     T3     T4     T5
## [1,] -0.464 -0.635 -0.687 -0.517 -0.578
## [2,]  0.987  0.583  0.455 -0.294 -0.675
## [3,]  0.636  1.261  2.430  2.273  1.496
## [4,]  0.092  0.292  0.704  0.622  0.233
```

(continued)

[16] Thanks to Peter Franz for sharing this dataset.

```
## [5,] -0.622 -0.931 -1.166 -1.204 -1.366
## [6,] -0.775 -1.140 -1.402 -1.564 -1.523
matplot(t(pertraj), type = "l", lty = 1, cex = 0.8,
 col = adjustcolor(1, alpha.f = 0.3),
 ylab = "person parameter", xlab = "time points",
 main = "Individual Trajectories")
```

The spaghetti plot suggests that there is a set of teenagers that started with a low consumption at time 1 which then decreased even more over the 5 years. There are a few outliers for which the consumption increased drastically. However, there is no obvious general trend in these trajectories.

This concludes the section on Bayesian IRT which adds a great deal of modeling flexibility to the IRT environment. Additional Bayesian IRT modeling options can be found in Fox (2010).

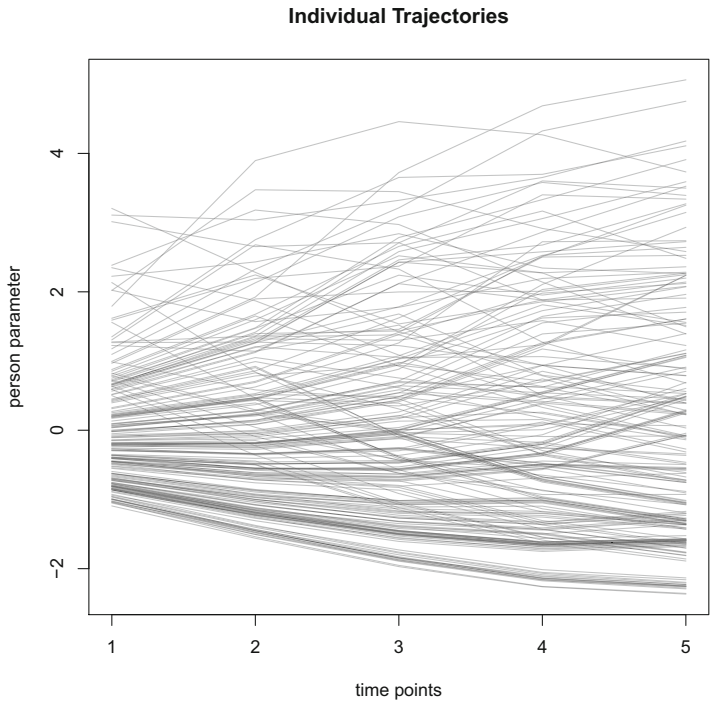

Fig. 4.22 Individual person parameter trajectories for health risk behavior dataset across five time points, resulting from a dynamic 2-PL fit

References

Andersen, E. B. (1973). A goodness of fit test for the Rasch model. *Psychometrika, 38*, 123–140.

Andrich, D. (1978). A rating formulation for ordered response categories. *Psychometrika, 43*, 561–573.

Beaujean, A. A. (2014). *Latent variable modeling using R: A step-by-step guide*. New York: Routledge.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In: F. M. Lord & M. R. Novick (Eds.), *Statistical theories of mental test scores* (pp. 395–479). Reading: Addison-Wesley.

Bock, R. D. (1972). Estimating item parameters and latent ability when responses are scored in two or more nominal categories. *Psychometrika, 37*, 29–51.

Bond, T. G., & Fox, C. M. (2015). *Applying the Rasch model: Fundamental measurement in the human sciences* (3rd ed.). New York: Routledge.

Cai, L. (2010). A two-tier full-information item factor analysis model with applications. *Psychometrika, 75*, 581–612.

Chalmers, R. P. (2012). **mirt**: A multidimensional item response theory package for the R environment. *Journal of Statistical Software, 48*(6), 1–29. http://www.jstatsoft.org/v48/i06/

Chalmers, R. P. (2017). **SimDesign**: Structure for organizing Monte Carlo simulation designs. R package version 1.6. https://CRAN.R-project.org/package=SimDesign

Choi, S., Gibbons, L., & Crane, P. (2011). **lordif**: An R package for detecting differential item functioning using iterative hybrid ordinal logistic regression/item response theory and Monte Carlo simulations. *Journal of Statistical Software, 39*(1), 1–30. https://www.jstatsoft.org/index.php/jss/article/view/v039i08

de Ayala, R. J. (2009). *The theory and practice of item response theory*. New York: Guilford Press.

De Boeck, P., Bakker, M., Zwitser, R., Nivard, M., Hofman, A., Tuerlinckx, F., & Partchev, I. (2011). The estimation of item response models with the `lmer` function from the **lme4** package in R. *Journal of Statistical Software 39*(1), 1–28. https://www.jstatsoft.org/index.php/jss/article/view/v039i12

Finch, W. H., Jr., & French, B. F. (2015). *Latent variable modeling with R*. New York: Routledge.

Fischer, G. H. (1995). Linear logistic models for change. In: G. Fischer & I. Molenaar (Eds.), *Rasch models: Foundations, recent developements, and applications* (pp. 157–180). New York: Springer.

Fischer, G. H., & Molenaar, I. W. (1995). *Rasch models: Foundations, recent developements, and applications*. New York: Springer.

Fox, J. P. (2010). *Bayesian item response modeling*. New York: Springer.

Funk, J. B., Fox, C. M., Chang, M., & Curtiss, K. (2008). The development of the children's empathic attitudes questionnaire using classical and Rasch analyses. *Journal of Applied Developmental Psychology, 29*, 187–196.

Glück, J., & Spiel, C. (1997). Item response models for repeated measures designs: Application and limitations of four different approaches. *Methods of Psychological Research, 2*(6). http://www.dgps.de/fachgruppen/methoden/mpr-online/issue2/art6/article.html

Hatzinger, R., & Rusch, T. (2009). IRT models with relaxed assumptions in **eRm**: A manual-like instruction. *Psychology Science Quarterly, 51*, 87–120.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning with applications in R*. New York: Springer.

Jiang, S., Wang, C., & Weiss, D. J. (2016). Sample size requirements for estimation of item parameters in the multidimensional graded response model. *Frontiers in Psychology, 7*(109), 1–10.

Koller, I., & Alexandrowicz, R. W. (2010). Eine psychometrische Analyse der ZAREKI-R mittels Rasch-Modellen [A psychometric analysis of the ZAREKI-R using Rasch-models]. *Diagnostica, 56*, 57–67.

Koller, I., Levenson, M. R., & Glück, J. (2017). What do you think you are measuring? A mixed-methods procedure for assessing the content validity of test items and theory-based scaling. *Frontiers in Psychology, 8*(126), 1–20.

Komboz, B., Zeileis, A., & Strobl, C. (2018, Forthcoming). Tree-based global model tests for polytomous Rasch models. *Educational and Psychological Measurement, 78*, 128–166.

Levenson, M. R., Jennings, P. A., Aldwin, C. M., & Shiraishi, R. W. (2005). Self-transcendence: Conceptualization and measurement. *The International Journal of Aging and Human Development, 60*, 127–143.

Luo, Y., & Jiao, H. (2017). Using the Stan program for Bayesian item response theory. *Educational and Psychological Measurement, 77*, 1–25.

Magis, D., Beland, S., Tuerlinckx, F., & De Boeck, P. (2010). A general framework and an R package for the detection of dichotomous differential item functioning. *Behavior Research Methods, 42*, 847–862.

Mair, P., & De Leeuw, J. (2017). **Gifi**: Multivariate analysis with optimal scaling. R package version 0.3-2. https://R-Forge.R-project.org/projects/psychor/

Mair, P., & Hatzinger, R. (2007a). CML based estimation of extended Rasch models with the **eRm** package in R. *Psychology Science Quarterly, 49*, 26–43.

Mair, P., & Hatzinger, R. (2007b). Extended Rasch modeling: The **eRm** package for the application of IRT models in R. *Journal of Statistical Software, 20*(9), 1–20.

Mair, P., Hofmann, E., Gruber, K., Hatzinger, R., Zeileis, A., & Hornik, K. (2015). Motivation, values, and work design as drivers of participation in the R open source project for statistical computing. *Proceedings of the National Academy of Sciences of the United States of America 112*(48), 14788–14792.

Martin, A. D., Quinn, K. M., & Park, J. H. (2011). **MCMCpack**: Markov Chain Monte Carlo in R. *Journal of Statistical Software, 42*(9), 1–22. http://www.jstatsoft.org/v42/i09/

Masters, G. N. (1982). A Rasch model for partial credit scoring. *Psychometrika, 47*, 149–174.

Maydeu-Olivares, A. (2015). Evaluating the fit of IRT models. In: S. P. Reise & D. A. Revicki (Eds.), *Handbook of item response theory modeling: Applications to typical performance assessment* (pp. 111–127). New York: Routledge.

Maydeu-Olivares, A., & Joe, H. (2005). Limited- and full-information estimation and goodness-of-fit testing in 2^n contingency tables: A unified framework. *Journal of the American Statistical Association, 100*, 1009–1020.

Morgeson, F. P., & Humphrey, S. E. (2006). The work design questionnaire (WDQ): Developing and validating a comprehensive measure for assessing job design and the nature of work. *Journal of Applied Psychology, 91*, 1321–1339.

Muraki, E. (1990). Fitting a polytomous item response model to Likert-type data. *Applied Psychological Measurement, 14*, 59–71.

Muraki, E. (1992). A generalized partial credit model: Application of an EM algorithm. *Applied Psychological Measurement, 16*, 159–176.

Natesan, P., Nandakumar, R., Minka, T., & Rubright, J. D. (2016). Bayesian prior choice in IRT estimation using MCMC and variational Bayes. *Frontiers in Psychology, 7*(1422), 1–11.

Osterlind, S. J., & Everson, H. T. (2009). *Differential item functioning* (2nd ed.). Thoursand Oaks: Sage.

Ostini, R., & Nering, M. L. (2005). *Polytomous item response theory models*. Thousand Oaks: Sage.

Ponocny, I. (2001). Nonparametric goodness-of-fit tests for the Rasch model. *Psychometrika, 66*, 437–459.

Rasch, G. (1960). *Probabilistic models for some intelligence and attainment tests*. Copenhagen: Danish Institute for Educational Research.

Rasch, G. (1961). On general laws and the meaning of measurement in psychology. In *Proceedings of the IV. Berkeley Symposium on Mathematical Statistics and Probability* (Vol. IV, pp. 321–333). Berkeley: University of California Press.

Reckase, M. D. (2009). *Multidimensional item response theory*. New York: Springer.

Reise, S. P. (2012). The rediscovery of bifactor measurement models. *Multivariate Behavioral Research, 47*, 667–696.

Revelle, W. (2017). **psych**: Procedures for psychological, psychometric, and personality research. R package version 1.7.8. http://CRAN.R-project.org/package=psych

Rizopoulos, D. (2006). **ltm**: An R package for latent variable modelling and item response theory analyses. *Journal of Statistical Software, 17*(5), 1–25. http://www.jstatsoft.org/v17/i05/

Robitzsch, A. (2017). **sirt**: Supplementary item response theory models. R package version 1.15-41. https://CRAN.R-project.org/package=sirt

Rosseel, Y. (2012). **lavaan**: An R package for structural equation modeling. *Journal of Statistical Software 48*(2), 1–36. http://www.jstatsoft.org/v48/i02/

Rusch, T., Lowry, P. B., Mair, P., & Treiblmaier, H. (2017). Breaking free from the limitations of classical test theory: Developing and measuring information systems scales using item response theory. *Information & Management, 54*, 189–203.

Samejima, F. (1969). *Estimation of latent ability using a response pattern of graded scores* (Psychometrika monograph supplement, Vol. 17). Chicago: Psychometric Society.

Sidanius, J., & Pratto, F. (2001). *Social dominance: An intergroup theory of social hierarchy and oppression*. Cambridge: Cambridge University Press.

Strobl, C., Kopf, J., & Zeileis, A. (2015). Rasch trees: A new method for detecting differential item functioning in the Rasch model. *Psychometrika, 80*, 289–316.

Suárez-Falcón, J. C., & Glas, C. A. W. (2003). Evaluation of global testing procedures for item fit to the Rasch model. *British Journal of Mathematical and Statistical Society*, 56, 127–143.

Takane, Y., & De Leeuw, J. (1986). On the relationship between item response theory and factor analysis of discretized variables. *Psychometrika, 52*, 393–408.

Tutz, G., & Schauberger, G. (2015). A penalty approach to differential item functioning in Rasch models. *Psychometrika, 80*, 21–43.

van Buuren, S., & Groothuis-Oudshoorn, K. (2011). **mice**: Multivariate imputation by chained equations in R. *Journal of Statistical Software, 45*(3), 1–67. http://www.jstatsoft.org/v45/i03/

Vaughn-Coaxum, R., Mair, P., & Weisz, J. R. (2016). Racial/ethnic differences in youth depression indicators: An item response theory analysis of symptoms reported by White, Black, Asian, and Latino youths. *Clinical Psychological Science, 4*, 239–253.

Verhelst, N. D., Hatzinger, R., & Mair, P. (2007). The Rasch sampler. *Journal of Statistical Software, 20*(4), 1–14. https://www.jstatsoft.org/article/view/v020i04/

von Aster, M., Weinhold Zulauf, M., & Horn, R. (2006). Neuropsychologische Testbatterie für Zahlenverarbeitung und Rechnen bei Kindern (ZAREKI-R) [Neuropsychological Test Battery for Number Processing and Calculation in Children]. Frankfurt: Harcourt Test Services.

Wang, X., Berger, J. O., & Burdick, D. S. (2013). Bayesian analysis of dynamic item response models in educational testing. *The Annals of Applied Statistics, 7*, 126–153.

Wilmer, J. B., Chabris L. G. C. F., Chatterjee, G., Gerbasi, M., & Nakayama, K. (2012). Capturing specific abilities as a window into human individuality: The example of face recognition. *Cognitive Neuropsychology, 29*, 360–392.

Wilson, G. D., & Patterson, J. R. (1968). A new measure of conservatism. *British Journal of Social and Clinical Psychology, 7*, 264–269.

Wirth, R. J., & Edwards, M. C. (2007). Item factor analysis: Current approaches and future directions. *Psychological Methods, 12*, 58–79.

Woolley, A. W., Gerbasi, M. E., Chabris, C. F., Kosslyn, S. M., & Hackman, J. R. (2008). Bringing in the experts: How team ability composition and collaborative planning jointly shape analytic effectiveness. *Small Group Research, 39*, 352–371.

Yen, W. (1981). Using simulation results to choose a latent trait model. *Applied Psychological Measurement, 5*, 245–262.

Zeileis, A., Hothorn, T., & Hornik, K. (2008) Model-based recursive partitioning. *Journal of Computational and Graphical Statistics, 17*, 492–514.

Zeileis, A., Strobl, C., Wickelmaier, F., Komboz, B., & Kopf, J. (2016). **psychotools**: Infrastructure for psychometric modeling. R package version 0.4-2. https://CRAN.R-project.org/package=psychotools

Zumbo, B. D. (1999). *A handbook on the theory and methods of differential item functioning (DIF): Logistic regression modeling as a unitary framework for binary and likert-type (Ordinal) item scores*. Ottawa: Directorate of Human Resources Research and Evaluation, Department of National Defense.

Chapter 5
Preference Modeling

5.1 Models for Paired Comparisons

Let us introduce paired comparison data by means of a simple toy example where 200 persons stated their preferences on five bands. An item consisted of a pair of bands, and participants had to state which of the two bands they prefer. Each participant was exposed to all possible pairs of bands. No undecided answers were allowed in this study.

```
library("MPsychoR")
data("bandpref")
bandpref
##       Band1     Band2 Win1 Win2
## 1    Slayer      Rush  142   58
## 2    Slayer     Death   54  146
## 3    Slayer   Emperor  158   42
## 4    Slayer Scorpions   34  166
## 5      Rush     Death  121   79
## 6      Rush   Emperor  147   53
## 7      Rush Scorpions  155   45
## 8     Death   Emperor   72  128
## 9     Death Scorpions  140   60
## 10 Emperor Scorpions  159   41
```

Each row corresponds to a series of 200 "games." The first row says that Slayer won 142 games against Rush (i.e., 142 persons like Slayer more than Rush), whereas Rush won 58 games against Slayer (i.e., 58 persons like Rush more than Slayer). Having five bands in total, there are $\binom{5}{2} = 10$ possible pairwise comparisons presented to each person (*complete design*). Note that most models presented below work for *incomplete designs* as well and also allow for undecided answers.

P. Mair, *Modern Psychometrics with R*, Use R!,
https://doi.org/10.1007/978-3-319-93177-7_5

5.1.1 Bradley-Terry Model

In this section we introduce a classical model for paired comparison data which maps the objects (i.e., bands in our example) on a latent continuum. The Bradley-Terry model (BT; Bradley and Terry, 1952) estimates a parameter for each band. In performance settings these parameters are often called "abilities," just as in item response theory (cf. Chap. 4). They tell us which bands have similar preferences, which is the least favorite band, which one the most favorite band, etc. In the BT model, the probability that object i is preferred over object j (or i "beats" j) is given by

$$P(i > j) = \frac{\alpha_i}{\alpha_i + \alpha_j}, \tag{5.1}$$

where α_i and α_j are the *object parameters* ("abilities") on an interval scale. Equivalently, the model can be expressed as

$$\text{logit}(P(i > j)) = \lambda_i - \lambda_j, \tag{5.2}$$

with $\lambda_i = \log(\alpha_i)$ as the object parameters on a log-scale.[1]

Let us fit a simple BT model on the band preferences using the **BradleyTerry2** package (Turner and Firth, 2012):

```
library("BradleyTerry2")
bandsBT <- BTm(cbind(Win1, Win2),Band1, Band2, data = bandpref)
bandsAbil <- BTabilities(bandsBT)
round(sort(bandsAbil[,1]), 3)
## Scorpions   Emperor    Slayer     Death      Rush
##    -0.312    -0.024     0.000     0.199     0.380
```

These values are the λ_i's, that is, the abilities on a log-scale (cf. Eq. (5.2)). One object (in this case Slayer) was taken as baseline category and gets a log-ability of 0. This restriction needs to be imposed in order to fix the latent trait at a particular point. The remaining bands are scaled relatively to this object. We see that Rush has the highest ability; in our setting this means that Rush is the most preferred band. The Scorpions are the least preferred band.

If the α-parameterization from Eq. (5.1) is preferred, we can simply say:

[1]Note that there is a strong connection between the BT model and the Rasch model from Sect. 4.2.1. In fact, it can be shown that the Rasch model in its multiplicative form is a special case of the BT model.

```
alphas <- exp(bandsAbil[,1])
sort(alphas)
## Scorpions   Emperor    Slayer     Death      Rush
## 0.7321361 0.9759494 1.0000000 1.2200734 1.4621035
```

This representation is attractive if we want to have an odds interpretation. For instance, the odds that Rush is preferred to Death is

```
alphas["Rush"]/alphas["Death"]
##     Rush
## 1.198373
```

This original version of the BT model can be extended by incorporating object-specific covariates and order effects. This can be achieved by extending Eq. (5.2) in terms of $\lambda_i = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$. That is, the ability parameter is formulated through a regression model with predictor matrix $\mathbf{X}$ and corresponding regression parameter $\boldsymbol{\beta}$. Simple examples can be found in Turner and Firth (2012). In the next two sections, we present two modern approaches for incorporating covariates: recursive partitioning trees and lasso.

5.1.2 Bradley-Terry Trees

Model-based partitioning trees (Zeileis et al., 2008) were already demonstrated in Sect. 4.6.2 within the context of item response models. In BT models the idea is to find predictor splits for which the object parameters differ between the terminal nodes. This strategy is implemented in the **psychotree** package (Strobl et al., 2011). Let us demonstrate the fit of a *BT tree* using the topmodel data from the **psychotree** package. In this experiment 192 respondents judged the attractiveness of the top six contestants (Barbara, Anni, Hana, Fiona, Mandy, Anja) in the TV show "Germany's Next Topmodel." The preferences are stated in terms of paired comparisons, organized as an object of class `"paircomp."`

We study the influence of the covariates gender, age, and three questions regarding whether the participant is familiar with the TV show (`q1`), whether he/she watches the TV show regularly (`q2`), and whether he/she watched the final show (`q3`). The following call computes the tree structure and returns the object parameters for the subgroups in the terminal nodes.

```
library("psychotree")
data("Topmodel2007")
topmtree <- bttree(preference ~ age + gender + q1 + q2 + q3,
                   data = Topmodel2007)
```

Figure 5.1, produced using `plot(topmtree)`, shows the fitted tree structure. Persons older than 52 years have a strong preference for Barbara and low preference for Anna. Participants of age 52 or younger who watch the TV show regularly (`q2="yes"`) have a strong preference for Hana. If they do not watch it regularly,

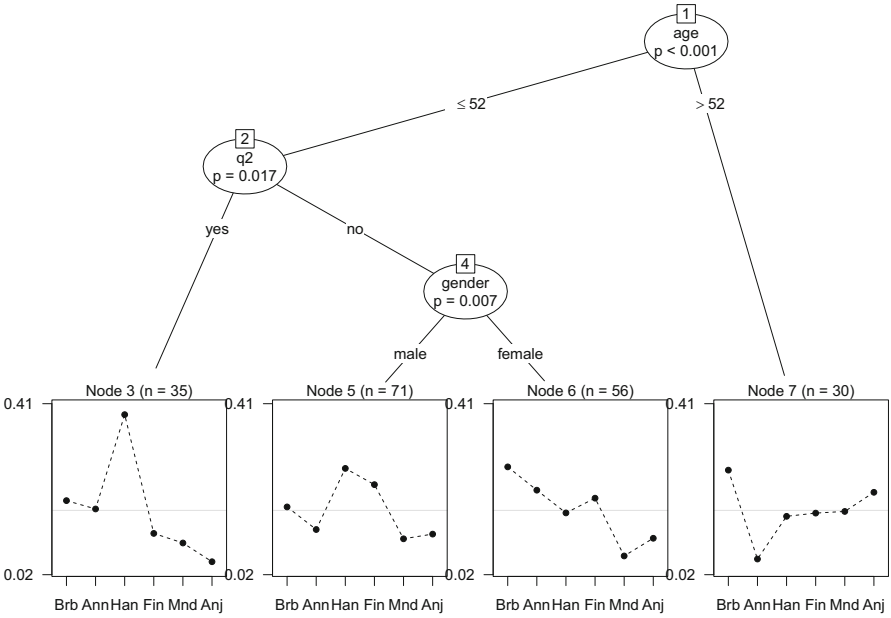

Fig. 5.1 BT tree structure with optimal predictor splits

the algorithm finds another split for gender, since the preferences for this subgroup are different for male and female participants.

5.1.3 Bradley-Terry Lasso

Another option, especially attractive for situations with many predictors/covariates, is to use a *lasso* approach (see James et al., 2013, for an introduction). *BT lasso* models were proposed by Schauberger and Tutz (2017) and perform a predictor

selection with respect to each predictor's influence on the paired comparisons. This approach is implemented in the **BTLLasso** package (Schauberger, 2017). Covariates can be *subject-specific* as well as *object-specific*. In the example below, once more using the topmodel dataset, we only consider subject-specific covariates.

The **BTLLasso** package requires the data in a slightly different form. First, let us convert the paired comparisons (given as `"paircomp"` object) into long format with values of 1 if the first model was preferred and values of 0 if the second model was preferred.

```
pcmat <- as.matrix(Topmodel2007[[1]])
pcvec <- as.vector(pcmat)
pcvec[pcvec == -1] <- 0
```

Now we need to get the labels for the paired comparisons (first model name, second model name) in shape:

```
library("stringr")
modnames <- t(str_split(colnames(pcmat), ":", simplify = TRUE))
```

Next, we extract the predictors, convert them into numerics (starting with 0), and standardize all variables (as generally required in lasso modeling).

```
preds <- Topmodel2007[2:6]
ind <- sapply(preds, is.factor)
preds[ind] <- sapply(preds[ind],
                     function(f) c(0:(length(levels(f))-1))[f])
preds <- scale(preds)
rownames(preds) <- paste0("P", 1:nrow(preds))
head(preds)
##        gender        age         q1         q2         q3
## P1 -0.9973924  2.1396096  1.1675757  0.4709234  0.8796174
## P2 -0.9973924 -0.9476026 -0.8520147 -2.1124278 -1.1309367
## P3  0.9973924 -0.9476026 -0.8520147 -2.1124278 -1.1309367
## P4  0.9973924 -1.0162074 -0.8520147  0.4709234 -1.1309367
## P5 -0.9973924 -0.9476026 -0.8520147 -2.1124278 -1.1309367
## P6  0.9973924 -1.0162074 -0.8520147  0.4709234 -1.1309367
```

The last line prints the covariates for the first six respondents. As a final data preparation step, we create the response object which will then be processed by the `BTLLasso` function. Note that we also need a subject ID since the data are in long format, as well as both object names involved in the paired comparisons.

```
library("BTLLasso")
sid <- rep(rownames(preds), ncol(pcmat))
mfirst <- rep(modnames[1,], each = nrow(preds))
msecond <- rep(modnames[2,], each = nrow(preds))
BTresp <- response.BTLLasso(response = pcvec, subject = sid,
  first.object = mfirst, second.object = msecond)
```

Lasso requires the specification of a *tuning parameter* λ (i.e., shrinkage parameter). Note that this parameter should not be confounded with the log-abilities λ_i in Eq. (5.2). The tuning parameter is typically determined by cross-validation (CV)[2] for which we define a grid of λ-parameters. A standard setup for this grid is values between 0 and 20; but for running time purposes, we keep the grid between 0 and 10 and perform five CV-folds only (10 is be the default). The following chunk fits the BT lasso model based on the optimal λ (CV happens internally).

```
set.seed(123)
lambda <- exp(seq(log(10), log(1), length = 20))-1
modLasso <- cv.BTLLasso(Y = BTresp, X = preds, lambda = lambda,
                        folds = 5, trace = FALSE)
```

The output below shows the parameter estimates. For each model we get an intercept, that is, a baseline preference parameter. The restriction used in this implementation is that the intercepts sum up to 0. For each model we also get the object parameters in relation to the subject-specific covariates. The final line shows the optimal λ found through CV.

```
modLasso
## Intercepts:
##    Anja    Anni Barbara   Fiona    Hana   Mandy
##  -0.371  -0.097   0.350   0.185   0.400  -0.467
##
## Object-specific effects for subject-specific covariate(s):
##          Anja   Anni Barbara  Fiona   Hana  Mandy
## gender -0.115  0.204   0.204 -0.064 -0.115 -0.115
## age     0.162 -0.322   0.162 -0.136 -0.029  0.162
## q1      0.165  0.123   0.012  0.012 -0.207 -0.105
## q2      0.230 -0.101   0.082  0.166 -0.277 -0.101
## q3     -0.152  0.033  -0.152  0.033  0.120  0.120
```

(continued)

[2] An easy-to-read introduction to CV can be found in James et al. (2013).

```
##
## ---
##
## Optimal lambda: 0.8329807
```

From the intercepts we see that Hana is the most preferred model. The object-specific parameters based on the subject-specific covariates across the entire λ-grid are visualized in Fig. 5.2 using `plot(modLasso)`. The vertical line is placed at an optimal λ value of 0.833. Let us have a look at the parameter values for the gender covariate (first row in the object-specific effects output above). We see that Anni and Barbara get the same effect (top path), Fiona has her own parameter (middle path), and Hana, Mandy, and Anja get a single parameter (bottom path). The remaining predictors in conjunction with the plots can be interpreted in the same way.

As a final step, we can request confidence intervals for the parameters using bootstrap. Note that the bootstrap is performed on the level of the CV. That is, the bootstrap samples are again cross-validated based on a specified λ grid. Here, we narrow it down to seven values, located around the optimal value from above. For running time purposes, we keep the number of bootstrap samples low (the user may set it to something like `B = 500`).

```
optind <- which.min(modLasso$criterion)
lambda2 <- lambda[(optind-3):(optind+3)]
set.seed(111)
bootLasso <- boot.BTLLasso(modLasso, lambda = lambda2,
    cores = 4, B = 50, trace = FALSE, trace.cv = FALSE)
```

The plot of the parameters including the confidence intervals (CIs) can be produced as follows and is given in Fig. 5.3.

```
plot(bootLasso, plots_per_page = 6, ask_new = FALSE)
```

It displays the bootstrap point estimates with the 95% CIs around it. Let us pick a few interesting effects which are in line with the BT tree from above (see Fig. 5.1). We see that there is a strong age effect for Anni and strong `q2` effects (in opposite directions) for Hana and Anja. The gender effect for Barbara and Anni came out in the BT tree as well (see Nodes 5 and 6 in Fig. 5.1). However, we have to keep in mind that the two approaches are different: the lasso approach describes the global influence of the covariates on the preferences, whereas the tree approach tries to find fine-grained subgroup combinations that influence the preferences.

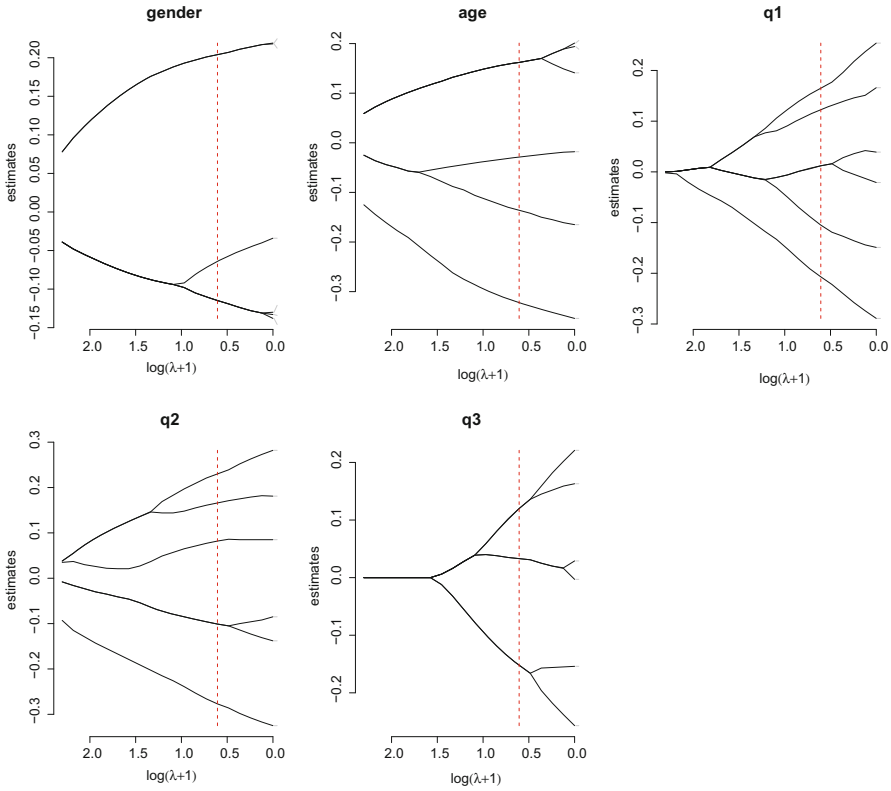

Fig. 5.2 BT lasso parameter trajectories for different λ values. The red line denotes the optimal shrinkage parameter λ found through CV

5.2 Log-Linear Models for Preference

Dittrich et al. (1998) proposed a *log-linear modeling* framework for paired comparisons, ratings, and rankings. It is implemented in the **prefmod** package (Hatzinger and Dittrich, 2012). In this section we focus on a particular version of log-linear preference models called *pattern models* (Dittrich et al., 2002). The *log-linear BT model* assumes that the recorded paired comparisons are independent. In contrast, *pattern models* use the responses of an individual simultaneously by formulating a joint probability distribution of the preferences. Depending on the type of preference data, the package provides the following functions to fit pattern models:

- `pattL.fit` for ratings (e.g., Likert items),
- `pattPC.fit` for paired comparisons,
- `pattR.fit` for (partial) rankings.

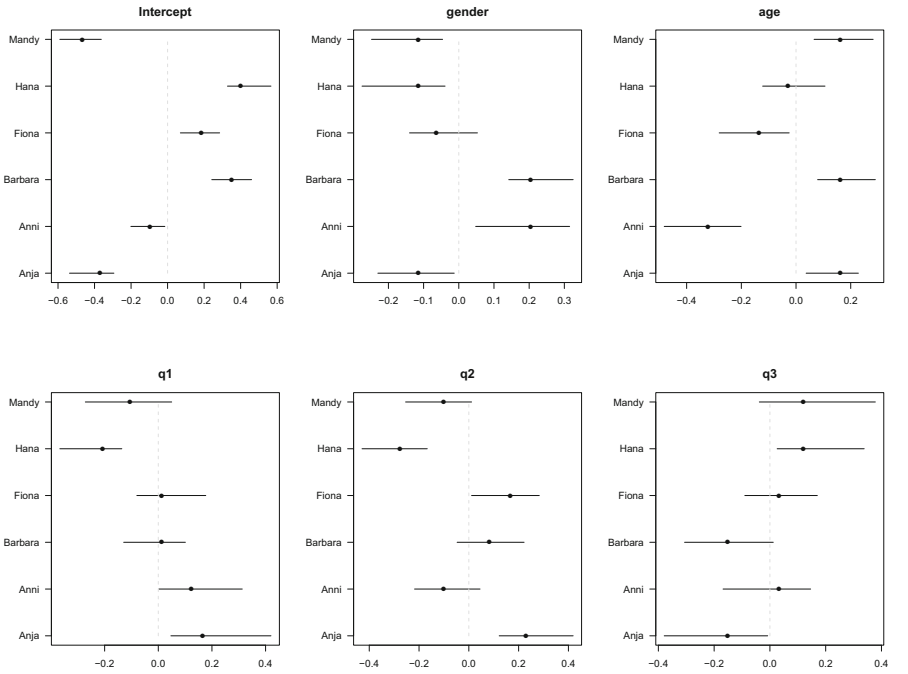

Fig. 5.3 BT Lasso bootstrap CIs for the intercepts and the subject-specific covariate effects

For the ratings and rankings versions, repeated measurement extensions are implemented (see `pattLrep.fit` and `pattRrep.fit`). The package also provides options for incorporating covariates, as we will show below, and is considerably fast for a small amount of rated objects.

5.2.1 Pattern Model for Ratings

Let us first consider a simple example using a dataset from the **prefmod** package related to musical preferences. Each participant had to rate music styles on a scale from 1 ("like it very much") to 5 ("dislike it very much"). Here we pick a subset of five musical styles: easy listening (`"mood"`), reggae (`"regg"`), rap (`"rap"`), heavy metal (`"hvym"`), and contemporary rock music (`"conr"`). We use `sex` as covariate (1 = male, 2 = female).

```
library("prefmod")
music5 <- music[,c("mood", "regg", "rap", "hvym",
                   "conr", "sex")]
head(music5)
##   mood regg rap hvym conr sex
## 1    1    5   5    5    4   1
## 2    3    3   4    5    3   1
## 3    1    4   4    4    3   2
## 4    3    5   5    5    3   2
## 5    3   NA  NA   NA    5   2
## 6    2    3   3    5    3   1
```

As we see, this dataset includes missing values which **prefmod** is able to handle. Note that by default the function estimates an *undecided* parameter. This parameter is useful in our example since we have many tied ratings. Let us fit a pattern model for the ratings with the covariate sex.

```
pattmus <- pattL.fit(music5, nitems = 5, formel = ~sex)
pattmus
##            estimate      se       z p-value
## mood        0.05640 0.02573   2.192  0.0284
## regg       -0.23223 0.02519  -9.219  0.0000
## rap        -0.56284 0.02718 -20.708  0.0000
## hvym       -0.56745 0.02732 -20.770  0.0000
## mood:sex2   0.03543 0.03440   1.030  0.3030
## regg:sex2  -0.00625 0.03332  -0.187  0.8517
## rap:sex2    0.03466 0.03425   1.012  0.3115
## hvym:sex2  -0.13737 0.03571  -3.847  0.0001
## U           0.63768 0.01235  51.634  0.0000
```

The undecided parameter (last row) is positive (and significant). This suggests that there is a strong tendency that participants assigned the same ratings to various musical styles. A negative parameter value would imply the opposite. As far as the music effects are concerned, by default, the function uses the last object (`"conr"`) as reference group. From the main effects, we see that easy listening is the most preferred style and heavy metal the least preferred style. The most interesting part of this output is the significant interaction between heavy metal and gender.

For an easier interpretation of the parameters, we can request the object parameters on a log-scale (i.e., λ_i; see Eq. (5.2)):

```
round(patt.worth(pattmus, outmat = "lambda"), 3)
##        sex1   sex2
## mood  0.056  0.092
```

(continued)

```
## regg -0.232 -0.238
## rap  -0.563 -0.528
## hvym -0.567 -0.705
## conr  0.000  0.000
## attr(,"class")
## [1] "wmat"   "matrix"
```

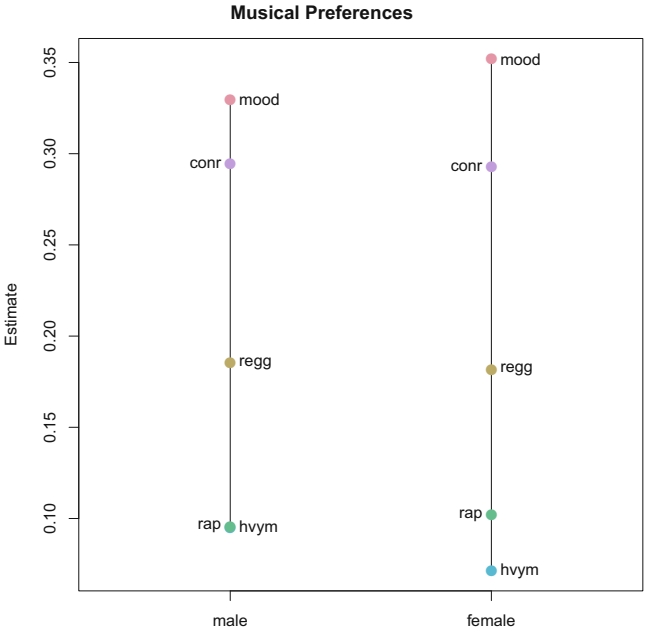

Fig. 5.4 Worth parameters for rating pattern model across gender. The larger the value of the estimate, the more the musical style is liked by the participants

This parameterization reflects nicely the difference in heavy metal preference across gender: females dislike it even more than males. We can also convert the λ parameterization into *worth parameters*. It simply re-parameterizes the output in terms of a sum-0 restriction instead of fixing one parameter to 0.

```
pworth <- patt.worth(pattmus)
plot(pworth, main = "Musical Preferences")
```

Figure 5.4 shows the corresponding plot. We see that easy listening music is highly preferred by both males and females. All musical styles are scored similarly across males and females, except for heavy metal. Males equally dislike heavy metal

and rap, whereas females dislike heavy metal even more than males. This explains the significant interaction in the output above.

5.2.2 Pattern Model for Paired Comparisons

In this section we adapt the pattern model to paired comparisons. We use a discretized version of the data presented in Grand and Dittrich (2015) on learning-related emotions in mathematics. The authors consider achievement emotions students typically experience when learning mathematics. The five emotions are the following: enjoyment (coded as 1), pride (2), anger (3), anxiety (4), and boredom (5). They were interested if the ordering of learning-related emotions depends on particular subject covariates. Let us have a look at the dataset structure:

```
library("MPsychoR")
data("learnemo")
head(learnemo)
##   pc1_2 pc1_3 pc2_3 pc1_4 pc2_4 pc3_4 pc1_5 pc2_5 pc3_5 pc4_5 sex
## 1     2     0     0     0     0     0     0     0     0     0   1
## 2     2     2     0     0     0     1     0     0     1     1   1
## 3     2     2     0     0     0     0     0     0     0     2   1
## 4     0     0     2     2     2     2     0     0     0     0   1
## 5     2     1     0     1     0     1     1     0     1     2   2
## 6     1     2     2     2     2     0     2     2     2     2   1
```

In the columns we have all possible $\binom{5}{2}$ paired comparisons according to the emotion codings. This particular order of paired comparisons is often referred to as *standard order*. The entries have the following meaning: 0 if the first emotion was "preferred" (i.e., more dominant when learning mathematics), 2 if the second emotion was preferred, and 1 if no decision was made. The entries have to be coded in this way such that **prefmod** recognizes the corresponding preference pattern. Each participant ($n = 111$) was exposed to all paired comparisons (*complete design*). We include sex as covariate (1 = male, 2 = female) as well as an explicit undecided parameter (default in the function below). Let us fit the pattern model with boredom as reference category.

```
pattemo <- pattPC.fit(learnemo, nitems = 5, formel = ~ sex,
  obj.names = c("enjoyment", "pride", "anger", "anxiety",
                "boredom"))
pattemo
##
##                estimate      se       z p-value
## enjoyment      -0.03143 0.10237  -0.307  0.7588
```

(continued)

```
## pride           0.27615 0.10435   2.646  0.0081
## anger           0.14168 0.10278   1.379  0.1679
## anxiety        -0.28450 0.10502  -2.709  0.0067
## enjoyment:sex2  0.15310 0.13377   1.145  0.2522
## pride:sex2      0.46263 0.13917   3.324  0.0009
## anger:sex2      0.28106 0.13468   2.087  0.0369
## anxiety:sex2    0.61195 0.13605   4.498  0.0000
## U              -1.44182 0.10070 -14.318  0.0000
```

The undecided parameter, here expressed on a log-odds scale, tells us something about the participants' tendency of stating a preference vs. being undecided. In case of $U = 0$, there is a 50-50 chance to make a decision or not. A positive U means that participants tend to be undecided. A negative U, as in our example, implies that participants tend to make a decision.

For the remaining parameters, it is easier to interpret them using the worth parameterization as produced by `patt.worth(pattemo)`. From the corresponding plot in Fig. 5.5, we see that boredom and anxiety are ordered differently by males and females. Pride is the top emotion for both groups but gets a considerably larger worth estimate for the females than for the males. Anger and enjoyment are fairly similarly scored.

5.2.3 Pattern Model for Rankings

The third pattern model we illustrate uses rankings as input data. In questionnaires or experiments, persons sometimes have to rank items/stimuli according to their preference. If they have to rank all of them, this is called a *full ranking*. In cases of many stimuli, persons are often required to rank only part of the stimuli. Or, as in the example below, it was left up to the respondent whether he/she wanted to rank all the stimuli or only a subset of it. This setup is often referred to as *partial ranking*.

The dataset we use here is from the marketing area and included in the **prefmod** package. Dabic and Hatzinger (2009) studied partial rankings of 435 participants who, through an online configuration system, ranked the following six car attributes: price, exterior design, brand, technical equipment, producing country, and interior design. A rank of 1 denotes the highest preference. Participants had the option to rank only a subset of attributes. This leads to the following partial ranking structure with age as categorical covariate (1 = 17–29 years, 2 = 30–49 years, 3 = 50 years and older).

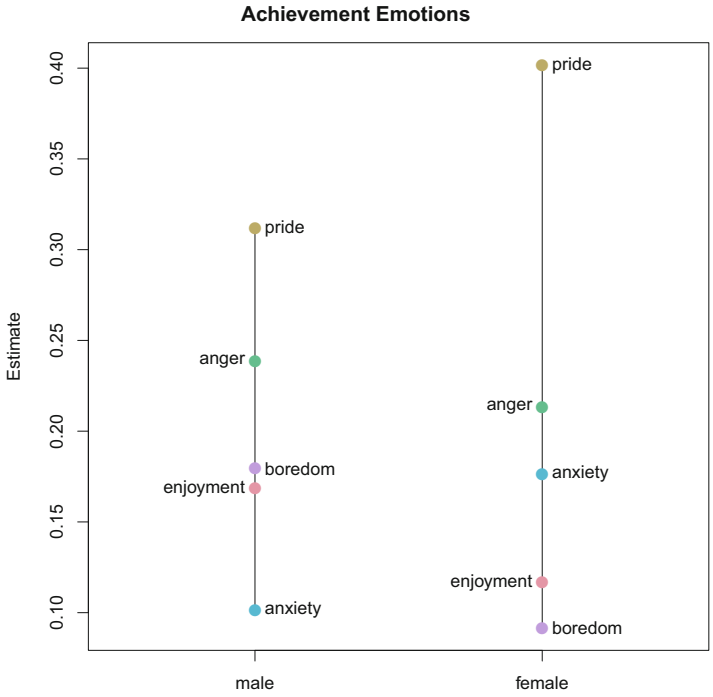

Fig. 5.5 Worth parameters for paired comparison pattern model across gender. The larger the value of the estimate, the more often the emotion is experienced when learning mathematics

```
carconf1 <- carconf[, c(1:6, 8)]
head(carconf1)
##   price exterior brand tech.equip country interior age
## 1     3        2     5          6       4        1   1
## 2     4        1     5          2       6        3   3
## 3     6        3     2          5       4        1   2
## 4     1        4     2          3       6        5   3
## 5    NA        2     4         NA       3        1   2
## 6    NA        2     4          3      NA        1   1
```

From printing the first six participants, we see that participants 1–4 provided a full ranking, whereas participants 5 and 6 ranked four attributes only. The ranking pattern model with age as covariate can be fitted as follows:

```
pattcar <- pattR.fit(carconf1, nitems = 6, formel = ~ age)
pattcar
##                  estimate      se      z p-value
## price           -0.11658 0.02949 -3.953  0.0001
## exterior         0.03656 0.02948  1.240  0.2150
## brand            0.02036 0.02930  0.695  0.4871
## tech.equip      -0.04360 0.02906 -1.500  0.1336
## country         -0.26050 0.03255 -8.003  0.0000
## price:age2       0.07488 0.04499  1.664  0.0961
## exterior:age2    0.01337 0.04521  0.296  0.7672
## brand:age2      -0.02031 0.04476 -0.454  0.6498
## tech.equip:age2  0.03947 0.04469  0.883  0.3772
## country:age2     0.09782 0.04837  2.022  0.0432
## price:age3       0.12133 0.04629  2.621  0.0088
## exterior:age3    0.04558 0.04653  0.979  0.3276
## brand:age3       0.02727 0.04595  0.593  0.5532
## tech.equip:age3  0.15565 0.04670  3.333  0.0009
## country:age3     0.06621 0.05138  1.289  0.1974
```

The first age category (`age1`) is used as baseline. We start interpreting the model by considering the interactions first (as in ANOVA). The 50+-year-old persons differ significantly from the young participants in terms of their price ranking and technical equipments ranking: both attributes are more important to older customers (see `price:age3` and `tech.equip:age3`). Middle-aged participants differ from young participants in terms of country (`country:age2`). The interpretable main effects for exterior and brand are not significant. The worth parameters obtained through `patt.worth(pattcar)` are given in Fig. 5.6 and can be interpreted in a similar fashion as in the other pattern models.

5.3 Other Methods for Preference Data

In Sect. 9.4 we present an exploratory technique for rankings and ratings called *unfolding*. Unfolding represents associations among the rows and the columns of a preference data matrix in a low-dimensional space.

Other paradigms to analyze preference data, beyond the scope of this book, are *conjoint analysis* and *discrete choice experiments*. Both of them have their roots in psychology: Luce and Tukey (1964) and Krantz and Tversky (1971) developed the theory of *conjoint measurement*, whereas Thurstone (1927) developed the *random utility theory* used in discrete choice experiments. Conjoint analysis involves a systematic manipulation of factors (attributes) in a factorial design. In a discrete choice experiment, the respondent has to state his/her preferences based on attribute bundles (choice sets) which are created according to a discrete choice design. A discussion regarding the differences between these two concepts can be found in

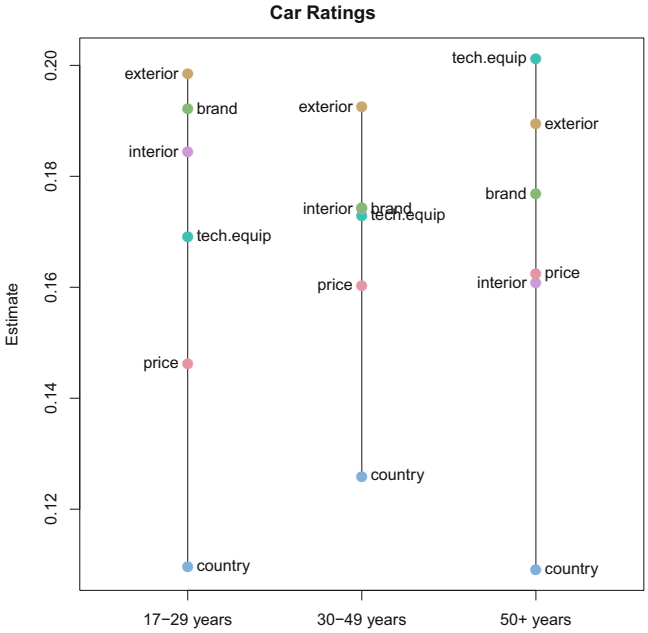

Fig. 5.6 Worth parameters for rating pattern model across age groups

Louviere et al. (2010). In R, conjoint analysis can be computed using the **conjoint** package (Bak and Bartlomowicz, 2012). Discrete choice experiments are typically modeled by discrete choice models (i.e., multinomial logistic regression models and related specifications). The **Rchoice** package (Sarrias, 2016) provides a flexible modeling infrastructure. How to design and analyze such experiments in R is described in detail in Aizaki et al. (2015).

References

Aizaki, H., Nakatami, T., & Sato, K. (2015). *Stated preference methods using R.* Boca Raton: CRC Press.

Bak, A., & Bartlomowicz, T. (2012). Conjoint analysis method and its implementation in conjoint R package. In: J. Pociecha & R. Decker (Eds.), *Data analysis methods and its applications* (pp. 239–248). Warsaw: Beck.

Bradley, R. A., & Terry, M. E. (1952). Rank analysis of incomplete block designs I: The method of paired comparisons. *Biometrika, 39*, 324–345.

Dabic, M., & Hatzinger, R. (2009). Zielgruppenadäquate Abläufe in Konfigurationssystemen: Eine empirische Studie im Automobilmarkt – Partial Rankings [Targeted processes in configuration systems: An empirical study on the car market – Partial rankings]. In R. Hatzinger, R. Dittrich, & T. Salzberger (Eds.), *Präferenzanalyse mit R: Anwendungen aus Marketing, Behavioural Finance und Human Resource Management [Analysis of preferences with R: Applications*

in marketing, behavioral finance and human resource management] (pp. 119–150). Vienna: Facultas.

Dittrich, R., Hatzinger, R., & Katzenbeisser, W. (1998). Modelling the effect of subject-specific covariates in paired comparison studies with an application to university rankings. *Journal of the Royal Statistical Society: Series C (Applied Statistics), 47*, 511–525.

Dittrich, R., Hatzinger, R., & Katzenbeisser, W. (2002). Modelling dependencies in paired comparison experiments. *Computational Statistics & Data Analysis, 40*, 39–57.

Grand, A., & Dittrich, R. (2015). Modelling assumed metric paired comparison data – Application to learning related emotions. *Austrian Journal of Statistics, 44*, 3–15.

Hatzinger, R., & Dittrich, R. (2012). **prefmod**: An R package for modeling preferences based on paired comparisons, rankings, or ratings. *Journal of Statistical Software, 48*(10), 1–31. https://www.jstatsoft.org/v048/i10

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning with applications in R*. New York: Springer.

Krantz, D. H., & Tversky, A. (1971). Conjoint measurement analysis of composition rules in psychology. *Psychological Review, 78*, 151–169.

Louviere, J. J., Flynn, T. N., & Carson, R. T. (2010). Discrete choice experiments are not conjoint analysis. *Journal of Choice Modelling, 3*, 57–72.

Luce, R. D., & Tukey, J. W. (1964). Simultaneous conjoint measurement: A new scale type of fundamental measurement. *Journal of Mathematical Psychology, 1*, 1–27.

Sarrias, M. (2016). Discrete choice models with random parameters in R: The **Rchoice** package. *Journal of Statistical Software, 74*(10), 1–31. https://www.jstatsoft.org/v074/i10

Schauberger, G. (2017). **BTLLasso**: Modelling heterogeneity in paired comparison data. R package version 0.1-7. https://CRAN.R-project.org/package=BTLLasso

Schauberger, G., & Tutz, G. (2017). Subject-specific modelling of paired comparison data: A lasso-type penalty approach. *Statistical Modelling, 17*, 223–243.

Strobl, C., Wickelmaier, F., & Zeileis, A. (2011). Accounting for individual differences in Bradley-Terry models by means of recursive partitioning. *Journal of Educational and Behavioral Statistics, 36*, 135–153.

Thurstone, L. L. (1927). A law of comparative judgment. *Psychological Review, 34*, 273–286.

Turner, H., & Firth, D. (2012). Bradley-terry models in R: The **BradleyTerry2** package. *Journal of Statistical Software, 48*(9), 1–21. http://www.jstatsoft.org/v48/i09/

Zeileis, A., Hothorn, T., & Hornik, K. (2008). Model-based recursive partitioning. *Journal of Computational and Graphical Statistics, 17*, 492–514.

Chapter 6
Principal Component Analysis and Extensions

6.1 Principal Component Analysis

Principal component analysis (PCA) is an exploratory, descriptive technique for dimensionality reduction of multivariate data. The aim of PCA is to form new "prototype" variables, called *principal components* (PCs), out of a set of possibly correlated, metric input variables (*indicators*). These PCs are independent from each other and are formulated through a linear combination of the indicators. The PCA problem is solved by means of a matrix decomposition technique called *singular value decomposition* (SVD).

In the tradition of Joliffe (2002), which is the most comprehensive PCA book on the market, we treat PCA as a purely descriptive multivariate data reduction technique, different from exploratory factor analysis (EFA; see Sect. 2.2), even though these two approaches often lead to similar results.

6.1.1 Singular Value and Eigenvalue Decomposition

The origins of PCA can be traced back to Pearson (1901), although he did not call it PCA. This expression was introduced later by Hotelling (1933), and important analytical developments were made by Eckart and Young (1936). These developments are based on matrix decomposition techniques called *singular value decomposition* (SVD) and *eigenvalue decomposition*. In this section we will invest some time in explaining these decompositions in a nontechnical fashion, since several multivariate methods in subsequent chapters use these techniques as well.

P. Mair, *Modern Psychometrics with R*, Use R!,
https://doi.org/10.1007/978-3-319-93177-7_6

Let $\mathbf{X}$ be our $n \times m$ data matrix. SVD decomposes $\mathbf{X}$ into three parts, each of them giving us valuable structural information about the data. Formally, it is defined as follows[1]:

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}'. \tag{6.1}$$

$\mathbf{U}$ is an $n \times m$ matrix containing the *left singular vectors*, $\mathbf{V}$ is an $m \times m$ matrix containing the *right singular vectors*, and $\mathbf{D}$ is an $m \times m$ diagonal matrix with the *singular values* on the diagonal.

Let us explore these three matrices in more detail based on an artificial example. We simulate a dataset $\mathbf{X}$ with $n = 1000$ observations and $m = 3$ variables, each of them centered (i.e., mean 0).

```
library("mvtnorm")
sigma <- matrix(c(2, 0.8, 0, 0.8, 0.5, 0, 0, 0, 0.1), ncol = 3)
set.seed(123)
baguette <- rmvnorm(1000, mean = c(0,0,0), sigma = sigma)
```

We can visualize the data by means of a 3D scatterplot and discover that our data look like a baguette. SVD aims to describe the structure and orientation of the baguette in the 3D space. Let us see what happens if we perform an SVD on the baguette.

```
svdb <- svd(baguette)
D <- diag(svdb$d)   ## singular values
round(D, 3)
##        [,1]   [,2]   [,3]
## [1,] 47.493  0.000  0.000
## [2,]  0.000 12.238  0.000
## [3,]  0.000  0.000 10.108
V <- svdb$v         ## right singular vectors
round(V, 3)
##        [,1]   [,2]   [,3]
## [1,]  0.917 -0.399 -0.020
## [2,]  0.399  0.916  0.048
## [3,] -0.001 -0.052  0.999
```

$\mathbf{V}$ tells us something about the orientation of the baguette in the 3D space. More technically, it defines three *orthogonal* vectors ($\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$), meaning that they are perpendicular to each other. Formally, this implies that the *inner*

[1]For readers who are more attracted to a sonic introduction to SVD rather than a formal one, Michael Greenacre composed an SVD song ("It had to be U"), available on YouTube.

product (also called *dot product* or *scalar product*) between two vectors is 0 (e.g., $\langle \mathbf{v}_1, \mathbf{v}_2 \rangle = \sum_i v_{i1} v_{i2} = 0$, where $\mathbf{v}_1$ and $\mathbf{v}_2$ are the first two right singular nvectors). In addition, each of these vectors is of *length* 1 which can be stated using the *Euclidean norm*. For instance, $\|\mathbf{v}_1\| = \sqrt{v_{11}^2 + v_{21}^2 + v_{31}^2} = 1$. Vectors that are orthogonal and of length 1 are called *orthonormal*. Let us check these properties for the first two vectors in R:

```
v1 <- V[,1]
v2 <- V[,2]
round(v1 %*% v2, 7)  ## inner product
##      [,1]
## [1,]    0
sqrt(sum(v1^2))      ## length
## [1] 1
sqrt(sum(v2^2))
## [1] 1
```

Back to the geometric interpretation in relation to our baguette. The first vector in **V** points in the "long" baguette direction; the remaining two vectors describe the "round" part of the baguette (i.e., a slice). The user can visualize the baguette and the vectors as follows (plot not shown here):

```
library("rgl")
plot3d(baguette, col = "gray", xlim = c(-4, 4), ylim = c(-4,4),
       zlim = c(-4,4), aspect = 1, size = 2)
arrow3d(c(0,0,0), V[,1], col = "red")
arrow3d(c(0,0,0), V[,2], col = "red")
arrow3d(c(0,0,0), V[,3], col = "red")
```

Each vector in **V** is associated with a singular value in the *diagonal matrix* **D**. These singular values tell us something about the variance of the points along these new axes. Note that singular values are always in descending order, that is, the first one is always the largest and the last one always the smallest. From the output above, we see that the first singular value is by far the largest since it tells us something about the stretch of the data on the first axis ("long" baguette direction). The remaining two singular values are of approximately the same length since they describe the round-ish slice. This information can be incorporated in the 3D plotting code above by, for instance, multiplying each singular vector with the square root of its associated singular value. This stretches/squeezes the arrows according to the variance in the respective direction.

If our data would look more like a pizza, the first two singular values would be roughly the same, whereas the third singular value would be tiny in case of a thin

crust pizza and increases as the pizza approaches Chicago style. If we do not care about whether the pizza is thin or thick crust, we can ignore the third eigenvalue: the first two eigenvalues describe the properties we are interested in. This idea is important when it comes to using SVD in PCA as a dimension reduction technique, since in practice we consider the first few eigenvalues/eigenvectors only, hoping that they describe the most important properties (i.e., variance) of the data.

The role of the **U** matrix will be described in the next section when looking at SVD within a PCA context. For the moment we will merely use it to check whether **X** can be reconstructed from the three SVD matrices, according to Eq. (6.1):

```
U <- svdb$u
baguette1 <- U %*% D %*% t(V)
identical(round(baguette, 7), round(baguette1, 7))
## [1] TRUE
```

Another decomposition, strongly related to SVD, is called *eigenvalue decomposition*. Whereas SVD works for general $n \times m$ matrices, an eigenvalue decomposition is restricted to *square matrices* (i.e., matrices with the same number of rows and columns). In statistical applications we often apply correlation or covariance matrices. For an $m \times m$ covariance matrix $\boldsymbol{\Sigma}$, we get

$$\boldsymbol{\Sigma} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}'. \tag{6.2}$$

U is the $m \times m$ orthogonal matrix containing the *eigenvectors*, and $\boldsymbol{\Lambda}$ an $m \times m$ diagonal matrix containing the *eigenvalues*. Here the eigenvalues play a similar role as the singular values in SVD: they provide us with information about the variance in the data.

```
covb <- cov(baguette)
eigend <- eigen(covb)
L <- diag(eigend$values)    ## eigenvalues
round(L, 3)
##       [,1] [,2]  [,3]
## [1,] 2.258 0.00 0.000
## [2,] 0.000 0.15 0.000
## [3,] 0.000 0.00 0.102
U <- eigend$vectors         ## eigenvectors
round(U, 3)
##        [,1]   [,2]   [,3]
## [1,]  0.917  0.399 -0.021
## [2,]  0.399 -0.916  0.050
## [3,] -0.001  0.054  0.999
```

The eigenvectors in **U** correspond to the right singular vectors in **V** from above. Since in the covariance formula we divide by $n - 1$, and given the fact that an eigenvalue is the square of a singular value, we can apply the following transformation in order to make eigenvalues and singular values match:

```
n <- nrow(baguette)
l <- (diag(D)/sqrt(n-1))^2  ## from singular values
round(l, 3)                 ## eigenvalues
## [1] 2.258 0.150 0.102
```

The bottom line is that SVD is a generalization of an eigenvalue decomposition in terms of input matrices with unequal number of rows and columns. The next section shows which role these matrix decompositions play in PCA.

6.1.2 PCA Computation

The general PCA data setup is the following. Let $\mathbf{X}$ be an $n \times m$ data matrix with m metric variables. PCA extracts m new independent variables $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \ldots, \boldsymbol{\xi}_m$ called PCs. The set of linear equations relating the PCs to the indicators is the following:

$$\begin{aligned}
\boldsymbol{\xi}_1 &= w_{11}\mathbf{x}_1 + w_{12}\mathbf{x}_2 + \cdots + w_{1m}\mathbf{x}_m \\
\boldsymbol{\xi}_2 &= w_{21}\mathbf{x}_1 + w_{22}\mathbf{x}_2 + \cdots + w_{2m}\mathbf{x}_m \\
&\vdots \\
\boldsymbol{\xi}_m &= w_{m1}\mathbf{x}_1 + w_{m2}\mathbf{x}_2 + \cdots + w_{mm}\mathbf{x}_m.
\end{aligned} \tag{6.3}$$

$\mathbf{W}$ is an $m \times m$ matrix containing the *weights* (also called *loadings*), and $\boldsymbol{\Xi}$ an $n \times m$ matrix containing the *principal component scores* (PC scores). In crisp matrix notation, Eq. (6.3) can be rewritten as

$$\boldsymbol{\Xi} = \mathbf{X}\mathbf{W}'. \tag{6.4}$$

Comparing these equations to Eqs. (2.2) and (2.3) from EFA (see Sect. 2.2.1), we see clear differences. EFA formulates a statistical model that includes an error term (unique factors), whereas PCA expresses the PCs by means of a simple linear combination, which is easy to solve. Thus, PCA is a purely descriptive method without involving a statistical model and which can be illustrated geometrically (see Sharma, 1996).

PCA can be computed in two ways: we can either perform an eigenvalue decomposition on the covariance or correlation matrix or an SVD on the centered

version of $\mathbf{X}$, divided by $\sqrt{n-1}$. In R, the `princomp` function takes the eigenvalue decomposition route, whereas `prcomp` uses SVD. As the `prcomp` help file points out, SVD is the preferred method for numerical accuracy reasons.

Let us illustrate both approaches by means of a simple dataset from Willerman et al. (1991), involving the following three indicators: verbal IQ (VIQ), performance IQ (PIQ), and brain size (measured in MRI pixel counts).

```
library("MPsychoR")
data("BrainIQ")
X <- BrainIQ[, c("VIQ", "PIQ", "MRI_Count")]
head(X, 4)
##   VIQ PIQ MRI_Count
## 1 132 124    816932
## 2 150 124   1001121
## 3 123 150   1038437
## 4 129 128    965353
```

In this dataset the variables are measured on different units: the brain size magnitude is in the 1M area, whereas the VIQ and PIQ are on the usual IQ scale. If we throw these data into a PCA, the brain size variable would dominate the solution because of the differences in magnitude compared to the other variables. The reason for this is that PCA tries to explain variance in the data. Due to the magnitude of the brain size values, the variance of this variable is much larger than the variance in the IQ values. Thus, the first PC would be fully determined by brain size simply because of the measurement units, something we most likely want to avoid. Whenever we encounter a scenario like this, the data should be standardized (mean 0, variance 1). If all indicators are on the same measurement units, we can either standardize or not, depending on to which degree the sample variances should affect the solution.

Let us proceed with this example and, first, compute a PCA via an eigenvalue decomposition (through `princomp` and by hand). Using the correlation matrix instead of the covariance matrix implies standardization.

```
PCAfit <- princomp(X, cor = TRUE)
PCAfit
## Call:
## princomp(x = X, cor = TRUE)
##
## Standard deviations:
##    Comp.1    Comp.2    Comp.3
## 1.4256144 0.8647244 0.4689087
##
##  3  variables and  40 observations.
round(unclass(PCAfit$loadings), 3) ## weights (loadings)
```

(continued)

```
##             Comp.1 Comp.2 Comp.3
## VIQ         -0.627 -0.356  0.693
## PIQ         -0.639 -0.272 -0.719
## MRI_Count -0.445  0.894  0.057
evIQ <- eigen(cor(X))              ## eigenvalue decomposition
sqrt(evIQ$values)                  ## standard deviations
## [1] 1.4256144 0.8647244 0.4689087
round(evIQ$vectors, 3)             ## weights (loadings)
##        [,1]   [,2]   [,3]
## [1,] -0.627 -0.356  0.693
## [2,] -0.639 -0.272 -0.719
## [3,] -0.445  0.894  0.057
```

Since we have three indicators, we can extract three PCs. These new variables have standard deviations that correspond to the square roots of the eigenvalues. The loadings matrix $\mathbf{W}$ corresponds to the $\mathbf{U}$ matrix from Eq. (6.2). Loadings tell us to which extend each indicator is associated with each component. Based on these loadings, we can label the PCs, similar to factors in EFA. From the eigenvalues we can directly determine the amount of explained variance of each component.

```
summary(PCAfit)
## Importance of components:
##                           Comp.1    Comp.2     Comp.3
## Standard deviation     1.4256144 0.8647244 0.46890866
## Proportion of Variance 0.6774588 0.2492495 0.07329178
## Cumulative Proportion  0.6774588 0.9267082 1.00000000
evIQ$values/sum(evIQ$values)
## [1] 0.67745877 0.24924945 0.07329178
```

Second, we perform the same PCA fit based on an SVD involving the standardized data matrix (and divided by $\sqrt{n-1}$). This is what `prcomp` is doing internally.

```
PCAfit2 <- prcomp(X, scale = TRUE)
print(PCAfit2, digits = 3)
## Standard deviations (1, .., p=3):
## [1] 1.426 0.865 0.469
##
## Rotation (n x k) = (3 x 3):
##              PC1    PC2     PC3
## VIQ         -0.627  0.356 -0.6926
## PIQ         -0.639  0.272  0.7191
## MRI_Count -0.445 -0.894 -0.0569
```

We get the same component standard deviations as above and the same loadings (here called `rotation`). Note that some signs can be switched which only matters with respect to the directional interpretation of the corresponding PC.

If we want to do the same computation by hand via an SVD call, we can say

```
n <- nrow(X)
svdIQ <- svd(scale(X)/sqrt(n-1))      ## SVD
round(svdIQ$d, 3)          ## singular values
## [1] 1.426 0.865 0.469
round(svdIQ$v, 3)          ## right singular vectors
##        [,1]   [,2]   [,3]
## [1,] -0.627  0.356 -0.693
## [2,] -0.639  0.272  0.719
## [3,] -0.445 -0.894 -0.057
```

This gives again the same result as in the `prcomp` call. The component scores can be computed from the SVD output using $\mathbf{UD}\sqrt{n-1}$ or simply via Eq. (6.4):

```
head(PCAfit2$x, 4)
##             PC1        PC2         PC3
## [1,] -0.3262983  1.5890699 -0.08878183
## [2,] -1.9374031 -0.4173517 -0.76173409
## [3,] -2.1895933 -0.9709755  0.83271829
## [4,] -1.2734095 -0.2432148  0.01031501
head(svdIQ$u %*% diag(svdIQ$d)*sqrt(n-1), 4)
##            [,1]       [,2]        [,3]
## [1,] -0.3262983  1.5890699 -0.08878183
## [2,] -1.9374031 -0.4173517 -0.76173409
## [3,] -2.1895933 -0.9709755  0.83271829
## [4,] -1.2734095 -0.2432148  0.01031501
```

So far we have shown how a PCA can be computed via simple matrix decompositions: we get the amount of explained variance, the loadings, and the PC scores. We see that PCA computation is much simpler than EFA, which typically uses maximum likelihood (ML) or least squares (LS) estimation. Another important difference between PCA and EFA arises when we consider fitting them on the basis of a covariance matrix $\boldsymbol{\Sigma}$. EFA concentrates on explaining the off-diagonal elements of $\boldsymbol{\Sigma}$, whereas PCA concentrates on the diagonal elements of $\boldsymbol{\Sigma}$. Thus, EFA tries to explain covariance in the data, and PCA tries to explain variance in the data (although it is not totally blind to the covariances). Regarding the PC scores, they are a direct result of the SVD, as opposed to EFA where the factor scores are computed post hoc using one out of several possible methods (see Sect. 2.2.3).

As far as the dimensionality p of the solution is concerned, in EFA, we had to fix p before fitting the model. In PCA we typically fit the full solution and

then determine p using simple ad hoc tools such as a scree plot (elbow criterion, eigenvalue larger 1 criterion). We used this strategy already in Sect. 2.2.4 when we produced a scree plot within an EFA context. What we actually did was fitting a PCA (since it is so simple) and explored the dimensionality structure of the PCA solution which gives us a good hint of the number of factors. This is feasible since EFA and PCA often lead to similar results, despite the differences in model formulation and computation.

Other indices from the EFA world (see Sect. 2.2.4) that can be used for PCA are parallel analysis (since it is simply a more sophisticated version of a scree plot with randomly (re-)sampled data) and very simple structure (VSS). They are implemented in the **psych** package (Revelle, 2017) which also provides the `principal` function for fitting PCA. If VSS is used for PCA, the user needs to set the argument `fm="pc"` in the corresponding `vss` call. As in EFA, we should look at these criteria in combination instead of relying on a single rule of thumb. Parametric EFA criteria such as the TLI and RMSEA cannot be applied to since they are directly based on ML or LS estimation outcomes (which are not used in PCA).

Finally, in EFA, we typically apply a rotation for better interpretability (see Sect. 2.2.2). Note that in EFA the rotation does not change the solution. In PCA, care is advised when applying rotation techniques since it changes the solution. For instance, rotation redistributes the component variances across the new, rotated components (in fact, we should not call them principal components anymore). Pros and cons of PCA rotations are discussed in detail in Joliffe (2002, Chapter 11). The `principal` function in the **psych** package has options for orthogonal and non-orthogonal component rotations. We will show how to rotate a PCA solution in the next section by means of a fully worked out real-life data example. We will also illustrate several practical issues related to dimensionality assessment, component interpretation, and visualization.

6.1.3 PCA Application and Practical Issues

To illustrate PCA on a real-life dataset, we use data from Treiblmaier (2006, see also Treiblmaier et al. 2011). The dataset contains items that measure various advantages and disadvantages online users perceive when providing personal information on the Internet. We consider six items relating to advantages of personal communication (variable names starting with `apc`) and four items related to disadvantages of personal communication (those starting with `dpc` in the variable names). Each item was measured using a slide bar (from 1 to 100). The correlation structure of these variables is shown in Fig. 2.6 (see Sect. 2.3).

Note that PCA does not assume that the indicators are (multivariate) normally distributed (Joliffe, 2002, p. 19) since, as pointed out above, it is a purely descriptive technique with no statistical model formulation underneath. This said, it certainly does not hurt if the data are somewhat multivariate normal which guarantees that

the relationship between all observed variables is linear. This is the only assumption we really have in PCA which stems from the fact that PCA can be fitted on the basis of a correlation matrix (and Pearson correlations are of course blind to nonlinear relationships). Linearity among the variables can be explored using the `scatterplotMatrix` function from the **car** (Fox and Weisberg, 2011). Standard transformations such as the logarithm can be applied, if it helps in terms of making the data "more linear" or "more normal." Details on transformations can be found in Fox and Weisberg (2011).

Let us continue with the privacy example. Since all the items are on the same 1–100 scale, we could fit either a PCA without standardization or a PCA with standardization, as discussed above. Here we use the standardized version and produce a scree plot, as introduced in Sect. 2.2.4, to get a first picture of the explained variance structure:

```
data("Privacy")
pcaPriv <- prcomp(Privacy, scale = TRUE)
screeplot(pcaPriv, type = "lines", main = "Privacy Scree Plot")
abline(h = 1, col = "gray", lty = 2)
```

The scree plot in Fig. 6.1 does not really show a clear elbow. The eigenvalues drop below 1 after three components. With three components we explain 59% of the variance in the original data. Let us proceed with $p = 3$ and explore the loadings:

```
round(pcaPriv$rotation[,1:3], 3)
##          PC1   PC2    PC3
## apc1 -0.442 0.079 -0.408
## apc2 -0.426 0.103 -0.414
## apc3 -0.388 0.191 -0.276
## apc4 -0.305 0.155  0.362
## apc5 -0.338 0.226  0.492
## apc6 -0.362 0.148  0.430
## dpc1  0.228 0.425 -0.001
## dpc2  0.164 0.500 -0.102
## dpc3  0.153 0.493 -0.129
## dpc4  0.179 0.420  0.029
```

The second component is the easiest one to interpret since all the disadvantage items load highly on it. We could label it as "disadvantages of personal communication." The advantage items load highly on component 1 (again, we use the absolute loading values to judge this). The third component discriminates between the first three `apc` items and the remaining three `apc` items. Still, these two components are tricky to interpret. In such cases a rotation can help, as implemented in the **psych** package. However, we need to keep in mind that it changes the solution, as discussed in the last paragraph of the previous section.

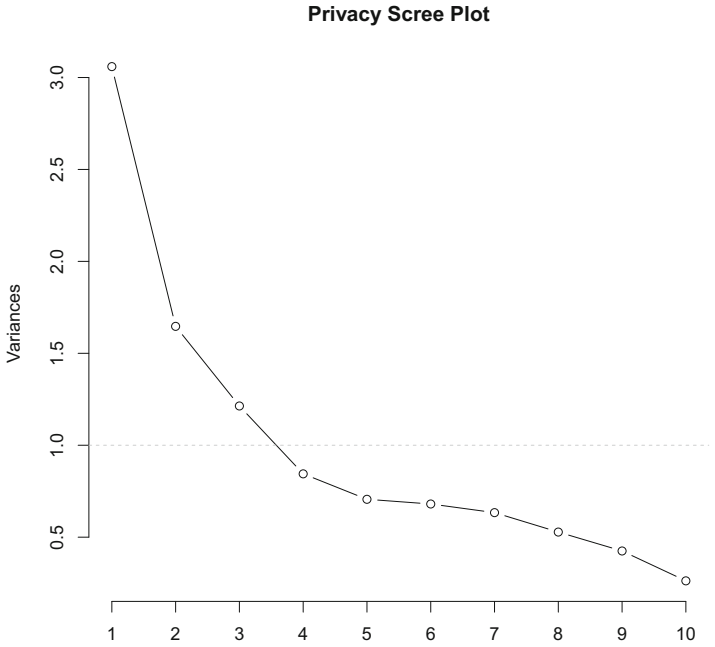

Fig. 6.1 Scree plot for Privacy dataset. The number of components is on the x-axis and eigenvalue on the y-axis

Let us first re-fit the unrotated solution using the `principal` function from **psych** in order to make sure that we are getting the same results.

```
library("psych")
pcaPriv1 <- principal(cor(Privacy), 3, rotate = "none")
pcaPriv1$loadings
##
## Loadings:
##       PC1    PC2    PC3
## apc1  0.772  0.101  0.450
## apc2  0.745  0.132  0.456
## apc3  0.679  0.245  0.304
## apc4  0.533  0.198 -0.398
## apc5  0.591  0.290 -0.542
## apc6  0.634  0.190 -0.474
## dpc1 -0.399  0.545
## dpc2 -0.287  0.642  0.112
## dpc3 -0.268  0.633  0.143
## dpc4 -0.314  0.539
##
```

(continued)

```
##                   PC1   PC2   PC3
## SS loadings     3.059 1.647 1.214
## Proportion Var  0.306 0.165 0.121
## Cumulative Var  0.306 0.471 0.592
```

Note that the loadings are not the same as in the `prcomp` output. The reason for this is that `principal` standardizes the loadings differently, their sum-of-squares (SS) equal to the square of the corresponding eigenvalue (i.e., the component variance; see `pcaPriv1$values`), whereas `prcomp` and `princomp` standardize them to SS equal to 1. However, these are just two different ways of standardizing; the loadings structure is the same, and it does not change anything in the results.

Let us proceed with an orthogonally rotated solution (varimax):

```
pcaPriv2 <- principal(cor(Privacy), 3, rotate = "varimax")
pcaPriv2$loadings
##
## Loadings:
##      RC1    RC3    RC2
## apc1  0.874  0.152 -0.149
## apc2  0.865  0.141 -0.109
## apc3  0.742  0.250
## apc4  0.163  0.673
## apc5  0.130  0.843
## apc6  0.183  0.786 -0.104
## dpc1 -0.166         0.651
## dpc2                0.710
## dpc3                0.697
## dpc4 -0.129         0.611
##
##                  RC1   RC3   RC2
## SS loadings    2.187 1.897 1.836
## Proportion Var 0.219 0.190 0.184
## Cumulative Var 0.219 0.408 0.592
```

Note that the components are labelled with `RC` (*rotated components*). We obtain a much clearer loadings picture: the first three `apc` items load highly on `RC1` (we could label it as "individualized communication"), the remaining `apc` items load highly on `RC3` ("providing correct data"), and the four `dpc` items determine `RC2` ("disadvantages of personal communication"). We see that a rotation makes the interpretation easier, but the price we pay is that important properties of PCA are getting lost. For instance, we see that the proportions of explained variance got redistributed across the components (second part of the output).

If desired, we can also perform a non-orthogonal promax rotation which, in this example, gives an even clearer loadings structure:

```
pcaPriv3 <- principal(cor(Privacy), 3, rotate = "promax")
pcaPriv3$loadings
##
## Loadings:
##      RC1    RC3    RC2
## apc1  0.905
## apc2  0.900
## apc3  0.749  0.106  0.101
## apc4         0.683
## apc5         0.875
## apc6         0.797
## dpc1 -0.126         0.639
## dpc2                0.720
## dpc3  0.101         0.711
## dpc4 -0.104         0.604
##
##                   RC1   RC3   RC2
## SS loadings     2.234 1.889 1.815
## Proportion Var 0.223 0.189 0.181
## Cumulative Var 0.223 0.412 0.594
round(pcaPriv3$Phi, 3)
##        RC1    RC3    RC2
## RC1  1.000  0.406 -0.201
## RC3  0.406  1.000 -0.127
## RC2 -0.201 -0.127  1.000
```

The last line shows the correlations among the rotated components. Not too surprisingly, the correlation between the two components involving the `apc` items (i.e., `RC1` and `RC3`) is fairly high.

A final remark is related to the visualization of PCA results. A standard plotting strategy is the *PCA biplot*, which maps both the component scores and the loadings (as vectors) into the component space. Using the `prcomp` fit, a biplot can be produced as follows (see Fig. 6.2):

```
biplot(pcaPriv, col = c("gray", "black"), cex = c(0.6, 0.9))
abline(h = 0, v = 0, lty = 2)
```

Since biplots are a general concept that can be applied beyond PCA, this book dedicates an entire chapter to this topic (see Chap. 10). For the particular case of PCA biplots, corresponding details regarding interpretation and various scaling options will be given in Sect. 10.3.

6.2 Some PCA Variants

Multiple regression models do not like highly correlated predictors (*multicollinearity*): parameter estimates become sensitive to small changes in the data. Assuming it makes sense conceptually, an elegant option to deal with this phenomenon is to run a PCA on the predictors, which gives p independent PCs that replace the original predictors. After regressing the response on the PCs, the parameter vector is transformed back to the scale of the original predictors. This simple strategy is called *principal component regression*. An extension of this method is called *partial least squares regression* which includes the response variable in the dimension reduction strategy as well. Both approaches are implemented in the **pls** package (Mevik et al., 2016), and for a simple description including R code, see James et al. (2013).

PCA, as introduced so far, is designed for metric indicators and not able to handle categorical input variables or mixed scale levels. *Categorical PCA*, also called *Princals*, will be introduced in Sect. 8.2 within a wider context of Gifi models.

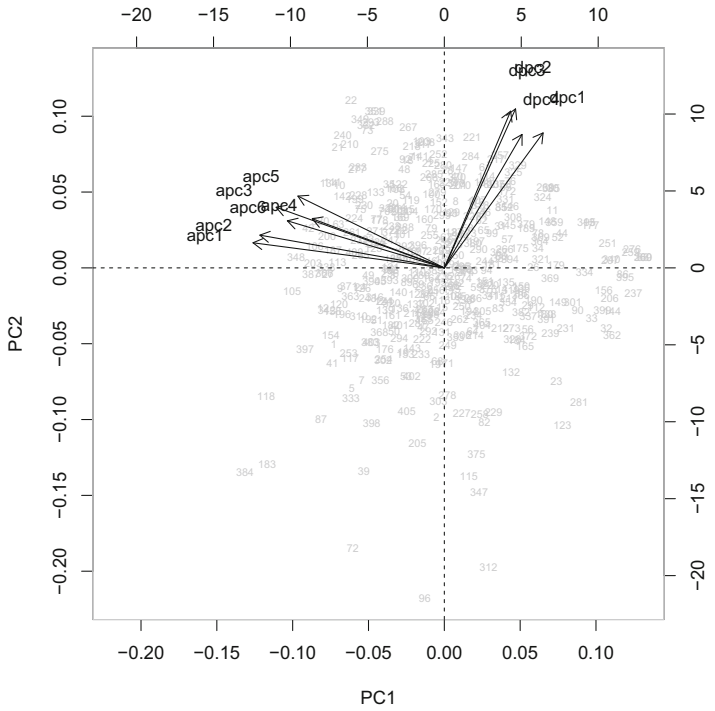

Fig. 6.2 Biplot for unrotated PCA on Privacy data (first two dimensions/components are plotted)

These models also aim to linearize relationships among the indicators which is a very attractive feature since, as we have discussed, PCA assumes linear relationships among the indicators.

Note that in Gifi slang, Princals is also called *nonlinear PCA*. This is due to the fact that it generally performs nonlinear transformations of the indicators. It

does not mean that Eq. (6.3) is extended to something nonlinear. If we are looking for a PCA version that generalizes principal components from straight lines to curves, the **pcaMethods** package (Stacklies et al., 2007) offers several possibilities. Along these lines we should also mention *kernel PCA*, implemented in the **kernlab** package (Karatzoglou et al., 2004).

If our input data contain outliers, a robust version of PCA can be considered. It is straightforward to compute. We can use the `princomp` function with a robust covariance matrix, computed via `cov.rob` from **MASS** (Venables and Ripley, 2002). If we want to have the data standardized (i.e., using a robust correlation matrix), we can simply set `cor=TRUE`. Let us illustrate this strategy using the Privacy data once more (output not shown here).

```
set.seed(123)
pcarob <- princomp(covmat = MASS::cov.rob(Privacy), cor = TRUE)
```

Since this dataset did not have any strong outliers, the results are approximately the same as for the regular PCA fit.

Another useful variant of PCA is called *sparse PCA* (Zou et al., 2006). We have seen in the examples so far that the loadings are in general non-zero, which often makes components difficult to interpret. For a fixed p, sparse PCA shrinks small loadings to 0 using the lasso principle. A simple way of fitting a sparse PCA is to use `spca` function from the **elasticnet** package (Zou and Hastie, 2012) and set `sparse="varnum"`. This way, through the `para` argument, we can tell the algorithm how many non-zero loadings we want to have on each component. Below we say that we want to have three non-zero loadings on PC1 and PC3 and four non-zero loadings on PC2. Alternatively, one could also specify a vector of penalty parameters (see help file).

```
library("elasticnet")
spcaPriv <- spca(scale(Privacy), K = 3, sparse = "varnum",
                 para = c(3, 4, 3))
spcaPriv
##
## 3 sparse PCs
## Pct. of exp. var. : 22.0 17.4 15.6
## Num. of non-zero loadings :  3 4 3
## Sparse loadings
##         PC1   PC2   PC3
## apc1 -0.618 0.000 0.000
## apc2 -0.603 0.000 0.000
## apc3 -0.505 0.000 0.000
## apc4  0.000 0.000 0.494
```

(continued)

```
## apc5  0.000 0.000 0.647
## apc6  0.000 0.000 0.581
## dpc1  0.000 0.486 0.000
## dpc2  0.000 0.534 0.000
## dpc3  0.000 0.520 0.000
## dpc4  0.000 0.456 0.000
```

Compared to the standard PCA fit from above, sparse PCA leads to a more restricted solution. In total, we explain 55.01% of the variance. This is only slightly less than the 59% from the standard PCA fit above, but the solution is much easier to interpret. A variant of sparse PCA is *nonnegative sparse PCA*, as implemented in the **nsprcomp** package (Sigg and Buhmann, 2008), with the option to avoid negative loadings.

Another version of PCA will be presented in Sect. 13.4.4. It operates on functional data and is correspondingly called *functional PCA*.

6.3 Three-Way Principal Component Analysis

In this section we introduce a structural extension of PCA in terms of three-way input data. In order to facilitate elaborations on this extension, let us first consider a data array taxonomy involving *ways* and *modes* (see Carroll and Arabie, 1980).

The most common way to indicate a data array $\mathbf{X}$ is a matrix of size[2] $I \times J$. $\mathbf{X}$ is a two-way two-mode data structure. The *way* indicates the shape of $\mathbf{X}$. Since $\mathbf{X}$ has rows and columns, it has two ways. The term *mode* refers to the content of each way. For many psychological data, the rows in a data matrix $\mathbf{X}$ represent persons, and the columns are the variables. Thus, there is a different entity (or content) for each way which makes the data two-mode. If we construct a covariance (or correlation) matrix $\boldsymbol{\Sigma}$, it has two ways but only one mode (usually variables).

We can extend the above terminology to three-way data for which we use the notation $\underline{\mathbf{X}}$. In such cases, $\underline{\mathbf{X}}$ is a three-way array (or *tensor*) of dimension $I \times J \times K$. In psychological applications, typical modes in $\underline{\mathbf{X}}$ are persons (first mode), variables (second mode), and time points, occasions, or within-subject conditions (third mode).

Below focus on the two most popular, classical methods for three-way (three-mode) data: the *Parafac* method and the *Tucker* method. Other multiway methods are presented in great detail in Kroonenberg (2008). For the particular case of

[2]In this section we follow a notation for indices of the data and components as well as the number of them which is different from that used before but which is standard in the literature of three-way analysis (see Kiers, 2000).

longitudinal three-way data (i.e., time as the third mode), elaborations can be found in Timmerman (2001).

6.3.1 Parafac

Parafac (parallel factor model) was proposed by Harshman (1970). Note that at the same time Carroll and Chang (1970) developed their *Candecomp* (canonical decomposition) method, which is equivalent to Parafac. Let P be the number of components to be extracted. Parafac decomposes the three-way array $\underline{\mathbf{X}}$ with elements x_{ijk} ($i = 1, \ldots, I$; $j = 1, \ldots, J$; $k = 1, \ldots, K$) into the following three components[3]:

$$\hat{x}_{ijk} = \sum_{p=1}^{P} a_{ip} b_{jp} c_{kp}, \tag{6.5}$$

resulting in three component matrices: $\mathbf{A}$ (of dimension $I \times P$) represents the *subject score matrix*, $\mathbf{B}$ ($J \times P$) is the *variable components matrix* (loadings), and $\mathbf{C}$ ($K \times P$) is the *occasion component matrix*. The elements a_{ip}, b_{jp}, and c_{kp} can be interpreted in terms of relative importance of component p for the i-th individual, the j-th variable, and the k-th occasion. Thus, Parafac finds components that are common to all three modes. Note that Eq. (6.5) is formulated in terms of the fitted values $\hat{x}_{ijk}$ for given dimensionality P. If the model fits our data well, $\hat{x}_{ijk}$ is close to the observed x_{ijk}.

Let us apply the Parafac model on a longitudinal three-way dataset from Sidanius et al. (2010) on social dominance orientation (SDO), introduced in Sect. 2.4.5. Four SDO items were presented to 612 participants at five points in time (1996–2000). Since the dataset is stored in wide format, the first step is to convert it into a three-way array structure such that we can fit a Parafac:

```
data("SDOwave")
SDOar <- array(unlist(SDOwave), dim = c(612, 4, 5))
dnames <- dimnames(SDOar) <- list(1:612, paste0("SDO", 1:4),
                                  1996:2000)
dim(SDOar)
## [1] 612   4   5
```

[3]We omit the matrix formulation for all three-way PCA models since it requires some 3D matrix operations that are beyond scope of this book (see, e.g., Kroonenberg, 2008, for corresponding expressions).

This gives the persons × item × time points three-way three-mode array structure.

Data preprocessing in terms of standardization is slightly more complex than in ordinary PCA (see Kroonenberg, 2008, Chapter 6 for details). In the SDO dataset, all variables are measured on the same units (7-point response scale). Thus, there is no need for full standardization. In order to facilitate the interpretation, we center the data for each variable at each occasion (i.e., we center across the first mode). As a consequence, the mean of the variables is located at the origin of the three-way PCA space.

```
library("multiway")
SDOarc <- ncenter(SDOar, mode = 1)
dimnames(SDOarc) <- dnames
```

Now we are ready to fit the Parafac model. R offers several packages to fit three-way PCA models. A comprehensive implementation is **ThreeWay** by Giordani et al. (2014) which uses a prompt input. Another package for three-way computations is **PTAk** (Leibovici, 2010). For our purposes, however, the most attractive package is **multiway** (Helwig, 2017). Let us fit a Parafac with $P = 2$ dimensions using this package:

```
set.seed(111)
sdopara <- parafac(SDOarc, nfac = 2)
str(sdopara[1:4])
## List of 4
##  $ A  : num [1:612, 1:2] -0.729 -0.784 0.2 0.65 -0.222 ...
##  $ B  : num [1:4, 1:2] 0.531 0.176 -1.495 -1.206 1.713 ...
##  $ C  : num [1:5, 1:2] 0.661 0.654 0.747 0.733 0.636 ...
##  $ SSE: num 10373
```

The function returns the three component matrices from Eq. (6.5). In Parafac, these matrices are generally not orthogonal which implies that the components are linearly related. Let us have a look at the variable components matrix **B**, containing the variable loadings:

```
dimnames(sdopara$B) <- list(dnames[[2]], paste0("Comp.", 1:2))
round(sdopara$B, 3)  ## loadings
##      Comp.1 Comp.2
## SDO1  0.531  1.713
## SDO2  0.176  0.963
```

(continued)

```
## SDO3 -1.495  0.213
## SDO4 -1.206  0.304
```

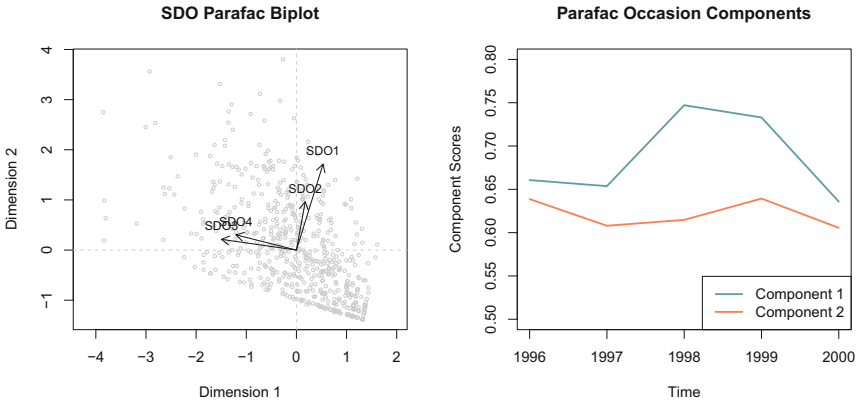

Fig. 6.3 Left panel: Parafac biplot (subject components and loadings; Parafac components are not orthogonal). Right panel: Component trajectories across occasions

We see that the last two items (`SDO3` and `SDO4`) load highly on the first component. This items are related to social equality aspects; thus, we can label the first component "equality" (as in standard PCA, the sign only matters in terms of interpretation direction). The first two items (`SDO1` and `SDO2`) are related to group hierarchies and load highly on the second component. This component can be labeled "group hierarchy."

The left panel of Fig. 6.3 shows the Parafac biplot[4] based on the matrices **A** (subject scores) and **B** (loadings). Since the components are not orthogonal, a more informative representation of this Parafac fit is displayed in the right panel. It shows the occasion component scores from matrix **C**, showing the overall SDO development trajectories of the two SDO subdimensions over 5 successive years. We see that "equality" (component 1) increases from 1997 to 1998 and decreases in the following years, whereas "group hierarchy" (component 2) is almost constant over time.

The **multiway** package also reports an overall R^2 value of 0.489 which tells us how much variance is accounted for by the two dimensions/components. The error sum-of-squares (SSE) can be computed from the *structural image* $\underline{\hat{\mathbf{X}}}$, scaled by the dimension product:

[4]Biplots are introduced in more detail in Chap. 10.

```
Xhat <- fitted(sdopara)  ## structural image
SSE <- sum((SDOar - Xhat)^2)/prod(dim(SDOar))
round(SSE, 3)
## [1] 5.522
```

For dimensionality assessment one can fit multiple Parafac models, produce a scree plot with the number of dimensions on the x-axis and the SSEs on the y-axis, and obtain a decision using the elbow criterion. As p increases, the SSE decreases. Note that the three-mode scree plot shows a point for each model, whereas the PCA scree plot gives points for each component separately. Additional tools for goodness-of-fit assessment can be found in Kroonenberg (2008, Chapter 8).

As a final remark, a few words regarding the *uniqueness* of Parafac. In Parafac, no rotations are possible without worsening the fit. It is especially this aspect which has given rise to its popularity in various fields. The model is properly identified, while Tucker methods, introduced below, are not.

6.3.2 Tucker

The Tucker method (Tucker, 1966) abandons the idea of only one set of components for all modes. This makes Tucker more flexible than Parafac, but, at the same time, it can be more difficult to interpret. In this section we focus on the so-called *Tucker3* model for three-way three-mode data (see Kroonenberg, 2008, for other Tucker model versions). It can be formulated as follows:

$$\hat{x}_{ijk} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} a_{ip} b_{jq} c_{kr} g_{pqr}. \tag{6.6}$$

Compared to Eq. (6.5), there are two obvious differences. First, each mode gets its own specification regarding the number of components to be extracted (in the current example, P for the persons, Q for the variables, and R for the occasions). The resulting *subject component scores matrix* $\mathbf{A}$ is of dimension $I \times P$, the *variable component matrix* $\mathbf{B}$ is $J \times Q$, and the *occasion component matrix* $\mathbf{C}$ is $K \times R$. Note that the number of components need to follow the *minimum-product rule*: the product of the number of components of two modes must always be equal or larger than the third mode component (i.e., $PQ \geq R$, $PR \geq Q$, and $QR \geq P$). As opposed to Parafac, the basic algorithm for the Tucker model produces orthogonal components. These components can be rotated, if desired.

Second, there is an additional term g_{pqr} which forms the so-called *core array* $\underline{\mathbf{G}}$ of dimension $P \times Q \times R$. We will illustrate the role of this matrix after fitting the Tucker3 model on the SDO data, using the **multiway** package. In the present

example, we use $P = 3$ dimensions for the persons, $Q = 2$ for the items, and $R = 2$ for the occasions.

```
set.seed(111)
ndims <- c(3, 2, 2)  ## set P, Q, R
sdotuck <- tucker(SDOar, nfac = ndims)
```

Let us have a closer look at the four matrices given in Eq. (6.6) and fitted in the `tucker` call. The person component matrix **A** can be interpreted in terms of P prototype individuals. Each person in the sample is represented according to a linear combination of these prototype individuals.

As in Parafac, the variable component matrix **B** contains the loadings:

```
dimnames(sdotuck$B) <- list(dnames[[2]], paste0("Comp.", 1:2))
round(sdotuck$B, 3)
##       Comp.1 Comp.2
## SDO1 -0.465 -0.721
## SDO2 -0.352 -0.380
## SDO3 -0.612  0.467
## SDO4 -0.533  0.343
```

Compared to the Parafac fit above, the separation of the items into two components is less clear. In the right panel of Fig. 6.4, loadings and subject scores are represented jointly by means of a biplot. The occasion component scores **C**, as visualized in the right panel of Fig. 6.4, are represented as prototype trajectories or latent curves. Note that in the Parafac model, we had only one interpretation of the components. In the Tucker3 model, the occasion components have a different meaning than the variable components and the person components. Here, they simply represent two prototype trajectories: one where SDO stays constant over time and another one where SDO generally increases (apart from 1999 to 2000).

The core array plays a similar role as the singular value matrix **D** in standard PCA, but it is structurally more complicated (it is a non-diagonal three-way array). From Eq. (6.6) we see that each g_{pqr} plays the role of a weight in the multiplicative three-way component expression $a_{ip}b_{jq}c_{kr}$. Thus, $\underline{\mathbf{G}}$ contains information about the relative importance of component combinations with respect to all modes, or, said differently, it indicates the strength of the links between the components.

```
round(sdotuck$G, 2)
## , , 1
```

(continued)

```
##
##          [,1]   [,2]
## [1,] -254.07   0.04
## [2,]    0.00  50.27
## [3,]    0.04   0.54
##
## , , 2
##
##         [,1]   [,2]
## [1,]   -0.29   0.79
## [2,]    0.75  -1.41
## [3,] -39.62  -0.26
```

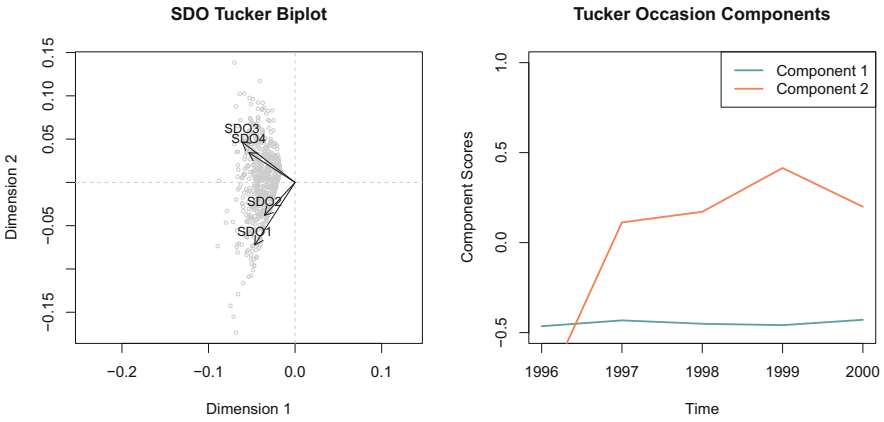

Fig. 6.4 Left panel: Tucker3 biplot (subject components and loadings). Right panel: Tucker3 occasion components

For instance, $g_{3,1,1}$ is very close to 0 (i.e., 0.04, to be precise). It means that this component combination barely contributes toward the structural image $\hat{\underline{\mathbf{X}}}$. This is not the case for $g_{3,1,2} = -39.62$, which is considerably large. This particular component combination (i.e., third-person component, first variable component, first occasion component "constant") contributes highly to the solution. In general, the squared core element divided by the total SS is the proportion of the total sum of squares accounted for by the combination of components.

In terms of dimensionality assessment, the same strategies can be applied as outlined above. One can fit a series of Tucker3 models with systematically varying numbers of P, Q, and R (the minimum-product rule needs to be fulfilled) and produce a scree plot based on the SSE. Rotation strategies can be applied to the Tucker3 model since, as opposed to Parafac, the solution is not unique and a rotated solution can be found without changing the fit.

To conclude, in this section, we applied three-way PCA strategies to study longitudinal developments (third mode). As mentioned at the beginning, the models presented here can handle other kinds of three-way data structures such as three-way profile data or three-way rating scale data. Again, details can be found in Kroonenberg (2008). In this book we will consider three-way data once more in Sect. 9.5.2, when introducing individual differences scaling (INDSCAL) within the context of multidimensional scaling.

6.4 Independent Component Analysis

6.4.1 ICA Formulation

Independent component analysis (ICA; Hyvärinen et al., 2001; Stone, 2004) is often regarded as an extension of PCA, even though it is formally more similar to EFA. It is especially attractive to apply on certain kinds of temporal, spatial, and spatio-temporal signals. In psychology, we encounter such data in experiments involving electroencephalography (EEG) measurements or functional magnetic resonance imaging (fMRI) scans. The aim of ICA is to separate (*unmix*) the data, resulting in a set of *independent components* (ICs). So far, in this chapter, we loosely used the term "independent" when talking about component properties in standard PCA. Actually, we should have used the term "uncorrelated" since independence is a stronger statistical requirement than uncorrelatedness (elaborations on independence vs. uncorrelatedness can be found in Stone, 2004, Chapter 5). ICA gives truly independent components.

The fundamental ICA equation is the following. Let $\mathbf{x} = (x_1(t), x_2(t), \ldots, x_n(t))'$ be a set of observed input signals. For instance, this could be a set of EEG signals from n electrodes or a set of fMRI signals from n voxels. ICA claims that this set of input signals is a mix of a set of independent source signals $\mathbf{s} = (s_1(t), s_2(t), \ldots, s_n(t))'$:

$$\begin{pmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{pmatrix} = \mathbf{A} \begin{pmatrix} s_1(t) \\ s_2(t) \\ \vdots \\ s_n(t) \end{pmatrix}. \tag{6.7}$$

In compact matrix form, we can write

$$\mathbf{x} = \mathbf{A}\mathbf{s}, \tag{6.8}$$

which looks similar to the basic EFA expression in Eq. (2.3). $\mathbf{A}$ is the so-called *mixing matrix* and plays a similar role as the loadings matrix $\boldsymbol{\Lambda}$ in EFA. However, EFA and ICA are estimated in a different way. As we have briefly outlined in

Sect. 2.2.1, some EFA estimation routines assume multivariate normality. ICA assumes that the data are non-normal and is therefore sometimes referred to as *non-Gaussian factor analysis*. This makes it well-suited for signal analysis since the distribution of a signal is typically "peakier" than a normal distribution would allow for.

An alternative representation of Eq. (6.8) is $\mathbf{s} = \mathbf{W}\mathbf{x}$, where the source signal is expressed by means of a *mixing matrix* $\mathbf{W}$, which mixes the input signal. Regardless which representation we use, ICA is a dimension reduction technique: we aim for a small number p of ICs (i.e., source signals), which approximate our observed input data in a satisfactory way.

Having spatiotemporal data as in EEG or fMRI, ICA can be used in two complementary ways: *temporal ICA* (tICA) which extracts temporally independent components and *spatial ICA* (sICA) which extracts spatially independent components. For instance, if we apply a tICA on EEG or fMRI signals, the ICs are temporal trajectories which are independent from each other. Of course, we can also plot an activity map for each component, but these activity maps are not independent from each other. Conversely, if we apply a sICA, we get a time trajectory for each component, but they are not independent from each other. In this case, the activation maps are independent from each other. In theory, we can apply both sICA and tICA on EEG/fMRI data. In practice, in EEG applications, tICA may be more informative, whereas in fMRI sICA is more frequently applied, as demonstrated in Sect. 14.4. Additional details on tICA vs. sICA can be found in Stone (2004, Section 7.7).

In R, ICA can be fitted using the **ica** package (Helwig, 2015b) or the **fastICA** package (Marchini et al., 2013). For EEG applications, the **eegkit** package (Helwig, 2015a) provides an ICA implementation as well as several other useful functions for this type of data, as shown in the next section.

6.4.2 Example: ICA on EEG Data

In this section we present an ICA application using an EEG dataset from an experiment on memory storage capacity.[5] To estimate individual differences in visual working memory storage capacity, a variant of the bilateral color identification task (see Vogel and Machizawa, 2004) was used. Participants were instructed to maintain eye gaze at a central fixation cross, while attending to a cued hemifield. Subsequently, a memory display was presented consisting of one or three colored circles within each hemifield. Participants had to encode the colors in the cued hemifield, store them in memory during a retention period, and subsequently report the identity of a target item. There were four conditions in the experiment, resulting from crossing the set size (number of colored circles presented) with the hemifield (ipsilateral vs. contralateral): set size 1/ipsilateral activity, set size 1/contralateral activity, set size 3/ipsilateral activity, and set size 3/contralateral activity.

[5] Thanks for Hrag Pailian for sharing this dataset.

We start our analysis with some EEG data manipulation and visualization using the **eegkit** package. First we import the data, create a vector with the electrode names as used in the experiment, and produce a 2D and a 3D plot showing the electrode positions (see Fig. 6.5):

```
library("eegkit")
data("storcap")
elecvec <- c("FP1","FP2", "F3", "F4", "FC5", "FC6", "T7", "T8",
  "FC1", "FC2", "C3", "C4", "P7", "P8", "P3", "P4", "CP5",
  "CP6", "CP1", "CP2", "PO7", "PO8", "PO3", "PO4", "O1", "O2")
eegcap(elecvec, type = "2d", col.point = "gray",
       col.label = "black", cex.label = 0.8, cex.point = 4)
data(eeghead)
shade3d(eeghead)
eeghead$material$color <- rep(1,length(eeghead$material$color))
wire3d(eeghead)
eegcap(elecvec, col.point = "coral4", cex.point = 0.3,
       col.label = "coral4", head = FALSE, add = TRUE)
```

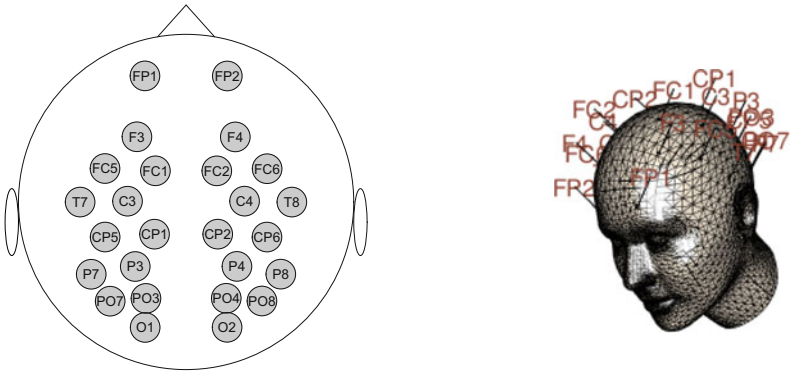

Fig. 6.5 Electrode arrangement in 2D and 3D

Note that in the `storcap` data object, the voltage of left-right hemisphere electrodes is averaged. For instance, channel `PO3_4` is the average of PO3 and PO4. Let us plot a single-channel voltage trajectory (we use `PO3_4`) for each of the four conditions. The following code chunk computes the average voltage (and standard deviation) across participants for each time point and within each condition.

```
library("plyr")
PO34 <- subset(storcap, channel == "PO3_4")    ## select PO3/4
PO34agg <- ddply(PO34, .(time, cond), summarize,
                 mean = mean(voltage), sd = sd(voltage))
```

Figure 6.6 is produced using the `eegtime` function:

```
library("colorspace")
ylims <- range(PO34agg$mean)
cols <- rainbow_hcl(4, 80)
eegtime(PO34agg$time[PO34agg$cond == "SS1_Ips"], xlab = "Time",
  voltage = PO34agg$mean[PO34agg$cond == "SS1_Ips"],
  main = "Electrode PO3/4", vcol = cols[1], ylim = ylims)
eegtime(PO34agg$time[PO34agg$cond == "SS1_Contra"],
  voltage = PO34agg$mean[PO34agg$cond == "SS1_Contra"],
  add = TRUE, vcol = cols[2])
eegtime(PO34agg$time[PO34agg$cond == "SS3_Ips"],
  voltage = PO34agg$mean[PO34agg$cond == "SS3_Ips"],
  add = TRUE, vcol = cols[3])
eegtime(PO34agg$time[PO34agg$cond == "SS3_Contra"],
  voltage = PO34agg$mean[PO34agg$cond == "SS3_Contra"],
  add = TRUE, vcol = cols[4])
abline(v = c(300, 1200), lty = 2, col = "gray")
legend("bottomright", c("Set Size 1 (Ipsilateral)",
  "Set Size 1 (Contralateral)", "Set Size 3 (Ipsilateral)",
  "Set Size 3 (Contralateral)"), lty = 1,col = cols, bty = "n")
```

As the experimental design controls for task-general processing across each hemifield, calculating the difference in neural activity between the "attended-to" (contralateral activity) and "not-attended-to" (ipsilateral activity) sides provides a way to isolate storage-related activity. This difference, called the *contralateral delay activity* (CDA; see Arend and Zimmer, 2011, for details), becomes increasingly more negative as a function of the number of items that are stored. The following code chunk takes the voltage difference between the contralateral and the ipsilateral activity and organizes the results in a new data frame including the new condition labels.

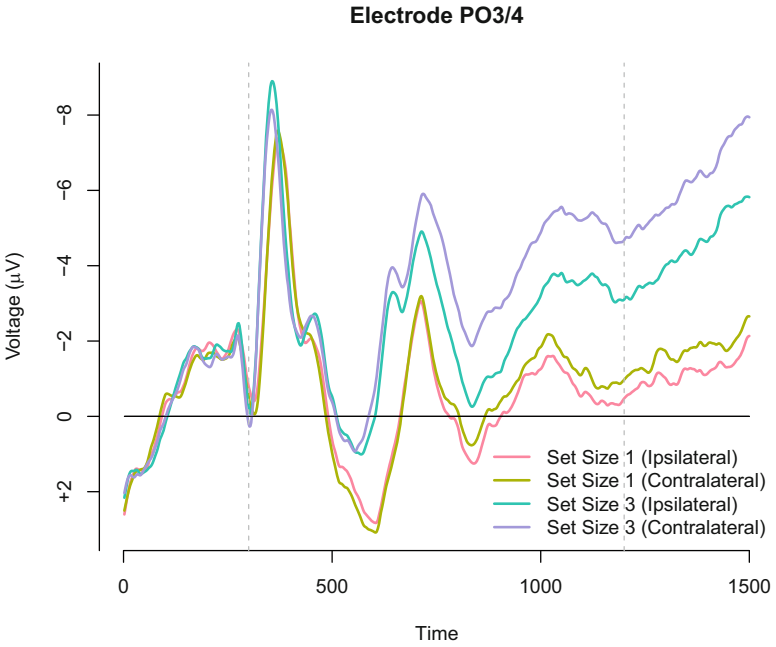

Fig. 6.6 Single channel (PO3/4) voltage trajectory plot for 4 conditions. The vertical lines separate the experimental periods: 0–300 ms memory display, 300–1200 ms consolidation period, >1200 ms test period

```
storcap1 <- storcap
SS1_CDA <- storcap1[storcap1$cond == "SS1_Contra",]$voltage -
  storcap1[storcap1$cond == "SS1_Ips",]$voltage
SS3_CDA <- storcap1[storcap1$cond == "SS3_Contra",]$voltage -
  storcap1[storcap1$cond == "SS3_Ips",]$voltage
n <- length(c(SS1_CDA, SS3_CDA))/2
storcap1 <- storcap1[1:(2*n),]
storcap1$voltage <- c(SS1_CDA, SS3_CDA)
storcap1$cond <- factor(rep(c("SS1_CDA", "SS3_CDA"), each = n))
```

Instead of plotting the activity in the temporal domain, let us plot the activation in the spatial domain for a fixed time point (we pick 700 ms). We first need to compute the voltage means within each condition and channel. Due to the fact that in the original data the voltage was already averaged for left-right electrodes, we subsequently need to produce a copy for each electrode mean in order to get the spatial coordinates right. This leads to a symmetric activation picture.

```
storcap700 <- subset(storcap1, time == 700)
CDA700agg <- ddply(storcap700, .(channel, cond), summarize,
                   mean = mean(voltage))
CDA700agg1 <- CDA700agg2 <- CDA700agg
even <- seq(2, nrow(CDA700agg), 2)    ## even row selector
odd <- seq(1, nrow(CDA700agg), 2)     ## odd row selector
CDA700agg1[even, ] <- CDA700agg1[odd, ]
CDA700agg2[odd, ] <- CDA700agg2[even, ]
data(eegcoord)        ## coordinate template from eegkit
coords <- eegcoord[elecvec, 1:3]      ## 3D coordinates
```

The `eegspace` function allows us to produce 2D as well as 3D spatial activation plots. Figure 6.7 shows an example of a 3D plot for each CDA condition.

```
eegspace(coords, CDA700agg1[,3], main = "Set Size 1 CDA",
         vlim = range(CDA700agg[,3]), colorlab = "")
eegspace(coords, CDA700agg2[,3], main = "Set Size 3 CDA",
         vlim = range(CDA700agg[,3]), colorlab = "")
```

So far we only did descriptive analyses. Let us now fit two tICAs: one for the set size 1 CDA condition and one for the set size 3 CDA condition. For each condition we compute the average voltage for each time point within each channel, organized as matrix of dimension 13×750 (i.e., electrodes in the rows and time points in the columns). This matrix acts as input to the `eegica` call. For both tICA fits, we extract four ICs:

```
CDA1 <- subset(storcap1, cond == "SS1_CDA")
CDA1agg <- daply(CDA1, .(channel, time),
                 function(x) mean(x$voltage))
tempICA1 <- eegica(CDA1agg, nc = 4, type = "time")
CDA3 <- subset(storcap1, cond == "SS3_CDA")
CDA3agg <- daply(CDA3, .(channel, time),
                 function(x) mean(x$voltage))
tempICA3 <- eegica(CDA3agg, nc = 4, type = "time")
```

Let us check how much variance accounted for (VAF) we get for both conditions:

```
round(tempICA1$vafs, 3)
## [1] 0.423 0.410 0.088 0.045
round(tempICA3$vafs, 3)
## [1] 0.885 0.066 0.020 0.016
```

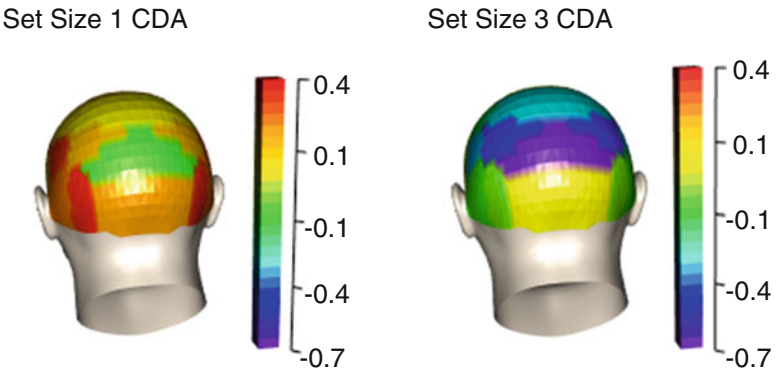

Fig. 6.7 3D spatial EEG activations plots for two different CDA conditions

The ICA for the first condition leads to two dominating components, whereas in the second solution, the first IC explains most of the variance. Let us plot the relevant ICs in the temporal as well as in the spatial domain, for both conditions separately. Again, since for the left-right electrodes the voltage was averaged, we need to apply the same "copy trick" as above in order to get the proper coordinates in the spatial plot. Since we performed a tICA, in the time domain, we use the corresponding column of the mixing matrix (**A** in Eq. (6.8); `tempICA1$M` and `tempICA3$M` in the code below), whereas in the spatial domain, we use the source signal estimates (**s** in Eq. (6.8); `tempICA1$S` in the code below).

```
tvec <- unique(storcap1$time)
mixmat1 <- tempICA1$M[rep(1:nrow(tempICA1$M), each = 2), ]
mixmat3 <- tempICA3$M[rep(1:nrow(tempICA3$M), each = 2), ]
vlims <- range(c(mixmat1[,1], mixmat1[,2], mixmat3[,1]))
eegtime(tvec, tempICA1$S[, 1],
        main = "Component 1 (Set Size 1 CDA)", vcol = 1)
eegspace(coords[, 1:2], mixmat1[, 1], vlim = vlims)
eegtime(tvec, tempICA1$S[, 2],
        main = "Component 2 (Set Size 1 CDA)", vcol = 1)
eegspace(coords[, 1:2], mixmat1[, 2], vlim = vlims)
eegtime(tvec, tempICA3$S[, 1],
        main = "Component 1 (Set Size 3 CDA)", vcol = 1)
eegspace(coords[, 1:2], mixmat3[, 1], vlim = vlims)
```

Figure 6.8 shows the resulting plots. Note that since we fitted a tICA, we obtained temporally independent components within each condition. For the set size 1/CDA condition, we get two important ICs (memory and filtering) which imply that the visual system may not be taxed enough such that there is some information transfer where the ipsilateral hemifield also starts to represent the colored item (two sources:

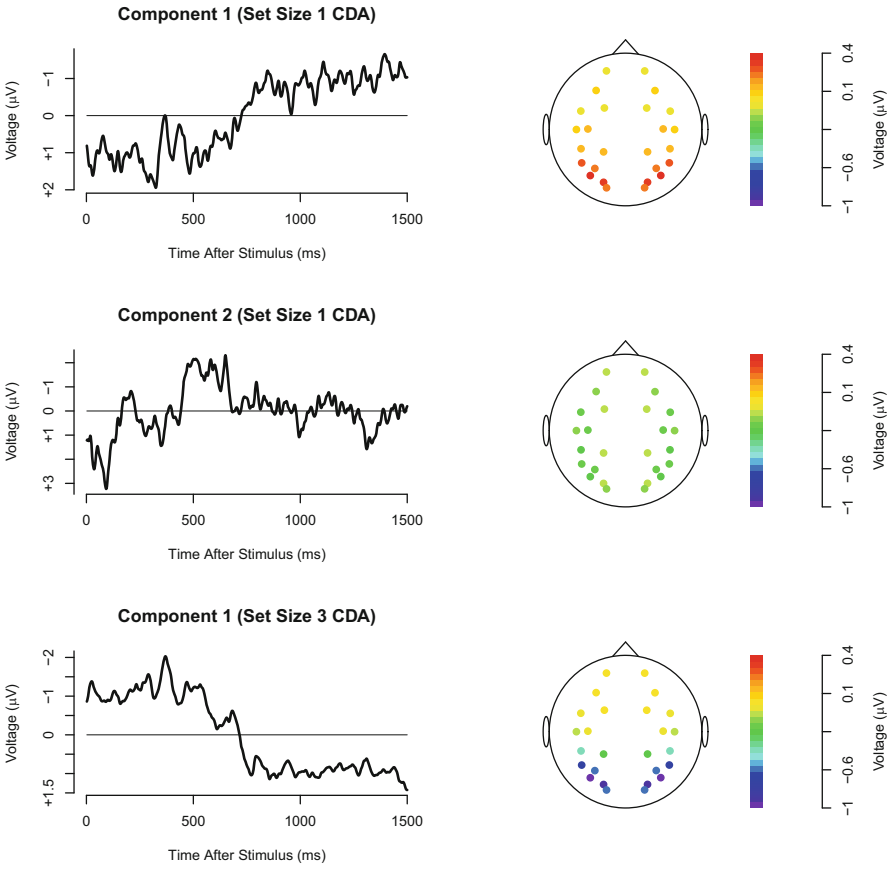

Fig. 6.8 First 2 ICs for set size 1 CDA condition, first IC for set size 3 condition (temporal and spatial domain)

contralateral and ipsilateral). For the set size 3/CDA condition, there is a single memory component only.

This analysis concludes the section on ICA. Another ICA application using fMRI data is presented in Sect. 14.4.

References

Arend, A. M., & Zimmer, H. D. (2011). What does ipsilateral delay activity reflect? Inferences from slow potentials in a lateralized visual working memory task. *Journal of Cognitive Neuroscience, 23*, 4048–4056.

Carroll, J. D., & Arabie, P. (1980). Multidimensional scaling. *Annual Review of Psychology, 31*, 607–649.

Carroll, J. D., & Chang, J. J. (1970). Analysis of individual differences in multidimensional scaling via an N-way generalization of Eckart-Young decomposition. *Psychometrika, 35*, 283–319.

Eckart, C., & Young, G. (1936). The approximation of one matrix by another of lower rank. *Psychometrika, 1*, 211–218.

Fox, J., & Weisberg, S. (2011). *An R companion to applied regression* (2nd ed.). Thousand Oaks: Sage.

Giordani, P., Kiers, H., & Del Ferraro, M. (2014). Three-way component analysis using the R package **ThreeWay**. *Journal of Statistical Software, 57*(1), 1–23. https://www.jstatsoft.org/index.php/jss/article/view/v057i07

Harshman, R. A. (1970). *Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis* (Technical report 16, UCLA Working Papers in Phonetics).

Helwig, N. E. (2015a). **eegkit**: Toolkit for electroencephalography data. R package version 1.0-2. https://CRAN.R-project.org/package=eegkit

Helwig, N. E. (2015b). **ica**: Independent component analysis. R package version 1.0-1. https://CRAN.R-project.org/package=ica

Helwig, N. E. (2017). **multiway**: Component models for multi-way data. R package version 1.0-3. https://CRAN.R-project.org/package=multiway

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology, 24*, 417–441.

Hyvärinen, A., Karhunen, J., & Oja, E. (2001). *Independent component analysis*. New York: Wiley.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning with applications in R*. New York: Springer.

Joliffe, I. T. (2002). *Principal component analysis* (2nd ed.). New York: Springer.

Karatzoglou, A., Smola, A., Hornik, K., & Zeileis, A. (2004). **kernlab**: An S4 package for kernel methods in R. *Journal of Statistical Software, 11*(9), 1–20. http://www.jstatsoft.org/v11/i09/

Kiers, H. A. L. (2000). Towards a standardized notation and terminology in multiway analysis. *Journal of Chemometrics, 14*, 105–122.

Kroonenberg, P. M. (2008). *Applied multiway data analysis*. Hoboken: Wiley.

Leibovici, D. (2010). Spatio-temporal multiway data decomposition using principal tensor analysis on *k*-modes: The R package **PTAk**. *Journal of Statistical Software, 34*(1), 1–34. https://www.jstatsoft.org/index.php/jss/article/view/v034i10

Marchini, J. L., Heaton, C., & Ripley, B. D. (2013). **fastICA**: FastICA algorithms to perform ICA and projection pursuit. R package version 1.2-0. https://CRAN.R-project.org/package=fastICA

Mevik, B. H., Wehrens, R., & Liland, K. H. (2016). **pls**: Partial least squares and principal component regression. R package version 2.6-0. https://CRAN.R-project.org/package=pls

Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine, 2*, 559–572.

Revelle, W. (2017). **psych**: Procedures for psychological, psychometric, and personality research. R package version 1.7.8. http://CRAN.R-project.org/package=psych

Sharma, S. (1996). *Applied multivariate techniques*. New York: Wiley.

Sidanius, J., Levin, S., van Laar, C., & Sears, D. O. (2010). *The diversity challenge: Social identity and intergroup relations on the college campus*. New York: The Russell Sage Foundation.

Sigg, C. D., & Buhmann, J. M. (2008). Expectation-maximization for sparse and non-negative PCA. In *Proceedings of the 25th International Conference on Machine Learning*.

Stacklies, W., Redestig, H., Scholz, M., Walther, D., & Selbig, J. (2007). **pcaMethods**: A bioconductor package providing PCA methods for incomplete data. *Bioinformatics, 23*, 1164–1167.

Stone, J. V. (2004). *Independent component analysis: A tutorial introduction*. Cambridge: The MIT Press.

Timmerman, M. E. (2001). *Component analysis of multisubject multivariate longitudinal data*. PhD thesis, University of Groningen, Groningen.

Treiblmaier, H. (2006). *Datenqualität und individualisierte Kommunikation [Data Quality and Individualized Communication]*. Wiesbaden: DUV Gabler Edition Wissenschaft.

Treiblmaier, H., Bentler, P. M., & Mair, P. (2011). Formative constructs implemented via common factors. *Structural Equation Modeling: A Multidisciplinary Journal, 18*, 1–17.

Tucker, L. R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika, 31*, 279–311.

Venables, W. N., & Ripley, B. D. (2002). *Modern applied statistics with S* (4th ed.). New York: Springer.

Vogel, E. K., & Machizawa, M. G. (2004). Neural activity predicts individual differences in visual working memory capacity. *Nature, 428*, 748–751.

Willerman, L., Schultz, R., Rutledge, J. N., & Bigler, E. (1991). In vivo brain size and intelligence. *Intelligence, 15*, 223–228.

Zou, H., & Hastie, T. (2012). **elasticnet**: Elastic-net for sparse estimation and sparse PCA. R package version 1.1. https://CRAN.R-project.org/package=elasticnet

Zou, H., Hastie, T., & Tibshirani, R. (2006). Sparse principal component analysis. *Journal of Computational and Graphical Statistics, 15*, 262–286.

Chapter 7
Correspondence Analysis

7.1 Simple Correspondence Analysis

7.1.1 Profiles, Masses, Inertia

The starting point for simple correspondence analysis (CA) is a two-dimensional contingency table $\mathbf{F}$ with $i = 1, \ldots, I$ rows and $j = i, \ldots, J$ columns. Let o_{ij} denote the observed cell frequencies, $o_{i\cdot}$ the row margins, and $o_{\cdot j}$ the column margins. N stands for the total number of observations in the table.

To illustrate various types of information contained in such contingency tables, let us create a simple toy example. Three music aficionados named Horst, Helga, and Klaus support their four favorite bands by going to as many concerts as they can. The corresponding frequency table with bands in the rows and fans in the columns is of dimension 4×3, and the cells contain the number of concert attendances ($N = 130$).

```
superfan <- as.table(matrix(c(9, 12, 8, 1, 13, 1, 6, 20, 15, 4,
  23, 18), ncol = 3))
attr(superfan, "dimnames") <- list(Band = c("Slayer",
 "Iron Maiden", "Metallica", "Judas Priest"), Fan = c("Horst",
 "Helga", "Klaus"))
superfan
##               Fan
## Band           Horst Helga Klaus
##   Slayer           9    13    15
##   Iron Maiden     12     1     4
##   Metallica        8     6    23
##   Judas Priest     1    20    18
```


P. Mair, *Modern Psychometrics with R*, Use R!,
https://doi.org/10.1007/978-3-319-93177-7_7

For such data structures, one of the first things students learn in undergraduate classes is to apply Pearson's χ^2-test in order to test for independence of the two variables involved. The well-known test statistic is

$$X^2 = \sum_{i=1}^{I} \sum_{j=1}^{J} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}. \tag{7.1}$$

The expected frequencies (under H_0 of independence) are $e_{ij} = o_{i\cdot} o_{\cdot j}/N$. This test on independence is important within a CA context because what CA eventually does is to show in a detailed way how the data deviate from independence. In our data example, we get

```
fit_chisq <- chisq.test(superfan)
fit_chisq
##
##  Pearson's Chi-squared test
##
## data:  superfan
## X-squared = 39.523, df = 6, p-value = 5.653e-07
```

We get a X^2 value of 39.523 and reject the null hypothesis of independence. The X^2 value will also become important later on since one of the main outputs in CA, called *inertia*, decomposes this value.

Based on the observed and expected frequencies, we can compute *standardized residuals* (also called *Pearson residuals*) as follows:

$$s_{ij} = \frac{o_{ij} - e_{ij}}{\sqrt{e_{ij}}}. \tag{7.2}$$

Let us organize these residuals in an $I \times J$ matrix $\mathbf{S}$, since they will also play an important role later on. We can simply extract them from the χ^2-test output object:

```
S <- fit_chisq$residuals
round(S, 3)
##                Fan
## Band             Horst  Helga  Klaus
##   Slayer         0.158  0.479 -0.503
##   Iron Maiden    4.078 -1.850 -1.373
##   Metallica     -0.184 -1.596  1.433
##   Judas Priest -2.667  2.309  0.000
```

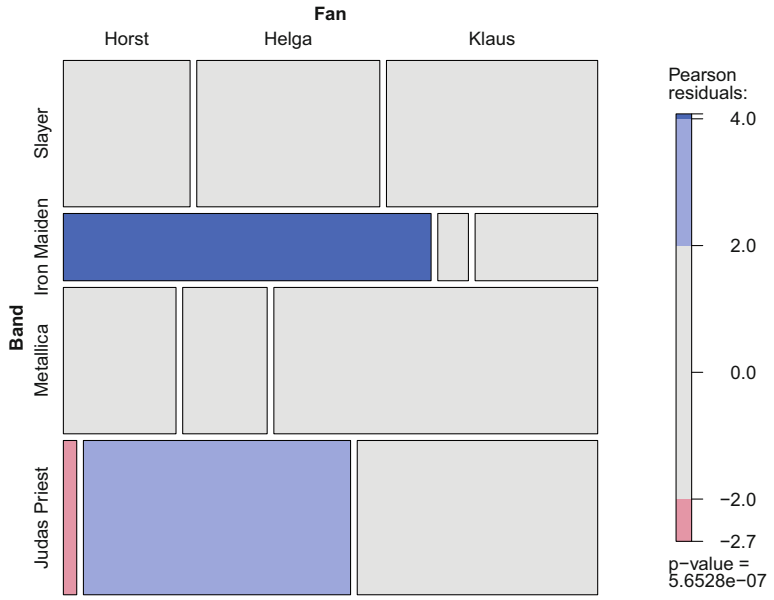

Fig. 7.1 Mosaic plot for superfan data with residual shading showing which cells are responsible for independence violation

We can display the residual information (as deviation from independence) using a shaded *mosaic plot* (see Fig. 7.1), as implemented in the **vcd** package (Zeileis et al., 2007). We see, for instance, that the cell related to Horst and Iron Maiden shows the largest deviation from independence.

```
library("vcd")
mosaic(superfan, shade = TRUE)
```

This concludes the χ^2-test part of our analysis. In the remainder of this section, we derive various other measures relevant to CA. To do so, we need a slight modification of the input table $\mathbf{F}$ in terms of the relative frequency table $\mathbf{P} = \mathbf{F}/N$, with elements p_{ij}.

```
P <- prop.table(superfan)
round(P, 3)          ## table with relative frequencies
##                Fan
## Band           Horst Helga Klaus
##   Slayer       0.069 0.100 0.115
```

(continued)

```
##   Iron Maiden  0.092 0.008 0.031
##   Metallica    0.062 0.046 0.177
##   Judas Priest 0.008 0.154 0.138
```

In CA slang the margins of this table are called *row masses* and *column masses*. Let us use r_i to denote a single-row mass element in vector $\mathbf{r}$ and c_j for a column mass element in vector $\mathbf{c}$.

```
r_mass <- margin.table(P, 1)
round(r_mass, 3)             ## row masses
## Band
##       Slayer  Iron Maiden    Metallica Judas Priest
##        0.285        0.131        0.285        0.300
c_mass <- margin.table(P, 2)
round(c_mass, 3)             ## column masses
## Fan
## Horst Helga Klaus
## 0.231 0.308 0.462
```

In a next step, we compute the conditional relative frequencies for the rows and the columns. CA calls it *row profiles* and *column profiles*.

```
r_profile <- prop.table(P, 1)
round(r_profile, 3) ## conditional relative frequencies rows
##               Fan
## Band           Horst Helga Klaus
##   Slayer       0.243 0.351 0.405
##   Iron Maiden  0.706 0.059 0.235
##   Metallica    0.216 0.162 0.622
##   Judas Priest 0.026 0.513 0.462
```

Let a_{ij} denote the j-th element of the i-th row profile $\mathbf{a}_i$. Let $\mathbf{A}$ be the corresponding matrix of row profiles.

Analogously, we can compute the column profiles where b_{ij} denotes the i-th element of the j-th column profile $\mathbf{b}_j$, again collected in a matrix $\mathbf{B}$.

```
c_profile <- prop.table(P, 2)
round(c_profile, 3) ## conditional relative frequencies columns
##               Fan
```

(continued)

```
## Band           Horst Helga Klaus
##   Slayer       0.300 0.325 0.250
##   Iron Maiden  0.400 0.025 0.067
##   Metallica    0.267 0.150 0.383
##   Judas Priest 0.033 0.500 0.300
```

Other measures we can compute are the *average row profile* (also called *row centroid*) and the *average column profile* (also called *column centroid*). The average row profile is $\mathbf{c} = \mathbf{A}'\mathbf{r}$, that is, the row profiles weighted with the row masses. We intentionally use $\mathbf{c}$ to emphasize that the average row profile corresponds to the column masses. Similarly, the average column profile is $\mathbf{r} = \mathbf{Bc}$, which correspond to the row masses. Let us compute these average row/column profiles and check whether they actually correspond to the respective masses.

```
ar_profile <- t(r_profile) %*% r_mass ## average row profile
round(as.vector(ar_profile), 3)
## [1] 0.231 0.308 0.462
round(as.vector(c_mass), 3)           ## column masses
## [1] 0.231 0.308 0.462
ac_profile <- c_profile %*% c_mass    ## average column profile
round(as.vector(ac_profile), 3)
## [1] 0.285 0.131 0.285 0.300
round(as.vector(r_mass), 3)           ## row masses
## [1] 0.285 0.131 0.285 0.300
```

They match. Let us proceed with the some plotting options involving row profiles and average row profiles. Since we have three fans only, their row profiles can be plotted in a 3D space (see Fig. 7.2) using the **plot3D** package (Soetaert, 2016). Each fan gets its own axis. Due to the nature of how the row profiles were computed, the band points lie on a plane spanned by the column (i.e., fan) vertices. To be more precise, they lie on a triangle in a 3D space. Such a structure is known as *simplex*.

As an alternative we can produce a *ternary plot*. It is a 2D plot where the axes are determined by the simplex. Note that a ternary plot is a generally applicable concept to situations where we have three (exclusive) variables which sum to a constant. Such data are called *compositional data* (see van den Boogaart and Tolosana-Delgado, 2013).

The ternary plot in Fig. 7.3, created using the **ggtern** package (Hamilton, 2015), gives us the full row profile information. The red dot represents the band centroid (i.e., average row profile) with the corresponding projections on the three fan axes. We can do the same projections for the single bands which result in the band profiles. We see that Slayer's profile is very close to the centroid. From the projections we can say that Iron Maiden scores high on Horst's axis, but low on Helga's and Klaus' axes (it is their least favorite band).

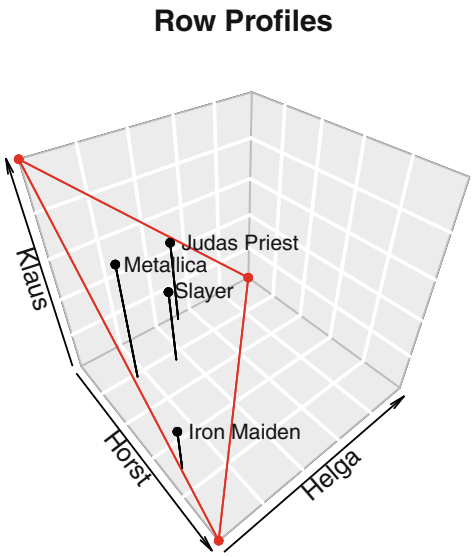

Fig. 7.2 3D scatterplot of row profiles. The band points lie on the fan simplex

We are now interested in quantifying the distances between two bands or between a band and the average row profile, as shown in the ternary plot. We could think of using a simple *Euclidean distance*. In contingency tables this is not such a good idea since, in the case of the row scores, those columns with high relative frequencies (i.e., those fans who attended many concerts) would dominate the distance. In order to avoid this distortion, we can use the χ^2-distance. It weights each term in the Euclidean distance by the expected profile element, that is, the element in the average row profile.

The following two equations compute these distances. The first one is the distance between a band and the average band profile:

$$d_{ic} = \sqrt{\sum_{j=1}^{J} \frac{(a_{ij} - c_j)^2}{c_j}}. \tag{7.3}$$

The second one computes the distance between two bands i and i'. In this case Eq. (7.3) changes to

$$d_{ii'} = \sqrt{\sum_{j=1}^{J} \frac{(a_{ij} - a_{i'j})^2}{c_j}}. \tag{7.4}$$

As an example, let us compute the distance between Slayer and Iron Maiden, as well as the distance between Slayer and Judas Priest. In addition, we also compute the distance between Slayer and Iron Maiden from the average row profile (centroid).

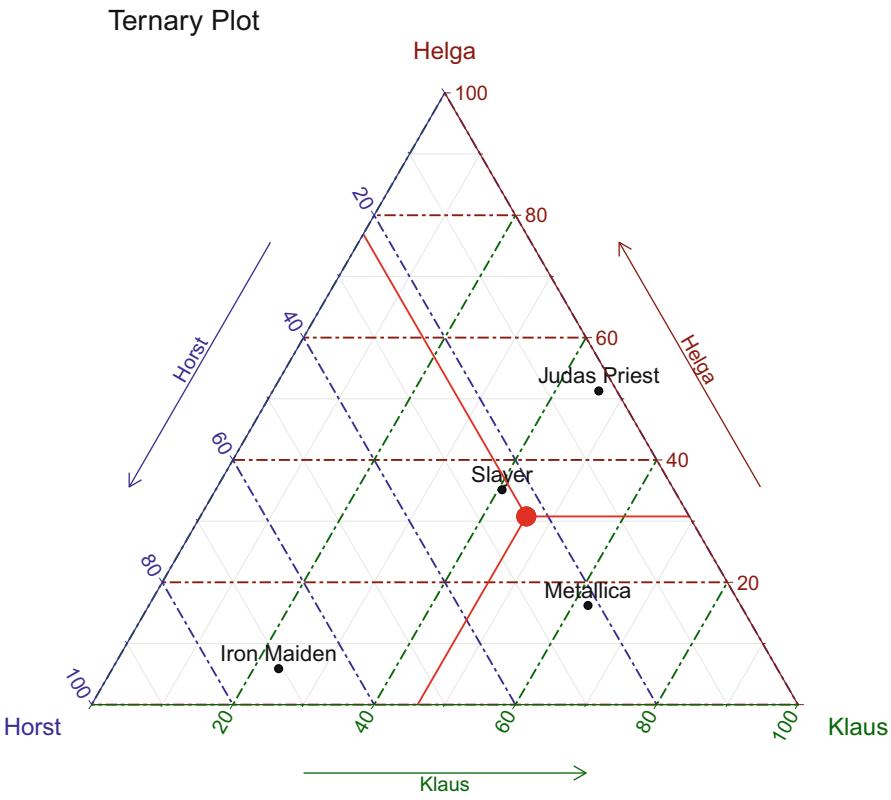

Fig. 7.3 Ternary plot of row (band) profiles. The red dot shows the average row (band) profile with the corresponding projections

```
sqrt(sum((r_profile["Slayer",] - r_profile["Iron Maiden",])^2/
           ar_profile))
## [1] 1.126186
sqrt(sum((r_profile["Slayer",] - r_profile["Judas Priest",])^2/
           ar_profile))
## [1] 0.5447462
sqrt(sum((r_profile["Slayer",] - ar_profile)^2/ar_profile))
## [1] 0.1170305
sqrt(sum((r_profile["Iron Maiden",]- ar_profile)^2/ar_profile))
## [1] 1.135944
```

We see that Slayer is closer to Judas Priest than to Iron Maiden. Also, we see that Slayer is closer to the band centroid than Iron Maiden. We can perform the same computations on the columns involving column profiles and the average column profile.

Using the row masses and the row profile distances to the centroid, we can construct a measure called *inertia*. The *row inertias* are

$$\phi_i = r_i d_{ic}^2. \tag{7.5}$$

The *total inertia* is simply the sum of the row inertias:

$$\phi = \sum_{i=1}^{I} \phi_i. \tag{7.6}$$

It can be shown that $\phi = X^2/N$. This implies that Eq. (7.5) decomposes the χ^2-statistic given in Eq. (7.1) in an additive way. In our example the total inertia is 39.523/130 = 0.304. It describes total amount of dispersion of the profiles around the centroid. If we were to use the distances based on the column profiles and the column masses, Eq. (7.5) changes to $\phi_j = c_j d_{jc}^2$ and Eq. (7.6) to $\phi = \sum_{j=1}^{J} \phi_j$. At this point we have all the ingredients together to compute and interpret a CA.

7.1.2 Simple CA Computation and Interpretation

The good news is that in order to fit a CA, we do not have to perform all the tedious computations from above. As so often in multivariate methods, a *singular value decomposition* (SVD; see Sect. 6.1.1) does all the magic. In the case of simple CA, we perform an SVD on the standardized residual matrix **S** given in Eq. (7.2).

$$\mathbf{S} = \mathbf{UDV}'. \tag{7.7}$$

Let p be the number of dimensions. The $p \times p$ diagonal matrix **D** contains the singular values σ_s ($s = 1, \ldots, p$) with $\lambda_s = \sigma_s^2$ as the corresponding eigenvalue (within a CA context often called *principal inertia*). On the basis of the eigenvalues, we can judge the importance of each dimension. This is due to the fact that an eigenvalue λ_s reflects the dispersion of profiles around the centroid on dimension s which is the contribution to the total inertia associated with that dimension. In order to obtain a measure for "explained variance"[1] for each dimension, we can compute $\lambda_s / \sum_{l=1}^{p} \lambda_l$. As in PCA, we can produce a scree plot with λ_s on the y-axis, and based on the elbow criterion, we pick a reasonable number of dimensions. Note that in simple CA, the maximum number of dimensions we can fit is $\min(I - 1, J - 1)$. In our simple example, we can maximally reduce to two dimensions since we have only three fans.

[1] We have to be careful with the term "variance" within a CA context since we are dealing with categorical data.

The matrix $\mathbf{U}$ in Eq. (7.7), containing the left singular vectors, is of dimension $I \times p$. These left singular vectors contain the *row category scores*, subject to further standardization. The matrix $\mathbf{V}$ contains the right singular vectors (i.e., *column category scores*) and is of dimension $J \times p$. There are various ways of standardizing the scores for subsequent plotting:

- *Row principal coordinates*: $\mathbf{U}^{(p)} = \mathbf{R}^{-\frac{1}{2}}\mathbf{U}\mathbf{D}$.
- *Row standard coordinates*: $\mathbf{U}^{(s)} = \mathbf{R}^{-\frac{1}{2}}\mathbf{U}$.
- *Column principal coordinates*: $\mathbf{V}^{(p)} = \mathbf{C}^{-\frac{1}{2}}\mathbf{V}\mathbf{D}$.
- *Column standard coordinates*: $\mathbf{V}^{(s)} = \mathbf{C}^{-\frac{1}{2}}\mathbf{V}$.

Here, the row masses $\mathbf{r}$ are stored in an $I \times I$ diagonal matrix $\mathbf{R}$. Analogously, $\mathbf{C}$ is the diagonal matrix containing the column masses $\mathbf{c}$ on the diagonal.

Mapping the principal coordinates $\mathbf{U}^{(p)}$ and $\mathbf{V}^{(p)}$ into the same plot gives us a *symmetric map*. This is the most popular CA plot. Another version is to use one set of standard coordinates and one set of principal coordinates. This leads to an *asymmetric map*. Asymmetric maps are biplots and will be discussed in more detail in Sect. 10.5.

Let us compute and interpret a simple CA on the superfan data. There are several packages available in R for fitting a CA. In this section we use the **anacor** package (De Leeuw and Mair, 2009) which provides some useful features such as the computation of confidence ellipsoids. In the section on multiple CA, we switch to the **ca** package (Nenadic and Greenacre, 2007) since **anacor** cannot handle multiple CA.

```
library("anacor")
ca_fans <- anacor(superfan, ellipse = TRUE)
ca_fans
##
## CA fit:
##
## Total chi-square value: 39.523
## Sum of eigenvalues (total inertia): 0.304
## Eigenvalues (principal inertias):
## 0.256 0.049
##
## Benzecri RMSE rows:  1.098991e-17
## Benzecri RMSE columns:  1.603251e-18
##
## Chi-square decomposition:
##              Chisq Proportion Cumulative Proportion
## Dimension 1 33.216       0.84                 0.84
## Dimension 2  6.307       0.16                 1.00
##
## z-test on singular values:
##             Singular Value Standard Error p-value
```

(continued)

```
## Dimension 1          0.505          0.073   0.000
## Dimension 2          0.220          0.082   0.004
```

We see that the first dimension explains 84% of the dispersion in the row/column profiles; the second dimension takes care of the remaining 16%. The package also prints out an explicit test on whether a dimension is needed (if significant, the dimension is needed). However, one should not base the dimensionality assessment solely on these p-values.

We can produce the symmetric map as follows:

```
plot(ca_fans, main = "Symmetric CA Map")
```

It plots the principal row and column coordinates in the same space (see Fig. 7.4), including confidence ellipsoids. Interpreting the dimensions in a CA map is not as crucial as in factor analysis or PCA. In practice, we sometimes have this interpretation opportunity when ordinal variables are involved.

Let us focus on the row coordinates interpretation (i.e., the bands). The row coordinates reflect the patterns we have seen in the ternary plot in Fig. 7.3. Slayer is close to the origin since their row profile is close to the average row profile. The most interesting points in CA are always the ones at the extremities of the space (i.e., they span the space). In this application Iron Maiden is the most extreme band since their row profile deviates heavily from the average one. In fact, it is the band with the least concert attendance, as we see from the original table. The distances among the bands in the plot can be interpreted as χ^2-distances (cf. Eq. (7.4)). In terms of column coordinates (i.e., fans), we see that Horst lies pretty far away from the other two fans. His column profile deviates clearly from the average column profile. Again, the distances among the fans in the plot are χ^2-distances.

Now, what about Horst and Iron Maiden? Or, in other words, what about interpreting row-column distances? By going back to the table, we see that Horst's favorite band is Iron Maiden. However, we have to be careful with drawing such conclusions from a symmetric map since CA defines no distances between row and column categories. In fact, the row and column profiles are computed separately. Therefore, we need to keep in mind that we are basically merging two separate plots (principal row coordinates and principal column coordinates) into one single space. As Greenacre (2007, p. 72) puts it: "Distances can be interpreted whenever the points concerned are located in the same space." He calls it the "golden rule for interpreting CA maps". As we will see in Sect. 10.5, asymmetric maps use a joint space, and row-column interpretations can be achieved.

From a practical point of view, the following aspect is important. Gabriel (2002) elaborates on the distortion induced by using symmetric maps as if they were asymmetric maps. Greenacre (2007, pp. 267–268) discusses in great detail the

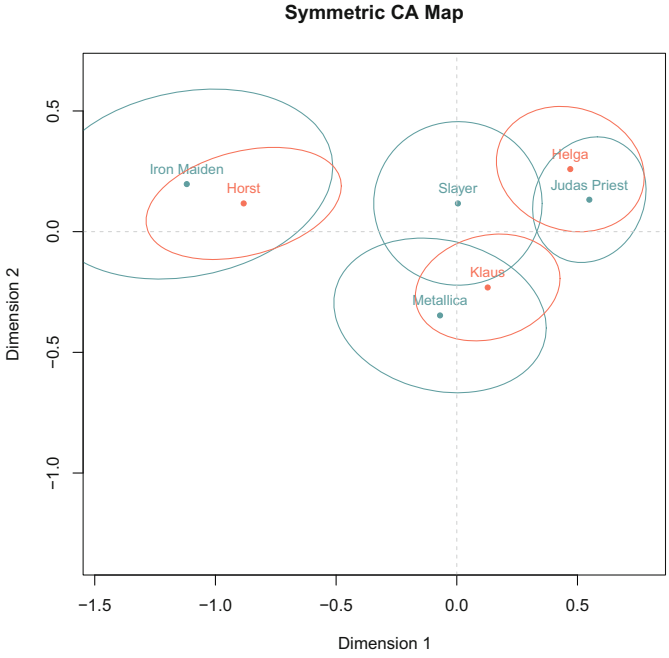

Fig. 7.4 Symmetric CA map for superfan data including confidence ellipsoids

practical implication of this distortion and concludes that in cases where the square roots of the principal inertias (i.e., the singular values as shown in the output above) are not heavily different, row-column relationships in a symmetric map can be interpreted with reasonable assurance. Thus, even though technically not entirely correct, under some circumstances, row-column distances can be interpreted in symmetric maps. Other options for interpretation are given in Bartholomew et al. (2008, Chapter 4).

Another useful tool within this context is to go back to the residual matrix and the mosaic plot when interpreting a CA solution. Below (see Sect. 7.3), we present an approach called *configural frequency analysis* which performs explicit tests on these residuals. This is helpful for interpretation because what CA maps show us are basically deviations from independence. If we feed a perfectly independent table into a CA function, all the row and column points would lie at the origin.[2] Therefore, the interesting points in a CA map, subject to interpretation, are far away from the origin. The boring ones are those close to the origin.

[2]The reader can try this out by saying `anacor(fit_chisq$expected)`.

7.1.3 Example: Harvard Psychology Faculty

When applying for a job at a large department, it is always good to know what research topics are covered by the faculty members. Also, it does not hurt to know which faculty members are working in similar areas. We could now spend several days reading through the professors' impressive CV's, or we can run a CA on their research statements (text data), typically available on the department's website. This gives us a first quick glimpse on what is going on at the institution.

In the case of the Harvard Psychology faculty, every faculty member provides a short research statement on the department website.[3] With some R skills, it is fairly easy to set up a scraping job in R and load these statements into the workspace. After performing some basic text preprocessing steps (e.g., eliminating punctuation, removing stopwords such as conjunctions, articles, etc.) using the **tm** package (Feinerer et al., 2008), we can organize the texts as *document-term matrix* (DTM). A DTM is simply a contingency table with, in our example, faculty members in the rows and research topics (keywords) in the columns.[4] A cell entry denotes how often a particular researcher mentioned a particular keyword in his/her research statement.

A two-dimensional, simple CA on the DTM based on the Harvard Psychology faculty statements can be fitted as follows. We use once more principal coordinates scaling for both rows and columns.

```
library("MPsychoR")
data("HarvardPsych")
dim(HarvardPsych)  ## researchers in rows, words in columns
## [1] 29 43
fit_HP <- anacor(HarvardPsych)
```

With two dimensions, we explain 24.93% of the inertia. This is not a lot, but it is sufficient to get a rough overview of what is going on in terms of research at the department.

The symmetric map of the 2D solution, which includes the row scores (last name researcher) and the column scores (research topic), is given in Fig. 7.5. Since the points are cluttered to the right of the origin, the plot in the bottom panel zooms into the respective area. As explained above, researcher-to-researcher distances are directly interpretable, as well as the word-to-word distances. Note that the singular values for the first two dimensions are 0.77 and 0.651. They do not differ heavily from each other. Thus, based on what we have said in the previous section, we can interpret the researcher-to-word associations with reasonable assurance.

[3] See http://psychology.fas.harvard.edu/faculty

[4] Only the most frequent keywords are considered for this analysis.

We will further investigate the output of this example in Sect. 7.3 where we perform a configural frequency analysis on this table.

7.2 Multiple Correspondence Analysis

Multiple CA extends simple CA in terms of higher-dimensional input tables. In this section we give a brief outline of the French approach to multiple CA which solves the problem analytically. We will discuss multiple CA in more detail in Chap. 8 where the problem is solved numerically (Dutch approach) within the larger framework of Gifi methods. This variant offers a large amount of flexibility.

In the French world, there are two basic approaches to handle higher-dimensional frequency tables: one is through an eigenvalue decomposition on the standardized residuals of the Burt matrix, and the other one is based on an SVD on the indicator matrix. In this section we illustrate the Burt matrix approach for which the starting point is the data matrix $\mathbf{H}$ with n observations and m categorical variables. In order to convert the raw data into an indicator matrix $\mathbf{G}$, we expand each observation into a vector which is of the same length as the number of categories of the corresponding variable. In this vector, the category answered by the participant gets a value of 1; the remaining ones become 0. For instance, if on a 5-point Likert scale a person scores, the indicator vector becomes 0, 0, 1, 0, 0. We do this for each variable. Based on the resulting indicator matrix, the Burt matrix $\mathbf{B}$ can be constructed as follows:

$$\mathbf{B} = \mathbf{G}'\mathbf{G}. \tag{7.8}$$

$\mathbf{B}$ is symmetric and contains all possible two-way cross classifications of the m variables involved. $\mathbf{B}$ can be treated as a standard contingency table, and we can compute the residual matrix $\mathbf{S}$ following the steps presented in the section above. Finally, we decompose $\mathbf{S}$ by either an eigenvalue decomposition or an SVD. Since $\mathbf{S}$ is symmetric, both approaches lead to the same results.

To illustrate, we use a subset of the youth depression dataset from Vaughn-Coaxum et al. (2016) introduced in Sect. 2.1. The children depression inventory (CDI) items considered here are item 15 ("I am bad all the time") and item 21 ("I never have fun at school"), each of them scored on three categories. In addition we use the variable `race` (four response categories). Thus, in total, we have ten categories, and $\mathbf{B}$ will be of dimension 10×10.

```
library("MPsychoR")
data("YouthDep")
cdisub <- YouthDep[, c("CDI15r", "CDI21r", "race")]
```

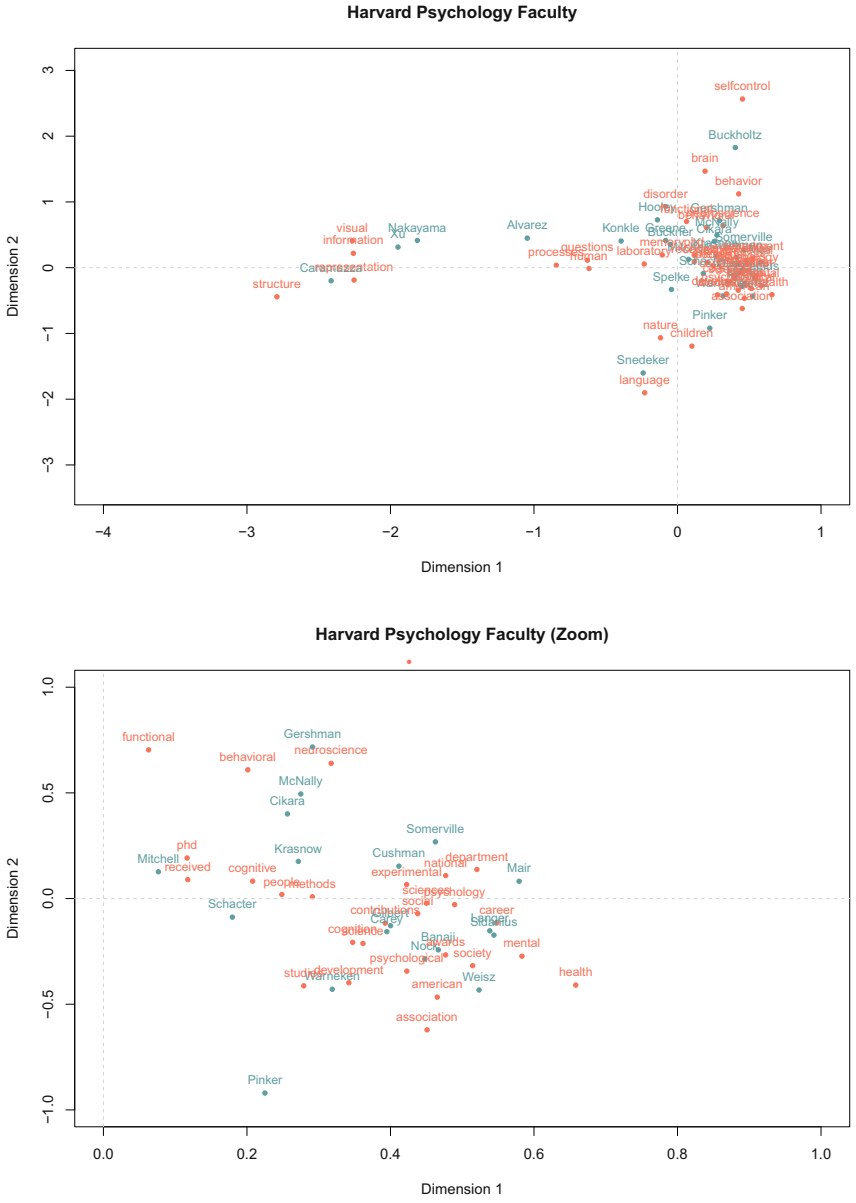

Fig. 7.5 Symmetric CA map for faculty members at the Harvard Department of Psychology. It gives the last name of the faculty member (rows) and the research topics (columns). Top panel: full CA map. Bottom panel: zoomed in around origin

The **anacor** package provides the following utility function to create a Burt matrix (output not shown here):

```
B <- burtTable(cdisub)
dim(B)
## [1] 10 10
```

Let us fit a multiple CA based on this Burt matrix in order to obtain the category scores. To fit a multiple CA, we can use the **ca** package (Nenadic and Greenacre, 2007) for corresponding computations. The Burt matrix is created internally. We also produce the symmetric map right away, for which the same issues regarding category distance interpretations among different variables apply as in simple CA.

```
library("ca")
fit_mca <- mjca(cdisub, lambda = "Burt")
plot(fit_mca, xlim = c(-0.5, 0.5))
```

By considering two dimensions, we explain 42.64% of the inertia. The resulting symmetric map is given in Fig. 7.6. Pairs of category scores across the two CDI items inhabit similar areas of the CA map. Regarding the race variable, we see a separation along dimension 1 between Latinos and Black kids on the one hand and Asian and White kids on the other hand. The order of the response categories of the CDI items along the first dimension is retained. Asian/White kids tend to score 0 on these items, whereas Latino/Black kids tend to score 1. As in simple CA, we have to be careful with making such inter-variable interpretations based on this plot.

There are additional multiple CA variants such as *joint correspondence analysis* which fit into this analytical framework. Details can be found in Greenacre (2007) and Husson et al. (2017).

7.3 Configural Frequency Analysis

In this section we present a method called *configural frequency analysis* (KFA; Lienert, 1968; von Eye, 2002; von Eye et al., 2010)[5] which, within a CA context, can be used to give us a more detailed insight into row-column associations.

[5]Note that in order to avoid confusion with confirmatory factor analysis (CFA), we use the acronym KFA introduced by its inventor Gustav Lienert who called his baby by the catchy name of *Konfigurationsfrequenzanalyse*.

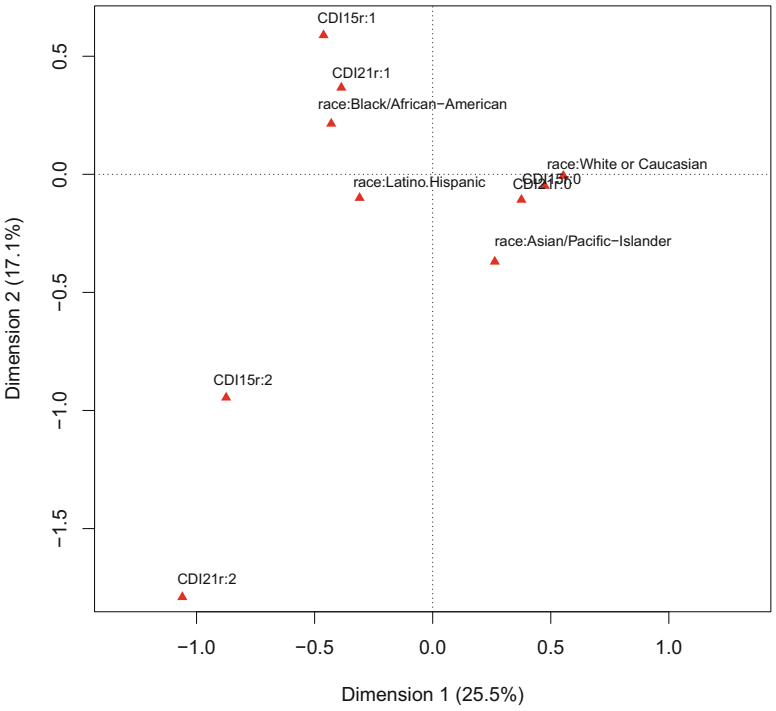

Fig. 7.6 Symmetric multiple CA map on youth depression dataset

KFA is a general method for contingency tables and aims to find so-called *types* and *antitypes*. The idea is very simple: we fit a base model on the frequency table such as an independence model. This base model gives us expected frequencies; that is, how would the frequencies look like if the variables were independent? Subsequently, we compare the observed frequencies with the expected frequencies by means of residual tests. Cells that have significantly more observations than expected are identified as types. Conversely, cells that have significantly less observations than expected are declared as antitypes. In R, KFA can be performed using the **cfa** package (Mair and Funke, 2017).

7.3.1 Two-Dimensional Tables

We reconsider the example presented in Sect. 7.1.3. Figure 7.5 showed the symmetric CA map involving the faculty members at the Harvard Psychology Department and the research topics. We can use KFA to further investigate the associations between a researcher and a research topic. For two-dimensional tables, the base model is easy to specify. It is simply the table of expected frequencies as used in a

χ^2-test. Let $\mathbf{F}$ be an $I \times J$ frequency table with N observations in total. Let $\mathbf{r}$ denote the vector with the row margins and $\mathbf{c}$ the column margins. The expected frequency table under independence is simply $(\mathbf{rc}')/N$.

The **cfa** package requires the data to be restructured. As first argument, the `cfa` function takes all possible configurations (i.e., combinations of row and column categories). As second argument, it needs the corresponding vector of counts. We also tell the function to use a binomial test for residual testing (which is followed by Bonferroni multiple testing correction; other options are available as well).

```
library("cfa")
data("HarvardPsych")
configs <- expand.grid(dimnames(HarvardPsych))
counts <- as.vector(HarvardPsych)
fit.cfa <- cfa(configs, counts,binom.test = TRUE, sorton = "n")
types <- fit.cfa$table[fit.cfa$table$sig.bin == TRUE, 1:3]
head(types, 10)
##                          label  n  expected
## 1  Schacter   memory          15 1.7685590
## 2  Xu         visual          10 1.1280932
## 3  Banaji     social          10 2.3158661
## 4  Buckholtz  brain           10 1.2576419
## 5  Snedeker   language         9 0.7423581
## 6  Caramazza  structure        8 0.4366812
## 7  Buckholtz  selfcontrol      8 0.4716157
## 8  Buckner    memory           8 1.0611354
## 9  Xu         human            8 1.4439592
## 10 Snedeker   children         8 1.0480349
```

The last line prints the top 10 types out of a total 95 types, as identified by the binomial test. We see that the strongest type is Dan Schacter who's researching (biological aspects) on human memory. The observed word count clearly exceeds the count as expected under independence. Note that there are no antitypes in this application since the frequency table is very sparse (i.e., lots of 0 entries). The results from KFA can be used in combination with the symmetric CA map in Fig. 7.5 for researcher-topic interpretation.

In KFA there is a variety of exact and asymptotic tests that can be applied to detect types and antitypes. As shown in simulation studies, tests that perform well under many conditions and under any sampling scheme are Pearson's χ^2-test, z-test, and the binomial test (see von Eye, 2002, for details).

7.3.2 Higher-Dimensional Tables

Having higher-dimensional tables, dependency structures become more complex, and simple χ^2-tests do not work anymore. In this case we need to use *log-linear models* (see, e.g., Wickens, 1989, for an excellent introduction). The model formulation is very similar to a higher-order ANOVA model, with the exception that on the left-hand side of the equation, we have the logarithm of the cell frequencies. For a cross classification of variables A ($i = 1, \ldots, I$ categories), B ($j = 1, \ldots, J$), and C ($k = 1, \ldots, K$), the saturated model specification looks as follows:

$$\log e_{ijk} = \lambda + \lambda_i^A + \lambda_j^B + \lambda_k^C + \lambda_{ij}^{AB} + \lambda_{ik}^{AC} + \lambda_{jk}^{BC} + \lambda_{ijk}^{ABC}, \tag{7.9}$$

where e_{ijk} is the expected frequency in cell (ijk). We have parameters for the intercept, the main effects, the two-way interactions, and the three-way interaction. Note that in log-linear models, the parameter interpretation and the concept of interactions are slightly more complicated as in ANOVA (Mair and von Eye, 2007).

Within the context of KFA, we need to define a log-linear base model. The model in Eq. (7.9) is *saturated*, that is, it fits the data perfectly. Consequently, using a saturated model in KFA would not detect any types or antitypes. The most popular base model strategy in KFA is to use an independence model which only includes the main effects:

$$\log e_{ijk} = \lambda + \lambda_i^A + \lambda_j^B + \lambda_k^C. \tag{7.10}$$

This is a so-called *first-order KFA*. However, we could also define a *second-order KFA* model by including all two-way interactions; this depends on the hypotheses we want to explore. In any case, we get a table with expected frequencies, and based on the resulting residuals, we can explore possible types and antitypes.

We use the youth depression dataset from Sect. 7.2 where we performed a multiple CA. The corresponding table is now subject to a KFA fit, and we detect types and antitypes by first using a χ^2-test and second a z-test which is less conservative:

```
countdf <- as.data.frame(table(cdisub))
fit.cdi <- cfa(countdf[,1:3], countdf[,4])
fit.cdi$table[fit.cdi$table$sig.chisq == TRUE, 1:3]  #chi2-test
##                         label   n   expected
## 1 0 0 White or Caucasian     516 371.813459
## 2 2 2 Latino.Hispanic         15   3.667286
```

The χ^2-test finds two types (i.e., o_{ijk} are significantly larger than e_{ijk}) which suggest that White kids tend to score 0 on both items and Latino kids tend to score 2 on both items.

Using the z-test, in addition to the two types from above, we obtain other types and antitypes:

```
fit.cdi$table[fit.cdi$table$sig.z == TRUE, 1:3] ## z-test
##                          label   n   expected
## 1 0 0 White or Caucasian     516 371.813459
## 2 2 2 Latino.Hispanic         15   3.667286
## 3 1 1 Black/African-American  67  37.388755
## 4 2 0 White or Caucasian      29  66.612357
## 5 1 0 White or Caucasian      87 141.582918
## 6 1 1 Latino.Hispanic         93  59.888182
## 7 2 1 Latino.Hispanic         50  28.176372
## 8 0 1 Latino.Hispanic        107 157.273437
```

For instance, Black kids who score 1 on both items are identified as types. This test also finds some antitypes where o_{ijk} is significantly smaller than e_{ijk}. For instance, Latino kids rarely score 0 on the first item and 1 on the second item (last line). The type/antitype patterns obtained using KFA are in line with what we have seen in the symmetric multiple CA map in Fig. 7.6 and give some additional insight into category associations across variables.

The examples considered here show some very basic applications of KFA within the context of CA. KFA is of course more generally applicable whenever someone has categorical data and is interested in finding cells that deviate from an underlying base model. Further details including more advanced models such as longitudinal KFA, moderator/mediator KFA, and functional (stepwise) KFA can be found in the textbooks by von Eye (2002) and von Eye et al. (2010).

References

Bartholomew, D. J., Steele, F., Moustaki, I., & Galbraith, J. I. (2008). *Analysis of multivariate social science data* (2nd ed.). Boca Raton: CRC Press.

De Leeuw, J., & Mair, P. (2009). Simple and canonical correspondence analysis using the R package **anacor**. *Journal of Statistical Software, 31*(5), 1–18. http://www.jstatsoft.org/v31/i05/

Feinerer, I., Hornik, K., & Meyer, D. (2008). Text mining infrastructure in R. *Journal of Statistical Software, 25*, 1–54. http://www.jstatsoft.org/v25/i05/

Gabriel, R. (2002). Goodness of fit of biplots and correspondence analysis. *Biometrika, 89*, 423–436.

Greenacre, M. (2007). *Correspondence analysis in practice* (2nd ed.). Boca Raton: Chapman & Hall/CRC.

Hamilton, N. (2015). **ggtern**: An extension to **ggplot2** for the creation of ternary diagrams. R package version 1.0.6.1. http://CRAN.R-project.org/package=ggtern

Husson, F., Lê, S., & Pageès, J. (2017). *Exploratory multivariate analysis by example using R* (2nd ed.). Boca Raton: CRC Press.

Lienert, G. (1968). Die Konfigurationsfrequenzanalyse als Klassifikationsmethode in the klinischen Psychologie [Configural frequency analysis as classification method in clinical psy-

chology]. Paper presented at the 26. Kongress der Deutschen Gesellschaft für Psychologie in Tübingen.

Mair, P., & Funke, S. (2017). **cfa**: Configural frequency analysis (CFA). R package version 0.10-0. http://CRAN.R-project.org/package=cfa

Mair, P., & von Eye, A. (2007) Application scenarios for nonstandard log-linear models. *Psychological Methods, 12*, 139–156.

Nenadic, O., & Greenacre, M. (2007). Correspondence analysis in R, with two- and three-dimensional graphics: The **ca** package. *Journal of Statistical Software, 20*(3), 1–13. http://www.jstatsoft.org/v20/i03/

Soetaert, K. (2016). **plot3D**: Plotting multi-dimensional data. R package version 1.1. https://CRAN.R-project.org/package=plot3D

van den Boogaart, K. G., & Tolosana-Delgado, R. (2013). *Analyzing compositional data with R*. New York: Springer.

Vaughn-Coaxum, R., Mair, P., & Weisz, J. R. (2016). Racial/ethnic differences in youth depression indicators: An item response theory analysis of symptoms reported by White, Black, Asian, and Latino youths. *Clinical Psychological Science, 4*, 239–253.

von Eye, A. (2002). *Configural frequency analysis: Methods, models, and applications*. Mahwah: Lawrence Erlbaum.

von Eye, A., Mair, P., & Mun, E. Y. (2010). *Advances in configural frequency analysis*. New York: Guilford Press.

Wickens, T. D. (1989). *Multiway contingency tables analysis for the social sciences*. Hillsdale: Erlbaum.

Zeileis, A., Meyer, D., & Hornik, K. (2007). Residual-based shadings for visualizing (conditional) independence. *Journal of Computational and Graphical Statistics, 16*, 507–525.

Chapter 8
Gifi Methods

8.1 Setting the Stage

8.1.1 Optimal Scaling: Measurement Levels as Functions

Are Likert items ordinal or metric? Questions like this are typically answered ad hoc by considering the classical measurement (or scale) level terminology introduced by Stevens (1946), nominal, ordinal, and metric, with the latter sometimes subdivided into interval and ratio scales. With respect to Likert items, researchers often pick "metric" as an answer, since it is certainly convenient to have metric variables for data analysis. In this section we take a radically different view on scale levels, compared to Stevens' taxonomy. For the moment we answer the question raised above with "it depends."

This seemingly coward answer brings us right into the world of *optimal scaling* (OS). In OS, scale levels are not ad hoc characteristics of the variables. Rather, they depend on the interaction between the data and the model to be used for the analysis. OS is defined as a data analysis technique which assigns numerical values to observation categories in a way which maximizes the relation between the observations and the data analysis model while respecting the measurement character of the data (Young, 1981).

OS involves the following three stages: First, we treat each variable as categorical. Second, we specify some basic *measurement characteristics* for each variable (i.e., measurement levels as functions). Third, we let an algorithm do the rest (i.e., compute a score or quantification for each category).

In the OS world, measurement levels are defined as functions, sometimes also called *analysis levels* (see Linting et al., 2007). Let x be a variable in a dataset. Let us assume it is a 5-point Likert item with response categories ranging from 1 to 5. Measurement levels as functions imply that we compute a new variable of the form $x^* = f(x)$ with new *category scores*.

P. Mair, *Modern Psychometrics with R*, Use R!,
https://doi.org/10.1007/978-3-319-93177-7_8

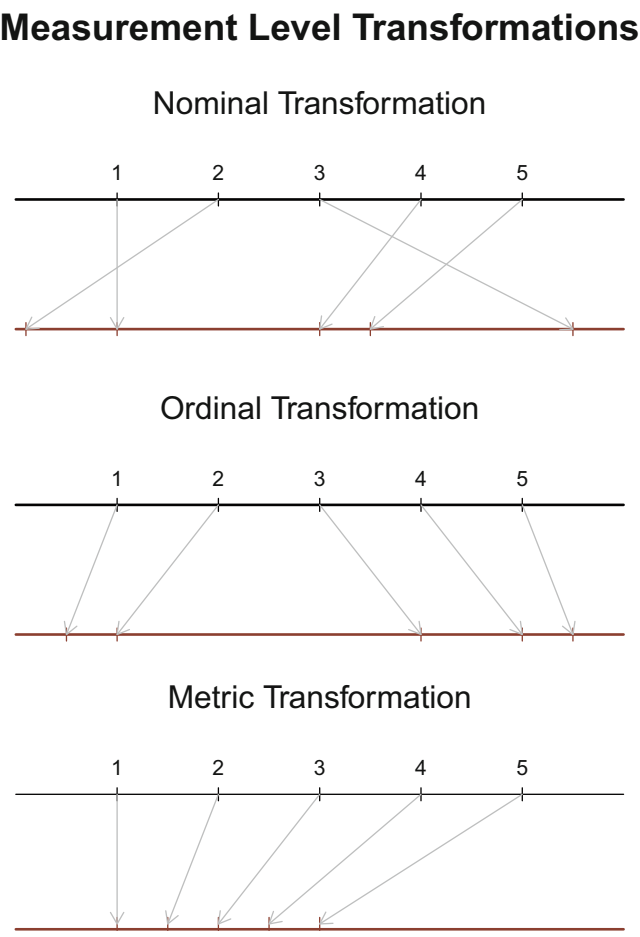

Fig. 8.1 Optimal scaling transformations for nominal, ordinal, and metric (i.e., linear) transformation functions. For each measurement level, the top line shows the original 1–5 scale and the bottom line the category quantifications (category scores), according to the measurement characteristic imposed on the data

For the moment, let us consider three simple transformation functions: nominal, ordinal, and metric. Figure 8.1 shows the transformations that are induced by these functions. The nominal transformation is the most general one, with $f(x)$ being completely unrestricted. The ordinal analysis level puts an ordinal restriction on the transformation function: The order of the transformed scores needs to obey the order of the original category scores. The metric transformation uses a linear transformation of the form $x^* = a + bx$: In addition to the ordinal restriction, the distances between the transformed scores have to be equal.

The last transformation opens up many doors. Think of it as a linear regression. How can we extend a linear regression? We can define a polynomial regression

function (e.g., quadratic or cubic) or, even more flexible, (monotone) spline regression. Simply speaking, *splines* are piecewise polynomials which connect at pre-specified x-values (called *knots*). They are often used in nonlinear regression techniques such as *generalized additive models* (GAM; see Wood 2017 for details). As we will see below, in Gifi it is useful to restrict this function to be monotone increasing (i.e., a *monotone spline*).

Figure 8.2 shows these transformations based on an input variable with 13 categories. The original scores are given on the x-axis and the OS-based quantifications on the y-axis. In the nominal case, anything can happen to the quantifications. In the ordinal case, we fit a monotone step function. In the linear case, we fit a line which keeps the category distances constant. In the monotone spline transformation, we fit a flexible, monotonically increasing function.

The bottom line is that in the OS world, we are not limited to Stevens' holy scale level trinity. Measurement levels are not necessarily an ad hoc thing, determined prior to an analysis. As Jacoby (1999) points out, measurement levels are determined by the researcher within the context of a model. Or, as Linting et al. (2007) put it, trying out measurement levels is part of the data-analytic task, in addition to common sense choices. For instance, we would not do anything else than a nominal transformation on hair color. For 5-point Likert items, we may consider trying out an ordinal and a linear transformation. If the transformation functions look similar, we have evidence to consider a Likert item as metric within the context of a particular method. Apart from rating scales, linear transformations can be applied on clearly metric variables. Spline transformations are attractive for variables with many categories.

At this point, all that is left to show is how to determine these optimally scaled category quantifications based on an input analysis level. We need a model to achieve that. The most flexible OS modeling framework is called *Gifi*, formally introduced in the next section. However, this OS idea is not limited to Gifi. In fact, other techniques such as multidimensional scaling (MDS; see Chap. 9) use OS as well when transforming the input dissimilarities. Further elaborations on OS and corresponding transformation functions can be found in Jacoby (1991).

8.1.2 Gifi Theory

Gifi is a pen name for a group of Dutch researchers, named after Francis Galton's manservant Albert Gifi. They developed an über-framework for nonlinear multivariate data analysis, starting in the 1970s, which culminated in the magnum opus, the Gifi (1990) book. They called the framework "nonlinear" since the transformation functions (cf. Fig. 8.2) are generally nonlinear (apart from the linear transformation, of course). The methodology behind Gifi is pretty challenging, with a healthy amount of matrix algebra under the hood. Details can be found in Gifi (1990), De Leeuw and Mair (2009a), Michailidis and De Leeuw (1998), and Mori et al. (2016).

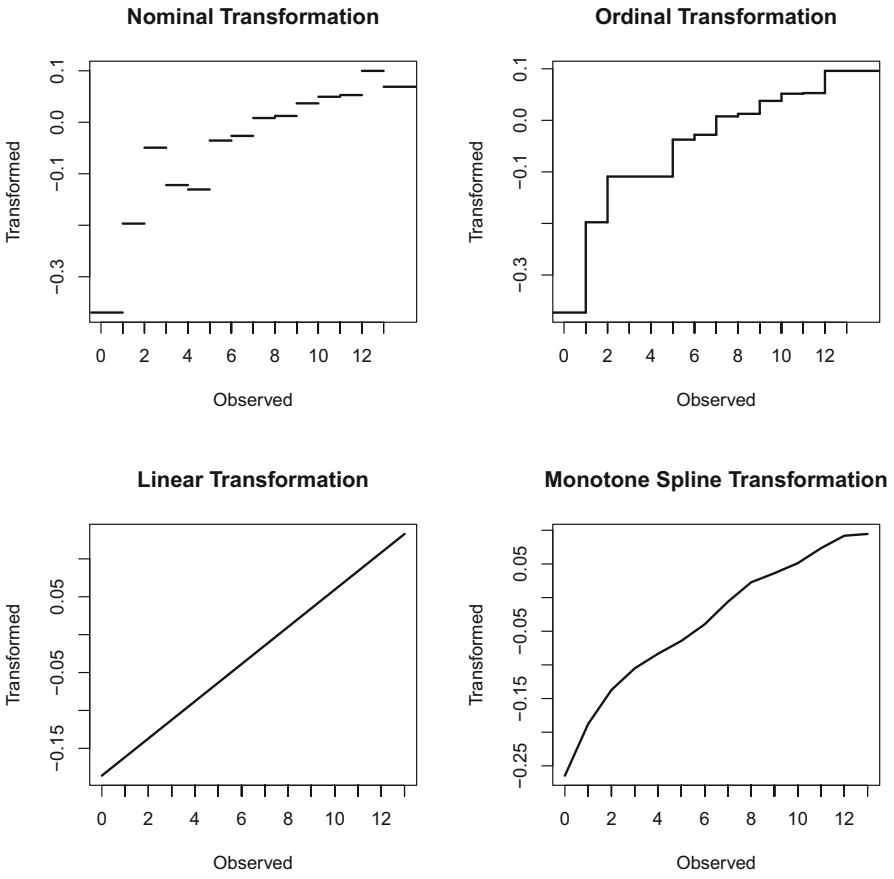

Fig. 8.2 Optimal scaling transformations for nominal, ordinal, linear, and monotone spline transformation functions. The original scores of the 13-point variable are on the x-axes and the transformed scores on the y-axes

First of all, Gifi models involve dimension reduction, just as principal component analysis (PCA) and correspondence analysis (CA). Let p be the number of dimensions which needs to be fixed a priori. Let $\mathbf{H}$ be an $n \times m$ data matrix. Correspondingly, $\mathbf{h}_j$ represents the column vector for variable j with k_j as the number of categories. For each variable we define an *indicator matrix* $\mathbf{G}_j$ of dimension $n \times k_j$, consisting of 0s and 1s in the case of categorical data. These indicator matrices can be then collected in an indicator supermatrix $\mathbf{G} = (\mathbf{G}_1 | \ldots | \mathbf{G}_m)$.

Each variable is associated with a matrix $\mathbf{Y}_j$ of dimension $k_j \times p$ containing the *category quantifications*. The final component we need is the matrix $\mathbf{X}$. It contains the so-called *object scores* and is of dimension $n \times p$. At the end of the day, each person gets a score in the p-dimensional space, and each category of variable j gets an optimally scaled category quantification in p dimensions. Since we scale both the

objects and the variables, these methods are sometimes referred to as *dual scaling* methods. Putting all these ingredients together, Gifi establishes the following loss function:

$$\sigma(\mathbf{X}, \mathbf{Y}_1, \ldots, \mathbf{Y}_m) = \sum_{j=1}^{m} \text{tr}(\mathbf{X} - \mathbf{G}_j\mathbf{Y}_j)'(\mathbf{X} - \mathbf{G}_j\mathbf{Y}_j). \tag{8.1}$$

The right-hand side of the equation represents a sum-of-squares (SS) expression that needs to be minimized. This can be achieved by an *alternating least squares* (ALS) algorithm. This loss formulation is very general, and, depending on the particular Gifi model we fit, it simplifies correspondingly, or, for some versions, it can even get more complicated (see De Leeuw and Mair, 2009a).

Classical multivariate models that fit into this framework are PCA (*Princals*), multiple CA (*Homals*), multiple regression (*Morals*), conjoint analysis (*Addals*), canonical correlation analysis (*Canals*), multiblock canonical correlation (*Overals*), and discriminant analysis (*Criminals*). The difference between the Gifi versions and the classical formulations is that Gifi integrates OS by means of nonlinear transformation functions.

An R implementation of the Gifi framework is provided by the **Gifi** package (Mair and De Leeuw, 2017). This package replaces the **homals** package (De Leeuw and Mair, 2009a). Compared to **homals**, the new package incarnation implements some of the Gifi variants mentioned above in a much more user-friendly way and is, at the same time, also more flexible since it uses a slightly different loss formulation (see De Leeuw et al., 2017).

From a practitioner's point of view, what is the value of Gifi? Gifi allows us to incorporate input variables of potentially mixed scale levels via the specification of analysis levels. Some of them, using Stevens' classical distinction once more, can be metric, some of them nominal, and some of them ordinal. Since it performs OS, we can treat variables by means of different transformation functions. We do not have to speculate anymore whether Likert items are metric or not; Gifi transforms them in an optimal way. We can try out different analysis levels on our data and see whether the results differ substantially. In the following sections, we focus on the two most popular Gifi methods: Princals and Homals.

8.2 Princals

Princals is Gifi's version of PCA, sometimes also referred to as *categorical PCA* or *nonlinear PCA*. For each of the input variables, we need to specify an analysis level. In the **Gifi** package, the transformation function of each variable is defined through splines for which the knots are specified via the `knotsGifi` function, prior to the model fit. In the following subsections, we will elaborate on this specification step-by-step.

8.2.1 Mimicking PCA with Princals

The first Princals version we fit is an emulation of a standard PCA (see Chap. 6). Even though this Princals version is not that intriguing from a practical point of view, it makes it easy to describe some of the main Gifi features. To illustrate, we use two subscales from the ASTI dataset (Koller et al., 2017) already used in the chapter on item response theory (IRT; see Sect. 4.3.2). The ASTI is a self-report scale measuring the complex target construct of wisdom, involving five subdimensions. Here we focus on the seven items belonging to the "self-transcendence" (ST) subscale and the six items belonging to "presence in the here-and-now and growth" (PG). Some items are scored on three categories; some others have four categories. First, let us fit a standard PCA on these data. This implies that we treat the items on a metric scale level.

```
library("MPsychoR")
data("ASTI")
st <- ASTI[ ,c(2,4,7,13,16,24,25)]
pg <- ASTI[ ,c(11,14,15,17,18,23)]
stpg <- data.frame(st = st, pg = pg)
pcafit <- prcomp(stpg, scale = TRUE)
```

Second, we show how to mimic this fit with Princals. Taking variables at a metric analysis level means that we perform a linear transformation. This function merely standardizes the variables; it does not change any of the interval scale properties (i.e., distances between two adjacent categories are constant within each variable).

In the code chunk below, the first line sets up the knot structure of the underlying spline. As mentioned above, a spline is a piecewise polynomial, specified through the polynomial degree and the knots. The simplest form of a spline is a line. There is one knot at the minimum and one knot at the maximum value of the variable under consideration. There are no knots in between the two (i.e., 0 *interior knots*). Thus, setting up the interior knots of a linear transformation is a trivial task: It is a list of zeros. Such knots can be produced using the Gifi utility function `knotsGifi` and setting `type="E"`. These knots act as input to the subsequent `Princals` call. The polynomial degree of a line is 1 and is specified through the `degrees` argument. We fix the number of dimensions to $p = 2$ (default).

```
library("Gifi")
knotslin <- knotsGifi(stpg, type = "E")
prlin <- princals(stpg, knots = knotslin, degrees = 1)
prlin
## Call:
```

(continued)

```
## princals(data = stpg, knots = knotslin, degrees = 1)
##
## Loss value: 0.847
## Number of iterations: 25
##
## Eigenvalues: 2.608 1.371
```

The output shows the loss value (cf. Eq. (8.1)), the number of iterations the ALS algorithm needed to converge, and the two eigenvalues, one for each dimension.

By means of the eigenvalues, let us check whether PCA and linear Princals lead to the same results:

```
(pcafit$sdev^2)[1:2]
## [1] 2.607748 1.371302
prlin$evals[1:2]
## [1] 2.607748 1.371302
```

They are the same. We can also look at the loadings (only the loadings of the first three items are shown here):

```
head(round(pcafit$rotation[,1:2], 3), 3)
##             PC1   PC2
## st.ASTI2 -0.406 0.437
## st.ASTI4 -0.309 0.124
## st.ASTI7 -0.405 0.479
head(round(prlin$loadings, 3), 3)
##              D1     D2
## st.ASTI2 -0.656 -0.514
## st.ASTI4 -0.498 -0.152
## st.ASTI7 -0.654 -0.561
```

The differences in the loadings are due to different standardizations. The `prcomp` function standardizes the loadings to $SS = 1$, whereas in `princals` they are standardized in a way that the SS correspond to the eigenvalues:

```
apply(pcafit$rotation[,1:2], 2, function(pc) sum(pc^2))
## PC1 PC2
##   1   1
apply(prlin$loadings, 2, function(pc) sum(pc^2))
```

(continued)

```
##       D1       D2
## 2.607748 1.371212
```

The next line standardizes the Princals loadings to $SS = 1$ such that they (closely) match the PCA loadings (apart from the sign switch on PC2, which does not matter).

```
prloads1 <- apply(prlin$loadings, 2, function(pc){
  pc/sqrt(sum(pc^2))
})
head(round(prloads1, 3), 3)
##              D1     D2
## st.ASTI2 -0.406 -0.439
## st.ASTI4 -0.309 -0.130
## st.ASTI7 -0.405 -0.480
```

The bottom line is that by using a linear Princals, we obtain the same solution as in standard PCA.

8.2.2 Princals on Ordinal Data

Let us now do something that goes beyond standard PCA and is of high practical relevance: PCA on ordinal input data (i.e., ordinal Princals). This implies that we use a monotone step function as transformation function. This is the default transformation setup in `princals`. We do not have to worry about any spline settings here; the defaults are just fine. All we have to say is:

```
prord <- princals(stpg)
prord
## Call:
## princals(data = stpg)
##
## Loss value: 0.841
## Number of iterations: 36
##
## Eigenvalues: 2.635 1.492
```

By comparing this loss value to the one obtained from the linear Princals fit above, we see that there is not much of an improvement by fitting a ordinal version.

Note that the ordinal version is statistically more complex than the linear version since we use a more flexible transformation function (linear vs. monotone step function). At this point we have good evidence that it is fine to treat these data on a metric scale level. This does not mean that the variables per se are metric in the sense of Stevens. It just means that, within the context of this particular method, it is fine to assume metric analysis levels.

Despite negligible differences between the linear and the ordinal fit, let us proceed with the ordinal solution[1] in order to illustrate some additional concepts and outputs. Figure 8.3 can be produced as follows:

```
plot(prord, plot.type = "transplot",
     var.subset = c(1:2, 8:9), lwd = 2)
```

It shows the transformation plots of four selected items. We see that ordinality of the original scale (x-axis) is maintained in the transformations (y-axis).

Next, we inspect some of the objects from the `princals` call in more detail, especially in relation to Eq. (8.1). The matrix $\mathbf{X}$ is of dimension $n \times 2$ since we fitted a $p = 2$ dimensional solution. Each person gets a score on both dimensions. Object scores are the Princals equivalent to PC scores in PCA (only the first three persons are shown here):

```
head(round(prord$objectscores, 3), 3)
##       D1     D2
## 1 -0.954 -1.041
## 2 -0.574  0.524
## 3  0.814  1.044
```

Furthermore, each category gets a quantification in the 2D space which can be extracted using `prord$quantifications`. Note that these quantifications on the second dimension are linearly dependent on the ones of the first dimension. This is a technical feature of Princals, often referred to as *rank-1 restriction* (see De Leeuw and Mair, 2009a). That is, from a transformation perspective, only the transformations on the first dimension matter. This is something very convenient because we can use the first dimension category quantifications which replace our original data values. This "new" data matrix is called *score matrix*. It is of the same dimension as our original data matrix. Let us print the original scores and the transformed scores for the first six observations (five items):

[1] We could argue that with ordinal Princals we did the "right thing", since it is certainly safer to treat the data as ordinal.

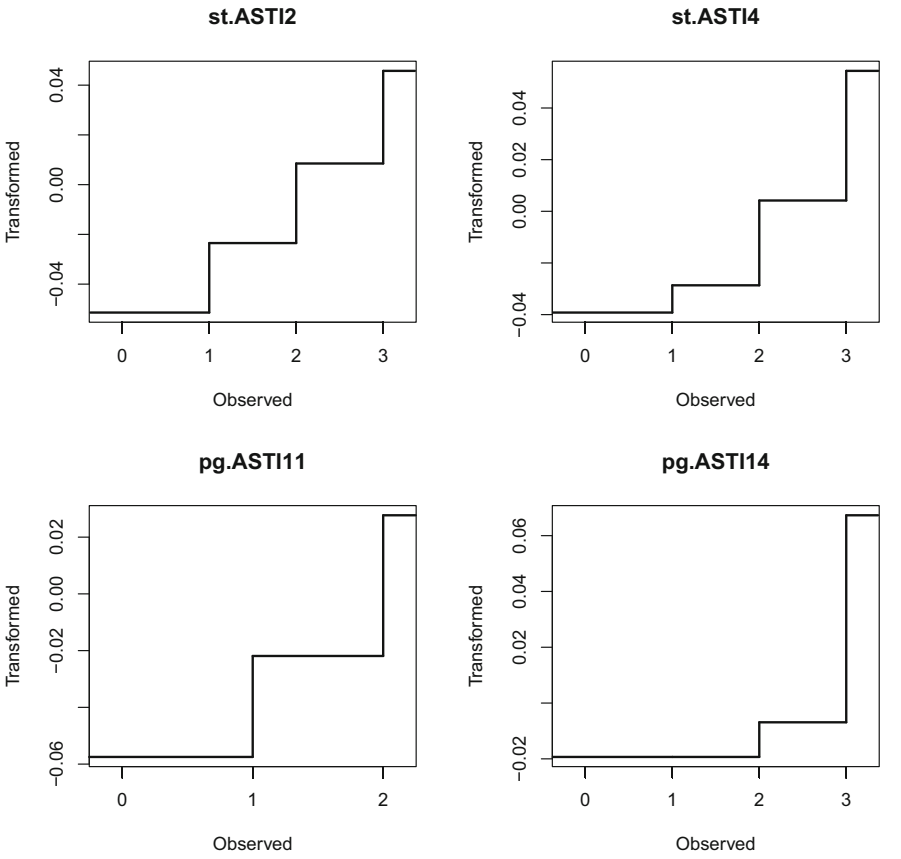

Fig. 8.3 Princals transformation plots: ordinal transformations shown for two items of the ST subscale and for two items of the PG subscale

```
head(stpg[,1:5])
##   st.ASTI2 st.ASTI4 st.ASTI7 st.ASTI13 st.ASTI16
## 1        2        3        2         2         3
## 2        2        2        2         2         2
## 3        1        1        0         2         1
## 4        2        3        2         2         3
## 5        3        3        3         2         3
## 6        1        3        3         1         2
head(round(prord$scoremat[,1:5], 3))
##   st.ASTI2 st.ASTI4 st.ASTI7 st.ASTI13 st.ASTI16
## 1   -0.005   -0.027   -0.011    -0.006    -0.027
## 2   -0.005   -0.002   -0.011    -0.006    -0.006
## 3    0.015    0.014    0.028    -0.006     0.012
```

(continued)

```
## 4   -0.005   -0.027   -0.011    -0.006    -0.027
## 5   -0.029   -0.027   -0.030    -0.006    -0.027
## 6    0.015   -0.027   -0.030     0.000    -0.006
```

We see that the original category scores have been replaced by new category quantifications. This new score matrix has several desirable features and can replace our original data matrix for subsequent analyses. More details on this topic will follow in Sect. 8.4.

Finally, we can extract the loadings via `prord$loadings`. Princals loadings can be interpreted in the same way as in PCA. Figure 8.4 shows the corresponding loadings plot. We see that, apart from items 24 and 25, the second dimension separates the ST items from the PG items.

```
plot(prord, main = "ASTI Loadings Plot")
```

Another plotting option is a *biplot* where we plot object scores and loadings in the same space. This will be illustrated within a larger biplot context in Chap. 10.

As in PCA, the number of dimensions can be explored using a *scree plot* ("elbow criterion"; plot not shown here):

```
plot(prord, plot.type = "screeplot")
```

Ordinal Princals is probably the most relevant Gifi method in practice, and we have used it already in the IRT chapter for dimensionality assessment of ordinal input variables (see Sect. 4.1.2).

8.2.3 Princals on Mixed Input Data

In this section we show another Princals flavor based on mixed scale input data. We use once more the same set of ASTI items and treat them as ordinal. In addition, for illustration purposes we create sum scores for the remaining three scales in the dataset: self-knowledge (SI), peace of mind (PM), and non-attachment (NA).

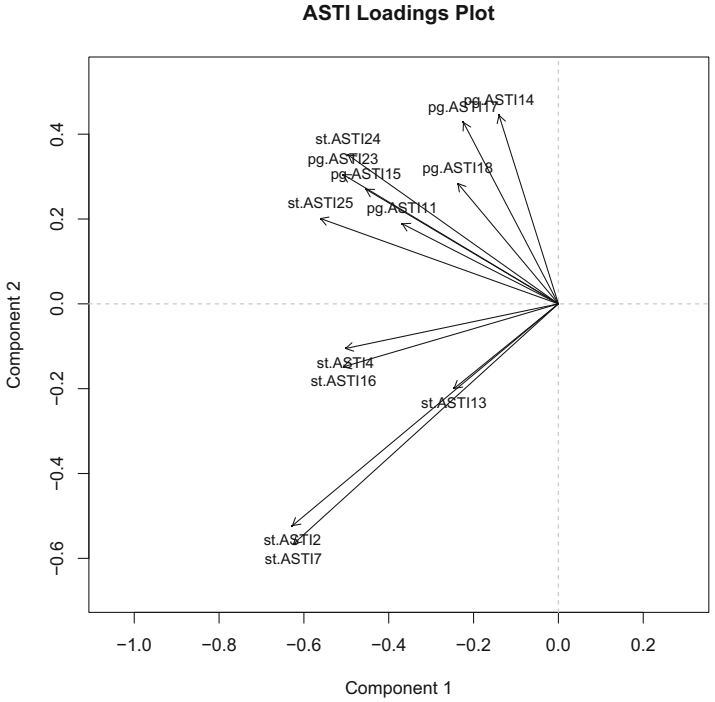

Fig. 8.4 Princals loadings plot: shows the loadings based on 2D ordinal Princals fit

```
si <- rowSums(ASTI[ ,c(10, 19, 20, 21)])
pm <- rowSums(ASTI[ ,c(1, 5, 9, 22)])
na <- rowSums(ASTI[ ,c(3, 6, 8, 12)])
asti2 <- data.frame(stpg, si, pm, na)
```

We apply linear transformations on the sum score variables. That is, we treat them as metric. We set up the knots for the two sets of transformations separately: The `type="D"` setting places knots at the data points, as needed for the ordinal transformation of the 13 items; `type="E"` implies a linear transformation of the three sum scores. Subsequently, we collect all of them in a list object.

```
knotsord <- knotsGifi(asti2[,1:13], type = "D")
knotslin <- knotsGifi(asti2[,14:16], type = "E")
knotslist <- c(knotsord, knotslin)
```

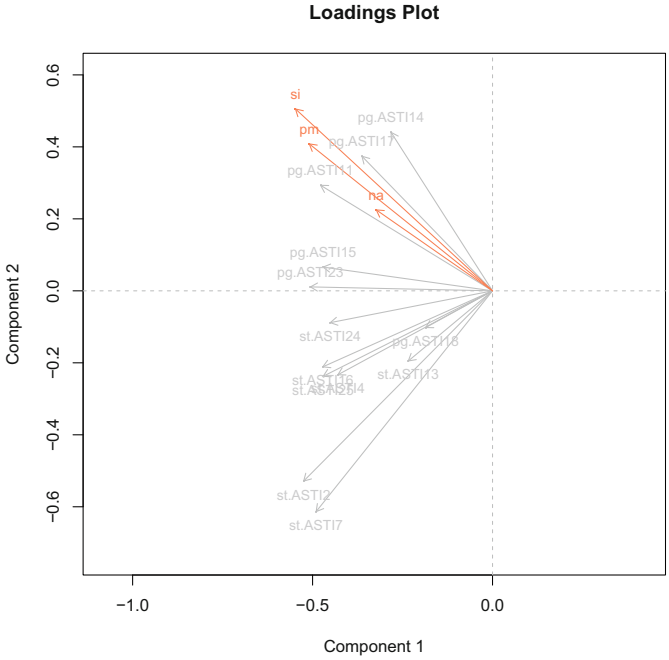

Fig. 8.5 Princals loadings plot for the first two dimensions: ordinal transformation on the items (gray) and linear transformation on the sum scores for SI, PM, and NA (red)

This time, for illustration purposes, we fit a 3D Princals. The argument setting `degrees=1` is internally used for the linear transformations only; the ordinal transformations are not affected by this specification.

```
prordlin <- princals(asti2, knots = knotslist, degrees = 1,
                     ndim = 3)
colvec <- c(rep("gray", 13), rep("coral", 3))
plot(prordlin, col.loadings = colvec, plot.dim = c(1, 2))
```

Figure 8.5 shows the loadings plot for the first two dimensions. The item loadings and the sum score loadings are colored differently. The sum scores for PM and SI are highly related to each other. The short NA loadings vector suggests that NA has smaller loadings on the first two dimensions than PM and SI. Some of the PG items (i.e., 11, 14, and 17) appear to be strongly related to the three sum score variables. However, for a closer examination, we would have to consider a 3D loadings plot.

Let us summarize what we have achieved so far. We have shown that in case of purely linear transformations, Princals leads to the same results as standard PCA. We have also demonstrated how to fit a Princals on ordinal data. This strategy is

very attractive for Likert items. Subsequently, we created some metric variables and showed how fit a Princals involving ordinal and linear transformations. In case of variables with many categories (e.g., 10 or so), a monotone spline transformation can be considered since an ordinal transformation might be too fine-grained and a linear transformation might be too strict. Such an example is shown in the next section where we also elaborate on nominal input variables, something we have not touched so far.

8.3 Homals

Homals is mathematically more general than Princals, and, in its basic form, it is designed for nominal input data. In the first part of this section, we use Homals as a tool for multiple correspondence analysis (CA) computation. Subsequently, we focus on how to incorporate mixed scale levels into Homals and how to combine it with Princals.

8.3.1 Multiple Correspondence Analysis Using Homals

In Sect. 7.2 we introduced the French approach to multiple CA which uses a singular value decomposition (SVD) on either the Burt matrix or the indicator matrix. The Dutch approach (i.e., Homals) solves the multiple CA problem numerically, which offers a great amount of flexibility.

Let us start with fitting a Homals solution using the Wilson-Patterson conservatism data. Part of this dataset was already analyzed in the IRT chapter, where we fitted a nominal response model (see Sect. 4.3.4). In total, the dataset contains 46 items, each of them with response categories 1 = "approve," 0 = "disapprove," and 2 = "don't know." Let us pick six items of this scale (gay marriage, sexual freedom, gay adoption, gender quotas, affirmative action, and legalized marijuana) plus the country variable (Hungary, the USA, and India).

```
library("MPsychoR")
library("Gifi")
data("WilPat")
WP6 <- WilPat[,c(32, 38, 41, 44, 45, 46, 47)]
```

Let us fit a 2D Homals solution which defines unrestricted, nominal transformation functions for all variables.

```
homwp <- homals(WP6)
homwp
## Call:
## homals(data = WP6)
##
## Loss value: 0.663
## Number of iterations: 22
##
## Eigenvalues: 2.901 1.818
```

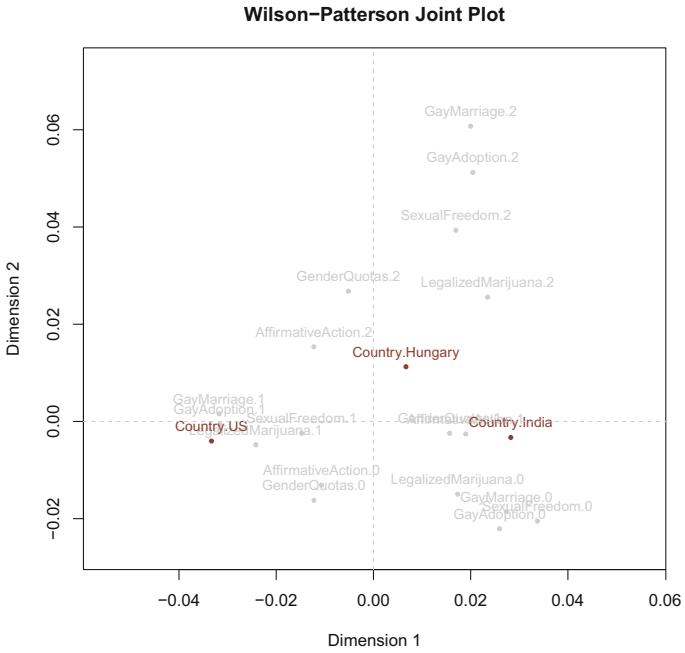

Fig. 8.6 Symmetric map (joint plot) for six selected items from the Wilson-Patterson scale and the respondents' country

In the Princals applications presented above, we were mostly interested in plotting the loadings. In Homals the most important plot is based on category quantifications, that is, a *symmetric CA map* which Gifi calls *joint plot* (see Fig. 8.6 which can be produced using `plot(homwp)`). In this plot we are interested in interpreting associations among the single item categories, associations among

countries, and how the countries are associated with the item categories.[2] Dimension 1 discriminates between 0 and 1 responses, whereas the second dimension is mostly determined by the "don't know" answers. Participants from India tend to disapprove on all items, US participants tend to approve, and Hungarian participants tend to respond "don't know."

8.3.2 Homals on Mixed Input Data

One of the main features of Gifi is the possibility to define different transformation functions for various input variables. In addition to the six items and country from above, in this section we include four more variables: First, we consider a self-reported liberal-conservative item (re-categorized into four categories), which we will take as ordinal. Second, we include a self-reported political left-right identification item on a 10-point scale. We use a spline transformation since there are many categories, but we do not quite want to do a linear transformation (i.e., not considering it entirely as metric). We will set two interior knots at the quantiles and use a polynomial degree of 2. Finally, we have gender (nominal) and age (linear). Note that even though **Gifi** can handle missing values, we only use the full responses here since there are no data from the USA pertaining to the left-right variable. This leaves us with two countries only (Hungary and India). Let us set up the knot specifications step-by-step.

```
WPmixed <- WilPat[,c(32, 38, 41, 44, 45, 46, 47:51)]
WPmixed$LibCons <- cut(WPmixed$LibCons, breaks = c(0,2,4,6,9), labels = 1:4)
WPmixed <- na.omit(WPmixed)
itknots <- knotsGifi(WPmixed[,1:6], "D")      ## item knots (data)
cknots <- knotsGifi(WPmixed[,7], "D")         ## country knots (data)
lcknots <- knotsGifi(WPmixed[,8], "D")        ## lib-cons knots (data)
lrknots <- knotsGifi(WPmixed[,9], "Q", n = 2) ## left-right (terciles)
genknots <- knotsGifi(WPmixed[,10], "D")      ## gender knots (data)
ageknots <- knotsGifi(WPmixed[,11], "E")      ## age knots (empty)
knotlist <- c(itknots, cknots, lcknots, lrknots, genknots, ageknots)
```

Now we define the vector for the ordinal restrictions, as well as the one for the spline degrees.

```
ordvec <- c(rep(FALSE, 6), FALSE, TRUE, TRUE, FALSE, TRUE)
degvec <- c(rep(-1, 7), 1, 2, -1, 1)
```

[2]Note that in Homals the same issues apply as in single and multiple CA when it comes to interpreting distances between categories of different items (see Sect. 7.1.2 for details).

The first vector specifies which variables should be ordinally restricted. For all the nominal variables, the element is set to `FALSE`. The degrees for the nominal variables are set to -1, as required by the `homals` function. For the left-right variable, we define a polynomial degree of 2. At this point we are all set to fit the model:

```
hommix <- homals(WPmixed, knots = knotlist, ordinal = ordvec,
                 degrees = degvec)
plot(hommix, "transplot", var.subset = 6:11)
```

Figure 8.7 shows the transformation plots. We see that some variables are transformed on two dimensions. The black lines reflect the transformations on the first dimension and the red lines the transformations on the second dimension. If we look at the liberal-conservative rating (ordinal) and left-right rating (spline), we see that our transformation restriction only holds for the first dimension. For the variables with two categories only (country and gender), and the ones with a linear restriction (age), we only get transformed scores on the first dimension. We will continue with this example in the next section, where we discuss in more detail the issue of transformations on both dimensions which seems awkward for non-nominal variables.

8.3.3 Combined Homals-Princals Strategies

What we have seen so far is that Princals is great for ordinal, linear, and other metric transformations. Homals is useful for nominal transformations since we are interested in the category quantifications. If we incorporate non-nominal analysis levels in Homals, the transformations can get a bit weird on the second dimension, as we have seen above. In this section we aim for a more sophisticated version of these two models in order to get the best of both worlds.

We have seen that Princals is a restricted version of Homals. For instance, if we fit a nominal Princals, the category quantifications on the second dimension are linearly related to the ones from dimension 1. Let us illustrate this property by fitting this model using the same six items as in Sect. 8.3.1 and plotting the loadings and category quantifications right away.

```
prinwp1 <- princals(WP6, ordinal = FALSE)
```

The top panel of Fig. 8.8 shows the Princals loadings and the bottom panel the Princals category quantifications. Due to the fact that the second dimension is

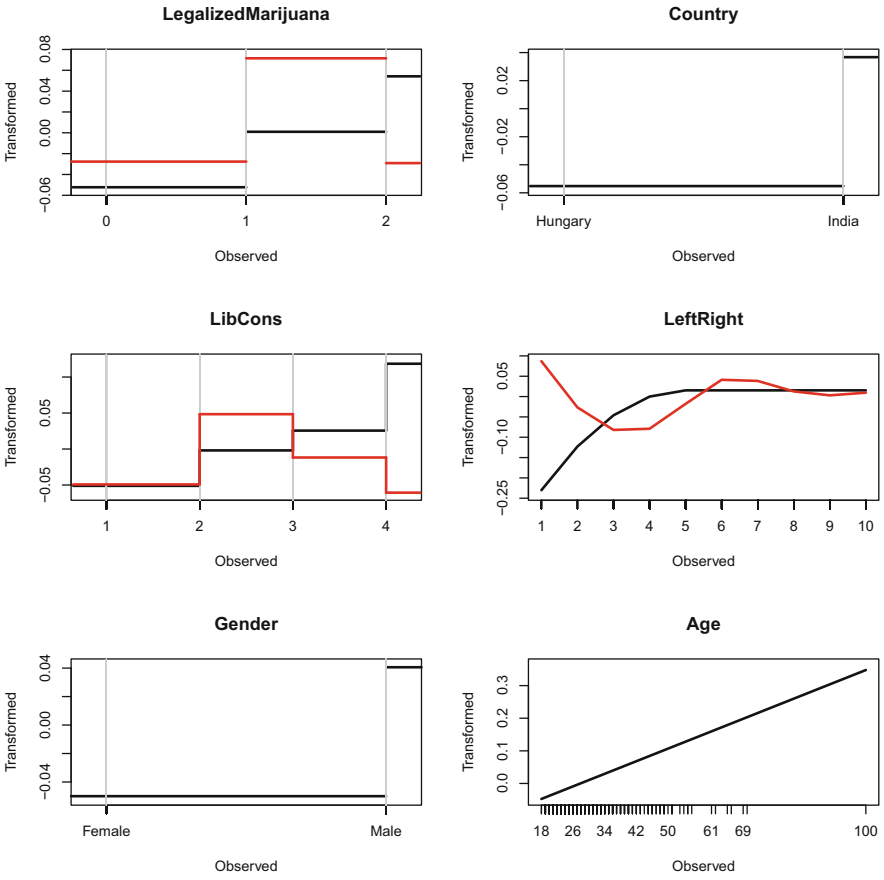

Fig. 8.7 Transformation plots for six variables of the Wilson-Patterson data. The black lines show the transformation on the first dimension and the red lines the transformations on the second dimension

linearly dependent on the first dimension, for each variable the quantifications have to be on a straight line. In Fig. 8.6 we did not have this restriction. The direction of this line is determined by the direction of the loadings vector. We should not be mistaken and interpret the loadings plot of a nominal Princals solution in a directional way (e.g., "higher" or "larger"), since we operate on a nominal level. It is easier to think of it as a straight line where the quantifications are located on (bottom panel of Fig. 8.8). Still, the loadings plot is informative for nominal transformations as well. For variables whose categories are quantified similarly (e.g., gay marriage, gay adoption, sexual freedom), the loading vectors point to the same direction. Thus, we get some insight which items are similarly scored. If we look at gender quotas and affirmative actions, the lines in the joint plot are very close to each other, whereas in the loadings plot, the arrows point into the opposite direction. This is

due to the fact that in the bottom-left quadrant of the plot, the quantification for 2 in affirmative action is close to 0 in gender quota, and gender quota 2 is close to affirmative action 0.

Using the `princals` function call, we can relax this linear restriction using the concept of *copies* (see De Leeuw et al., 2017). For each variable we produce as many copies,[3] as there are number of dimensions (in our case two).

```
prinwp2 <- princals(WP6, ordinal = FALSE, copies = 2)
prinwp2
## Call:
## princals(data = WP6, ordinal = FALSE, copies = 2)
##
## Loss value: 0.663
## Number of iterations: 22
##
## Eigenvalues: 2.901 1.818
```

The loss value and the eigenvalues are exactly the same as in the Homals call in Sect. 8.3.1; to put it simple, we mimicked Homals with a Princals call.

The copy concept is very appealing for mixed scale levels. For nominal transformations we are mostly interested in unrestricted category quantifications (sometimes also called *multiple nominal*), whereas for metric transformations (e.g., linear or monotone spline), we want to obtain loadings based on a restricted solution. For ordinal transformations both an unrestricted and a restricted solution can be considered. It depends on whether we want to put more emphasis on the category quantifications or the loadings.

Let us fit such a model via `Princals` using the same data setup as in Sect. 8.3.2. As we will see, this gives a more intuitive output by avoiding the seemingly awkward second-dimension transformations for the non-nominal variables. We use exactly the same knot specifications, ordinal restrictions, and degrees as in Sect. 8.3.2. The only new thing here is that we need to specify a copies vector for the nominal variables (i.e., six items, country, and gender).

```
copvec <- c(rep(2, 7), 1, 1, 2, 1)
prinmix <- princals(WPmixed, knots = knotlist,ordinal = ordvec,
                    degrees = degvec, copies = copvec)
```

Figure 8.9 shows the transformation plots for one of the items (legalized marijuana) and the remaining variables. For the legalized marijuana item, we

[3] A "copy" is literally a copy of a variable, achieved by adding it to the input matrix, as the function does internally.

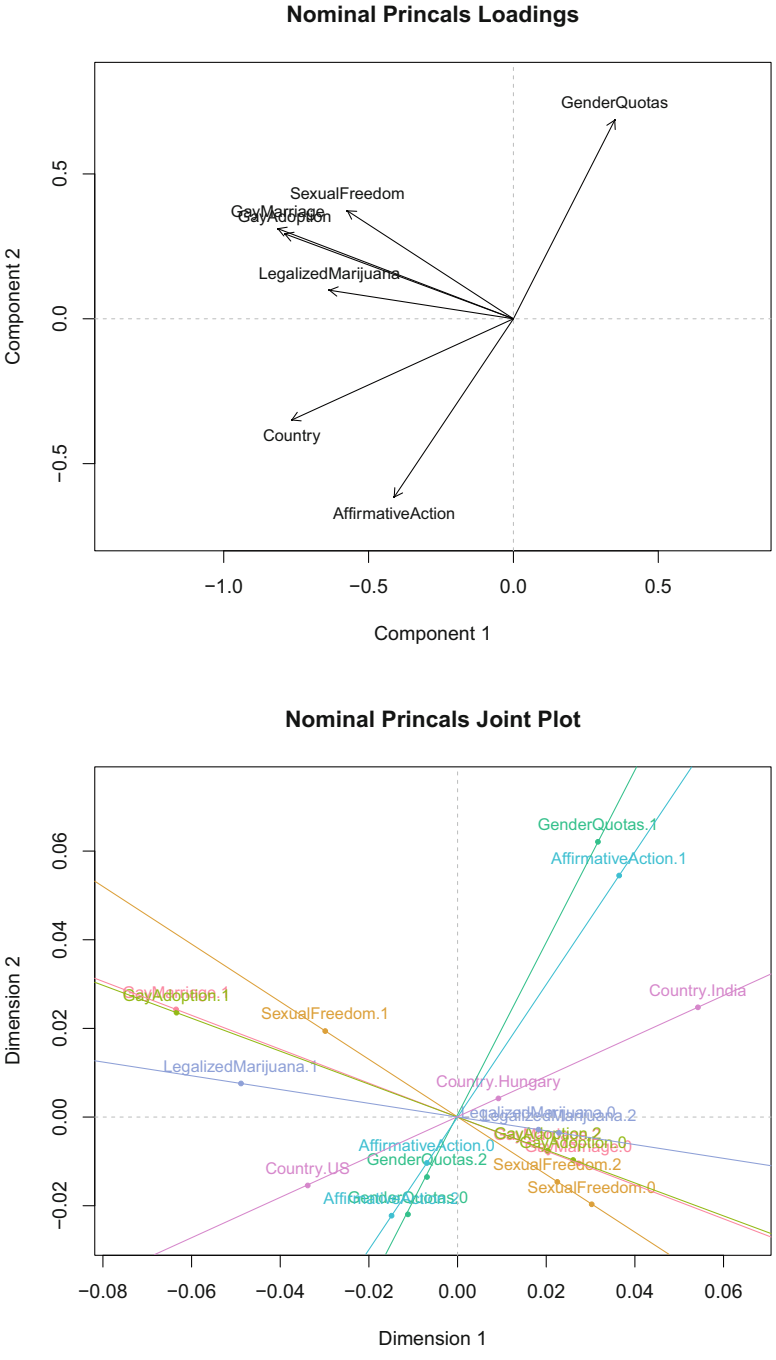

Fig. 8.8 Top panel: nominal Princals loadings plot. Bottom panel: joint plot with restricted category quantifications

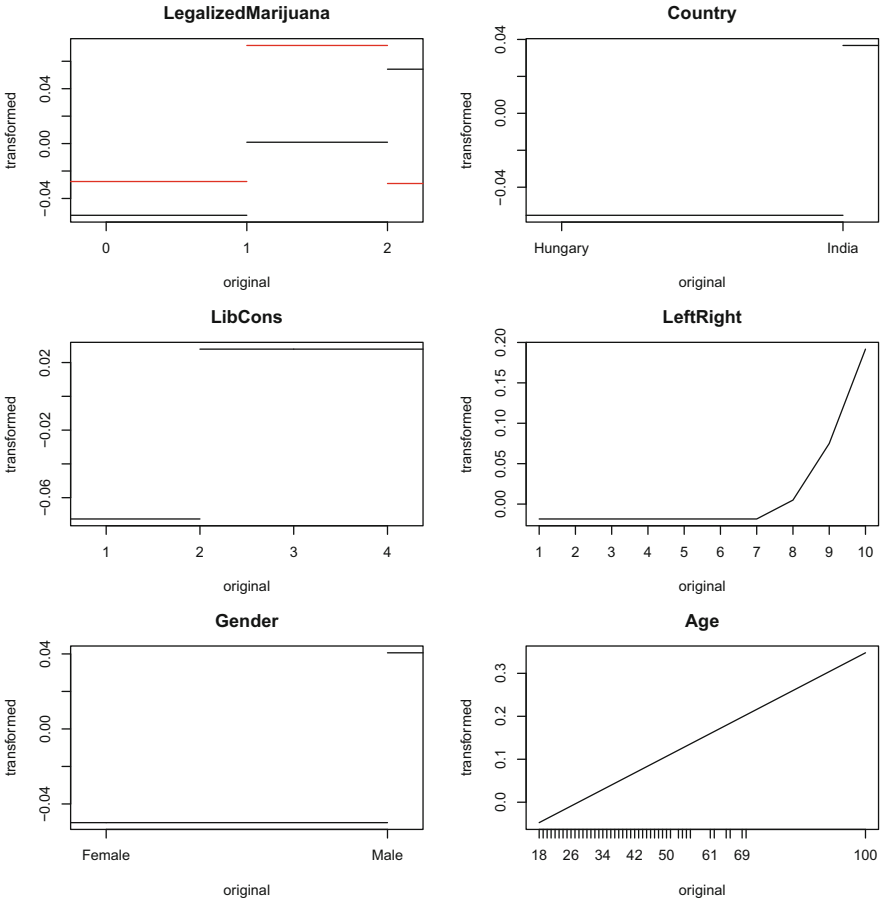

Fig. 8.9 Transformation plots for six variables of the Wilson-Patterson data. The black lines show the transformation on the first dimension and the red lines the transformations on the second dimension

get a transformation on each dimension. The original categories of the liberal-conservative item were merged into 1 vs. higher by the ordinal transformation. For the left-right item, we see the spline pattern (categories 1–7 get the same transformed scores). Age is taken as metric and, therefore, linearly transformed.

Figure 8.10 shows the joint plot with category quantifications for the variables transformed using a linear function, spline, and step function. For the liberal-conservative variable, in addition to the loading, we also plot the category quantifications (gray labels) which are on a straight line since we did not make a copy. All nominal quantifications were unrestricted and therefore free to vary in the 2D space.

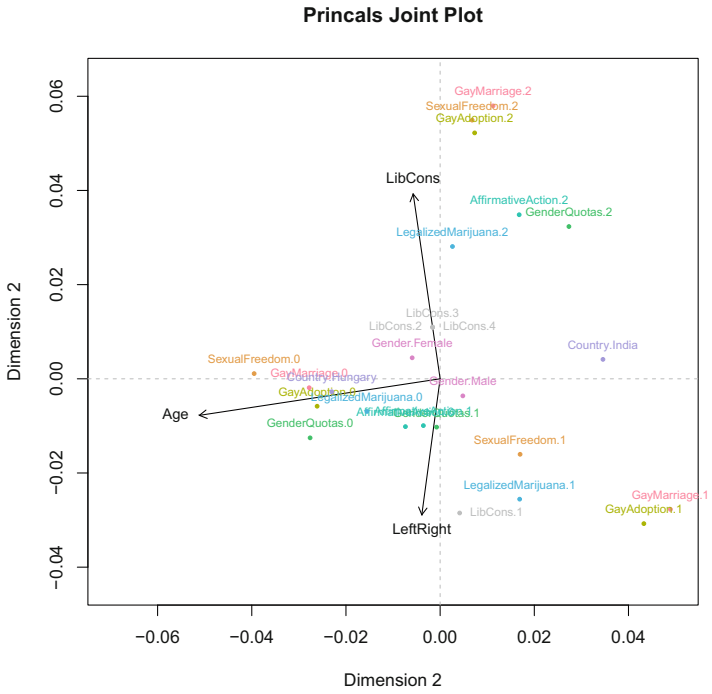

Fig. 8.10 Joint plot for combined Homals-Princals. For the nominal transformed variables, the category quantifications are plotted. For the linear and spline transformed variables (age, left-right), the loadings are plotted. For the ordinal transformed variable (liberal-conservative), we plot both (quantifications in gray). The loadings are divided by 10 for better mapping into the category quantification space

This analysis concludes Homals and Princals. Other interesting features of these methods are that they can handle missing data and that they allow one to specify variables as *passive*. In the examples considered so far, all variables were defined as *active*. That is, the solution was based on the full set of input variables. In case of external validations (e.g., treatment vs. control group), corresponding variables can be set to passive through the `active=FALSE` argument. Passive variables are not part of the optimization. After loadings, category quantifications and object scores are obtained; passive variables are scaled accordingly and can be added to the plot.

8.4 Lineals for CFA/SEM Preprocessing

An important "side effect" of the methods presented so far is that they implicitly aim to *linearize* the relationships between the variables involved. In a simple two-variable setting, this means that these approaches stretch and squeeze the categories

in a way such that the relationship between these variables becomes as linear as possible. For instance, as discussed in De Leeuw and Mair (2009b), in simple CA we always achieve perfect linearizability. In general multivariate settings such as in Princals or Homals, we do not achieve perfect linearization in general.

Why is linearization important? Multivariate methods such as exploratory/confirmatory factor analysis (EFA/CFA; see Chap. 2) and structural equation models (SEM; see Sect. 3.3) are based on an input correlation (or covariance) matrix. Correlation coefficients are only able to capture linear relationships among the variables (see Sect. 2.1); they are blind to nonlinear dependency patterns. When applying CFA or SEM, we often assume that linearity holds without further examination.[4]

Gifi models can be used to preprocess the data in terms of OS of the categories in order to make the relationships as linear as possible. The original values in the data matrix (first dimension) are then replaced by the score matrix containing optimally scaled category quantifications. Based on this data matrix, we compute a correlation matrix which acts as input to a CFA/SEM fit. Having Likert-type input data, this approach avoids the computation of polychoric correlation matrices.

In Sect. 8.2.2 we fitted an ordinal Princals model. The score matrix (first dimension) and the corresponding correlation matrix can be extracted as follows:

```
newdat <- prord$scoremat
newR <- prord$rhat
```

The object `newdat` replaces the initial input data, whereas `newR` is simply the Pearson correlation matrix based on the score matrix. Princals targets linearization implicitly.

A simple approach that targets linearization in a more explicit way is called *Lineals* (De Leeuw, 1988; Mair and De Leeuw, 2010). Note that in Lineals, we fit a single dimension solution only, as opposed to Princals/Homals. The idea of Lineals is based on the following two ingredients: Pearson's correlation coefficient and Pearson's *correlation ratio*.

The Pearson correlation matrix $\mathbf{R}$ of dimension $m \times m$ with elements r_{ij} is computed on the basis of the input data matrix $\mathbf{H}$. $\mathbf{R}$ captures the linear dependencies in the data. For Pearson's correlation ratio, we need to follow components. Let $\mathbf{B}$ be the Burt matrix as introduced in Sect. 7.2. Note that the Burt matrix can be obtained from the indicator matrix $\mathbf{G}$ via $\mathbf{B} = \mathbf{G}'\mathbf{G}$ and consists of the submatrices $\mathbf{B}_{ij} = \mathbf{G}_i'\mathbf{G}_i$. Let $\mathbf{D}_i$ be the diagonal matrix with the univariate margins on the diagonal, that is, $\mathbf{D}_i = \mathbf{B}_{ii}$. Finally, $\mathbf{y}_i$ are the optimally scaled category scores of variable i (score matrix $\mathbf{Y}$) which replace the original scores in $\mathbf{h}_i$. We can compute the *squared Pearson's correlation ratio* using

$$\eta_{ij}^2 = \mathbf{y}_i'\mathbf{C}_{ij}\mathbf{D}_i^{-1}\mathbf{C}_{ij}\mathbf{y}_i. \tag{8.2}$$

[4] Of course, the multivariate normality assumption implies linearity.

This squared correlation portion accounts for the nonlinear part of the data. Thus, perfect bilinearizability is achieved if $r_{ij}^2 - \eta_{ij}^2 = 0$. The optimization problem we have to tackle is the following:

$$\phi(\mathbf{Y}) = \sum_{i=1}^{m} \sum_{j=1}^{m} \left(r_{ij}^2 - \eta_{ij}^2 \right)^2 \rightarrow \text{min!} \tag{8.3}$$

In plain English this means that we transform the data in a way such that the squared difference between the linear portion and the nonlinear portion becomes minimal. The smaller the target value, the closer we are to perfect linearization.

Let us look at a simple example for Lineals CFA preprocessing using a dataset from Haegeli et al. (2012), who were interested in various risk-taking behaviors of out-of-bounds skiers. The skiers where exposed to the "Brief Sensation Seeking Scale" (BSSS-8; Hoyle et al., 2002). It is a short 8-item scale with 5-point response categories. The scale has four subscales (with two items each): experience seeking (ES), boredom susceptibility (BS), thrill and adventure seeking (TAS), and disinhibition (DIS). The Lineals fit using the **aspect** package (Mair and De Leeuw, 2010) can be achieved as follows:

```
library("MPsychoR")
library("aspect")
data("BSSS")
linbs <- lineals(BSSS)
linbs
##
## Call:
## lineals(data = BSSS)
##
## Value target function: 0.102
##
## Correlation matrix of the transformed data:
##             Explore Restless Frightning Party  Trip Friends Bungee Illegal
## Explore       1.000    0.354      0.282 0.184 0.271   0.232  0.105   0.269
## Restless      0.354    1.000      0.334 0.158 0.180   0.246  0.172   0.195
## Frightning    0.282    0.334      1.000 0.470 0.310   0.455  0.438   0.505
## Party         0.184    0.158      0.470 1.000 0.326   0.508  0.349   0.532
## Trip          0.271    0.180      0.310 0.326 1.000   0.542  0.266   0.353
## Friends       0.232    0.246      0.455 0.508 0.542   1.000  0.400   0.507
## Bungee        0.105    0.172      0.438 0.349 0.266   0.400  1.000   0.455
## Illegal       0.269    0.195      0.505 0.532 0.353   0.507  0.455   1.000
```

The output shows the value of the loss function from Eq. (8.3) and the new, induced correlation matrix based on the score matrix $\mathbf{Y}$.

Now we fit a second-order CFA (see Sect. 2.4.2) using **lavaan** (Rosseel, 2012). This model involves a general risk factor associated with each of the four subfactors. We use an asymptotical distribution-free estimator which **lavaan** calls weighted least squares (WLS). For this type of estimator, we need to provide the full dataset, as opposed to other estimation methods which only need the correlation/covariance

matrix. The crucial point is the following: instead of the original data matrix, we throw the optimally scaled data matrix into the CFA call, as determined by Lineals.

```
library("lavaan")
BSSSnew <- linbs$scoremat
BSSS.model <- 'ES =~ Explore + Trip
               BS =~ Restless + Friends
               TAS =~ Frightning + Bungee
               DIS =~ Party + Illegal
               Risk =~ ES + BS + TAS + DIS'
cfabs <- cfa(BSSS.model, data = BSSSnew,
             estimator = "WLS")
```

The fit of this model is not terribly great (RMSEA = 0.098), but this example should be sufficient to give the reader an idea of how Lineals preprocessing works for CFA/SEM models.

Mair and De Leeuw (2010) provide a large set of other OS transformations of correlation matrices (called *correlational aspects*) with respect to a particular target to be optimized. All of them are implemented in the **aspect** package.

References

De Leeuw, J. (1988). Multivariate analysis with linearizable regressions. *Psychometrika, 53*, 437–454.

De Leeuw, J., & Mair, P. (2009a). Gifi methods for optimal scaling in R: The package **homals**. *Journal of Statistical Software, 31*(1), 1–21. https://www.jstatsoft.org/index.php/jss/article/view/v031i04

De Leeuw, J., & Mair, P. (2009b). Simple and canonical correspondence analysis using the R package **anacor**. *Journal of Statistical Software, 31*(5), 1–18. http://www.jstatsoft.org/v31/i05/

De Leeuw, J., Mair, P., & Groenen, P. J. F. (2017). *Multivariate analysis with optimal scaling*. http://gifi.stat.ucla.edu/gifi/_book/

Gifi, A. (1990). *Nonlinear multivariate analysis*. Chichester: Wiley.

Haegeli, P., Gunn, M., & Haider, W. (2012). Identifying a high-risk cohort in a complex and dynamic risk environment: Out-of-bounds skiing—An example from avalanche safety. *Prevention Science, 13*, 562–573.

Hoyle, R. H., Stephenson, M. T., Palmgreen, P., Pugzles Lorch, E., & Donohew, R. L. (2002). Reliability and validity of a brief measure of sensation seeking. *Personality and Individual Differences, 32*, 401–414.

Jacoby, W. G. (1991). *Data theory and dimensional analysis*. Thousand Oaks: Sage.

Jacoby, W. G. (1999). Levels of measurement and political research: An optimistic view. *American Journal of Political Science, 43*, 271–301.

Koller, I., Levenson, M. R., & Glück, J. (2017). What do you think you are measuring? A mixed-methods procedure for assessing the content validity of test items and theory-based scaling. *Frontiers in Psychology, 8*(126), 1–20.

Linting, M., Meulman, J. J., Groenen, P. J. F., & van der Kooij, A. J. (2007). Nonlinear principal components analysis: Introduction and application. *Psychological Methods, 12*, 336–358.

Mair, P., & De Leeuw, J. (2010). A general framework for multivariate analysis with optimal scaling: The R package **aspect**. *Journal of Statistical Software, 32*(1), 1–23. https://www.jstatsoft.org/index.php/jss/article/view/v032i09

Mair, P., & De Leeuw, J. (2017). **Gifi**: Multivariate analysis with optimal scaling. R package version 0.3-2. https://R-Forge.R-project.org/projects/psychor/

Michailidis, G., & De Leeuw, J. (1998). The Gifi system of descriptive multivariate analysis. *Statistical Science, 13*, 307–336.

Mori, Y., Kuroda, M., & Makino, N. (2016). *Nonlinear principal component analysis and its applications*. New York: Springer.

Rosseel, Y. (2012). **lavaan**: An R package for structural equation modeling. *Journal of Statistical Software, 48*(2), 1–36. http://www.jstatsoft.org/v48/i02/

Stevens, S. S. (1946). On the theory of scales of measurement. *Science, 103*, 677–680.

Wood, S. N. (2017). *Generalized additive models: An introduction with R* (2nd ed.). Boca Raton: CRC Press.

Young, F. W. (1981). Quantitative analysis of qualitative data. *Psychometrika, 46*, 357–388.

Chapter 9
Multidimensional Scaling

9.1 Proximities

Multidimensional scaling (MDS) is a technique that represents *proximities* among *objects* as *distances* among points in a low-dimensional space (with given dimensionality). It allows researchers to explore or test similarity structures among objects in a multivariate dataset (Mair et al., 2016). Let us disentangle this definition step-by-step.

"Proximities" is an umbrella term for *similarities* and *dissimilarities* and has a long tradition in psychological research. It simply refers to the closeness of objects in a certain domain, that is, how similar/dissimilar objects are to each other. There are two basic strategies to obtain proximities:

- *Directly observed proximities*: They are directly collected within an experimental setting. For instance, pairs of stimuli are presented to a participant, and he/she has to rate the perceived similarity using a slide bar.
- *Derived proximities*: They are derived from a persons $\times$ variables data matrix (e.g., by computing a correlation matrix).

A classical example of directly observed proximities, often used in textbooks on MDS, are Ekman's color data. In this psychophysical experiment, participants had to rate the similarities between pairs of 14 colors (Ekman, 1954). Here, the colors are the objects, and a simple proximity matrix Δ can be derived by averaging the ratings across all participants. Δ is symmetric and of dimension $n \times n$, where n denotes the number of objects. This dataset is included in the **smacof** package (De Leeuw and Mair, 2009) which will be used throughout this chapter.

All the examples presented below are based on derived proximities, which we will now discuss in detail. First of all, we need to decide in which direction of the data matrix we want to scale. Objects can be persons (compute proximities between pairs of persons) or variables (compute proximities between variables). The latter is certainly the more common case.

P. Mair, *Modern Psychometrics with R*, Use R!,
https://doi.org/10.1007/978-3-319-93177-7_9

Next, the MDS definition above includes the terms "proximities" and "distances." Distances are a special kind of dissimilarities which fulfill the following three properties: they are nonnegative real values, they are symmetric (i.e., the distance from x to y is the same as the distance from y to x), and they satisfy the triangle inequality (i.e., the "direct" distance from x to z is not greater than the distance from x to z via y). Corresponding technical details can be found in Gower and Legendre (1986).

In MDS we cannot generally assume that the input data fulfill these properties; therefore we use the term dissimilarities. These dissimilarities are always represented as distances after fitting an MDS. More precisely, they are Euclidean distances (i.e., distances that correspond to our natural notion of the distance between two points in geometric space "as the crow flies").

Since we derive the proximities from our persons $\times$ variables data structure, various proximity measures can be considered (see Cox and Cox, 2001, for a detailed overview). The most popular proximity measure is the Pearson correlation coefficient $r(\mathbf{x}, \mathbf{y})$, which is a similarity measure, of course (i.e., the higher the correlation, the more similar two objects are to each other). However, the **smacof** package requires input dissimilarities rather than similarities. Thus, we need to convert the correlation into a dissimilarity measure. There are several ways of achieving this such as $d(\mathbf{x}, \mathbf{y}) = 1 - r(\mathbf{x}, \mathbf{y})$ or $d(\mathbf{x}, \mathbf{y}) = \sqrt{1 - r(\mathbf{x}, \mathbf{y})}$. The latter transformation is typically preferred since the resulting dissimilarities are Euclidean (see Gower and Legendre, 1986, p. 10). If we want to disregard negative correlations, these formulas change to $d(\mathbf{x}, \mathbf{y}) = 1 - |r(\mathbf{x}, \mathbf{y})|$ and $d(\mathbf{x}, \mathbf{y}) = \sqrt{1 - |r(\mathbf{x}, \mathbf{y})|}$, respectively.

Instead of computing correlations and subsequently converting them into dissimilarities, another popular strategy to compute derived proximities is to use one of the following distance measures:

- Euclidean distance (metric data).
- Jaccard distance (binary data).
- Gower coefficient (mixed scale levels).

In R, such dissimilarity computations can be carried out using the `dist` function; more exotic measures can be found in the **proxy** (Meyer and Buchta, 2015) package. Borg et al. (2018) elaborate on how various types of input dissimilarities affect the MDS solution.

In the following sections, we introduce MDS formally and elaborate on different types of MDS. In terms of additional literature, the MDS bible is Borg and Groenen (2005). A less comprehensive, more applied MDS book is Borg et al. (2018). Yet another good book on MDS is Cox and Cox (2001).

9.2 Exploratory MDS

Using MDS as an exploratory technique has a long tradition in the psychometric literature. It goes back to an article by Torgerson (1952). The author introduced *classical scaling* which solves the MDS problem analytically. Other important early

MDS contributions were the articles by Shepard (1962a,b) and Kruskal (1964a,b). An optimization breakthrough was achieved by De Leeuw (1977) who proposed a numerical approach (*majorization*) to solve the MDS problem. This idea led to the SMACOF (Scaling by MAjorizing a COmplicated Function) framework. Because of its flexibility, SMACOF is nowadays considered as the state-of-the-art approach for fitting MDS models.

9.2.1 SMACOF Theory

The input dissimilarities (either directly observed or derived) based on n objects and denoted by δ_{ij} are collected in a symmetric dissimilarity matrix $\boldsymbol{\Delta}$ of dimension $n \times n$. The problem we solve is to locate i, $j = 1, \ldots, n$ points in a low-dimensional space (p as the number of dimensions) such that the Euclidean distances between these points are as close as possible to the given input dissimilarities δ_{ij}. We aim to find an $n \times p$ matrix $\mathbf{X}$ containing the *configuration*, that is, the coordinates in the MDS space. The fitted Euclidean distances are

$$d_{ij}(\mathbf{X}) = \sqrt{\sum_{s=1}^{p}(x_{is} - x_{js})^2} \tag{9.1}$$

We typically aim for a low p (i.e., $p = 2$ or $p = 3$) such that we can produce a configuration plot.

There is one important twist in modern MDS techniques pertaining to possible transformations of the input dissimilarities. Instead of using the raw δ_{ij}'s, *optimal scaling* (OS) transformations are applied, as introduced in Sect. 8.1.1 within the context of Gifi methods. In MDS, the most popular transformations are the following (all of them are monotone):

- Ratio MDS: $\hat{d}_{ij} = b\delta_{ij}$.
- Interval MDS: $\hat{d}_{ij} = a + b\delta_{ij}$.
- Ordinal MDS: $\hat{d}_{ij} = f(\delta_{ij})$ where f is a monotone step function.

The corresponding $\hat{d}_{ij}$ are referred to as *disparities* or, simply, *d-hats*. Ratio MDS fits a slope parameter b only, interval MDS fits both intercept a and slope b, and ordinal MDS fits a monotone step function.

In ordinal MDS, where we define the dissimilarity scale level as ordinal, *tied* dissimilarities require special treatment. The two common approaches are:

- *Primary approach* ("break ties"): if δ_{ij} and $\delta_{i'j'}$ are equal, the disparities $\hat{d}_{ij}$ and $\hat{d}_{i'j'}$ are not necessarily equal.
- *Secondary approach* ("keep ties tied"): if δ_{ij} and $\delta_{i'j'}$ are equal, $\hat{d}_{ij}$ and $\hat{d}_{i'j'}$ will be equal as well.

Note that ordinal MDS is sometimes also called *nonmetric* MDS, whereas interval/ratio MDS go under the umbrella term *metric* MDS.

How do we pick the right transformation function? If there is metric information in the δ_{ij}'s, interval MDS can be used. Interval MDS is nowadays typically preferred over ratio MDS since it provides a better fit. If the δ_{ij}'s are ordinal or we are not sure and want to "play it safe," an ordinal MDS can be used. Alternatively, we can also decide on a data-driven base using Shepard plots. This idea will be presented in Sect. 9.2.3.

At this point we have all the ingredients together in order to establish the MDS target function which is called *stress*:

$$\sigma(\mathbf{X}) = \sum_{i<j} (\hat{d}_{ij} - d_{ij}(\mathbf{X}))^2. \tag{9.2}$$

This expression is the *raw stress*, and its computation is based on all lower triangular elements of $\boldsymbol{\Delta}$. MDS programs typically print out a standardized version of it, called *stress-1*, such that the resulting stress value does not depend on the absolute magnitudes of the input dissimilarities. In either case, the lower the stress value $\sigma(\mathbf{X})$, the better the fit. The lower bound for the stress-1 is 0, in case of a perfect fit.[1]

9.2.2 Exploratory MDS Example: PTSD Symptoms

The **smacof** package offers implementations for a large variety of MDS models. The function to fit a basic exploratory MDS is called `mds()`. In SMACOF slang this option is called *symmetric* SMACOF since we operate on a symmetric input dissimilarity matrix. Each of the δ_{ij}'s has to be nonnegative; missing values are allowed.

The dataset we use to illustrate an exploratory MDS fit is from the clinical psychology area. McNally et al. (2015) collected data on PTSD (post-traumatic stress disorder) symptoms reported by survivors of the Wenchuan earthquake in China using the *PTSD checklist-civilian* (PCL-C; Weathers et al., 1993). In total, there are 17 PTSD symptom items scaled on a 5-point rating scale (1 ... "not at all"; 5 ... "extremely"). We are interested in representing associations among the PTSD symptoms. First, we compute derived dissimilarities using the Euclidean distance, resulting in a 17×17 symmetric dissimilarity matrix. For the moment, let us consider an ordinal scale level for the input dissimilarities and fit a two-dimensional, ordinal MDS:

[1] The upper stress value is not trivial to determine (see De Leeuw and Stoop, 1984), but it cannot become larger than 1.

```
library("MPsychoR")
library("smacof")
data(Wenchuan)
Wdelta <- dist(t(Wenchuan))      ## Euclidean distances
fit.wenchuan1 <- mds(Wdelta, type = "ordinal")  ## MDS fit
fit.wenchuan1
##
## Call:
## mds(delta = Wdelta, type = "ordinal")
##
## Model: Symmetric SMACOF
## Number of objects: 17
## Stress-1 value: 0.133
## Number of iterations: 35
```

The MDS fit results in a stress value[2] of 0.133. Whether the stress value in this example suggests a good model fit is not trivial to answer. It requires detailed examination, as elaborated in the next section. For the moment we are happy with it. Now we can produce the configuration plot:

```
plot(fit.wenchuan1, main = "Wenchuan MDS")
```

Figure 9.1 shows how the PTSD symptoms are related to each other. For instance, we see that "avoiding thinking about or talking about a stressful experience from the past or avoiding having feelings related to it" (`avoidth`) and "avoiding activities or situations because they reminded you of a stressful experience from the past" (`avoidact`) are virtually on the same spot. We also see that "feeling very upset when something reminded you of a stressful experience from the past?" (`upset`) and "feeling emotionally numb or being unable to have loving feelings for those close to you" (`numb`) are the farthest points on the first dimension, whereas "trouble remembering important parts of a stressful experience from the past" (`amnesia`) and "feeling irritable or having angry outbursts" (`anger`) are the extreme points in the second dimension. These pairs of symptoms are pretty much unrelated to each other.

Can we interpret the dimensions as, for instance, in exploratory factor analysis (EFA) or principal component analysis (PCA)? As Borg et al. (2018) point out, there is no natural law that guarantees that the dimensions are always interpretable. In this application we cannot establish a meaningful clinical interpretation of the dimensions. Note that an MDS configuration can be rotated arbitrarily, if it helps for interpretation.

[2]From now on, whenever we say "stress," we are referring to "stress-1."

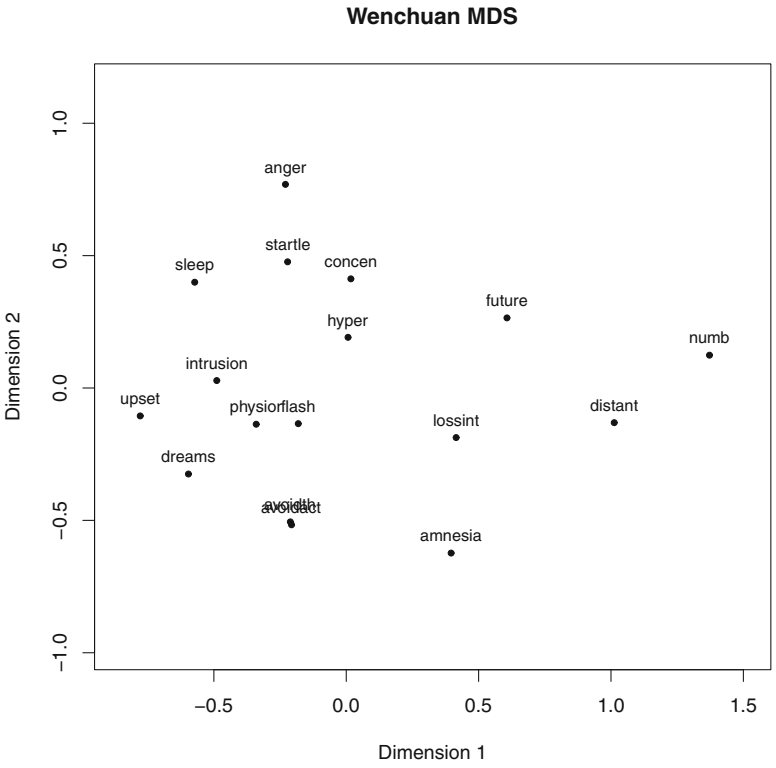

Fig. 9.1 Wenchuan MDS: configuration plot for the 2D, ordinal MDS solution

Another possibility to interpret a configuration plot is through regional interpretation where we divide the configuration space into regions (or *facets*) of points. For this example we will consider such an interpretation after a thorough goodness-of-fit examination.

9.2.3 Goodness of Fit in MDS

Goodness-of-fit examination in MDS is discussed in detail in Mair et al. (2016). Here we focus on the most important tools. A common mistake is to interpret stress too mechanically by relying on Kruskal's rules of thumb (see Kruskal, 1964a, p. 3): 0.20 …"poor," 0.10 …"fair," 0.05 …"good," 0.025 …"excellent," and 0 …"perfect." This is problematic due to the fact that the magnitude of the stress depends on the number of objects n (the larger n, the larger the stress). In modern MDS applications, n can be fairly large. Instead of using these rules of thumb, we can consider simulation approaches. An old approach (Spence and Ogilvie, 1973) is

to simulate random dissimilarities for a fixed n and p and fit the corresponding MDS model on these matrices. This leads to *random stress norms*. Here is an example using the ordinal Wenchuan fit from above.

```
set.seed(123)
rsvec <- randomstress(n = attr(Wdelta, "Size"), ndim = 2,
                      nrep = 500, type = "ordinal")
mean(rsvec)
## [1] 0.2770934
mean(rsvec) - 2*sd(rsvec)
## [1] 0.2548536
```

The `randomstress` call gives 500 random stress values. These stress norms represent a "bad fit" benchmark. Our solution should be clearly below the average random stress. Often, in the literature, an observed stress values is considered "significant" if it is smaller than the lower $2 \times sd$ random stress boundary, which is clearly the case in our example (the stress was 0.133).

Note that this random stress criterion is not very sharp; as Cliff (1973) puts it, it tests the "nullest of all null hypotheses." A sharper approach is to use a permutation test as described in Mair et al. (2016). For derived dissimilarities, it resamples the original data, and for each resulting dissimilarity matrix, an MDS fit is carried out. This gives a stress distribution under the H_0: "stress/configuration is obtained from a random permutation of dissimilarities."

```
set.seed(123)
permmds <- permtest(fit.wenchuan1, data = Wenchuan,
                    method.dat = "euclidean", nrep = 500,
                    verbose = FALSE)
permmds
##
## SMACOF Permutation Test
## Number of objects: 17
## Number of replications (permutations): 500
##
## Observed stress value: 0.133
## p-value: 0
```

Since $p < 0.05$, we reject the H_0 and conclude that stress/configuration is obtained from something other than a random permutation of dissimilarities.

However, goodness of fit in MDS should not solely be judged using random stress norms and permutation tests. As in PCA, we can produce a scree plot with the stress-1 values on the y-axis (see Fig. 9.2) and apply the elbow criterion, if possible. The larger the p, the smaller the stress. Using the Wenchuan dataset from above, we fit an ordinal MDS with varying dimensions (from $p = 1$ to the maximum of $p = n - 1$).

```
n <- attr(Wdelta, "Size")
svec <- NULL
for (i in 1:(n-1)) {
  svec[i] <- mds(Wdelta, ndim = i, type = "ordinal")$stress
}
```

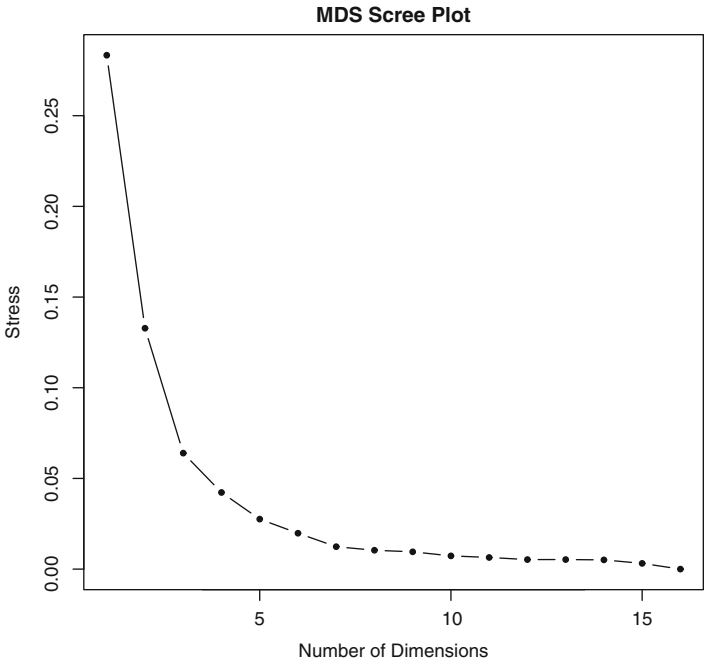

Fig. 9.2 MDS scree plot for Wenchuan data

Compared to PCA and EFA, in MDS we aim for a $p = 2$ or 3 since the main output is the configuration plot. In PCA/FA a high-dimensional solution is not at all problematic, as long as we can name the components/factors. Based solely on the scree plot, we would probably pick a 3D solution, but the stress of the 2D solution is not too bad either, as judged by the random stress norms, the permutation test results, and Kruskal's stress benchmarks (considered here since $n = 17$ is fairly small).

The stress target function is known to be bumpy. Thus, it can easily happen that we end up in a local minimum (i.e., we do not obtain the best possible solution). Where the algorithm ends up in the end depends on where it starts. By default, the functions in the **smacof** package use a classical scaling solution (Torgerson, 1952) as *starting configuration*. This is not necessarily always the best choice. The local

minimum problem including a systematic starting solution search is described in detail Borg and Mair (2017). Here we use a simple ad hoc strategy by trying out different random starts and check whether the best random start solution leads to a lower stress value than the default setup. Below we examine 100 random starts and extract the stress values.

```
set.seed(123)
fit.wenchuan <- NULL  ## 100 random starts
for(i in 1:100) {
  fit.wenchuan[[i]] <- mds(Wdelta, type = "ordinal",
                           init = "random")
}
## extract the best solution
ind <- which.min(sapply(fit.wenchuan,
                        function(x) x$stress))
fit.wenchuan2 <- fit.wenchuan[[ind]]
fit.wenchuan2$stress        ## lowest stress (random start)
## [1] 0.1327943
fit.wenchuan1$stress        ## stress (classical scaling start)
## [1] 0.1328058
```

We see that a particular random start provided a slightly better stress than our original solution. Since the stress difference is so minimal, we could pick either solution. Let us proceed with the best random start solution.

In the previous section, we addressed briefly the option for a data-driven choice of the transformation function. A plot that gives us insight into these OS transformations is the *Shepard diagram* (De Leeuw and Mair, 2015). In our example the symptoms were scored on 5-point rating scales. We can assume that there is some metric information in it and, therefore, fit an interval MDS. Naturally, the stress value of the interval MDS will be worse than the ordinal stress. However, if there are only slight differences, we may prefer the interval solution. The corresponding linear transformation is much simpler to interpret, most likely more robust across multiple samples (i.e., replicable), and, from a statistical point of view, more parsimonious than the ordinal step function.

```
fit.wenchuan3 <- mds(Wdelta, type = "interval")
fit.wenchuan3
##
## Call:
## mds(delta = Wdelta, type = "interval")
##
## Model: Symmetric SMACOF
## Number of objects: 17
```

(continued)

```
## Stress-1 value: 0.184
## Number of iterations: 9
```

The resulting interval stress is clearly larger than the ordinal stress of 0.133. The Shepard plots for both MDS models fitted so far can be produced as follows:

```
plot(fit.wenchuan2, plot.type = "Shepard",
     main = "Shepard Diagram (Ordinal MDS)")
plot(fit.wenchuan3, plot.type = "Shepard",
     main = "Shepard Diagram (Interval MDS)")
```

Figure 9.3 gives us a detailed insight into the dissimilarity transformations. We see that, especially for smaller dissimilarities, the step function approximates the points much better than the interval fit. This explains the differences in the stress values.

Equation (9.2) shows that stress is an aggregate index of individual components. The contribution of each object to the total stress can be computed easily. The

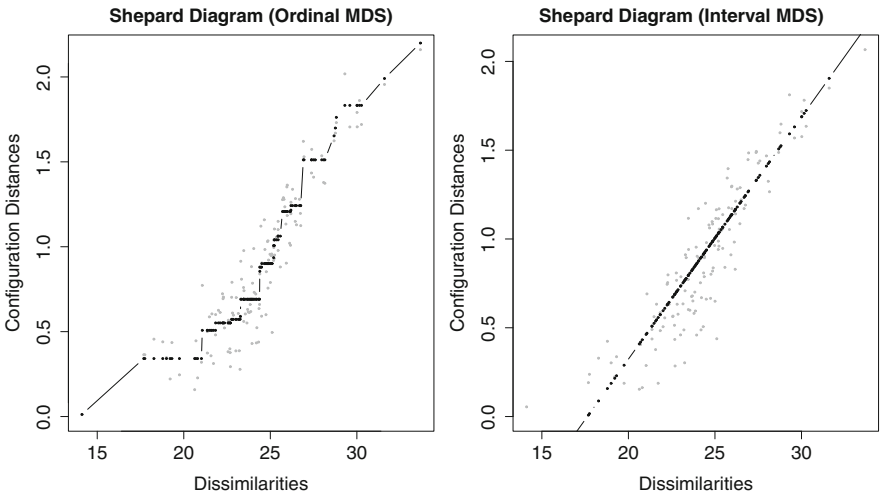

Fig. 9.3 Shepard plots with black dots showing the disparities. Left panel: ordinal MDS. Right panel: interval MDS

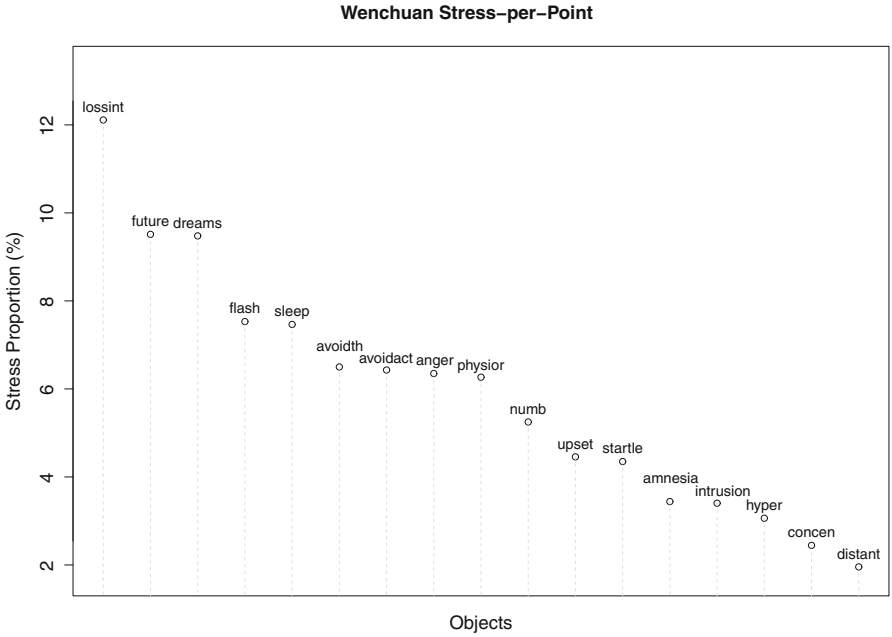

Fig. 9.4 Stress-per-point plot for Wenchuan MDS

resulting values, typically reported as percentages, are called *stress-per-point* (SPP). Figure 9.4 shows a graphical illustration.

```
plot(fit.wenchuan2, plot.type = "stressplot",
     main = "Wenchuan Stress-per-Point")
```

We can think of points with high SPP contribution in a similar way as influential outliers in regression. In our example, we see that "loss of interest in activities that you used to enjoy" (`lossint`) provides a fairly high stress contribution of 12.11%. From a statistical point of view, this indicates that this symptom has a "special" relationship to the remaining symptoms in the solution. A nice way to incorporate the SPP information into a configuration plot are *bubble plots* (see `plot.smacof` function; large bubbles indicate points with high SPP values).

At this point we have two options: either we keep the point in the model and proceed with additional goodness-of-fit examination or we eliminate the point from the analysis. Let us keep this point for the moment and decide later whether it should be kicked out.

A useful option to examine the stability of an MDS solution is to use a bootstrap strategy, as proposed in Jacoby and Armstrong (2014). Simply speaking,

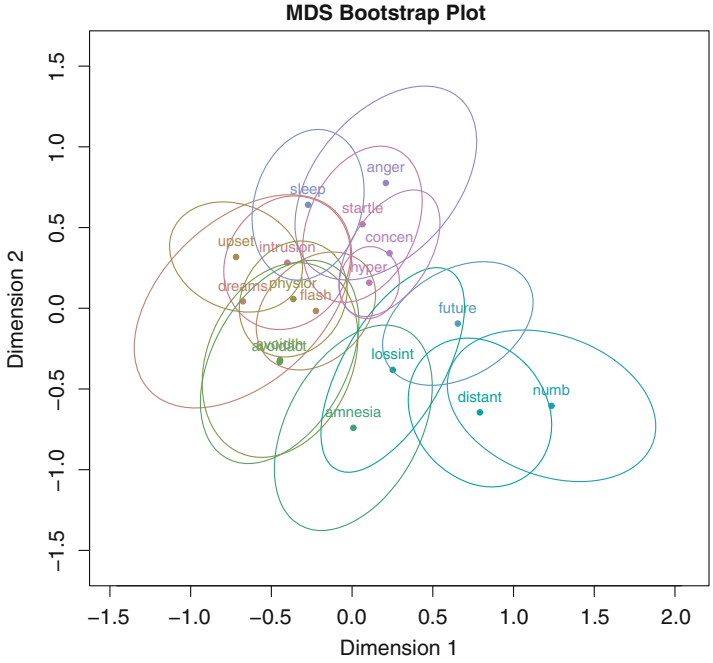

Fig. 9.5 Bootstrapped Wenchoun MDS with 95% confidence ellipses

it resamples the rows in the original persons × variable matrix and fits an MDS solution for each of the subsamples. Based on these bootstrapped MDS solutions, confidence ellipses can be plotted around the original points. The following chunk performs the bootstrap (100 samples) and plots the configuration including the 95% confidence ellipses.

```
library("colorspace")
set.seed(123)
bootWen <- bootmds(fit.wenchuan2, data = Wenchuan,
                   method.dat = "euclidean", nrep = 100)
cols <- rainbow_hcl(17, l = 60)
plot(bootWen, col = cols)
```

Figure 9.5 shows the variations of each object across multiple samples. This gives us an idea of how stable the MDS configuration is across multiple samples. Another option for stability assessment is to apply a jackknife, as implemented in **smacof**'s `jackknife` function.

This concludes our prototype goodness-of-fit evaluation of an MDS solution. It is imperative to look at all these results in combination instead of doing a "mechanical"

evaluation of the stress value. What are the options in case of a misfit? Basically, we have two options: either we keep the dimensionality and remove one (or more) objects with the highest SPP values or we increase the number of dimensions and examine the goodness of fit again. It can happen that in some applications, more than three dimensions are required. For such solutions we can produce configuration plots for pairs of dimensions (e.g., D1 vs. D2, D1 vs. D3, D2 vs. D3, etc.). Note that if, for interpretation purposes, the solution needs to be rotated, none of the goodness-of-fit statistics changes.

What is left to do in our Wenchuan example is the clinical interpretation of the configuration. The interpretability of a solution is in general an important criterion when it comes to goodness-of-fit assessment. The final configuration plot is given in Fig. 9.6. None of the two dimensions can be meaningfully interpreted. Rather, we can establish three regions of symptoms which correspond exactly to the three symptom clusters conceptualized in the DSM-IV (American Psychiatric Association, 1994): intrusive recollection (*B-cluster*), avoidance/numbing (*C-cluster*), and arousal (*D-cluster*). In addition, we can interpret the position of each point to all the other points since all distances are Euclidean (e.g., the symptoms `avoidth` and `avoidact` are highly similar since they are very close to each other in the configuration plot). There is no need here to exclude symptoms with high SPP since the obtained stress value is fairly low and the interpretation is quite clear.

Note that in this example we assigned the symptoms to regions based on the information from the DSM-IV. From a more data-driven point of view, we can also compute a cluster analysis on the fitted MDS configuration distances (e.g., by using hierarchical clustering; see Mair et al. (2014) for an example) and incorporate the cluster membership information into the configuration plot by coloring the objects accordingly.

9.3 Confirmatory MDS

The MDS fit in the example above was carried out in an exploratory manner in the sense that there were no restrictions on the configuration $\mathbf{X}$ when optimizing the stress in Eq. (9.2). In some applications we have a priori information about the configuration. This could be, for instance, an underlying psychological theory that tells us in which region of the MDS space certain groups of objects are supposed to be located. If this information is directly incorporated into the stress equation, we call this an MDS with *external constraints* on the configuration.[3]

Another example of a priori information includes circular theories about the configuration. In such cases we can force the points to be located on a circle (or, more generally, on a sphere for $p > 2$). This is an example of an MDS with

[3]Note that in the example above we had such an information about symptom regions by means of the DSM-IV clusters, but this was not anyhow incorporated into the model fit.

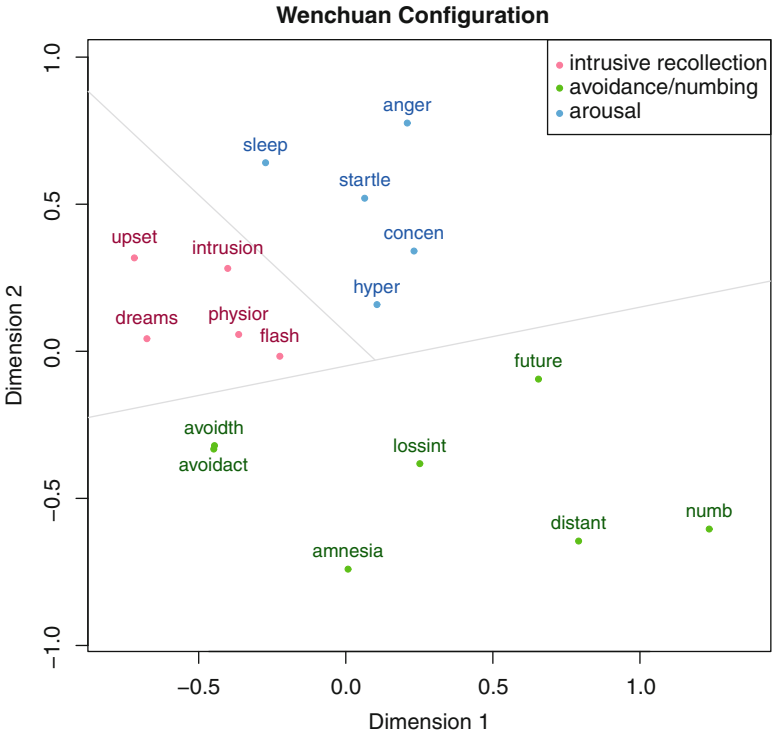

Fig. 9.6 Final configuration plot for Wenchuan MDS. Regions represent the three DSM-IV clusters: intrusive recollection, avoidance/numbing, and arousal

internal constraints on the configuration. Both MDS variants belong to the family of *confirmatory* MDS models and will be illustrated in the following two subsections. For either scenario it can be of interest to compare the stress value of the restricted solution with the stress from an unrestricted solution.

9.3.1 MDS with External Constraints

Borg and Lingoes (1980), De Leeuw and Heiser (1980), and Heiser and Meulman (1983) proposed restricted MDS models with external constraints on the configuration. The simplest form of a constraint is to formulate a linear restriction on the $n \times p$ configuration matrix $\mathbf{X}$:

$$\mathbf{X} = \mathbf{ZC}. \tag{9.3}$$

$\mathbf{Z}$ is a known predictor matrix of dimension $n \times q$ which imposes a dimensionality system on $\mathbf{X}$. The predictors can be, for instance, numerical in terms of external covariates, or one can specify an ANOVA-like indicator matrix. $\mathbf{C}$ is a $q \times p$ matrix with weights to be estimated. Equation (9.3) can be incorporated into the stress function (see Eq. (9.2)) and solved in an ordinal, interval, and ratio MDS manner.

There is one more technical aspect we can take into account for this type of MDS models. Let us consider a simple example where our predictor is categorical (let us say, four groups) and the matrix $\mathbf{Z}$ is established accordingly. In a configuration plot, we would have four different points only: one for the group 1 objects, one for the group 2 objects, etc. This can be interesting for some special applications, but in general this is not desirable. Rather, we want the objects belonging to a group to inhabit a certain MDS region, but they should not be exactly on the same spot. In order to fit such regionally constrained restrictions, we can incorporate the idea of OS into Eq. (9.3). Instead of using the observed restriction matrix $\mathbf{Z}$, we perform an additional OS step on the predictors. Equation (9.3) becomes

$$\mathbf{X} = \hat{\mathbf{Z}}\mathbf{C}, \tag{9.4}$$

with $\hat{\mathbf{Z}} = f(\mathbf{Z})$, where $f(\cdot)$ can be a ratio, interval, or ordinal transformation.

To illustrate a regionally constrained MDS with OS on the external variables, we use an example from work psychology. Bilsky and Jehn (2002) and Borg et al. (2011) use data based on the *organizational culture profile* (OCP; O'Reilly et al., 1991). The OCP is an instrument that contains a set of value statements that can be used to idiographically assess both the extent to which certain values characterize a target organization and an individual's preference for that particular configuration of values. The OCP requires individuals to sort 54 items into nine ordered categories. Most of the 54 items can be classified into four classes derived from Schwartz' theory of values (Schwartz, 1992): conservation, openness to change, self-transcendence, and self-enhancement. Twelve items remain unclassified. The data are included in the **smacof** package by means of a 54×54 correlation matrix and an additional column with the items' codings.

We convert the similarities into dissimilarities and fit an unrestricted, ordinal 2D MDS solution first. The resulting configuration plot is given in Fig. 9.7.

```
library("MPsychoR")
library("smacof")
ocpD <- sim2diss(OCP[,1:54])
fit.ocp1 <- mds(ocpD, type = "ordinal")
```

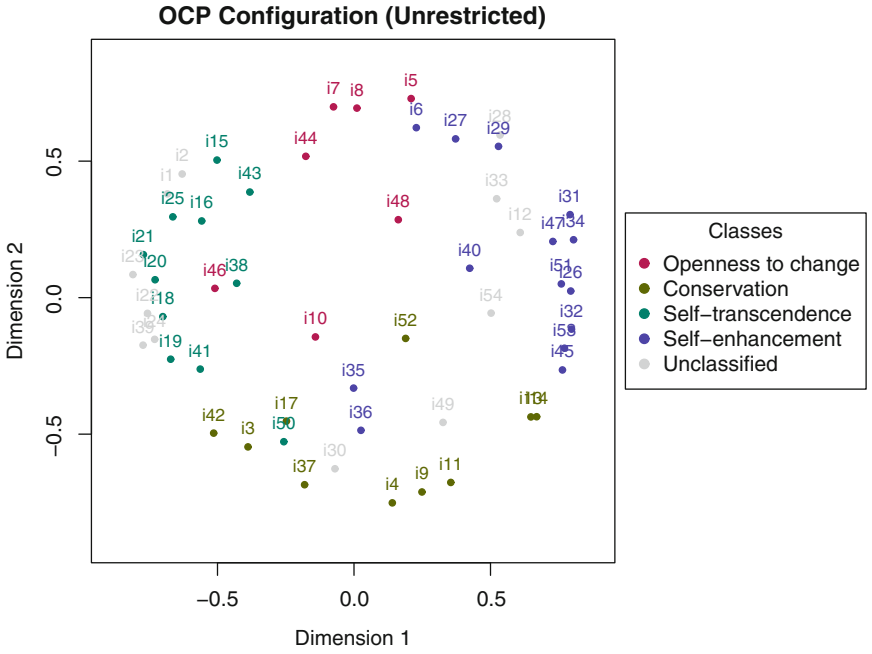

Fig. 9.7 OCP configuration: unrestricted MDS solution, objects colored according to OCP classes

The stress value for this unrestricted solution is 0.233. Not surprisingly, the unclassified items are scattered all over. For the classified items, we see that each group inhabits a certain region of the MDS space—with a few exceptions: in the "openness to change" class, items 10 and 46 are far off from the other items belonging to this group; in the "self-enhancement" class, MDS moved items 35 and 36 away from their friends.

In the next step, we fit a regionally constrained MDS solution by forcing the items belonging to the same class to inhabit the same region in the MDS space. The first six rows of **Z** are the following:

```
Z <- OCP[,56:57]
head(Z)
##   z1 z2
## 1 NA NA
## 2 NA NA
## 3  2  1
## 4  2  1
## 5  1  1
## 6  1  2
```

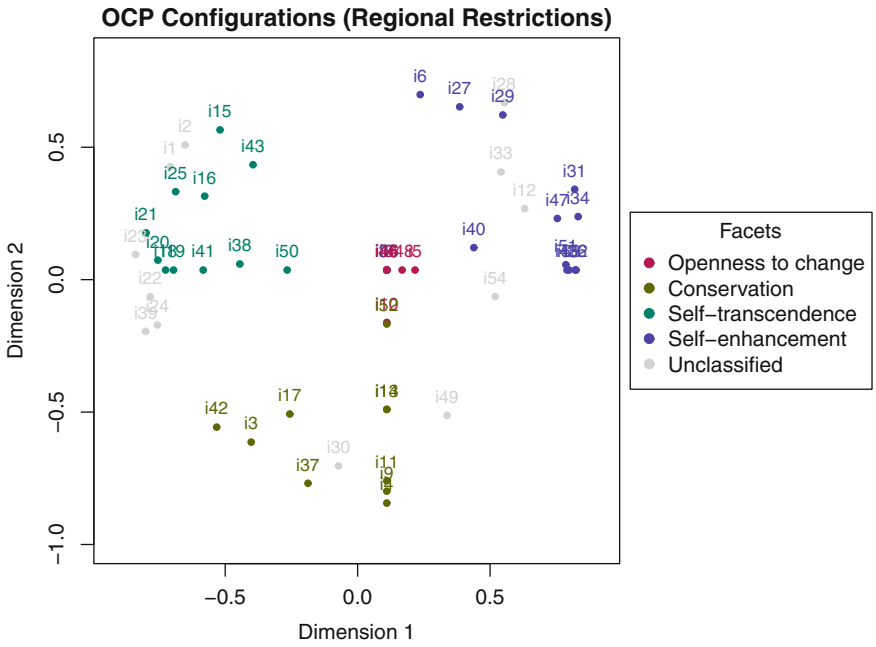

Fig. 9.8 OCP configuration: regionally restricted MDS (with OS on the external variables), objects colored according to OCP facets

We see, for example, that the first two items have no regional constraints, whereas items 3 and 4 belong to the same class. In addition to **Z**, we define the following setup for the `smacofConstraint` call. We use the final configuration of the first model as starting solution.[4] Through the `constraint` argument, we force the matrix **C** to be diagonal, and through `constraint.type`, we tell the function to perform a monotone transformation of the predictors in **Z** using the primary approach to ties (default).

```
fit.ocp2 <- smacofConstraint(ocpD, type = "ordinal",
                constraint = "diagonal", init = fit.ocp1$conf,
                external = Z,
                constraint.type = "ordinal")
```

This call leads to a stress-1 value of 0.318 which is considerably larger than the unrestricted stress value. The configuration plot is given in Fig. 9.8. The "openness

[4]Fitting an exploratory MDS first and then using the resulting configurations as starting values are in general a good strategy, unless we have a starting solution that is based on a theory.

to change" region is quite problematic in the sense that the region seems to be collapsed (degenerated). In addition, several points of the "conservation" class lie on the vertical border of the region. The unclassified items are nicely scattered in the space since we did not assign any restriction to them.

For this example we conclude that the regionally constrained MDS is too restrictive. Based on the exploratory MDS configuration, we could eliminate items that were far off from the remaining items in a class (e.g., `i46`, `i10`, `i35`, `i36`) and subsequently refit the restricted MDS without these items.

9.3.2 MDS with Internal Constraints: Spherical SMACOF

An alternative way to impose constraints on MDS configurations is through geometric restrictions. The most popular version involves spherical restrictions (Cox and Cox, 1991). That is, in a 2D space, we force the configurations to be on a circle, whereas in the 3D space, they lie on a sphere. In the SMACOF dictionary, this technique can be found under *spherical* SMACOF, other sources sometimes call it *weakly constrained MDS*. Examples of circular theories in psychology are Schwartz' circumplex theory of basic values (Schwartz, 1992) and color circles (Ekman, 1954). The concept of internally restricted MDS can be carried further to more complicated geometric shapes such as brain surfaces. Elad et al. (2005) present a large variety of MDS projections on non-sphere-like objects.

In this section we outline two algorithms for imposing spherical restrictions on MDS configuration. Technical details can be found in De Leeuw and Mair (2009). The first approach is called *primal method*. This algorithm incorporates the constraints directly into the stress loss function. Using this method, the objects are always mapped perfectly on a circle. The algorithm is considerably slow for large data settings, especially when it comes to ordinal versions of spherical MDS.

The second approach is called *dual method* which uses two auxiliary matrices $\mathbf{P}_1$ and $\mathbf{P}_2$: $\mathbf{P}_1$ contains the dissimilarities and $\mathbf{P}_2$ the restrictions. Correspondingly, the stress consists of two parts:

$$\sigma(\mathbf{X}; \mathbf{P}_1, \mathbf{P}_2) = \sigma(\mathbf{X}; \mathbf{P}_1) + \lambda\sigma(\mathbf{X}; \mathbf{P}_2). \tag{9.5}$$

The first term reflects the ordinary (unrestricted) MDS fit. The second term is responsible for the restrictions and includes the penalty term λ. The larger the λ, the harder we force the points to be perfectly on a circle, with the cost that the fit gets worse. For a low λ, the configuration might slightly deviate from the circle, but it gives us a better fit. Therefore, it might be the case that we have to play around with the penalty parameter in order to have satisfactory circle fit/goodness-of-fit trade-off.

As in MDS with external constraints, a good strategy in practice is to fit both an unrestricted solution and a restricted version and then compare the stress values. In this section we use the OCP data once more. In Fig. 9.7 we had an unrestricted,

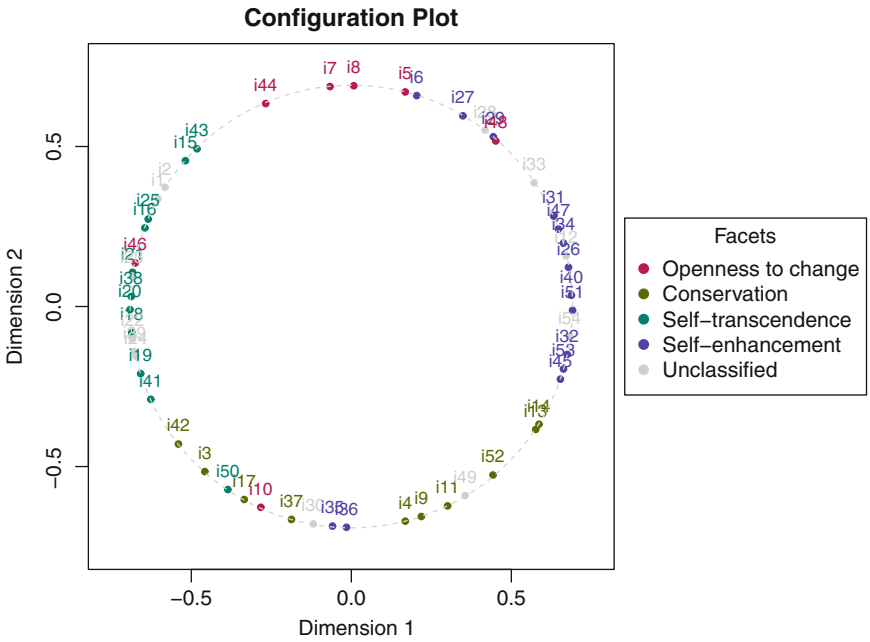

Fig. 9.9 OCP configurations: circular restriction

ordinal 2D solution. This plot suggests that many items are aligned approximately on a circle. Some points deviate from this circular structure and are shifted toward the center. Using a spherical MDS, we can force all the points to be on a circle. The distances between the objects can be interpreted by means of the distances on the circumference (i.e., *geodesic distances*).

Let us fit an ordinal, spherical 2D MDS on the OCP data. Remember that the stress of the ordinal, unrestricted MDS solution was 0.233. We use the dual method and increase the penalty term such that the points are exactly on a circle.

```
fit.ocp3 <- smacofSphere(ocpD, penalty = 1000, type = "ordinal")
```

The resulting stress value is 0.262 which is barely worse than the unrestricted stress. The corresponding configuration plot is given in Fig. 9.9. Items deviating from the circular structure (e.g., `i35` and `i36`) deserve closer attention. Of course, we could simply eliminate them and refit the model, but we could also aim to reformulate these items in a subsequent study. Note that goodness-of-fit assessment for both externally and internally constrained MDS models can be carried out in the same way as described in Sect. 9.2.3.

9.4 Unfolding

9.4.1 Data Structure for Unfolding

The first versions of *unfolding* models (i.e., *unidimensional unfolding*) can be traced back to Coombs (1950). In its basic form, unfolding is a model of preferential choice: n judges rank m stimuli according to their preference. As a simple toy example, let us consider five bands, ranked by three fans:

	Iron Maiden	Metallica	Slayer	Judas Priest	Megadeth
Horst	1	3	2	4	5
Klaus	1	2	3	4	5
Helga	1	5	4	2	3

A unidimensional unfolding fit leads to the configuration given in Fig. 9.10. First of all we see that both rows and columns of the data matrix are being scaled: individuals are represented by *ideal points*, objects by *object points*. This makes unfolding a *dual scaling* method. As we will see, in unfolding distances among row and column objects are defined, as opposed to correspondence analysis (CA; see Sect. 7.1.2 for a discussion regarding the interpretation). This property is valuable when it comes to interpretation.

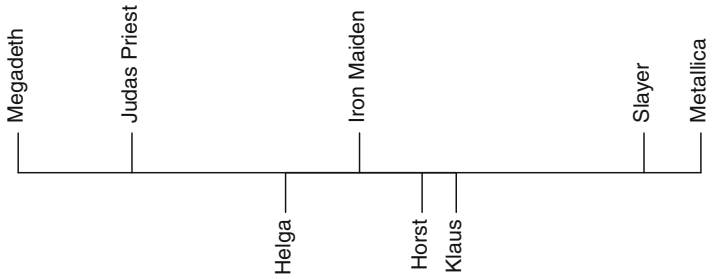

Fig. 9.10 Unidimensional unfolding for band rankings: bands and fans are located in a unidimensional space

In our example we see that Iron Maiden is everyone's favorite band. Thus, Iron Maiden is scaled as close as possible to all three raters. Horst and Klaus have a very similar taste, both like Slayer and Metallica. Klaus likes Metallica a bit more than Horst and, consequently, he is closer to them than Horst. Apart from Iron Maiden, Helga's taste is fairly different from the other two guys: she likes Judas Priest and Megadeth better than Metallica and Slayer. Thus, she is placed closer to these two bands.

Let us look at the data from a dissimilarity angle. For instance, we can say that Horst is closer to Slayer than to Metallica. Or Helga is closer to Megadeth than to Metallica. The fact that we are dealing with dissimilarities implies that unfolding is a variant of MDS and can be incorporated into the SMACOF framework.

This perspective suggests that unfolding is not only applicable to *rankings* but can also be applied to other dissimilarity settings such as *ratings*. We considered this distinction already in Sect. 5.2 within the context of preference modeling. Let us look at another toy example, this time with ratings:

	I1	I2	I3	I4	I5
Horst	1	5	3	1	1
Klaus	1	2	3	3	3
Helga	4	1	4	5	5

We have five items (5-point rating scale; with 1 ... "fully agree" and 5 ... "fully disagree"). For Horst this implies that he is close to item 1 and far away from item 2. Note that in order to apply unfolding, the data have to be dissimilarities, which is the case in our toy example. If the category labels would be 5 ... "fully agree" and 1 ... "fully disagree," the scores have to be reversed; otherwise they represent similarities. The unidimensional unfolding solution for the toy example is given in Fig. 9.11.

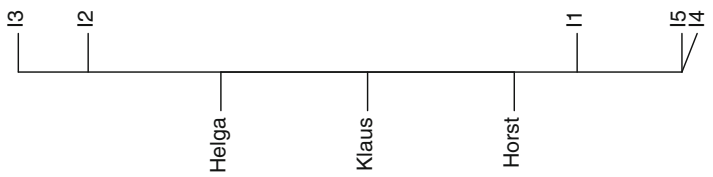

Fig. 9.11 Unidimensional unfolding for ratings: persons and items are located in a unidimensional space

The natural extension of unidimensional unfolding is multidimensional unfolding (Coombs, 1964; Mair et al., 2015) since in many practical applications we certainly need more than one dimension.

9.4.2 Rectangular SMACOF: Theory

In SMACOF slang, unfolding is called *rectangular SMACOF*. Our observed (rectangular) dissimilarity matrix $\boldsymbol{\Delta}$ is of dimension $n \times m$ with elements δ_{ij} ($i = 1, \ldots, n$ and $j = 1, \ldots, m$). The stress (cf. Eq. (9.2)) can be written as

$$\sigma(\mathbf{X}_1, \mathbf{X}_2) = \sum_{i=1}^{n} \sum_{j=1}^{m} (\hat{d}_{ij} - d_{ij}(\mathbf{X}_1, \mathbf{X}_2))^2. \tag{9.6}$$

The fitted Euclidean distances (p-dimensional space) become

$$d_{ij}(\mathbf{X}_1, \mathbf{X}_2) = \sqrt{\sum_{s=1}^{p} (x_{1is} - x_{2js})^2}. \tag{9.7}$$

Obviously, two configuration matrices are involved: the $n \times p$ matrix $\mathbf{X}_1$ containing the row configuration and the $m \times p$ matrix $\mathbf{X}_2$ with the column configuration. Both can be represented in the same MDS space. In terms of dissimilarity transformations $\hat{d}_{ij} = f(\delta_{ij})$, the same transformation functions (i.e., ratio, linear, ordinal) can be applied as in ordinary MDS. Ordinal unfolding is technically more complicated because a penalization terms needs to be included (Busing et al., 2005) in order to avoid a degenerate solution. In any case, the stress value can again be normalized to stress-1.

9.4.3 Unfolding Example: Personal Values

The following example uses a dataset analyzed in Borg et al. (2017). Their research question was whether the value circle exists within persons and not only across persons. The instrument they used to measure the values was the Schwartz Value Survey (SVS; Schwartz et al., 2000). In total, the dataset has 327 persons and 10 variables representing value scores (dissimilarities): power, achievement, hedonism, stimulation, self-direction, universalism, benevolence, tradition, conformity, and security. Using **smacof**, the unfolding model can be fitted as follows:

```
library("MPsychoR")
library("smacof")
fitSchwartz <- unfolding(indvalues, type = "interval")
```

We get a stress-1 value of 0.171. Goodness-of-fit evaluation of unfolding models can be performed in the same way as in ordinary MDS, as described in Mair et al. (2016).

Let us represent the solution by means of a *joint configuration plot* which plots the row and column scores into the same space. The distances between each pair of points can be interpreted. Note that in this application, we are not really interested in interpreting associations among individuals. Rather, we focus on the value scores and see whether they are approximately arranged on a circle. The resulting plot is given in Fig. 9.12, which also includes a least-squares-based circle fit on top of the column configuration.

```
plot(fitSchwartz, label.conf.rows = list(label = FALSE))
circ <- fitCircle(fitSchwartz$conf.col[,1],
                  fitSchwartz$conf.col[,2])
draw.circle(circ$cx, circ$cy, circ$radius, border = "gray",
            lty = 2)
```

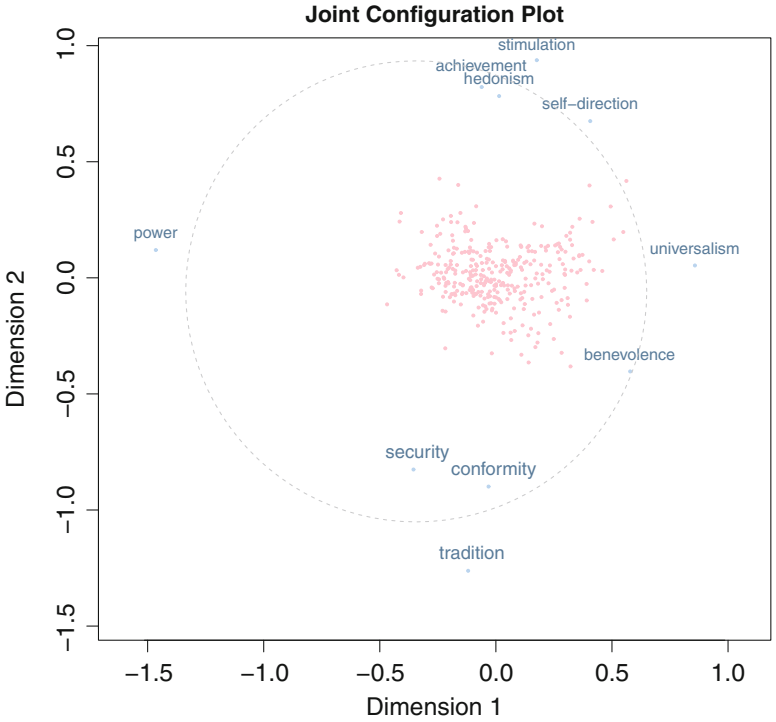

Fig. 9.12 Unfolding on individual values. On top of the column configuration a circle is drawn that shows the approximate circular alignment of the values

Borg et al. (2017) came to the conclusion that the value circle also exists within persons.

An interesting variant of this example would be to fit a spherically restricted unfolding model (*circular unfolding*) which forces the column configuration to be on a circle, similar to MDS with internal constraints. Subsequently, the stress values for both models can be compared. Using the `unfolding` function, this can be achieved by setting `circle="column"`.

Another unfolding extension is *row-conditional unfolding*. It abandons the assumption of homogeneous or consistent response scores across all individuals. For

instance, it is known from the literature that there are cultural scoring differences, especially when it comes to extreme responses (see, e.g., Hui and Triandis, 1989). In such cases, row-conditional unfolding can be applied. In the `unfolding` function, this can be achieved by setting `conditionality="row"`.

As mentioned above, different transformation functions can be considered in unfolding as well. In the corresponding function in **smacof**, the `type` argument allows for different specifications. For instance, using `type="ordinal"`, ordinal unfolding can be performed.

9.5 MDS Extensions and Related Models

In this section we discuss some other extensions and variants of MDS, such as Procrustes and models for individual differences scaling. Other approaches, worthwhile mentioning here since they are implemented in **smacof**, are the following: First, unidimensional scaling is an MDS where we project the objects on a single dimension. This approach is relevant for applications where we have a single underlying trait (e.g., easiness-difficulty continuum, time, etc.). A standard majorization solution suffers from the fact that we always end up in a local minimum. The **smacof** package offers a combinatorial solution to this problem which works reasonably fast for a small number of objects (Mair and De Leeuw, 2015).

Second, asymmetric MDS can be applied to scenarios where the dissimilarity between two objects is not symmetric (i.e., the directed dissimilarity from object 1 to object 2 is not the same as the one from object 2 to object 1). In this case one can apply the *drift vector* model, which produces a configuration plot that displays the symmetric part and the skew-symmetric part of the data as a field of arrows attached to these points. In **smacof** this can be achieved using the `driftVector` function; details can be found in Borg and Groenen (2005).

9.5.1 Procrustes

Sometimes it is of interest to compare two (or more) MDS configurations. For instance, we could fit one MDS model for the females in a dataset and a second one for the males and compare the configurations. Even though the solutions may actually be very similar, the configuration plots can look quite differently (e.g., because of different spatial orientations). Procrustes, named after "Procrustes the stretcher" in Greek mythology, deals with this problem and matches configurations. Note that Procrustes does not change the fit of an MDS model: it only performs rotations, dilations, and translations of the configuration.

Conceptually, it works as follows. One configuration $\mathbf{X}$ is considered to be the *target configuration*. It does not matter which one of the two we pick as target.

The second configuration $\mathbf{Y}$ is the *testee configuration* subject to three possible transformations: rotation with rotation matrix $\mathbf{T}$, dilation with dilation factor s, and translation with translation vector $\mathbf{t}$. The resulting Procrustes configuration $\hat{\mathbf{Y}}$ can be plotted into the same MDS space as $\mathbf{X}$, subject to visual inspection of configuration differences. In case of more than two MDS solutions (e.g., from different experimental conditions), each of them can be Procrustes transformed with respect to a target configuration.

As a general descriptive measure that quantifies the configurational similarity between the two configurations $\mathbf{X}$ and $\mathbf{Y}$, we can compute a *congruence coefficient* based on the configuration distances:

$$c(\mathbf{X}, \mathbf{Y}) = \frac{\sum_{i<j} d_{ij}(\mathbf{X}) d_{ij}(\mathbf{Y})}{\sqrt{\sum_{i<j} d_{ij}^2(\mathbf{X})} \sqrt{\sum_{i<j} d_{ij}^2(\mathbf{Y})}}. \tag{9.8}$$

The congruence coefficient becomes 1 if the configurations perfectly match and 0 if they are maximally dissimilar.

Note that correlating distances does not properly assess the similarity of configurations (see Borg and Groenen, 2005, p. 440). In addition, it is important to point out that the congruence coefficient as well as Procrustes can be applied to any point configuration in an Euclidean space and is therefore not limited to MDS.

To illustrate Procrustes, we use data derived from an fMRI experiment on goal-directed visual processing (Vaziri-Pashkam and Xu, 2017).[5] In this dataset the following eight objects were presented to participants: body (`BD`), cat (`CT`), chair (`CH`), car (`CR`), elephant (`EL`), face (`FA`), house (`HO`), and scissors (`SC`). The experiment involved three experimental conditions (color on objects and background, color on dots, color on objects), three brain regions of interest, and two tasks (color and shape).

Below we focus on the V1 (primary visual cortex) activation in the "color on objects and background" condition. We have two dissimilarity matrices[6]: one for the color task and one for the shape task. First, we fit an interval MDS on each of the two dissimilarity matrices separately. The left panel of Fig. 9.13 plots both configurations in the same space. They look seemingly different.

```
library("MPsychoR")
data("Pashkam")
fitcolor <- mds(Pashkam$color, type = "interval") ## color task
fitshape <- mds(Pashkam$shape, type = "interval") ## shape task
```

[5] In the fMRI area, MDS applications fall under the umbrella term *representational similarity analysis* (see Sect. 14.5).

[6] Details on how the dissimilarity matrices were assessed can be found in Vaziri-Pashkam and Xu (2017).

Now we apply Procrustes with the color configuration as target **X** and the shape configuration as testee **Y**, which will be transformed accordingly.

```
fitproc <- Procrustes(X = fitcolor$conf, Y = fitshape$conf)
fitproc
##
## Call: Procrustes(X = fitcolor$conf, Y = fitshape$conf)
##
## Congruence coefficient: 0.99
##
## Rotation matrix:
##       D1     D2
## D1 0.880  0.475
## D2 0.475 -0.880
##
## Translation vector: 0 0
## Dilation factor: 0.978
```

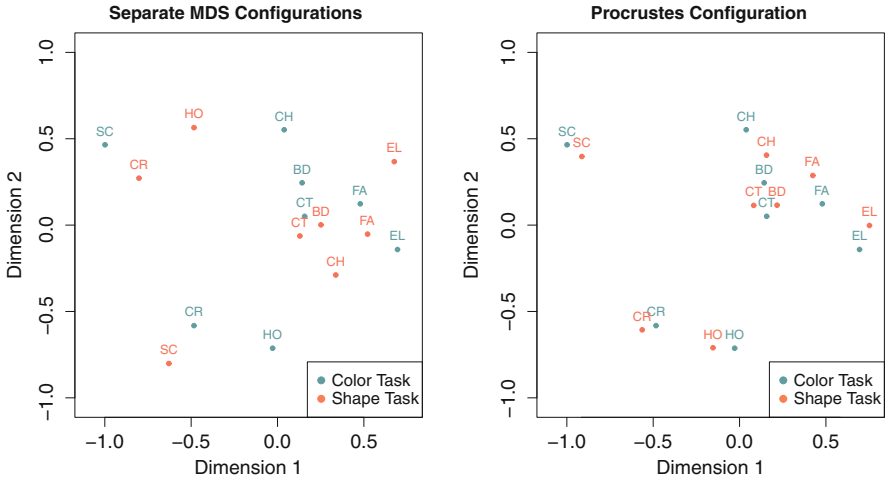

Fig. 9.13 V1 activation in the "color on objects and background" condition on eight objects. Left panel: Configurations based two separate MDS fits. Right panel: Procrustes transformed solution

The function returns the rotation matrix **T** and the dilation factor s. In MDS applications the translation vector **t** contains 0s due to normalized elements in the configuration matrix. We get a congruence coefficient of 0.99 which suggests that both configurations are highly similar. This is shown in the right panel of Fig. 9.13 where we see that all pairs of objects are located close to each other.

9.5.2 Individual Differences Scaling

Another important MDS extension involves multiple $n \times n$ input dissimilarity matrices $\boldsymbol{\Delta}_1, \boldsymbol{\Delta}_2, \ldots, \boldsymbol{\Delta}_K$. This set of matrices can be, for instance, based on dissimilarity ratings of K individuals. By considering the models we have seen so far, we can think of two approaches. First, a simple approach would be to fit K individual MDS models and apply Procrustes transformations to make the K configurations comparable. Second, another approach would involve averaging the dissimilarities across the K matrices and fitting an MDS on the resulting dissimilarity matrix. By doing this we obviously lose individual configuration information, and we assign the same weight to each individual.

A different MDS approach for such scenarios are models for *individual differences scaling*, sometimes also called *three-way MDS*. Note that they extend MDS in the same way as three-way PCA extends PCA (see Sect. 6.3). Three-way MDS models allow for large systematic differences among the $\boldsymbol{\Delta}_k$'s. They involve K individual configurations $\mathbf{X}_k$, and the joint configuration space $\mathbf{X}$, called *group stimulus space*. The most prominent three-way MDS model is called INDSCAL model (Carroll and Chang, 1970). INDSCAL assigns a different weight to each individual. Formally, it uses a weighted Euclidean distance on the individual configurations:

$$d_{ij}(\mathbf{X}_k) = \sqrt{\sum_{s=1}^{p} c_{sk}(x_{is} - x_{js})^2}, \tag{9.9}$$

where the weights c_{sk} can be interpreted as *salience* of dimension s for individual k (Borg et al., 2018). This equation implies that the individual spaces can be generated from a single group stimulus space by appropriate weighting of the dimensions. The basic stress equation extends to

$$\sigma(\mathbf{X}) = \sum_{k=1}^{K} \sum_{i<j} (\hat{d}_{ij,k} - d_{ij}(\mathbf{X}_k))^2. \tag{9.10}$$

We can plot the group stimulus space $\mathbf{X}$ as well as the individual spaces $\mathbf{X}_k$, in case we are interested in individual configurations. We can perform three-way MDS in an ordinal, ratio, and interval manner.

INDSCAL assumes common dimensions for each individual. This is a convenient property when it comes to interpretation. It implies that the dimensions are fixed and the group stimulus space should not be subject to rotations. Abandoning the assumption of common dimensions leads to another prominent model, called IDIOSCAL model (Carroll and Wish, 1974). For additional variants of three-way MDS models, see De Leeuw and Mair (2009).

The dataset we use for illustrating INDSCAL is from the cognitive neuroscience area and based on fMRI scans. Tamir et al. (2016) scanned 20 participants while

performing a task designed to elicit their thoughts about 60 mental states. On each trial, participants saw the name of a mental state (e.g., "awe") and decided which of two scenarios would better evoke that mental state in another person (e.g., "seeing the Pyramids" or "watching a meteor shower"). Based on these measures, they derived a 60 × 60 correlation matrix for each state, subsequently converted into a dissimilarity matrix. In total, we have 20 such dissimilarity matrices, one for each participant.[7]

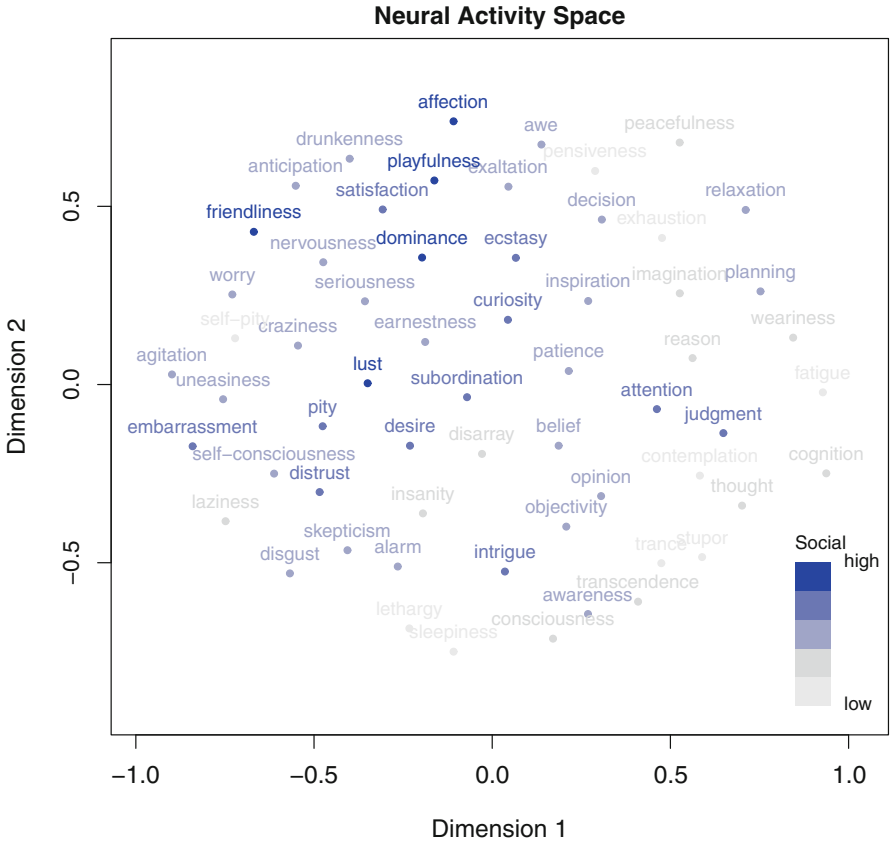

Fig. 9.14 Group stimulus space for neural activity data. Color gradient reflects magnitude on external social dimension

Let us fit an interval INDSCAL using **smacof** on a subset of these data (ten individuals only) since it takes a while to estimate the model on the full dataset.

[7] How these matrices are derived from the brain scans is described in detail in Sects. 14.2.4 and 14.5.

```
data("NeuralActivity")
fitNeuro <- indscal(NeuralActivity[1:10], type = "interval",
                    itmax = 5000)
```

The stress is 0.387. A thorough INDSCAL goodness-of-fit analysis would involve the usual steps as for any other MDS model.

The configuration plot (group stimulus space) is given in Fig. 9.14. As an additional plotting flavor, we incorporate external information (social dimension scores) in the configuration plot and color the points according to their scores on this scale. We see that states where interaction with other people is required (e.g., affection, friendliness, playfulness, dominance, lust) score highly on the social dimension. A more sophisticated way to include external variable information into an MDS solution is presented in Sect. 10.4, when we introduce MDS biplots. In case it is of interest, the individual configurations can be extracted using `fitNeuro$conf` and subject to plotting. This allows researchers to explore individual configuration differences. No need for Procrustes here, since this is done internally.

References

American Psychiatric Association. (1994). *Diagnostic and statistical manual of mental disorders (DSM-IV)* (4th ed.). Washington, DC: American Psychiatric Association.

Bilsky, W., & Jehn, K. A. (2002). Organisationskultur und individuelle Werte: Belege für eine gemeinsame Struktur [Organizational culture and individual values: Evidence for a common structure]. In M. Myrtek (Ed.), *Die Person im biologischen und sozialen Kontext [The person in biological and social context]* (pp. 211–228). Göttingen: Hogrefe.

Borg, I., & Groenen, P. J. F. (2005). *Modern multidimensional scaling: Theory and applications* (2nd ed.). New York: Springer.

Borg, I., & Lingoes, J. C. (1980). A model and algorithm for multidimensional scaling with external constraints on the distances. *Psychometrika, 45*, 25–38.

Borg, I., & Mair, P. (2017). The choice of initial configurations in multidimensional scaling: Local minima, fit, and interpretability. *Austrian Journal of Statistics, 46*, 19–32.

Borg, I., Groenen, P. J. F., Jehn, K. A., Bilsky, W., & Schwartz, S. H. (2011). Embedding the organizational culture profile into Schwartz's theory of universals in values. *Journal of Personnel Psychology, 10*, 1–12.

Borg, I., Bardi, A., & Schwartz, S. H. (2017). Does the value circle exist within persons or only across persons? *Journal of Personality, 85*, 151–162.

Borg, I., Groenen, P. J. F., & Mair, P. (2018). *Applied multidimensional scaling and unfolding* (2nd ed.). New York: Springer.

Busing, F. M. T. A., Groenen, P. J. F., & Heiser, W. J. (2005). Avoiding degeneracy in multidimensional unfolding by penalizing on the coefficient of variation. *Psychometrika, 70*, 71–98.

Carroll, J. D., & Chang, J. J. (1970). Analysis of individual differences in multidimensional scaling via an N-way generalization of Eckart-Young decomposition. *Psychometrika, 35*, 283–319.

Carroll, J. D., & Wish, M. (1974). Models and methods for three-way multidimensional scaling. In D. H. Krantz, R. C. Atkinson, R. D. Luce, & P. Suppes (Eds.), *Contemporary developments in mathematical psychology* (Vol. II, pp. 57–105). San Francisco: Freeman.

Cliff, N. (1973). Scaling. *Annual Review of Psychology, 24*, 473–506.
Coombs, C. H. (1950). Psychological scaling without a unit of measurement. *Psychological Review, 57*, 145–158.
Coombs, C. H. (1964). *A theory of data*. New York: Wiley.
Cox, T. F., & Cox, M. A. A. (1991). Multidimensional scaling on a sphere. *Communications in Statistics: Theory and Methods, 20*, 2943–2953.
Cox, T. F., & Cox, M. A. A. (2001). *Multidimensional scaling* (2nd ed.). Boca Raton: Chapman & Hall/CRC.
De Leeuw, J. (1977). Applications of convex analysis to multidimensional scaling. In J. Barra, F. Brodeau, G. Romier, & B. van Cutsem (Eds.), *Recent developments in statistics* (pp. 133–145). Amsterdam: North Holland Publishing Company.
De Leeuw, J., & Heiser, W. J. (1980). Multidimensional scaling with restrictions on the configuration. In P. R. Krishnaiah (Ed.), *Multivariate analysis* (Vol. V, pp. 501–522). Amsterdam: North Holland Publishing Company.
De Leeuw, J., & Mair, P. (2009). Multidimensional scaling using majorization: SMACOF in R. *Journal of Statistical Software, 31*(3), 1–30. http://www.jstatsoft.org/v31/i03/
De Leeuw, J., & Mair, P. (2015). Shepard diagram. In *Wiley statsRef: Statistics reference online*. New York: Wiley. https://onlinelibrary.wiley.com/doi/book/10.1002/9781118445112
De Leeuw, J., & Stoop, I. (1984). Upper bounds for Kruskal's stress. *Psychometrika, 49*, 391–402.
Ekman, G. (1954). Dimensions of color vision. *Journal of Psychology, 38*, 467–474.
Elad, A., Keller, Y., & Kimmel, R. (2005). Texture mapping via spherical multidimensional scaling. In R. Kimmel, N. Sochen, & J. Weickert (Eds.), *Scale space and PDE methods in computer vision* (Lecture notes in computer science, Vol. 3459, pp. 443–455). Berlin: Springer.
Gower, J. C., & Legendre, P. (1986). Metric and Euclidean properties of dissimilarity coefficients. *Journal of Classification, 3*, 5–48.
Heiser, W. J., & Meulman, J. (1983). Constrained multidimensional scaling, including confirmation. *Applied Psychological Measurement, 7*, 381–404.
Hui, C. H., & Triandis, C. H. (1989). Effects of culture and response format on extreme response style. *Journal of Cross-Cultural Psychology, 20*, 296–309.
Jacoby, W. G., & Armstrong, D. A. (2014). Bootstrap confidence regions for multidimensional scaling solutions. *American Journal of Political Science, 58*, 264–278.
Kruskal, J. B. (1964a). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika, 29*, 1–27.
Kruskal J. B. (1964b). Nonmetric multidimensional scaling: A numerical method. *Psychometrika, 29*, 115–129.
Mair, P., & De Leeuw, J. (2015). Unidimensional scaling. In *Wiley statsRef: Statistics reference online*. New York: Wiley.
Mair, P., Rusch, T., & Hornik, K. (2014). The grand old party – A party of values? SpringerPlus 3. http://www.springerplus.com/content/3/1/697
Mair, P., De Leeuw, J., & Wurzer, M. (2015). Multidimensional unfolding. In *Wiley statsRef: Statistics reference online*. New York: Wiley.
Mair, P., Borg, I., & Rusch, T. (2016). Goodness-of-fit assessment in multidimensional scaling and unfolding. *Multivariate Behavioral Research, 51*, 772–789.
McNally, R. J., Robinaugh, D. J., Wu, G. W. Y., Wang, L., Deserno, M. K., & Borsboom, D. (2015). Mental disorders as causal systems: A network approach to posttraumatic stress disorder. *Clinical Psychological Science, 3*, 836–849.
Meyer, D., & Buchta, C. (2015). **proxy**: Distance and similarity measures. R package version 0.4–15. http://CRAN.R-project.org/package=proxy
O'Reilly, C. A., Chatman, J. A., & Caldwell, D. F. (1991). People and organizational culture: A profile comparison approach to assessing person-organization fit. *Academy of Management Journal, 34*, 487–516.
Schwartz, S. H. (1992). Universals in the content and structure of values: Theoretical advances and empirical tests in 20 countries. *Advances in Experimental Social Psychology, 25*, 1–62.

Schwartz, S. H., Sagiv, L., & Boehnke, K. (2000). Worries and values. *Journal of Personality, 68*, 309–346.

Shepard, R. N. (1962a). The analysis of proximities: Multidimensional scaling with an unknown distance function I. *Psychometrika, 27*, 125–140.

Shepard, R. N. (1962b). The analysis of proximities: Multidimensional scaling with an unknown distance function II. *Psychometrika, 27*, 219–246.

Spence, I., & Ogilvie, J. C. (1973). A table of expected stress values for random rankings in nonmetric multidimensional scaling. *Multivariate Behavioral Research, 8*, 511–517.

Tamir, D. I., Thornton, M. A., Contreras, J. M., & Mitchell, J. P. (2016). Neural evidence that three dimensions organize mental state representation: Rationality, social impact, and valence. *Proceedings of the National Academy of Sciences of the United States of America, 113*, 194–199.

Torgerson, W. S. (1952). Multidimensional scaling: I. Theory and method. *Psychometrika, 17*, 401–419.

Vaziri-Pashkam, M., & Xu, Y. (2017). Goal-directed visual processing differentially impacts human ventral and dorsal visual representations. *The Journal of Neuroscience, 37*, 8767–8782.

Weathers, F. W., Litz, B. T., Herman, D. S., Huska, J. A., & Keane, T. M. (1993). *The PTSD checklist (PCL): Reliability, validity, and diagnostic utility*. Paper presented at the meeting of the International Society for Traumatic Stress Studies, San Antonio.

Chapter 10
Biplots

10.1 Variable Space and Subject Space Representation

Biplots were introduced Gabriel (1971) and are a powerful tool for visualizing complex, multivariate data settings. As an example we can think of modern psychological experiments where it is often of interest to display associations between neural and behavioral data.

An easy way to approach biplots is through the distinction between *variable space* and *subject space* representation (Reyment and Jöreskog, 1996; Borg and Staufenbiel, 2007). Simply speaking, we look at the person $\times$ variable matrix from two directions: column-wise and row-wise.

Let us start with the *variable space*. Everyone who has ever seen a scatterplot knows what a variable space representation is. To illustrate, we use the brain size/IQ data by Willerman et al. (1991), already considered in Sect. 6.1.2. In this study the authors collected a sample of 40 psychology students. The students took subtests of the Wechsler Adult Intelligence Scale-Revised (WAIS-R) which resulted in a full scale IQ (FSIQ), a verbal IQ (VIQ), and a performance IQ (PIQ). In addition to gender, body height, and body weight, the dataset includes an MRI pixel count variable which measures the brain size. For the moment, we focus on the two variables VIQ and PIQ only. A scatterplot of these two variables is given in the left panel of Fig. 10.1. The axes are determined by the two variables. Each participant gets a point in this 2D space corresponding to his/her IQ values.

For a simple *subject space* representation, the axes are determined by individuals (e.g., the first two participants). We would get a point for each variable, assuming that they are on the same scale. This Cartesian representation is not of any practical relevance since we typically have a large amount of subjects and the dimensionality of the space would be considerably high. However, we can apply the following trick: we get rid of the axes and represent each variable as vector. Such representations are often used in more technical books on regression and multivariate statistics (see, e.g., Wickens, 1995). We make use of the geometric idea that each variable can be

P. Mair, *Modern Psychometrics with R*, Use R!,
https://doi.org/10.1007/978-3-319-93177-7_10

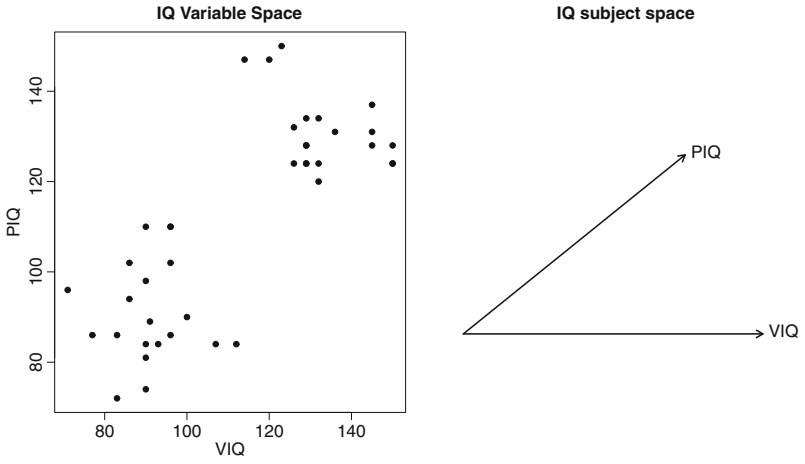

Fig. 10.1 Left panel: Variable space representation of verbal and performance IQ. Right panel: verbal IQ and performance IQ in subject space

represented as a vector: $\mathbf{x}$ of length $\|\mathbf{x}\|$ (Euclidean norm[1]) and $\mathbf{y}$ of length $\|\mathbf{y}\|$. This representation, as given in the right panel of Fig. 10.1, contains three points only: one point at the origin (variables should be centered), one point for $\mathbf{x}$ (determined by its length; VIQ in our example), and one point for $\mathbf{y}$ (determined by its length and the angle with respect to $\mathbf{x}$; PIQ in our example).

The steps to create such a plot are the following:

1. center the variables (let $\mathbf{x}$ and $\mathbf{y}$ denote the centered vectors);
2. compute the vector norms of the centered variables, i.e., $\|\mathbf{x}\|$, $\|\mathbf{y}\|$;
3. compute the angle θ between the vectors[2]: $\theta = \cos^{-1} \frac{\langle \mathbf{x}, \mathbf{y} \rangle}{\|\mathbf{x}\| \|\mathbf{y}\|}$;
4. plot the vectors in a 2D space.

As mentioned, we drop the usual Cartesian coordinate system for the moment since we do not have any dimensional reference system. We draw one vector horizontally and represent the other vector in relation to it.

The vectors in the subject space contain important information. The length of the vectors reflect the dispersion of each variable (precisely: $\sigma(\mathbf{x})\sqrt{n-1}$ with n as the number of observations). The angle θ between the vectors translates into a correlation:

$$r(\mathbf{x}, \mathbf{y}) = \cos(\theta). \tag{10.1}$$

[1] $\|\mathbf{x}\| = \sqrt{\sum_i x_i^2}$

[2] The inner product is $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_i x_i y_i$.

In our example, the length of the VIQ vector is 147.48, the length of the PIQ vector is 140.33, and the angle is 38.91°. Note that instead of just centering, we can also standardize, which changes the length of the vectors (they have the same length) but not the angle. It will be important for later applications that these vectors actually define new axes. That is, we can think of a line that extends a vector in both directions.

A key feature of the subject space representation is that we can add additional variables to this plot. The feasibility of the interpretation in terms of lengths and angles depends on the number of dimensions of the subject space: by projecting many variables into a 2D subject space, we might lose substantial structural information.

The geometric idea behind biplots is to merge variable space and subject space. That is, we represent the rows (e.g., persons) of a matrix and its columns (variables) in a single plot. This representation occurs within the context of a particular statistical model such as regression, principal component analysis (PCA; see Chap. 6), multidimensional scaling (MDS; see Chap. 9), correspondence analysis (CA; see Chap. 7), etc., as shown in the following sections. Note that in each section, we focus on two-dimensional biplots only; higher-dimensional biplots can be produced in an analogous section.

10.2 Regression Biplots

The starting point of a regression biplot is the predictor matrix $\mathbf{X}$ of dimension $n \times 2$. As mentioned above, we limit our explanations to two predictors only so that we can plot in a 2D space. The two variables $\mathbf{x}_1$ and $\mathbf{x}_2$ in $\mathbf{X}$ are standardized (i.e., we subtract the mean and divide by the sd). The second ingredient we need is a bunch response variables $\mathbf{y}_k$ ($k = 1, \ldots, K$); all of them standardized as well. This leads to the following multivariate set of regression equations (cf. Sect. 3.1), here expressed in terms of the fitted values:

$$\begin{aligned} \hat{\mathbf{y}}_1 &= \hat{\beta}_{11}\mathbf{x}_1 + \hat{\beta}_{12}\mathbf{x}_2 \\ \hat{\mathbf{y}}_2 &= \hat{\beta}_{21}\mathbf{x}_1 + \hat{\beta}_{22}\mathbf{x}_2 \\ &\vdots \\ \hat{\mathbf{y}}_K &= \hat{\beta}_{K1}\mathbf{x}_1 + \hat{\beta}_{K2}\mathbf{x}_2 \end{aligned} \tag{10.2}$$

Since all variables are standardized, there are no intercepts, and we get standardized slope estimates.

In a regression biplot, we aim to project the K response variables into the 2D scatterplot based on $\mathbf{x}_1$ on the x-axis and $\mathbf{x}_2$ on the y-axis. For the K responses, we need to find a subject space representation and map them as vectors on top of

the scatterplot. The β-parameter estimates are crucial: they are the bridge between the subject space and variable space since they reflect the coordinates of each $\mathbf{y}_k$ variable in the scatterplot. Thus, the necessary steps to produce a regression biplot are the following:

1. Standardize all variables involved in the analysis.
2. Produce the scatterplot based on $\mathbf{X}$.
3. Regress each response $\mathbf{y}_k$ on $\mathbf{X}$.
4. The standardized regression coefficients give the coordinates of the corresponding response vector.
5. Plot these vectors on top of the predictor scatterplot.

Obviously we plot multiple $\mathbf{y}_k$ variables in a 2D variable space, spanned by the variables in $\mathbf{X}$. This involves some loss of information. The R^2 resulting from the individual regressions can be used as a goodness-of-fit measure for each single $\mathbf{y}_k$ projection.

For illustration, we continue with the Willerman et al. (1991) data from above. We are interested in mapping two IQ variables (VIQ as $\mathbf{y}_1$ and PIQ as $\mathbf{y}_2$) and body weight ($\mathbf{y}_3$) into the scatterplot defined by body height ($\mathbf{x}_1$) and brain size ($\mathbf{x}_2$). First, we standardize all variables and compute the regressions which give us the set of standardized β-coefficients. We also extract the R^2 values right away since we need them later.

```
library("MPsychoR")
data("BrainIQ")
BrainIQ <- na.omit(BrainIQ[,-1])  ## we omit NAs and gender
rownames(BrainIQ) <- 1:nrow(BrainIQ)  ## relabel persons
BrainIQ1 <- as.data.frame(scale(BrainIQ)) ## standardize
regfit <- lm(cbind(VIQ, PIQ, Weight) ~
               -1 + Height + MRI_Count, data = BrainIQ1)
colnames(regfit$coef) <- c("VIQ", "PIQ", "Weight")
round(regfit$coef, 3)   ## vector coordinates
##              VIQ    PIQ Weight
## Height    -0.452 -0.482  0.608
## MRI_Count  0.566  0.662  0.156
R2vec <- sapply(summary(regfit), `[[`, "r.squared")
```

The regression biplot, as shown in Fig. 10.2, can be obtained by producing the scatterplot based on the standardized predictors in $\mathbf{X}$ (variable space) and by drawing arrows from the origin to the respective variable coordinate determined by the regression coefficients (subject space). For all biplots it is important to set the aspect ratio to 1 by means of `asp = 1`, such that distances between points are represented accurately on screen.

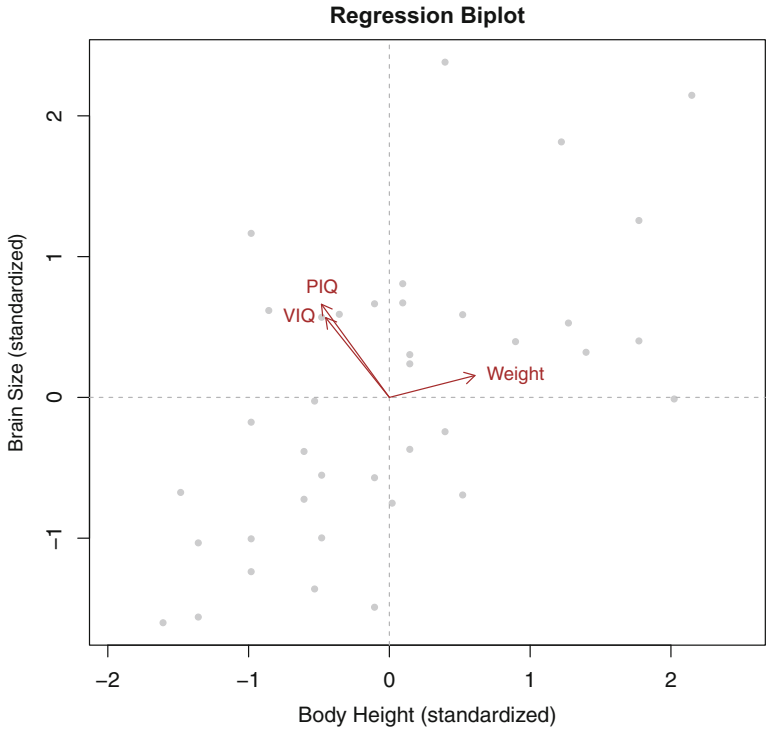

Fig. 10.2 Regression biplot for brain size—IQ data: VIQ, PIQ, and weight are regressed on brain size and body height

The information contained in this simple plot is already quite comprehensive. Let us start with the vectors. Each of them starts at the origin, and their direction is determined by the regression coefficients. Let us have a closer look at the body weight vector. It shows us that body height has a high impact on body weight, whereas brain size has a low impact on body weight. For both IQ variables, brain size has a positive impact, whereas body height is weakly related to them (in negative direction). Vectors that point in a similar direction correspond to variables that have similar response profiles (i.e., they are correlated with each other). Let us check this by computing the correlation matrix:

```
round(cor(BrainIQ[, 2:4]), 3)
##           VIQ   PIQ Weight
## VIQ     1.000 0.776 -0.076
## PIQ     0.776 1.000  0.003
## Weight -0.076 0.003  1.000
```

We see that VIQ and PIQ are highly correlated with each other; correspondingly their vectors point in a similar direction. Their correlation with weight is close to 0. Thus, both IQ vectors are almost perpendicular to the weight vector.

The vector length $\|\mathbf{y}_k\|$ of each response corresponds to the sum of the squared standardized regression coefficients. As extreme cases, we could have a variable which is weakly related to both predictors. The vector will be short. We could have another variable which is super strongly related with both predictors. Consequently, the vector will be long.

What we have seen in Fig. 10.2 is the *vector version* of a biplot in the tradition of Gabriel (1971) and Greenacre (2010). There is an alternative version based on *biplot axes*, which is advocated in Gower and Hand (1996) and Gower et al. (2011). We mentioned above that the vectors actually create new axes, extending the vectors in both directions. Using this concept we can project each single point in the scatterplot on these new axes. This gives us the possibility of a multivariate interpretation of each participant.

In Fig. 10.3, we show again the biplot, this time with the VIQ axis only, which extends the VIQ vector in Fig. 10.2 in both directions. The individuals are labeled with the corresponding row index in the data matrix, and each individual's point is subject to an orthogonal projection on the VIQ axis. The **calibrate** package (Graffelman, 2013) can be used to produce (i.e., calibrate) such biplot axes and, optionally, show the projection lines.

```
library("calibrate")
plot(BrainIQ1$Height, BrainIQ1$MRI_Count, pch = 20, cex = 0.8,
     xlab = "Height", ylab = "MRI", col = "darkblue", asp = 1,
     main = "Orthogonal Projections")
text(BrainIQ1$Height, BrainIQ1$MRI_Count, labels = 1:nrow(BrainIQ1),
     cex = 0.7, pos = 3, col = "darkblue")
abline(h = 0, v = 0, lty = 2, col = "darkgray")
calibrate.Z <- calibrate(regfit$coef[,1], BrainIQ1$VIQ, seq(-2,2, by = 0.5),
  cbind(BrainIQ1$Height, BrainIQ1$MRI_Count), dp = TRUE, axiscol = "brown",
  axislab = "VIQ", labpos = 3, verb = FALSE)
```

The projections correspond to the fitted values in the VIQ regression above. Note that these projections are on the standardized VIQ scale. We can scale the fitted values back to the original scale by "destandardizing," i.e., multiplying by the standard deviation (sd) and adding the mean:

```
VIQcal <- calibrate.Z$yt*sd(BrainIQ$VIQ) + mean(BrainIQ$VIQ)
```

Let us pick the two most extreme observations. Participant #20 is quite tall and has an average brain size, and, according to Fig. 10.3, he/she should have the lowest VIQ. Participant #11 is of average height and has the biggest brain, and the plot says that he/she should have the highest VIQ. Let us check this by looking at the original data:

```
summary(BrainIQ[, 2])   ## basic location measures
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   71.00   90.25  113.00  112.13  129.00  150.00
BrainIQ[c(20, 11), -c(1, 4)]  ## biplot projections
##    VIQ PIQ Height MRI_Count
## 20 107  84   76.5    905940
## 11 150 128   70.0   1079549
```

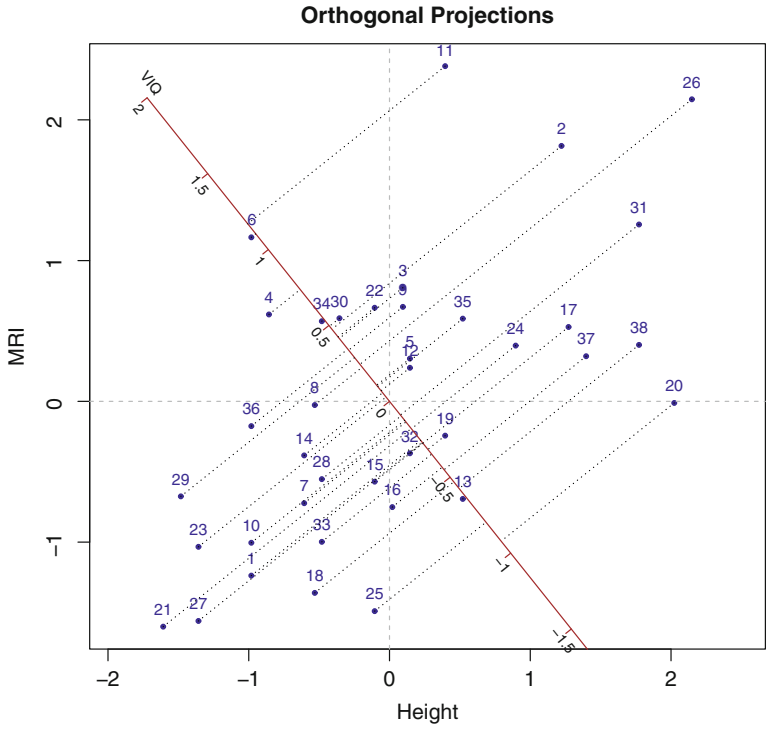

Fig. 10.3 Orthogonal projections on FSIQ axis

From this output we see that participant #11 has in fact the highest VIQ. The VIQ of participant #20 is between the first and second quartile, certainly not the lowest. Therefore, our fitted values are not entirely consistent with the observed values. Why is that? We have to keep in mind that we are losing information. This plot reflects the result of the regression model: if body height and brain size would perfectly explain the VIQ, there would be no discrepancies. Thus, the feasibility of the biplot interpretation depends on the goodness of fit of the regression model. The overall goodness of fit is given by the corresponding R^2. Let us print the R^2 values for all three responses considered in Fig. 10.2.

```
round(R2vec, 3)
##    Response VIQ    Response PIQ Response Weight
##           0.224           0.295           0.505
```

We see that the R^2 value for the VIQ regression is moderately low; our projections are only as good as the model. We could play the same projection game for the two remaining axes. For the weight axis, we would get a better fit since the R^2 value is larger, whereas for the PIQ axis, we would get a similar fit as for VIQ. In case of a perfect fit (i.e., $R^2 = 1$), there would be no such discrepancies.

How relevant are regression biplots in practice? In the first place, they are illustrative to show the biplot concept. Each biplot variant we present in the subsequent sections is based on the same principle. However, as presented here they are only of limited practical relevance since we considered two variables only. This said, we sometimes have a situation where we regress multiple responses on the same set of two or three predictors. In the case of three predictors, we end up with 3D scatterplots. Corresponding 3D illustrations are given in Greenacre (2010) and Gower et al. (2011).

For the next sections, it is important to keep in mind that the target variable space does not have to be a scatterplot based on observed variables defining the axes. It can be basically any variable space representation such as principal component (PC) scores in PCA, row or column points in CA, or an MDS configuration.

10.3 Principal Component Analysis Biplots

Based on what we have learned in the chapter on PCA (see Sect. 6.1), establishing a biplot is embarrassingly simple: the PCA output already provides us with all the necessary ingredients to produce a biplot (i.e., PC scores and loadings). In fact, the original, classical biplot by Gabriel (1971) was proposed within a PCA context and is nowadays considered as a standard graphical representation of PCA results.

One thing we can play around with in PCA biplots is the scaling factor α. This gives us the opportunity to produce biplots with different scaling properties. The trick is to rewrite the PCA singular value decomposition (SVD)[3] $\mathbf{X} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{V}'$ with $\mathbf{X}$ as the $n \times m$ data matrix (either standardized or not), $\mathbf{U}$ as $n \times m$ matrix containing the left singular vectors, $\mathbf{V}$ as $m \times m$ matrix containing the right singular vectors, and $\boldsymbol{\Lambda}$ as $m \times m$ diagonal matrix with the singular values $\lambda_1^{1/2}, \lambda_2^{1/2}, \ldots, \lambda_m^{1/2}$ in its main diagonal (the λ's are the eigenvalues):

$$\mathbf{X} = \mathbf{U}\boldsymbol{\Lambda}^{1-\alpha}\boldsymbol{\Lambda}^{\alpha}\mathbf{V}'. \tag{10.3}$$

[3]Note that we use $\boldsymbol{\Lambda}$ instead of $\mathbf{D}$ as in Sect. 6.1.1, in order to be consistent with the settings in R's `biplot` function.

Using this expression, the diagonal elements of $\boldsymbol{\Lambda}^{\alpha}$ are $\lambda_1^{\alpha/2}, \lambda_2^{\alpha/2}, \ldots, \lambda_m^{\alpha/2}$. Note that we will only produce biplots in $p = 2$ dimensions. Thus, the dimensions of the SVD matrices reduce according to $m = 2$ (i.e., we obtain a lower-rank approximation of $\mathbf{X}$). In Eq. (10.3), the subjects ($\mathbf{U}$ matrix) are scaled according to $\mathbf{G} = \mathbf{U}\boldsymbol{\Lambda}^{1-\alpha}$, and the variables ($\mathbf{V}$ matrix) are scaled according to $\mathbf{H} = \boldsymbol{\Lambda}^{\alpha}\mathbf{V}'$.

Let us have a closer look at the α parameter, which can take any value between 0 and 1. The choice of α affects the interpretation of the biplot. Two appealing α-choices are the following:

- *Row metric preserving* ($\alpha = 0$): the plot approximates the Euclidean distances among the persons in $\mathbf{X}$.
- *Column metric preserving* ($\alpha = 1$): the plot approximates the covariance structure of the variables in $\mathbf{X}$; the distances between the persons are determined by the Mahalanobis distance.[4]

Eventually, the choice of α depends on what we want to emphasize in the biplot: row metric preserving focuses on the persons and column metric preserving on the variables. The default choice in R's `biplot` function is $\alpha = 1$ (i.e., column metric preserving).

Let us put this principle into practice. First of all, PCA biplots are also a good opportunity to display the effect of standardized vs. unstandardized input variables in $\mathbf{X}$, as already discussed in Sect. 6.1.2. Using the brain size/IQ data from above once more, we see that the variable values are of different magnitudes, apart from the three IQ variables which are obviously on the same IQ scale.

```
library("MPsychoR")
data("BrainIQ")
BrainIQ1 <- na.omit(BrainIQ[,-1])
head(BrainIQ1, 3)
##   FSIQ VIQ PIQ Weight Height MRI_Count
## 1  133 132 124    118   64.5    816932
## 3  139 123 150    143   73.3   1038437
## 4  133 129 128    172   68.8    965353
```

The MRI count values are in the 1M magnitude, the IQ values are around 100, and weight and height are again on different scale magnitudes. By performing a PCA on these variables without standardizing, we can expect that the first PC will be totally dominated by the MRI count, something that is not very informative. Let us fit a PCA on the raw data and one on the standardized data:

[4]For our purposes it is sufficient to know that the Mahalanobis distance is similar to the Euclidean distance, but it incorporates weights in terms of the variance-covariance matrix (see, e.g., Joliffe, 2002, for details).

```
pca_biq1 <- prcomp(BrainIQ1)
pca_biq2 <- prcomp(BrainIQ1, scale = TRUE)
```

Using the `biplot` function, Fig. 10.4 can be produced. The unstandardized version is shown in the left panel and the standardized version in right panel. In both cases we keep the $\alpha = 1$ default. We see that the unstandardized solution is not desirable. The standardized solution gives a much more meaningful biplot since all variables are treated equally in terms of magnitude.

Let us consider a second example and then provide a full interpretation of what is actually shown in the biplot. This dataset, from the clinical area,[5] contains 30

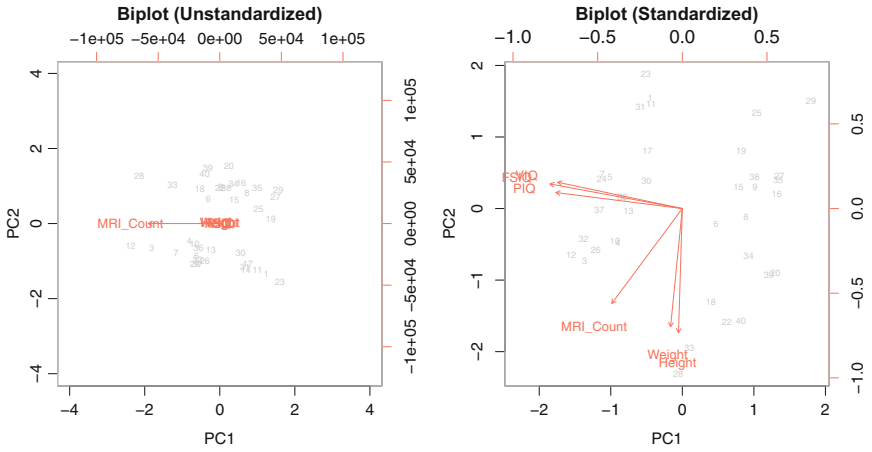

Fig. 10.4 Left panel: PCA biplot on raw brain size IQ data. Right panel: PCA biplot on standardized brain size IQ data

participants, of which 17 are of high-risk psychosis and 13 are healthy controls. We have three metric variables pertaining to behavioral measures: affective empathy (AE), positive social experience (PSE), and perspective taking (PT). Two additional measures come from fMRI scans (right-hand fRH and left/right foot fLRF). The variable scores are on the same scale: they are mean centered, but there are slight differences in the sd's. Let us fit once more two PCAs (unstandardized and standardized). Note that in the standardized version, we lose the sample variance information.

[5]Thanks to Christine Hooker for sharing this dataset.

```
data("yaass")
pca_yaass1 <- prcomp(yaass[,1:5])
pca_yaass2 <- prcomp(yaass[,1:5], scale = TRUE)
```

Out of these two fits, we produce four biplots using the `biplot` function once more (see Fig. 10.5): for each PCA version, we scale one biplot with $\alpha = 0$ (argument `scale = 0`) and the other one with $\alpha = 1$ (argument `scale = 1`).

First of all, the axes are determined by the first two PCs. The gray points are the PC scores of the participants, and the variables are represented by arrows. Participants that are close to each other have similar response profiles in $\mathbf{X}$. Both left panels are row metric preserving ($\alpha = 0$), meaning that they approximate the Euclidean distances among the participants. Both right panels are column metric preserving ($\alpha = 1$): they approximate the covariance (top right) and correlation (bottom right) structure, in addition to the Mahalanobis distances between persons. The arrows of variables covarying/correlating highly with each other should point into the same direction. For instance, for the bottom right panel, we can confirm this by looking at the correlation matrix:

```
round(cor(yaass[,1:5]), 2)
##       PSE   AE   PT  fRH fLRF
## PSE  1.00 0.88 0.64 0.43 0.45
## AE   0.88 1.00 0.92 0.41 0.30
## PT   0.64 0.92 1.00 0.20 0.07
## fRH  0.43 0.41 0.20 1.00 0.47
## fLRF 0.45 0.30 0.07 0.47 1.00
```

Another striking difference between the unstandardized and the standardized version is the vector length. Let us compute the sd's on the original data:

```
round(apply(yaass[,1:5], 2, sd), 3)
##   PSE    AE    PT   fRH  fLRF
## 0.701 0.758 0.710 0.437 0.421
```

The vector lengths in the column metric preserving, unstandardized plot (top right panel) reflect the sd's in a relative manner.

In the standardized version, the sd information is getting lost since all variables are scaled to $sd = 1$. For instance, in the bottom right panel, the variable vectors lengths are less than one. We could draw a unit circle; vectors close to this circle imply that these variables fit better. In this example we do not really have a short arrow since each variable loads highly on either PC1 or PC2. A two-dimensional

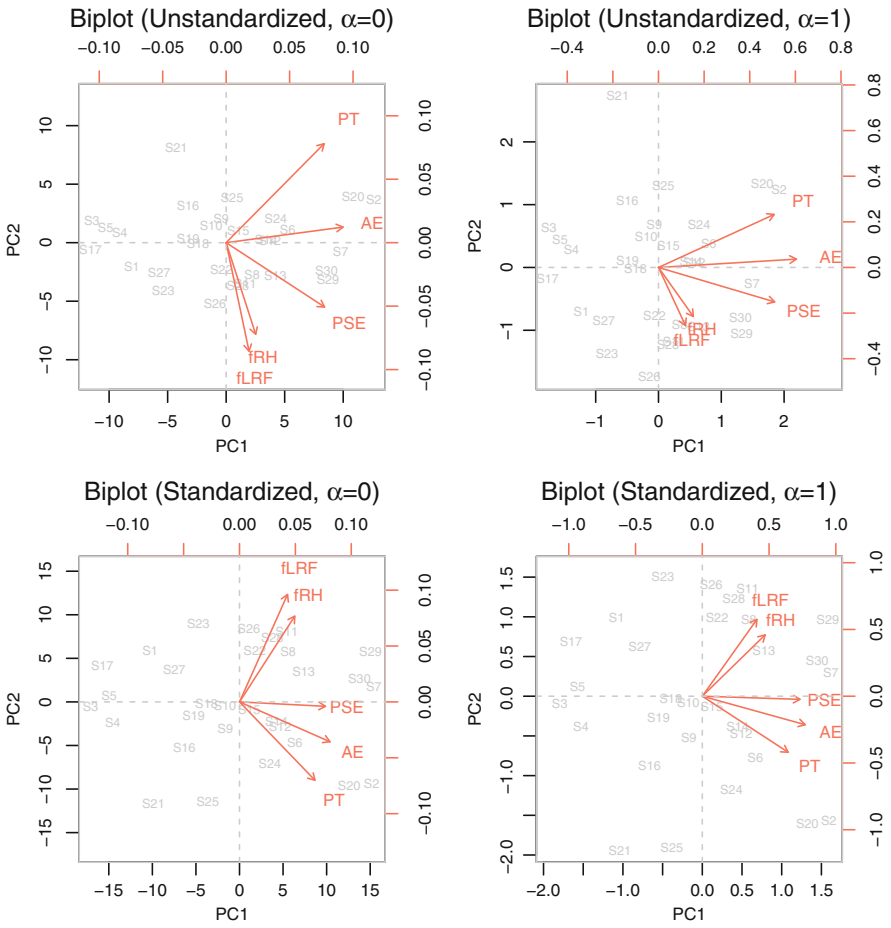

Fig. 10.5 Top panel: PCA biplot on raw data with different scale parameters (left one is row metric preserving and the right one column metric preserving). Bottom panel: PCA biplot on standardized data, again row vs. column metric preserving

solution is certainly sufficient here. However, we could have an additional variable which requires a third dimension (i.e., small loadings on PC1 and PC2, high loading on PC3). Thus, its vector in the 2D space would be pretty short. Note that the biplot panels include two more axes to the right and at the top. These are the units on the vectors, since each vector defines a new biplot axes.

Let us now illustrate that the principle of constructing a regression biplot holds for the PCA biplot as well. We also show how to create a biplot axis. PC1 and PC2 are our "predictors." We regress the five standardized variables onto the PC scatterplot.

```
X <- pca_yaass2$x[, 1:2]  ## extract PC scores
Y <- scale(yaass[,1:5])   ## standardize variables
fitlms <- lm(Y ~ -1 + X)  ## fit regressions
```

The regression coefficients should be the same as the loadings. Let us check:

```
round(coef(fitlms), 3)
##          PSE     AE     PT   fRH  fLRF
## XPC1   0.527  0.555  0.463 0.339 0.294
## XPC2 -0.027 -0.244 -0.480 0.525 0.658
round(t(pca_yaass2$rotation[,1:2]), 3)
##         PSE     AE     PT   fRH  fLRF
## PC1   0.527  0.555  0.463 0.339 0.294
## PC2 -0.027 -0.244 -0.480 0.525 0.658
```

We see that the regression biplot principle holds. This is not surprising since PCA consists of a set of linear combinations (see Eq. (6.3) in Sect. 6.1.2) solved by SVD. In order to produce the biplot axes, we establish the PC scatterplot by the hand and let the **calibrate** package do the rest. We draw an axis for affective empathy (AE) only. The projections are given in Fig. 10.6.

```
plot(X[,1], X[,2], pch = 20, xlab = "PC1", ylab = "PC2",
     col = "darkblue", asp = 1, main = "Biplot Axis",
     xlim = c(-3.2, 3.2))
text(X[,1], X[,2], labels = rownames(X), cex = 0.7,
     pos = 3, col = "darkblue")
abline(h = 0, v = 0, lty = 2, col = "gray")
calAE <- calibrate(fitlms$coef[,"AE"], Y[,"AE"],
  tm = seq(-2, 2, by = 0.5), Fr = X, dp = TRUE,
  axiscol = "brown", axislab = "AE", labpos = 3, verb = FALSE)
```

We extract the R^2 values and explore to which degree we can trust the projections on the AE axis.

```
R2vec <- sapply(summary(fitlms), "[[", "r.squared")
round(R2vec[2], 3)
## Response AE
##       0.999
```

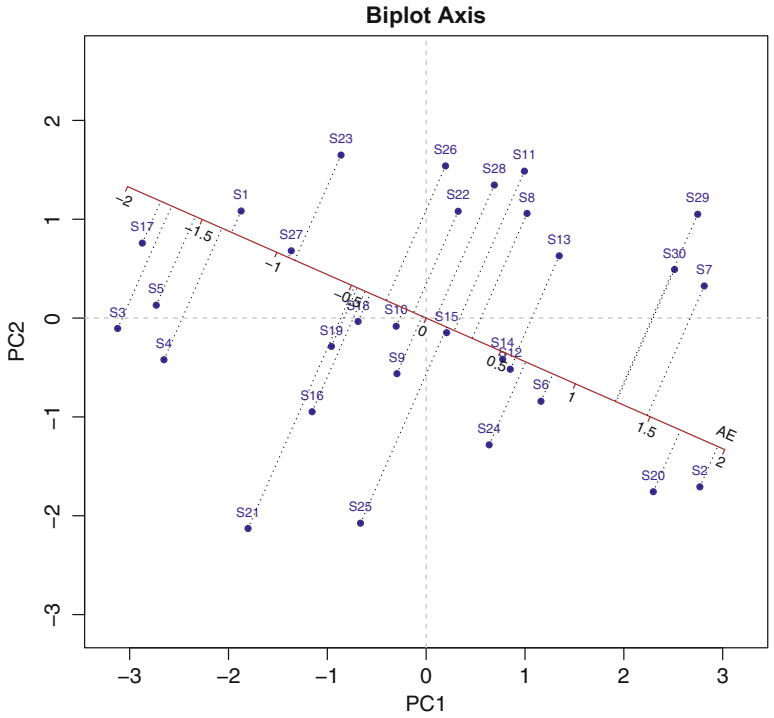

Fig. 10.6 Projections on affective empathy (AE) biplot axis

We see that the R^2 for AE is basically 1; therefore our projections have a super high accuracy. This implies that by means of these two PCs, variation in AE is fully explained.

Another package for PCA biplots is **bpca** (Faria et al., 2017) which offers some additional plotting options. We create once more a row metric preserving biplot on the unstandardized data and color the person points according to the group variable. The `bpca` function has the method argument options `"gh"` which corresponds to column metric preserving ($\alpha = 1$) and `"jk"` which means row metric preserving ($\alpha = 0$), plus some additional symmetric ones not discussed here.

Note that in biplots, it is not uncommon that the vector coordinates are much larger or smaller than the PC score coordinates. In such cases we can multiply the vector coordinates by a scaling factor in order to get a better picture. This is feasible since only the relative length of the variable vectors is important. In the biplot given in Fig. 10.7, we use a scaling factor (`var.factor` argument) of 3.

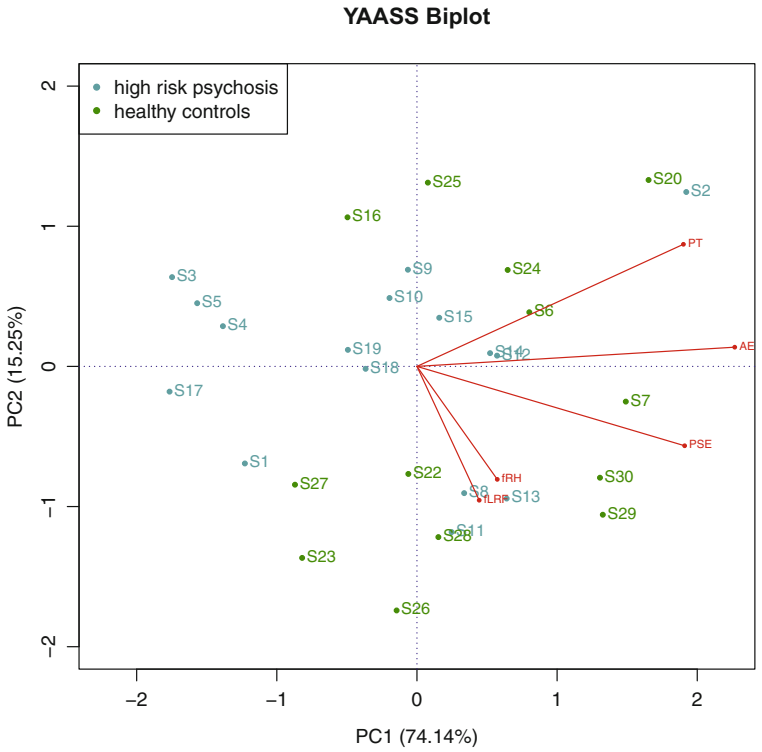

Fig. 10.7 PCA biplot (row metric preserving, data unstandardized) for behavioral and neural measures in YAASS dataset

```
library("bpca")
resbi <- bpca(yaass[, 1:5], scale = FALSE, method = c("gh"))
colvec <- c("cadetblue", "chartreuse4")[unclass(yaass$Group)]
plot(resbi, main = "YAASS Biplot", obj.color = colvec, var.factor = 3,
     obj.cex = 0.8, asp = 1, xlim = c(-2, 2), ylim = c(-2, 2))
legend("topleft", legend = c("high risk psychosis", "healthy controls"),
       pch = 20, col = unique(colvec))
```

To conclude the interpretation of this example: with two PCs, we explain 89.4% of the variance in the data. The first dimension reflects the behavioral measures and the second dimension the neural measures. The length of the vectors is proportional to their standard deviations.

The final PCA biplot version presented here is based on Princals. In Princals, as introduced in Sect. 8.2, biplots can be produced in the same way. We do not have to worry about any standardization issues since the Gifi framework performs optimal scaling anyway and the scores are normalized. Let us illustrate a Princals biplot using six items from a dataset on customer satisfaction (Kenett and Salini, 2012):

equipment, sales, technical support, training, purchasing support, and pricing. All these items are scored on a 5-point rating scale indicating the customer satisfaction level (from 1 . . . "very low" to 5 . . . "very high"). Let us fit an ordinal Princals using the **Gifi** package (Mair and De Leeuw, 2017):

```
library("Gifi")
ABC6 <- ABC[,6:11]
fitabc <- princals(ABC6)
```

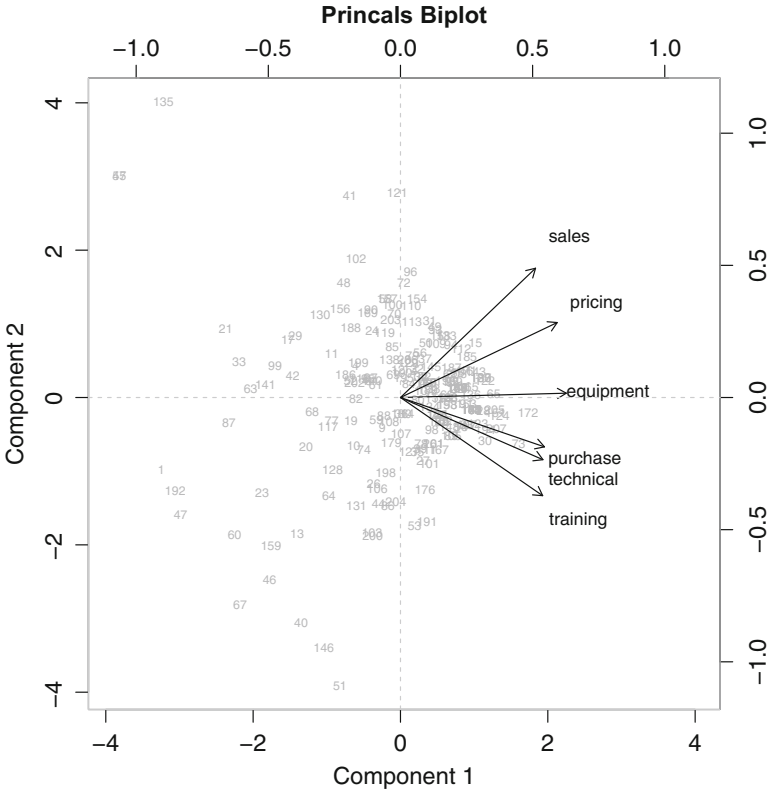

Fig. 10.8 Princals biplot for six items of ABC dataset: object scores (gray) and loadings (black arrows)

In the following plot function, we make use of the `expand` argument in order to shorten the vectors for better representation (Fig. 10.8).

```
plot(fitabc, plot.type = "biplot", main = "Princals Biplot",
     expand = 0.7, cex.scores = 0.6, col.scores = "gray")
abline(h = 0, v = 0, lty = 2, col = "gray")
```

The interpretation can be achieved in an analogous manner as in an ordinary PCA biplot, as presented above.

10.4 Multidimensional Scaling Biplots

In the regression biplot, we regressed responses on the predictor scatterplot, in PCA biplots the original variables on the PC scores scatterplot. We can do things in a similar way in MDS. The idea is to consider the configuration plot as "scatterplot" (variable space). The coordinates are given in the configuration matrix $\mathbf{X}$. We can now regress (external) variables on $\mathbf{X}$ as in Eq. (10.2). The **smacof** package (De Leeuw and Mair, 2009) provides the convenience function `biplotmds` which does the regression fit. The user only needs to provide the external variables.

To illustrate a MDS biplot, we use the mental states dataset from Tamir et al. (2016). Each individual is represented by a 60×60 dissimilarity matrix (60 mental states as objects), derived from fMRI scans. In Sect. 9.5.2 we fitted an INDSCAL solution on these data. This time we keep the MDS computation simpler and fit a single MDS on the averaged dissimilarities across individuals. The following code chunk does the data preparation and fits a 2D interval MDS using **smacof**.

```
library("MPsychoR")
library("smacof")
data("NeuralActivity")
delta <- Reduce("+", NeuralActivity)/length(NeuralActivity)
fit_neural <- mds(delta, type = "interval")
fit_neural
##
## Call:
## mds(delta = delta, type = "interval")
##
## Model: Symmetric SMACOF
## Number of objects: 60
## Stress-1 value: 0.281
## Number of iterations: 68
```

The external scales we use to produce a biplot are based on questionnaire data and contain proportions, telling us to which degree people associate each of the 60 mental states with 16 theoretical dimensions the authors extracted from the literature

(see Tamir et al., 2016, for details). We are now going to map these external variables on the MDS configuration. By default, the `biplotmds` function standardizes these variables and, therefore, returns the standardized regression coefficients (as well as the R^2 vector).

```
data("NeuralScales")
mdsbi <- biplotmds(fit_neural, NeuralScales)
```

The `plot(mdsbi)` call gives the MDS biplot in Fig. 10.9. The direction and length of the vectors is determined by the underlying regression parameters, just as in the regression biplot.

We can also do the projections using the **calibrate** package. Let us focus on the theoretical dimension "emotion," since this is the regression with the highest R^2 value (0.438).

```
X <- fit_neural$conf
Y <- scale(NeuralScales, scale = TRUE)
plot(X, pch = 20, cex = 0.6, xlab = "Dimension 1", ylab = "Dimension 2",
     col = "darkblue", asp = 1, main = "Biplot MDS Emotion Axis")
text(X, labels = rownames(X), cex = 0.7, pos = 3, col = "darkblue")
abline(h = 0, v = 0, lty = 2, col = "gray")
calEm <- calibrate(mdsbi$coef[,"Emotion"], Y[,"Emotion"],
  tm = seq(-2, 1.5, by = 0.5), Fr = X, dp = TRUE,
  axiscol = "brown", axislab = "Emotion", labpos = 3, verb = FALSE)
```

The corresponding biplot axis with projections is given in Fig. 10.10. States like laziness, agitation, disgust, and awe are the most important mental states determining the emotion dimension. Cognition is the least important state. In the same way, we can calibrate axes for other external variables of interest.

10.5 Correspondence Analysis Biplots

CA biplots are *asymmetric maps*. In Sect. 7.1.2 we mentioned we can standardize the scores in terms of *principal coordinates* (sometimes also referred to as *Benzecri coordinates*) or *standard coordinates* and showed how to compute them. In that section, we only used principal coordinates and produced a symmetric map. We have discussed in detail that we have to be careful with the interpretation of row-column associations since corresponding distances are not defined by the CA model. In order to properly interpret row-column distances, they need to be plotted in the same space, something we do not achieve using symmetric maps.

Asymmetric maps provide a way to plot row and column categories into the same space. For such an asymmetric map, we keep one dimension of the table in

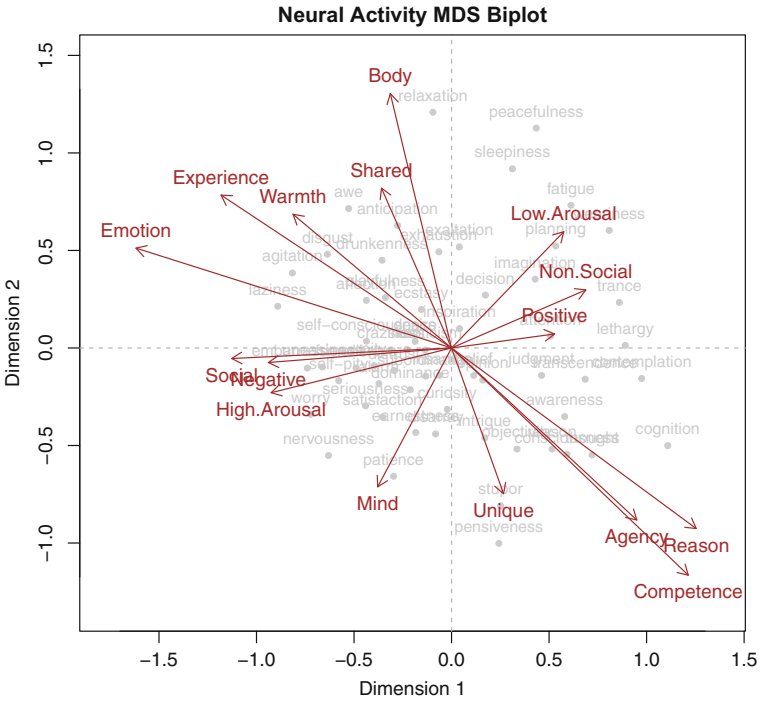

Fig. 10.9 MDS biplot for neural activity MDS fit (MDS configuration in gray). The length of each external variable vector is proportional to the R^2 value of the underlying regressions

standard coordinates and the other one in principal coordinates. Let us illustrate this concept using the superfan data from Sect. 7.1.1. We keep the rows (bands) in principal coordinates and the columns (fans) in standard coordinates, using the **anacor** package (De Leeuw and Mair, 2009).

```
library("anacor")
superfan <- as.table(matrix(c(9, 12, 8, 1, 13, 1, 6, 20, 15, 4, 23, 18),
                            ncol = 3))
attr(superfan, "dimnames") <- list(c("Slayer", "Iron Maiden", "Metallica",
  "Judas Priest"), c("Horst", "Helga", "Klaus"))
fit_fans <- anacor(superfan, scaling = c("Benzecri", "standard"))
plot(fit_fans, main = "Asymmetric Superfan CA Map")
```

Figure 10.11 nicely reproduces the ternary plot from Fig. 7.3, since the maximum number of dimensions in this setup is two. We plot the vertices of the fan simplex which define the most extreme band profiles possible. The distances between the fans are not reflected in this plot; we lose this information in an asymmetric map. However, the distance between, for instance, Iron Maiden and Horst is now defined and can be interpreted from the plot. The distances among the bands are defined

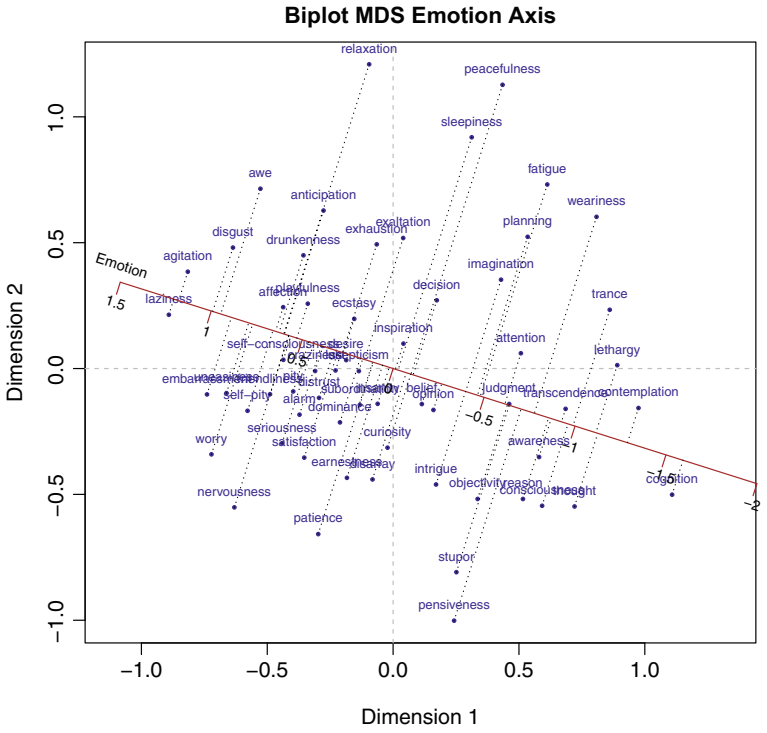

Fig. 10.10 MDS biplot axis for "emotion" with corresponding MDS object projections

as well and can be interpreted as in a symmetric map. In order to make this plot more coherent with the biplots presented so far, we could also draw arrows from the origin to the fan coordinates using the `arrows` argument in `plot.anacor`.

When should we use asymmetric maps? As Greenacre (2007) points out, asymmetric maps can be used if the table is to be interpreted in an asymmetric way. For instance, if the rows represent observational units (such as persons, groups, etc.) and the columns represent variables. In such scenarios we are probably most interested in interpreting distances among the rows and the distances among observational units and variables. Thus, we can use standard coordinates for the columns and principal coordinates for the rows.

Let us illustrate this strategy using a simple dataset from Srole et al. (1962), as presented in Weller and Romney (1990). The data consist of a cross-classification of social economic status (SES; six levels) and mental health (four categories).

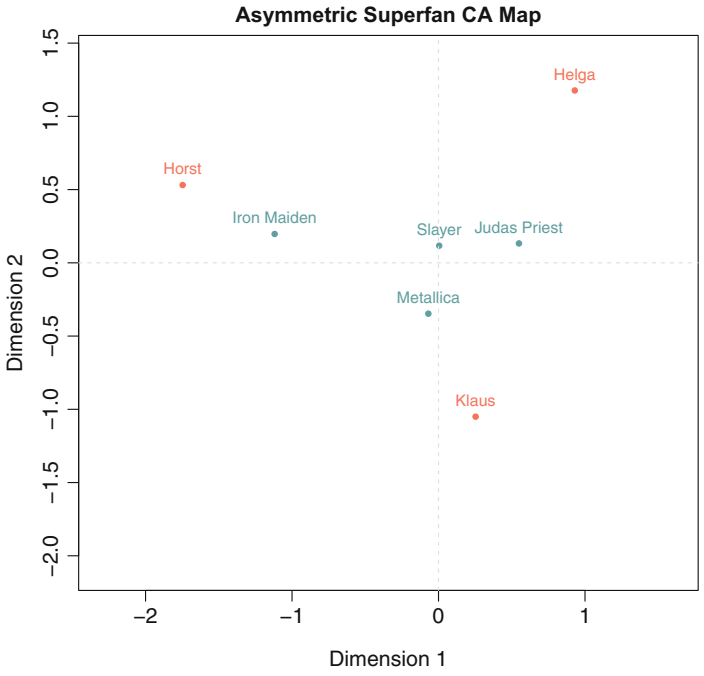

Fig. 10.11 Asymmetric CA map: bands in principal coordinates, fans in standard coordinates

```
Srole <- as.table(matrix(c(64, 94, 58, 46,
                           57, 94, 54, 40,
                           57, 105, 65, 60,
                           72, 141, 77, 94,
                           36, 97, 54, 78,
                           21, 71, 54, 71), nrow = 4))
attr(Srole, "dimnames") <- list(mhealth = c("well", "mild",
  "moderate", "impaired"), ses = LETTERS[1:6])
Srole
##           ses
## mhealth     A   B   C   D   E   F
##   well     64  57  57  72  36  21
##   mild     94  94 105 141  97  71
##   moderate 58  54  65  77  54  54
##   impaired 46  40  60  94  78  71
```

We are interested in how the SES categories are associated among themselves and in relation to the mental health categories. Thus, we keep the columns (SES) in principal coordinates, since they represent observational units, and put the rows in standard coordinates. The CA biplot based on the code chunk below is given in Fig. 10.12.

```
fit_ses <- anacor(Srole, scaling = c("standard", "Benzecri"))
plot(fit_ses, arrows = c(T, F), main = "Asymmetric CA Map")
```

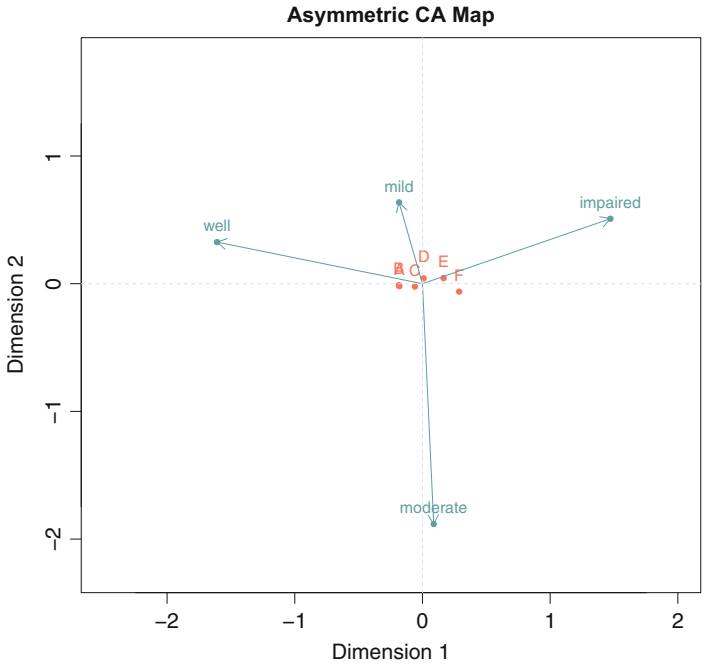

Fig. 10.12 Asymmetric CA map: SES in principal coordinates, mental health in standard coordinates

The mental health categories define the simplex (i.e., a tetrahedron plotted in 2D). However, we see a picture that occurs frequently in asymmetric CA maps. The set in principal coordinates are plotted as a tight cluster, whereas the standard coordinate categories are spread out. A strategy to fix this is called *contribution biplot*, where the vector coordinates are multiplied by the square roots of the corresponding masses. Such plots are implemented in the **ca** package (Nenadic and Greenacre, 2007).

Asymmetric CA maps are mostly used for simple CA. It is possible, however, to produce CA biplots for multiple CA as well, as described in Greenacre (2010), but they are less popular. The same applies to *Homals biplots* where we typically use symmetric maps, which Gifi calls *joint plot* (see Sect. 8.3.1).

This concludes the biplot chapter. Other types of biplots, not covered here, can be found in Greenacre (2010) and Gower et al. (2011).

References

Borg, I., & Staufenbiel, T. (2007). *Theorien und Methoden der Skalierung [Theory and methods of scaling]*. Bern: Verlag Hans Huber.

De Leeuw, J., & Mair, P. (2009). Multidimensional scaling using majorization: SMACOF in R. *Journal of Statistical Software, 31*(3), 1–30. http://www.jstatsoft.org/v31/i03/

Faria, J. C., Demétrio, C. G. B., & Allaman, I. B. (2017). **bpca**: Biplot of multivariate data based on principal components analysis. R package version 1.2.2. http://CRAN.R-project.org/package=bpca

Gabriel, K. R. (1971). The biplot graphical display of matrices with application to principal component analysis. *Biometrika, 58*, 453–457.

Gower, J. C., & Hand, D. J. (1996). Biplots. Boca Raton: Chapman & Hall/CRC.

Gower, J., Lubbe, S., & le Roux, N. (2011). *Understanding biplots*. Chichester: Wiley.

Graffelman, J. (2013). **calibrate**: Calibration of scatterplot and biplot axes. R package version 1.7.2. http://CRAN.R-project.org/package=calibrate

Greenacre, M. (2007). *Correspondence analysis in practice* (2nd ed.). Boca Raton: Chapman & Hall/CRC.

Greenacre, M. (2010). *Biplots in practice*. Bilbao: Fundación BBVA.

Joliffe, I. T. (2002). *Principal component analysis* (2nd ed.). New York: Springer.

Kenett, R. S., & Salini, S. (2012). *Modern analysis of customer surveys with applications in R*. New York: Wiley.

Mair, P., & De Leeuw, J. (2017). **Gifi**: Multivariate analysis with optimal scaling. R package version 0.3-2. https://R-Forge.R-project.org/projects/psychor/

Nenadic, O., & Greenacre, M. (2007). Correspondence analysis in R, with two- and three-dimensional graphics: The **ca** package. *Journal of Statistical Software, 20*(3), 1–13. http://www.jstatsoft.org/v20/i03/

Reyment, R. A., & Jöreskog, K. G. (1996). *Applied factor analysis in the natural sciences*. Cambridge: Cambridge University Press.

Srole, L., Langner, T. S., Michael, S. T., Opler, M. K., & Rennie, T. A. C. (1962). *Mental health in the metropolis: The midtown Manhattan study*. New York: McGraw-Hill.

Tamir, D. I., Thornton, M. A., Contreras, J. M., & Mitchell, J. P. (2016). Neural evidence that three dimensions organize mental state representation: Rationality, social impact, and valence. *Proceedings of the National Academy of Sciences of the United States of America, 113*, 194–199.

Weller, S. C., & Romney, A. K. (1990). *Metric scaling: Correspondence analysis*. Thousand Oaks: Sage.

Wickens, T. D. (1995). *The geometry of multivariate statistics*. New York: Psychology Press.

Willerman, L., Schultz, R., Rutledge, J. N., & Bigler, E. (1991). In vivo brain size and intelligence. *Intelligence, 15*, 223–228.

Chapter 11
Networks

11.1 Network Basics: Relational Data Structures

Networks have a long tradition in the area of sociology where researchers have been interested in analyzing relationships between members of social systems (e.g., friendship networks). Such classical *social networks* are based on *relational* input data, where actors such as persons interact with each other in some way.

As a toy example, let us consider our friends Horst, Helga, and Klaus, this time joined by Gertrud. In networks, each of the four actors will be represented by a *node* (or *vertex*). In this example, interactions occur when the actors trade albums. Let us assume that Helga and Horst traded some albums. In a network they are therefore connected by an *edge* (or *arc*). Helga gave Horst three albums, whereas Horst gave Helga two albums. We have one Helga → Horst edge and one Horst → Helga edge. The first edge gets an *edge weight* of 3, and the second edge gets a weight of 2.

The entire set of album trades can be represented by an *edge list*. Using the **igraph** package (Csárdi and Nepusz, 2006; Kolaczyk and Csárdi, 2014), we create the corresponding edge list (as `igraph` object) and assign the edge weights as follows:

```
library("igraph")
album_df <- matrix(c("Helga", "Gertrud", "Horst", "Klaus",
  "Horst", "Helga", "Gertrud", "Horst", "Klaus", "Horst",
  "Horst", "Helga", "Helga", "Klaus", "Klaus", "Klaus",
  "Gertrud", "Gertrud"), ncol = 2)
album_el <- graph.edgelist(album_df, directed = TRUE)
E(album_el)$weight = c(2, 3, 3, 4, 5, 1, 7, 7, 2)
```

A second, equivalent way of representing network data is through an *adjacency matrix*, where the rows reflect "from" and the columns reflect "to."

P. Mair, *Modern Psychometrics with R*, Use R!,
https://doi.org/10.1007/978-3-319-93177-7_11

```
album_ad <- get.adjacency(album_el, sparse = FALSE,
                          attr = "weight")
album_ad
##         Helga Horst Gertrud Klaus
## Helga       0     2       0     1
## Horst       3     0       7     5
## Gertrud     0     3       0     7
## Klaus       4     0       2     0
```

The corresponding *network graph* can be produced as follows:

```
plot(album_el, vertex.size = 0, edge.arrow.size = 0.5,
     vertex.label.dist = 0.8, vertex.color = "black",
     vertex.label.color = "black", vertex.label.cex = 1.5)
```

The graph in Fig. 11.1 represents a *directed* and *weighted* network since the edges have a "from-to" relation, and their values represent the connection strength. We could easily turn it into an *unweighted* network by replacing each of the edge values larger than 0 by 1 (i.e., trade vs. no trade). We could also change it into an *undirected* network where we abandon the "from-to" relation by taking the sum of corresponding elements in the upper and lower triangular part of the adjacency matrix, which would then become symmetric. As we see, there are four basic network types based on the combination of directed/undirected and weighted/unweighted edges.

11.2 Correlation Networks

In psychology we rarely collect data in a relational form. Similar to derived proximities in multidimensional scaling (MDS; see Sect. 9.1), we can compute the correlation matrix $\mathbf{R}$ out of a standard $n \times m$ person-variable data matrix. Such correlation networks are popular in the clinical area where they are used to detect associations among symptoms (see, e.g., Borsboom and Cramer, 2013).

In this section we illustrate correlation networks using a dataset from McNally et al. (2017). This dataset on comorbid obsessive compulsive disorder (OCD) and depression contains 16 depression items (QIDS; 5-point scale) and 10 OCD items (Y-BOCS; 4-point scale). There are $n = 408$ patients in the sample. As a first step, let us compute $\mathbf{R}$ using Pearson correlations:

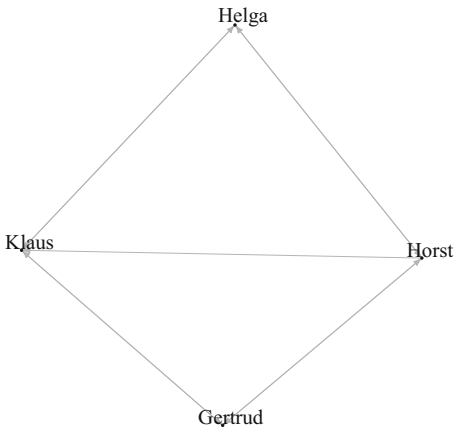

Fig. 11.1 Directed, weighted network plot for tool dataset on album trades

```
library("MPsychoR")
data(Rogers)
cormat <- cor(Rogers)
```

One issue that arises from using correlation matrices within a network context is that they typically lead to a fully connected graph, since empirical correlations are in general different from 0. Simple approaches to deal with this issue are blanking out low correlations by choosing a correlation threshold (e.g., using effect size standards), or keeping significant correlations only (after a proper p-value correction due to the multiple testing problem). For the network plot in Fig. 11.2, produced using the **qgraph** package (Epskamp et al., 2012), we set the lower absolute correlation threshold to 0.2 (i.e., small-medium effect size). The thicker an edge, the higher the correlation. Edges with negative correlations are colored differently.

```
library("qgraph")
cornet <- qgraph(cormat, layout = "spring", minimum = 0.2,
  graph = "cor", groups = list(Depression = 1:16, OCD = 17:26),
  color = c("white", "gray"), labels = colnames(Rogers),
  title = "Depression/OCD Correlation Network")
```

From Fig. 11.2 we see that there are two densely connected groups of items: all OCD items and a bunch depression items except those in the periphery related to sleep and appetite. Based on such a correlation network, we have the option to compute various typical network measures to characterize each symptom's role in the network. One such network characteristic is called *centrality*. The

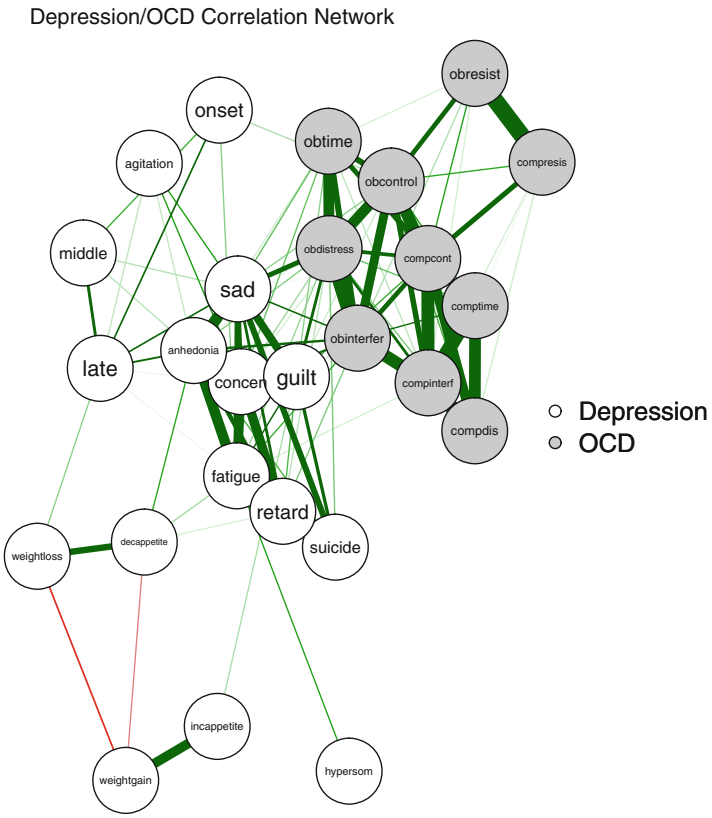

Fig. 11.2 Correlation network for depression and OCD items. Green lines reflect positive correlations and red lines negative correlations

qgraph package provides the following options, implemented in the `centrality` function:

- *Degree centrality* (strength): number of edges connected with a node.
- *Closeness centrality*: how close one node is to all the other nodes based on the shortest paths.
- *Betweenness centrality*: Interactions between nodes depend on the other nodes who lie on the path between them (i.e., they "control" the interactions).

A corresponding plot can be produced as follows:

```
centralityPlot(cornet)
```

Figure 11.3 shows the z-standardized centrality values. Betweenness centrality is an interesting property to look at in this example. Items that score high on this

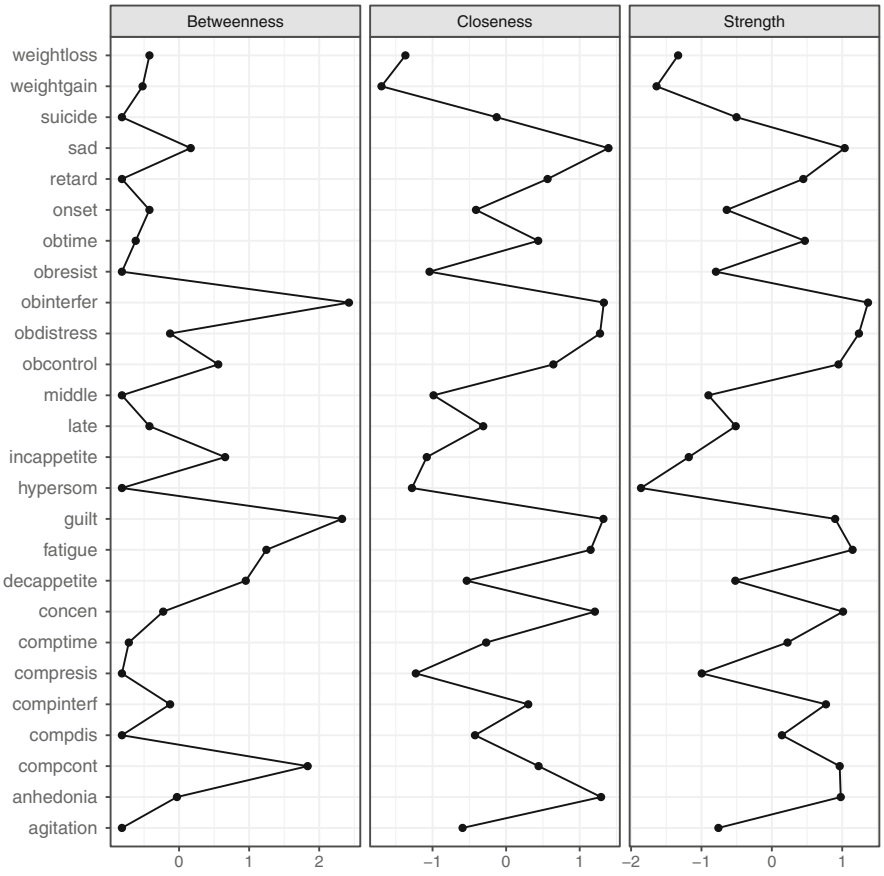

Fig. 11.3 Centrality measures (z-standardized) of depression and OCD symptoms

measure are potentially connector items between OCD and depression. Details on other network measures can be found in Wasserman and Faust (1994) and Kolaczyk and Csárdi (2014).

Let us extend this simple correlation network as follows. First, instead of using simple Pearson correlations, we use partial correlations. A partial correlation involving two nodes controls for the influence of all the remaining nodes in the network. Thus, the inferred edges are more reflective of direct influence among nodes. Second, instead of using a somewhat arbitrary correlation threshold or a Bonferroni corrected significance level, a *graphical lasso* (Friedman et al., 2008) can be considered.

The lasso principle (see James et al., 2013, for a gentle introduction) is highly attractive within the context of correlation networks since such networks typically have many edges. The idea of the graphical lasso is to shrink low correlations (i.e., edge weights) to 0 such that they disappear from the graph. Let $\mathbf{S}$ be the sample variance-covariance matrix and λ the penalization parameter. The graphical lasso target function to be maximized is

$$\sigma(\mathbf{K}) = \log(\det \mathbf{K}) - \text{tr}(\mathbf{SK}) - \lambda \sum_{i,j} |k_{ij}|. \tag{11.1}$$

$\mathbf{K}$ is a *precision matrix* (i.e., the inverse of the model variance-covariance matrix) with elements k_{ij} to be estimated. From these elements we can compute the *regularized partial correlations*:

$$\rho_{ij}^{(\text{partial})} = -\frac{k_{ij}}{\sqrt{k_{ii}}\sqrt{k_{jj}}}. \tag{11.2}$$

The **qgraph** package provides an easy-to-use, high-level implementation for regularized partial correlations which builds on the **glasso** package (Friedman et al., 2014). Internally it establishes a grid of penalization parameters λ (the higher λ, the higher the number of 0 edge weights) and picks the best λ according to an extended BIC criterion (see `EBICglasso` function). The graphical lasso computation including the plot in Fig. 11.4 can be produced as follows:

```
qgraph(cormat, layout = "spring", sampleSize = nrow(Rogers),
       graph = "glasso",
       groups = list(Depression = 1:16, OCD = 17:26),
       color = c("white", "gray"), labels = colnames(Rogers),
       title = "Depression/OCD Graphical Lasso")
```

Note that, unlike in MDS, the distance between nodes in network plots such as in Figs. 11.2 and 11.4 has no meaning. The nodes are positioned such that the graph is produced in an aesthetically pleasing way.[1] Therefore, scaling approaches such as MDS are highly illustrative within the context of correlation networks since they position the nodes according to their similarity (i.e., high edge values in network slang). The next section shows how to integrate scaling and dimension reduction into networks. The resulting node positions can be subject to further interpretation.

[1] This is of course a bit of an oversimplification; more details on graph layouts such as Fruchterman-Reingold and Kamada-Kawai can be found in Kolaczyk and Csárdi (2014).

11.3 Latent Network Models

Before we start elaborating on latent network models, it is important to point out that here the term "latent" is not used in a strict psychometric sense as in factor analysis, where an underlying latent variable causes the indicators (e.g., a general depression factor causes the symptoms). Within the network context, "latent" simply implies that we perform a dimension reduction in the sense of a principal component analysis (PCA; see Chap. 6). Based on the node positions in a low-dimensional space, we might be able to interpret the corresponding dimensions.

In the following subsections, we present two approaches, both belonging to the general class of *exponential random graph models* (ERGMs). ERGMs are designed in analogy to generalized linear models (GLMs) and are typically estimated using Markov Chain Monte Carlo (MCMC; see Hoff et al., 2002; Kolaczyk and Csárdi,

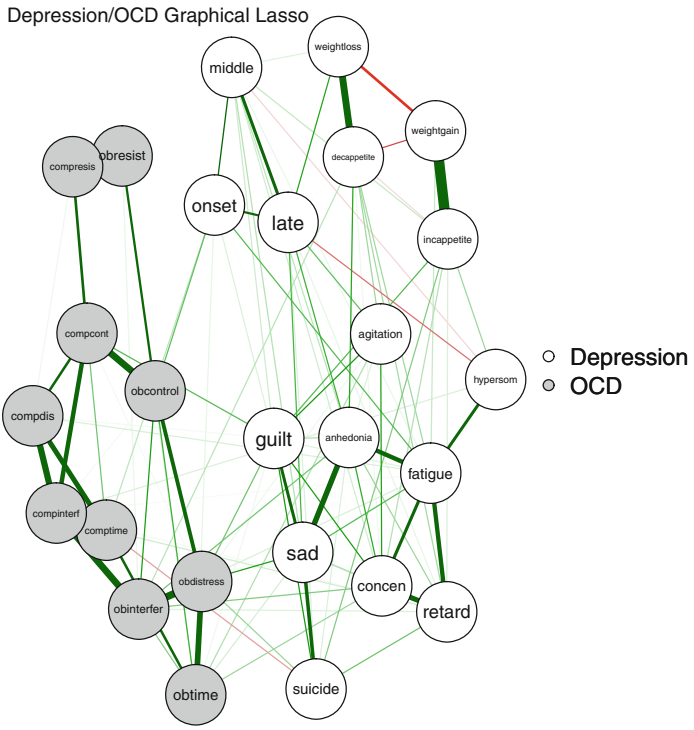

Fig. 11.4 Partial correlation graphical lasso network for depression and OCD items. Green lines reflect positive partial correlations and red lines negative partial correlations

2014). The first type of latent networks we are going to present are *eigenmodels*. Subsequently, they will be extended in terms of incorporating node clustering.

11.3.1 Eigenmodels

Eigenmodels were proposed by Hoff (2008). The input data consist of an $n \times n$ adjacency matrix $\mathbf{Y}$ with elements Y_{ij} $(i, j = 1, \ldots, n)$. Y_{ij} is defined as a function of variables $\mathbf{u}_i$ and $\mathbf{u}_j$ and, optionally, a regression model with pair-specific predictor vector $\mathbf{x}_{ij}$. The number of dimensions p needs to be fixed a priori. The general eigenmodel expression is as follows:

$$Y_{ij} = f(\boldsymbol{\beta}' \mathbf{x}_{ij} + \mathbf{u}_i \boldsymbol{\Lambda} \mathbf{u}_j). \tag{11.3}$$

By using MCMC, we estimate the matrix $\hat{\boldsymbol{\Lambda}}$ of dimension $p \times p$ which gives the relative importance of each dimension and the matrix $\hat{\mathbf{U}}$ of dimension $n \times p$. In this matrix, each node i gets a row vector $\hat{\mathbf{u}}_i = (\hat{u}_{i1}, \ldots, \hat{u}_{ip})$ containing the unobserved node characteristics. Nodes with similar characteristics will get similar $\hat{\mathbf{u}}$-vectors. The regression parameter estimates are collected in the $\boldsymbol{\beta}$-vector.

There is one more tweak to be applied on this solution. The vectors $\hat{\mathbf{u}}_i$ are not orthogonal, a property which is important for interpretation and plotting. Applying an eigenvalue decomposition on the fitted matrix $\hat{\mathbf{U}}\hat{\boldsymbol{\Lambda}}\hat{\mathbf{U}}'$ does the trick. The resulting eigenvectors reflect the coordinates in the p-dimensional space and can be subject to plotting.

Let us apply this concept to the OCD/depression data using the **eigenmodel** package (Hoff, 2012). As input adjacencies we use the correlation matrix. There is no need here to blank out low correlations. We fit a $p = 2$ dimensional model and use 1000 MCMC iterations with a burn-in of 200 steps.[2]

```
library("eigenmodel")
diag(cormat) <- NA   ## NA diagonals required
fitEM <- eigenmodel_mcmc(cormat, R = 2, S = 1000,
                         burn = 200, seed = 123)
```

Let us look at the posterior means of the eigenvalues first.

```
evals <- colMeans(fitEM$L_postsamp)
evals
## [1] 0.2590746 0.6107728
```

[2]For running time purposes we keep the burn-in and the number of iterations low. In practice they should be higher.

Unlike in PCA, the eigenvalues are not of decreasing order since they were estimated using MCMC. We see that the second eigenvalue dominates the solution. Thus, the second latent variable (or dimension) is relatively more important than the first one. Now we apply the additional eigenvalue decomposition on $\hat{\mathbf{U}}\hat{\boldsymbol{\Lambda}}\hat{\mathbf{U}}'$ and plot the solution. For plotting, we set a correlation threshold of 0.2 in order to blank out weak connections.

```
evecs <- eigen(fitEM$ULU_postmean)$vec[, 1:2]
cols <- c("coral", "cadetblue")
plot(evecs, type = "n", xlab = "Dimension 1",
  ylab = "Dimension 2", xlim = c(-0.30, 0),
  main = "Depression/OCD Eigenmodel Network")
corthresh <- 0.2                    ## correlation threshold
addlines(evecs, abs(cormat) > corthresh, col = "gray")
ind <- c(rep(1, 16), rep(2, 10))
text(evecs, labels = rownames(cormat), col = cols[ind],
  cex = 0.8)
legend("topright", legend = c("Depression", "OCD"),
       col = cols, pch = 19)
```

Figure 11.5 gives the resulting plot. We see the clear separation between OCD and depression itemsets along dimension 2. This confirms what the eigenvalue above told us: the second dimension is the more important one. Dimension 1 mostly discriminates between the "core" depression items and the "periphery" depression items, as already identified in the correlation networks above.

One variation of the eigenmodel fitted above is the inclusion of predictors as formalized in Eq. (11.3). To explore the predictor effects visually, we can fit an eigenmodel with predictors and one without predictors and plot the two networks. The dimensionality of an eigenmodel (with or without predictors) can also be varied, and goodness of fit can be judged using cross-validation. Corresponding examples are given in Kolaczyk and Csárdi (2014) and Hoff (2008).

11.3.2 Latent Class Network Models

So far we have assumed that we know which items are associated with depression scale and which ones are OCD items. Let us pretend for a moment that we do not know this, and let a network algorithm try to detect corresponding node clusters. Latent class network models integrate the idea of *parametric clustering* (see Chap. 12) into network analysis.

Formally, this model represents an extension of the eigenmodel approach from above since, in addition to scaling, it also estimates the class membership of each node in a probabilistic way. A thorough description of the algorithm can be found

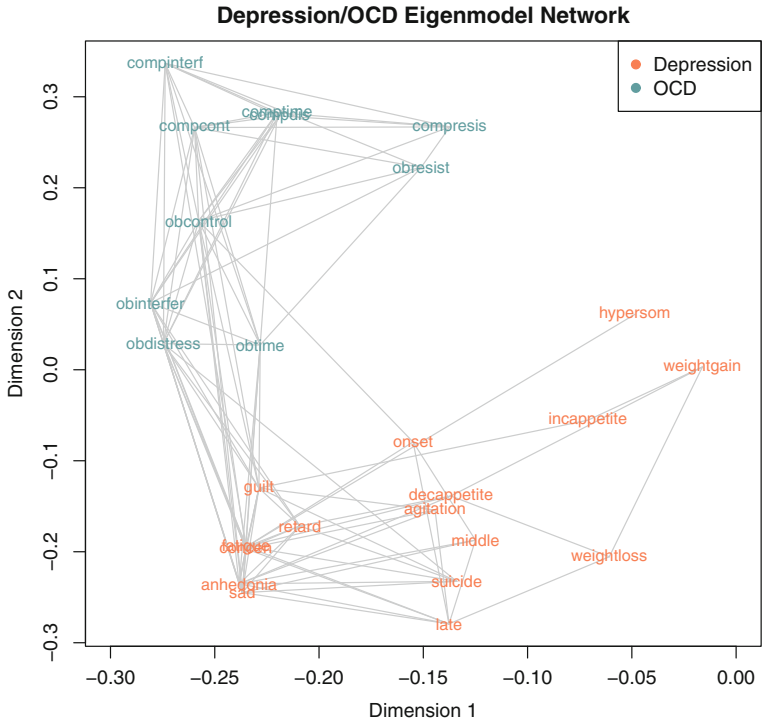

Fig. 11.5 Eigenmodel network plot for depression and OCD items

in Hoff et al. (2002), and the relation to the eigenmodel is explained in Hoff (2008). Latent class network models can be fitted using the **latentnet** package (Krivitsky and Handcock, 2008) and belong to the ERGM model class as well.

Note that the package does not support fully connected graphs, as implied by a correlation matrix. We fit a very simple model based on a dichotomized correlation matrix. By using a correlation threshold of 0.2, absolute correlations below 0.2 are set to 0, whereas the remaining ones are set to 1.

```
thresh <- 0.2
cormat01 <- ifelse(abs(cormat) > thresh, 1, 0)
```

In the next data preparation step, we need to convert this adjacency matrix into a network object (undirected graph), using the **network** package (Butts, 2008).

```
library("network")
cornet <- network(cormat01, matrix.type = "adjacency",
                  directed = FALSE)
```

This object acts as input to the `ergmm()` function. Let us fix the number of dimensions to $p = 2$ (argument `d`) and vary the number of classes G from 1 to 3. We keep the MCMC settings at the function's default values. For each model we extract the BIC for goodness-of-fit assessment. Note that, similar to a GLM, latent class networks allow for different distributional family specifications. Since in our example we operate on a binary input matrix, we use the Bernoulli family (default).

```
library("latentnet")
set.seed(111)
fitLN1 <- ergmm(cornet ~ euclidean(d = 2, G = 1))
summary(fitLN1)$bic$Z
## [1] 247.6176
fitLN2 <- ergmm(cornet ~ euclidean(d = 2, G = 2))
summary(fitLN2)$bic$Z
## [1] 237.839
fitLN3 <- ergmm(cornet ~ euclidean(d = 2, G = 3))
summary(fitLN3)$bic$Z
## [1] 237.5217
```

According to the BIC, the two- and three-cluster solutions fit equally well. Let us focus on the two-cluster solution first. The corresponding network plot involving the posterior means as node coordinates is given in Fig. 11.6, produced using `plot(fitLN2)`. We see that dimension 2 discriminates between the set of OCD items and the set of depression items. The circles give the soft cluster boundary.[3] These circles should not be interpreted in a deterministic way since, as in parametric clustering in general, the latent class network algorithm estimates probabilistic class memberships. Let us extract the posterior probabilities of three selected symptoms:

```
clusmemb2 <- fitLN2$mkl$mbc$Z.pZK
dimnames(clusmemb2) <- list(colnames(cormat01),
                            paste("Cluster", 1:2))
clusmemb2[c("comptime", "suicide", "weightgain"), ]
##            Cluster 1 Cluster 2
```

(continued)

[3]To be more precise, their radius is equal to the square root of the posterior intra-cluster variance estimates.

```
## comptime      0.9620    0.0380
## suicide       0.0255    0.9745
## weightgain    0.0015    0.9985
```

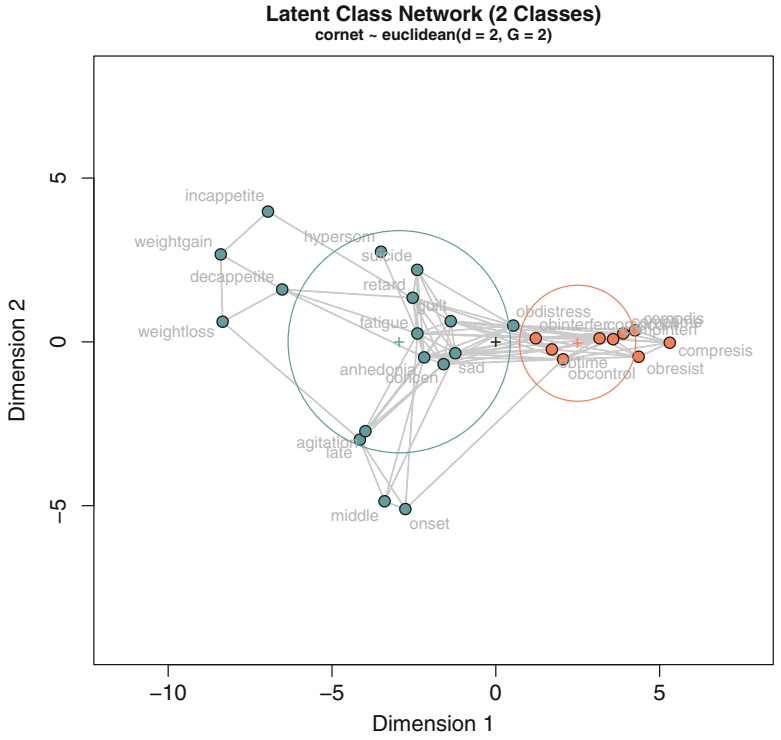

Fig. 11.6 Latent class network solution in two dimensions and with two clusters. Cluster 1 (red) can be labeled as "OCD" and cluster 2 (blue) as "depression"

We see that `comptime` has a high probability to belong to cluster 1, whereas `suicide` and `weightgain` have high probabilities to belong to cluster 2. From the posterior probabilities of the remaining items and in conjunction with Fig. 11.6, we can conclude that cluster 1 is a tight OCD cluster, and cluster 2 a wider depression cluster. Therefore, the algorithm successfully associated the symptoms with the respective disorder (except for `obdistress`).

Now let us examine the three-cluster solution in more detail, as shown in Fig. 11.7 and produced using `plot(fitLN3)`. We see that the depression cluster from the two-cluster solution is split up into two overlapping clusters: a wide, general depression cluster (golden) and a depression subcluster (red), closer to

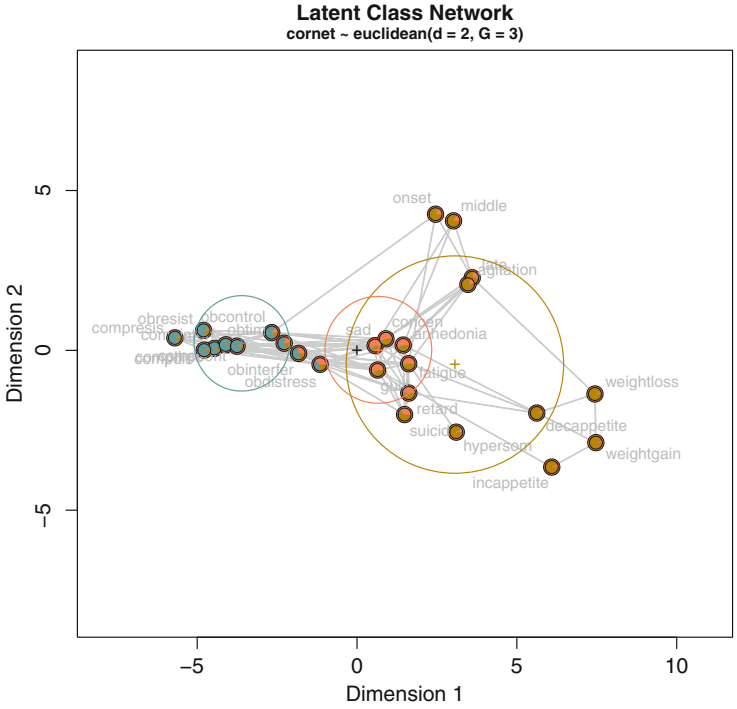

Fig. 11.7 Latent class network solution in two dimensions and with three clusters. Cluster 1 (red) is a subcluster of a general depression cluster (cluster 3; golden) and cluster 2 (blue) is the OCD cluster

the OCD cluster (blue). As in Fig. 11.6, this plot also includes information on the posterior class memberships by means of a pie chart representation of the nodes. For instance, `obdistress` has approximately equal probabilities to belong to cluster 1, 2, or 3.

As an alternative representation, we can also plot the posterior densities of the clusters as given in Fig. 11.8. This plot gives additional information on the cluster separation by means of plotting the cluster densities in separate panels.

```
plot(fitLN3, main = "Latent Class Network",
     cluster.col = c("coral", "cadetblue", "darkgoldenrod"),
     what = "density")
```

As mentioned above, the **latentnet** package allows for Gaussian and other distribution networks where are not required to operate on the binary input adjacency matrix but rather on the edge weights reflected in the correlations. Other options are described in Krivitsky and Handcock (2008).

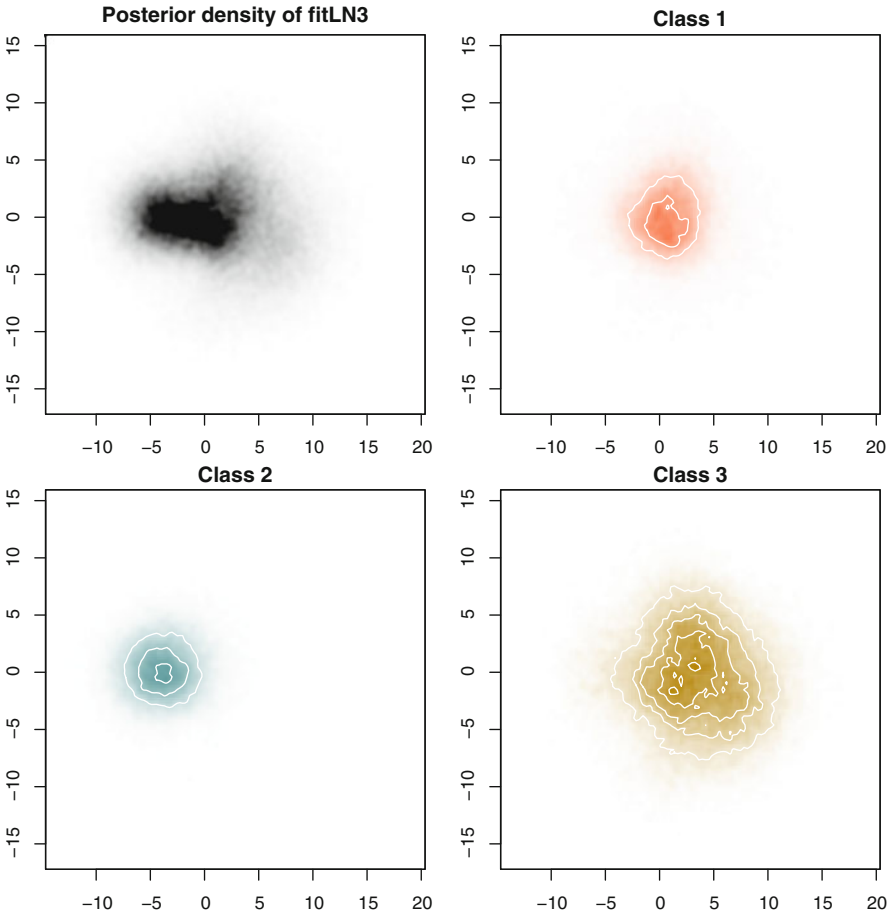

Fig. 11.8 Posterior densities. Top left panel: joint density. Top right panel: OCD density. Bottom panels: densities for depression subclusters

11.4 Bayesian Networks

11.4.1 Directed Acyclic Graphs

Correlation networks do not allow for any directed interpretation. In this section we introduce a network approach that is not based on input correlations: *Bayesian networks* (BNs). The graph theory foundation of BNs are *directed acyclic graphs* (DAGs). What makes these structures so important is that they allow us to convert causal assumptions into conditional independence statements which, in turn, can be subject to statistical hypothesis testing.

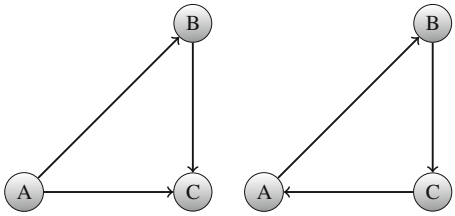

Fig. 11.9 Left panel: directed, acyclic graph. Right panel: directed cyclic graph

DAGs are graph structures that contain no cycles. An example of a DAG and a cyclic graph involving three nodes A, B, and C is given in Fig. 11.9. This idea can be integrated in much more complex network structures involving many nodes. BNs are DAGs. That is, for a fitted BN, we will not be able to find a cycle in our network, no matter how hard we try. How such DAG structures can be translated into conditional dependence statements is elaborated in Scutari and Denis (2015). More technical details on DAGs can be found in Pearl (2009) and Højsgaard et al. (2012).

11.4.2 Bayesian Networks Taxonomy

When fitting a BN, we can distinguish between two basic paradigms. First, driven by hypotheses, the user can set up one or more DAG structures, reflecting various sets of hypotheses, and perform statistical tests on the structure. Such significance tests can involve testing the presence/absence of specific edges, testing particular conditional independence assumptions, and comparing various fitted networks using BIC, for instance. Second, we can fit BNs in a data-driven way and let the algorithm come up with a network structure, potentially subject to further refinements. Such networks are called *learning networks*. In this chapter we focus on learning networks.

An additional distinction can be made based on the scale level of the input data:

- Multinomial BNs: input data are categorical and the joint distribution will be multinomial.
- Gaussian BNs: input data are metric and normally distributed; the joint distribution will be multivariate normal.
- Hybrid BNs: input variables are of mixed scale levels.

For hypothesis-driven networks, in the multinomial case, we can perform χ^2-tests for conditional independence hypotheses involving the corresponding probability subtables. In the Gaussian case, conditional independence hypotheses can be examined through tests on partial correlation coefficients.

The field of BNs is comprehensive and things can get fairly technical. There are many algorithms that can be used to fit a network. Some of them are faster than others; some of them are more favorable under certain circumstances than

others. However, regardless of which algorithm we use, the work flow is inherently Bayesian: at the end we get a posterior distribution involving the graph structure and the parameters, given the data. An excellent, easy-to-read treatment of BNs is the textbook by Scutari and Denis (2015). A comprehensive BN implementation in R is provided by the **bnlearn** package (Scutari, 2010) which we will use below.

11.4.3 Bayesian Network Depression/OCD Data

In this section we show a Bayesian learning network application using the depression/OCD dataset from above. It replicates some of the analyses presented in McNally et al. (2017). We fit a Gaussian BN since the variables, even though on 4 and 5-point scales, are not heavily skewed. We use a simple algorithm called *hill-climbing* to learn the structure of the network and its parameters. This algorithm starts with a representation without any edge and, subsequently, adds, deletes, and reverses one edge at a time until a target score (which judges the goodness of fit) cannot be improved any further. We use the BIC as target score. It is good practice to perform multiple restarts of the algorithm and do several perturbations on every random restart (i.e., adding/removing/deleting edges). In our application we perform 10 random restarts with 100 perturbations in order to avoid nasty local minima.

In BNs we use the persons $\times$ variables matrix as input; no correlation computation is involved. Note that **bnlearn** fits a multinomial network if the variables are declared as factors and a Gaussian network otherwise. Since we want to fit a Gaussian network, we declare the variables as numeric. The following code performs this data preparation step and fits the Bayesian learning network:

```
library("bnlearn")
Rogers2 <- as.data.frame(apply(Rogers, 2, as.numeric))
set.seed(123)
fitBN <- hc(Rogers2, restart = 10, perturb = 100)
```

After the fit, we can compute a measure for the strength of the connections. We use the BIC as criterion for edge strength. The smaller the BIC value, the stronger the connection. Let us print out the five strongest edges.

```
estrength <- arc.strength(fitBN, Rogers2, "bic-g")
head(estrength[order(estrength[,3]), ], 5)
##           from         to   strength
## 34    obresist  compresis -102.56289
## 35  obinterfer obdistress  -59.36705
```

(continued)

```
## 31 incappetite weightgain  -50.46129
## 61    compcont compinterf  -46.23236
## 39  obinterfer compinterf  -40.49906
```

The resulting, funky network plot is given in Fig. 11.10 and can be produced as follows:

```
strength.plot(fitBN, estrength,
              main = "Bayesian Network Depression/OCD",
              shape = "ellipse")
```

One problem with this network, even though we did random restarts and perturbations, is that the graph structure is not very stable since, having 26 nodes, a sample size of 408 is not very large.[4] A trick to stabilize the network is to perform *model averaging* based on bootstrap samples of the data. In **bnlearn** this can be achieved through the following function call where we use 500 bootstrap samples. We print out the results for the first few edges.

```
set.seed(123)
bootnet <- boot.strength(Rogers2, R = 500, algorithm = "hc")
head(bootnet)
##    from          to strength direction
## 1 onset      middle    0.872 0.5596330
## 2 onset        late    0.972 0.3724280
## 3 onset    hypersom    0.032 0.0937500
## 4 onset         sad    0.092 0.3913043
## 5 onset decappetite    0.074 0.8108108
## 6 onset incappetite    0.066 0.3636364
```

The strength value in the first line says that 87.2% of the fitted networks include the connection from onset to middle. The probability that the corresponding edge goes in that direction is 0.56. This probability is helpful when it comes to directional interpretation, which we will discuss at the end of this section.

Based on these networks, we can now compute an average network for which we need to set a threshold for including edges. Below we set a threshold for the strength of 0.85 which means that only edges should only be retained if they appear

[4]The reader is encouraged to refit the model multiple times (of course, without setting a random number seed), and it will be obvious that the graph changes from fit to fit.

in at least 85% of the networks. This is an ad hoc value according to Sachs et al. (2005). The averaged network can be computed as follows:

```
avgnet <- averaged.network(bootnet, threshold = 0.85)
```

Note that a more sophisticated version in terms of a statistically motivated threshold selection is presented in Scutari and Nagarajan (2013) and implemented in `averaged.network` as default setting.

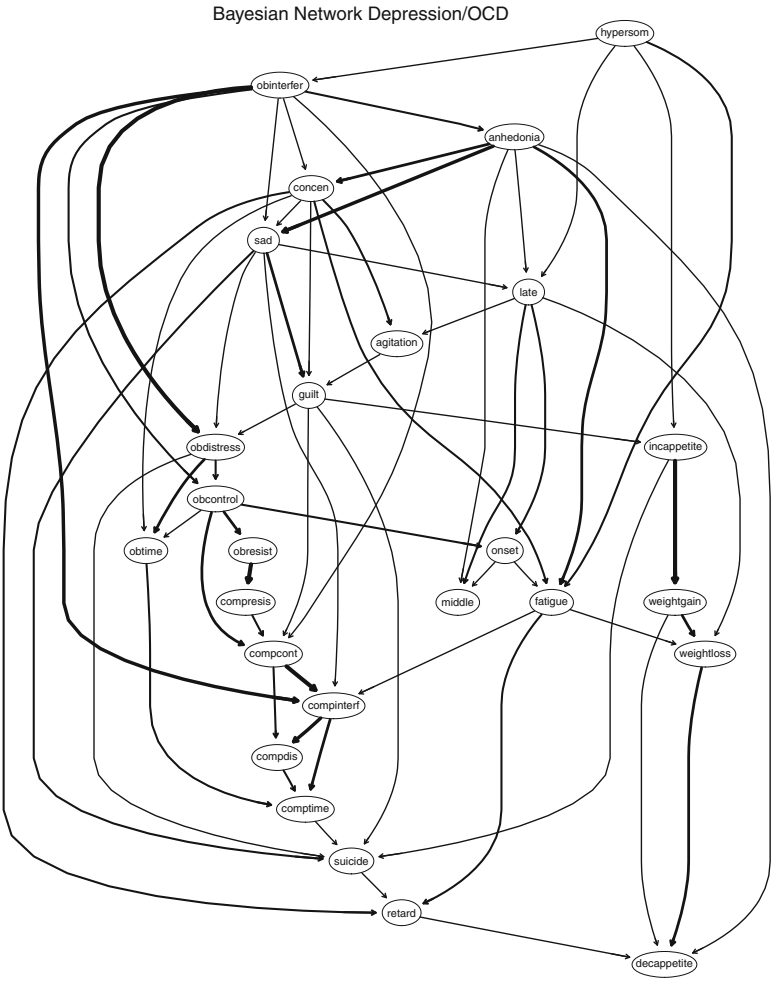

Fig. 11.10 Bayesian network for depression/OCD data

For the averaged network fitted above, we can compute the edge strengths using the BIC once more. The larger the edge BIC, the more damaging it would be to model fit if we were to remove the edge from the network.

```
estrength <- arc.strength(avgnet, Rogers2, "bic-g")
strength.plot(avgnet, estrength, shape = "ellipse")
```

The resulting network plot is given in Fig. 11.11. We see two isolated "depression islands" (late/onset/middle) and items related to appetite/weight with no influence on the other items. This is consistent with the findings in the correlation and latent networks where these items were placed in the periphery of the depression cluster.

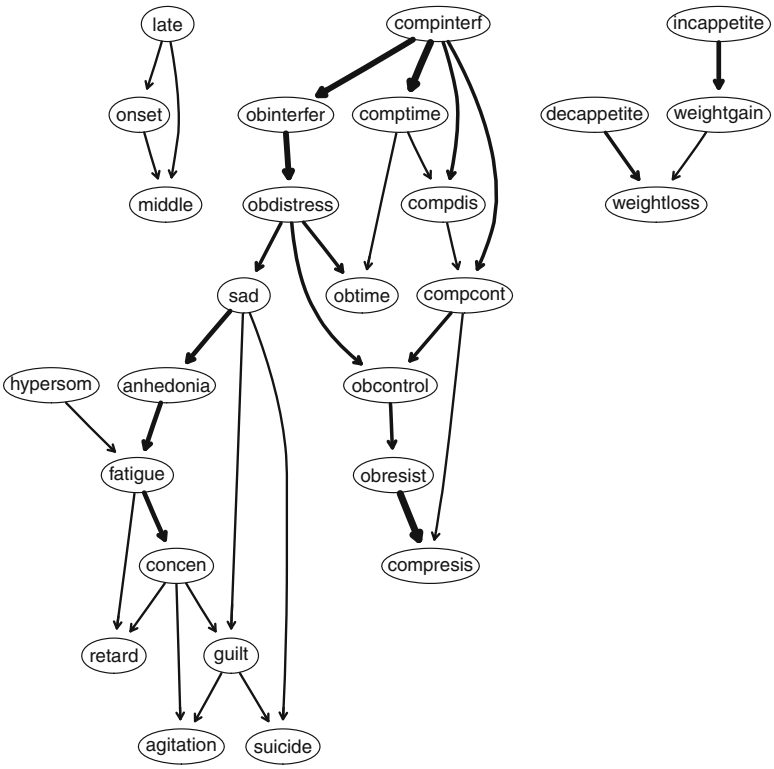

Fig. 11.11 Averaged Bayesian network for depression/OCD data. The thickness of an edge reflects the magnitude of its BIC value

The remaining depression and the OCD items are separated, with "sad" being an important connector between the two.

The final question is: Can we really interpret the node connections in a strict causal way? The diplomatic answer is that we get hints for causal relationships. Scutari and Denis (2015, Section 4.7) provide a good discussion on this topic and suggest that the term "direction" should be used instead of causality. For causal interpretations, additional assumptions are needed which are difficult to verify in practical applications. Pearl (2009) elaborates on these assumptions in a detailed (and technical) way; more narrative explanations can be found in McNally et al. (2017).

From a statistical point of view, the probabilities of the edge directions from the bootstrap networks give us some support. Let us take a subset the strongest connections:

```
subedge <- head(bootnet[bootnet$strength > 0.95, ])
subedge
##          from      to strength direction
## 2       onset    late    0.972 0.3724280
## 27     middle    late    0.982 0.3635438
## 51       late   onset    0.972 0.6275720
## 52       late  middle    0.982 0.6364562
## 88  hypersom fatigue    1.000 0.6390000
## 110       sad   guilt    0.972 0.9074074
```

In the first row, we see that the onset → late probability is 0.372. Since it is smaller than 0.5, the edge in Fig. 11.11 goes the other way. However, we have a high probability for the sad → guilt edge (0.907).

Instead of using the BIC for edge thickness as in Fig. 11.11, the following code chunk weights the edges by their direction probability and produces Fig. 11.12.

```
boottab <- bootnet[bootnet$strength > 0.85 &
                   bootnet$direction > 0.5, ]
astr <- boottab
astr$strength <- astr$direction
strength.plot(avgnet, astr, shape = "ellipse")
```

This representation gives good "hints" in terms of a directional interpretation of the symptom nodes (e.g., sad → suicide and sad → guilt). This concludes the section on BNs; further details can be found in Scutari and Denis (2015).

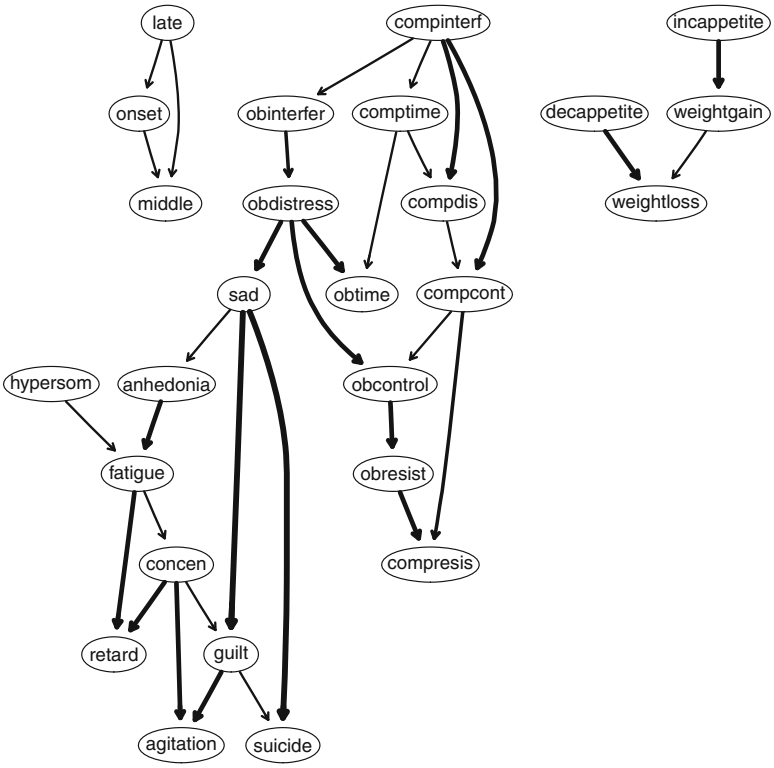

Fig. 11.12 Averaged Bayesian network for depression/OCD data. The thickness of an edge reflects the direction probability

References

Borsboom, D., & Cramer, A. O. J. (2013). Network analysis: An integrative approach to the structure of psychopathology. *Annual Review of Clinical Psychology, 9*, 91–121.

Butts, C. (2008). **network**: A package for managing relational data in R. *Journal of Statistical Software, 24*(2), 1–36. http://www.jstatsoft.org/v24/i02/paper

Csárdi, G., & Nepusz, T. (2006). The **igraph** software package for complex network research. *InterJournal Complex Systems*, 1695. http://igraph.org/

Epskamp, S., Cramer, A. O. J., Waldorp, L. J., Schmittmann, V. D., & Borsboom, D. (2012). **qgraph**: Network visualizations of relationships in psychometric data. *Journal of Statistical Software, 48*(4), 1–18. http://www.jstatsoft.org/v48/i04/

Friedman, J., Hastie, T., & Tibshirani, R. (2008). Sparse inverse covariance estimation with the graphical lasso. *Biostatistics, 9*, 432–441.

Friedman, J., Hastie, T., & Tibshirani, R. (2014). **glasso**: Graphical lasso estimation of Gaussian graphical models. R package version 1.8. https://CRAN.R-project.org/package=glasso

Hoff, P. D. (2008). Modeling homophily and stochastic equivalence in symmetric relational data. In J. C. Platt, D. Koller, Y. Singer, & S. Roweis (Eds.) *Advances in neural information processing systems 20* (pp. 657–664). Cambridge: MIT Press.

Hoff, P. (2012). **eigenmodel**: Semiparametric factor and regression models for symmetric relational data. R package version 1.01. https://CRAN.R-project.org/package=eigenmodel

Hoff, P. D., Raftery, A. E., & Handcock, M. S. (2002). Latent space approaches to social network analysis. *Journal of the American Statistical Association, 97*, 1090–1098.

Højsgaard, S., Edwards, D., & Lauritzen, S. (2012). *Graphical models with R*. New York: Springer.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning with applications in R*. New York: Springer.

Kolaczyk, E. D., & Csárdi, G. (2014). *Statistical analysis of network data with R*. New York: Springer.

Krivitsky, P. N., & Handcock, M. S. (2008). Fitting position latent cluster models for social networks with **latentnet**. *Journal of Statistical Software, 24*(5), 1–23. https://www.jstatsoft.org/article/view/v024i05

McNally, R. J., Mair, P., Mugno, B. L., & Riemann, B. C. (2017). Comorbid obsessive-compulsive disorder and depression: A Bayesian network approach. *Psychological Medicine, 47*, 1204–1214.

Pearl, J. (2009). *Causality*. New York: Cambridge University Press.

Sachs, K., Perez, O., Pe'er, D., Lauffenburger, D. A., & Nolan, G. P. (2005). Causal protein-signaling networks derived from multiparameter single-cell data. *Science, 308*, 523–529.

Scutari, M. (2010). Learning Bayesian networks with the **bnlearn** R package. *Journal of Statistical Software, 35*, 1–22. http://www.jstatsoft.org/v35/i03/

Scutari, M., & Denis, J. B. (2015). *Bayesian networks with examples in R*. Boca Raton: CRC Press.

Scutari, M., & Nagarajan, R. (2013). On identifying significant edges in graphical models of molecular networks. *Artificial Intelligence in Medicine, 57*, 207–217.

Wasserman, S., & Faust, K. (1994). *Social network analysis: Methods and applications*. Cambridge: Cambridge University Press.

Chapter 12
Parametric Cluster Analysis and Mixture Regression

12.1 Model-Based Clustering Approaches: Mixture Models

The aim of clustering is to find homogeneous groups of persons based on several variables under consideration. Basic clustering strategies involve techniques like *hierarchical clustering* and *K-means clustering*. R is of course well equipped with corresponding functions (see, e.g., `hclust` and `K-means` functions, as well as some extensions in the **cluster** package Maechler et al., 2017). A nice feature of hierarchical techniques is that we get a good insight into the cluster amalgamation structure, typically represented as *dendrogram*. Apart from the standard silhouette information (see `silhouette` function), the **NbClust** package (Charrad et al., 2014) offers various tools to support the user with selecting the number of clusters.

In this chapter we do not cover hierarchical techniques or K-means clustering, since these approaches are already explained in detail in other books such as Everitt and Hothorn (2011) and Everitt (2011). Rather, we focus on parametric clustering approaches based on *mixture distributions*. Properties of this clustering concept are the following:

- We can account for various scale properties of the variables under consideration by specifying according distributions.
- We estimate parameters, subject to further statistical inference.
- Goodness-of-fit measures such as the BIC are computed, which give statistical arguments for choosing the number of clusters.
- The assignment of each person to a cluster is done in a *probabilistic* (i.e., *soft*) manner, as opposed to a *deterministic* (i.e., *hard*) assignment as in hierarchical or K-means clustering.
- The parametric clustering concept can be embedded into regression modeling frameworks.

In the remainder of this chapter, we will explore these properties in detail by means of several examples. But first, let us have a look at the theory.

P. Mair, *Modern Psychometrics with R*, Use R!,
https://doi.org/10.1007/978-3-319-93177-7_12

The underlying principle of model-based clustering approaches are mixture distributions. An example of a simple mixture of normal distributions is given in Fig. 12.1, where we try to describe the distribution of variable X by a mixture of $K = 3$ normal distributions. Each *mixture component* (also called *cluster* or *latent class*) j ($j = 1, \ldots, K$) is distributed with density $f_j(x_i; \mu_j, \sigma_j^2)$. The mixture normal density value of an individual observation x_i ($i = 1, \ldots, n$) is given by

$$f(x_i; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) = \sum_{j=1}^{K} \pi_j f_j(x_i; \mu_j, \sigma_j^2). \tag{12.1}$$

The π_j's are the so-called *mixture weights* with $\sum_{j=1}^{K} \pi_j = 1$.

With respect to the densities $f_j(\cdot)$, we are not limited to normal distributions. We can use other distributions $f_j(x_i; \boldsymbol{\theta}_j)$, where $\boldsymbol{\theta}_j$ acts as a general placeholder for the corresponding distributional parameter vector in component j. For instance, having several count variables as inputs, we could define a mixture of Poisson distributions. For such general settings, Eq. (12.1) can be written as

$$f(x_i; \boldsymbol{\theta}) = \sum_{j=1}^{K} \pi_j f_j(x_i; \boldsymbol{\theta}_j). \tag{12.2}$$

This is all the math we need for this chapter since all methods presented below are based on this simple principle. Mixture models are typically estimated by means of the EM algorithm (expectation-maximization), but for some of the more complicated ones, we need Markov chain Monte Carlo (MCMC). Additional technical details on mixture models can be found in McLachlan and Peel (2000).

12.1.1 Normal Mixture Models

Let us focus on normal mixtures first. In psychometrics, this approach is often called *latent profile analysis*. We illustrate this concept using a simple dataset from Mair et al. (2015) who explored what drives developers to contribute to R (see Sect. 4.2.2 for an earlier analysis of these data). We consider three metric variables (factor scores) from the work design questionnaire (WDQ): task, social, and knowledge characteristics.

```
library("MPsychoR")
library("mclust")
data("Rmotivation2")
Rwd <- Rmotivation2[, c(4:6)]   ## work design variables
```

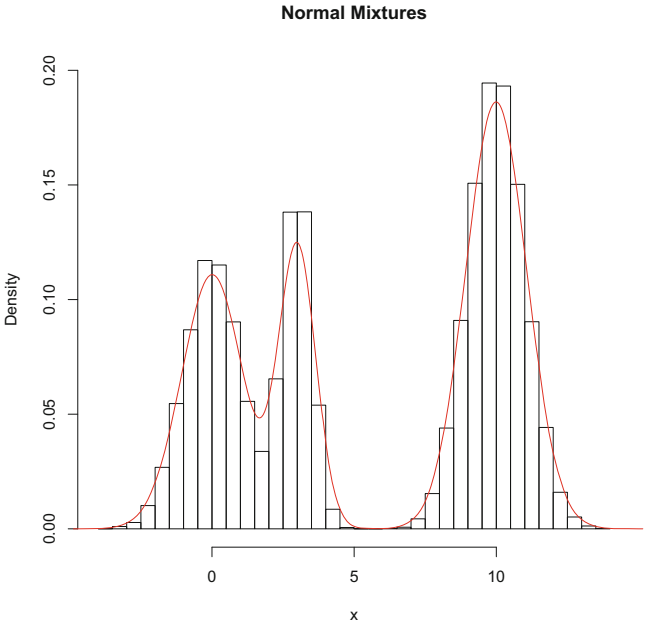

Fig. 12.1 Mixture of three univariate normal distributions

Since we have three input variables, instead of fitting a unidimensional normal mixture as in Eq. (12.1), we are going fit a mixture model based on three-dimensional normal distributions.

Before we start the actual clustering process, there are a few things that need to be pointed out. First, the data itself do not have to be normally distributed since the distribution is approximated by mixtures anyway (cf. Fig. 12.1). Thus, we do not have to worry about individual normality violations; we just need to make sure that our data fit conceptually into a normal distribution framework (e.g., having categorical data or skewed counts, this would not be the case). Second, we need to fix the number of clusters K before fitting the model. In practice, a BIC selection strategy is applied by fitting several candidate models with varying K and, subsequently, pick the solution with the best BIC.[1] The **mclust** package (Fraley and Raftery, 2002), which we are going to use to fit the normal mixture model, provides the following utility function to pursue this strategy. In this example we specify a range for K from 2 to 10 clusters (see `G` argument):

[1] Note that in **mclust** a maximum-BIC strategy is used; that is, the higher the BIC, the better the fit.

```
set.seed(123)
clustbic <- mclustBIC(Rwd, G = 2:10)
clustbic
```

From the output (not shown here), we see that for each K, the algorithm fits a variety of normal mixture models. They are mostly based on particular shape restrictions in the variance-covariance (VC) matrix (see `?mclustModelNames` for details). Let us refit the model that has been identified as the best one and then have a look at the output parameters:

```
clusfit <- Mclust(Rwd, x = clustbic)
summary(clusfit, parameters = TRUE)
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VVI (diagonal, varying volume and shape)
## model with 5 components:
##
##  log.likelihood   n df       BIC       ICL
##       -2453.813 764 34 -5133.337 -5436.177
##
## Clustering table:
##   1   2   3   4   5
## 175 241 232  56  60
##
## Mixing probabilities:
##          1          2          3          4          5
## 0.21506086 0.36475594 0.28597464 0.06099677 0.07321179
##
## Means:
##                   [,1]         [,2]       [,3]       [,4]       [,5]
## wtask      -0.3383670 -0.05376995 0.1284806 1.2132427 -0.5847775
## wsocial    -0.9491831  0.35219937 0.2831322 0.6514594 -0.3949974
## wknowledge -0.1682797 -0.51916620 0.6747767 0.1525975 -0.5591489
```

The BIC search suggests a five-cluster solution based on a VVI model (diagonal, varying volume, and shape), meaning that the VC matrix is diagonal (uncorrelated variables within each cluster) and that the shape and volume of the 3D normal components are unrestricted.

Let us explore this output in more detail. In terms of cluster assignments, we have already mentioned that model-based clustering approaches give probabilistic cluster membership for each individual. We can extract these *posterior probabilities* as follows (only first six persons are shown here):

```
head(round(clusfit$z, 3))   ## soft
##     [,1]  [,2]  [,3] [,4]  [,5]
## 1 0.000 0.056 0.909    0 0.035
## 4 0.000 0.713 0.287    0 0.000
## 5 0.000 0.297 0.703    0 0.000
## 6 0.967 0.001 0.032    0 0.000
## 8 0.000 0.987 0.013    0 0.000
## 9 0.000 0.997 0.003    0 0.000
cluster <- as.factor(clusfit$classification)  ## hard
```

Based on the maximum posterior probability, each person can be assigned to a cluster in a deterministic way. For instance, the first person is in cluster 3, the second person in cluster 2, the third person in cluster 3, etc. The clustering table in the summary output above results from these hard assignments and tells us how many persons there are in each cluster. The mixing probabilities are the weights of the density mixtures (i.e., π_j in Eq. (12.1)).

In the summary output, we also get the mean vectors $\boldsymbol{\mu}_j$ for each component. On the basis of these means, we can describe the clusters. Cluster 1 is dominated by persons with high knowledge characteristics, cluster 2 consists of persons with very low knowledge characteristics, and cluster 3 has persons with high task and social characteristics, whereas in cluster 4 we find people that score low on these two variables.

Typically, a parametric cluster analysis is based on several input variables which make it difficult to plot the mixture densities. The trick is to apply dimensionality reduction on the variables by means of a principal component analysis (PCA; see Chap. 6) and plot the clusters in a 2D space. The following call performs the dimensionality reduction:

```
clustred <- MclustDR(clusfit)
```

Based on this object, the plots in Fig. 12.2 can be produced as follows:

```
plot(clustred, what = "boundaries", ngrid = 200)
plot(clustred, what = "density", dimens = 1)
```

The top panel shows a 2D representation of the cluster boundaries; the persons are colored according to their hard cluster membership. We see that the clusters are clearly separated. The bottom panel shows a 1D representation along the first dimension, similar to what we had in Fig. 12.1. Additional options can be found in `?plot.MclustDR`.

For further cluster inspection, we can consider external variables that were not part of the clustering process. Here we use the number of packages (`npkgs`; metric) and whether a developer has been active on R-help or similar lists (`lists`; categorical) and produce the following plots:

```
library("lattice")
densityplot(~npkgs|cluster, data = Rmotivation2, col = "gray",
            index.cond = list(c(3, 4, 1, 2)))
histogram(~lists|cluster, data = Rmotivation2, col = "gray",
          index.cond = list(c(3, 4, 1, 2)))
```

The cluster-specific density plots for the number of packages are given in the top panel of Fig. 12.3. Cluster 4 differs clearly from the remaining three clusters, containing mostly users who developed a low number of packages. The bottom panel shows the cluster-specific bar charts for list participation. Again, cluster 4 is distinct form the remaining three clusters since it is the only cluster that contains more respondents who do not participate in lists.

At this point we have fully interpreted the four-cluster solution of our work design data. Note that the aspect of cluster interpretability is highly important in mixture models (and cluster analysis in general). Sometimes a solution suggested by the BIC does not provide good interpretability. In such cases we can pick a solution with slightly higher or lower K (which does not lead to a drastically worse BIC), if this helps with interpretability. On a related note, regardless which clustering method we use, it can happen that all clusters but one ("junk cluster") can be meaningfully interpreted. Such a solution is not problematic since the junk cluster contains a heterogeneous set of persons, which simply did not fit into any of the remaining clusters.

12.1.2 Latent Class Analysis

Latent class analysis (LCA) was developed in the area of psychometrics (Lazarsfeld and Henry, 1968) and is designed for categorical input data. From a modern perspective, LCA is nothing else than a mixture distribution model where the underlying densities are specified using a binomial distribution for dichotomous items, or a multinomial distribution for polytomous items.

In order to show a simple application of LCA, we use a dataset from Haegeli et al. (2012) who studied high-risk cohorts in a complex and dynamic risk environment. There are four input variables related to preparedness before going backcountry skiing. The variables under consideration have three to five ordinal response categories. We use the **poLCA** package (Linzer and Lewis, 2011) to compute the LCA. As above, we fit a sequence of LCA solutions with varying K (here

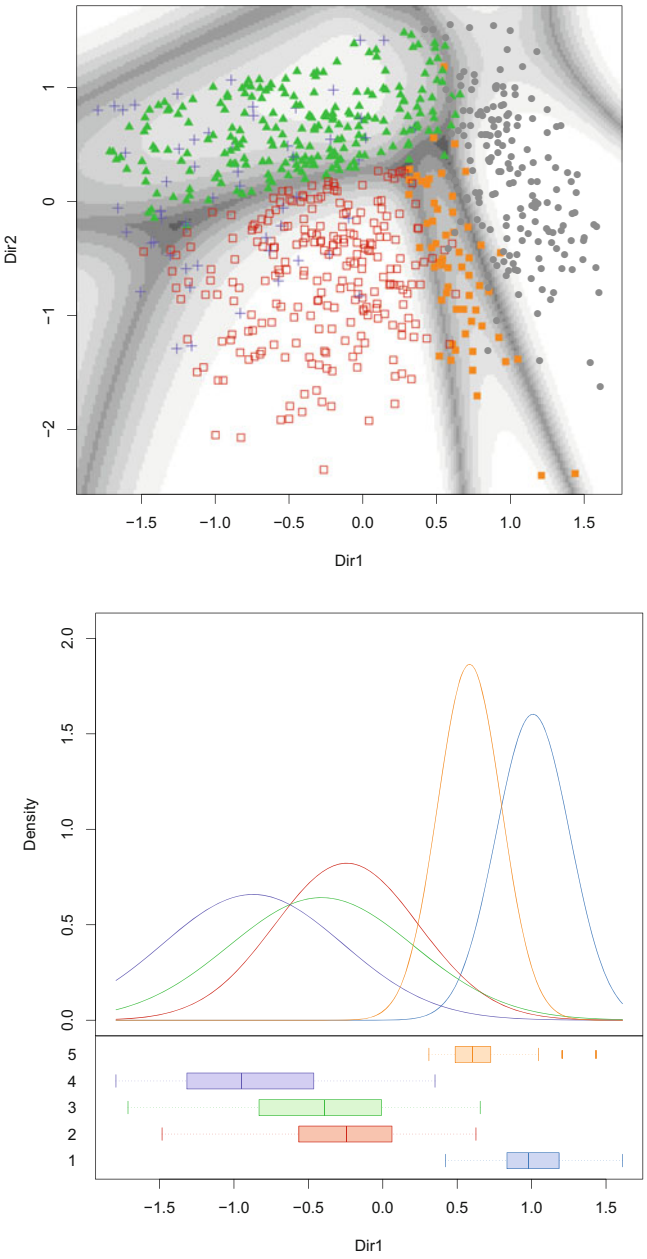

Fig. 12.2 Top panel: 2D cluster boundaries. Bottom panel: 1D densities of mixture components

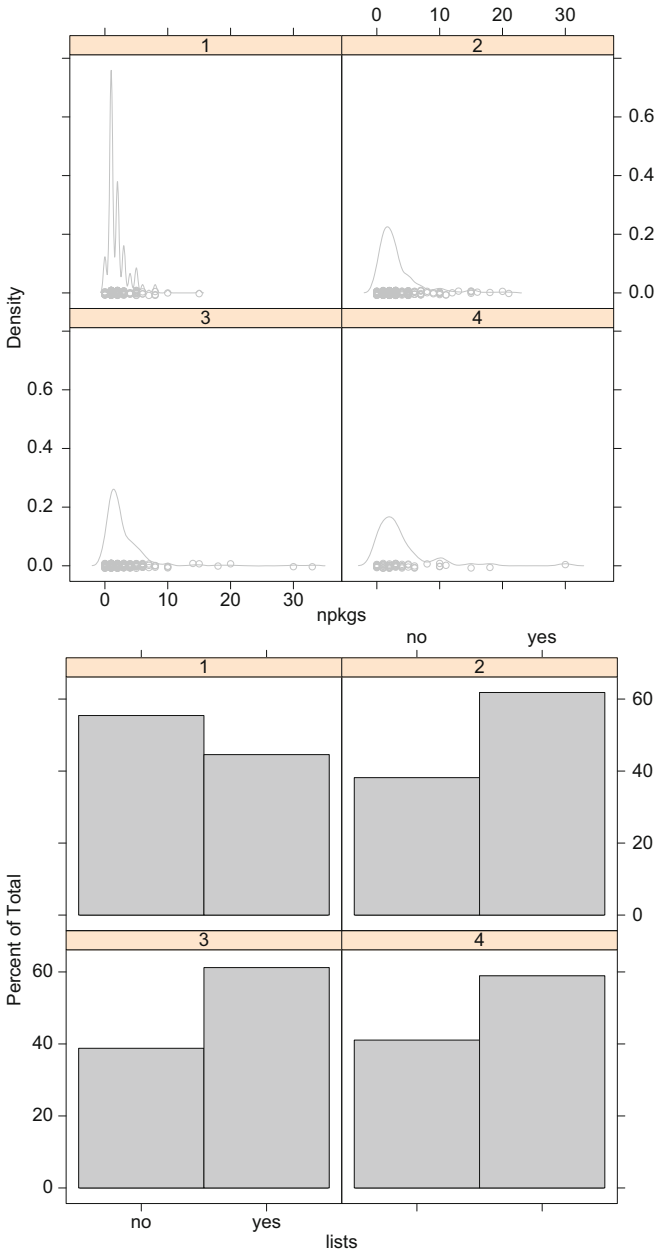

Fig. 12.3 Cluster evaluation using external variables. Top panel: number of packages (metric). Bottom panel: participation in lists (categorical)

from 1 to 4). We refit each model three times in order to avoid ending up in a local maximum, since the EM algorithm is sensitive to starting values. The `poLCA` function returns the model with the best fit for a given K.

```
library("MPsychoR")
library("poLCA")
data("AvalanchePrep")
formula <- cbind(info, discuss, gear, decision) ~ 1
set.seed(1)
fitlca1 <- poLCA(formula, data = AvalanchePrep, nclass = 1,
                 nrep = 3)
fitlca2 <- poLCA(formula, data = AvalanchePrep, nclass = 2,
                 nrep = 3)
fitlca3 <- poLCA(formula, data = AvalanchePrep, nclass = 3,
                 nrep = 3, maxiter = 2000)
fitlca4 <- poLCA(formula, data = AvalanchePrep, nclass = 4,
                 nrep = 3, maxiter = 5000)
```

We extract the BICs (here the usual minimum BIC rule applies) in order to determine the number of clusters K:

```
c(fitlca1$bic, fitlca2$bic, fitlca3$bic, fitlca4$bic)
## [1] 7249.481 6698.558 6563.062 6629.098
```

The BIC suggests that the three-cluster solution fits the data best.

```
fitlca3
## Conditional item response (column) probabilities,
##  by outcome variable, for each class (row)
##
## $info
##               1      2      3      4
## class 1:  0.9682 0.0218 0.0053 0.0047
## class 2:  0.8290 0.0562 0.0635 0.0513
## class 3:  0.8436 0.0195 0.0000 0.1369
##
## $discuss
##               1      2      3     4
## class 1:  0.5738 0.4195 0.0067 0.000
## class 2:  0.0000 0.6822 0.2867 0.031
## class 3:  0.0000 0.0000 0.0000 1.000
##
## $gear
##               1      2      3      4      5
## class 1:  0.8147 0.0738 0.0630 0.0358 0.0126
## class 2:  0.0142 0.0725 0.2515 0.4996 0.1622
## class 3:  0.0000 0.0000 0.0000 0.0000 1.0000
```

(continued)

```
##
## $decision
##                1      2      3
## class 1:  0.9739 0.0038 0.0223
## class 2:  0.8790 0.0303 0.0908
## class 3:  0.0034 0.0000 0.9966
##
## Estimated class population shares
##  0.8218 0.1405 0.0377
##
## Predicted class memberships (by modal posterior prob.)
##  0.8354 0.1269 0.0376
```

For each category we get a class-specific endorsement probability. For instance, for the "check avalanche danger information" item (`info`), the probability for scoring 1 ("check conditions on internet prior to leaving home") is high in all three clusters. For the decision-making item, the first two clusters contain people who score 1 ("dedicated leader or everybody contributes"), whereas cluster 3 members mostly score 3 ("everybody makes their own choices or solo traveler"). Based on these probabilities, clusters can be interpreted. The authors of this study named cluster 1 "good habits," cluster 2 "poor habits," and cluster 3 "deficient habits." A plot of these probabilities is given in Fig. 12.4, by simply saying `plot(fitlca3)`. The hard cluster memberships, potentially subject to further external evaluation, can be extracted a follows:

```
clusmemb <- as.factor(fitlca3$predclass)
```

This latent class idea (or, more general, mixture distribution idea) has been integrated into other psychometric techniques. For instance, the **psychomix** package (Frick et al., 2012) allows for fitting Rasch models (see Sect. 4.2.1) and Bradley-Terry models (see Sect. 5.1.1) in a mixture fashion.

12.1.3 Parametric Clustering with Mixed Scale Levels

To illustrate a mixed scale level clustering, we use the ZAREKI-R data (Koller and Alexandrowicz, 2010) introduced in Sect. 4.1.2. This dataset uses eight addition and eight subtraction items for the assessment of dyscalculia in children. Below we select four addition/subtraction items (dichotomous), and the time it took the children to complete the test (metric). For the binary items, we use mixtures of binomial distributions and for "time" a normal mixture. To fit the model, we use the **flexmix** package (Grün and Leisch, 2008) which, as the name already suggests, provides a flexible infrastructure for fitting all sorts of mixture models. The core

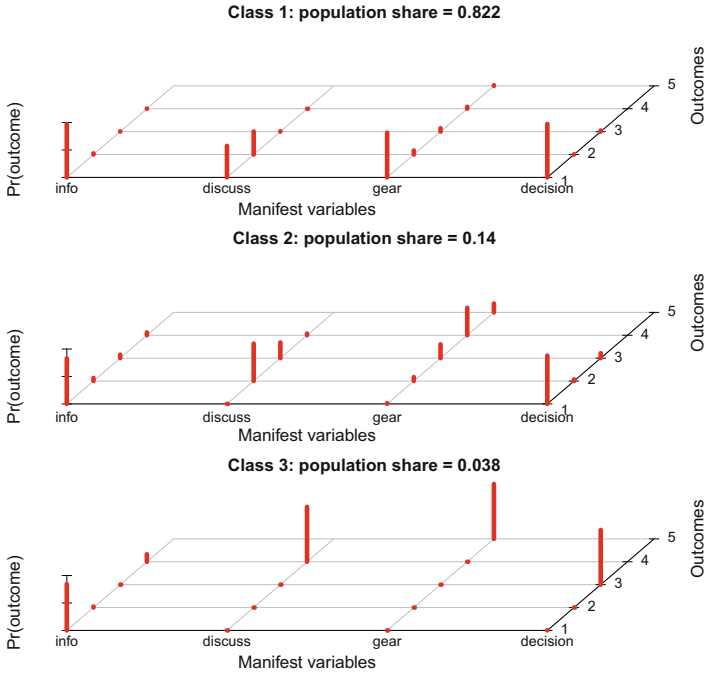

Fig. 12.4 Class-specific item category probabilities resulting from a 3-cluster LCA solution

function of the package is `flexmix`. As above, we vary the number of clusters K, and for each K we compute several replications. The following function call performs the model search.

```
library("flexmix")
data("zareki")
set.seed(123)
zarflex <- stepFlexmix(~ 1, data = zareki, k = 1:4, nrep = 3,
    model = list(FLXMRmultinom(addit7 ~ .),
                 FLXMRmultinom(addit8 ~ .),
                 FLXMRmultinom(subtr3 ~ .),
                 FLXMRmultinom(subtr7 ~ .),
                 FLXMRglm(time ~ ., family = "gaussian")))
```

Note that if we had a count variable in our dataset, a Poisson distribution can be used, specified through `family="poisson"`. In case of a right-skewed metric variable with a lower bound of 0 (e.g., reaction time often behaves this way), the gamma distribution can be used (`family="gamma"`). For polytomous items we can use the same `FLXMRmultinom` call as above.

Continuing with this example, let us pull out the best model according to the BIC criterion:

```
zarflex2 <- getModel(zarflex, "BIC")
cluster <- zarflex2@cluster
table(cluster)
## cluster
##   1   2
## 147 194
```

A two-cluster solution fits the data well. With `parameters(zarflex2)`, we can extract the distribution parameters. Since the parameters of the four categorical items are on a logit scale, we can apply the usual exponential transformation (as, e.g., in logistic regression) in order to get an odds ratio interpretation.

```
catpars <- sapply(parameters(zarflex2)[1:4], exp)
colnames(catpars) <- c("addit7", "subtr3", "addit8", "subtr7")
catpars                    ## categorical variables
##                addit7    subtr3    addit8    subtr7
## Comp.1.coef 11.052397 4.4732715 4.2684181 3.5306428
## Comp.2.coef  3.858138 0.6085434 0.3224599 0.4472089
parameters(zarflex2)[[5]]   ## time variable
##                      Comp.1    Comp.2
## coef.(Intercept) 22.979615 30.302752
## sigma             3.119102  6.084501
```

We see that members of cluster 1 have higher odds to solve any item than cluster 2 members. The last part shows the parameters for the time variable. The intercepts correspond to the cluster-specific means and suggest that children in cluster 2 need more time for test completion, compared to those in cluster 1.

Based on the hard cluster assignments in `zarflex2@cluster`, we can produce mosaic plots using the **vcd** package (Meyer et al., 2006) for each categorical input variable (see Fig. 12.5). We see that these plots confirm what we have seen from the odds: cluster 1 contains predominantly children who solved the items correctly.

12.1.4 Concomitant Variables

The weights π_j in a mixture model indicate an a priori probability for an individual to be in component j. So far, these weights were updated during the clustering process and were not influenced by any specific variable. Mixture models give

us the possibility to specify covariates (e.g., sociodemographic variables) that influence these weights. In cluster analysis slang, such a covariate w is called *concomitant variable*. Concomitant variables affect the weights and, therefore, the cluster assignments. In Eq. (12.2) we can replace π_j by $\pi_j(w, \alpha)$, where $\boldsymbol{\alpha}$ is the parameter vector associated with the concomitant variable.

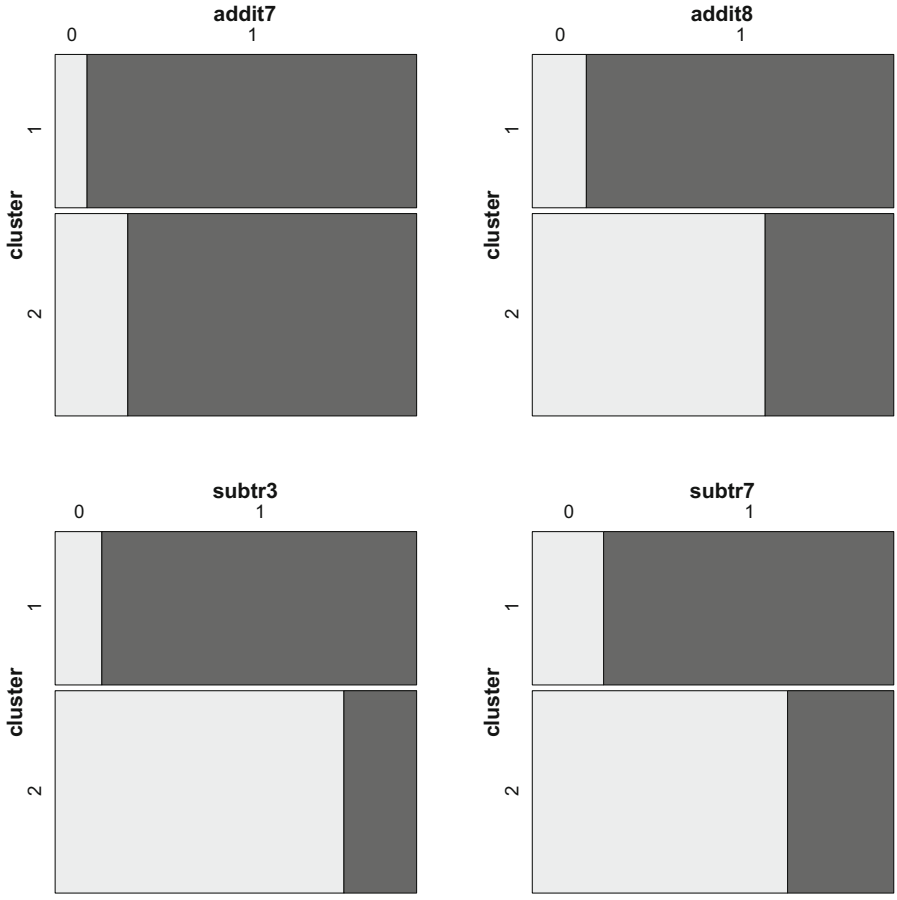

Fig. 12.5 Mosaic plots for cluster evaluation: four items cross-classified with hard cluster membership

To illustrate this concept, we refit the two-cluster model from above using `class` (2nd, 3rd, 4th grade) as concomitant variable. In order to support the algorithm with reasonable starting values, we can use the posterior probabilities from above as starting solution.

```
zarflexc <- flexmix(~ 1, data = zareki,
      cluster = posterior(zarflex2),
      concomitant = FLXPmultinom(~ class),
      model = list(FLXMRmultinom(addit7 ~ .),
                   FLXMRmultinom(subtr3 ~ .),
                   FLXMRmultinom(addit8 ~ .),
                   FLXMRmultinom(subtr7 ~ .),
                   FLXMRglm(time ~ ., family = "gaussian")))
zarflexc@prior   ## weights with concomitant
## [1] 0.5861112 0.4138888
zarflex2@prior   ## weights without concomitant
## [1] 0.433666 0.566334
```

We see that the mixture weights have changed. The parameter vector $\boldsymbol{\alpha}$ on an odds scale is:

```
exp(zarflexc@concomitant@coef)
##             1           2
## (Intercept) 1 11.00637734
## classthird  1  0.01691578
## classfourth 1  0.01176601
```

The concomitant parameters for cluster 1 are fixed to 1, and 2nd grade is used as reference category. The output suggests that for 2nd graders, the odds for ending up in cluster 2 is 11 times higher than ending up in cluster 1. For 3rd and 4th graders, these odds are low; most likely these kids fall into the first cluster. In order to explore this output in more detail, let us extract the cluster assignments and cross-classify them with grade. For comparison, we also present the same table from the fit above without a concomitant variable:

```
clusterc <- zarflexc@cluster
table(zareki$class, clusterc)  ## with concomitant variable
##         clusterc
##            1   2
##   second   8 114
##   third   88  11
##   fourth 108  12
table(zareki$class, cluster)   ## without concomitant variable
##         cluster
##            1   2
##   second  13 109
##   third   58  41
##   fourth  76  44
```

The concomitant variable solution provides a clearer separation of the school classes across the clusters: the second cluster mostly consists of 2nd-grade pupils, whereas cluster 1 contains largely 3rd- and 4th-grade pupils. Without class as concomitant variable, the 3rd and 4th graders are distributed more evenly across the two clusters. After including the class variable into our model, there are still some "special" kids who do not belong to the same cluster than their peers. For instance, there are 8 second graders who perform like third (or even fourth) graders, and there are 11 third and 12 fourth graders who perform weaker than their peers.

Note that the concomitant variable also changes the distributional parameters, as can be explored via `parameters(zarflexc)`. Still, the general picture holds: in the first cluster there are students with a high probability to solve the items correctly and having a low average test taking time.

12.2 Mixture Regression Models

12.2.1 Mixture Regression Theory

The **flexmix** package is even more flexible than what we have shown so far. It is possible to incorporate the mixture distribution concept into a regression framework. This approach is sometimes referred to as *mixture regression* or *latent class regression*. The basic idea is to perform regression fit and clustering simultaneously. That is, for a given K, we estimate K regression models. Let us have a closer formal look at how this idea fits into the mixture density framework.

Let y_i be the score on the response variable of individual i, and $\mathbf{x}_i$ the corresponding predictor vector (including a value of 1 for the intercept). For the associated random variable Y_i, in an ordinary linear regression, we assume that $Y_i \sim N(\mathbf{x}_i'\boldsymbol{\beta}, \sigma^2)$. Thus, the distribution of Y_i can be expressed as a normal density given the predictors, that is, $f(y_i|\mathbf{x}_i; \boldsymbol{\beta}, \sigma^2)$. Let us integrate this idea into Eq. (12.1) by considering $j = 1, \ldots, K$ components:

$$f(y_i|\mathbf{x}_i; \boldsymbol{\beta}, \boldsymbol{\sigma}^2) = \sum_{j=1}^{K} \pi_j f_j(y_i|\mathbf{x}_i; \boldsymbol{\beta}_j, \sigma_j^2) \tag{12.3}$$

For each component j, we estimate a set of regression parameters $\boldsymbol{\beta}_j$ and the corresponding residual variance σ_j^2. Again, π_j are the mixture weights.

As Eq. (12.1) was generalized to Eq. (12.2), we can do the same thing for Eq. (12.3):

$$f(y_i|\mathbf{x}_i; \boldsymbol{\theta}) = \sum_{j=1}^{K} \pi_j f_j(y_i|\mathbf{x}_i; \boldsymbol{\theta}_j), \tag{12.4}$$

where $\boldsymbol{\theta}_j$ is a general placeholder for a variety of regression parameters. This formulation allows us to extend the normal mixture regression from Eq. (12.3) to generalized linear model (GLM) settings, linear mixed-effects models (LME), generalized additive models (GAM), etc. Below we show some examples.

12.2.2 Mixture Regression Applications

In this section we present three applications of increasing complexity. Let us start with a toy example based on simulated data in order to show a simple linear mixture regression fit.

```
set.seed(123)
X <- rnorm(100)
Y1 <- 2*X + rnorm(100, sd = 0.5)
Y2 <- (-2)*X + rnorm(100, sd = 0.5)
toydat <- data.frame(X = c(X, X), Y = c(Y1, Y2))
lm(Y ~ X, data = toydat)
##
## Call:
## lm(formula = Y ~ X, data = toydat)
##
## Coefficients:
## (Intercept)            X
##    0.007454    -0.046723
```

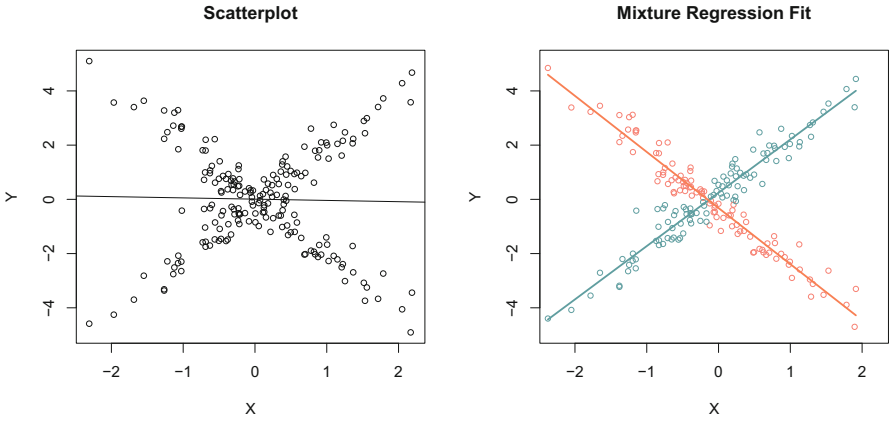

Fig. 12.6 Left panel: scatterplot with simple linear regression fit. Right panel: mixture regression fit including the two fitted regression lines and the hard cluster memberships

The scatterplot in the left panel of Fig. 12.6 takes the form of an "X." A simple linear regression cannot capture this pattern; it simply fits a horizontal line. Let us fit a two-component mixture regression model using **flexmix** on these data and see what happens.

```
library("flexmix")
toymix <- flexmix(Y ~ X, k = 2, data = toydat)
parameters(toymix)
##                      Comp.1      Comp.2
## coef.(Intercept) -0.08339254  0.06157986
## coef.X            1.96650205 -2.06844244
## sigma             0.50933200  0.47158323
```

We get two sets of regression parameters, one for each component: intercept, slope, and residual standard deviation. The right panel in Fig. 12.6 confirms that the mixture regression nicely captures the scatterplot structure. Again, the assignment of each individual to a cluster is carried out in a probabilistic way, which subsequently can be turned into a hard membership (see `toymix@cluster`). Similar toy examples involving quadratic regression and Poisson regression for counts are given in Grün and Leisch (2008).

Now let us use a real-life dataset which leads to a *mixture mixed-effects model* since the design involves repeated measurements. The data are taken from Winter and Grawunder (2012) who investigated properties of formal and informal speech registers in Korean. Six persons (three males, three females) were part of a linguistic experiment. For each person we have 14 measurements in total (except for one person where one response is missing). The response variable is the pitch frequency in Hz. Below we use attitude (polite vs. informal) as fixed effect and take into account the repeated measurement nature of the experiment through a subject random effect. We fit a $K = 2$ cluster solution.[2] Note that for illustration purposes, we do not include gender in the model.

```
library("MPsychoR")
data("KoreanSpeech")
set.seed(123)
koreamix <- flexmix(frequency ~ attitude|subject, k = 2,
                    data = na.omit(KoreanSpeech))
table(koreamix@cluster)
##
```

(continued)

[2]In practice, the user should again try out different numbers of clusters and pick the one with the lowest BIC.

```
##  1  2
## 41 42
parameters(koreamix)
##                       Comp.1    Comp.2
## coef.(Intercept) 144.49048 260.68571
## coef.attitudepol -11.51048 -27.40000
## sigma             38.48414  31.72443
```

The two clusters are of approximately the same size. Note that the cluster size is determined by the total number of observations and not by the number of persons (since we have a repeated measurement design). Each cluster gets an individual set of mixed-effects regression parameters: intercept, slope, and residual variance. We see that in the second cluster the average pitch frequency (informal attitude is used as reference category) is clearly higher than in the first cluster. The slopes are negative meaning that the average pitch decreases when the participants switch from informal to polite (in cluster 2 somewhat more than in cluster 1). Let us evaluate this solution by cross-classifying with gender:

```
table(Cluster = clusters(koreamix),
      Gender = na.omit(KoreanSpeech)$gender)
##         Gender
## Cluster  F  M
##       1  0 41
##       2 42  0
```

We see that the algorithm is pretty smart and the obtained clusters separate perfectly between males and females: cluster 1 consists of all the male measurements and cluster 2 of all the female ones.

Internally, the packages uses the `FLXMRlmm` driver which allows us to incorporate a single random effect only. If multiple random effects are needed, the `FLXMRlmer` driver can be used which uses an **lme4** style specification. For GLMs, the `FLXMRglm` driver does the job. The corresponding help files give details on how to specify these models.

The final example, where we fit a GAM mixture, is fairly advanced. GAMs are a nonlinear regression framework which typically uses splines or smoothers in order to achieve a nonlinear fit. This model class is described in detail in Wood (2017). We use a simple dataset from the **gamair** package from the same author which includes 2D spatial voxel coordinates (predictors in our model) and a brain activity measure (median of three replicate "fundamental power quotient" (FPQ) values at the voxel) as response. Let us exclude some outliers and plot the activation map (see Fig. 12.7):

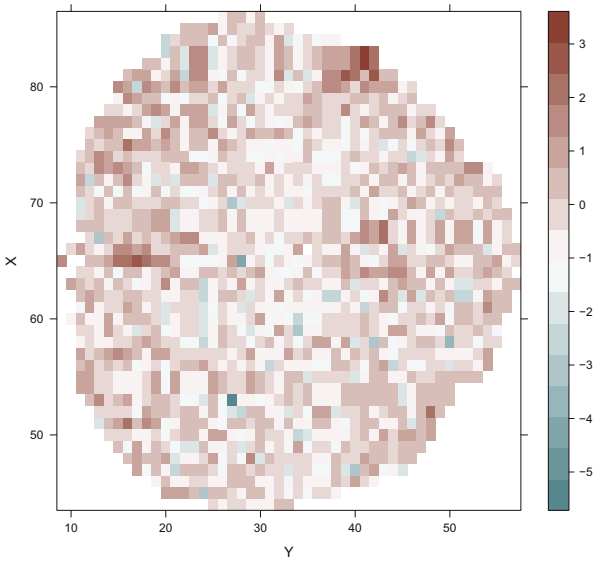

Fig. 12.7 Brain activation levelplot. Levels are colored according to the brain activation as reflected by the FPQ (log-scale)

```
library("gamair")
library("lattice")
data("brain")
brain <- brain[brain$medFPQ > 5e-3,] ## exclude outliers
trellis.par.set(regions = list(col = colorRampPalette(
  c('cadetblue4', 'white', 'coral4'))))
levelplot(log(medFPQ) ~ Y*X, data = brain)
```

Let us fit a mixture GAM using $K = 3$ components. The **flexmix** package offers the `FLXMRmgcv` driver in order to interact with the **mgcv** package for fitting GAMs. We use **mgcv**'s default smoother with a fairly low basis dimension k. Details on this smoother can be found in Wood (2017, Chapters 5 and 7).

```
set.seed(123)
fitgammix <- flexmix(log(medFPQ) ~ s(Y, X, k = 30),
                     model = FLXMRmgcv(), k = 3, data = brain,
                     control = list(tolerance = 10^-3))
table(clusters(fitgammix))
##
##   1   2   3
## 450 870 244
```

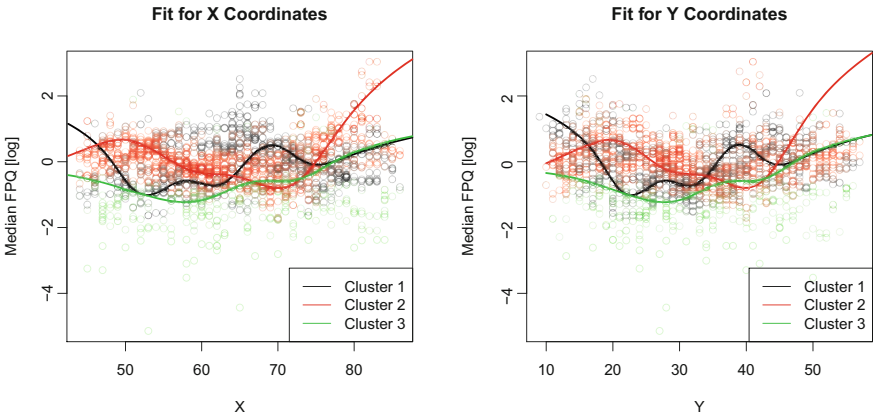

Fig. 12.8 Scatterplots with fitted nonlinear regression functions ($K = 3$). Left panel: voxel x-coordinates. Right panel: voxel y-coordinates

The last line extracts the hard voxel cluster memberships. Figure 12.8 shows the fitted mixture GAM functions. We see that for each predictor we fit a smooth nonlinear function into the scatterplot.

This concludes the section on mixture regression models. Note that the **flexmix** package can also handle multiple input regression models and allows users to include concomitant variables. Examples and further details can be found in Leisch (2004) and Grün and Leisch (2008).

12.3 Dirichlet-Based Clustering

In all approaches presented so far, we had to fix the number of clusters K a priori. Recent developments, within a Bayesian estimation framework, make it possible to infer the number of clusters directly from the data. This idea is called *Dirichlet process mixture* (DPM) model, sometimes also coined as *Chinese restaurant process*. The math behind these models can get pretty technical and is omitted here; a good introduction to this topic at a moderate technical level is given in Gershman and Blei (2012). An R package that implements this approach for MRI image segmentation is **dpmixsim** (da Silva, 2012).

In this section we present two Dirichlet-based approaches. In the first part we continue the mixture regression example from above and let the algorithm determine the number of clusters. In the second part we present an approach designed to cluster text data.

12.3.1 Dirichlet Process Regression

Even though technically different, the conceptual idea of a *Dirichlet process regression* is exactly the same as in the mixture regression approach presented in Sect. 12.2. The difference, from a practicioner's point of view, is that we do not have to specify the number of clusters a priori. Formal details on these models can be found in Shotwell (2013) and are implemented in the **profdpm** package.

We use once more the Korean speech data with the same mixed-effects model specification as above. Note that technically we are not really fitting a mixed-effects model here. Rather, we specify a group argument reflecting which observations are grouped according to our repeated measurement design. This way the function makes sure that multiple observations from a single person end up in the same cluster.

```
library("profdpm")
koreamix2 <- profLinear(frequency ~ attitude, group = subject,
                        data = na.omit(KoreanSpeech))
koreamix2$m           ## intercept and slope parameters
## [[1]]
## [1] 260.67200 -27.38498
##
## [[2]]
## [1] 161.787867  -8.079553
##
## [[3]]
## [1] 109.85354 -15.36563
```

The algorithm finds three clusters with corresponding intercept (first element) and slope parameters (second element). Let us evaluate the solution once more with gender, before interpreting the parameters:

```
table(Cluster = koreamix2$clust,
      Gender = na.omit(KoreanSpeech)$gender)
##         Gender
## Cluster  F  M
##       1 42  0
##       2  0 27
##       3  0 14
```

All the females end up in the first cluster, whereas for the males we get a more fine-grained separation across two clusters. By considering the corresponding intercept and slope parameters for these two clusters, we see that cluster 2 consists of males with a higher average pitch frequency for which the pitch decreases only

slightly when switching from informal to polite. Cluster 3 consists of a single male only, since we had 14 measurements per person. His pitch frequency is lower than the one of the cluster 2 members and decreases more drastically when switching from informal to polite.

The **profdpm** package also includes the `profBinary` function which allows for a Dirichlet process extension of LCA (binary input variables only, no covariates allowed). The number of clusters is determined by the algorithm. Another package, more specifically designed for longitudinal designs and thus actually fitting Dirichlet process mixed-effects models, is **growcurves** (Savitsky and Paddock, 2014) with the `dpgrow` function at its core.

12.3.2 Clustering Texts: Topic Models

Analyzing text data is of increasing interest in psychology. The field concerned with text analysis is often called *natural language processing* (NLP). The R environment offers numerous packages for NLP.[3] Introductory R books on this topic are Kwartler (2017) and Silge and Robinson (2017).

In Sect. 7.1.3 we presented an application of text data by fitting a correspondence analysis on a document-term matrix (DTM) in order to explore word associations. In this section we present *topic models* which aim to find *topics*, that is, probabilistic collections or clusters of words that co-occur in a meaningful way. At the same time, we cluster the rows of the DTM (e.g., a statement by a single individual) in a model-based way by assigning them to topics in a probabilistic way. The particular topic model approach we focus on in this section is called *latent Dirichlet allocations*, which is a special flavor of Dirichlet-based clustering. However, as opposed to the approach in the previous subsection, the number of clusters (i.e., topics) K needs to be fixed a priori.

Before showing how to fit topic models, we demonstrate how to get a raw text file in shape for NLP analysis in R. The data we use are taken from Mair et al. (2014), who scraped self-reported statements of Republican voters from the Republican website. The voters had to complete the sentence "I am a Republican because ..." (254 voters in total). First, we import the raw statements into R and make the voters a bit more sparkling by adding some fake names using the **randomNames** package (Betebenner, 2017).

[3]For an overview see the corresponding task view on CRAN (URL: https://cran.r-project.org/web/views/NaturalLanguageProcessing.html).

```
library("MPsychoR")
library("randomNames")
gopraw <- readLines(paste0(path.package("MPsychoR"),
                           "/GOPstatements.txt"))
set.seed(123)
rnames <- randomNames(length(gopraw), which.names = "first",
  ethnicity = 5, sample.with.replacement = FALSE)
names(gopraw) <- rnames
```

Second, we convert these statements into a corpus and perform some basic cleanup using the **tm** package (Feinerer et al., 2008): convert all words to lowercase, remove stopwords (including some user defined ones), remove numbers and punctuation, and strip white space. These steps reflect a standard text data preparation pipeline. Sometimes *stemming* or *lemmatization*, which reduces inflectional forms of a word to a common base form, is performed at this stage as well.

```
library("tm")
library("tidyr")
myStopwords <- c("beleive", "shld", "-", "wenot", "etc", "im",
  "conservatismthe", "fatherthe", "conservativebelieve",
  "governmentprolife2nd", "amendmentand", "valuessmall", "ive",
  "familyrepublican", "government", "1st", "dont", "get",
  "given", "people", "better", "system", "always", "enough",
  "yet", "hand")
gopcorp <- Corpus(VectorSource(gopraw)) %>%
  tm_map(content_transformer(tolower)) %>%
  tm_map(removePunctuation) %>%
  tm_map(removeWords, c(stopwords("english"), myStopwords)) %>%
  tm_map(removeNumbers) %>%
  tm_map(stripWhitespace)
```

In order to get a first overview of the words used by the voters, we produce a word cloud using the **wordcloud** package (Fellows, 2014, see Fig. 12.9).

```
library("wordcloud")
set.seed(1)
wordcloud(gopcorp, colors = brewer.pal(8, "Dark2"),
          min.freq = 3, random.order = FALSE)
```

The next step is to create the DTM:

```
DTmat <- DocumentTermMatrix(gopcorp)
dim(DTmat)
## [1] 254 725
```

A DTM is simply a frequency table with the voters ("documents") in the rows and the words ("terms") in the columns. It reflects how often each word was mentioned in a voter's statement. In most applications, this table is sparse (i.e., it contains many 0's).

We may not want to use all of the 725 words for clustering, but only select the most important ones. This can be achieved by computing the *term frequency-inverse document frequency* (tf-idf) which gives us an importance measure for each word. Using functions from **slam** (Hornik et al., 2016), it can be computed as follows (see Grün and Hornik, 2011):

Fig. 12.9 Word cloud of Republican voters completing the sentence "I am a Republican because"

```
library("slam")
tfidf <- tapply(DTmat$v/row_sums(DTmat)[DTmat$i], DTmat$j,
  mean) * log2(nDocs(DTmat)/col_sums(DTmat > 0))
DTmat2 <- DTmat[, tfidf >= median(tfidf)]
```

We keep 50% of the most important words for subsequent analysis, organized as a new DTM. As a final data preparation step, we eliminate voters who did not mention any of the words encoded in the reduced DTM.

```
ind <- which(rowSums(as.matrix(DTmat2)) > 0)
DTmat2 <- DTmat2[ind, ]
dim(DTmat2)
## [1] 195 385
```

This leaves us with 195 voters and 385 terms. As mentioned above, the number of topics needs to be specified a priori. One option to determine a reasonable K is through cross-validation (see Ellis, 2017, for an example). Another option is to use the **ldatuning** package (Murzintcev, 2016) which computes various metrics for different K-values. We vary K from 2 to 30, compute four metrics, and plot their trajectories along K (see Fig. 12.10).

```
library("ldatuning")
Ktopics <- FindTopicsNumber(DTmat2, topics = 2:30,
   metrics = c("Arun2010", "CaoJuan2009", "Griffiths2004",
               "Deveaud2014"))
FindTopicsNumber_plot(Ktopics)
```

Unfortunately, we do not get a strikingly clear picture; a K in the range from 6 to 11 seems to be reasonable. Let us use $K = 6$ since the metrics in the lower panel intersect at this value. However, the user easily consider a slightly larger (or lower) K, depending on the interpretability of the solution.

To fit the topic model, we use the `LDA` function implemented in the **topicmodels** package (Grün and Hornik, 2011)[4] and use MCMC (Gibbs sampler) to estimate the model.

```
library("topicmodels")
K <- 6
goplda <- LDA(DTmat2, k = K, method = "Gibbs",
  control = list(seed = 123, iter = 50000, burnin = 1000))
```

The words as well as the voters are assigned to the topics in a probabilistic way.

[4]Other packages for topic modeling in R are **mallet** (Mimno, 2013) and **lda** (Chang, 2015).

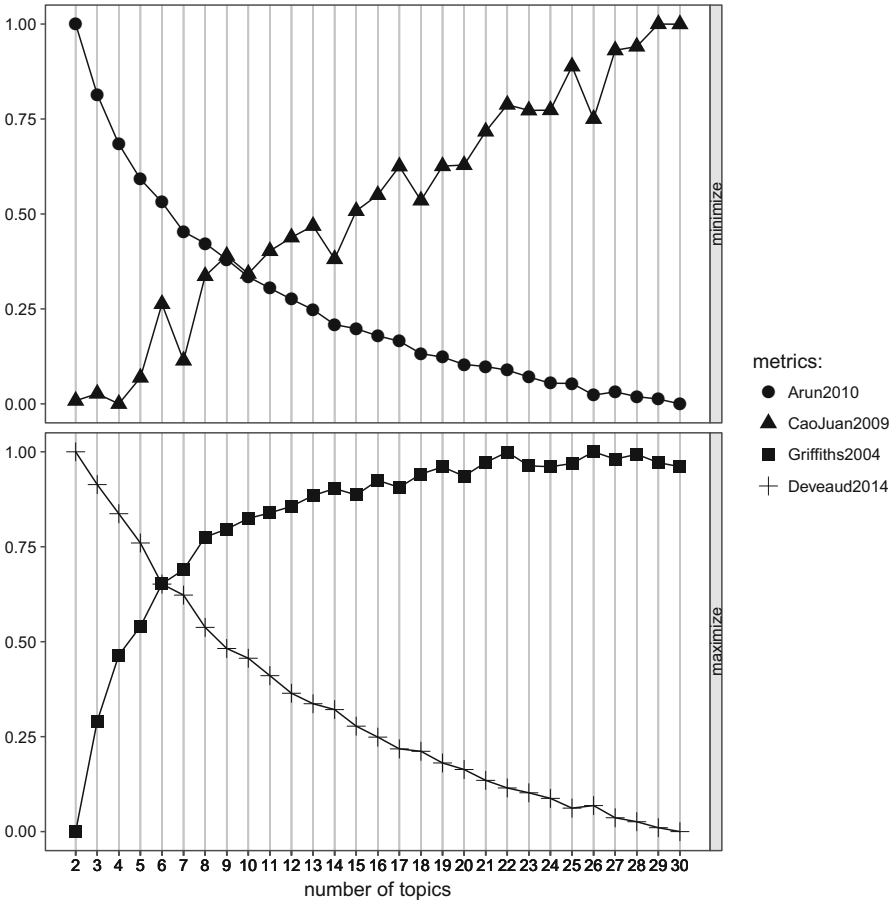

Fig. 12.10 Four metrics for selecting the number of topics K in the GOP document-term matrix. Top panel: Arun and Cao-Juan metrics should be minimized. Bottom panel: Griffiths and Deveaud metrics should be maximized

```
postprob <- posterior(goplda)
pterms <- as.data.frame(t(postprob$terms))
round(head(pterms, 10), 4)
##                    1      2      3      4      5      6
## creativity    0.0009 0.0008 0.0009 0.0009 0.0079 0.0008
## power         0.0099 0.0008 0.0182 0.0009 0.0007 0.0088
## believed      0.0009 0.0008 0.0009 0.0182 0.0007 0.0008
## everything    0.0009 0.0008 0.0095 0.0009 0.0079 0.0008
## started       0.0009 0.0088 0.0009 0.0009 0.0079 0.0008
## blackamerican 0.0009 0.0008 0.0009 0.0009 0.0079 0.0008
```

(continued)

```
## emancipation  0.0009 0.0088 0.0009 0.0009 0.0007 0.0008
## proclamation  0.0009 0.0008 0.0009 0.0009 0.0007 0.0088
## firmly        0.0009 0.0008 0.0009 0.0009 0.0151 0.0008
## minimally     0.0099 0.0008 0.0009 0.0009 0.0007 0.0008
```

This printout shows the posterior probabilities for ten words related each of the six topics. The probabilities represent weights, that is, how important a term is in a particular topic. We use the terms and the weights to produce separate word clouds for each topic (see Ellis, 2017). The six topic clouds are given in Fig. 12.11. Topic 1 is the patriotism cluster containing words such as love (in the sense of loving the country), protect, and power. Topic 2 seems to be the economy cluster (capitalism, fiscal conservatism). Note that "power" appears in this cluster as well. This is an important feature of topic models since words are assigned to topics in a probabilistic way; therefore, they can appear in multiple clusters with varying importance. In topics 3 and 5, the word "responsibility" is of key importance.

A printout of the top five terms in each topic can be obtained as follows:

```
terms(goplda, 5)
##      Topic 1    Topic 2      Topic 3       Topic 4     Topic 5
## [1,] "love"     "capitalism" "moral"       "run"       "live"
## [2,] "morals"   "fiscally"   "think"       "platform"  "responsible"
## [3,] "care"     "true"       "equal"       "believed"  "problems"
## [4,] "mine"     "children"   "power"       "local"     "actions"
## [5,] "godgiven" "suffer"     "responsible" "beautiful" "successful"
##      Topic 6
## [1,] "solution"
## [2,] "honest"
## [3,] "preserving"
## [4,] "giving"
## [5,] "peace"
```

For a fancy visualization of the topics, the **LDAvis** package (Sievert and Shirley, 2015) can be used which also performs multidimensional scaling on the topics (see Gandrud, 2017).

The hard assignments of voters to topics can be extracted as follows (only the first six voters are shown here):

```
topics(goplda)[1:6]
##    Jonica     Nolan Elizabeth   Brianna   Zachary    Hannah
##         3         3         2         1         5         5
```

Voters who used similar words in their statements end up in the same cluster.

Fig. 12.11 GOP voter topics: for each topic a word cloud is plotted. The posterior probabilities are used as weights for the word size

Note that the `LDA` function call from above assumes that the topics are uncorrelated. For correlated topic models, `LDA` can be replaced by `CTM` (see Grün and Hornik, 2011, for details). At the time this book was written, R had no implementation of a hierarchical Dirichlet process for topic modeling where the number of topics does not have to be specified in advance.

References

Betebenner, D. W. (2017). **randomNames**: Function for generating random names and a dataset. R package version 0.4-0. https://cran.r-project.org/package=randomNames

Chang, J. (2015). **lda**: Collapsed Gibbs sampling methods for topic models. R package version 1.4.2. https://CRAN.R-project.org/package=lda

Charrad, M., Ghazzali, N., Boiteau, V., & Niknafs, A. (2014). **NbClust**: An R package for determining the relevant number of clusters in a dataset. *Journal of Statistical Software, 61*(6), 1–36. http://www.jstatsoft.org/v61/i06/

da Silva, A. F. (2012). **dpmixsim**: Dirichlet process mixture model simulation for clustering and image segmentation. R package version 0.0-8. https://CRAN.R-project.org/package=dpmixsim

Ellis, P. (2017). Cross-validation of topic modelling. R-bloggers. http://www.r-bloggers.com/cross-validation-of-topic-modelling/

Everitt, B. S. (2011). *Cluster analysis* (5th ed.). New York: Wiley.

Everitt, B., & Hothorn, T. (2011). *An introduction to applied multivariate analysis with R*. New York: Springer.

Feinerer, I., Hornik, K., & Meyer, D. (2008). Text mining infrastructure in R. *Journal of Statistical Software, 25*(5), 1–54. http://www.jstatsoft.org/v25/i05/

Fellows, I. (2014). **wordcloud**: Word clouds. R package version 2.5. https://CRAN.R-project.org/package=wordcloud

Fraley, C., & Raftery, A. E. (2002). Model-based clustering, discriminant analysis and density estimation. *Journal of the American Statistical Association, 97*, 611–631.

Frick, H., Strobl, C., Leisch, F., & Zeileis, A. (2012). Flexible Rasch mixture models with package **psychomix**. *Journal of Statistical Software, 48*(7), 1–25. http://www.jstatsoft.org/v48/i07/

Gandrud, C. (2017). A link between topicmodels LDA and **LDAvis**. R-bloggers. https://www.r-bloggers.com/a-link-between-topicmodels-lda-and-ldavis/

Gershman, S. J., & Blei, D. M. (2012). A tutorial on Bayesian nonparametric models. *Journal of Mathematical Psychology, 56*, 1–12.

Grün, B., & Hornik, K. (2011). **topicmodels**: An R package for fitting topic models. *Journal of Statistical Software, 40*(13), 1–30. https://doi.org/10.18637/jss.v040.i13

Grün, B., & Leisch, F. (2008). **FlexMix** version 2: Finite mixtures with concomitant variables and varying and constant parameters. *Journal of Statistical Software, 28*(4), 1–35. http://www.jstatsoft.org/v28/i04/

Haegeli, P., Gunn, M., & Haider, W. (2012). Identifying a high-risk cohort in a complex and dynamic risk environment: Out-of-bounds skiing—An example from avalanche safety. *Prevention Science, 13*, 562–573.

Hornik, K., Meyer, D., & Buchta, C. (2016). **slam**: Sparse lightweight arrays and matrices. R package version 0.1-40. https://CRAN.R-project.org/package=slam

Koller, I., & Alexandrowicz, R. W. (2010). Eine psychometrische Analyse der ZAREKI-R mittels Rasch-Modellen [A psychometric analysis of the ZAREKI-R using Rasch models]. *Diagnostica, 56*, 57–67.

Kwartler, T. (2017). *Text mining in practice with R*. New York: Wiley.

Lazarsfeld, P. F., & Henry, N. W. (1968). *Latent structure analysis*. Boston: Houghton Mifflin.

Leisch, F. (2004). **FlexMix**: A general framework for finite mixture models and latent class regression in R. *Journal of Statistical Software, 11*(8), 1–18. https://www.jstatsoft.org/v011/i08

Linzer, D. A., & Lewis, J. B. (2011). **poLCA**: An R package for polytomous variable latent class analysis. *Journal of Statistical Software, 42*(10), 1–29. http://www.jstatsoft.org/v42/i10/

Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., & Hornik, K. (2017). **cluster**: Cluster analysis basics and extensions. R package version 2.0.6.

Mair, P., Rusch, T., & Hornik, K. (2014). The grand old party: A party of values? *SpringerPlus, 3*(697), 1–10.

Mair, P., Hofmann, E., Gruber, K., Zeileis, A., & Hornik, K. (2015). Motivation, values, and work design as drivers of participation in the R open source project for statistical computing. *Proceedings of the National Academy of Sciences of the United States of America, 112*, 14788–14792.

McLachlan, G., & Peel, D. (2000). *Finite mixture models*. New York: Wiley.

Meyer, D., Zeileis, A., & Hornik, K. (2006). The strucplot framework: Visualizing multi-way contingency tables with **vcd**. *Journal of Statistical Software, 17*(3), 1–48. http://www.jstatsoft.org/v17/i03/

Mimno, D. (2013). **mallet**: A wrapper around the Java machine learning tool MALLET. R package version 1.0. https://CRAN.R-project.org/package=mallet

Murzintcev, N. (2016). **ldatuning**: Tuning of the latent Dirichlet allocation models parameters. R package version 0.2.0. https://CRAN.R-project.org/package=ldatuning

Savitsky, T. D., & Paddock, S. M. (2014). Bayesian semi- and non-parametric models for longitudinal data with multiple membership effects in R. *Journal of Statistical Software, 57*(3), 1–35. http://www.jstatsoft.org/v57/i03/

Shotwell, M. S. (2013). **profdpm**: An R package for MAP estimation in a class of conjugate product partition models. *Journal of Statistical Software, 53*(8), 1–18. http://www.jstatsoft.org/v53/i08/

Sievert, C., & Shirley, K. (2015). **LDAvis**: Interactive visualization of topic models. R package version 0.3.2. https://CRAN.R-project.org/package=LDAvis

Silge, J., & Robinson, D. (2017). *Text mining with R: A tidy approach*. Sebastopol: O'Reilly Media.

Winter, B., & Grawunder, S. (2012). The phonetic profile of Korean formality. *Journal of Phonetics, 40*, 808–815.

Wood, S. N. (2017). *Generalized additive models: An introduction with R* (2nd ed.). Boca Raton: CRC Press.

Chapter 13
Modeling Trajectories and Time Series

13.1 Introductory Remarks

Whenever "time" enters an analysis, statistical modeling becomes more challenging since we lose a crucial assumption that many statistical tests and models require to be fulfilled: independence of observations. In modern psychological experiments, participants are often exposed to the same variable or stimulus at multiple points in time. Such designs are often called *repeated measures designs* or *within-subjects designs* (e.g., multiple runs in an fMRI experiment, multiple eye-tracking tasks, pre- and posttreatment clinical assessments). The state-of-the-art analysis framework for such settings are *mixed-effects models* (aka *hierarchical models* aka *multilevel models*). Excellent R books on this topic with special focus on psychological applications are Long (2011) and Mirman (2014), the latter focusing on modeling nonlinear time trajectories (sometimes called *growth curve models*).

Typically, such designs involve a few repeated measurements only. In this chapter we deal with more "heavy duty" longitudinal data settings, that is, a sequence of measurements over time with many measurement points. This type of data is often called *time series*. In the following three sections, we are interested in (a) finding clusters in such trajectories (*hidden Markov models*), (b) exploring time series patterns and predicting future observations (*time series analysis*), and (c) modeling multiple trajectories (*functional data analysis*).

13.2 Hidden Markov Models

Mixture models for parametric clustering, as introduced in Sect. 12.1, assume that the observations are independent from each other. In longitudinal data settings, this assumption is typically violated due to temporal dependencies in a single participant's measurements over time. Thus, we need a dependent version of a

P. Mair, *Modern Psychometrics with R*, Use R!,
https://doi.org/10.1007/978-3-319-93177-7_13

mixture model. Hidden Markov models (HMMs; sometimes also called *latent Markov models*) fall into this model class and integrate two statistical concepts: Markov chains and mixture distributions (latent classes). An excellent tutorial article on HMMs and their use in psychological research is Visser (2011).

13.2.1 Markov Chains

The first important ingredient in HMMs are *Markov chains*. They allow us to describe the behavior of a random variable over time in a probabilistic manner. Markov chains involve *states* (e.g., if we model the weather, states could be "sunny," "cloudy," and "rainy") and aim to model transition probabilities between the states assuming the following critical property (called the *Markov assumption*): the current state at time t depends only upon the previous state $t-1$ and not on other past states. Formally, $P(S_t|S_1, S_2, \ldots, S_{t-1}) = P(S_t|S_{t-1})$. Within the context of Markov chains, these conditional probabilities are called *transition probabilities* and can be expressed as

$$a_{ij} = P(S_{t+1} = j|P(S_t = i)). \tag{13.1}$$

That is, given that someone is in state i at time t, what is the probability for switching to state j at $t + 1$.

Let us illustrate this concept by means of a simple toy example. We aim to model what a student goes through during statistics lecture. The four states we consider are "bored," "horrified," "sleeping," and "somewhat awake."[1] For these four states, we can compute 16 transition probabilities. They can be calculated empirically if, for instance, every 5 min during a 90 min class each student has to report the state he/she is in. We can organize these probabilities as a 4×4 matrix $\mathbf{A}$ with elements a_{ij}. Let us create such a *transition matrix* in R using the **markovchain** package (Spedicato, 2017):

```
library("markovchain")
mcStats <- new("markovchain",
  state = c("bored", "horrified", "sleeping", "somewhat awake"),
  transitionMatrix = matrix(c(0.60, 0.05, 0.25, 0.10,
                              0.15, 0.30, 0.10, 0.45,
                              0.10, 0.10, 0.50, 0.30,
                              0.30, 0.10, 0.50, 0.10),
                              byrow = TRUE, nrow = 4),
```

(continued)

[1] We omit any positive feelings here since this is simply not realistic.

```
  name = "StatsClass")
print(mcStats)
##                bored horrified sleeping somewhat awake
## bored           0.60      0.05     0.25           0.10
## horrified       0.15      0.30     0.10           0.45
## sleeping        0.10      0.10     0.50           0.30
## somewhat awake  0.30      0.10     0.50           0.10
```

The rows of this matrix represent "from" (i.e., the state at time t), whereas the columns represent "to" (i.e., the state at time $t+1$). For instance, if a student is bored at time t, the probability of remaining bored at time $t+1$ is 0.60, the probability of switching to horrified is 0.05, etc. Within each row the probabilities sum up to 1, as we can easily check:

```
A <- mcStats@transitionMatrix
rowSums(A)
##          bored      horrified       sleeping somewhat awake
##              1              1              1              1
```

Within each column they are not required to sum up to 1.

Note that the transition matrix $\mathbf{A}$ does not depend on t (*stationarity assumption*). In other words, it does not matter at which time point t we want to study a transition, these probabilities are constant over time. We can visualize this matrix using the **qgraph** package (Epskamp et al., 2012), resulting in Fig. 13.1.

```
library("qgraph")
qgraph(A, edge.labels = TRUE, edge.color = "black")
```

What can we do with this matrix? First, we can make predictions beyond a simple "from t to $t+1$" transition as encoded in $\mathbf{A}$. For a given state vector $\boldsymbol{\pi}$, we can compute state predictions for 10 min later (i.e., $t+2$), 15 min later (i.e., $t+3$), etc. Let us assume that the student is "horrified" which defines the following initial state vector:

```
pi <- c(0, 1, 0, 0)
```

After 10 and 15 min, respectively, the state probability predictions for this students are:

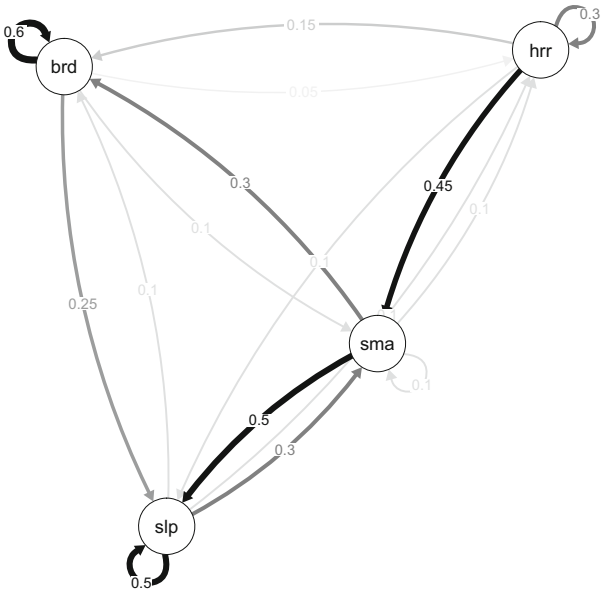

Fig. 13.1 Transition probabilities of states (bored, sleeping, horrified, somewhat awake) students go through during a statistics class

```
pi * mcStats^2  ## 10 minutes later
##      bored horrified sleeping somewhat awake
## [1,]  0.28    0.1525   0.3425          0.225
pi * mcStats^3  ## 15 minutes later
##         bored horrified sleeping somewhat awake
## [1,] 0.292625    0.1165    0.369       0.221875
```

Given a student is horrified, the probability that the student will sleep in 10 or 15 min is considerably high. In the 10 min forecast, this is due to the fact that there is a high probability of switching from horrified to somewhat awake within the first 5 min and a high probability of switching from somewhat awake to sleeping within the next 5 min segment.

It might be of interest in which state a student ends up in the long run, independently from his/her initial state. The resulting probability vector **p** is called the *stationary distribution* and can be computed as follows:

```
p <- steadyStates(mcStats)       ## limiting distribution
p
```

(continued)

```
##          bored horrified  sleeping somewhat awake
## [1,] 0.2962246  0.106486 0.3833495        0.21394
```

This stationary distribution fulfills the equation $\mathbf{p} = \mathbf{pA}$, which we can check as follows:

```
p %*% A
##          bored horrified  sleeping somewhat awake
## [1,] 0.2962246  0.106486 0.3833495        0.21394
```

This results in **p** again. The not-so-surprising message in this example is that students most likely end up sleeping in the long run, independent from their initial states.

This is all we need to know about Markov chains in order explain HMMs. Examples of Markov chains applications in psychology can be found in Wickens (1982), Bornstein and Daw (2012), and Griffiths et al. (2007).

13.2.2 Simple Hidden Markov Modeling Strategies

In Markov chains the states are known, whereas in HMMs the states are "hidden" (i.e., unknown, latent) and need to be found through parametric clustering. Therefore, HMMs integrate the concepts of transition probabilities (from Markov chains) and mixture distributions (see Sect. 12.1) and are sometimes referred to as *dependent mixture models* (Visser and Speekenbrink, 2010). By fixing number of states K a priori, HMMs can be fitted using the EM (expectation-maximization) algorithm. The main output components are:

- the parameters of the mixture distribution,
- the transition probabilities among the K states.

In terms of distributional families for the mixture specification, standard generalized linear model (GLM) distributions such as normal, gamma, binomial, Poisson, etc. can be used.

Let us start with a real-life HMM application. We use data collected within the context of the *implicit association test* (IAT; Greenwald and Banaji, 1995; Greenwald et al., 1998). The IAT measures differential association of two target concepts with an attribute. Here we use the "face IAT". During the experiment participants saw images of people with long and wide faces, as well as positively and negatively valenced words. In the first critical block ("congruent block"), participants were asked to press a response key if they saw a long-faced person/positive word and a

different response key if they saw a wide-faced person/negative word. In the second critical block ("incongruent block"), the pairing was reversed. IAT theory states that participants are expected to be able to respond fast in the congruent condition and slow in the incongruent condition. The response variable used here is the response time latency in ms.

The dataset below includes responses of four participants. Each participant was exposed to 80 trials: 40 congruent block trials, followed by 40 incongruent block trials. For such settings, where each individual produces his/her own trajectory, an HMM is typically fitted for each person individually.

The goal of the first analysis is to find latency clusters, ignoring the condition. In a perfect world, we can expect to get two states corresponding to the blocks: the participant remains in the first state for the first 40 trials and then switches to the second state for the remaining 40 trials. Let us consider the first participant and fit an HMM using the **depmixS4** package (Visser and Speekenbrink, 2010). Note that

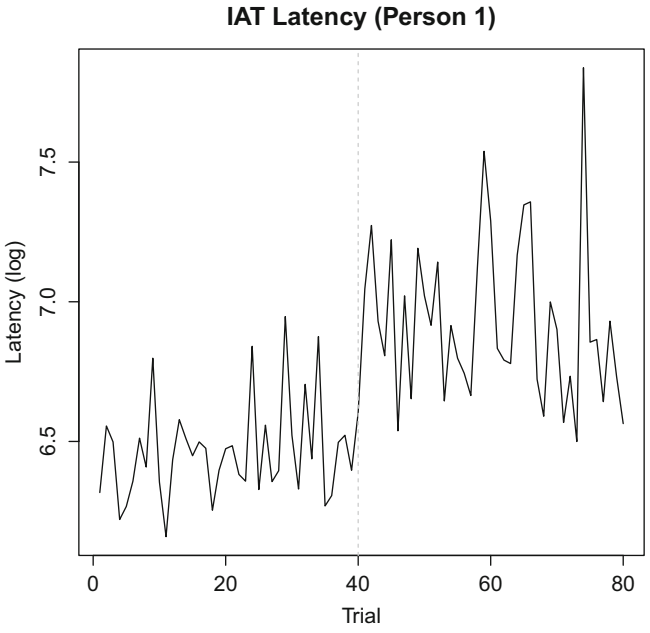

Fig. 13.2 IAT latency trajectory for first participant. The vertical, dashed line denotes the condition switch (congruent to incongruent block)

here we take the log of the latency response in order to achieve a "healthier"-looking distribution with respect to normality, since we are going to fit a Gaussian HMM. The first person's trajectory, extracted below, is plotted in Fig. 13.2.

```
library("MPsychoR")
library("depmixS4")
data("iatfaces")
p1dat <- subset(iatfaces, id == 1)
```

In the following code chunk we fit a sequence of HMMs with varying numbers of states K (from 1 to 4). For each model, the first line defines the `depmix` object, and the second line fits the actual HMM.

```
set.seed(123)
p1obj1 <- depmix(log(latency) ~ 1, data = p1dat, nstates = 1)
p1fit1 <- fit(p1obj1)
p1obj2 <- depmix(log(latency) ~ 1, data = p1dat, nstates = 2)
p1fit2 <- fit(p1obj2)
p1obj3 <- depmix(log(latency) ~ 1, data = p1dat, nstates = 3)
p1fit3 <- fit(p1obj3)
p1obj4 <- depmix(log(latency) ~ 1, data = p1dat, nstates = 4)
p1fit4 <- fit(p1obj4)
```

In terms of goodness-of-fit assessment, the models can be compared via the usual BIC strategy:

```
c(BIC(p1fit1), BIC(p1fit2), BIC(p1fit3), BIC(p1fit4))
## [1] 60.28411 27.59495 43.90186 76.55090
```

The two-state solution provides clearly the best fit. We get the following results:

```
summary(p1fit2)
## Initial state probabilties model
## pr1 pr2
##   1   0
##
## Transition matrix
##         toS1  toS2
## fromS1 0.975 0.025
## fromS2 0.000 1.000
##
## Response parameters
## Resp 1 : gaussian
##     Re1.(Intercept) Re1.sd
```

(continued)

```
## St1           6.465  0.174
## St2           6.926  0.293
```

The transition matrix reflects the dynamics between the two states. The transition probabilities can be interpreted in the same way as in Markov chains. For the two-state example, **A** has the following structure:

$$\mathbf{A} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}. \tag{13.2}$$

If the participant is in state 1, he/she tends to stay there ($a_{11} = 0.975$). There is a low probability for switching to state 2 ($a_{12} = 0.025$). Once switched to state 2, he/she remains there ($a_{22} = 1$), and there is virtually no chance to go back to state 1 ($a_{21} = 0$).

The HMM fits a state posterior probability for each measurement point. For each of the 80 measurements, we get a probability to be in state 1 and state 2, respectively. Let us print out the observations around the condition switch at time point 41:

```
round(posterior(p1fit2)[35:45, ], 3)
##    state    S1    S2
## 35     1 0.978 0.022
## 36     1 0.998 0.002
## 37     1 0.995 0.005
## 38     1 0.994 0.006
## 39     1 0.997 0.003
## 40     1 0.989 0.011
## 41     2 0.212 0.788
## 42     2 0.000 1.000
## 43     2 0.000 1.000
## 44     2 0.000 1.000
## 45     2 0.000 1.000
```

The first column shows the hard state assignment, the remaining two columns the posterior probabilities for being in state 1 and 2. We see that the probability pattern changes at the 41st measurement point.

The bottom part of the `summary(p1fit2)` output above shows the parameters of the underlying normal distributions. These parameters result from the simple regression equation $Y_t = \mu_i + \varepsilon_t$ (with $i = 1, 2$), since we fixed the number of latent states to $K = 2$. We get two means on the log-latency scale (intercept) and, since we assume that $\varepsilon_t = N(0, \sigma_i^2)$, two standard deviations. Thus, we estimate one set of parameters for each state. In the first state the average latency is much lower (i.e., fast-responding state) than in the second state (i.e., slow-responding state).

In this example we can externally evaluate the hard state assignments by cross-classification with the experimental condition variable.

```
table(state = p1fit2@posterior$state, block = p1dat$block)
##      block
## state congruent incongruent
##     1        40           0
##     2         0          40
```

For this participant the states correspond perfectly to the experimental blocks. In order to explore whether this interpretation applies to the remaining three individuals as well, we can fit corresponding two-state models on the remaining data:

```
set.seed(123)
p2dat <- subset(iatfaces, id == 2)
p2obj <- depmix(log(latency) ~ 1, data = p2dat, nstates = 2)
p2fit <- fit(p2obj)
p3dat <- subset(iatfaces, id == 3)
p3obj <- depmix(log(latency) ~ 1, data = p3dat, nstates = 2)
p3fit <- fit(p3obj)
p4dat <- subset(iatfaces, id == 4)
p4obj <- depmix(log(latency) ~ 1, data = p4dat, nstates = 2)
p4fit <- fit(p4obj)
```

Again, these solutions are evaluated through cross-classification with the block variable:

```
table(state = p2fit@posterior$state, block = p2dat$block)
##      block
## state congruent incongruent
##     1         8          14
##     2        32          26
table(state = p3fit@posterior$state, block = p3dat$block)
##      block
## state congruent incongruent
##     1         1          40
##     2        39           0
table(state = p4fit@posterior$state, block = p4dat$block)
##      block
## state congruent incongruent
##     1        24           0
##     2        16          40
```

Person 3 behaves very similar to person 1. The other two individuals show a somewhat different state pattern, not in line with the experimental condition. For these participants a different choice of K can lead to a better fit, and the states need to be interpreted differently.

13.2.3 Hidden Markov Models with Covariates

The **depmixS4** package is highly flexible and provides several options to extend the basic HMMs from above. One of these options is to include covariates on the prior state probabilities (*concomitant variables*; see Sect. 12.1.4) through a formula specification in the `prior` argument. Another option is to fix and constrain parameters. The package also allows the user to fit HMMs on multiple response variables simultaneously (*multivariate HMM*). Various examples can be found in Visser and Speekenbrink (2010).

In this section we focus on a different extension in terms of specifying covariate effects on the response as well as on the transition probabilities. Let us start with the first scenario by considering the trajectory produced by the first person once more. We illustrate how to include the block variable directly in the HMM specification. Note that so far, block was not included in any model fitting process; we merely used it for external evaluation of the HMM solution. Specifying a one-state HMM with block z_t as effect on log-latency implies fitting a simple regression of the form $Y_t = \mu + \beta_1 z_t + \varepsilon_t$.

```
p1obj2 <- depmix(log(latency) ~ block, data = p1dat, ns = 1)
p1fit2a <- fit(p1obj2, verbose = FALSE)
## converged at iteration 2 with logLik: 0.6253951
summary(p1fit2a, which = "response")
## Response parameters
## Resp 1 : gaussian
##     Re1.(Intercept) Re1.blockincongruent Re1.sd
## St1           6.466                0.464   0.24
lm(log(latency) ~ block, data = p1dat)
##
## Call:
## lm(formula = log(latency) ~ block, data = p1dat)
##
## Coefficients:
##      (Intercept)  blockincongruent
##           6.4658            0.4641
```

The HMM and the `lm` output match. The *sd* parameter in the HMM output corresponds to the residual *sd* in the `lm` fit.

Let us move on with a two-state model, where two regression models for latency on block i are fitted ($i = 1, 2$): $Y_t = \mu_i + \beta_{1i} z_t + \varepsilon_t$. Note that clustering and the regression fit happen simultaneously; thus, we cannot achieve identical results with simple `lm` calls anymore.

```
set.seed(123)
p1obj3 <- depmix(log(latency) ~ block, data = p1dat, ns = 2)
p1fit2b <- fit(p1obj3, verbose = FALSE)
summary(p1fit2b)
## Initial state probabilties model
## pr1 pr2
##   1   0
##
## Transition matrix
##         toS1  toS2
## fromS1 0.605 0.395
## fromS2 0.593 0.407
##
## Response parameters
## Resp 1 : gaussian
##     Re1.(Intercept) Re1.blockincongruent Re1.sd
## St1           6.407                0.333  0.120
## St2           6.573                0.603  0.238
```

The transition probabilities change drastically compared to the two-state model with no covariate as fitted in Sect. 13.2.2. This is due to the fact that the states are fully determined by the condition switch. The transition probabilities are close to a coin toss, since we account for the block effect in the regression specification. Of course, the states have to be interpreted differently compared to the HMM with no covariates. The BIC (one-state covariate model vs. two-state covariate model) suggests that we should go with the one-state model:

```
c(BIC(p1fit2a), BIC(p1fit2b))
## [1] 11.89529 16.60824
```

Again, for the remaining persons, the results (i.e., number of states needed, block effect, transition probabilities, state interpretation) may look quite differently.

Another way of studying the condition effect is to hypothesize that block has a direct influence on the transition probabilities. That is, we model the transition probabilities as a function of the block condition z_t. This time we do not include any direct effect on the latency response. Models with covariates on the transitions are a bit tricker to interpret. Here we illustrate it by using a two-state model for which the **depmixS4** package, internally, establishes the following set of logit equations (see

Visser, 2011):

$$\begin{aligned} \text{logit}(1 - a_{11}) &= \eta_0^{(1)} + \eta_1^{(1)} z_t, \\ \text{logit}(a_{22}) &= \eta_0^{(2)} + \eta_1^{(2)} z_t. \end{aligned} \tag{13.3}$$

A baseline category multinomial logit model is used to quantify effects of z_t on the transition probabilities. Note that $1 - a_{11} = a_{12}$ (i.e., switching from state 1 to state 2) and $1 - a_{22} = a_{21}$ (i.e., switching from state 2 to state 1). The model can be fitted as follows:

```
set.seed(123)
p1obj4 <- depmix(log(latency) ~ 1, data = p1dat, nstates = 2,
                 transition = ~ block)
p1fit2c <- fit(p1obj4,emcontrol = em.control(maxit = 5000))
```

Let us focus on the transition output for which we get one set of parameters for the congruent condition, and one set of parameters for the incongruent condition:

```
summary(p1fit2c, which = "transition")
## Transition model for state (component) 1
## Model of type multinomial (mlogit), formula: ~block
## Coefficients:
##                   St1       St2
## (Intercept)         0 -1.381049
## blockincongruent    0  8.614802
## Probalities at zero values of the covariates.
## 0.7991594 0.2008406
##
## Transition model for state (component) 2
## Model of type multinomial (mlogit), formula: ~block
## Coefficients:
##                   St1       St2
## (Intercept)         0 -9.185617
## blockincongruent    0 20.639862
## Probalities at zero values of the covariates.
## 0.9998975 0.0001024926
```

The output reports the η-parameters from Eq. (13.3) on a logit scale. Where it says "probabilities at zero values of the covariates," the η-parameters are converted into probabilities given the covariate value is 0 (here, congruent block due to dummy coding). The following code chunks illustrate how to achieve this transformation, resulting in two transition matrices. Let us extract the η-parameters and organize them as matrices:

```
eta1 <- matrix(getpars(p1fit2c)[3:6], 2, byrow = TRUE)
eta1
##      [,1]      [,2]
## [1,]    0 -1.381049
## [2,]    0  8.614802
eta2 <- matrix(getpars(p1fit2c)[7:10], 2, byrow = TRUE)
eta2
##      [,1]      [,2]
## [1,]    0 -9.185617
## [2,]    0 20.639862
```

Let us have a closer look at state 1, congruent block condition (baseline category due to dummy coding). We carry out an exponential transformation of η_1, as usual in logistic/multinomial regression models, in order to eventually get a probability interpretation.

```
exps1b1 <- exp(eta1[1,])
a11c <- 1/sum(exps1b1)
a11c
## [1] 0.7991594
```

This value denotes the probability a_{11}. That is, how likely person 1 stays in state 1 when `block=congruent`. This probability, including $1 - a_{11}$, was already shown in the `summary` output above. What happens to this probability when `block=incongruent`? Let us perform the same transformations for this condition:

```
exps1b2 <- exp(eta1[2,])
a11ic <- 1/sum(exps1b2)
a11ic
## [1] 0.0001813678
```

We see that a_{11} becomes almost 0; that is, the person is likely to switch the state. Note that this value was not shown in the `summary` output above.

Now we look at state 2 and perform the same computations. First we look at a_{22} when `block=congruent`:

```
exps2b1 <- exp(eta2[1,])
a22c <- 1-1/sum(exps2b1)
a22c
## [1] 0.0001024926
```

This value and $1 - a_{22}$ match the probabilities in the `summary` output. Finally, we compute a_{22} for `block=incongruent`:

```
exps2b2 <- exp(eta2[2,])
a22ic <- 1-1/sum(exps2b2)
a22ic
## [1] 1
```

The person stays in state 2. We can organize these probabilities as 2×2 transition matrices, one matrix for each condition:

```
Acong <- round(matrix(c(a11c, 1-a11c, 1-a22c, a22c), 2,
                      byrow = TRUE), 5)
Aicong <- round(matrix(c(a11ic, 1-a11ic, 1-a22ic, a22ic), 2,
                       byrow = TRUE), 5)
dimnames(Acong) <- dimnames(Aicong) <- list(
        c("fromS1", "fromS2"), c("toS1", "toS2"))
Acong           ## congruent condition
##           toS1    toS2
## fromS1 0.79916 0.20084
## fromS2 0.99990 0.00010
Aicong          ## incongruent condition
##           toS1    toS2
## fromS1 0.00018 0.99982
## fromS2 0.00000 1.00000
```

These results show again that the first person's state switching behavior is almost fully determined by the block condition. We can interpret state 1 as the "congruent state" and state 2 as the "incongruent state."

We can perform the same computations for the remaining persons. For some of them, the results look quite different, suggesting that here may be variables other than the block condition influencing the state transition behavior.

This concludes the section in HMM. Other options for applying HMMs on time series data are described in Zucchini et al. (2016). A useful HMM extension are *state space models*. In state space models, the latent variables (states) are continuous, whereas in HMMs they are discrete (i.e., clusters). Details on how to fit such models in R are given in Petris and Petrone (2011).

13.3 Time Series Analysis

In time series data, we observe a random variable Y_t over time ($t = 1, \ldots, T$). The trajectories in the previous section on HMMs are examples of time series data. Similar to HMMs, in time series analysis, we typically model a single time trajectory only. Our aim is (a) to describe trajectory patterns (e.g., *trend*, *cycles*, *seasonality*) and (b) to predict (forecast) future values. Time series data can be collected either at regular intervals (*seasonal* data; e.g., months or quarters within a year, hours within a day, etc.) or without such intervals (*nonseasonal* data; e.g., only one observation per year). These intervals can be equally spaced or irregularly spaced.

In this section we focus on regularly spaced seasonal series. However, most of the methodology presented below can be applied to nonseasonal data as well. Time series analysis is widely used in economics and finance, but there has been an increasing interest in psychological applications in recent years (see Jebb et al., 2015). An easy-to-access text on time series analysis using R is Hyndman and Athanasopoulos (2014); more in-depth treatments are given in Cryer and Chan (2008) and Cowpertwait and Metcalfe (2009).

13.3.1 Linear Models and Structural Change Detection

Before focusing on specific time series techniques, we apply simple linear modeling techniques to time series data. Once more we use a dataset related to the implicit association test (IAT). The particular IAT form considered here is the "age IAT", where participants tend to have an implicit preference for young over old people. As response variable, Cohen's d (here we call it d-measure) is used, computed on the basis of the IAT D-measure which divides the difference between congruent and incongruent block means by the pooled standard deviation (Greenwald et al., 2003). The data were collected through the *ProjectImplicit*[2] (Nosek et al., 2002), consisting of monthly d-measures from January 2007 until December 2015. This constitutes a seasonal time series (monthly observations within a year).

Let us import the data vector into R. When working with time series data, the first thing we should do is to convert the data into a `"ts"` object and plot the trajectory in order to get a first impression. Figure 13.3 suggests that there is a decrease in the d-measure starting at around 2011.

[2]See https://implicit.harvard.edu/; data are publicly available on https://osf.io/y9hiq/.

```
library("MPsychoR")
data("ageiat")
yts <- ts(ageiat, start = c(2007, 1), frequency = 12)
plot(yts, ylab = "d-measure", main = "Age IAT Time Series")
```

The most naive modeling strategy is to fit a simple linear regression model with time as predictor and d as response. Below we use the `tslm` function from the

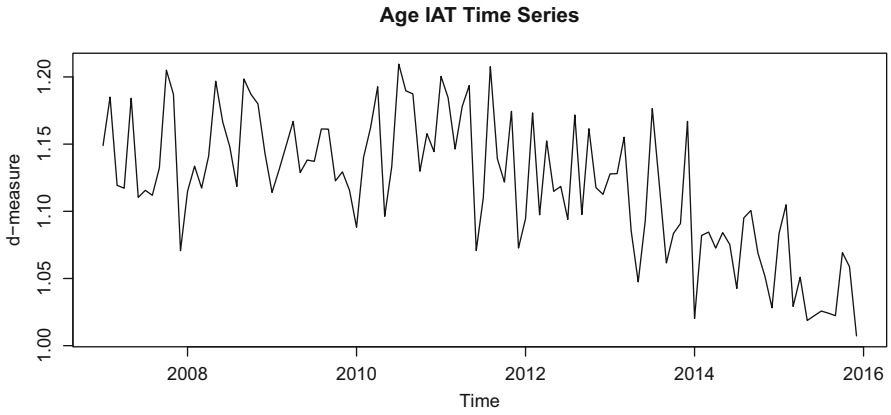

Fig. 13.3 Age IAT time series (monthly observations from 01/2007 to 12/2016)

forecast package (Hyndman and Khandakar, 2008) which is doing the same thing as `lm` but is designed for time series objects. For better interpretability of the intercept, we subtract the starting year 2007 from the data. This way the intercept reflects the fitted d-measure at the point where the series starts.

```
library("forecast")
tr <- time(yts) - 2007
fitlm <- tslm(yts ~ tr)
fitlm
##
## Call:
## tslm(formula = yts ~ tr)
##
## Coefficients:
## (Intercept)           tr
##     1.17423     -0.01211
```

There are at least two problems with such a linear model approach. First, a line is often too simple to capture fluctuating and nonlinear patterns. Second, time series data often violate a crucial regression assumption: uncorrelated residuals. Residual correlation in time series is expressed via the *autocorrelation function* (ACF), which systematically correlates observations (or, within a regression context, residuals) at time t with observations (residuals) from the past $t - l$, where l is the *lag*. The following call produces the *correlogram* (i.e., plotting the autocorrelation for different lags):

```
Acf(residuals(fitlm), main = "ACF of Residuals")
```

From Fig. 13.4 we see that there is a substantial autocorrelation up to a lag of 6 months. We can also run an explicit test for autocorrelation: the *Durbin-Watson test*. By default, the function from the **lmtest** package (Zeileis and Hothorn, 2002)

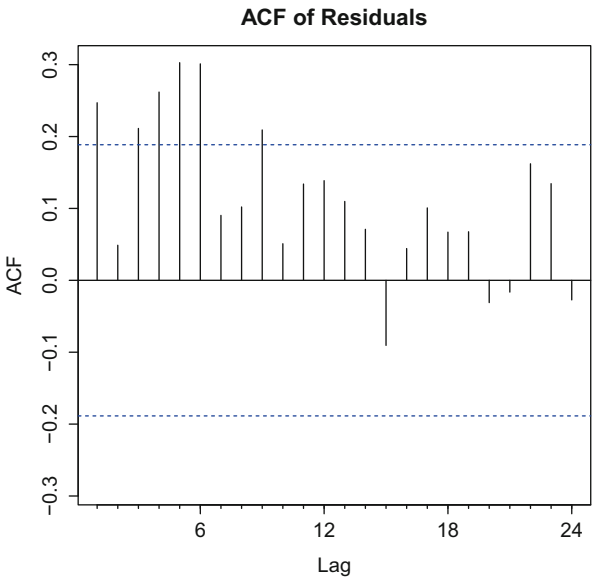

Fig. 13.4 Residual ACF (maximum lag: 24 months). The dashed blue lines show the significance bounds

employs a one-sided test (H_0: no autocorrelation). The closer the DW-value to 2, the better (in the sense of no autocorrelation).

```
library("lmtest")
dwtest(ageiat ~ tr)
##
##  Durbin-Watson test
##
## data:  ageiat ~ tr
## DW = 1.4811, p-value = 0.002393
```

We see that fitting a linear regression is not such a good idea since the residuals are clearly correlated. In addition, a linear trend is too simple to capture the nonlinear trend pattern shown in Fig. 13.3.

A more sophisticated regression modeling option is to fit a *structural change model* which detects time points where the regression line breaks. If the algorithm finds such breakpoints, separate regression parameters are fitted for each segment. The `breakpoints` function from the **strucchange** package (Zeileis et al., 2002) does the job:

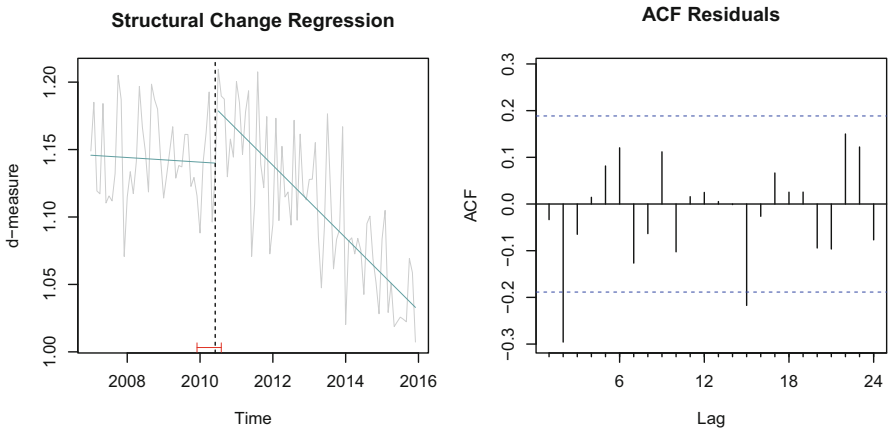

Fig. 13.5 Left panel: regression lines based on structural change detection including CI for the structural break. Right panel: autocorrelation function of the structural change model

```
library("strucchange")
bp <- breakpoints(yts ~ tr)
bp
##
##   Optimal 2-segment partition:
##
```

(continued)

```
## Call:
## breakpoints.formula(formula = yts ~ tr)
##
## Breakpoints at observation number:
## 42
##
## Corresponding to breakdates:
## 2010(6)
round(coef(bp), 3)
##                     (Intercept)     tr
## 2007(1) - 2010(6)         1.146 -0.002
## 2010(7) - 2015(12)        1.273 -0.027
```

The procedure finds a single breakpoint in June/July 2010. The resulting regression lines are plotted in the left panel of Fig. 13.5. From the right panel, we see that this model reduced the residual autocorrelation drastically, but there is still a significant lag-2 autocorrelation, as suggested by the *Box-Pierce test* (H_0: independent observations at fixed lag):

```
Box.test(residuals(bp), lag = 2)
##
##  Box-Pierce test
##
## data:  residuals(bp)
## X-squared = 9.565, df = 2, p-value = 0.008375
```

To conclude, using a simple linear regression model on time series data is, in most applications, too restrictive. Structural change regression is an attractive alternative if the trajectory can be approximated by piecewise linear regression fits. Still, the residual independence assumption needs to be fulfilled.

In the next section, we present a regression modeling framework, specifically developed for time series data, which does not suffer from the same limitations as linear model approaches.

13.3.2 ARIMA Models

ARIMA stands for *autoregressive integrated moving average* and is a general modeling framework for time series data. Before actually fitting the ARIMA model, it is suggested to perform several exploratory steps since, as we will see, the determination of the ARIMA parameters can be quite challenging. A typical time series analysis work flow is the following:

1. additive vs. multiplicative representation;
2. time series decomposition;
3. differencing time series (stationarity);
4. explore autocorrelation (correlogram);
5. fit multiple ARIMA (or other) models and perform model selection;
6. make predictions.

Let us start with step 1: additive vs. multiplicative series. A seasonal time series y_t can be decomposed into a trend component T_t, a seasonal component S_t, and an irregular component E_t (random, white noise)[3]:

- Additive: $y_t = T_t + S_t + E_t$.
- Multiplicative: $y_t = T_t \times S_t \times E_t$

A multiplicative formulation is needed if seasonal (and random) fluctuations increase over time (economic time series often show this behavior). In this case, applying a log transformation $\log(Y_t)$ makes the model additive again. As a simple diagnostic tool, we plot both the untransformed series (often called *level series*) and the log-transformed series (`log(yts)`) and examine whether they look strikingly different. In this example (plot not shown here), there are no obvious differences between the two series.

Now let us decompose the time series into the three components (step 2): trend, seasonality, and noise.

```
tsdec <- decompose(yts)
plot(tsdec)
```

Figure 13.6 shows the resulting decomposition.[4] We see a nonlinear trend in the d-measures, starting to decrease at around 2011. The seasonal component is very small (see y-axis scaling), fluctuating within an interval of $[-0.02, 0.02]$. If we had a strong seasonal component, the time series can be seasonally adjusted:

```
sadj <- yts - tsdec$seasonal
```

At this point we have a good exploratory picture of our time series. The remaining steps are related to various properties of the series, relevant for the ARIMA specification later on.

[3]The `SMA` function in **TTR** (Ulrich, 2017) can be used for decomposition nonseasonal data.

[4]Instead of `decompose` the `stl` function can be used which provides additional decomposition options.

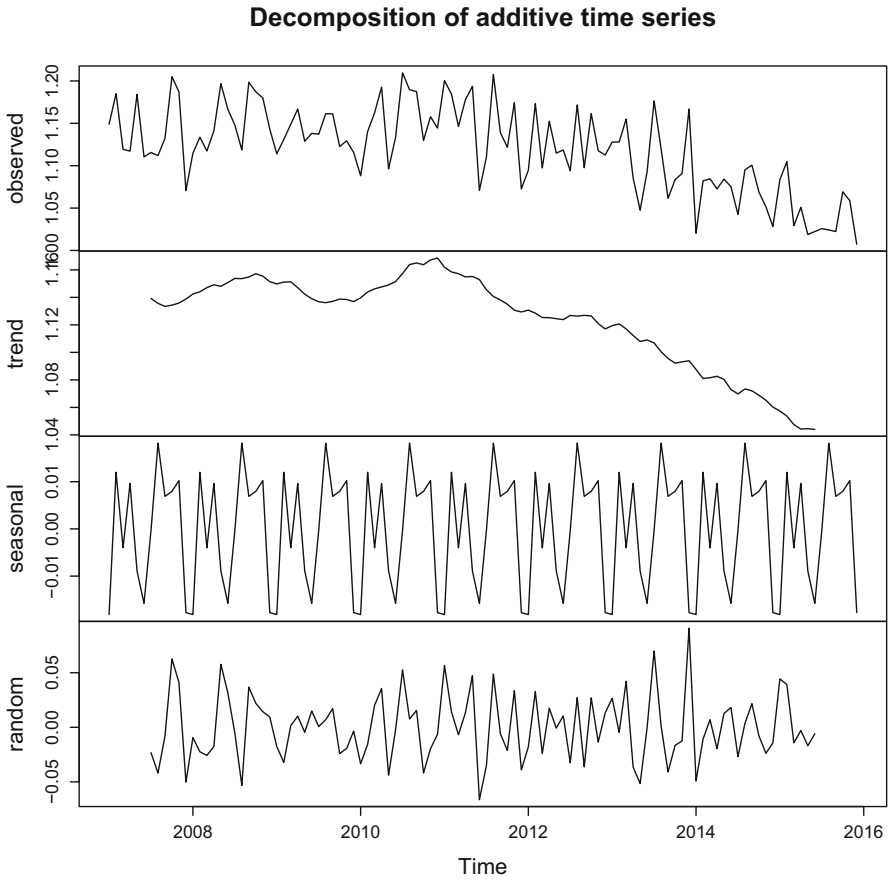

Fig. 13.6 Classical decomposition of the IAT time series: trend, seasonality, noise

Step 3 tackles an important concept in time series data: *stationarity*. A stationary time series is one whose properties do not depend on the time at which the series is observed (Hyndman and Athanasopoulos, 2014, p. 214). Thus, time series with trend and/or seasonality patterns are non-stationary. A stationary time series looks like the white noise series in the bottom panel of Fig. 13.6. If a series in non-stationary, *differencing* (with parameter d) does the trick to make it stationary. First, we compute differences of successive observations ($d = 1$; $y'_t = y_t - y_{t-1}$). Second, we can compute differences based on the first-order differences ($d = 2$; $y''_t = y'_t - y'_{t-1}$). If necessary, higher-order differences can be obtained in the same way. We difference until we reach stationarity. For illustration, let us compute the first-order and second-order differences.

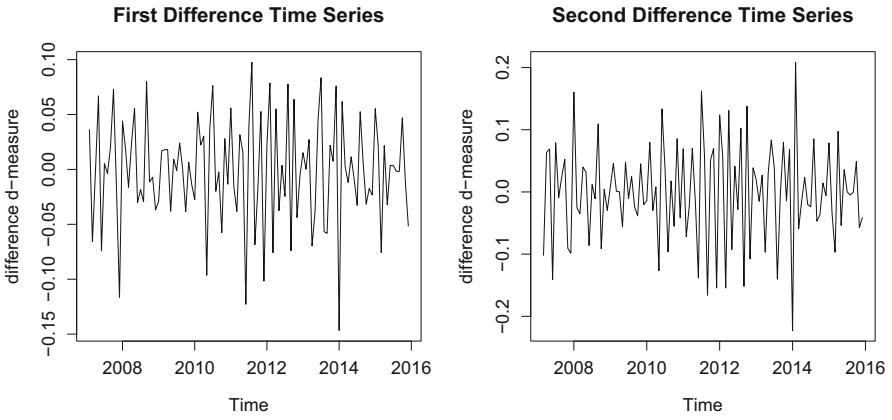

Fig. 13.7 Left panel: first-order differenced series. Right panel: second-order differenced series

```
yts1 <- diff(yts, difference = 1)
yts2 <- diff(yts, difference = 2)
plot(yts1, ylab = "difference d-measure",
     main = "First Difference Time Series")
```

Figure 13.7 shows the resulting time series. The left panel looks already pretty stationary; no obvious trend/seasonality pattern is visible. Thus, there is no need to consider second-order differenced series (right panel). An explicit test for assessing stationarity is the *augmented Dickey-Fuller test* (H_0: non-stationary), as implemented in the **tseries** package (Trapletti and Hornik, 2017):

```
library("tseries")
adf.test(yts)
##
##  Augmented Dickey-Fuller Test
##
## data:  yts
## Dickey-Fuller = -2.2221, Lag order = 4, p-value = 0.4845
## alternative hypothesis: stationary
adf.test(yts1)
##
##  Augmented Dickey-Fuller Test
##
## data:  yts1
## Dickey-Fuller = -9.6341, Lag order = 4, p-value = 0.01
## alternative hypothesis: stationary
```

The first test result tells us that our original time series is non-stationary. We achieved stationarity through first-order differencing, as suggested by the second test result. Thus, we proceed with the first-order differenced version.

In step 4 we explore the autocorrelation structure using the ACF. In Sect. 13.3.1 we computed the ACF for the residuals of a regression model. Here we apply it on

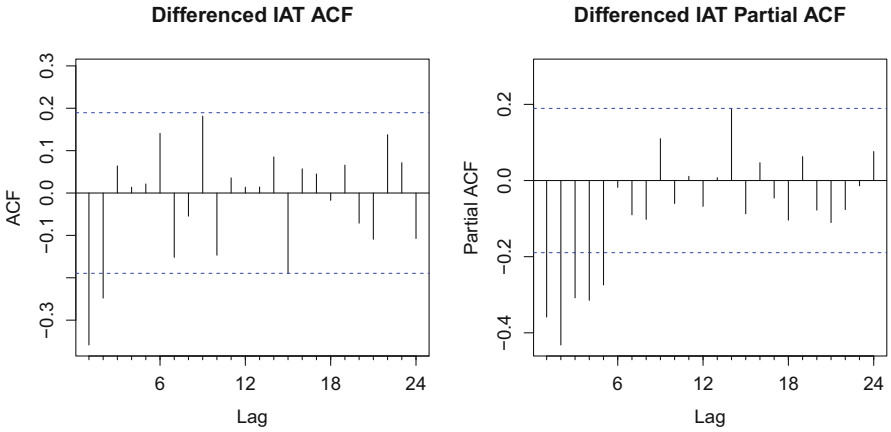

Fig. 13.8 Left panel: autocorrelation function of the first-order differenced time series (correlogram). Right panel: partial autocorrelation function of the first-order differenced time series (partial correlogram)

the differenced observations. In addition, we compute the *partial autocorrelation function* (PACF) which, similar to the ACF, correlates the time series with its own lagged values but controls for the values at all shorter lags.

```
Acf(yts1, main = "Differenced IAT ACF")
Pacf(yts1, main = "Differenced IAT Partial ACF")
```

The plots in Fig. 13.8 are important. Let us focus on the left-hand side first. We see that the ACF drops quickly (i.e., below significance after lag 2). Such a pattern constitutes a *moving average* (MA) model of order $q = 2$ (short, an MA(2) model). The general expression for an MA(q) model is the following[5]:

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}. \tag{13.4}$$

[5]For simplicity in notation, let us ignore for the moment that we differenced the time series and write y_t instead of y'_t.

The ε-terms denote the (lagged) errors and the θ-parameters reflect the influence of the lagged errors on y_t (c is a constant). For our MA(2) example, the expression simplifies accordingly (we need terms up to $q = 2$).

The right panel in Fig. 13.8 shows that it takes a while until the partial ACF drops below significance. From lag six onward, the PACF drops quickly. Such a pattern implies an *autoregressive* (AR) model of order $p = 5$ (short, an AR(5) model). The general AR(p) model formulation is

$$y_t = c + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \varepsilon_t, \tag{13.5}$$

where the ϕ-parameters denote the influence of the lagged observations on y_t. In our particular application, the model expression simplifies accordingly (we need terms up to $p = 5$).

AR(p) and MA(q) models can be combined into a model class which is called ARMA(p, q) (*autoregressive moving average*). This results in

$$y_t = c + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t. \tag{13.6}$$

If the time series needed to be differenced, as in our example, the ARMA model needs to be extended by including differencing parameter d (we say the series is integrated of order d). This results in an ARIMA(p,d,q) model:

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t. \tag{13.7}$$

Note that ARIMA models can be again extended by, for instance, including a seasonal component or adding linear drift parameters.

At this point it is wrong to conclude that, in our application, the model of choice is an ARIMA(5,1,2) model. This is due to the fact that in the ACF initial coefficients depend on the MA part, whereas a later decay is dictated by the AR part. Conversely, in the PACF initial values depend on the AR order, followed by the decay due to the MA part. Based on these considerations, we can specify the following candidate models:

- ARIMA(0,1,2): MA(2) model (left panel Fig. 13.8) with first-order differencing (ignore the AR part).
- ARIMA(5,1,0): AR(5) model (right panel Fig. 13.8) with first-order differencing (ignore the MA part).
- ARIMA(p, 1, q): Combine the AR and MA models with smaller values p and q as in the individual AR and MA models above (e.g., something like an ARIMA(2,1,1)).

With three parameters the most parsimonious model of these three candidates is ARIMA(0,1,2), which can be fitted as follows:

```
iatima <- Arima(yts, order = c(0, 1, 2))
iatima
## Series: yts
## ARIMA(0,1,2)
##
## Coefficients:
##           ma1      ma2
##       -0.7819  -0.0457
## s.e.   0.1359   0.1320
##
## sigma^2 estimated as 0.001406:  log likelihood=199.95
## AIC=-393.89   AICc=-393.66   BIC=-385.87
```

In terms of finding the best model (step 5), we could proceed with fitting an ARIMA(5,1,0) or other ARIMA($p, 1, q$) specifications and pick the model with the lowest AIC/BIC. However, the **forecast** package offers a magic function called `auto.arima` (see Hyndman and Khandakar, 2008, for details) which attempts to find a good candidate model by internally setting up a model sequence (i.e., (p, d, q) parameter combinations and some additional parameter flavors such as seasonal parameters and drift terms) and returns the best model based on the lowest AIC/BIC.

```
iatauto <- auto.arima(yts)
iatauto
## Series: yts
## ARIMA(0,1,1) with drift
##
## Coefficients:
##           ma1    drift
##       -0.8614  -0.0011
## s.e.   0.0426   0.0005
##
## sigma^2 estimated as 0.001363:  log likelihood=201.51
## AIC=-397.02   AICc=-396.78   BIC=-389
```

This search procedure suggests to use a model with a single MA parameter only, differenced once, with a linear *drift* term. Technically, this means that we fit a linear regression with ARIMA errors and with the drift parameter as slope. Since the slope is negative, it suggests a decay of the d-measure across time. Let us compare the ARIMA(0,1,2) from above with the ARIMA(0,1,1) with drift, both of them having two parameters:

```
BIC(iatima, iatauto)[,2]
## [1] -385.8745 -388.9987
AIC(iatima, iatauto)[,2]
## [1] -393.8930 -397.0171
```

The AIC/BIC values for the `auto.arima` model are smaller. Thus, the procedure outsmarts us with our proudly established ARIMA(0,1,2) model. Let us continue with the `auto.arima` model.

First, we can do some residual checks. Figure 13.9 shows some residual diagnostic plots obtained by saying `tsdiag(iatauto)`. As in standard regression analysis, the standardized residuals should fluctuate randomly around 0, which is the case in our example (top panel). Also, the ACF looks pretty healthy (even through there is a slight autocorrelation at lag-2).[6] We can also plot the fitted values of ARIMA model (plot now shown here).

```
plot(iatauto$x, ylab = "D-measure", main = "IAT Time Series")
lines(fitted(iatauto), col = "salmon", lwd = 2)
```

If an R^2 is needed, we can use the squared correlation between the fitted values and the original observations:

```
cor(fitted(iatauto), yts)^2
## [1] 0.4788475
```

Once we have found a model that fits, in time series analysis, we are typically interested in forecasting. This brings us to the final step 6. Based on the ARIMA(0,1,1) model with drift, we are now going to make a forecast until the year 2020.

```
plot(forecast(iatauto, h = 48), ylab = "d-measure",
     main = "Age IAT Forecasts")
```

Figure 13.10 shows the corresponding IAT forecast including the 80% and 95% prediction intervals. We see that implicit age associations will continue to decrease

[6]Note that the first value in the correlogram is the lag-0 correlation which is of course 1 and can be ignored.

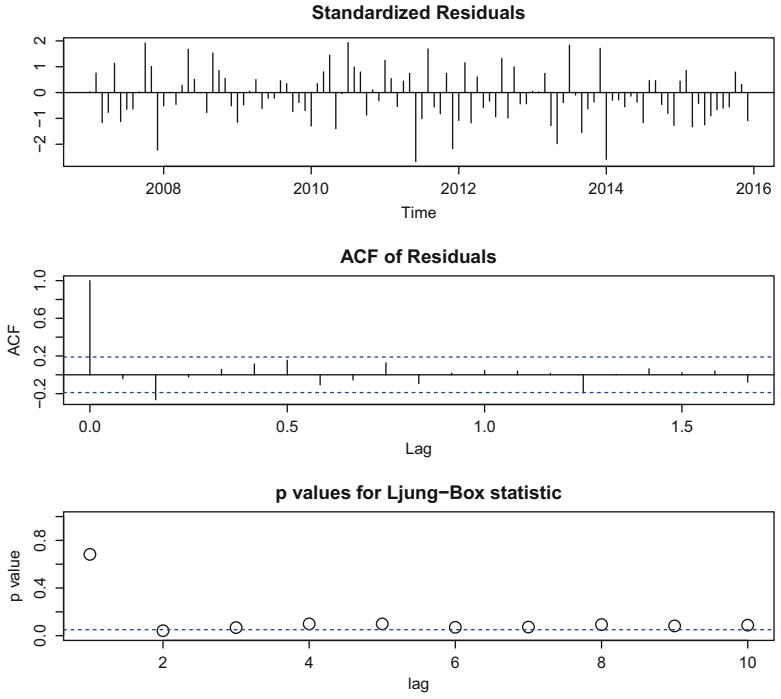

Fig. 13.9 Residual diagnostics for ARIMA(0,1,1) model with drift. Top panel: standardized residuals against time. Center panel: ACF with significant bounds (a value of 1 on x-axis reflects a lag of 12 months). Lower panel: p-values resulting from Ljung-Box testing (H_0: no autocorrelation)

(i.e., the preference for young over is expected to decrease). However, we have to keep in mind that there could be "shocks" (e.g., a presidential election which causes changes in people's implicit associations).

Let us conclude this section with some final remarks. It is certainly tempting to just say `auto.arima` on our original data. The danger is, at least for people new to time series modeling, that the model is pretty much a black box. It is suggested to explore the time series in detail using the steps presented in this section. This gives us a good insight into various aspects of the time series. Note that the parameters in ARIMA models are typically not subject to substantive interpretation. ARIMA models are also robust against violations of normal error terms.

For some time series (especially those with irregular shocks), ARIMA models sometimes predict a horizontal line. There is nothing wrong with this; it simply tells us that the best guess for future developments is a horizontal line. If the aim of a time series analysis is forecasting only, the user can consider exponential smoothing techniques such as Holt-Winters (see `HoltWinters` function), or exponential smoothing state space models (see `ets` function in **forecast**). Examples can be found in Hyndman and Athanasopoulos (2014, Chapter 7). Within such a prediction

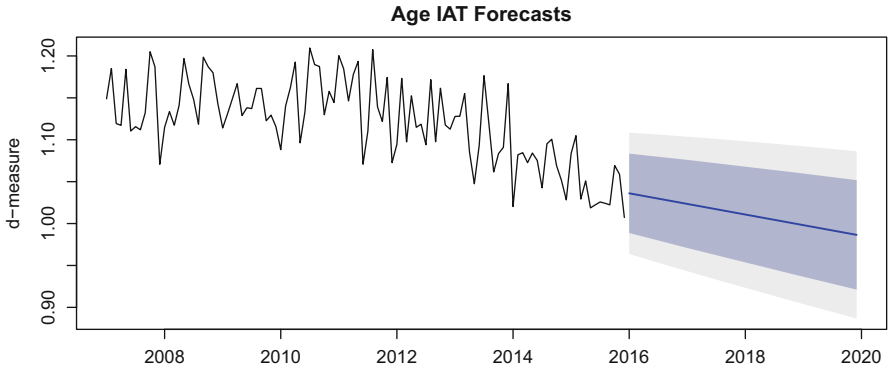

Fig. 13.10 IAT d-measure predictions based on the ARIMA(0,1,1) model with drift until the year 2020

context, it can be helpful to fit several candidate models on the time series without considering the last few observations. We then forecast the left-out observations and check how close these predictions match the observed (left-out) values.

13.3.3 Time Series with Covariates: Intervention Analysis

In Sect. 13.3.1 we have illustrated a procedure that automatically finds points where the series breaks. In this section we are interested whether a specific event, occurred at a known point in time, has an influence on the time series. We focus on a method called *intervention analysis*, introduced by Box and Tiao (1975), and consider the simplest functional form of such an intervention: a step function consisting of 0's (before event) and 1's (from event until the end of the series).

Let us illustrate this strategy using our age IAT series. On March 23, 2010, the Affordable Care Act (ACA; also known as Obamacare) was passed. We are interested in whether this event had an impact on our time series. We could hypothesize that it might have increased empathy toward the needs of elderly individuals. We start with specifying the ACA intervention vector with a jump at the 40th observation (April, 2010).

```
aca <- c(rep(0, 39), rep(1, 69))
```

The vector needs to be of the same length as the time series. Note that this is the simplest intervention function which requires the time series to react immediately. More complex and at the same time more realistic interventions (in terms of lagged influences) can be found in Cryer and Chan (2008, Chapter 11).

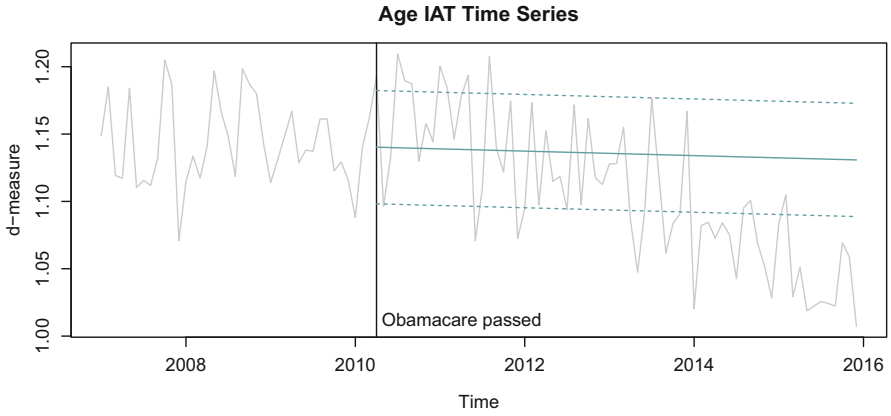

Fig. 13.11 Predictions based on pre-Obamacare time series (incl. 80% prediction interval)

First, we fit a pre-event ARIMA model (we use the ARIMA(0,1,1) drift specification from above) and predict the remaining values up to December 2015. Figure 13.11 shows that the predictions deviate clearly from the actually observed post-Obamacare time series, especially from 2012 onward.

```
preaca <- window(yts, end = c(2010, 3)) ## pre-event series
tspre <- Arima(preaca, order = c(0,1,1), include.drift = TRUE)
preds <- forecast(tspre, h = 69)    ## post-event predictions
```

Now let us include the ACA vector as covariate in our ARIMA model. This can be achieved via the `xreg` argument. Such models are sometimes also referred to as ARIMAX models. The `coeftest` function from the **lmtest** package can be used for significance testing. We are mostly interested in the ACA effect.

```
tsreg <- Arima(yts, order = c(0, 1, 1), include.drift = TRUE,
               xreg = aca)
print(coeftest(tsreg), 3, signif.legend = FALSE)
##
## z test of coefficients:
##
##         Estimate Std. Error z value Pr(>|z|)
## ma1    -0.904720   0.040880  -22.13   <2e-16 ***
## drift -0.001540   0.000426   -3.62   0.0003 ***
## aca    0.044923   0.017808    2.52   0.0116 *
```

We see that the introduction of Obamacare had a significant influence on our time series; implicit associations changed after this intervention in favor of older people. However, we should not overemphasize this result since there could be of course other sociopolitical events at around that time that contributed to the change in the time series. In addition, as mentioned above, we used a simple step intervention function.

The `xreg` argument can also handle multiple covariates. In this case they need to be provided as data frame. In general, such covariates do not have to be discrete; they can also be metric, which would constitute a second time series. In relation to such setups, the concept of *Granger causality* can be used if we want to study causal influence of one time series on a second one. The **lmtest** package offers a corresponding function called `grangertest`.

This concludes the section on time series analysis. As mentioned at the beginning of this section, we focused on univariate time series modeling (i.e., modeling a single series) only. For modeling multiple series jointly, *vector autoregressive models* (VAR) can be used, as implemented in the **vars** package (Pfaff, 2008). For settings where each individual produces a series and we are not interesting in making predictions, functional data analysis is an attractive modeling framework, as shown in the next section.

13.4 Functional Data Analysis

The main aspect of *functional data analysis* (FDA) is that a single measurement on an individual is a *function*, rather than a single value (i.e., *scalar*) as usual. In many applications, this function is measured in the time domain, and we will limit our explanations in this section to this type of data. However, other domains such as spatial data or spatiotemporal combinations can be subject to FDA as well. Many classical statistical techniques, such as ANOVA, regression, and PCA, have been extended in terms of corresponding functional variants.

In this section we avoid giving much technical details because the math required to formulate various models is beyond the scope of this book. An excellent, nontechnical introduction to FDA with focus on psychological applications is given in Levitin et al. (2007). More in-depth explanations and case studies can be found in the books by Ramsay and Silverman (2002, 2005), Ramsay et al. (2009), and Kokoszka and Reimherr (2017). In this chapter we will mostly use the **fda.usc** package (Febrero-Bande and de la Fuente, 2012) which builds on the **fda** package (Ramsay et al., 2009) but is easier to use and offers a wider range of functional analysis techniques.

The example we use throughout this section is taken from Vines et al. (2006), also analyzed in Levitin et al. (2007). The authors were interested in how physical gestures of professional musicians contribute to the perception of emotion in a musical performance. Twenty-nine participants were exposed to the performance by either just listening (auditory condition), just seeing (visual condition), or both

(auditory-visual condition). During the performance the participants had to move a slider to indicate the experienced tension they felt. They listened to the piece for 80 s, and every 10 ms the tension score (0–127) was recorded which leaves us with 800 tension measurement points per person. Let us import the raw data into R and convert them into an `fdata` object, in order to proceed with computations using the **fda.usc** package. Note that the original tension scores (0–127) were z-standardized.

```
library("MPsychoR")
library("fda.usc")
data("tension")
tension1 <- as.matrix(tension[,1:800])  ## tension time series
cond <- tension$cond                    ## condition
ftension <- fdata(tension1,
  argvals = seq(1, 80, length.out = 800),
  names = list(main = "Music tension", xlab = "Time (sec)",
               ylab = "Tension"))
```

Unlike in our example, in other applications it is sometimes necessary to align the curves in the time domain (e.g., if the starting point is located arbitrarily in time). This process is called *registration*. Details can be found in Ramsay et al. (2009, Chapter 8); the `register.fd` function in **fda** provides corresponding options.

13.4.1 Smoothing Curves and Derivatives

Having functional data, the first data preparation step is typically to smooth the raw input trajectories. This process reduces noise and gives us a mathematical description of each trajectory. There are several options for smoothing the input curves such as various spline-based approaches and nonparametric kernel smoothing (see Ramsay and Silverman, 2005, for details). Here we focus on the latter approach.

Let $\mathbf{y} = (y_1, \ldots, y_m)'$ be the observed data vector ($j = 1, \ldots, m$ measurement points) of a single individual. In FDA we assume that there is an "true" underlying trajectory $x(t_j)$, where t_j denotes the j-th time point. This trajectory is related to y_j in the following way:

$$y_j = x(t_j) + \varepsilon_j. \tag{13.8}$$

This implies that y_j is measured with error.[7] In order to determine $x(t_j)$, kernel smoothing uses

[7] Note the similarity of this equation to the classical true score model in the first chapter.

$$\hat{x}(t) = \sum_{j=1}^{m} S_j(t) y_j \tag{13.9}$$

as its basic expression. Here, $S_j(t)$ is a suitably defined weight function, the most popular being the so-called *Nadaraya-Watson kernel estimator* (see Ramsay and Silverman, 2005, Chapter 4, for the kernel expression). This estimator involves a smoothing parameter (or bandwidth) h which steers the degree of smoothness. The optimal h can be found through cross-validation (CV). The larger h, the smoother the curves.

At the end of this procedure, we get a smooth function $\hat{x}(t)$ which replaces the raw input vector $\mathbf{y}$ in all subsequent FDA computations. This smoothing is performed on all individual trajectories (h constant across individuals). Let us perform such a nonparametric kernel smoothing using our music tension dataset (CV to find h is carried out internally) and plot the original trajectories as well as the smoothed ones.

```
ftensionNP <- min.np(ftension)
plot(ftension, main = "Original Data")
plot(ftensionNP$fdata.est, main = "Smooth Data")
```

The bottom panel of Fig. 13.12 shows the smoothed versions of the input data. Using such a nonparametric kernel strategy with h determined through CV often leads to good results, but not always. The user is encouraged to try out different smoothing specifications and produce similar plots as Fig. 13.12. At the end of the day, the curves should look reasonably smooth (i.e., we reduced the noise), but, at the same time, they should capture important peaks and valleys of the raw input signal (i.e., not over-smoothing). In our example we proceed with the smoothed curves from the `min.np` call.

At this point we can compute the first-order derivative, often interpreted as *velocity*.

```
ftension1 <- ftensionNP$fdata.est
deriv1 <- fdata.deriv(ftension1)
```

Figure 13.13 presents a single tension trajectory (person 7) and its derivative. The person experiences a tension peak at around 30 s. The velocity reflects the *rate of change* in tension, that is, how rapidly it rises and how rapidly it drops. In some applications the second-order derivative (*acceleration*) can be of interest as well.

13.4.2 FDA Descriptives and Bootstrap

After smoothing the curves, we can compute basic functional location and dispersion measures. It is important to point out that each of these measures is a function itself. Let us split the music tension data according to the conditions and compute *functional means* and *functional medians*.

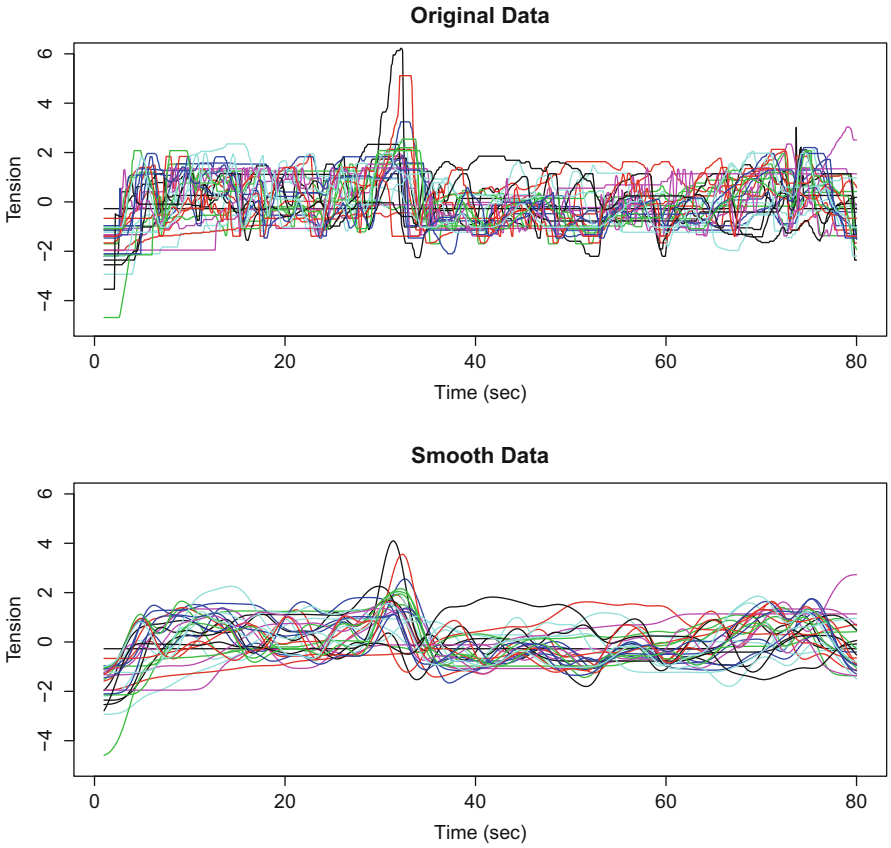

Fig. 13.12 Nonparametric kernel smoothing on music tension data. Top panel: observed trajectories. Bottom panel: smooth trajectories

```
fsplit <- split.fdata(ftension1, cond)
Amean <- func.mean(fsplit$Auditory)
Vmean <- func.mean(fsplit$Visual)
```

(continued)

```
AVmean <- func.mean(fsplit$AuditoryVisual)
Amedian <- func.med.FM(fsplit$Auditory)
Vmedian <- func.med.FM(fsplit$Visual)
AVmedian <- func.med.FM(fsplit$AuditoryVisual)
```

The top panel of Fig. 13.14 shows the smoothed curves for the means and the bottom panel the curves for the median. Corresponding functions for other location measures (e.g., trimmed mean) or dispersion measures are implemented in **fda.usc** as well.

If we want to explore to which degree a location function changes across multiple samples, we can apply a simple bootstrap strategy. Let us perform a functional mean bootstrap (200 bootstrap samples) for the auditory-visual condition (see Fig. 13.15).

```
set.seed(123)
AVboot <- fdata.bootstrap(fsplit$AuditoryVisual, draw = TRUE)
```

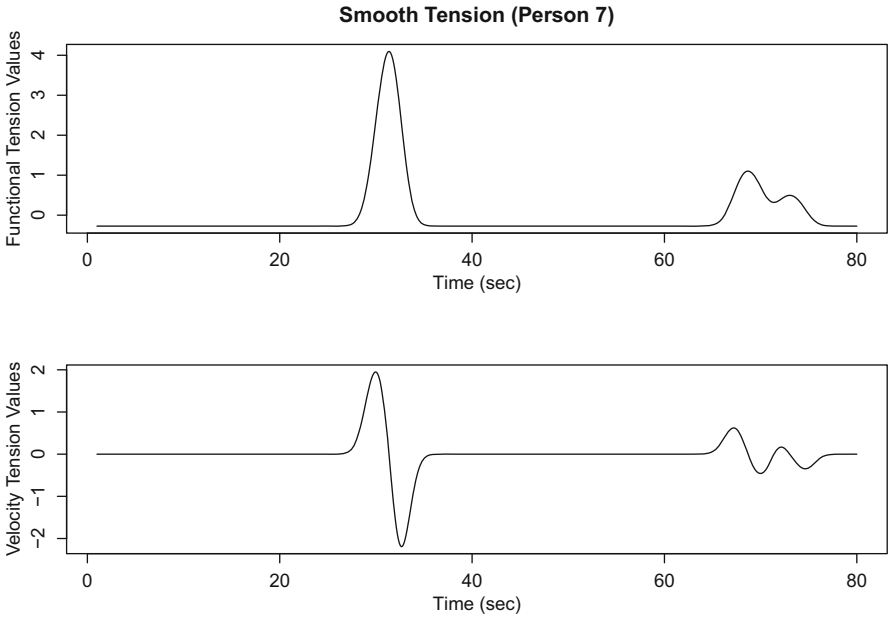

Fig. 13.13 Top panel: smooth trajectory person 7. Bottom panel: first order derivative (velocity) person 7

Similar bootstrap strategies can be applied to other descriptive measures. In case it is of interest, the same descriptive measures can be computed on the derivatives of the smooth trajectories.

13.4.3 Functional ANOVA and Regression Modeling

Functional regression refers to data settings where at least one of the variables involved (i.e., response and/or predictors) is of a functional form. In our example, the response is functional and we have one categorical predictor (scalar). For such settings there are various options for computing a *functional one-way ANOVA*. A simple implementation is the `anova.onefactor` function in **fda.usc**, which tells us whether there is an overall difference in the functional trajectories across the three conditions. We keep the number of bootstrap samples fairly low because the computation is quite time-consuming.

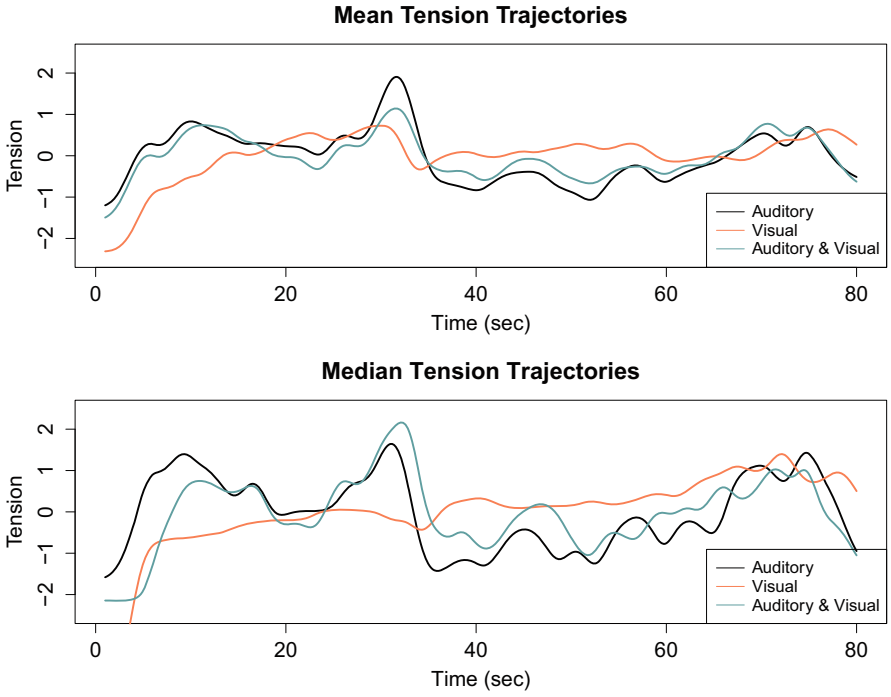

Fig. 13.14 Top panel: means tension trajectories for each condition. Right panel: median tension trajectories for each condition

```
set.seed(123)
tension1way <- anova.onefactor(ftension1, cond, nboot = 50)
tension1way$pvalue
## [1] 0
```

The result suggests that there is an overall significant difference among the three smoothed mean curves.

Another option is to use the idea of random projections (Cuesta-Albertos and Febrero-Bande, 2010), which transform the functional data into univariate data, solve the ANOVA problem for this simple situation, and obtain conclusions for the functional data by collecting the information from several projections. The authors suggest that the number of projections to be used should be $\min(30, n)$. Since in our data $n = 800$, we are good with 30 projections. This approach allows us to include special contrasts. Below we use a simple dummy coding with the auditory condition as baseline.

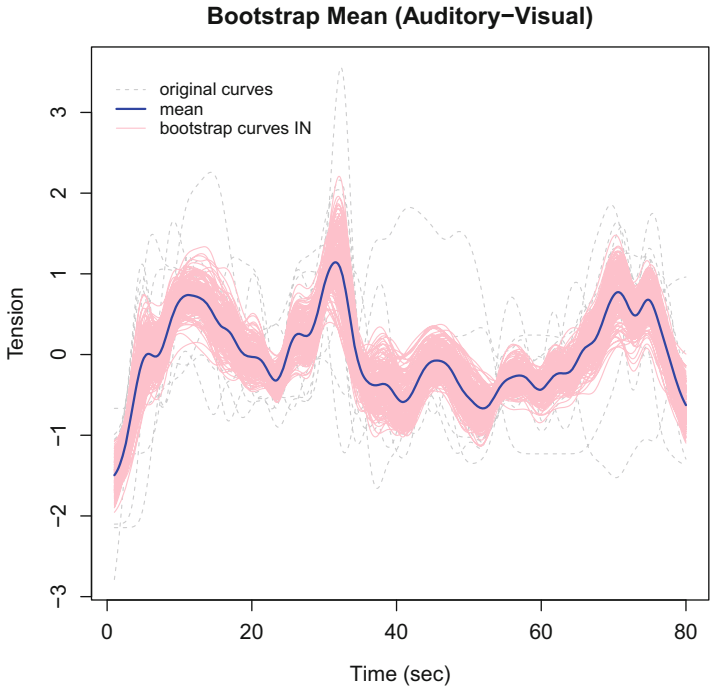

Fig. 13.15 Bootstrap of functional means for auditory-visual condition

```
fdat <- as.data.frame(ftension1$data)
gdat <- as.data.frame(cond)
ctrAudio <- contr.treatment(3)
set.seed(222)
fitrpm <- anova.RPm(fdat, ~ cond, gdat, RP = 30,
                    contrast = list(cond = ctrAudio))
summary.anova(fitrpm)
##       - SUMMARY anova.RPm -
##
##  p-value for Bonferroni method
##          cond C1.cond C2.cond
## RP30 0.00125   3e-05 0.26624
##
##   p-value for False Discovery Rate method
##          cond C1.cond C2.cond
## RP30 0.00124   3e-05 0.16946
```

It presents two different types of p-value corrections (Bonferroni, false discovery rate) which, in our application, lead to consistent decisions. The first p-value (`cond`) reflects overall group differences in the sense of an F-test: it suggests that there is a significant overall difference in the means. The second p-value (`C1.cond`) tests visual vs. auditory (significant difference), whereas the third p-value tests visual vs. auditory-visual (not significant).

So far we have tested for differences at a global level, that is, across all 800 measurements. In FDA we are often interested in how predictors behave along the time continuum. The **refund** package (Goldsmith et al., 2016) is designed for advanced regression modeling on functional data. First, we need to convert the data into an `fd` object and then set up the design matrix. We use the same dummy coding

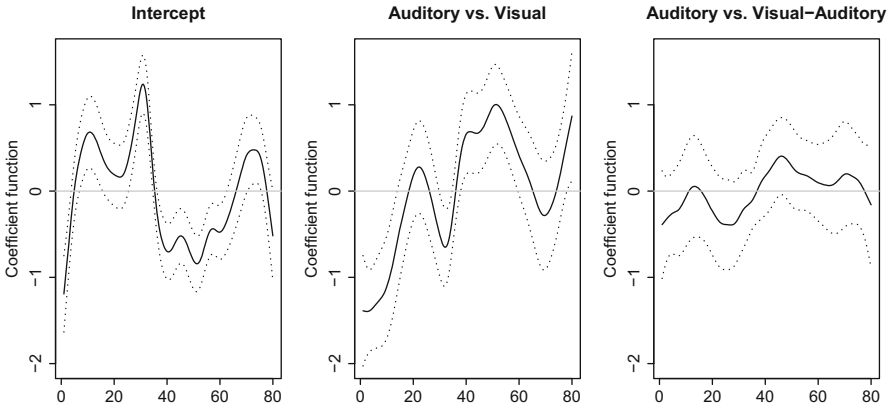

Fig. 13.16 Smoothed functional parameter estimates for functional regression including 95% CI: intercept, auditory vs. visual contrast, auditory vs. visual-auditory contrast

strategy as above. The next line in the chunk below fits the functional regression model. Note that instead of obtaining a single value for each regression parameter as in ordinary regression, in functional regression each regression parameter is a function along time, potentially subject to further smoothing. Here, mostly for running time purposes, we set the smoothing parameter to a single value of $\lambda = 100$ which works well in our example. A good, albeit time-consuming option is to let the algorithm pick the optimal λ by means of CV (default in the function below). Let us fit the functional regression model:

```
library("refund")
ftension2 <- fdata2fd(ftension1)
X <- model.matrix(~ cond)  ## auditory as baseline
tenreg <- fosr(fdobj = ftension2, X = X, lambda = 100)
```

By saying `plot(tenreg)`, the plot in Fig. 13.16 is produced. We see that in certain segments along the time continuum (i.e., within first 20 and 40–60 s, approximately) the effect of auditory vs. visual differs significantly from 0. There is no significant difference at any point in time between auditory and visual-auditory condition.

Other functions implemented in **refund** for regression with functional responses are `pffr` for additive functional regression and `fosr2s` which fits a separate linear model at each point along the continuum and smooths the coefficients in a second step. For hierarchical data settings (e.g., within-subject designs), the `bayes_fosr` function replaces `fosr`. The random effect (let us call it `id`) can be added using `re(id)` in the formula specification. If the response is a scalar and the predictors are functional, **refund** offers `pfr` (penalized functional regression), `lpeer` (longitudinal model with structures penalties), `fpcr` (principal component regression), and `fgam` (functional generalized additive models). Details can be found in Greven and Scheipl (2017), Scheipl et al. (2015), and McLean et al. (2014).

The **fda.usc** package provides `fregre.lm` (regression with functional and non-functional covariates), `fregre.glm` (functional GLM), `fregre.pc` (principal component regression), and more (see Febrero-Bande and de la Fuente, 2012). Extensive treatments of functional ANOVA and regression can be found in Zhang (2014) and Kokoszka and Reimherr (2017).

13.4.4 Functional Principal Component Analysis

Principal component analysis (PCA) was introduced in Sect. 6.1 and aims to reduce the complexity of the data by computing principal components. In functional PCA (fPCA), the resulting *functional principal components* (fPCs) are again functions.

Using the **fda.usc** package, the `fdata2pc` can be used to compute an fPCA. Let us fit a 2D-fPCA on the music tension data:

```
fpca <- fdata2pc(ftension1, ncomp = 2)
summary(fpca, biplot = FALSE)
##
##      - SUMMARY:   fdata2pc  object   -
##
## -With 2  components are explained  53.32 %
##  of the variability of explicative variables.
##
## -Variability for each component (%):
##   PC1   PC2
## 34.14 19.19
```

This output shows the amount of explained variance if we extract two components. A scree plot with the standard deviations of all the fPCs extracted (see `fpc1$d` object) can be used for dimensionality assessment.

Figure 13.17 shows the functional loadings for each component in the left panel.[8] Using this representation, the components are tricky to interpret. Below we show a different representation that eases their interpretation. Plotting the PC scores in the 2D-component space (right panel) gives us a nice picture: we see a clear separation between participants in the visual condition and participants in the auditory/auditory-visual conditions. The scores for auditory vs. auditory-visual are poorly separated in two dimensions. These findings substantiate the functional ANOVA and regression results from above.

In terms of interpreting the fPCs, Ramsay and Silverman (2005) suggest plotting the functional mean trajectory and then adding and subtracting a suitable multiple of each PC curve. The necessary computations are the following:

```
fmean <- func.mean(ftension1)
pc1plus <- fmean$data[1,] + 3*fpca$rotation$data[1,]
pc1minus <- fmean$data[1,] - 3*fpca$rotation$data[1,]
pc2plus <- fmean$data[1,] + 3*fpca$rotation$data[2,]
pc2minus <- fmean$data[1,] - 3*fpca$rotation$data[2,]
```

The first line computes the functional mean trajectory. The subsequent two lines add/subtract the first fPC from the functional mean. The last two lines do the same thing for the second fPC. Note that the multiple of 3 is pretty much arbitrary; it merely amplifies the distance of the fPC trajectories from the mean such that we get

[8] A simplified version of this plot can be produced via `summary(fpca)`.

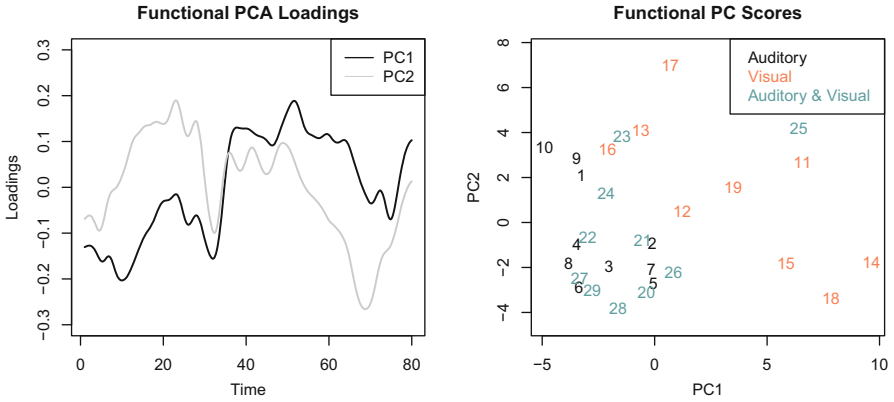

Fig. 13.17 Left panel: functional loadings for each PC. Right panel: PC scores colored according to the experimental condition

a clearer picture for interpretation. The following code chunk produces the plots in Fig. 13.18.

```
plot(fmean, lwd = 2, main = "Mean (PC1)", ylim = c(-2, 2))
lines(ftension1$argvals, pc1plus, col = "salmon")
lines(ftension1$argvals, pc1minus, col = "cadetblue")
legend("bottomright", legend = c("mean + PC1", "mean - PC1"),
       lty = 1, col = c("salmon", "cadetblue"))
plot(fmean, lwd = 2, main = "Mean (PC2)", ylim = c(-2, 2))
lines(ftension1$argvals, pc2plus, col = "salmon")
lines(ftension1$argvals, pc2minus, col = "cadetblue")
legend("bottomright", legend = c("mean + PC2", "mean - PC2"),
       lty = 1, col = c("salmon", "cadetblue"))
```

The first component, accounting for 34.14% of the variance, can be interpreted as follows. Participants who score low on this component (blue line) have a strong initial increase in tension and are highly affected by the peak at around 30 s, and then their tension drops drastically. Participants who score high on this component (red line) have a considerably slow tension increase, are less affected by the tension peak, and remain pretty much constant for the rest of the time (especially in the 40–60 s area). Thus, the first fPC can be interpreted as "compression/expansion."

The second component, accounting for 19.19% of the variance, discriminates between tension values at the beginning of the piece (approx. 10–30 s) and at the end (approx. 60–80 s). Participants scoring high on this dimension (red line) experience high tension at the beginning and considerably low tension at the end. For those who score low (blue line), these tension experiences are reversed. We could label this fPC as "edge effects."

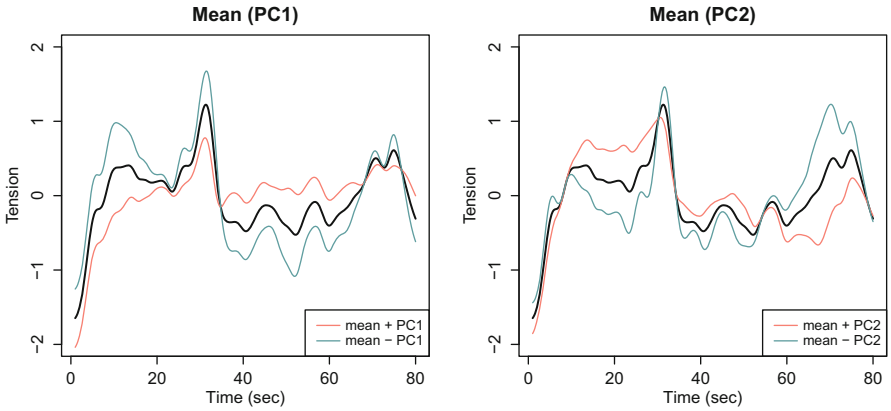

Fig. 13.18 Left panel: functional mean trajectory ± first functional PC. Right panel: functional mean trajectory ± second functional PC

Further details on the fPCA fit on these data including additional interpretation flavors can be found in Levitin et al. (2007), who fitted a 3D-fPCA solution. Another reference for an fPCA application in psychology is Burns et al. (2013). The next chapter is dedicated to a special type of functional data in psychology: functional magnetic resonance imaging data.

References

Bornstein, A. M., & Daw, N. D. (2012). Dissociating hippocampal and striatal contributions to sequential prediction learning. *European Journal of Neuroscience, 35*, 1011–1023.

Box, G. E. P., & Tiao, G. (1975). Intervention analysis with applications to economic and environmental problems. *Journal of the American Statistical Association, 70*, 70–79.

Burns, D. M., Houpt, J. W., Townsend, J. T., & Endres, M. J. (2013). Functional principal components analysis of workload capacity functions. *Behavior Research Methods, 45*(4), 1–18.

Cowpertwait, P. S. P., & Metcalfe, A. V. (2009). *Introductory time series with R*. New York: Springer.

Cryer, J. D., & Chan, K. S. (2008). *Time series analysis with applications in R* (2nd ed.). New York: Springer.

Cuesta-Albertos, J. A., & Febrero-Bande, M. (2010). A simple multiway anova for functional data. *Test, 19*, 537–557.

Epskamp, S., Cramer, A. O. J., Waldorp, L. J., Schmittmann, V. D., & Borsboom, D. (2012). **qgraph**: Network visualizations of relationships in psychometric data. *Journal of Statistical Software, 48*(4), 1–18. http://www.jstatsoft.org/v48/i04/

Febrero-Bande, M., & de la Fuente, M. (2012). Statistical computing in functional data analysis: The R package **fda.usc**. *Journal of Statistical Software, 51*(4), 1–28. https://www.jstatsoft.org/v051/i04

Goldsmith, J., Scheipl, F., Huang, L., Wrobel, J., Gellar, J., Harezlak, J., McLean, M. W., Swihart, B., Xiao, L., Crainiceanu, C., & Reiss, P. T. (2016). **refund**: Regression with functional data. R package version 0.1-16. https://CRAN.R-project.org/package=refund

Greenwald, A. G., & Banaji, M. R. (1995). Implicit social cognition: Attitudes, self-esteem, and stereotypes. *Psychological Review, 102*, 4–27.

Greenwald, A. G., McGhee, D. E., & Schwartz, J. K. L. (1998). Measuring individual differences in implicit cognition: The implicit association test. *Journal of Personality and Social Psychology, 74*, 1464–1480.

Greenwald, A. G., Nosek, B. A., & Banaji, M. R. (2003). Understanding and using the implicit association test: I. An improved scoring algorithm. *Journal of Personality and Social Psychology, 85*, 197–216.

Greven, S., & Scheipl, F. (2017). A general framework for functional regression modelling. *Statistical Modelling, 17*, 1–35.

Griffiths, T. L., Steyvers, M., & Firl, A. (2007). Google and the mind: Predicting fluency with PageRank. *Psychological Science, 18*, 1069–1076.

Hyndman, R. J., & Athanasopoulos, G. (2014). *Forecasting: Principles and practice*. Melbourne: OTexts. http://www.otexts.org

Hyndman, R. J., & Khandakar, Y. (2008). Automatic time series forecasting: The **forecast** package for R. *Journal of Statistical Software, 26*(3), 1–22. http://www.jstatsoft.org/article/view/v027i03

Jebb, A. T., Tay, L., Wang, W., & Huang, Q. (2015). Time series analysis for psychological research: Examining and forecasting change. *Frontiers in Psychology, 6*(727), 1–24. http://journal.frontiersin.org/article/10.3389/fpsyg.2015.00727

Kokoszka, P., & Reimherr, M. (2017). *Introduction to functional data analysis*. Boca Raton: CRC Press.

Levitin, D. J., Nuzzo, R. L., Wines, B. W., & Ramsay, J. O. (2007). Introduction to functional data analysis. *Canadian Psychology, 48*, 135–155.

Long, J. D. (2011). *Longitudinal data analysis for the behavioral sciences using R*. Thousand Oaks: SAGE Publishing.

McLean, M. W., Hooker, G., Staicu, A. M., Scheipl, F., & Ruppert, D. (2014). Functional generalized additive models. *Journal of Computational and Graphical Statistics, 23*, 249–269.

Mirman, D. (2014). *Growth curve analysis and visualization using R*. Boca Raton: Chapman & Hall/CRC.

Nosek, B. A., Banaji, M. R., & Greenwald, A. G. (2002). Harvesting implicit group attitudes and beliefs from a demonstration web site. *Group Dynamics: Theory, Research, and Practice, 6*, 101–115.

Petris, G., & Petrone, S. (2011). State space models in R. *Journal of Statistical Software, 41*(4), 1–25. https://www.jstatsoft.org/v041/i04

Pfaff, B. (2008). VAR, SVAR and SVEC models: Implementation within R package **vars**. *Journal of Statistical Software, 27*(4), 1–32. https://www.jstatsoft.org/article/view/v027i04

Ramsay, J. O., & Silverman, B. W. (2002). *Applied functional data analysis: Methods and case studies*. New York: Springer.

Ramsay, J. O., & Silverman, B. W. (2005). *Functional data analysis* (2nd ed.). New York: Springer.

Ramsay, J. O., Hooker, G., & Graves, S. (2009). *Functional data analysis with R and MATLAB*. New York: Springer.

Scheipl, F., Staicu, A. M., & Greven, S. (2015). Functional additive mixed models. *Journal of Computational and Graphical Statistics, 24*, 477–501.

Spedicato, G. A. (2017). **markovchain**: Easy handling of discrete time Markov chains. R package version 0.6.8.1.

Trapletti, A., & Hornik, K. (2017). **tseries**: Time series analysis and computational finance. R package version 0.10–42. https://CRAN.R-project.org/package=tseries

Ulrich, J. (2017). **TTR**: Technical trading rules. R package version 0.23–2. https://CRAN.R-project.org/package=TTR

Vines, B. W., Krumhansl, C. L., Wanderley, M. M., & Levitin, D. J. (2006). Cross-modal interactions in the perception of musical performance. *Cognition, 101*, 80–113.

Visser, I. (2011). Seven things to remember about hidden Markov models: A tutorial on Markovian models for time series. *Journal of Mathematical Psychology, 55*, 403–415.

Visser, I., & Speekenbrink, M. (2010). **depmixS4**: An R package for hidden Markov models. *Journal of Statistical Software, 36*(7), 1–21. http://www.jstatsoft.org/v36/i07/

Wickens, T. D. (1982). *Models for behavior: Stochastic processes in psychology*. San Francisco: W.H. Freeman and Co.

Zeileis, A., & Hothorn, T. (2002). Diagnostic checking in regression relationships. *R News, 2*(3), 7–10. https://CRAN.R-project.org/doc/Rnews/

Zeileis, A., Leisch, F., Hornik, K., & Kleiber, C. (2002). **strucchange**: An R package for testing for structural change in linear regression models. *Journal of Statistical Software, 7*(2), 1–38. http://www.jstatsoft.org/v07/i02/

Zhang, J. T. (2014). *Analysis of variance for functional data*. Boca Raton: CRC Press.

Zucchini, W., MacDonald, I. L., & Langrock, R. (2016). *Hidden Markov models for time series: An introduction using R* (2nd ed.). Boca Raton: Chapman & Hall/CRC.

Chapter 14
Analysis of fMRI Data

14.1 fMRI Data Manipulation in R

14.1.1 fMRI Data Structures

In fMRI, the brain of a participant is scanned over time. During the experiment, the participant in the scanner is exposed to experimental stimuli. The resulting data are 4D: The first two dimensions x and y span a 2D voxel grid, and the third dimension reflects the brain slices in z direction. A static 3D brain image is fully characterized by these three dimensions (spatial domain). The "functional" part (fourth dimension) in the fMRI acronym relates to the change of brain activation over time (temporal domain). The time between successive brain scans is called *repetition time*, commonly abbreviated as TR ("time of repetition").

There are several different formats to store raw fMRI data such as DICOM, Analyze, NIfTI, or MINC. In this chapter we work with Analyze and NIfTI data. Analyze data consist of a pair of files: a `.img` file containing the 4D brain data and a `.hdr` file with meta information. NIfTI data come as a single file with the ending `.nii`, containing all the necessary information.

In order to get an initial idea of such a data structure, let us import a scan (based on a single run of a participant) from the mental states study by Tamir et al. (2016a). The experiment itself will be described in more detail in Sect. 14.2.4. The authors made the fMRI data publicly available through a Dataverse repository (Tamir et al., 2016b), stored as NIfTI files. In order to reproduce the R code in this chapter, the following files need to be downloaded from this repository[1]:

- `s01_r01_FUNC.nii`;
- `wFUN1.nii`; `wFUN2.nii`; ... `wFUN16.nii`;

[1] See http://dx.doi.org/10.7910/DVN/ELLLZM

P. Mair, *Modern Psychometrics with R*, Use R!,
https://doi.org/10.1007/978-3-319-93177-7_14

- `globalmask.img`; `globalmask.hdr`;
- `con_s01_0002.img`; `con_s01_0002.img`; ... `con_s20_0002.img`;
- `con_s01_0002.hdr`; `con_s01_0002.hdr`; ... `con_s20_0002.hdr`.

Let us import the first run (each participant was exposed to multiple runs) of the first participant using the **fmri** package (Polzehl and Tabelow, 2007; Tabelow and Polzehl, 2011), show a basic summary output, and extract the 4D array.

```
library("fmri")
scanS1R1 <- read.NIFTI("s01_r01_FUNC.nii")
summary(scanS1R1)
## Object of class fmridata
## Data Dimension: 86 86 43 162
## Data Range    : 0 ... 2870
## Voxel Size    : 2.511628 2.511628 2.75
## File(s)       : s01_r01_FUNC.nii
imageS1R1 <- fmri::extract.data(scanS1R1)
```

In this example the 2D voxel grid is of dimension 86×86 (x and y coordinates). The third dimension represents 43 axial slices (z coordinates). Thus, the brain is represented by $86 \times 86 \times 43 = 318{,}028$ voxels. The fourth dimension represents the scans over time t (here, 162 scans). In total, we have $86 \times 86 \times 43 \times 162 = 51{,}520{,}536$ measurements for one run on a single participant. A single voxel value (e.g., $x = 50$, $y = 50$, $z = 10$, $t = 100$) can be extracted as follows:

```
imageS1R1[50, 50, 10, 100]
## [1] 598
```

This value reflects the observed BOLD (*blood oxygen level-dependent*) response which corresponds to the concentration of deoxyhemoglobin. An increased neuronal activity in a particular voxel is due to an increased cerebral blood flow in this brain area (i.e., an increase in the ratio of oxygenated hemoglobin relative to deoxygenated hemoglobin).

During a single run, an individual is typically exposed to several stimuli. The points in time at which the stimuli are presented are the *onsets*. Each stimulus is presented for a particular *duration*. Depending on the nature of stimuli presentation, we can distinguish between the following experimental designs:

- In an *event-related design*, each functional run is broken down into a sequence of stimuli presentations from multiple conditions.
- In a *block design*, we have a continuous presentation of many stimuli from a single condition.

Regardless which design is used in the experiment, statistical analyses can be performed at various aggregation levels: sometimes we fit models for each participant separately; sometimes we fit models across all participants. We can be interested in either single voxel activations or the activation of a *region of interest* (ROI).

There is no single "can-do-everything" fMRI package in R. However, the R package ecosystem provides tons of useful functions; we just need to put the pieces of the puzzle together such that R becomes a powerful engine for fMRI data analysis. Two major fMRI packages are **fmri** and **AnalyzeFMRI** (Bordier et al., 2011). Roughly speaking, the **fmri** package is strong for univariate analyses, whereas **AnalyzeFMRI** focuses on multivariate methods. A quick-start guide for fMRI data analysis in R by means of these two packages is given in Eloyan et al. (2014). Another interesting recent development is the **neuropointilist** package (Madhyastha, 2017) which allows researchers to efficiently fit voxel-wise models in R.

14.1.2 fMRI Preprocessing

In the code chunk above, we imported a raw fMRI scan. Such raw scans are pretty much useless. Participants may move their head slightly during the experiment, which changes the voxel alignment. In addition, brains differ in size across individuals. Thus, we need to apply some data preprocessing steps before computing statistical models. The standard fMRI preprocessing pipeline is the following:

- *Slice-timing correction*: It takes time to scan the brain in slices. Slice-timing correction replaces the observed BOLD response with the BOLD response we would expect if all slices were scanned at the same time.
- *Head motion corrections*: Small head movements can already heavily bias the fMRI data. Methods like *rigid body registration* correct for such movements.
- *Normalization*: There are differences in size and shape of human brains. Normalization methods warp individual brains to a common template (e.g., the Montreal Neurological Institute (MNI) template is widely used).
- *Spatial smoothing*: The BOLD value of each voxel is replaced by a weighted average of the BOLD responses of neighboring voxels. This helps overcome imperfect functional alignment between subjects and reduces noise in the image.

Unfortunately, at the time this book was written, R is poorly equipped with preprocessing tools such that other tools such as the SPM suite of MATLAB functions (The FIL Methods Group, 2016) or FSL (Smith et al., 2004) need to be used. What we can show in R, however, is spatial smoothing.

Let us import a preprocessed version of the file above, available in the same Dataverse repository. All the preprocessing steps listed above were performed, except for smoothing. We also import a binary 3D mask in Analyze format that separates background voxels (value of 0) from non-background voxels (value of 1), since we have a 3D rectangular voxel grid in which the brain is embedded.

```
mask <- read.ANALYZE("globalmask.img")
mask <- extract.data(mask)[,,,1]
imageS1R1r <- read.NIFTI("wFUN1.nii")
imageS1R1 <- fmri::extract.data(imageS1R1r)
dim(imageS1R1)
## [1]  79  97  80 162
```

Note that compared to the raw scan above, the spatial brain dimensions have changed. This is due to the normalization template the authors used (to be precise, they used the MNI ICBM 152 template, 2 mm voxels).

When it comes to smoothing, we can smooth across all four dimensions. To keep it simple, let us smooth across the spatial dimensions at time $t = 1$ only.

```
imageS1R1T1 <- imageS1R1[,,,1]
```

A standard smoothing choice are *Gaussian spatial smoothing kernels* f_x, f_y, and f_z. In x-direction we specify

$$f_x(x - x_i) = \exp\left(-\frac{(x - x_i)^2}{2\sigma_x^2}\right). \tag{14.1}$$

The standard deviation σ_x steers the degree of smoothing: the larger σ_x, the more we smooth. In fMRI, the kernel width is often expressed by the *full width at half maximum* (FWHM):

$$\text{FWHM} = \sigma_x \sqrt{8 \ln 2}. \tag{14.2}$$

A standard choice for the FWHM is between 1 and 3 voxel widths. In the scan we imported, the voxel size is 2 mm. A reasonable choice for the FWHM is $2 \times 2.5 = 5$.

The **AnalyzeFMRI** package has a function that performs Gaussian smoothing. Instead of the FWHM as input argument, the smoothing degree needs to be provided as variance-covariance matrix (with σ_x^2, σ_y^2, and σ_z^2 on its diagonal). By using the conversion formula in Eq. (14.2), σ-values of 2 correspond approximately to an FWHM of 5 (we use the same values for each dimension). The function also takes a mask argument in order to avoid smoothing outside the actual brain region. For illustration, we present a non-smoothed version (all σ's are 0) and a smoothed version with all σ's equal to 2.

```
library("AnalyzeFMRI")
sigma <- 2
unsmooth <- GaussSmoothArray(imageS1R1T1, sigma = diag(0, 3),
              mask = mask)
smooth <- GaussSmoothArray(imageS1R1T1,
              sigma = diag(sigma^2, 3), mask = mask)
```

For each of the two smoothers, a single brain slice (slice number 50) is plotted in Fig. 14.1 using the **ggBrain** package (Fisher, 2016).

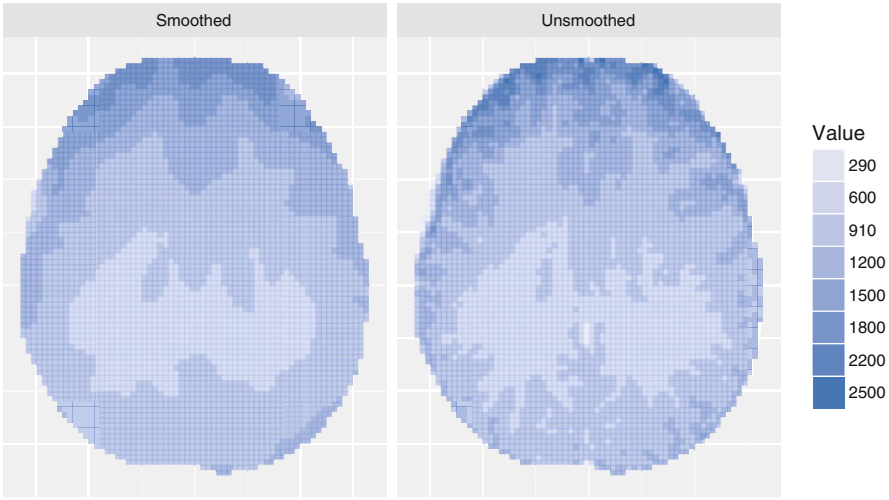

Fig. 14.1 Left panel: smoothed BOLD with $\sigma_x = \sigma_y = \sigma_z = 2$ (corresponds approximately to FWHM $= 5$). Right panel: unsmoothed BOLD image

```
library("ggBrain")
plotsmooth <- ggBrain(brains = list(unsmooth, smooth),
  mask = mask, mar = c(3,3), mar_ind = c(50,50),
  brain_ind = c(1,2), col_ind = c("Unsmoothed", "Smoothed"),
  type = 'signed')
plotsmooth
```

This figure shows nicely what smoothing is doing to the activation patterns. It removes noise and provides smooth activation areas reflecting functional similarities (i.e., spatial correlations) of adjacent voxel regions.

14.1.3 Registration and Regions of Interest

As illustrated above, the **ggBrain** package produces plots for single slices at a fixed point in time. If we want to plot 3D or even 4D brain images, the **brainR** package (Muschelli et al., 2014) does the job. Let us import the MNI 152 (2 mm) voxel template used in our data example, included in the **brainR** package. Since it is

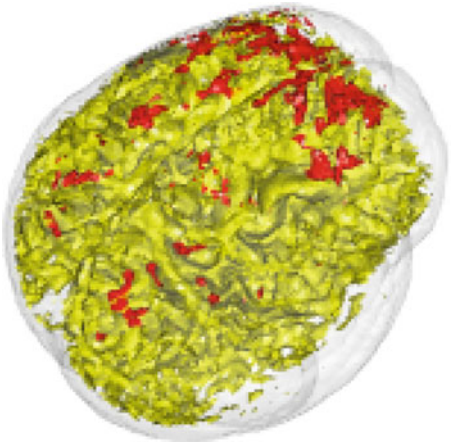

Fig. 14.2 3D brain image: activation patterns on top of MNI brain template

in compressed format, we use the import function from the **RNiftyReg** package (Clayden, 2016b).

```
library("RNiftyReg")
template <- readNifti(system.file("MNI152_T1_2mm_brain.nii.gz",
                                  package = "brainR"))
dim(template)
## [1]  91 109  91
```

Now we re-import our preprocessed scan using the same function and extract the scan at $t = 10$.

```
imageS1R1a <- readNifti("wFUN1.nii")[,,,10]
dim(imageS1R1a)
## [1] 79 97 80
```

We see that the spatial dimensions of these two scans do not match, that is, they are in different voxel spaces. In order to visualize the brain template and the

activation pattern in the same plot, we need to bring them into the same space. This process is called *image registration*. We pick the scanned image as target space and "downscale" the MNI brain template accordingly.

```
template1 <- niftyreg(source = template, target = imageS1R1a,
                      scope = "affine")$image
template1[is.na(template1)] <- 0
dim(template1)
## [1] 79 97 80
```

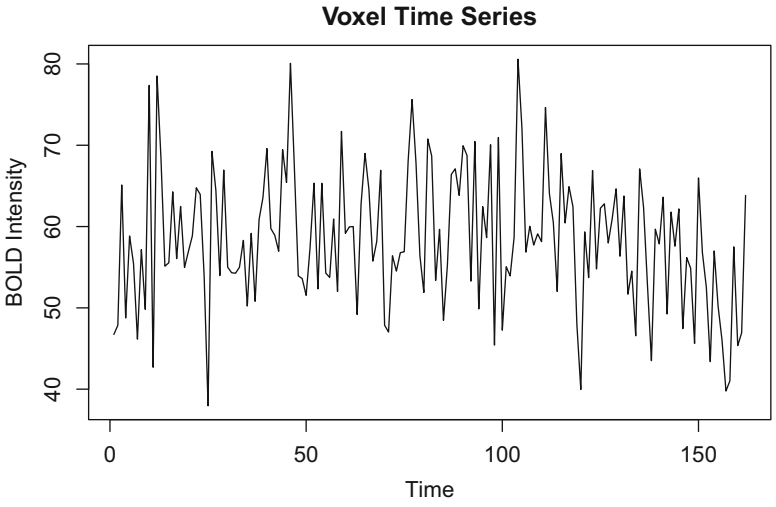

Fig. 14.3 Activation of a single voxel over time (unsmoothed)

After registration, the dimensions match (the missing values were replaced by 0's according to the mask), and we are ready to plot. The resulting 3D plot produced using the **brainR** package is given in Fig. 14.2.

```
library("brainR")
contour3d(template1, level = 3500, alpha = 0.1, draw = TRUE)
contour3d(imageS1R1a, level = c(1000, 2000), add = TRUE,
  alpha = c(0.8, 0.9), color = c("yellow", "red"), mask = mask)
```

So far we have plotted in the spatial domain. Another way to look at fMRI data is to plot a single voxel trajectory over time (i.e., temporal domain). Figure 14.3 shows such an activation (or intensity) BOLD trajectory for voxel (50, 50, 10).

Let us have a closer look at various data structures we can consider for fMRI data. There are two basic representations of fMRI data in R. The first representation is a 4D structure, as used so far. The second representation is based on flattening (or vectorizing) each 3D voxel representation at time t in order to represent it as a 2D *time* $\times$ *voxel* matrix. Below we show how to convert back and forth between these two representations. Let T be the number of time points and N be the number of non-background voxels. The corresponding matrix is of dimension $T \times N$.

```
dim(imageS1R1)                          ## 4D represenation
## [1]  79  97  80 162
nt <- dim(imageS1R1)[4]                 ## number of scans
imFlat <- fourDto2D(imageS1R1, nt)      ## flatten 4D input data
dim(imFlat)                             ## all voxels
## [1]    162 613040
maskind <- which(mask > 0)
imFlatm <- imFlat[, maskind]
dim(imFlatm)                            ## non-background voxels
## [1]    162 170686
```

In order to reconstruct the original 4D structure (i.e., without the mask), the following code chunk does the 2D-4D conversion:

```
im4D <- twoDto4D(imFlat, dim = dim(imageS1R1))
dim(im4D)
## [1]  79  97  80 162
```

Note that so far we have considered a single run on a single participant only. We will show below, when we get to the corresponding statistical methods, how to organize data from multiple runs and multiple participants. However, the basic 4D-2D conversion principle remains the same: it is just a matter of stacking the 4D or 2D structures along multiple runs/participants.

Instead of looking at single voxels, researchers often aggregate voxels to clusters called *regions of interest* (ROI). This process is called *parcellation*. Several ROI atlases have been proposed in the literature, and the most relevant ones are included in the **brainGraph** package (Watson, 2016). For illustration, we pick the AAL90 (*automates anatomical labeling*) according to Tzourio-Mazoyer et al. (2002), which divides each cerebral hemisphere into 45 anatomical ROIs.

```
library("brainGraph")
head(aal90)
##        name  x.mni y.mni  z.mni    lobe hemi index
## 1:  PreCG.L -38.65 -5.68  50.94 Frontal    L     1
## 2:  PreCG.R  41.37 -8.21  52.09 Frontal    R     2
## 3: SFGdor.L -18.45 34.81  42.20 Frontal    L     3
## 4: SFGdor.R  21.90 31.12  43.82 Frontal    R     4
## 5: ORBsup.L -16.56 47.32 -13.31 Frontal    L     5
## 6: ORBsup.R  18.49 48.10 -14.02 Frontal    R     6
```

There are three columns containing the corresponding MNI coordinates of each ROI. Let us extract these coordinates and the name of the particular ROI:

```
ROI <- factor(aal90$name)
mnixyz <- as.data.frame(aal90[,2:4, with = FALSE])
```

In the following little analysis, our goal is to use the imported brain scan from above, determine the MNI coordinates of the voxels, assign each voxel to an ROI, and average the BOLD values within the ROI. The temporal dimension is irrelevant here. In the brain scan, the spatial representation of the voxels is a 3D array. Thus, the voxels are in a matrix coordinate space.

```
dims <- dim(imageS1R1)[1:3]  ## spatial dimensions
Cmat <- cbind(expand.grid(z = 1:dims[1], y = 1:dims[2],
                          x = 1:dims[3])[,3:1], 1)
```

The matrix $\mathbf{C}$ is of dimension $613{,}040 \times 4$ and represents an index matrix, which reflects the position of each voxel in the matrix space. Even though the temporal dimension is irrelevant, we need a fourth column consisting of 1's in order to match the dimensionality of the input data.

The second component required to convert the matrix coordinates into the MNI coordinates is a transformation matrix $\mathbf{T}$. The information to construct this matrix (i.e., row affine transformations) can be extracted from the NIfTI file:

```
Tmat <- rbind(imageS1R1r$header$srowx, imageS1R1r$header$srowy,
              imageS1R1r$header$srowz, c(0, 0, 0, 1))
Tmat
##      [,1] [,2] [,3] [,4]
```

(continued)

```
## [1,]   -2    0    0   78
## [2,]    0    2    0 -117
## [3,]    0    0    2  -72
## [4,]    0    0    0    1
```

We add the vector $(0, 0, 0, 1)$ at the bottom in order to account for the fourth dimension. Now we apply the transformation $\mathbf{M} = \mathbf{TC}'$ which changes the matrix coordinates in $\mathbf{C}$ to MNI coordinates given in $\mathbf{M}$. Then we transpose the matrix and delete the fourth column.

```
mni <- Tmat %*% t(Cmat)
mni <- (t(mni))[,-4]
```

After eliminating the last column, it gives a matrix of dimension $613{,}040 \times 3$ including all voxels. Now we select the non-background voxels using the mask (Boolean version):

```
maskTF <- as.logical(mask)  ## convert into boolean mask
mni <- mni[maskTF, ]        ## apply mask
```

This results in a matrix of dimension $170{,}686 \times 3$. Note that the conversion in the other direction, that is, from MNI space to coordinate space, can be achieved through $\mathbf{C} = (\mathbf{T}^{-1}\mathbf{M})'$.

At this point we have both the atlas and the voxels in MNI coordinates. Now we need to assign each voxel to an ROI. To illustrate this process, we apply a *k-nearest neighbor* (kNN) classification[2] as implemented in the **class** package (Venables and Ripley, 2002).

```
library("class")
fitknn <- class::knn(mnixyz, mni, ROI, k = 3)
length(fitknn)
## [1] 170686
```

[2]This is certainly not the best way of doing it since ROIs can have irregular (non-spherical) shapes. A better alternative is to import the parcellation boundaries of the ROIs into R, if available, and assign the voxels accordingly.

In the resulting output vector, each voxel is assigned to an ROI. Let us flatten the 4D array (after smoothing across all four dimensions) and compute the average activation within each ROI.

```
imageS1R1 <- GaussSmoothArray(imageS1R1,
                sigma = diag(sigma^2, 3), mask = mask)
imgMat <- fourDto2D(imageS1R1, dim(imageS1R1)[4])
imgMat <- imgMat[,maskTF]        ## apply mask
ROIMat <- t(apply(imgMat, 1,
                function(vox) tapply(vox, fitknn, mean)))
dim(ROIMat)
## [1] 162  90
```

We see that we have massively reduced the complexity of the data. Instead of the initial 170,686 voxels, we now have 90 ROIs only. Whether to proceed with statistical analysis on a voxel level or an ROI level depends on the research questions involved in the fMRI experiment.

14.2 Linear Modeling of fMRI Data

In order to avoid confusion right from the beginning, in the fMRI community, the (general) linear model is abbreviated as "GLM." In statistics (and throughout this book) the acronym is used for *generalized linear models*. Thus, we will not adopt the GLM acronym for the analyses below since what we are going to fit is a basic linear model (i.e., a multiple regression) with some optional time series flavors. In the first part of this section, we focus on the analysis of a single subject. We will establish a design matrix and, subsequently, fit a linear model where the observed BOLD signal of a particular voxel i is regressed on the design matrix. Later on, we present how to perform analyses at a group level.

14.2.1 The Correlational Approach

One way of setting up the linear model is to first predict as accurately as possible what the BOLD response should look like in voxels affected by the stimulus. In order to take this route, we need to introduce a few things. The ability to measure the BOLD signal at time t allows us to assess an important quantity called *hemodynamic response function* (HRF) $h(t)$. The HRF is the BOLD signal in response to a stimulus with stimulus function $s(t)$. One way to assess $s(t)$ is to assume that the neural activation turns on instantly when the stimulus is presented, remains constant

throughout the duration of the stimulus, and drops instantly when the stimulus ends. This is the so-called *boxcar model*.

This *expected* or *predicted* BOLD signal depends on the stimulus function $s(t)$ and the HRF. The HRF is unknown. A common strategy, as used in the **fmri** package, is to compute the HRF by taking the difference of two gamma functions with some default parameterization (see Tabelow and Polzehl, 2011, for details). Subsequently, using a little mathematical trick called *convolution*, we can combine the stimulus function and the HRF in order to get the expected BOLD values $x(t)$:

$$x(t) = \int_0^\infty h(u)s(t-u)du \tag{14.3}$$

In order to compute $x(t)$ in R, we need the following information about the stimuli: the onset times, the duration of each stimulus, and the repetition time (TR). For the example used in this section, this information is provided in the **MPsychoR** package. Let us illustrate this computation for the first run of the first participant. In the experiment by Tamir et al. (2016a), each participant was exposed to 60 stimuli (mental states) in a single run, using an event-related design. Thus, we get a vector of length 60 containing onset times in sec and a vector of the same length for the durations. Note that we use the reaction times as a proxy for the duration. We add 1 sec since the stimulus was presented for 1 sec before the actual testing scenario appeared on the screen.

```
library("MPsychoR")
data("NeuralScanner")
onsetsS1R1 <- NeuralScanner$TRIAL_START[NeuralScanner[,2] == 1]
durationS1R1 <- NeuralScanner$RT[NeuralScanner[,2] == 1]  + 1
nt <- 162
```

A single run had 162 scans over time, with a TR of 2.5 s. Based on this information, the following function from the **fmri** package creates the expected BOLD response according to Eq. (14.3):

```
library("fmri")
xt <- fmri.stimulus(scans = nt, onsets = onsetsS1R1,
                    dur = durationS1R1, TR = 2.5, times = TRUE)
```

The corresponding time series is plotted in Fig. 14.4. This expected BOLD response does not necessarily look like a nice textbook BOLD response. The reason for this is that there are 60 conceptually different stimuli in this run, each of them representing a potentially different mental state. In addition, the stimulus duration depends on the reaction time, which makes them unequally spaced. More details will be given in Sect. 14.2.4 when we consider multiple runs.

14.2.2 Design Matrix

The next step is to establish a design matrix **X**. Design matrices in fMRI consist of two types of regressors. First, we have regressors reflecting *meaningful* contrasts related to experimental conditions. They involve the expected BOLD $x(t)$ and the associated regression parameters, subject to interpretation. Second, we have

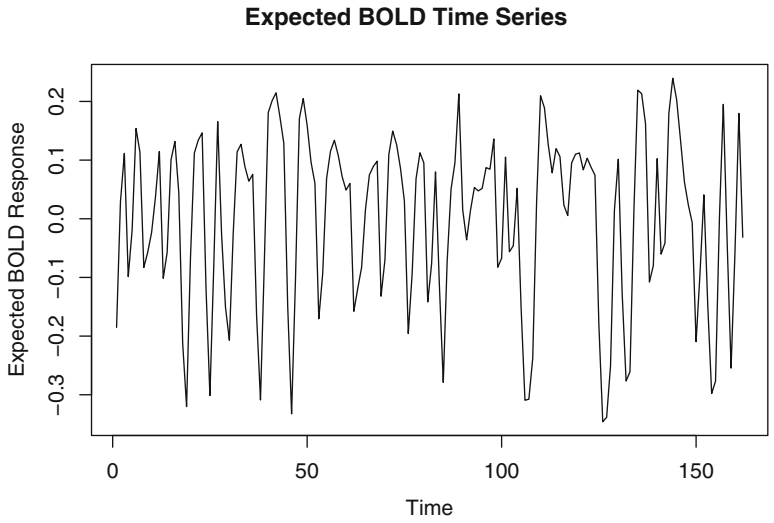

Fig. 14.4 Expected BOLD response over time (60 different stimuli, event-related design)

nuisance regressors of various kinds, not subject to interpretation. Typically, we include some drift terms which take into account general activation shifts during a run. Another option is to include head motion regressors as well, which we will do later on. For the moment let us keep it as simple as possible and create the following design matrix which includes a single $x(t)$ vector in the first column and three drift nuisance regressors (intercept, linear trend, polynomial trend) in the remaining columns:

```
X <- fmri.design(xt, order = 2)
head(X)
##             [,1] [,2]        [,3]         [,4]
## [1,] -0.18509908    1 -0.01464243 -0.064495515
## [2,]  0.02944904    1  0.02882160  0.010732222
## [3,]  0.11138879    1  0.05300512  0.039899573
## [4,] -0.09862145    1  0.03474075 -0.032611612
## [5,] -0.01960191    1  0.05849969 -0.004010516
```

(continued)

```
## [6,]  0.15387387    1  0.09599203  0.057787021
dim(X)
## [1] 162   4
```

This matrix is of dimension 162×4. The first column contains $x(t)$. The associated regression parameter will be of main interest, as we will elaborate on further below. The intercept incorporates the baseline activation. As a consequence, the slope for $x(t)$ will be different from 0 in task-related voxels only. In addition, a linear trend is included which captures the amount of linear signal drift that occurs

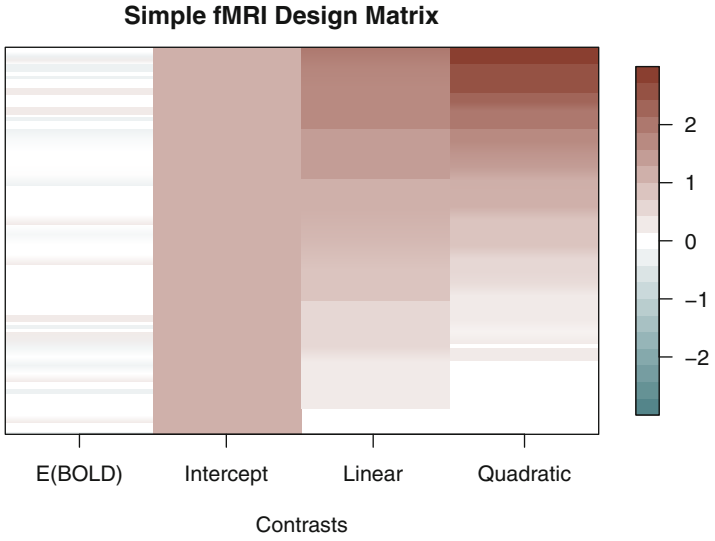

Fig. 14.5 Visual representation of a simple fMRI design matrix

at each TR. Here we also consider a quadratic drift term (fourth column), in case we do not believe in a simple linear drift.

Design matrices in fMRI can be very large. Therefore, they are often represented by means of an image plot as given in Fig. 14.5, which can be produced as follows:

```
library("fields")
col5 <- colorRampPalette(c('cadetblue4', 'white', 'coral4'))
collev <- 21
maxval <- max(abs(X))
colbreak <- seq(-maxval, maxval, length.out = collev + 1)
op <- par(mar = c(5,5,5,7))
image(t(X), axes = FALSE, main = "Simple fMRI Design Matrix",
```

(continued)

```
      xlab = "Contrasts", col = col5(n = collev), breaks = colbreak)
axis(1, at = c(0, 0.33, 0.66, 1),
     labels = c("E(BOLD)", "Intercept", "Linear", "Quadratic"))
box()
image.plot(t(X), legend.only = TRUE , col = col5(n = collev),
           breaks = colbreak)
par(op)
```

Since this run involves 60 different conditions, the first design vector does not really provide us with any helpful, substantive interpretation. Again, for illustration, we kept it as simple as possible here; a theoretically more meaningful design matrix involving multiple runs will be shown in Sect. 14.2.4.

14.2.3 Fitting the Linear Model

Now we are ready to fit the regression model. We work with our observed, preprocessed activation responses, using the flattened 2D $T \times N$ representation of the input data. Note that for each single voxel i, we have 162 measurements over time (indexed by t). This constitutes a time series for each voxel where the observed voxel response Y_{ti} is regressed on the design matrix:

$$Y_{ti} = \mathbf{X}\boldsymbol{\beta}_i + \varepsilon_{it}. \tag{14.4}$$

Technically, we could use the design matrix from above and perform an `lm` call for each voxel individually. This may oversimplify our setting since, as usual in time series, we cannot assume independence over time.[3] One strategy is to fit an autoregressive model of the form $\varepsilon_{it} = \varepsilon_{it-1}\rho_i + u_{it}$ (i.e., an AR(1) model) with ρ_i as the corresponding autocorrelation coefficient (see Sect. 13.3.2). The `fmri.lm` function in the **fmri** package provides an implementation including other time series flavors such as bias correction and smoothing of the autocorrelation coefficients. Details can be found in Tabelow and Polzehl (2011). Let us fit the linear model based on a single preprocessed scan:

```
imageS1R1 <- read.NIFTI("wFUN1.nii")
dim(extract.data(imageS1R1))
## [1]  79  97  80 162
spmS1R1 <- fmri.lm(imageS1R1, X)
```

[3] If we are not interested in inference on single subjects, autocorrelation is often ignored, and a simple `lm` call does the job.

Note that this function call fits the time series model for each voxel separately. Thus, we fit $79 \times 97 \times 80 = 613{,}040$ regression models, and each of them has four parameters as encoded in the design matrix. Let us extract the regression parameters, here organized as 4D structure:

```
dim(spmS1R1$beta)
## [1] 79 97 80  4
```

The resulting parameter structure is called a *statistical parametric map* (SPM), or a *subject specific contrast image*, or, simply, a β-map. As mentioned above, we are interested in the slope of the $x(t)$ effect. The structure in `spm$beta` returns all the regression parameters. The parameters of interest are the ones in `spmS1R1$beta[,,1]`: for each voxel we get a single meaningful β-parameter. This parameter denotes the correlation between the observed BOLD responses and the expected BOLD responses. Voxels with large parameter values (this can be tested, of course, as shown in the next section) are then identified as task related.

Note that we did not smooth the input data before fitting the model. As Tabelow and Polzehl (2011) suggest, we can apply a smoother on the SPM after fitting the model. This post-processing step typically reduces the variance of the estimated parameters and, in addition, the number of independent tests. The **fmri** package implements *structural adaptive smoothing*:

```
spmsmoothS1R1 <- fmri.smooth(spmS1R1)
```

If there is a need for a simpler, but faster, linear model approach, one can use the `lm4d` function from the **vows** package (Reiss et al., 2016). This implementation does not take into account autocorrelation. Smoothing can be achieved using `semipar4d` according to the methodology described in Reiss et al. (2014).

14.2.4 Example: Neural Representation of Mental States

Let us now extend the regression example from above to a more realistic scenario involving multiple runs per participant and a more informative design matrix within the context of this particular experiment. In the study by Tamir et al. (2016a), the authors were interested in neural activation patterns of 60 mental states. As mentioned, they chose an event-related design where, within each run, a participant was exposed to 60 stimuli. Each stimulus represented a different mental state. The mental states were presented in random order within each run. That is, in the second run, the participant was exposed to the same 60 conditions (under varying scenarios,

however) but in a different order. The onset times varied as well and, as outlined above, the stimulus duration was determined by the reaction time. Each participant had to perform 16 runs. Thus, each participant was exposed to $16 \times 60 = 960$ stimuli in total.

Let us now establish a design matrix involving theoretically meaningful regressors as well as some additional nuisance regressors. First, we reload the scanner information file which contains all necessary information about the experimental design and the scanning times and compute some basic quantities necessary for further analyses.

```
library("MPsychoR")
data("NeuralScanner")
nruns <- max(NeuralScanner[,2])   ## number of runs (16)
ncond <- max(NeuralScanner[,3])   ## number of conditions (60)
nmeas <- nrow(NeuralScanner)      ## number of measurents (960)
nt <- 162                         ## number of scans per run
durRun <- 400                     ## 1 run is 400sec
```

In the scanner file, after each run (400 s) the starting time of the new run is set to 0. Consequently, the onset times are relative to the starting time. Since we want to incorporate multiple runs, we compute the cumulative onset times.

```
add <- rep(seq(0, 6000, by = durRun), each = ncond)
TrStart <- NeuralScanner$TRIAL_START + add
```

We reorganize the onset times of each individual stimulus across the runs.

```
onsetsMat <- do.call(rbind, split(TrStart,
                     NeuralScanner$STATE_INDEX))
```

This leads to a matrix with 60 rows and 16 columns. For instance, element (2, 3) represents the onset of stimulus 2 on the third run.

We can restructure the stimulus duration in the same fashion.

```
durationMat <- do.call(rbind, split(NeuralScanner$RT + 1,
                       NeuralScanner$STATE_INDEX))
```

Now we compute the expected BOLD response for each condition separately:

```
library("fmri")
Xcond <- matrix(0, nt*nruns, ncond)
for (i in 1:ncond) {
  Xcond[,i] <- fmri.stimulus(scans = nt*nruns,
                onsets = onsetsMat[i,],
                dur = durationMat[i,], TR = 2.5, times = TRUE)
}
```

This leads to a matrix of dimension 2592×60. The number of rows results from the fact that we have 162 number of scans per run and 16 runs. This concludes the "meaningful regressors" part of the design matrix, containing the expected BOLD responses for each condition across all trials.

Let us work on the nuisance part, starting with the linear and quadratic trend parameterization.

```
ind <- rep(1, nt)           ## interept
lin <- (1:nt)/nt            ## linear trend
quad <- (0.5 - lin)^2       ## quadratic trend
Xtrend1 <- cbind(ind, lin, quad)
```

Note that this matrix contains the trend specification (intercept, linear, quadratic) for a single run. We can use the same values for the remaining runs. All we need to do is to "blow up" this matrix into a block-diagonal structure. What this means exactly will be obvious from the design matrix plot below. This matrix expansion can be achieved through the Kronecker product:

```
blow <- diag(1, nruns)
Xtrend <- blow %x% Xtrend1     ## expand to 16 runs
```

This matrix is of dimension 2592×48. The number of columns results from the fact that we have three trend vectors for each of the 16 runs.

Finally, we include some head motion nuisance parameters. They are provided in the **MPsychoR** package but can also be downloaded from the author's Dataverse repository (see Tamir et al., 2016b, 16 `txt`-files starting with `rp`). Here, they are organized as a list of length 16. Let us incorporate them into the design matrix by a similar block-diagonal strategy as above. We need to take into account that they are different across the runs.

```
library("magic")
data("NeuralHM")
Xhm <- NeuralHM[[1]]
for (i in 2:length(NeuralHM)) {
  Xhm <- adiag(Xhm, NeuralHM[[i]])
}
```

The resulting matrix is of dimension 2592×96, since we have six head motion parameters for each of the 16 runs. All that is left to do is to combine these three matrices into the full design matrix.

```
X <- cbind(Xcond, Xtrend, Xhm)
```

The final design matrix is of dimension 2592×204 and is plotted in Fig. 14.6. The block structures for the nuisance parameters become obvious using this visualization.

For illustration, in order to save running time (and memory), we will fit the regression model on 3 runs instead of the full 16 runs. We subset the design matrix accordingly.[4]

```
nruns <- 3
X3 <- X[1:(nt*nruns), ]
X3 <- X3[, colSums(X3) != 0]
```

The reduced design matrix is of dimension 486×87. Let us re-import the global 3D mask which applies to all runs.

```
mask <- read.ANALYZE("globalmask.img")
mask <- extract.data(mask)[,,,1]
```

Next we import the scans. Note that these scans are preprocessed according to the pipeline described in Sect. 14.1.2, but without smoothing. Therefore, let us apply Gaussian kernel smoothing:

[4]The reader can change `nruns` argument for fitting a different number of runs.

```
scanfiles <- paste0("wFUN", 1:nruns, ".nii")
imageS1 <- array(0, dim = c(dim(mask), nt*nruns))
sigma <- 2
for (i in 1:nruns) {
  scanRi <- read.NIFTI(scanfiles[i])
  scanRi <- extract.data(scanRi)
  scanRi <- GaussSmoothArray(scanRi, sigma = diag(sigma^2,3),
                             mask = mask)
  start4d <- (i-1)*nt + 1
  end4d <- i*nt
  imageS1[,,,start4d:end4d] <- scanRi
}
```

If we want to use the `fmri.lm` function, the data need to be provided as an object of class `"fmridata"`, whereas our `imageS1` is in array format. One easy option to do this conversion is to first store the image on the hard drive.

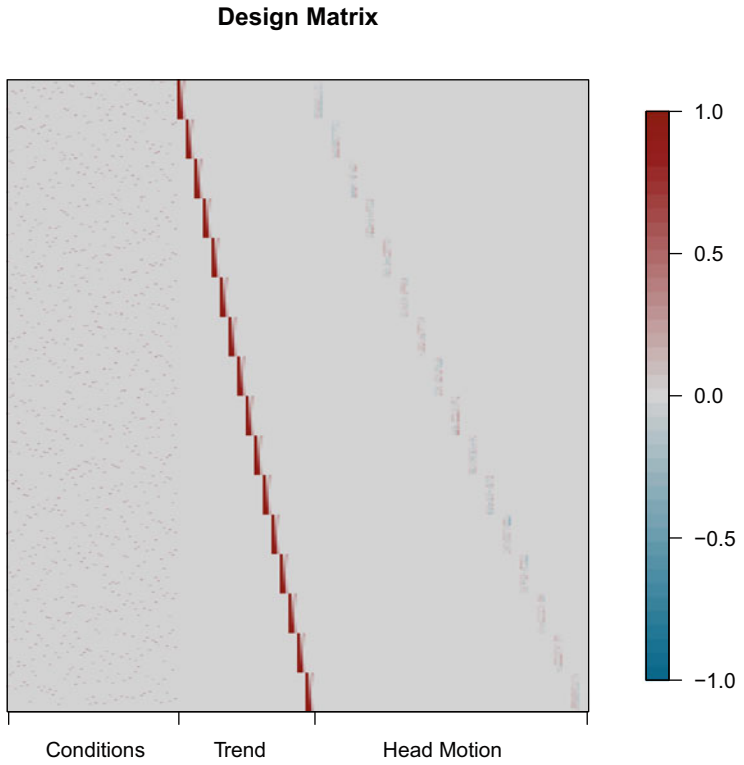

Fig. 14.6 Visual representation of a complex fMRI design matrix with conditional, trend, and head motion components

```
f.write.nifti(imageS1, "imageS1", nii = TRUE)
```

Second, we re-import the data and the new object is of class `"fmridata"`.

```
imageS1 <- read.NIFTI("imageS1")
```

As a final data preparation step, we assign the mask (Boolean version) to this object:

```
maskTF <- twoDto4D(as.logical(mask), dim = dim(mask))
imageS1$mask <- maskTF
```

Finally, we are ready to fit the regression model using `fmri.lm`. For running time and memory purposes, we avoid computing the autocorrelation.

```
spm3 <- fmri.lm(imageS1, X3, actype = "noac")
```

The contrast map is of the following structure:

```
str(spm3$beta)
##  num [1:79, 1:97, 1:80, 1:87] 0 0 0 0 0 0 0 0 0 0 ...
```

The first three dimensions are the voxel grid. Note that for each voxel a regression model was estimated ($N = 79 \times 97 \times 80 = 613{,}040$ models). The fourth dimension relates to the 87 parameters in the regression model. For instance, the regression parameter for voxel (1, 1, 1) can be extracted by `spm3$beta[1, 1, 1, ]`, which results in a vector of length 87. We can flatten the 4D SPM structure to a 2D SPM as follows:

```
spm3_2D <- t(fourDto2D(spm3$beta, dim(spm3$beta)[4]))[,1:60]
dim(spm3_2D)
## [1] 613040     60
```

In the first line, we transpose the SPM such that we have the contrasts in the columns and select the first 60 columns right away since they contain the meaningful parameters (stimulus contrasts). The remaining 27 parameters are nuisance (trend, head motion) and are of no further interest.

This concludes our basic linear modeling strategy. Note that we ran the analysis for a single participant only. We need to apply the same modeling strategy to each of the 20 individuals which leads to 20 different SPMs of dimension $N \times 60$.

14.2.5 Group Analysis

At this point we assume that a linear model was fitted for each of the $n = 20$ individuals. In this section we perform between-subject tests, that is, one-sample t-tests for each voxel separately (H_0: $\beta_i = 0$) based on the parameters of the 20 subjects. The corresponding SPMs can be downloaded from the Dataverse repository (Tamir et al., 2016b). The authors provide SPMs for each of the 60 stimulus separately, one for each participant. The files are in Analyze format (`img/hdr`) and start with `con`. Let us import the one for the first stimulus/condition (mental state "affection") for all n participants, apply a smoother, vectorize the voxel parameters, and restructure it as a matrix of dimension $n \times N$.

```
mask <- read.ANALYZE("globalmask.img")
mask <- extract.data(mask)[,,,1]
imgdim <- dim(mask)
n <- 20
sigma <- 2
files <- paste0("con_s", sprintf("%02d", 1:20), "_0002")
spmC1 <- numeric()
for (i in 1:n) {
  spmi <- read.ANALYZE(files[i])
  spmi <- extract.data(spmi)[,,,1]
  spmiSmooth <- GaussSmoothArray(spmi, sigma = diag(sigma^2,3),
                                 mask = mask)
  spmiSmooth <- spmiSmooth[mask != 0]
  spmC1 <- rbind(spmC1, spmiSmooth)
}
```

By considering group-level data, we have a hierarchical structure: voxels within subjects. Thus, a mixed-effects model would be an appealing modeling approach. However, the data are too large for explicit hierarchical modeling using standard R mixed-effects packages. An efficient implementation of a Bayesian hierarchical model is provided by the **cudaBayesreg** package (da Silva, 2011), which requires a specific GPU (graphics processing unit) setup, however.

A simpler approach to tackle this problem, commonly used in fMRI, is referred to as *summary statistics*. After fitting the linear model for each participant, a one-sample t-test is applied on each voxel individually.

```
library("genefilter")
fitT <- colttests(spmC1, gl(1, nrow(spmC1)))
```

Instead of the basic `t.test` function, we use a fast t-test implementation from the **genefilter** package (Gentleman et al., 2016) available from the Bioconductor repository. We get 170,686 t-values and the same amount of uncorrected p-values, one for each voxel. The following call examines how many p-values are significant (i.e., responsive to the stimulus):

```
sum(fitT$p.value <= 0.05)
## [1] 49048
```

As we will explain in the next section, there is a problem with such an uncorrected significance evaluation since we have a massive multiple testing problem.

14.3 Multiple Comparisons in fMRI

In order to illustrate the multiple testing problem in fMRI, let us consider a simple example from Ashby (2011). Let us assume that the data consist of 100,000 voxels, and for each voxel i, we computed a t-value t_i using the strategies presented above. Each of these test statistics was constructed under the H_0 that the voxel was not responsive to the stimulus. We perform statistical inference simultaneously on $N = 100{,}000$ voxels. Fixing the Type I error rate to $\alpha = 0.05$ and assuming that none of the voxels is responsive, we can expect that 5000 tests would nevertheless reject H_0 (*false positives*) simply due to chance. Let us assume that 6000 t_i's were significant. We know that only 1000 out of these 6000 are actually significant due to activation; the remaining 5000 ones are significant due to chance. However, we do not know which ones are significant due to activation and chance, respectively. This is obviously a problem. The good news is that such (huge) multiple testing problems are nothing new and entire books have been written on this topic (see, e.g., Bretz et al., 2011). The **multcomp** package (Hothorn et al., 2008) provides corresponding implementations.

Let us formalize this multiple testing problem using a much simpler setup and briefly elaborate on some classical statistical approaches, before showing more specific techniques used in fMRI. If we test at an $\alpha = 0.05$ level, we know that the

probability for falsely rejecting H_0 is 0.05 (Type I error). Therefore, the probability for not making a Type I error is 0.95. In the case of three simultaneous t-tests, the probability of no Type I error at all is $0.95 \times 0.95 \times 0.95 = 0.857$. The probability of making at least one Type I error is correspondingly $1 - 0.857 = 0.143$. Thus, across this group of tests, the probability of making a Type I error has increased from 5% to 14.3%. We have the problem of multiple testing and this error rate is called *family-wise error* (FWE):

$$\text{FWE} = 1 - (1 - \alpha)^N, \tag{14.5}$$

with N equal to the number of comparisons. In the case of $N = 100{,}000$, as above, the FWE is virtually 1.

Early approaches such as Sidák and Bonferroni corrected α-level as follows: $\alpha^* = 1 - (1 - \alpha)^{1/N}$ (Sidák) and $\alpha^* = \alpha/N$ (Bonferroni). For illustration, let us compute a simple Bonferroni correction using the t-test output from the previous section.

```
nvox <- length(fitT$statistic)
alphanew <- 0.05/nvox
sum(fitT$p.value <= alphanew)
## [1] 5
```

This result suggests that only 5 voxels are activated. Such simple approaches are typically not used in fMRI, for the following reasons. First, the number of tests is huge. Thus, there is a good chance that we miss many true positives. Bonferroni, Sidák, and friends are too conservative. Second, these approaches assume that the tests are independent from each other. This is not met in fMRI since the activation of a particular voxel i influences the activation in its neighborhood. Therefore, we need something more sophisticated. State-of-the-art multiple testing approaches in fMRI are the following:

1. controlling for the *false discovery rate* (FDR);
2. *Gaussian random field* (GRF) theory;
3. *permutation tests*.

For permutation tests, in addition to voxel-based thresholding, we also show cluster-based thresholding. Another option, not shown in this chapter, is to use Monte Carlo approaches. An excellent overview of these techniques is given in Nichols (2012).

14.3.1 Controlling for the FDR

Let us start with the FDR approach developed by Benjamini and Hochberg (1995) and described in Genovese et al. (2002) within an fMRI context. Instead of controlling for the FWE, approaches based on the FDR aim to directly limit the proportion of significant results that are false positive. We can compute the FDR for a limit q (e.g., $q = 0.05$), which denotes the proportion of the voxels for which H_0 is incorrectly rejected. Whereas FWE approaches control for the probability of at least one Type I error, by using the FDR, we have a less rigorous control of the Type I error (but we have a higher power).

An easy way to apply the FDR procedure in R is to use the `p.adjust` function with the vector of p-values from the t-tests as input.

```
pFDR <- p.adjust(fitT$p.value, method = "fdr")
sum(pFDR <= 0.05)
## [1] 22506
```

We see that, compared to Bonferroni, the number of significant voxels increased drastically.

14.3.2 Gaussian Random Fields

The second approach uses GRFs. The theory behind this method is quite complicated and beyond the scope of this book. Interested readers can check out Adler (1981) and Worsley et al. (1996) for more technical details. As Ashby (2011) elaborates, the basic idea is that BOLD responses collected on any trial of an fMRI experiment can be described as a random field over a lattice of points. A random field is an ordered collection of random variables; a GRF assumes that these random variables have a joint multinormal distribution. GRFs are, to some extent, able to model spatial correlations among voxels. They can be constructed as follows:

1. Create an empty voxel map of the same voxel dimensions as the one from the actual data.
2. Draw independent samples from an $N(0, 1)$ distribution and insert these values into the voxel map.
3. Spatially smooth these voxel values in the same way as elaborated in Sect. 14.1.2 on preprocessing.

In the last step, the kernel width is again determined by the FWHM, one for each dimension. The total amount of smoothing can be expressed by the so-called *resel* (resolution elements), which is a virtual voxel with dimensions equal to

$FWHM_x \times FWHM_y \times FWHM_z$. Note that when preprocessing fMRI data, we smoothed over correlated data. Here, we smooth over independent data. Thus, using the GRF for multiple comparisons underestimates the correlation and is known to be conservative. However, the GRF approach gives us a threshold T above which a voxel can be considered as activated.

The following function from the **AnalyzeFMRI** package calculates the threshold. We use the same σ as in the preprocessing part.

```
sigma <- 2
covmat <- diag(sigma^2,3)
trf <- Threshold.RF(p.val = 0.05, sigma = covmat,
                    num.vox = nvox, type = "t", df = nvox-1)
trf
## [1] 4.75902
sum(fitT$statistic >= trf)
## [1] 3195
```

This is the number of activated voxels. It is clearly smaller than one obtained using FDR which reflects the conservative nature of this procedure.

14.3.3 Permutation Tests

Finally, we can also define a permutation setup to deal with the multiple testing problem. Permutation tests are a general nonparametric testing framework where the sampling distribution under the null is created by shuffling the observed data accordingly. On each permutation sample, the corresponding test statistic is computed. These test statistics form a null distribution, the observed value of the test statistic is then mapped onto this distribution, and a corresponding test decision can be achieved.

In fMRI, permutation tests are often used to account for the multiple comparisons problem. The standard reference for permutation tests in fMRI is Nichols and Holmes (2001). The advantage of this technique is that it is very flexible and requires only minimal assumptions (i.e., *exchangeability*). Note that using permutation tests on the basic fMRI time series data would violate exchangeability due to autocorrelation. However, applying permutation tests on a group-level parametric map is feasible. The disadvantage is that it can be very time-consuming; using the **genefilter** for t-test computations is a life-saver here in terms of running time.

We use a one-sample t-test strategy in order to search over the whole brain for significant changes. Nichols and Holmes (2001) use the *sign test* principle to obtain the null distribution. They consider subject labels of "+1" and "−1" indicating an unflipped or flipped sign of the data. Under the null hypothesis, the distribution is

symmetric around 0. Thus, for a particular participant, the sign of the observed data can be flipped without changing its distribution. Exchangeability across subjects holds since we can safely assume that they are independent from each other.

In our example, with $n = 20$, we could perform 2^{20} different permutations in total. This would take a long time to compute and it is actually not necessary. Let us generate 1000 random sign permutations for the participants. For each of the 1000 permutation matrices, we compute t-tests for each voxel and pull out the maximum t-statistic, since we are interested in searching over the whole brain for significant changes. At the end of this procedure, which takes about a minute to run, we have 1000 max-t values which form the permutation distribution.

```
set.seed(123)
nperm <- 1000
maxtvec <- numeric(nperm)
for (i in 1:nperm) {
   print(i)
   signperm <- sample(c(-1, 1), n, replace = TRUE)
   permmat <- spmC1*signperm
   maxtvec[i] <- max(colttests(permmat,
                     gl(1, nrow(permmat)))$statistic)
}
```

The permutation distribution is given in Fig. 14.7. Based on this distribution, we can easily obtain the critical t-value (one-sided testing, therefore 95% quantile on the sampling distribution) and count how many voxels are above this threshold (i.e., the ones that show a significant activation).

```
tcrit <- quantile(maxtvec, probs = 0.95)
tcrit
##      95%
## 6.052349
sum(fitT$statistic >= tcrit)
## [1] 410
```

So far we performed voxel-wise tests only. Let us extend the voxel-wise permutation framework to cluster-wise testing (*cluster-based thresholding*). The idea is the following. We choose a p-value threshold of 0.001 (default in the literature; see, e.g., Woo et al., 2014) and assign a value of 1 to the voxels which are below this threshold and 0 otherwise. We also restructure the outcomes as a 3D array right away.

```
pthresh <- 0.001
p01obs <- array(ifelse(fitT$p.value <= pthresh, 1, 0), imgdim)
```

The next step is find voxel clusters based on this binary matrix. We need an algorithm which works in the 3D space and leads to *contiguous* clusters (i.e., a point is closer to one or more other points in its cluster than to any other point not in that cluster). The **mmand** package (Clayden, 2016a) implements a corresponding function based on a kernel. Here we use a width of 5 voxels in each direction. In general it applies that the larger the kernel width the larger the clusters will be.[5]

```
library("mmand")
kwidth <- 5
kernel <- shapeKernel(kwidth, dim = 3)
compsobs <- components(p01obs, kernel)
```

Next we extract the size of each cluster:

```
clustsize <- table(c(compsobs), useNA = "no")
```

We get 506 clusters of different sizes. Now we need to find a null distribution. We apply exactly the same permutation principle as above, extract the p-values from the t-tests, dichotomize them using the 0.001 threshold, perform the clustering, and extract the maximum cluster size S for each permutation. The resulting permutation distribution is given in the bottom panel of Fig. 14.7.

```
set.seed(111)
nperm <- 1000
maxS <- numeric(nperm)
for (i in 1:nperm) {
  print(i)
  signperm <- sample(c(-1, 1), n, replace = TRUE)
  permmat <- spmC1*signperm
  pvec <- colttests(permmat, gl(1, nrow(permmat)))$p.value
  p01perm <- array(ifelse(pvec <= pthresh, 1, 0), imgdim)
```

(continued)

[5] A width of 5 led to a reasonable number of clusters in our example. Typically we would use a width of 1 such that we get completely contiguous clusters.

```
  comps <- components(p01perm, kernel)
  maxS[i] <- suppressWarnings(max(table(c(comps), useNA = "no")))
}
```

All that is left to do is to compare the observed cluster sizes with the critical value of the permutation distribution, which gives us the activated clusters.

```
Scrit <- quantile(maxS, probs = 0.95)
Scrit
## 95%
##  11
sum(clustsize >= Scrit)
## [1] 66
```

A recent discussion on cluster-based thresholding with respect to inflated false-positive rates can be found in Eklund et al. (2016).

Additional options for applying permutation tests in fMRI include scenarios where groups are involved. A simple example with a treatment and control group is presented in Eloyan et al. (2014). In this case the one-sample t-test approach from above generalizes to a two-sample t-test. For more than two groups, more general ANOVA/regression specifications can be considered. The function `permF.mp` in the **vows** package can be used for these purposes. For other permutation (and bootstrap) strategies that can be used for fMRI data settings, see the `MTP` function in **multtest** (Pollard et al., 2005).

14.4 Independent Component Analysis in fMRI

Using linear model approaches as above, we conduct our analyses on a univariate voxel level. A multivariate method that can be applied for complexity reduction is *principal component analysis* (PCA; see Chap. 6). However, PCA is not widely applied in fMRI except to eliminate noise in the data (see Ashby, 2011, Chapter 10). A more attractive method for complexity reduction in fMRI data is *independent component analysis* (ICA), introduced in Sect. 6.4. Having spatiotemporal data as in fMRI, ICA can be carried out in two directions:

- *spatial ICA*: assumes that the spatial brain networks are independent.
- *temporal ICA*: assumes that the time courses are independent.

In spatial ICA, components are spatial maps, and the weights contain temporal information. In temporal ICA, components are temporal waveforms, and a weight is estimated for every voxel (Ashby, 2011). The **AnalyzeFMRI** package provides

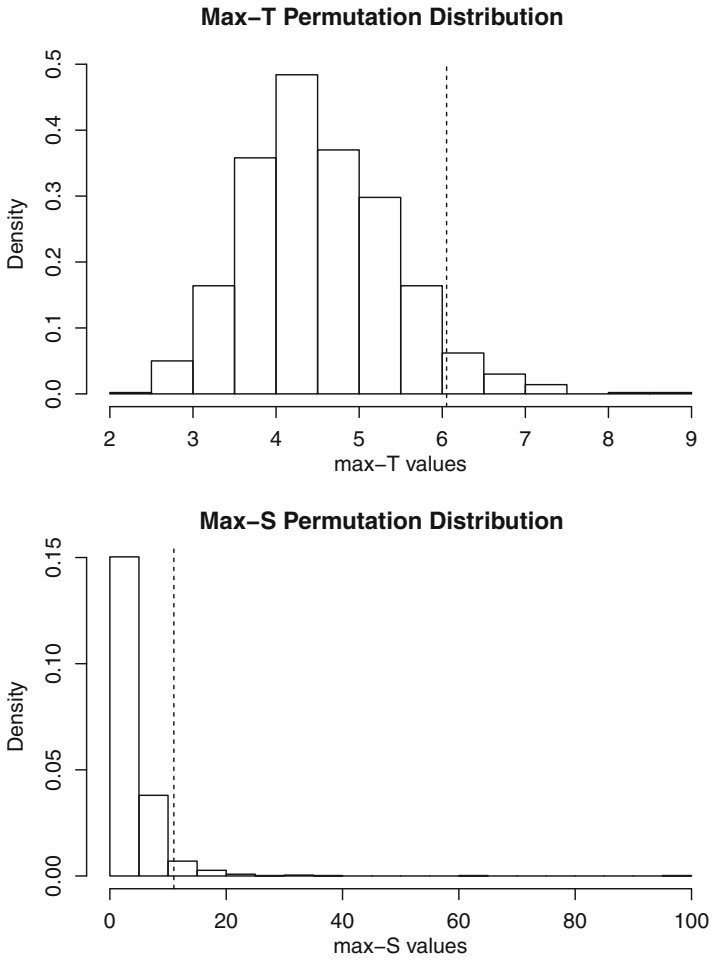

Fig. 14.7 Top panel: sampling distribution of the max-T statistics (voxel-wise). Bottom panel: sampling distribution of the max-S statistics (cluster-wise). The dotted lines represent the critical values

the function `f.icast.fmri` which can do both spatial and temporal ICA (see Bordier et al., 2011, for details). Spatial ICA is more common in fMRI.

The core ICA function in **AnalyzeFMRI** is `f.icast.fmri`. It imports the data from a local file, either in NIfTI or in Analyze format, rather than having it as a preexisting R object. Let us start with fitting a spatial ICA on a single subject, involving three runs. The data were prepared in Sect. 14.2.4 (`imageS1` object). The following line stores the data object in NIfTI format on the hard drive:

```
library("AnalyzeFMRI")
f.write.nifti(imageS1, "imageS1", nii = TRUE)
```

The code chunk below performs the spatial ICA. By default, the algorithm estimates the number of components m automatically (see Bordier et al., 2011, for a corresponding algorithm description). Alternatively, they can also be fixed using the `n.comp` argument.

```
f.icast.fmri("imageS1.nii", "globalmask.img", is.spatial = TRUE)
```

The results are written into two different output files. The first one is `"imageS1-ICAs-time-series.dat"` which contains the time series for each component. Let us import this file into R.

```
tsICAs <- as.matrix(read.table("imageS1-ICAs-time-series.dat",
                               header = TRUE, row.names = 1))
str(tsICAs)
##  num [1:486, 1:33] -0.6 -1.004 -1.316 -0.675 -0.771 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:486] "1" "2" "3" "4" ...
##   ..$ : chr [1:33] "V1" "V2" "V3" "V4" ...
```

We see that 33 independent components (ICs) have been extracted. The rows reflect the number of scans across the three runs (i.e., $162 \times 3 = 486$).

The second file (`"imageS1_ICAs.nii"`) contains the new images according to the ICs, which can be used for plotting by calling `f.plot.volume.gui`. In the first graphical user interface (GUI), prompt the new image file and the time series file need to be selected. The anatomical scan is optional.

Here, we plot some results directly from the command line without using the GUI. For illustration we select the first IC and the 40th slice and then plot the activation pattern as well as the time series containing the weights (see Fig. 14.8).

```
library("ggplot2")
library("cowplot")
imgICAS1 <- read.NIFTI("imageS1_ICAs.nii")
imgICAS1 <- extract.data(imgICAS1)[,,,1]
tsdf <- data.frame(time = 1:486, IC1 = tsICAs[,1])
bp <- ggBrain(brains = imgICAS1, mask = mask, mar = 3,
```

(continued)

```
              mar_ind = 40, type = 'signed') + theme_gray()
tsp <- ggplot(tsdf,
              aes(time, IC1)) + geom_line() + theme_gray()
plot_grid(bp, tsp, labels=c("Activation", "Time Series"),
          ncol = 1, nrow = 2)
```

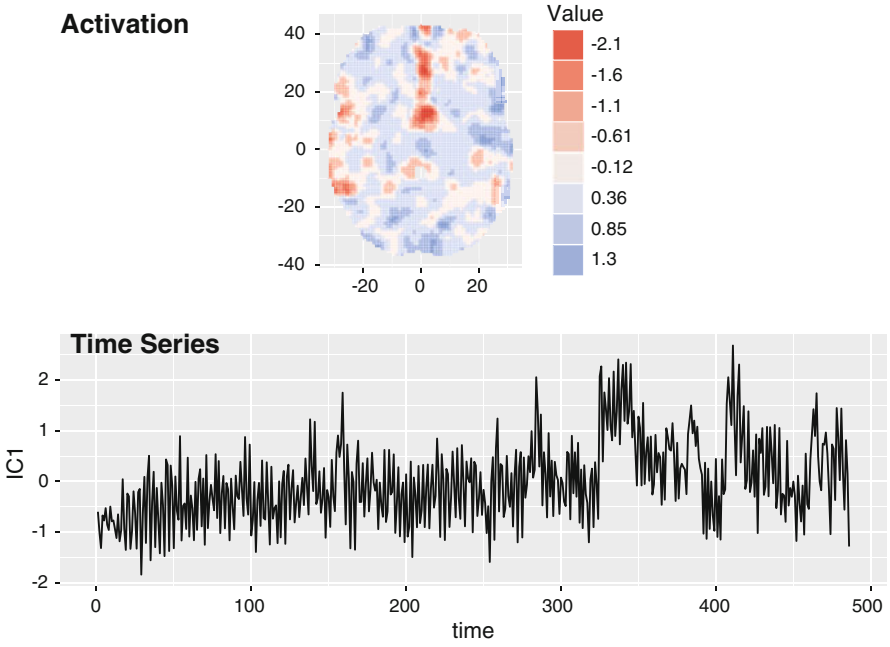

Fig. 14.8 Top panel: activation pattern of the first IC. Bottom panel: time series of the first IC

Another version of ICA, occasionally used in fMRI, is called *group ICA*, which represents a multi-subject version of ICA (see Calhoun et al., 2009, for an overview). From a data preparation point of view, the trick is to stack the single subject input data on top of each other and carry out the ICA as elaborated above. R code for group ICA can be found in Eloyan et al. (2014).

14.5 Representational Similarity Analysis

Representational similarity analysis (RSA Kriegeskorte et al., 2008) is an umbrella term for fMRI approaches that aim to represent voxel/ROI similarities in multivariate way. Here we focus on a variant which makes use of what we have learned in Chap. 9 on multidimensional scaling (MDS).

The starting point for our RSA is the SPM output from the general linear model fit as described in Sect. 14.2.4. Continuing with the dataset used so far in this chapter, we have a 2D-SPM (`spm3_2D` object created in Sect. 14.2.4) of dimension 613,040 $\times$ 60 for each participant. Let K denote the number of columns (i.e., stimuli/conditions) and N the number of voxels. The first step in RSA is to calculate a *representational dissimilarity matrix* $\boldsymbol{\Delta}$ for the conditions. Let us compute the correlations across the 60 mental states and, subsequently, convert them into dissimilarities. The correlation matrix contains Pearson correlations for each pair of conditions. These correlations are based on voxel activity patterns. That is, if there is a high correlation between two stimuli, this suggests that similar voxels have been activated with similar intensities.

```
library("smacof")
R <- cor(spm3_2D)
RDM <- sim2diss(R)
```

This results a 60 $\times$ 60 dissimilarity matrix for a single participant i, here stored as object of class `"dist"`. We need to compute a $\boldsymbol{\Delta}_i$ for each of the 20 participants ($i = 1, \ldots, 20$). These matrices are already prepared in the **MPsychoR** package, stored as a list object.

```
library("MPsychoR")
data("NeuralActivity")
```

Note that these $\boldsymbol{\Delta}_i$ are based on the β-maps resulting from a linear model fit on the full 16 runs. Remember that the `spm3_2D` object above was based on three runs only and, therefore, it does not exactly match the first matrix in the list. Let us continue with the full list.

The next step is to choose a data-analytic method that allows us to represent these dissimilarities. We could now fit the INDSCAL model, as we did in Sect. 9.5.2, based on all 20 matrices. We would get group stimulus space which shows the group configuration of the conditions. Here we keep it simple and compute the element-wise means across 20 dissimilarity matrices.

```
Delta <- Reduce("+", NeuralActivity)/n
```

This leads to a new dissimilarity matrix Δ of dimension 60 $\times$ 60, subject to MDS using the **smacof** package (De Leeuw and Mair, 2009). We use an interval MDS and project into three dimensions.

```
RSAfit <- mds(Delta, ndim = 3, type = "interval")
RSAfit
##
## Call:
## mds(delta = Delta, ndim = 3, type = "interval")
##
## Model: Symmetric SMACOF
## Number of objects: 60
## Stress-1 value: 0.21
## Number of iterations: 206
```

Figure 14.9 represents the 3D configuration plot. We color the labels according to the SPP (stress-per-point; see Sect. 9.2.3). Remember that a point with high

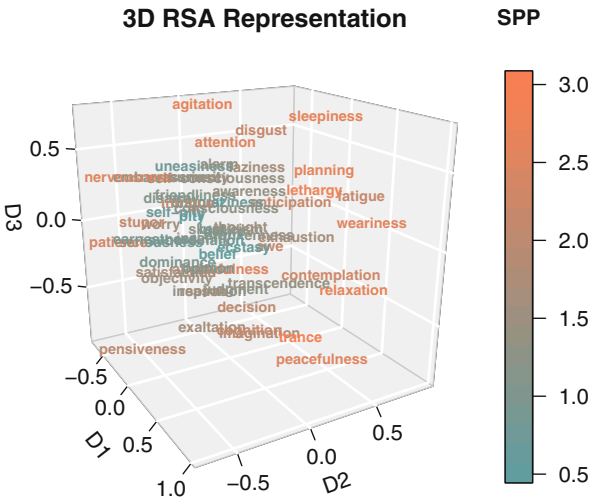

Fig. 14.9 RSA representation of 60 conditions based on a 3D MDS fit. The labels are colored according to the stress-per-point (SPP)

SPP suggests that this point does not fit well into the configuration with given dimensionality.

This analysis is a simple form of RSA on neural activity patterns. Additional details on RSA involving other scaling techniques and various model evaluation approaches are presented in Kriegeskorte et al. (2008).

14.6 Functional Connectivity Analysis

Functional connectivity analysis refers to methods involving the statistical analysis of anatomically distinct time series (Friston, 1994). At this point of the chapter, we have to abandon our example dataset since the following methods are difficult to apply on event-related designs.

The example dataset we use is from the **brainwaver** package (Achard, 2012) and comes from a simple resting-state experiment performed on a single subject (Achard et al., 2006). Regional parcellation was achieved through the AAL90 atlas.

```
library("brainGraph")
library("brainwaver")
data(brain)
colnames(brain) <- aal90$name
dim(brain)
## [1] 2048   90
```

In this dataset we have 2048 measurements (time series) on 90 ROIs (45 for each hemisphere), collected while the participant was lying quietly at rest. Using these data we elaborate on two approaches for connectivity analysis.

14.6.1 Seed-Based Correlational Analysis

The first modeling option we present is a very simple one, called *seed-based correlational analysis* (SCA). We start with picking a *seed* ROI (or voxel) of interest. For this particular ROI (or voxel), we extract the time series. Here we use the left precentral gyrus as seed time series.

```
PreCG.L <- brain[,1]  ## seed time series
```

The idea of SCA is to correlate the seed time series with the time series of the remaining ROIs:

```
library("psych")
ROIs <- brain[,-1]    ## remaining ROI time series
corvec <- cor(PreCG.L, ROIs)  ## correlation
zvec <- fisherz(corvec)       ## z-transformation
```

The last line using a function from the **psych** package (Revelle, 2017) transforms the correlations into z-scores according to Fisher's classical formula, often applied in SCA:

$$z = 0.5\frac{\log(1+r)}{\log(1+r)}. \tag{14.6}$$

Based on the AAL90 spatial coordinates and the z-transformed correlation vector, Fig. 14.10 can be produced as follows:

```
SCAdf <- data.frame(x = aal90$x, y = aal90$y, names = aal90$name,
                    z = c(NA, zvec))
sca <- ggplot(SCAdf, aes(x = x, y = y, label = names, color = z)) +
  scale_color_gradient(low = "white", high = "cadetblue", na.value = "black")
sca + geom_text(size = 3) + ggtitle("Seed-Based Correlations") +
  theme(axis.title.x = element_blank(), axis.text.x = element_blank(),
        axis.ticks.x = element_blank(), axis.title.y = element_blank(),
        axis.text.y = element_blank(),  axis.ticks.y = element_blank())
```

The color shading is based on the z-transformed correlations and indicates how high each ROI is correlated with the left precentral gyrus seed. Note that we could have also used partial correlations and produce the plot in an analogous manner.

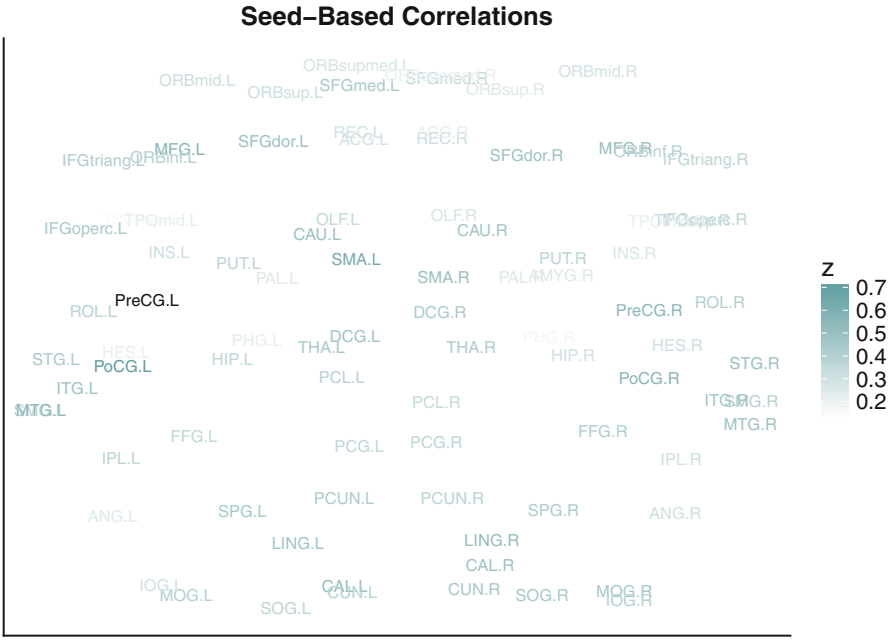

Fig. 14.10 Seed correlation plot (z-transformed correlation values) with left precentral gyrus (PreCG.L) as seed time series. The ROIs are plotted according to the AAL90 coordinates

14.6.2 Wavelet Correlational Analysis

Now let us do something more sophisticated and compute the correlation matrix based on a *discrete wavelet transform*. Wavelets are widely used in signal analysis for information extraction. In fMRI they are typically applied on the activation time series in order to estimate frequency-dependent correlation matrices to characterize functional connectivity among ROIs. Extensive treatments of wavelets in fMRI can be found in Bullmore et al. (2004) and Lazar (2008).

Let us compute the wavelet correlation matrices using the **brainwaver** package. We choose a decomposition depth of six, resulting in the following six frequency intervals: 0.23–0.45 Hz, 0.11–0.23 Hz, 0.06–0.11 Hz, 0.03–0.06 Hz, 0.01–0.03 Hz, and 0.007–0.01 Hz.

```
wavecor <- const.cor.list(as.matrix(brain), n.levels = 6)
```

The matrices in this object (`$d1` through `$d6`) can be understood as representing the interregional functional connectivity that is subtended by time series components in the frequency bands.

Let us consider the correlation matrix of the fourth interval (0.03–0.06 Hz; see Fig. 14.11, left panel). As Achard et al. (2006) point out, in this particular experiment the brain functional connectivity was most salient in this frequency band.

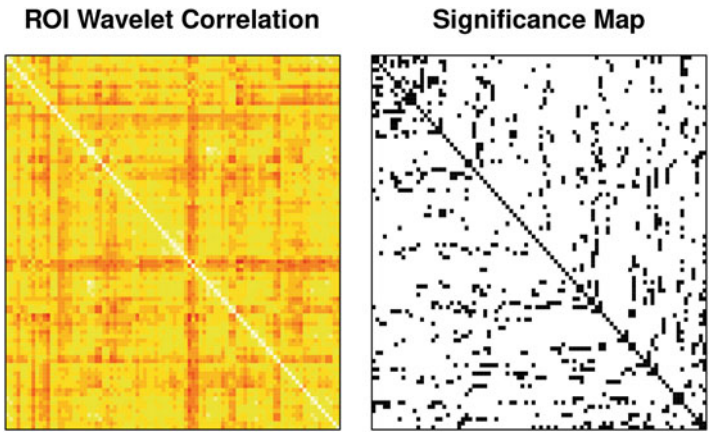

Fig. 14.11 Left panel: wavelet correlation matrix (0.03–0.06 Hz) for 90 ROIs. Right panel: significant correlations (black) for correlation threshold $R = 0.4$

We can now set a correlation threshold (we use a value of 0.4 since the mean wavelet correlation coefficient was maximal for this value), compute p-values, and determine via the FDR ($q = 0.05$) which cells are significant.

```
R <- 0.4
pvalues <- p.value.compute(wavecor[[4]],
  proc.length = nrow(brain), sup = R, num.levels = 4)
FDRthresh <- compute.FDR(pvalues, q = 0.05)
p01vec <- ifelse(pvalues <= FDRthresh, 1, 0)
p01 <- diag(0, ncol(brain))
p01[lower.tri(p01)] <- p01vec
p01 <- p01 + t(p01)
diag(p01) <- 1
```

The right panel in Fig. 14.11 shows which ROI correlations are significantly larger than the threshold.

The binary significance correlation matrix can be subject for further scaling and representations. For instance, we could run an MDS either on the binarized matrix or also directly on the wavelet correlation matrix (after proper conversion into a dissimilarity matrix). We can also apply simple network approaches from Chap. 11. Here we compute an *eigenmodel network*, using the **eigenmodel** package (Hoff, 2012), as described in Sect. 11.3.1.

```
library("eigenmodel")
diag(p01) <- NA
colnames(p01) <- rownames(p01) <- colnames(brain)
brainNet <- eigenmodel_mcmc(p01, R = 2, S = 1000, burn = 200,
                            seed = 123)
evecs <- eigen(brainNet$ULU_postmean)$vec[, 1:2]
```

The resulting eigenvectors are subject to plotting. Figure 14.12 shows the ROI network plot.

Further options to perform analyses using wavelet correlations including other ways of representing the results can be found in Achard et al. (2006). In addition, the **brainGraph** package offers a wide range of functionalities for computing and plotting brain networks.

14.7 Conclusion and Outlook

There are several other options to analyze fMRI data in R. Some of them will be pointed out here. Dynamic causal modeling is a technique that allows the user to test specific assumptions about functional connectivity. The **FIAR** package (Roelstraete and Rosseel, 2011) provides a corresponding implementation.

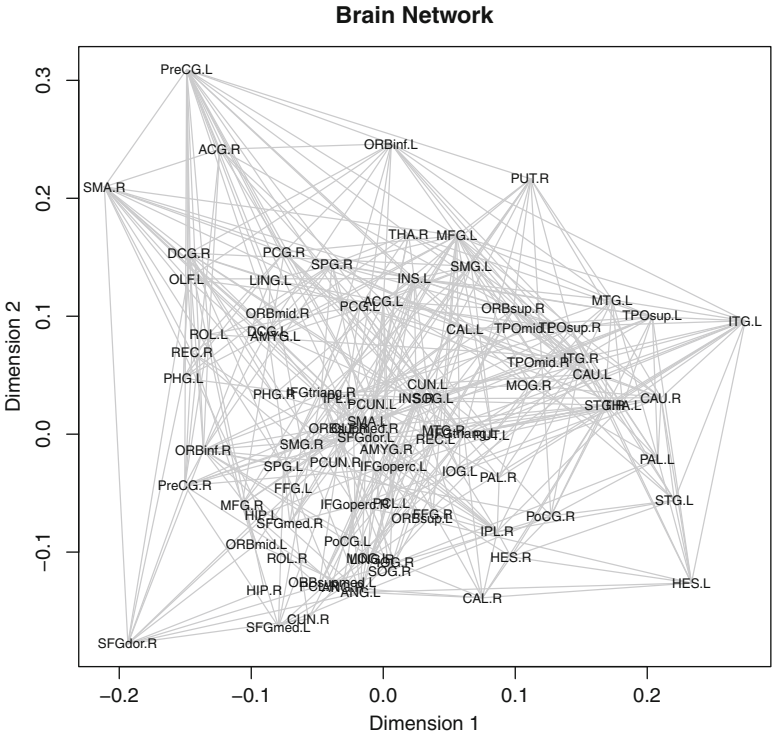

Fig. 14.12 Eigenmodel network for binary ROI correlation matrix

Another functional connectivity method is Granger causality, which is, however, subject to some controversy in fMRI (see Wen et al., 2013). Given two activation time series (e.g., related to two ROIs), the idea is to detect whether a particular activation in one time series such as a spike leads to a similar, potentially lagged pattern in the second ROI time series. Again, the **FIAR** package can be used to compute the Granger causality.

A further fMRI modeling approach is called *encoding*. Encoding models predict activity in single voxels that is evoked by different sensory, cognitive, or task conditions (Naselaris et al., 2011). An interesting application can be found in Huth et al. (2016).

The use of support vector machines (SVMs) within the context of *multivoxel pattern analysis* (MVPA) is described in De Martino et al. (2008). In R, the **kernlab** package (Karatzoglou et al., 2004) can be used for fitting SVMs. Other MVPA approaches including tools for running ROI and searchlight analysis are implemented in **rMVPA** (Buchsbaum, 2016).

References

Achard, S. (2012). **brainwaver**: Basic wavelet analysis of multivariate time series with a visualisation and parametrisation using graph theory. R package version 1.6. https://CRAN.R-project.org/package=brainwaver

Achard, S., Salvador, R., Whitcher, B., Suckling, J., & Bullmore, E. (2006). A resilient, low-frequency, small-world human brain functional network with highly connected association cortical hubs. *The Journal of Neuroscience, 26*, 63–72.

Adler, R. J. (1981). *The geometry of random fields*. New York: Wiley.

Ashby, F. G. (2011). *Statistical analysis of fMRI data*. Cambridge: The MIT Press.

Benjamini, Y., & Hochberg, Y. (1995). Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society, Series B, 57*, 289–300.

Bordier, C., Dojat, M., & de Micheaux, P. L. (2011). Temporal and spatial independent component analysis for fMRI data sets embedded in the **AnalyzeFMRI** R package. *Journal of Statistical Software, 44*(9), 1–24. http://www.jstatsoft.org/v44/i09/

Bretz, F., Hothorn, T., & Westfall, P. (2011). *Multiple comparisons using R*. Boca Raton: Chapman & Hall/CRC.

Buchsbaum, B. R. (2016). **rMVPA**: Multivoxel pattern analysis in R. R package version 0.1.1.

Bullmore, E., Fadili, J., Maxim, V., Xendur, L., Whitcher, B., Suckling, J., Brammer, M., & Breakspear, M. (2004) Wavelets and functional magnetic resonance imaging of the human brain. *NeuroImage, 23*, 234–249.

Calhoun, V. D., Liu, J., & Adali, T. (2009). A review of group ICA for fMRI data and ICA for joint inference of imaging, genetic, and ERP data. *NeuroImage, 45*, 163–172.

Clayden, J. (2016a). **mmand**: Mathematical morphology in any number of dimensions. R package version 1.4.1. https://CRAN.R-project.org/package=mmand

Clayden, J. (2016b). **RNiftyReg**: Image registration using the NiftyReg library. R package version 2.4.0. https://CRAN.R-project.org/package=RNiftyReg

da Silva, A. F. (2011). **cudaBayesreg**: Parallel implementation of a Bayesian multilevel model for fMRI data analysis. *Journal of Statistical Software, 44*(1), 1–24. https://www.jstatsoft.org/index.php/jss/article/view/v044i04

De Leeuw, J., & Mair, P. (2009). Multidimensional scaling using majorization: SMACOF in R. *Journal of Statistical Software, 31*(3), 1–30. http://www.jstatsoft.org/v31/i03/

De Martino, F., Valente, G., Staeren, N., Ashburner, J., Goebel, R., & Formisano, E. (2008). Combining multivariate voxel selection and support vector machines for mapping and classification of fMRI spatial patterns. *NeuroImage, 43*, 44–58.

Eklund, A., Nichols, T. E., & Knutsson, H. (2016). Cluster failure: Why fMRI inferences for spatial extent have inflated false-positive rates. *Proceedings of the National Academy of Sciences of the United States of America, 113*, 7900–7905.

Eloyan, A., Li, S., Muschelli, J., Pekar, J. J., Mostofsky, S. H., & Caffo, B. S. (2014). Analytic programming with fMRI data: A quick-start guide for statisticians using R. *PLoS ONE, 9*(2), e89470.

Fisher, A. (2016). **ggBrain**: ggplot brain images. R package version 0.1.

Friston, K. J. (1994). Functional and effective connectivity in neuroimaging: A synthesis. *Human Brain Mapping 2*, 56–78.

Genovese, C. R., Lazar, N. A., & Nichols, T. (2002). Thresholding of statistical maps in functional neuroimaging using the false discovery rate. *NeuroImage, 15*, 870–878.

Gentleman, R., Carey, V., Huber, W., & Hahne, F. (2016). **genefilter**: Methods for filtering genes from high-throughput experiments. R package version 1.54.2.

Hoff, P. (2012). **eigenmodel**: Semiparametric factor and regression models for symmetric relational data. R package version 1.01. https://CRAN.R-project.org/package=eigenmodel

Hothorn, T., Bretz, F., & Westfall, P. (2008). Simultaneous inference in general parametric models. *Biometrical Journal, 50*, 346–363.

Huth, A. G., de Heer, W. A., Griffiths, T. L., Theunissen, F. E., & Gallant, J. L. (2016). Natural speech reveals the semantic maps that tile human cerebral cortex. *Nature, 532*, 453–458.

Karatzoglou, A., Smola, A., Hornik, K., & Zeileis, A. (2004). **kernlab**: An S4 package for kernel methods in R. *Journal of Statistical Software, 11*(9), 1–20. http://www.jstatsoft.org/v11/i09/

Kriegeskorte, N., Mur, M., & Bandettini, P. (2008). Representational similarity analysis—Connecting the branches of systems neuroscience. *Frontiers in Systems Neuroscience, 2*(4), 1–28.

Lazar, N. A. (2008). *The statistical analysis of functional MRI data*. New York: Springer.

Madhyastha, T. (2017). **neuropointilist**: Flexible parallel modeling of neuroimaging data, point by point. R package version 0.0.0.9000. https://github.com/IBIC/neuropointillist

Muschelli, J., Sweeney, E., & Crainiceanu, C. (2014). **brainR**: Interactive 3 and 4D images of high resolution neuroimage data. *The R Journal, 6*(1), 41–48.

Naselaris, T., Kay, K. N., Nishimoto, S., & Gallant, J. L. (2011). Encoding and decoding in fMRI. *NeuroImage, 56*, 400–410.

Nichols, T. E. (2012). Multiple testing corrections, nonparametric methods, and random field theory. *NeuroImage, 15*, 811–815.

Nichols, T. E., & Holmes, A. P. (2001). Nonparametric permutation tests for functional neuroimaging: A primer with examples. *Human Brain Mapping, 15*, 1–25.

Pollard, K. S., Dudoit, S., & van der Laan, M. J. (2005). Multiple testing procedures: R **multtest** package and applications to genomics. In R. Gentleman, V. Carey, W. Huber, R. Irizarry, & S. Dudoit (Eds.) *Bioinformatics and computational biology solutions using R and bioconductor* (pp. 251–272). New York: Springer.

Polzehl, J., & Tabelow, K. (2007). **fmri**: A package for analyzing fmri data. *R News, 7*(2), 13–17.

Reiss, P. T., Huang, L., Chen, Y. H., Huo, L., Tarpey, T., & Mennes, M. (2014). Massively parallel nonparametric regression, with an application to developmental brain mapping. *Journal of Computational and Graphical Statistics, 23*, 232–248.

Reiss, P. T., Chen, Y. H., Huang, L., Huo, L., Tan, R., & Jiao, R. (2016). **vows**: Voxelwise Semiparametrics. R package version 0.5. https://CRAN.R-project.org/package=vows

Revelle, W. (2017). **psych**: Procedures for psychological, psychometric, and personality research. R package version 1.7.8. http://CRAN.R-project.org/package=psych

Roelstraete, B., & Rosseel, Y. (2011). **FIAR**: An R package for analyzing functional integration in the brain. *Journal of Statistical Software, 44*(1), 1–32. https://www.jstatsoft.org/index.php/jss/article/view/v044i13

Smith, S. M., Jenkinson, M., Woolrich, M. W., Beckmann, C. F., Behrens, T. E. J., Johansen-Berg, H., Bannister, P. R., De Luca, M., Drobnjak, I., Flitney, D. E., Niazy, R. K., Saunders, J., Vickers, J., Zhang, Y., De Stefano, N., Brady, J. M., & Matthews, P. M. (2004). Advances in functional and structural MR image analysis and implementation as FSL. *NeuroImage, 23*, 208–219.

Tabelow, K., & Polzehl, J. (2011). Statistical parametric maps for functional MRI experiments in R: The package **fmri**. *Journal of Statistical Software, 44*(11), 1–21. http://www.jstatsoft.org/v44/i11/

Tamir, D. I., Thornton, M. A., Contreras, J. M., & Mitchell, J. P. (2016a). Neural evidence that three dimensions organize mental state representation: Rationality, social impact, and valence. *Proceedings of the National Academy of Sciences of the United States of America, 113*, 194–199.

Tamir, D. I., Thornton, M. A., Contreras, J. M., & Mitchell JP (2016b). Neural evidence that three dimensions organize mental state representation: rationality, social impact, and valence. Harvard Dataverse, V3. https://doi.org/10.7910/DVN/ELLLZM

The FIL Methods Group. (2016). SPM12 manual. Functional Imaging Laboratory, Wellcome Trust Centre for Neuroimaging, Institute of Neurology, University College London, London. http://www.fil.ion.ucl.ac.uk/spm/

Tzourio-Mazoyer, N., Landeau, B., Papathanassiou, D., Crivello, F., Etard, O., Delcroix, N., Mazoyer, B., & Joliot, M. (2002). Automated anatomical labeling of activations in SPM using

a macroscopic anatomical parcellation of the MNI MRI single-subject brain. *NeuroImage, 15*, 273–289.

Venables, W. N., & Ripley, B. D. (2002). *Modern applied statistics with S* (4th ed.). New York: Springer.

Watson, C. G. (2016). **brainGraph**: Graph theory analysis of brain MRI data. R package version 0.62.0. https://CRAN.R-project.org/package=brainGraph

Wen, X., Rangarajan, G., & Ding, M. (2013). Is granger causality a viable technique for analyzing fMRI data? *PLoS ONE, 8*(7), 1–11.

Woo, C. W., Krishnan, A., & Wager, T. D. (2014). Cluster-extent based thresholding in fMRI analyses: Pitfalls and recommendations. *NeuroImage, 91*, 412–419.

Worsley, K. J., Marrett, S., Neelin, P., Vandal, A. C., Friston, K. J., & Evans, A. C. (1996). A unified statistical approach for determining significant signals in images of cerebral activation. *Human Brain Mapping, 4*, 58–73.

Index

P. Mair, *Modern Psychometrics with R*, Use R!,
https://doi.org/10.1007/978-3-319-93177-7