R syntax to Accompany *Best Practices in Exploratory Factor Analysis* (2014) by Jason Osborne

A. Alexander Beaujean
Baylor University

August 2014

Contents

1 Introduction to Exploratory Factor Analysis **1**

2 Extraction and Rotation **2**

2.1 Importing Data . . . 2

2.2 Example 1: Engineering Data . . . 2

2.3 Example 2: Self-Description Questionnaire Data . . . 8

2.4 Example 3: Geriatric Depression Scale Data . . . 12

2.4.1 Traditional EFA . . . 13

2.4.2 EFA with Tetrachoric Correlations . . . 17

2.4.3 Full-Information Item Factor Analysis . . . 20

3 Sample Size Matters **23**

3.1 Selecting a Random Sample from the Dataset . . . 23

3.2 Determining Sample Size for a Study Before Collecting Data . . . 24

4 Replication Statistics in EFA **25**

4.1 Import School Perceptions Questionnaire Data . . . 25

4.2 Splitting a Dataset in Half with Random Observations . . . 25

4.3 Comparing Factor Loadings . . . 26

5 Bootstrapping Applications in EFA **28**

5.1 Bootstrapping Initial Eigenvalues . . . 28

5.2 Bootstrapping Factor Loadings . . . 31

6 Data Cleaning and EFA **33**

7 Are Factor Scores a Good Idea? **34**

8 Higher Order Factors **35**

8.1 Importing and Subsetting the Data . . . 35

8.2 Higher Order EFA . . . 35

8.3 Alternatives for Higher Order Factors in Exploratory Factor Analysis . . . 38

9 After the EFA: Internal Consistency **42**

9.1 Coefficient Alpha . . . 42

9.2 Bootstrapped Confidence Intervals for Alpha . . . 46

9.3 Other Measures of Reliability . . . 47

References **49**

Note. Text in **red** is an internal hyperlink to a page, text in **orange** is an internal hyperlink to a reference, and text in **blue** is an external hyperlink.

Preface

This is designed to show how to conduct EFA in the **R** program (R Development Core Team, 2014) following Osborne's (2014) *Best Practices in Exploratory Factor Analysis* book.

Venables, Smith, and R Development Core Team (2014) provides a free introduction to **R**. In addition, the first chapter of my latent variable book (Beaujean, 2014) is devoted to introducing how to use **R**.

1 | Introduction to Exploratory Factor Analysis

There are no examples for this chapter.

2 | Extraction and Rotation

Chapter Contents

2.1 Importing Data 2
2.2 Example 1: Engineering Data 2
2.3 Example 2: Self-Description Questionnaire Data 8
2.4 Example 3: Geriatric Depression Scale Data 12
2.4.1 Traditional EFA 13
2.4.2 EFA with Tetrachoric Correlations 17
2.4.3 Full-Information Item Factor Analysis 20

2.1 Importing Data

There are three datasets used in this chapter. So, first we need to import them. As they are SPSS files, I use the *foreign* package along with the `read.spss()` function.

```
# load foreign package
library(foreign)

# engineering data
eng.data <- read.spss("CH02_ex1_engdata.sav", to.data.frame = TRUE)

# Marsh Self-Description Questionnaire (SDQ)
sdq.data <- read.spss("Ch02_ex2_MarshSPQNELS_marsh.SAV", to.data.frame = TRUE,
use.value.labels = FALSE)

# Geriatric Depression Scale data
gds.data <- read.spss("Ch02_GDS_ex3.sav", to.data.frame = TRUE,use.value.labels = FALSE)
```

2.2 Example 1: Engineering Data

For all the EFA, I use the *psych* packages's (Revelle, 2014) `fa()` function. There are four main argument for the `fa()` function. The fist is the name of the data, which can either be dataset or a correlation matrix. The second argument, `nfactors`, indicates the number of factors to extract. If not specified, it will select one factor. The third argument is `rotate`, which indicates the way to rotate the extracted factors. If not specified, it will use direct oblimin rotation. The fourth argument, `fm`, indicates the method to extract the factor. If not specified, it uses ordinary least squares (i.e, minimum residual) extraction. This is the same as unweighted least squares estimation.

In what follows, I extract two factors from the engineering data using maximum likelihood (ML), principal axis (PAF), unweighted/ordinary least squares (ULS/OLS), and generalized least squares (GLS) extraction. There is currently not an **R** package that will conduct alpha factor extraction. In all these initial extractions, I do not rotate the factors.

Table 2.1 Communality Values for Engineering Data

	Initial	ML	PAF	ULS/OLS	GLS
EngProbSolv1	0.74	0.71	0.73	0.71	0.72
EngProbSolv2	0.69	0.66	0.67	0.67	0.67
EngProbSolv3	0.75	0.77	0.77	0.77	0.77
EngProbSolv4	0.79	0.81	0.81	0.81	0.81
EngProbSolv5	0.79	0.81	0.80	0.81	0.80
EngProbSolv6	0.77	0.77	0.77	0.77	0.77
EngProbSolv7	0.79	0.78	0.78	0.78	0.78
EngProbSolv8	0.67	0.67	0.67	0.67	0.67
INTERESTeng1	0.67	0.67	0.67	0.67	0.67
INTERESTeng2	0.80	0.83	0.83	0.83	0.84
INTERESTeng3	0.82	0.85	0.84	0.84	0.85
INTERESTeng4	0.81	0.83	0.82	0.83	0.82
INTERESTeng5	0.78	0.78	0.80	0.78	0.79
INTERESTeng6	0.74	0.75	0.75	0.75	0.75

```
library(psych)

# ml
eng.ml <- fa(eng.data, nfactors = 2, rotate = "none", fm = "ml")

# paf
eng.paf <- fa(eng.data, nfactors = 2, rotate = "none", fm = "pa")

# uls (ols)
eng.ols <- fa(eng.data, nfactors = 2, rotate = "none", fm = "minres")

# gls
eng.gls <- fa(eng.data, nfactors = 2, rotate = "none", fm = "gls")
```

Next, I extract the communality values, which I show in Table 2.1. The initial communality estimates are just the squared multiple correlation of a variable with all the remaining variables. It can be found using the `smc()` function.

```
# communalities initial
smc(eng.data)
# ml
eng.ml$communality
# paf
eng.paf$communality
# uls (ols)
eng.ols$communality
# gls
eng.gls$communality
```

The `cor()` function returns a correlation matrix, while the `eigen()` function returns the eigenvalues from a correlation matrix. I show the eigenvalues in Table 2.1.

Table 2.2 Eigenvalues for Engineering Data

	Initial	ML	PAF	ULS/OLS	GLS
1	7.65	7.42	7.42	7.42	7.42
2	3.50	3.28	3.28	3.28	3.28
3	0.46	0.19	0.19	0.19	0.19
4	0.36	0.08	0.08	0.08	0.08
5	0.31	0.06	0.06	0.06	0.06
6	0.28	0.04	0.04	0.04	0.04
7	0.27	0.03	0.03	0.03	0.03
8	0.21	-0.00	-0.00	-0.00	-0.00
9	0.21	-0.02	-0.02	-0.02	-0.02
10	0.18	-0.03	-0.03	-0.03	-0.03
11	0.16	-0.05	-0.06	-0.05	-0.05
12	0.14	-0.08	-0.08	-0.08	-0.08
13	0.13	-0.09	-0.09	-0.09	-0.09
14	0.13	-0.13	-0.12	-0.13	-0.12

```
eng.cor <- cor(eng.data)
# eigenvalues initial
eigen(eng.cor)$values
# ml
eng.ml$values
# paf
eng.paf$values
# ols
eng.ols$values
# gls
eng.gls$values
```

The `scree()` function returns a scree plot. The `factors=TRUE` argument produces eigenvalues from a factor analysis, while the `pc=TRUE` argument produces eigenvalues from a principal component analysis.[1] The scree plot using the eigenvalues from principal components extraction is shown in Figure 2.1.

```
# scree plot
scree(eng.data, factors = FALSE, pc = TRUE, hline = -1)
```

A parallel analysis is done using the `fa.parallel()` function. The resulting plot is shown in Figure 2.2.

```
# parallel analysis
fa.parallel(eng.data, fa = "fa", show.legend = FALSE, fm = "ml")
```

MAP values are produced as part of the `VSS()` function.[2] Appending the `$map` operator to the function returns only the MAP values.

[1] The `hline=-1` argument removes the default horizontal line at $y = 1$.

[2] VSS stands for very simple structure.

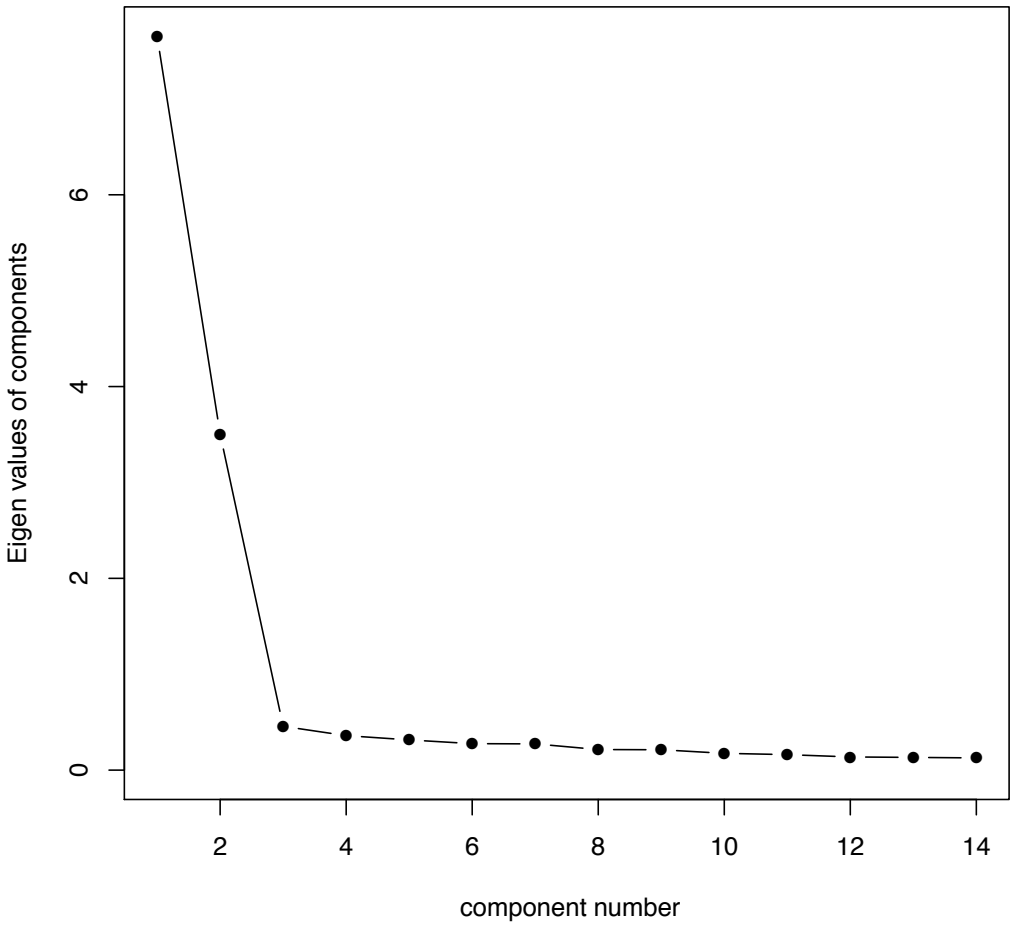


Figure 2.1 Scree plot for engineering data.

```
# MAP values
VSS(eng.data, n = 7, plot = FALSE, fm = "ml")$map


## [1] 0.246 0.024 0.029 0.044 0.058 0.072 0.093
```

There is not a function to plot the factor loadings, but creating one is relatively simple using the `plot()` function. In Figure 2.3 I give a factor loadings plot. I narrowed the X and Y axes so as to make the labels more readable.

```
# plot of factor loadings
plot(eng.ml$loadings[,1],eng.ml$loadings[,2],xlim=c(0,1),ylim=c(-.6,.6),
xlab="Factor 1",ylab="Factor 2")

# label the points
text(eng.ml$loadings[,1], eng.ml$loadings[,2], labels=rownames(eng.ml$loadings), cex= 0.4)

# horizonal and vertical line at 0
abline(h=0); abline(v=0)
```

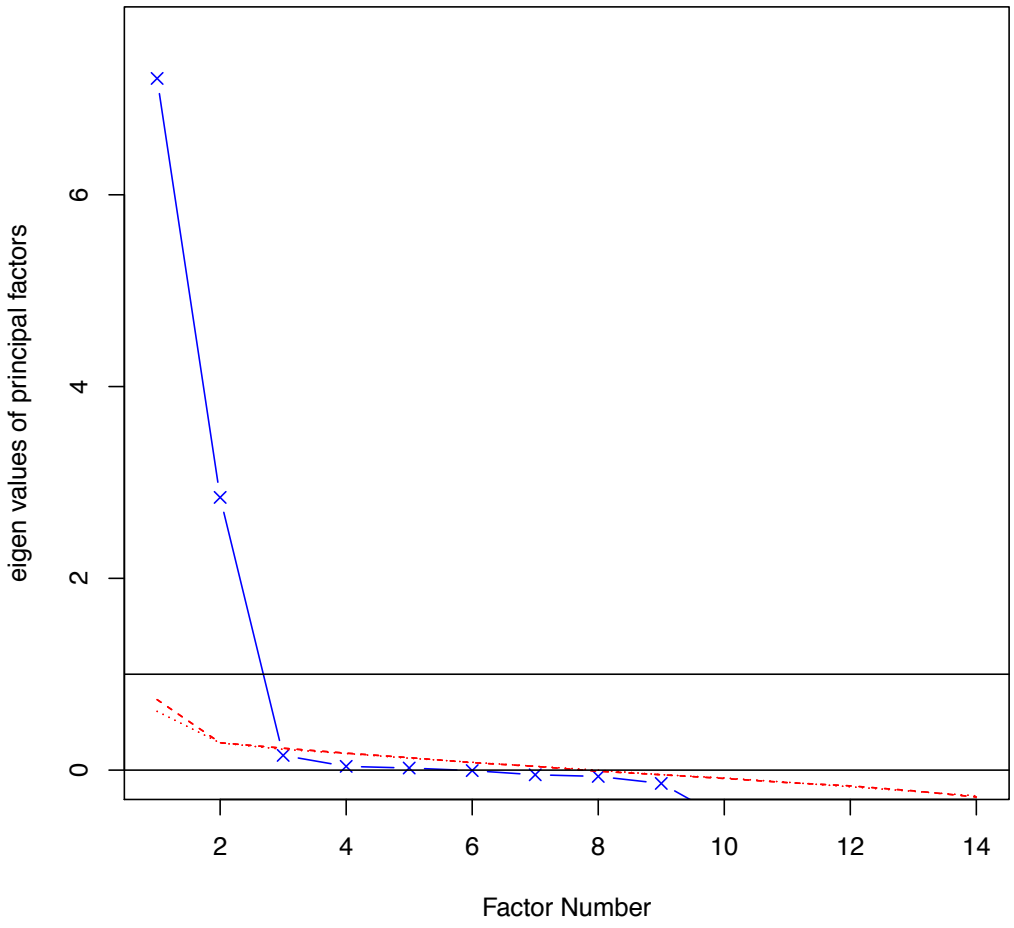


Figure 2.2 Plot of parallel analysis results for engineering data.

There are a large variety of factor rotations available in **R**. Below I show the ones used in Osborne (2014), but elsewhere give a wider variety of them (Beaujean, 2013).[3] To show the factor loadings, append the `$loadings` argument to the object retuned from the `fa()` function. By default, this will not print loadings that are low in value. To show all the values, wrap the loadings object in the `print()` function using the `cut=0` argument.

```
# unrotated loadings
print(eng.paf$loadings, cut = 0)


##
## Loadings:
##               PA1   PA2
## EngProbSolv1  0.76 -0.39
## EngProbSolv2  0.70 -0.42
## EngProbSolv3  0.78 -0.39
## EngProbSolv4  0.80 -0.42
## EngProbSolv5  0.81 -0.38
## EngProbSolv6  0.80 -0.37
```

[3] Browne (2001) provides an good introduction to the variety of factor rotations available.

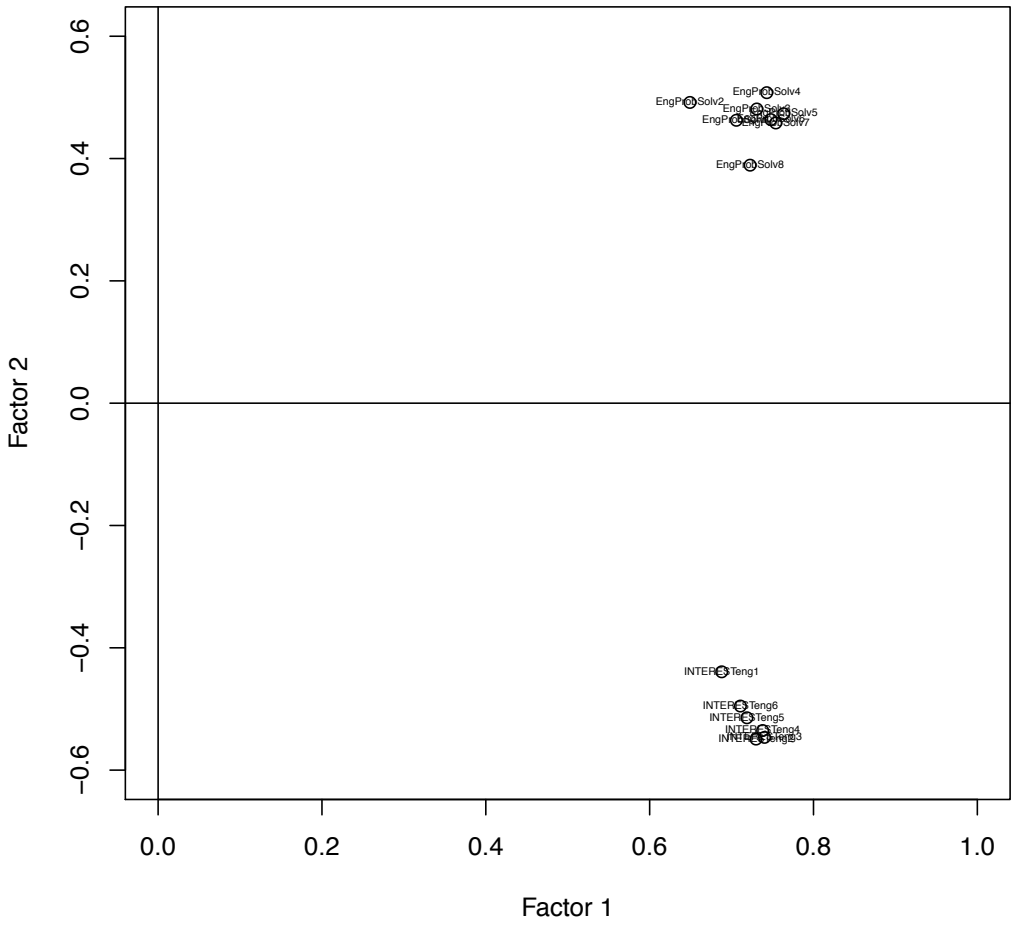


Figure 2.3 Factor loadings plot for engineering data.

```
## EngProbSolv7  0.80 -0.36
## EngProbSolv8  0.76 -0.30
## INTERESTeng1  0.63  0.52
## INTERESTeng2  0.66  0.63
## INTERESTeng3  0.67  0.63
## INTERESTeng4  0.67  0.61
## INTERESTeng5  0.66  0.61
## INTERESTeng6  0.65  0.58
##
##                  PA1  PA2
## SS loadings    7.42 3.28
## Proportion Var 0.53 0.23
## Cumulative Var 0.53 0.76
```

```
# oblimin loadings
eng.paf.oblimin <- fa(eng.data, nfactors = 2, rotate = "oblimin", fm = "pa")
print(eng.paf.oblimin$loadings, cut = 0)


##
## Loadings:
##              PA1    PA2
```

```
## EngProbSolv1  0.859 -0.017
## EngProbSolv2  0.841 -0.072
## EngProbSolv3  0.880 -0.008
## EngProbSolv4  0.909 -0.025
## EngProbSolv5  0.886  0.020
## EngProbSolv6  0.869  0.019
## EngProbSolv7  0.868  0.032
## EngProbSolv8  0.791  0.071
## INTERESTeng1  0.045  0.800
## INTERESTeng2 -0.020  0.920
## INTERESTeng3 -0.011  0.921
## INTERESTeng4  0.002  0.903
## INTERESTeng5 -0.004  0.896
## INTERESTeng6  0.011  0.863
##
##                  PA1  PA2
## SS loadings    5.97 4.71
## Proportion Var 0.43 0.34
## Cumulative Var 0.43 0.76
```

```
# variamx loadings
eng.paf.varimax <- fa(eng.data, nfactors = 2, rotate = "variamx", fm = "pa")
print(eng.paf.varimax$loadings, cut = 0)
```

```
##
## Loadings:
##               PA1   PA2
## EngProbSolv1  0.76 -0.39
## EngProbSolv2  0.70 -0.42
## EngProbSolv3  0.78 -0.39
## EngProbSolv4  0.80 -0.42
## EngProbSolv5  0.81 -0.38
## EngProbSolv6  0.80 -0.37
## EngProbSolv7  0.80 -0.36
## EngProbSolv8  0.76 -0.30
## INTERESTeng1  0.63  0.52
## INTERESTeng2  0.66  0.63
## INTERESTeng3  0.67  0.63
## INTERESTeng4  0.67  0.61
## INTERESTeng5  0.66  0.61
## INTERESTeng6  0.65  0.58
##
##                  PA1  PA2
## SS loadings    7.42 3.28
## Proportion Var 0.53 0.23
## Cumulative Var 0.53 0.76
```

2.3 Example 2: Self-Description Questionnaire Data

The syntax for this example largely follows that from Section 2.2, so I present it below with little explanation.

```r
# factor extraction

# ml
sdq.ml <- fa(sdq.data, nfactors = 3, rotate = "none", fm = "ml")
# paf
sdq.paf <- fa(sdq.data, nfactors = 3, rotate = "none", fm = "pa")
# uls (ols)
sdq.ols <- fa(sdq.data, nfactors = 3, rotate = "none", fm = "minres")
# gls
sdq.gls <- fa(sdq.data, nfactors = 3, rotate = "none", fm = "gls")


# communalities
smc(sdq.data)
```

```
##  Eng1  Eng2  Eng3  Eng4 Math1 Math2 Math3 Math4  Par1  Par2  Par3  Par4  Par5
##  0.54  0.58  0.61  0.45  0.70  0.67  0.70  0.39  0.45  0.41  0.57  0.41  0.48
```

```r
sdq.ml$communality
```

```
##  Eng1  Eng2  Eng3  Eng4 Math1 Math2 Math3 Math4  Par1  Par2  Par3  Par4  Par5
##  0.62  0.68  0.72  0.40  0.79  0.75  0.78  0.37  0.53  0.43  0.69  0.39  0.56
```

```r
sdq.paf$communality
```

```
##  Eng1  Eng2  Eng3  Eng4 Math1 Math2 Math3 Math4  Par1  Par2  Par3  Par4  Par5
##  0.62  0.66  0.72  0.41  0.79  0.74  0.80  0.37  0.51  0.45  0.68  0.42  0.54
```

```r
sdq.ols$communality
```

```
##  Eng1  Eng2  Eng3  Eng4 Math1 Math2 Math3 Math4  Par1  Par2  Par3  Par4  Par5
##  0.62  0.67  0.72  0.41  0.79  0.75  0.78  0.37  0.52  0.44  0.69  0.40  0.55
```

```r
sdq.gls$communality
```

```
##  Eng1  Eng2  Eng3  Eng4 Math1 Math2 Math3 Math4  Par1  Par2  Par3  Par4  Par5
##  0.62  0.68  0.74  0.41  0.80  0.75  0.80  0.37  0.52  0.44  0.69  0.41  0.55
```

```r
# eigenvalues

# initial
eigen(cor(sdq.data))$values
```

```
##  [1] 4.08 2.55 2.21 0.91 0.52 0.49 0.46 0.39 0.35 0.32 0.30 0.23 0.19
```

```r
# ml
sdq.ml$values
```

```
##  [1]  3.6898  2.2250  1.8080  0.3745  0.0619  0.0301  0.0026 -0.0094 -0.0341 -0.0572
## [11] -0.0865 -0.1126 -0.1739
```

```
# paf
sdq.paf$values
```

```
##  [1]  3.68914  2.22597  1.80421  0.37940  0.06227  0.02648  0.00031 -0.01710 -0.04077
## [10] -0.06105 -0.08563 -0.10250 -0.16206
```

```
# ols
sdq.ols$values
```

```
##  [1]  3.6890  2.2244  1.8067  0.3752  0.0609  0.0285  0.0017 -0.0115 -0.0357 -0.0586
## [11] -0.0865 -0.1112 -0.1718
```

```
# gls
sdq.gls$values
```

```
##  [1]  3.6932  2.2300  1.8102  0.3784  0.0680  0.0314  0.0058 -0.0115 -0.0338 -0.0547
## [11] -0.0847 -0.1043 -0.1664
```

```
# scree plot
scree(sdq.data, factors = FALSE, pc = TRUE, hline = -1)
```

```
# parallel analysis
fa.parallel(sdq.data, fa = "fa", show.legend = FALSE, fm = "ml")
```

```
# MAP values
VSS(sdq.data, n = 6, plot = FALSE, fm = "ml")$map
```

```
## [1] 0.099 0.085 0.035 0.040 0.061 0.088
```

```
# unrotated loadings
print(sdq.ml$loadings, cut = 0)
```

```
##
## Loadings:
##       ML1    ML2    ML3
## Eng1   0.348  0.634  0.309
## Eng2   0.310  0.636  0.419
## Eng3   0.406  0.644  0.378
## Eng4  -0.257 -0.552 -0.179
## Math1  0.802 -0.373  0.088
## Math2  0.810 -0.298  0.070
## Math3  0.828 -0.301  0.083
## Math4 -0.572  0.212 -0.004
## Par1   0.360  0.349 -0.524
## Par2  -0.252 -0.293  0.533
## Par3   0.426  0.370 -0.613
## Par4  -0.359 -0.335  0.388
```

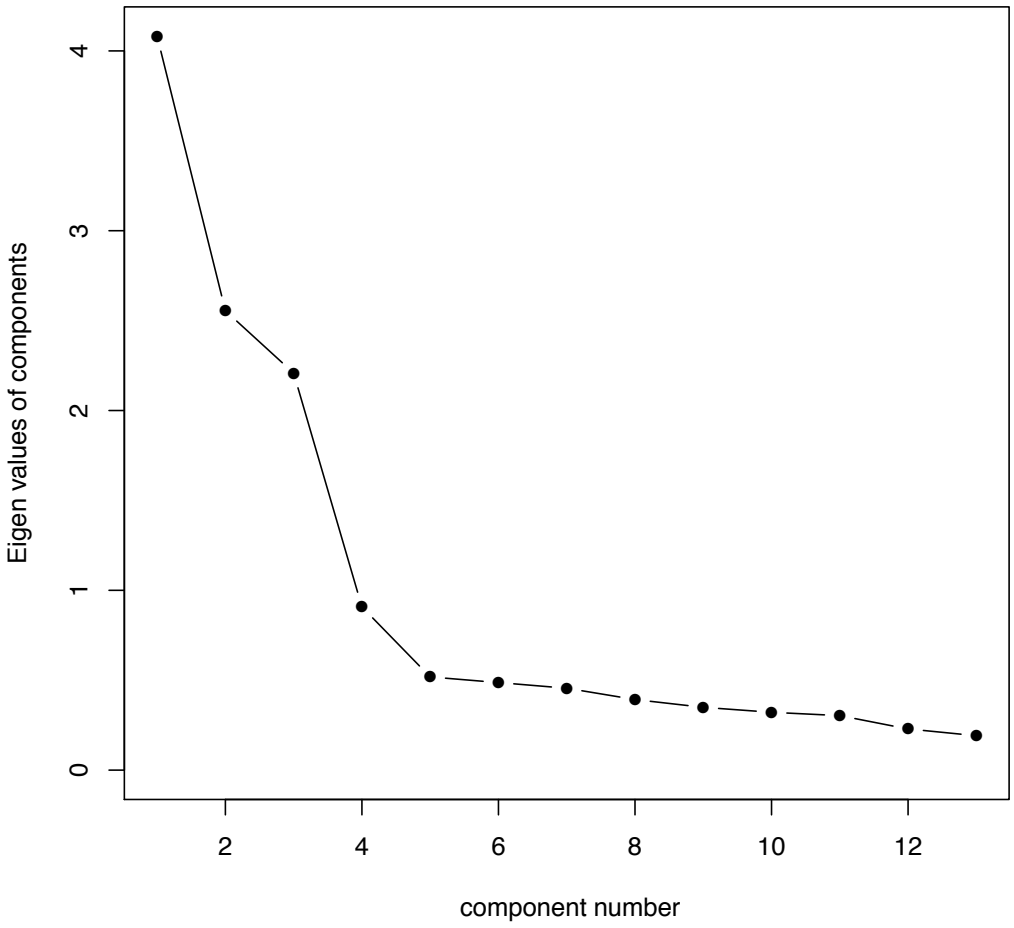


Figure 2.4 Scree plot for SDQ data.

```
## Par5   0.367  0.315 -0.568
##
##                 ML1  ML2  ML3
## SS loadings    3.40 2.45 1.87
## Proportion Var 0.26 0.19 0.14
## Cumulative Var 0.26 0.45 0.59
```

```
# promax loadings
sdq.ml.promax <- fa(sdq.data, nfactors = 3, rotate = "promax", fm = "ml")
print(sdq.ml.promax$loadings, cut = 0)
```

```
##
## Loadings:
##       ML1    ML3    ML2
## Eng1  -0.002  0.025  0.779
## Eng2  -0.011 -0.090  0.845
## Eng3   0.056 -0.024  0.847
## Eng4   0.059 -0.097 -0.606
## Math1  0.898 -0.033 -0.024
## Math2  0.859  0.014  0.024
## Math3  0.878  0.005  0.035
```

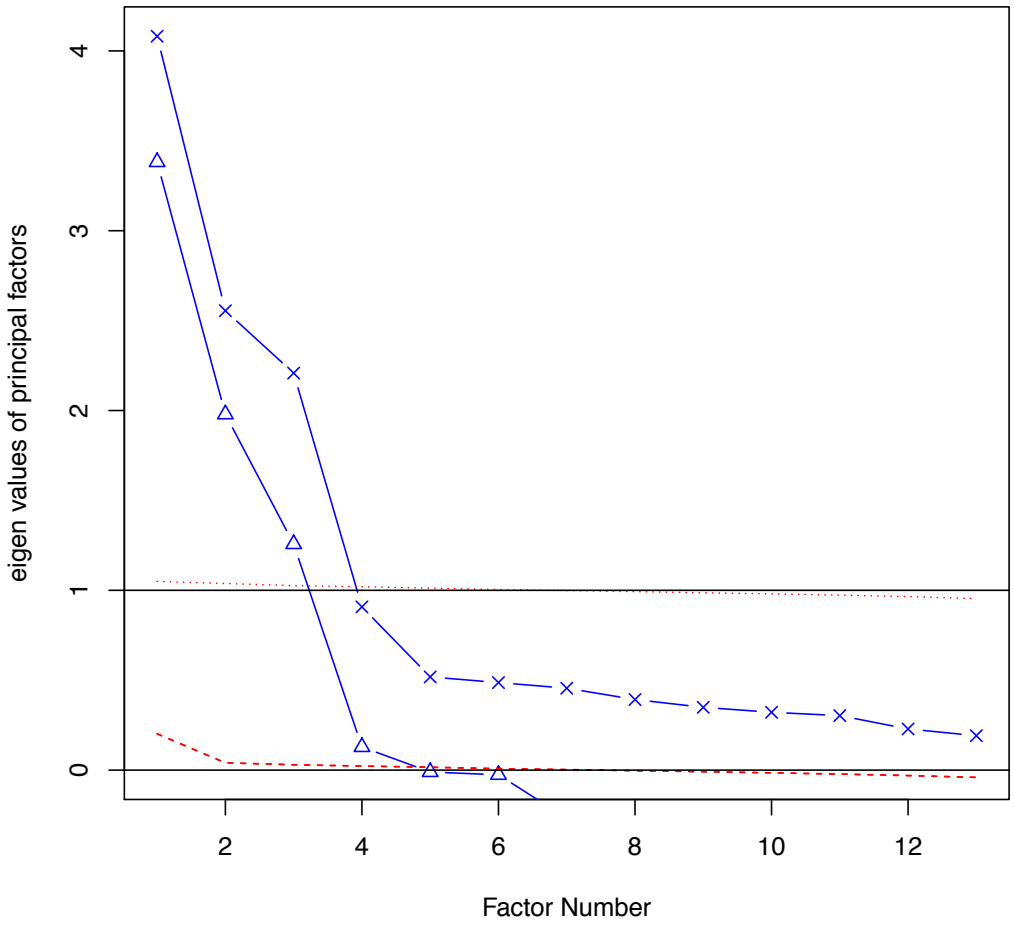


Figure 2.5 Plot of parallel analysis results for SDQ data.

```
## Math4 -0.598 -0.054  0.014
## Par1  -0.008  0.725  0.007
## Par2   0.069 -0.687  0.069
## Par3   0.016  0.835 -0.020
## Par4  -0.028 -0.588 -0.087
## Par5   0.007  0.757 -0.047
##
##                  ML1  ML3  ML2
## SS loadings     2.69 2.64 2.42
## Proportion Var 0.21 0.20 0.19
## Cumulative Var 0.21 0.41 0.60
```

2.4 Example 3: Geriatric Depression Scale Data

The *Geriatric Depression Scale* (GDS) data is binary. Such data are often better factor analyzed using tetrachoric correlations of the variables (Holgado-Tello, Chacón-Moscoso, Barbero-García, & Vila-Abad, 2010) or using full-information item factor analysis (Bock, Gibbons, & Muraki, 1988). Wirth and Edwards (2007) give a nice introduction to the issues involved in such models. In what follows, I show **R** syntax for traditional EFA, EFA with tetrachoric correlations, and full-information factor analysis.

2.4.1 Traditional EFA

```
# factor extraction

# ml
gds.ml <- fa(gds.data, nfactors = 8, rotate = "none", fm = "ml")
# paf
gds.paf <- fa(gds.data, nfactors = 8, rotate = "none", fm = "pa")
# uls/ols
gds.ols <- fa(gds.data, nfactors = 8, rotate = "none", fm = "minres")
# gls
gds.gls <- fa(gds.data, nfactors = 8, rotate = "none", fm = "gls")

# communalities
smc(gds.data)
gds.ml$communality
gds.paf$communality
gds.ols$communality
gds.gls$communality


# eigenvalues initial
eigen(cor(gds.data, use = "complete.obs"))$values
# ml
gds.ml$values
# paf
gds.paf$values
# uls/ols
gds.ols$values
# gls
gds.gls$values
```

```
# scree plot using traditional EFA
scree(gds.data, factors = FALSE, pc = TRUE, hline = -1)
```

```
# parallel analysis using traditional EFA
fa.parallel(gds.data, fa = "fa", show.legend = FALSE, fm = "pa")
```

```
## Parallel analysis suggests that the number of factors =  7  and the number of components =  4
```

```
# MAP values using traditional EFA
VSS(gds.data, n = 7, plot = FALSE, fm = "pa")$map


## [1] 0.0106 0.0094 0.0092 0.0100 0.0114 0.0128 0.0144
```

```
# factor loadings using traditional EFA

# single factor
gds.paf.one <- fa(gds.data, nfactors = 1, fm = "pa")
print(gds.paf.one$loadings, cut = 0)
```

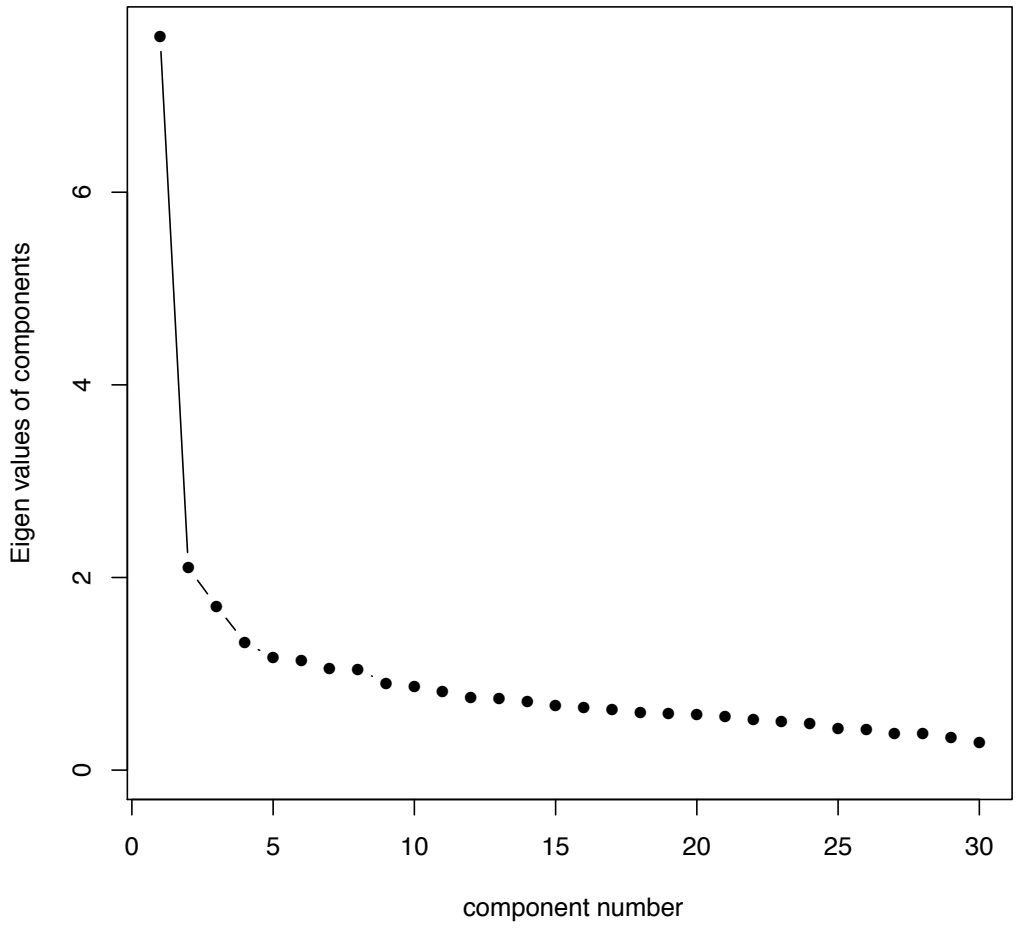


Figure 2.6 Scree plot for GDS data using traditional EFA methods.

```
##
## Loadings:
##       PA1
## GDS01 0.59
## GDS02 0.41
## GDS03 0.61
## GDS04 0.51
## GDS05 0.52
## GDS06 0.50
## GDS07 0.48
## GDS08 0.40
## GDS09 0.59
## GDS10 0.59
## GDS11 0.40
## GDS12 0.35
## GDS13 0.42
## GDS14 0.23
## GDS15 0.47
## GDS16 0.68
## GDS17 0.66
## GDS18 0.39
## GDS19 0.55
```

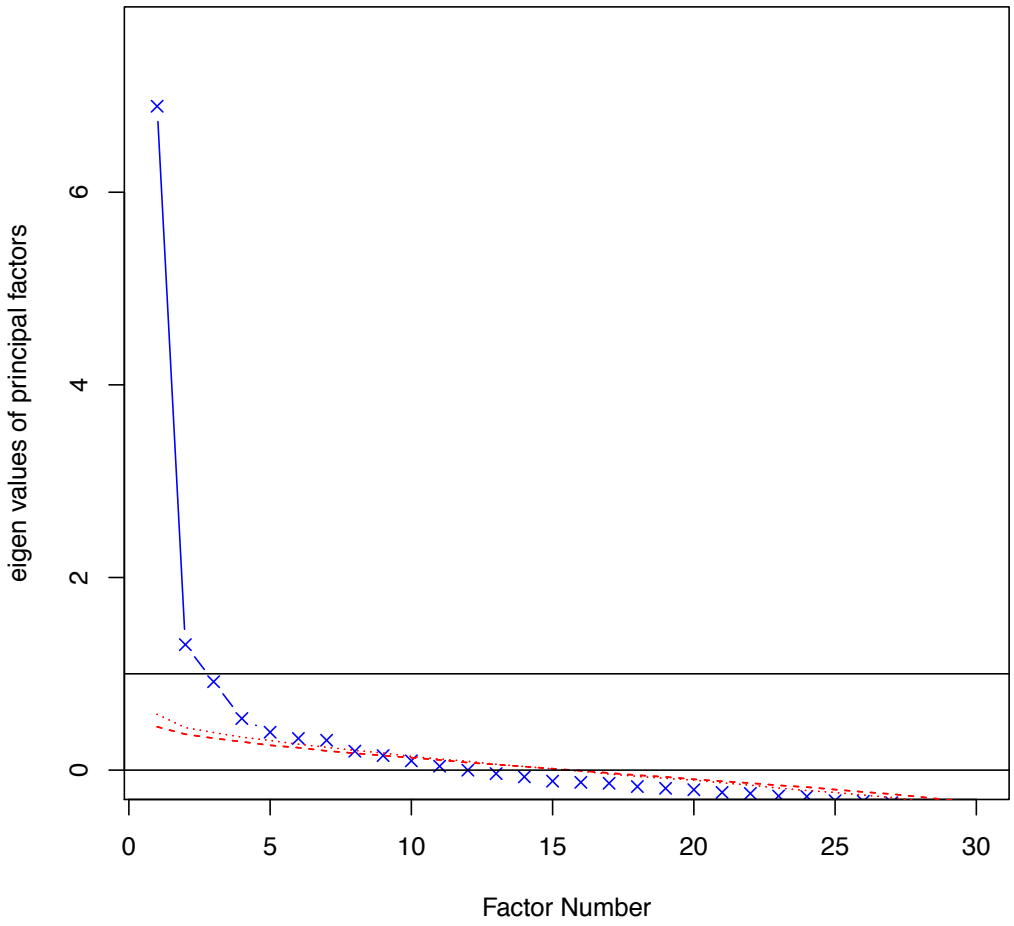


Figure 2.7 Plot of parallel analysis results for GDS data.

```
## GDS20 0.40
## GDS21 0.46
## GDS22 0.57
## GDS23 0.44
## GDS24 0.40
## GDS25 0.57
## GDS26 0.45
## GDS27 0.36
## GDS28 0.40
## GDS29 0.30
## GDS30 0.33
##
##                  PA1
## SS loadings    6.89
## Proportion Var 0.23



# five factor
gds.paf.five <- fa(gds.data, nfactors = 5, fm = "pa", rotate = "oblimin")
print(gds.paf.five$loadings, cut = 0)
```

```
##
## Loadings:
##       PA4    PA5    PA1    PA3    PA2
## GDS01  0.407 -0.013  0.333  0.070  0.025
## GDS02 -0.087 -0.028  0.260  0.375  0.132
## GDS03  0.210  0.195  0.443  0.055 -0.114
## GDS04  0.004  0.587  0.002  0.182 -0.103
## GDS05  0.404 -0.052  0.184  0.241 -0.036
## GDS06  0.048  0.376  0.074 -0.039  0.287
## GDS07  0.696  0.061  0.039 -0.127  0.002
## GDS08  0.244  0.194  0.104 -0.086  0.107
## GDS09  0.755  0.079 -0.007  0.026  0.001
## GDS10  0.055  0.175  0.346  0.108  0.173
## GDS11 -0.035  0.538 -0.035  0.009  0.083
## GDS12 -0.098  0.102  0.074  0.471 -0.002
## GDS13  0.188  0.365 -0.005 -0.071  0.097
## GDS14 -0.087 -0.012  0.068 -0.076  0.607
## GDS15  0.624 -0.104  0.041  0.119  0.011
## GDS16  0.160  0.450  0.252  0.085 -0.040
## GDS17  0.047  0.146  0.436  0.248  0.061
## GDS18  0.037  0.281  0.183 -0.096  0.134
## GDS19  0.233  0.119 -0.025  0.489  0.025
## GDS20 -0.155  0.060  0.134  0.401  0.253
## GDS21  0.163  0.031 -0.115  0.558  0.148
## GDS22  0.114 -0.033  0.660 -0.022  0.075
## GDS23  0.007  0.244  0.339 -0.064  0.070
## GDS24  0.054  0.470 -0.083 -0.019  0.155
## GDS25  0.037  0.352  0.358 -0.062  0.083
## GDS26  0.027  0.041  0.103  0.084  0.553
## GDS27  0.265 -0.073  0.070  0.245  0.051
## GDS28 -0.094  0.104  0.234  0.400 -0.064
## GDS29  0.061  0.081 -0.033  0.187  0.206
## GDS30  0.102  0.016 -0.155  0.162  0.530
##
##                  PA4  PA5   PA1   PA3   PA2
## SS loadings    2.189 1.80 1.660 1.610 1.338
## Proportion Var 0.073 0.06 0.055 0.054 0.045
## Cumulative Var 0.073 0.13 0.188 0.242 0.287
```

```
# eight factor
gds.paf.eight <- fa(gds.data, nfactors = 8, fm = "pa", rotate = "oblimin")
print(gds.paf.eight$loadings, cut = 0)
```

```
##
## Loadings:
##       PA8    PA1    PA2    PA3    PA6    PA4    PA5    PA7
## GDS01  0.168 -0.008  0.042  0.010  0.054 -0.045  0.090  0.688
## GDS02 -0.179  0.073  0.128  0.290  0.114  0.086 -0.076  0.372
## GDS03  0.065  0.242 -0.080 -0.075  0.183  0.059  0.092  0.443
## GDS04  0.009 -0.034 -0.035  0.013  0.745  0.038  0.004  0.097
## GDS05  0.413  0.168 -0.057  0.215  0.013  0.043 -0.071  0.084
## GDS06 -0.030  0.117  0.200  0.008  0.092  0.068  0.416 -0.042
## GDS07  0.694  0.010  0.051 -0.152  0.027 -0.012  0.031  0.060
```

```
## GDS08  0.025 -0.028  0.000 -0.060 -0.070  0.059  0.520  0.191
## GDS09  0.738  0.004  0.012  0.026  0.070 -0.007  0.024  0.047
## GDS10 -0.015  0.313  0.106  0.120  0.025  0.067  0.205  0.137
## GDS11  0.021  0.057  0.145 -0.034  0.498 -0.035  0.072 -0.070
## GDS12 -0.012 -0.030  0.008 -0.016 -0.013  0.877  0.011 -0.040
## GDS13 -0.010  0.008 -0.065  0.062  0.046 -0.024  0.607  0.002
## GDS14 -0.020  0.006  0.707 -0.098 -0.029  0.033 -0.031  0.021
## GDS15  0.627  0.048 -0.018  0.126 -0.094  0.038 -0.015  0.014
## GDS16  0.210  0.382 -0.041  0.010  0.244  0.114  0.157 -0.072
## GDS17  0.049  0.470  0.021  0.264  0.117  0.011 -0.010  0.124
## GDS18 -0.047  0.215  0.064 -0.027  0.039 -0.002  0.321  0.021
## GDS19  0.247  0.001 -0.009  0.367  0.129  0.184  0.018  0.028
## GDS20 -0.135  0.202  0.170  0.454  0.067  0.045 -0.009  0.001
## GDS21  0.113 -0.142  0.050  0.479  0.034  0.201  0.119  0.065
## GDS22  0.084  0.573  0.045 -0.038 -0.152  0.078  0.066  0.201
## GDS23  0.041  0.394  0.077 -0.065  0.110  0.015  0.074  0.002
## GDS24  0.086  0.110  0.131  0.044  0.271 -0.032  0.195 -0.176
## GDS25  0.098  0.536  0.068 -0.023  0.147 -0.007  0.104 -0.103
## GDS26  0.065  0.078  0.536  0.127  0.057  0.007  0.003  0.030
## GDS27  0.206  0.093 -0.070  0.328 -0.125  0.031  0.111  0.024
## GDS28  0.003  0.159  0.004  0.015  0.078  0.479 -0.081  0.079
## GDS29  0.031  0.041  0.119  0.324  0.064 -0.082  0.074 -0.019
## GDS30  0.085 -0.132  0.441  0.253 -0.003  0.003  0.114 -0.038
##
##                  PA8   PA1   PA2   PA3   PA6   PA4   PA5   PA7
## SS loadings    1.877 1.529 1.177 1.150 1.140 1.137 1.127 1.011
## Proportion Var 0.063 0.051 0.039 0.038 0.038 0.038 0.038 0.034
## Cumulative Var 0.063 0.114 0.153 0.191 0.229 0.267 0.305 0.338
```

2.4.2 EFA with Tetrachoric Correlations

The `tetrachoric()` function estimates tetrachoric correlations as well as thresholds. To examine/use only the correlations, append `$rho` to the **R** object.

```
# tetrachoric correlations
gds.tet <- tetrachoric(gds.data)
# examine tetrachoric correlation values
gds.tet$rho
```

To conduct the EFA with the, either use the `irt.fa()` function with the raw data or the `fa()` function with tetrachoric correlations. They produce equivalent results.

```
# EFA using tetrachoric correlations
gds.ifa <- irt.fa(gds.data, nfactors = 8, plot = FALSE, fm = "minres")
gds.ifa <- fa(gds.tet$rho, nfactors = 8, rotate = "none", fm = "minres")
```

```
# communalities
gds.ifa$communality
```

```
## GDS01 GDS02 GDS03 GDS04 GDS05 GDS06 GDS07 GDS08 GDS09 GDS10 GDS11 GDS12 GDS13 GDS14 GDS15
##  0.85  0.83  0.86  0.82  0.76  0.70  0.87  0.83  0.89  0.66  0.74  0.89  0.73  0.85  0.85
## GDS16 GDS17 GDS18 GDS19 GDS20 GDS21 GDS22 GDS23 GDS24 GDS25 GDS26 GDS27 GDS28 GDS29 GDS30
##  0.84  0.81  0.67  0.74  0.78  0.79  0.88  0.72  0.71  0.81  0.72  0.67  0.75  0.72  0.83
```

```
# eigenvalues

# initial using tetrachoric correlations
eigen(gds.tet$rho)$values


##  [1] 1.3e+01 2.5e+00 2.0e+00 1.4e+00 1.2e+00 1.1e+00 1.0e+00 9.7e-01 7.3e-01 7.1e-01
## [11] 6.1e-01 5.5e-01 5.2e-01 4.5e-01 4.0e-01 3.8e-01 3.5e-01 3.2e-01 3.0e-01 2.7e-01
## [21] 2.2e-01 1.8e-01 1.5e-01 1.0e-01 7.5e-02 2.8e-02 6.2e-03 2.2e-14 2.2e-14 2.1e-14



# EFA using tetrachoric correlations
gds.ifa$values


##  [1] 13.226  2.323  1.781  1.256  0.979  0.863  0.775  0.769  0.496  0.464  0.348  0.302
## [13]  0.273  0.216  0.185  0.174  0.120  0.100  0.078  0.062  0.018 -0.029 -0.060 -0.093
## [25] -0.108 -0.156 -0.159 -0.186 -0.204 -0.228
```

```
# scree plot from tetrachoric correlations
scree(gds.tet$rho, factors = FALSE, pc = TRUE, hline = -1)
```

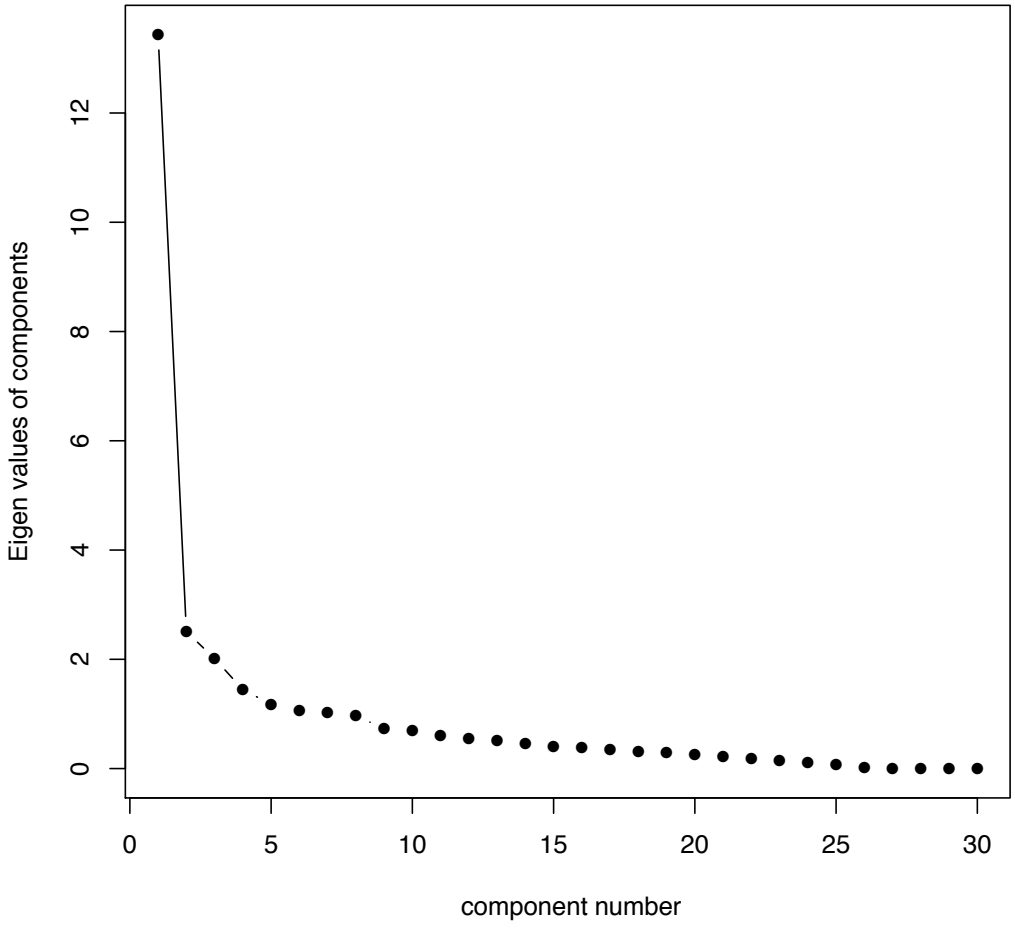


Figure 2.8 Scree plot for Geriatric Depression Scale data using tetrachoric correlations.

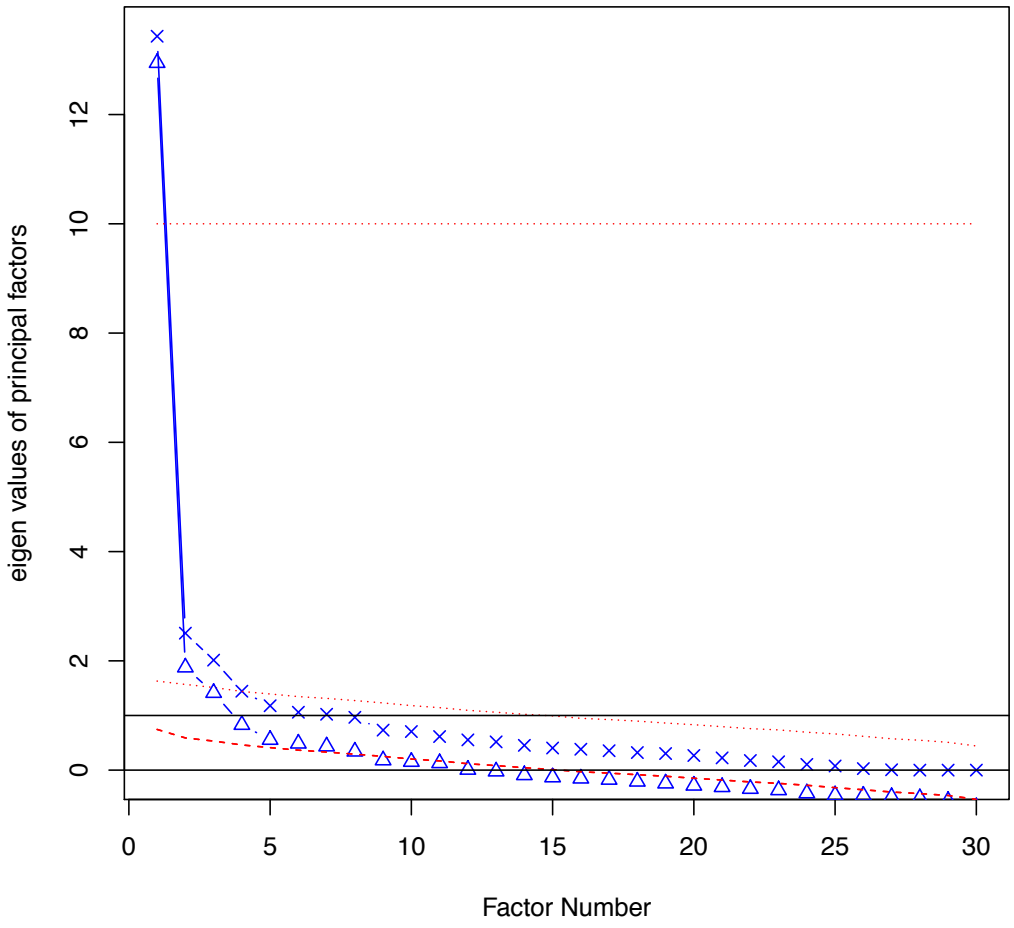


Figure 2.9 Plot of parallel analysis results for GDS data using item data and tetrachoric correlations.

To conduct a parallel analysis using tetrachoric correlations of randomly generated item data, use the `fa.parallel.poly()` function. Depending on the size of the data, this can take a long time to compute. Thus, the default number of random datasets is 10, but this can be changed using the `n.iter` argument.

```
# parallel analysis using tetrachoric correlations--WARNING: Takes a long time
fa.parallel.poly(gds.data)
# MAP values using tetrachoric correlations
VSS(gds.tet$rho, n = 7, plot = FALSE, n.obs = 656)$map
```

```
# factor loadings using tetrachoric correlations

# single factor -- tetrachoric
gds.paf.one.tet <- fa(gds.tet$rho, nfactors = 1)
print(gds.paf.one.tet$loadings, cut = 0)

# five factor -- tetrachoric
gds.paf.five.tet <- fa(gds.tet$rho, nfactors = 5)
print(gds.paf.five.tet$loadings, cut = 0)

# eight factor -- tetrachoric
```

```
gds.paf.eight.tet <- fa(gds.tet$rho, nfactors = 8)
print(gds.paf.eight.tet$loadings, cut = 0)
```

2.4.3 Full-Information Item Factor Analysis

For full-information item factor analysis (FIIFA) use the *mirt* package. It has the `mirt()` function which will conduct FIIFA with item-level data. It needs three arguments. The first is the data, the second is the number of factors to extract using the `model` argument, and the third is the rotation to use using the `rotate` argument. By default, it uses direct oblimin rotation. As a warning, it can take long time for the estimation to coverage if there are a lot of variables or you are extracting a lot of factors. In fact, the eight factor solution is not even viable for this method of factor analysis.

```
# full information factor analysis
library(mirt)
# single factor
gds.fiifa.1factor <- mirt(gds.data, model = 1)
summary(gds.fiifa.1factor)
# five factor
gds.fiifa.5factor <- mirt(gds.data, model = 5)
summary(gds.fiifa.5factor)
```

To see the factor loadings, use the `summary()` function on an **R** object result from `mirt()`. The *h2* column contains the communality values.

```
# one factor loadings
summary(gds.fiifa.1factor)
```

```
##
## Factor loadings metric:
##          F1    h2
## GDS01 0.829 0.687
## GDS02 0.598 0.358
## GDS03 0.868 0.753
## GDS04 0.710 0.504
## GDS05 0.730 0.533
## GDS06 0.678 0.459
## GDS07 0.813 0.660
## GDS08 0.695 0.484
## GDS09 0.838 0.703
## GDS10 0.808 0.653
## GDS11 0.571 0.326
## GDS12 0.490 0.241
## GDS13 0.577 0.333
## GDS14 0.441 0.194
## GDS15 0.744 0.553
## GDS16 0.887 0.786
## GDS17 0.875 0.766
## GDS18 0.700 0.490
## GDS19 0.725 0.526
## GDS20 0.606 0.367
## GDS21 0.673 0.453
```

```
## GDS22 0.878 0.771
## GDS23 0.695 0.483
## GDS24 0.558 0.311
## GDS25 0.808 0.652
## GDS26 0.642 0.412
## GDS27 0.516 0.266
## GDS28 0.542 0.294
## GDS29 0.446 0.199
## GDS30 0.527 0.278
##
## SS loadings:  14
##
## Factor covariance:
##    F1
## F1  1
```

```
# five factor loadings
summary(gds.fiifa.5factor)
```

```
##
## Rotation:  oblimin
##
## Rotated factor loadings:
##
##          F_1    F_2    F_3    F_4    F_5    h2
## GDS01  0.223  0.071 -0.153  0.497  0.558 0.921
## GDS02 -0.030 -0.169  0.099 -0.109  0.885 0.860
## GDS03  0.533  0.276  0.035  0.282  0.429 0.920
## GDS04  0.651  0.121  0.090  0.022  0.259 0.648
## GDS05 -0.070 -0.008  0.144  0.737  0.169 0.719
## GDS06  0.616 -0.291  0.145  0.005 -0.085 0.632
## GDS07  0.211  0.060 -0.080  0.902 -0.127 0.903
## GDS08  0.463 -0.081  0.011  0.372 -0.027 0.558
## GDS09  0.189 -0.042 -0.051  0.817  0.055 0.894
## GDS10  0.424 -0.227  0.144  0.189  0.184 0.681
## GDS11  0.743 -0.075 -0.027 -0.092  0.049 0.548
## GDS12  0.019  0.016  0.926  0.028 -0.095 0.820
## GDS13  0.587 -0.054 -0.022  0.258 -0.139 0.507
## GDS14  0.232 -0.771  0.020 -0.199 -0.015 0.727
## GDS15 -0.055 -0.027  0.091  0.916 -0.038 0.828
## GDS16  0.673  0.027  0.190  0.270  0.019 0.860
## GDS17  0.345 -0.176  0.168  0.206  0.382 0.809
## GDS18  0.756 -0.133  0.001 -0.002  0.008 0.659
## GDS19 -0.041 -0.124  0.322  0.569  0.128 0.678
## GDS20  0.114 -0.346  0.232  0.055  0.262 0.490
## GDS21 -0.188 -0.296  0.326  0.454  0.228 0.647
## GDS22  0.419 -0.044  0.195  0.302  0.238 0.780
## GDS23  0.682 -0.044  0.067 -0.002  0.097 0.588
## GDS24  0.683 -0.173  0.035 -0.004 -0.127 0.523
## GDS25  0.800 -0.088  0.056  0.048  0.015 0.784
## GDS26  0.239 -0.635 -0.004  0.029  0.194 0.703
## GDS27 -0.120 -0.103  0.163  0.520  0.104 0.378
## GDS28  0.178  0.083  0.717 -0.091  0.142 0.630
## GDS29  0.111 -0.356 -0.036  0.147  0.169 0.289
```

```
## GDS30 -0.049 -0.778 -0.027  0.220  0.056 0.673
##
## Rotated SS loadings:  5.6 2.3 1.9 4.6 1.9
##
## Factor correlations:
##
##       F_1   F_2   F_3   F_4   F_5
## F_1  1.00 -0.33  0.30  0.52  0.40
## F_2 -0.33  1.00 -0.32 -0.11 -0.22
## F_3  0.30 -0.32  1.00  0.32  0.36
## F_4  0.52 -0.11  0.32  1.00  0.41
## F_5  0.40 -0.22  0.36  0.41  1.00
```

3 | Sample Size Matters

Chapter Contents

3.1 Selecting a Random Sample from the Dataset 23
3.2 Determining Sample Size for a Study Before Collecting Data 24

3.1 Selecting a Random Sample from the Dataset

The novel thing in this chapter concerning writing **R** syntax is selecting a random sample of size n from the entire dataset. This is most easily done using the `sample()` function. The first argument for the `sample()` function is a vector of values that contains all possible values to sample. The `size` argument specifies how many observations to sample. The third major argument is `replace=FALSE`, which means the same observation cannot be selected more than once.[1] After selecting the observations with the `sample()` function, subset the original data by selecting only those observations. To select observations, use the square bracket, `[ ]`, operator.

Exercise 1a requires drawing a random sample from the *SDQ data* to have a subject:item ratio of 2:1. Since the number of items is 13, the number of observations should be 26.

```
# random sample 2:1 -> n=26
sdq.data.n26 <- sdq.data[sample(1:nrow(sdq.data), 26, replace = FALSE), ]
```

The first argument in the `sample()` function creates an object with all the observations in the SDQ data (*sdq.data*). I saved the resulting subset data into a new **R** object named *sdq.data.n26*. Thus, conducting an EFA with this data is just a matter of using the syntax discussed in Chapter 2.

```
fa(sdq.data.n26, nfactors = 3, rotate = "promax", fm = "pa")
```

Below I give the syntax for creating the other subsets for Exercise 1 and their subsequent EFAs.

```
# random sample 5:1 -> n=65
sdq.data.n65 <- sdq.data[sample(1:nrow(sdq.data), 65, replace = FALSE), ]
fa(sdq.data.n65, nfactors = 3, rotate = "promax", fm = "pa")

# random sample 10:1 -> n=130
sdq.data.n130 <- sdq.data[sample(1:nrow(sdq.data), 130, replace = FALSE), ]
fa(sdq.data.n130, nfactors = 3, rotate = "promax", fm = "pa")

# random sample 20:1 -> n=260
sdq.data.n260 <- sdq.data[sample(1:nrow(sdq.data), 260, replace = FALSE), ]
fa(sdq.data.n260, nfactors = 3, rotate = "promax", fm = "pa")
```

[1] If the `replace=TRUE` argument was used, this would do sampling *with* replacement. Another term for sampling with replacement is *bootstrapping*, the topic of Chapter 5.

3.2 Determining Sample Size for a Study Before Collecting Data

When planning a study using that will use factor analysis, there are different methods available to determine the sample size needed (e.g., Lai & Kelley, 2011; MacCallum, Browne, & Sugawara, 1996). Muthén and Muthén (2002) discussed a very general method to this using Monte Carlo methods, which Beaujean (2014) showed how to implement using **R**.

4 | Replication Statistics in EFA

Chapter Contents

4.1 Import School Perceptions Questionnaire Data 25
4.2 Splitting a Dataset in Half with Random Observations 25
4.3 Comparing Factor Loadings 26

4.1 Import School Perceptions Questionnaire Data

This chapter uses a new dataset of items from the *School Perceptions Questionnaire* (SPQ). Thus, the first step is to import this data using the same `read.spss()` function I discussed in Chapter 2.

```
# School Perceptions Questionnaire (SPQ) data (chapter 4)
spq.data <- read.spss("CH02_ex1_engdata.savCh04_SPQ.SAV", to.data.frame = TRUE)
```

Before proceeding, I change the names of the variables in the dataset to match those in the book.

```
# rename variables
names(spq.data) <- paste("SPQ", seq(1, 13), sep = "")
```

4.2 Splitting a Dataset in Half with Random Observations

The novel thing in this chapter concerning writing **R** syntax is splitting a dataset into two random parts. This is most easily done using the `sample()` function, which I introduced in Chapter 3. There are two main differences in splitting a dataset in half and selecting a given number of observations.

First, instead of selecting a known number of observations, I select 50% of the observations. If the number of observations were always even, then selecting 50% would not be a problem as half of an even number is always an integer. When the number of observations is odd, though, taking half gives a number with a fractional component. Thus, one way to guarantee getting a integer when when splitting a data in half is to use the floor function, which takes any real number an returns the largest previous integer. In **R**, the floor function is called using `floor()`.

Second, I do not want to discard the observations not selected. In fact, I want to use the unselected observations to comprise the second sample. I can so this by first saving the selected observations as an **R** object. Then, second, create one dataset with the selected observations and create the another dataset with the opposite of the selected observations (i.e., the unselected). Do do the latter, use the minus sign - in front of observations object when creating the data subset.

```
# split data randomly into (approximaltely) two equal size groups
spq.size <- floor(0.5 * nrow(spq.data))
# select sample for group 1
```

```
spq.sample <- sample(seq(length.out = nrow(spq.data)), size = spq.size)
# create two groups
spq.data.1 <- spq.data[spq.sample, ]
spq.data.2 <- spq.data[-spq.sample, ]
```

After creating the new datasets, it is always a good idea to make sure they are the appropriate size.

```
# check sample sizes
nrow(spq.data.1)
nrow(spq.data.2)
```

4.3 Comparing Factor Loadings

The EFAs are conducted the same way I discussed in Chapter 2. As an aside, the factor loadings produced will not match those given by Osborne (2014) because the two groups are comprised of different observations.

```
# EFA: 2 factors
spq.efa.1 <- fa(spq.data.1, nfactors = 2, rotate = "oblimin", fm = "ml")
spq.efa.2 <- fa(spq.data.2, nfactors = 2, rotate = "oblimin", fm = "ml")

# loadings
print(spq.efa.1$loadings, cut = 0)


##
## Loadings:
##       ML1    ML2
## SPQ1   0.617  0.051
## SPQ2   0.494 -0.049
## SPQ3  -0.086  0.439
## SPQ4  -0.054  0.498
## SPQ5   0.697 -0.026
## SPQ6   0.163  0.523
## SPQ7   0.134  0.412
## SPQ8   0.569  0.107
## SPQ9  -0.407  0.391
## SPQ10  0.560  0.058
## SPQ11  0.698 -0.049
## SPQ12 -0.307  0.463
## SPQ13  0.160  0.410
##
##                ML1  ML2
## SS loadings    2.6 1.44
## Proportion Var 0.2 0.11
## Cumulative Var 0.2 0.31


print(spq.efa.2$loadings, cut = 0)


##
## Loadings:
```

```
##       ML1    ML2
## SPQ1   0.639  0.034
## SPQ2   0.469 -0.151
## SPQ3  -0.177  0.452
## SPQ4  -0.048  0.617
## SPQ5   0.631 -0.048
## SPQ6   0.090  0.502
## SPQ7   0.029  0.342
## SPQ8   0.535  0.019
## SPQ9  -0.404  0.294
## SPQ10  0.555  0.136
## SPQ11  0.733  0.017
## SPQ12 -0.349  0.341
## SPQ13  0.143  0.509
##
##                  ML1  ML2
## SS loadings    2.51 1.46
## Proportion Var 0.19 0.11
## Cumulative Var 0.19 0.30
```

There is not a native function to comparing the factor loadings using the method advocated by Osborne (2014). Nonetheless, such comparisons are not difficult to program in **R**. First, I create two **R** objects that contain the names of the variables that primarily load on each of the two factors.

```
# create scales
spq.scale.1 <- paste("SPQ", c(1, 2, 5, 8, 9, 10, 11, 12, 13), sep = "")
spq.scale.1


## [1] "SPQ1"  "SPQ2"  "SPQ5"  "SPQ8"  "SPQ9"  "SPQ10" "SPQ11" "SPQ12" "SPQ13"


spq.scale.2 <- paste("SPQ", c(3, 4, 6, 7, 9, 12, 13), sep = "")
spq.scale.2


## [1] "SPQ3"  "SPQ4"  "SPQ6"  "SPQ7"  "SPQ9"  "SPQ12" "SPQ13"
```

Subsequently, I subset the loadings separately for each group, subtract them, and square the difference.

```
# compare loadings

# factor 1
(spq.efa.1$loadings[spq.scale.1, 1] - spq.efa.2$loadings[spq.scale.1, 1])^2


##      SPQ1      SPQ2      SPQ5      SPQ8      SPQ9     SPQ10     SPQ11     SPQ12     SPQ13
## 0.0004671 0.0006090 0.0044335 0.0012205 0.0000096 0.0000340 0.0012602 0.0017946 0.0002782


# factor 2
(spq.efa.1$loadings[spq.scale.2, 2] - spq.efa.2$loadings[spq.scale.2, 2])^2


##    SPQ3    SPQ4    SPQ6    SPQ7    SPQ9   SPQ12   SPQ13
## 0.00018 0.01427 0.00042 0.00502 0.00939 0.01494 0.00988
```

5 | Bootstrapping Applications in EFA

Chapter Contents

5.1 Bootstrapping Initial Eigenvalues 28
5.2 Bootstrapping Factor Loadings 31

5.1 Bootstrapping Initial Eigenvalues

There are multiple ways to produce bootstrapped parameter estimates in **R**. I use the *boot* package and its `boot()` function as it can bootstrap just about any statistic and is relatively efficient. Moreover, it has the `boot.ci()` function, which estimates bootstrapped confidence intervals. The price for such a general and efficient function is that its use is not very intuitive. To use it requires writing your own function that will calculate the statistics of interest and also allows the statistic to be calculated over multiple datasets (i.e, the bootstrap samples). Canty (2002) does a good job introducing the package and function.

First, write a function to calculate and extract the eigenvalues from the bootstrapped samples.

```
# function to obtain eigenvalues from the data
eig.boot <- function(data, indices) {
    d <- data[indices, ]  # allows boot to select sample
    e.value <- eigen(cor(d))
    return(e.value[[1]])
}
```

Next, feed the function to the `boot()` function as well as the original dataset and the number of desired bootstrapped samples. If your computer has multiple cores or you have access to parallel processing, then you will want to use the `parallel` argument, too.

```
library(boot)
# bootstrap the data R=5000 (using multicore processing)
eig.bootstrapped <- boot(data = eng.data, statistic = eig.boot, R = 5000, parallel = "multicore")
```

To view the bootstrapped results, add `$t` to the **R** object returned from the `boot()` function.

```
# view all bootstrapped values
eig.bootstrapped$t
```

To plot the bootstrapped values for the first eigenvalue, use the `plot()` function along with the `index=1` argument. It returns a histogram as well as a Q-Q plot.

To estimate descriptive statistics for the first three eigenvalues, use the *psyh* package's `describe` function with a subset version of the bootstrapped samples data (i.e., `eig.bootstrapped$t`)

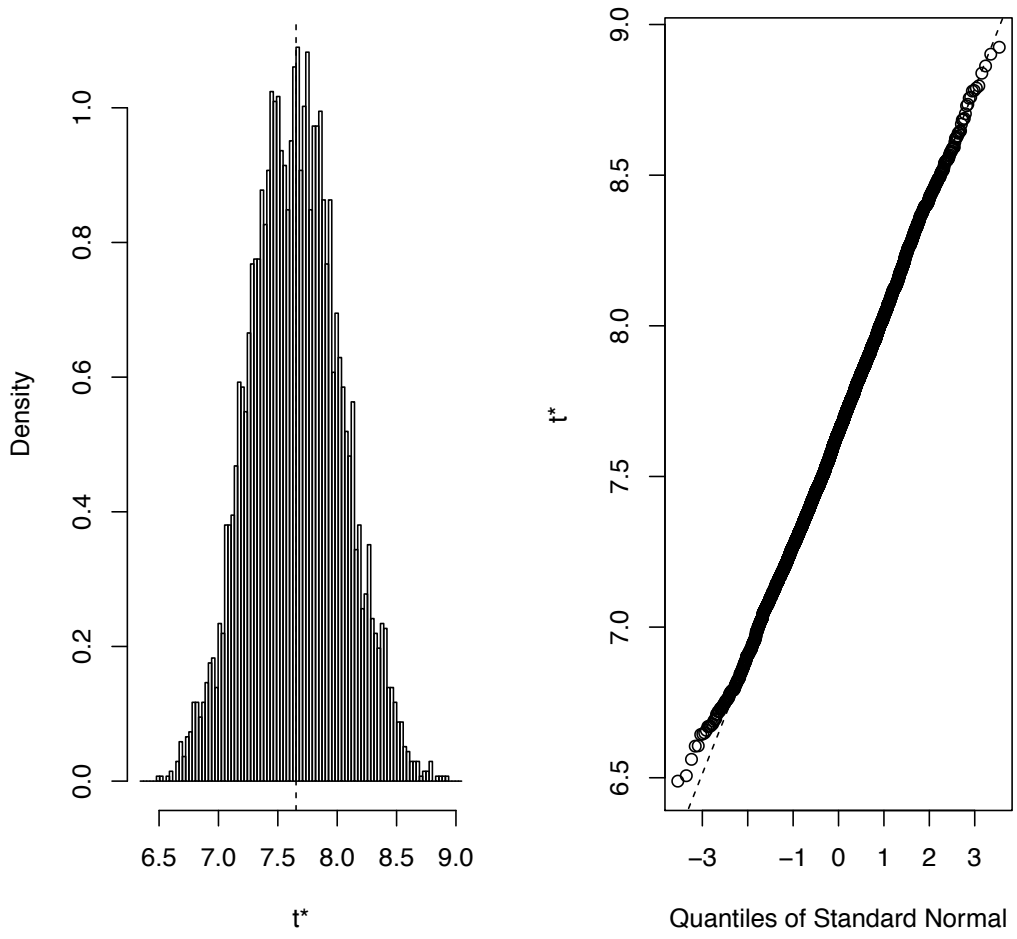


Figure 5.1 Distribution of first eigenvalue extracted from 5000 bootstrap samples of the engineering data.

```
# descriptive values for first three factors
describe(eig.bootstrapped$t[, 1:3])
```

The `boot.ci()` function gives different confidence intervals for the bootstrapped samples. I just examine the percentile and BCA confidence intervals.

```
# confidence intervals (percentile and BCA)
boot.ci(eig.bootstrapped, conf = 0.95, type = c("perc", "bca"))


## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 5000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = eig.bootstrapped, conf = 0.95, type = c("perc",
##     "bca"))
##
## Intervals :
## Level     Percentile            BCa
## 95%   ( 6.9,  8.4 )   ( 7.0,  8.5 )
## Calculations and Intervals on Original Scale
```

```
boot.ci(eig.bootstrapped, conf = 0.95, type = c("perc", "bca"), index = 2)


## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 5000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = eig.bootstrapped, conf = 0.95, type = c("perc",
##     "bca"), index = 2)
##
## Intervals :
## Level     Percentile            BCa
## 95%   ( 3,  4 )   ( 3,  4 )
## Calculations and Intervals on Original Scale


boot.ci(eig.bootstrapped, conf = 0.95, type = c("perc", "bca"), index = 3)


## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 5000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = eig.bootstrapped, conf = 0.95, type = c("perc",
##     "bca"), index = 3)
##
## Intervals :
## Level     Percentile            BCa
## 95%   ( 0.39,  0.60 )   ( 0.35,  0.53 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

To estimate the % variance explained, divide the eigenvalue by the number of variables in the dataset. To get confidence intervals for this function of the eigenvalue, define the function in `boot.ci()` using the `h` argument.

```
# mean of % variance explained
mean(eig.bootstrapped$t[,1]/14)*100


## [1] 55


# SD of % variance explained
sd(eig.bootstrapped$t[,1]/14)*100


## [1] 2.7



# confidence intervals (percentile and BCA) for % varaince explained
boot.ci(eig.bootstrapped, conf = 0.95, type = c("perc", "bca"), index=1,
h=function(x) x/14*100)


## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
```

```
## Based on 5000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = eig.bootstrapped, conf = 0.95, type = c("perc",
##     "bca"), index = 1, h = function(x) x/14 * 100)
##
## Intervals :
## Level     Percentile            BCa
## 95%   (49, 60 )   (50, 61 )
## Calculations and Intervals on  Transformed Scale


boot.ci(eig.bootstrapped, conf = 0.95, type = c("perc", "bca"), index=2,
h=function(x) x/14*100)


## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 5000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = eig.bootstrapped, conf = 0.95, type = c("perc",
##     "bca"), index = 2, h = function(x) x/14 * 100)
##
## Intervals :
## Level     Percentile            BCa
## 95%   (21, 29 )   (21, 29 )
## Calculations and Intervals on  Transformed Scale


boot.ci(eig.bootstrapped, conf = 0.95, type = c("perc", "bca"), index=3,
h=function(x) x/14*100)


## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 5000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = eig.bootstrapped, conf = 0.95, type = c("perc",
##     "bca"), index = 3, h = function(x) x/14 * 100)
##
## Intervals :
## Level     Percentile            BCa
## 95%   ( 2.8,  4.3 )   ( 2.5,  3.8 )
## Calculations and Intervals on  Transformed Scale
## Some BCa intervals may be unstable
```

5.2 Bootstrapping Factor Loadings

The `fa()` function has the ability to bootstrap the factor loading. This is done using the `n.iter` argument with a value > 1, which indicates the number of bootstrapped samples to estimate.

```
# bootstrapped loaadings
eng.paf.oblimin.boot <- fa(eng.data, nfactors = 2, rotate = "oblimin", fm = "pa", n.iter = 5000)
```

```
eng.paf.oblimin.boot



..<Output Omitted>..

##  Coefficients and bootstrapped confidence intervals
##                low   PA1 upper   low   PA2 upper
## EngProbSolv1  0.81  0.86  0.90 -0.07 -0.02  0.04
## EngProbSolv2  0.78  0.84  0.90 -0.14 -0.07 -0.01
## EngProbSolv3  0.84  0.88  0.92 -0.05 -0.01  0.04
## EngProbSolv4  0.87  0.91  0.94 -0.06 -0.03  0.01
## EngProbSolv5  0.85  0.89  0.92 -0.02  0.02  0.07
## EngProbSolv6  0.83  0.87  0.91 -0.03  0.02  0.07
## EngProbSolv7  0.82  0.87  0.91 -0.02  0.03  0.08
## EngProbSolv8  0.73  0.79  0.85  0.01  0.07  0.13
## INTERESTeng1 -0.02  0.04  0.11  0.72  0.80  0.88
## INTERESTeng2 -0.07 -0.02  0.02  0.88  0.92  0.96
## INTERESTeng3 -0.05 -0.01  0.03  0.89  0.92  0.95
## INTERESTeng4 -0.04  0.00  0.04  0.84  0.90  0.97
## INTERESTeng5 -0.05  0.00  0.05  0.85  0.90  0.94
## INTERESTeng6 -0.03  0.01  0.06  0.81  0.86  0.92
##
##  Interfactor correlations and bootstrapped confidence intervals
##         lower estimate upper
## PA1-PA2  0.25     0.37  0.48
```

6 | Data Cleaning and EFA

There are no reproducible examples for this chapter.

7 | Are Factor Scores a Good Idea?

There are no reproducible examples for this chapter.

8 | Higher Order Factors

Chapter Contents

8.1 Importing and Subsetting the Data 35
8.2 Higher Order EFA 35
8.3 Alternatives for Higher Order Factors in Exploratory Factor Analysis 38

8.1 Importing and Subsetting the Data

This analysis requires loading a new data set, which is the engineering data augmented with questions about feelings of belongingness in engineering.

```
# extended engineering data
eng.extended.data <- read.spss("ex3-efa-VTdata.2.sav", to.data.frame = TRUE)
```

This dataset has variables other than the 22 items of interest. To subset the data, make a **R** object that has the names of the 22 items. Then, create a new **R** object with just the final items.

```
# subset data
eng.scale <- c(paste("BELONGeng", 1:8,sep=""), paste("EngProbSolv", 1:8,sep=""),
paste("INTERESTeng", 1:6,sep=""))

# create dataset with only items of interest
eng.extended.data <- eng.extended.data[eng.scale]
```

8.2 Higher Order EFA

For the first-order EFA, follow the steps from Chapter 2. To extract the structure coefficients, append `$Structure` to the `fa()` object. Likewise, to extract the factor correlations, append `$Phi` to the `fa()` object.

```
# first order EFA
eng.extended.paf <- fa(eng.extended.data, nfactors = 3, rotate = "promax", fm = "pa")

# eigenvalues
eng.extended.paf$values


##  [1]  9.2760  3.4734  1.4078  0.7575  0.2556  0.1852  0.0770  0.0721  0.0539  0.0344
## [11]  0.0074 -0.0148 -0.0311 -0.0343 -0.0601 -0.0824 -0.1000 -0.1213 -0.1446 -0.1789
## [21] -0.2880 -0.3880



# pattern coefficients
print(eng.extended.paf$loadings, cut = 0)
```

```
##
## Loadings:
##              PA1    PA2    PA3
## BELONGeng1    0.263 -0.012  0.535
## BELONGeng2   -0.079  0.093  0.516
## BELONGeng3   -0.067 -0.142  0.886
## BELONGeng4   -0.112 -0.079  0.872
## BELONGeng5    0.005  0.073  0.299
## BELONGeng6    0.216 -0.074  0.659
## BELONGeng7   -0.259  0.369  0.378
## BELONGeng8    0.141  0.068  0.541
## EngProbSolv1  0.850 -0.016  0.015
## EngProbSolv2  0.827 -0.067  0.019
## EngProbSolv3  0.874 -0.008  0.013
## EngProbSolv4  0.924  0.002 -0.044
## EngProbSolv5  0.885  0.034 -0.009
## EngProbSolv6  0.881  0.047 -0.040
## EngProbSolv7  0.856  0.050 -0.002
## EngProbSolv8  0.772  0.060  0.045
## INTERESTeng1  0.038  0.790  0.027
## INTERESTeng2  0.015  0.954 -0.092
## INTERESTeng3  0.024  0.950 -0.078
## INTERESTeng4  0.040  0.945 -0.097
## INTERESTeng5 -0.014  0.871  0.039
## INTERESTeng6  0.013  0.865 -0.005
##
##                  PA1  PA2  PA3
## SS loadings    6.14 5.04 3.09
## Proportion Var 0.28 0.23 0.14
## Cumulative Var 0.28 0.51 0.65
```

```
# structure coefficients
print(eng.extended.paf$Structure, cut = 0)
```

```
##
## Loadings:
##              [,1]  [,2]  [,3]
## BELONGeng1   0.550 0.384 0.671
## BELONGeng2   0.236 0.356 0.525
## BELONGeng3   0.365 0.335 0.769
## BELONGeng4   0.335 0.374 0.766
## BELONGeng5   0.195 0.244 0.343
## BELONGeng6   0.549 0.376 0.735
## BELONGeng7   0.078 0.490 0.445
## BELONGeng8   0.460 0.424 0.656
## EngProbSolv1 0.852 0.295 0.469
## EngProbSolv2 0.814 0.239 0.433
## EngProbSolv3 0.879 0.311 0.486
## EngProbSolv4 0.900 0.306 0.461
## EngProbSolv5 0.892 0.344 0.492
## EngProbSolv6 0.876 0.339 0.467
## EngProbSolv7 0.873 0.354 0.493
## EngProbSolv8 0.818 0.361 0.500
```

```
## INTERESTeng1 0.334 0.819 0.494
## INTERESTeng2 0.305 0.907 0.456
## INTERESTeng3 0.320 0.915 0.472
## INTERESTeng4 0.324 0.904 0.459
## INTERESTeng5 0.318 0.889 0.524
## INTERESTeng6 0.318 0.867 0.491
##
##                [,1] [,2] [,3]
## SS loadings    7.74 6.66 6.40
## Proportion Var 0.35 0.30 0.29
## Cumulative Var 0.35 0.66 0.95
```

```
# factor correlations coefficients
eng.extended.paf$Phi
```

```
##      [,1] [,2] [,3]
## [1,] 1.00 0.36 0.55
## [2,] 0.36 1.00 0.57
## [3,] 0.55 0.57 1.00
```

There is no raw data for the second-order EFA. Instead, the first-order factors' correlations are directly used as input for the `fa()` function.

```
# second order EFA
eng.extended.second.paf <- fa(eng.extended.paf$Phi, nfactors = 1, fm = "pa")

# eignevalues
eng.extended.second.paf$values
```

```
## [1]  1.5721  0.0031 -0.0041
```

```
# pattern coefficients
print(eng.extended.second.paf$loadings, cut = 0)
```

```
##
## Loadings:
##      PA1
## [1,] 0.59
## [2,] 0.61
## [3,] 0.92
##
##                  PA1
## SS loadings    1.57
## Proportion Var 0.52
```

8.3 Alternatives for Higher Order Factors in Exploratory Factor Analysis

Interpreting second-order factors based on their loadings on first-order factors is difficult. It would be much easier to interpret the second-order factors if we knew how the original variables loaded onto them. Schmid and Leiman (1957) developed a way to do this, in a procedure that is now called Schmid-Leiman (SL) transformation. The gist of the transformation is that it combines the first- and second-order factor loadings to produce loadings for the original variables onto all the factors.[1]

If there is a only a single second-order factor, then the *psych* package's `schmid()` function will produce loadings from the SL transformation. Its arguments are the same as those for the the `fa()` function. In the output, the column labeled g contains the loadings on the second-order factor.[2]

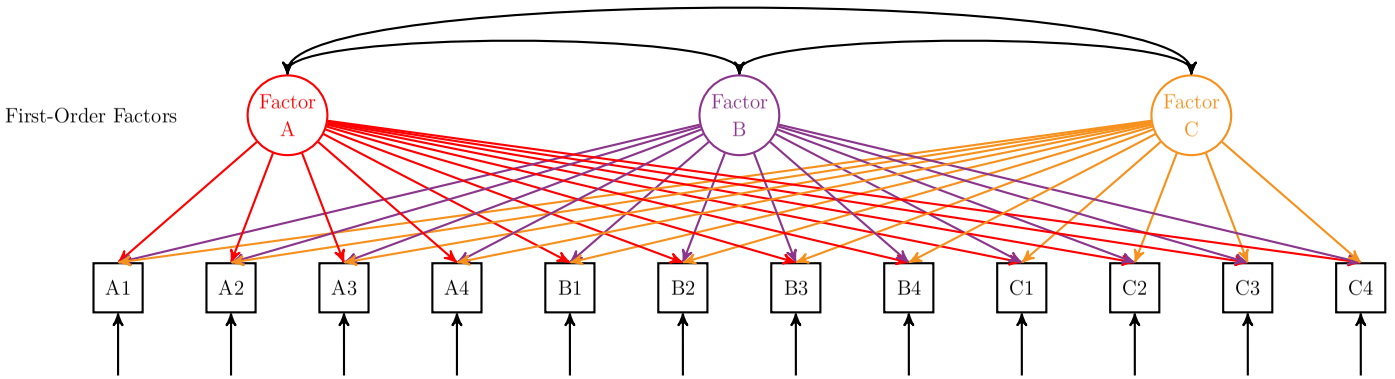
```
# schmid-leiman
eng.extended.sl.paf <- schmid(eng.extended.data, nfactors = 3, rotate = "promax", fm = "pa")

# loadings
print(eng.extended.sl.paf$sl, cut = 0)
```

```
##                  g     F1*     F2*      F3*   h2   u2   p2
## BELONGeng1    0.64  0.2124 -0.0095  0.20473 0.50 0.50 0.82
## BELONGeng2    0.49 -0.0635  0.0736  0.19745 0.29 0.71 0.83
## BELONGeng3    0.69 -0.0543 -0.1124  0.33910 0.61 0.39 0.79
## BELONGeng4    0.69 -0.0905 -0.0626  0.33374 0.60 0.40 0.80
## BELONGeng5    0.32  0.0044  0.0575  0.11458 0.12 0.88 0.86
## BELONGeng6    0.69  0.1743 -0.0585  0.25247 0.58 0.42 0.83
## BELONGeng7    0.42 -0.2093  0.2920  0.14457 0.33 0.67 0.54
## BELONGeng8    0.62  0.1143  0.0537  0.20706 0.45 0.55 0.87
## EngProbSolv1 0.50  0.6874 -0.0128  0.00558 0.73 0.27 0.35
## EngProbSolv2 0.46  0.6688 -0.0529  0.00742 0.67 0.33 0.32
## EngProbSolv3 0.52  0.7068 -0.0063  0.00516 0.77 0.23 0.35
## EngProbSolv4 0.50  0.7469  0.0017 -0.01700 0.81 0.19 0.31
## EngProbSolv5 0.53  0.7158  0.0265 -0.00360 0.80 0.20 0.36
## EngProbSolv6 0.51  0.7119  0.0375 -0.01516 0.77 0.23 0.34
## EngProbSolv7 0.53  0.6924  0.0396 -0.00069 0.76 0.24 0.37
## EngProbSolv8 0.53  0.6245  0.0477  0.01726 0.68 0.32 0.42
## INTERESTeng1 0.53  0.0306  0.6261  0.01016 0.67 0.33 0.42
## INTERESTeng2 0.51  0.0123  0.7555 -0.03505 0.83 0.17 0.31
## INTERESTeng3 0.52  0.0192  0.7530 -0.03001 0.84 0.16 0.32
## INTERESTeng4 0.51  0.0322  0.7486 -0.03706 0.82 0.18 0.32
## INTERESTeng5 0.56 -0.0110  0.6903  0.01505 0.79 0.21 0.40
## INTERESTeng6 0.53  0.0104  0.6856 -0.00199 0.75 0.25 0.37
```

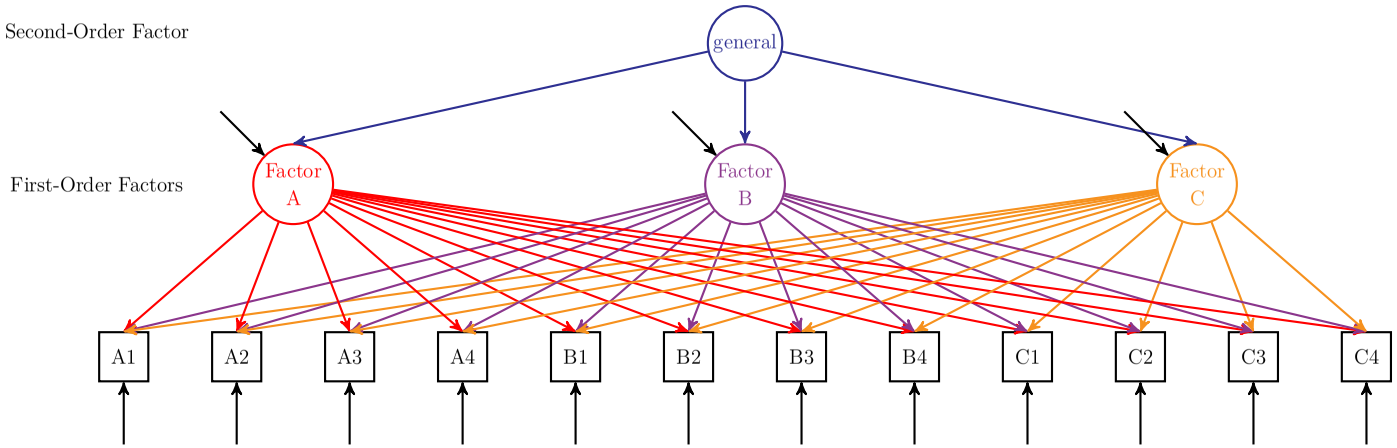
While the SL transformation makes it easier to interpret second-order factors, it comes at a price. The first issue is that the first-order factors are now orthogonal factors. This is shown graphically in Figure 8.1b. This orthogonalization is not really a problem for most constructs, however, as typically the second-order factor is thought to explain all the covariance among the first order factors.

[1] Gorsuch (1983) provides an accessible explanation of the steps involved in the SL transformation.

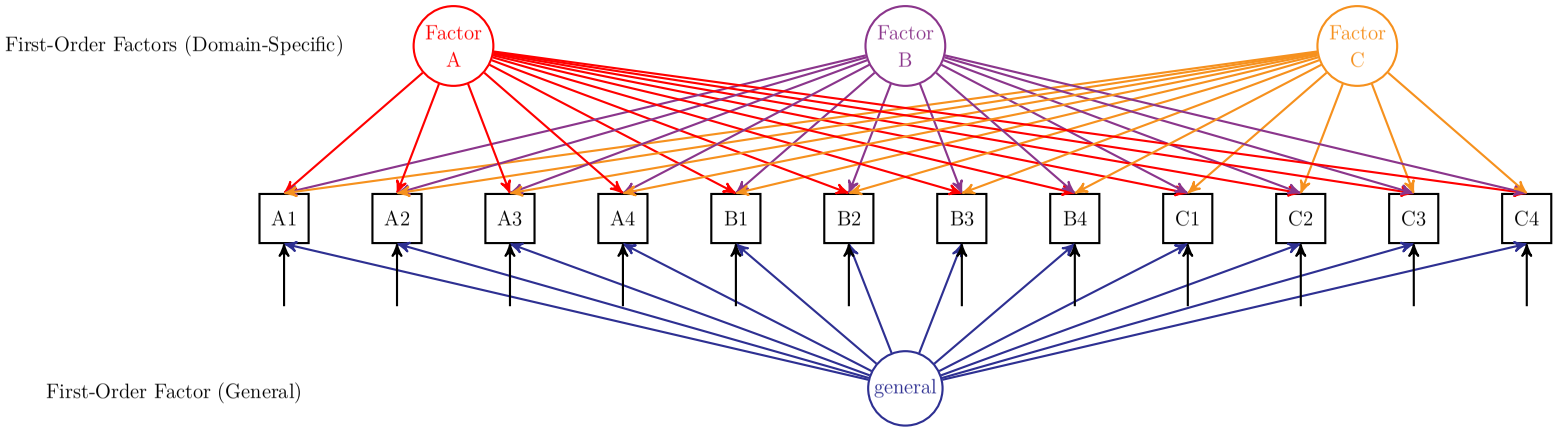
[2] g stands for Spearman's (1995) general intelligence factor. It is named that because historically intelligence researchers have been the main users of higher order factor models (e.g., Carroll, 1993).



(a) Oblique model.



(b) Higher order model with first- and second-order factors.



(c) Bi-factor model.

Figure 8.1 Graphical representations of hierarchical factor models. All error variances shown with arrow with no attached node. Colors are used to highlight the loadings from the differnt factors.

An issue of greater concern with the SL transformation is that it imposes proportionality constraints. Specifically, for a given set of variables that load onto a first-order factor, the ratio of the variables' variance due to the first-order factor to the variance attributable to the second-order factor are constrained to be the same.

The proportionality constraint issue can be difficult to grasp initially. Beaujean, Parkin, and Parker (2014) and Schmiedek and Li (2004) explain it in more detail. The main problem it produces with EFA is that it places limits the values of the variables' loadings on the second-order factor.

An alternative to a higher order model is a bifactor model (Holzinger & Swineford, 1937), sometimes called a direct hierarchical or nested-factors model. An example is shown in Figure 8.1c. Like the SL transformation, the bifactor model posits that there is a general factor that influences all the variables as well as orthogonal, domain-specific factors that primarily only influence a subset of the variables. Unlike the SL transformation, in the bifactor model all the factors are first-order. This provides two advantages over the higher order model for EFA. First, there is no need to use the SL transformation and its proportionality constraints. Second, it directly produces factor loadings for the general factor's influence on the variables.

Historically, bifactor models have only been used in confirmatory factor analysis. Recently, Jennrich and Bentler (2011; 2012) developed bifactor rotations for EFA. To use the bifactor rotation in **R**, use the `fa()` function with the `rotate="bifactor"` argument. Be sure to add one extra factor to the `nfactors` argument to account for the general factor. In the output, the first factor is the general factor and the subsequent factors are the domain-specific factors.

```
# bi-factor rotation
eng.extended.bifactor.paf <- fa(eng.extended.data, nfactors = 4, rotate = "bifactor", fm = "pa")

# pattern coefficients
print(eng.extended.bifactor.paf$loadings, cut = 0)


##
## Loadings:
##               PA1    PA2    PA3    PA4
## BELONGeng1    0.577  0.141  0.376  0.062
## BELONGeng2    0.375 -0.020  0.122  0.775
## BELONGeng3    0.439  0.014  0.686  0.094
## BELONGeng4    0.444 -0.048  0.677  0.093
## BELONGeng5    0.275  0.029 -0.012  0.551
## BELONGeng6    0.574  0.132  0.542 -0.020
## BELONGeng7    0.355 -0.313  0.060  0.545
## BELONGeng8    0.546  0.034  0.356  0.118
## EngProbSolv1  0.698  0.481  0.021 -0.082
## EngProbSolv2  0.641  0.503  0.004 -0.040
## EngProbSolv3  0.724  0.488  0.025 -0.096
## EngProbSolv4  0.734  0.518 -0.036 -0.068
## EngProbSolv5  0.752  0.477 -0.012 -0.060
## EngProbSolv6  0.739  0.474 -0.052 -0.028
## EngProbSolv7  0.748  0.464 -0.044  0.015
## EngProbSolv8  0.718  0.400  0.020 -0.030
## INTERESTeng1  0.699 -0.433  0.008  0.006
```

```
## INTERESTeng2  0.732 -0.538 -0.069 -0.017
## INTERESTeng3  0.746 -0.530 -0.066 -0.007
## INTERESTeng4  0.741 -0.517 -0.080 -0.009
## INTERESTeng5  0.732 -0.513  0.028 -0.010
## INTERESTeng6  0.718 -0.489 -0.014  0.005
##
##                  PA1  PA2  PA3   PA4
## SS loadings    9.02 3.49 1.53 1.265
## Proportion Var 0.41 0.16 0.07 0.057
## Cumulative Var 0.41 0.57 0.64 0.696
```

9 | After the EFA: Internal Consistency

Chapter Contents

9.1 Coefficient Alpha 42
9.2 Bootstrapped Confidence Intervals for Alpha 46
9.3 Other Measures of Reliability 47

9.1 Coefficient Alpha

There are many **R** packages that can estimate the Guttman (1945)-Cronbach (1951) α measure of internal consistency. I demonstrate it using a function from the *MBESS* (Kelley, 2007) package as it has many options other packages' functions do not have.[1]

The first example uses only the parent items from the SDQ data. Before estimating the reliability of the parent score, use the subsetting procedures from Chapter 8 to create an **R** object with just the items of interest.

```
# create R object with parent item names
parent <- paste("Par", 1:5, sep = "")
# subset data with only parent items
sdq.parent.data <- sdq.data[, parent]
```

To estimate α, use the `ci.reliability()` function with the `type="alpha"` argument.

```
# load MBESS package
library(MBESS)
# alpha
ci.reliability(sdq.parent.data, type = "alpha")


## $est
## [1] -0.34
##
## $se
## [1] 0.017
##
## $ci.lower
## [1] 0
##
## $ci.upper
## [1] -0.31
##
## $conf.level
## [1] 0.95
##
```

[1] *MBESS* stands for *M*ethods for the *B*ehavioral, *E*ducational, and *S*ocial *S*ciences.

```
## $est.type
## [1] "alpha"
##
## $analysis.type
## [1] "analytic"
##
## $interval.type
## [1] "normal-theory"
##
## $call
## ci.reliability(data = sdq.parent.data, type = "alpha")
##
## $boot
## NULL
```

The α value is the number under *est*. As expected, because the second and fourth items are reverse coded the α value is very low, -0.34.[2] Unlike most other packages estimating α, the results from the `ci.reliability()` function include the standard error and 95% confidence interval (CI). As the value for α is negative, however, the 95% CI should be interpreted with great caution.

An easy way to change the values of reverse-coded items whose scale starts at one is to subtract the reverse-coded items' values from a constant that is one unit higher than the highest value of the scale. As the highest value on the SDQ scale is 6, subtracting the reverse-coded items' values from 7 places their values on the same scale as the other items.

```
# reverse code items 2 and 4, then replace original values
sdq.parent.data[, c("Par2", "Par4")] <- 7 - sdq.parent.data[, c("Par2", "Par4")]
```

```
# alpha reliability with reverse-coded items coded correctly
ci.reliability(sdq.parent.data, type = "alpha")


## $est
## [1] 0.83
##
## $se
## [1] 0.0021
##
## $ci.lower
## [1] 0.83
##
## $ci.upper
## [1] 0.84
##
## $conf.level
## [1] 0.95
##
## $est.type
## [1] "alpha"
##
## $analysis.type
```

[2] As an aside, the cost of using reverse-coded items typically far outweighs any benefit they might provide (e.g., Herche & Engelland, 1996; Sliter & Zickar, 2014).

```
## [1] "analytic"
##
## $interval.type
## [1] "normal-theory"
##
## $call
## ci.reliability(data = sdq.parent.data, type = "alpha")
##
## $boot
## NULL
```

The SDQ parent score's α reliability is now estimated to be 0.83. The standard error of that estimate is 0.002 and the 95% CI is 0.83-0.84.

There are **R** packages that have functions to estimate the item-total correlations and corrected item-total correlations, but the calculation is relatively simple (see McDonald, 1999). Thus, I calculate the values directly by first calculating the total scores, and then estimating the correlation and corrected correlation between the items and the total score.

```
# total score
sdq.parent.total <- apply(sdq.parent.data, 1, sum)
# item-total correlations
sdq.parent.item.total <- cor(sdq.parent.data, sdq.parent.total, use = "complete")
sdq.parent.item.total


##      [,1]
## Par1 0.77
## Par2 0.74
## Par3 0.83
## Par4 0.74
## Par5 0.81



# corrected total score
sdq.parent.total.corrected <- sdq.parent.total - sdq.parent.data
# corrected item-total correlations
sdq.parent.item.total.corrected <- diag(cor(sdq.parent.data, sdq.parent.total.corrected,
use = "complete"))
sdq.parent.item.total.corrected


## Par1 Par2 Par3 Par4 Par5
## 0.64 0.60 0.73 0.57 0.66
```

The procedures to estimate α and the item-total correlations for the GDS score are the same as for the SDQ data, only there are no reverse-coded items to recode.

```
# alpha for GDS score
ci.reliability(gds.data, type = "alpha")


## $est
## [1] 0.89
```

```
##
## $se
## [1] 0.0057
##
## $ci.lower
## [1] 0.88
##
## $ci.upper
## [1] 0.9
##
## $conf.level
## [1] 0.95
##
## $est.type
## [1] "alpha"
##
## $analysis.type
## [1] "analytic"
##
## $interval.type
## [1] "normal-theory"
##
## $call
## ci.reliability(data = gds.data, type = "alpha")
##
## $boot
## NULL
```

```
# GDS item-total correlations

# total score
gds.total <- apply(gds.data, 1, sum)
# item-total correlations
gds.item.total <- cor(gds.data, gds.total, use = "complete")
gds.item.total
```

```
##       [,1]
## GDS01 0.56
## GDS02 0.47
## GDS03 0.58
## GDS04 0.56
## GDS05 0.50
## GDS06 0.56
## GDS07 0.44
## GDS08 0.40
## GDS09 0.58
## GDS10 0.62
## GDS11 0.47
## GDS12 0.44
## GDS13 0.44
## GDS14 0.32
## GDS15 0.46
## GDS16 0.65
```

```
## GDS17 0.65
## GDS18 0.43
## GDS19 0.61
## GDS20 0.51
## GDS21 0.53
## GDS22 0.51
## GDS23 0.42
## GDS24 0.45
## GDS25 0.56
## GDS26 0.54
## GDS27 0.42
## GDS28 0.48
## GDS29 0.40
## GDS30 0.44
```

```
# corrected total score
gds.total.corrected <- gds.total - gds.data
# corrected item-total correlations
gds.item.total.corrected <- diag(cor(gds.data, gds.total.corrected, use = "complete"))
gds.item.total.corrected
```

```
## GDS01 GDS02 GDS03 GDS04 GDS05 GDS06 GDS07 GDS08 GDS09 GDS10 GDS11 GDS12 GDS13 GDS14 GDS15
##  0.52  0.40  0.55  0.50  0.46  0.51  0.40  0.36  0.54  0.57  0.41  0.37  0.38  0.26  0.42
## GDS16 GDS17 GDS18 GDS19 GDS20 GDS21 GDS22 GDS23 GDS24 GDS25 GDS26 GDS27 GDS28 GDS29 GDS30
##  0.61  0.61  0.39  0.55  0.45  0.46  0.48  0.37  0.38  0.52  0.48  0.36  0.42  0.33  0.36
```

9.2 Bootstrapped Confidence Intervals for Alpha

One of the reasons I used the *MBESS* package's `ci.reliability()` function is that it is designed to estimate bootstrapped CI for α. To obtain the bootstrapped CI use either the `interval.type="perc"` (for a percentile CI) or `interval.type="bca"` (for bias-corrected and accelerated CI) argument. By default, the number of bootstrapped resampled is 1,000, but this can be changed using the `B` argument.

```
# bootstrapped CI for alpha of SDQ parent score with 500 bootstrapped samples
ci.reliability(sdq.parent.data, type = "alpha", interval.type = "perc", B = 500)
```

```
## $est
## [1] 0.83
##
## $se
## NULL
##
## $ci.lower
## [1] 0.83
##
## $ci.upper
## [1] 0.84
##
## $conf.level
```

```
## [1] 0.95
##
## $est.type
## [1] "alpha"
##
## $analysis.type
## [1] "analytic"
##
## $interval.type
## [1] "perc"
##
## $call
## ci.reliability(data = sdq.parent.data, type = "alpha", interval.type = "perc",
##     B = 500)
##
## $boot
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = data, statistic = .bs.alpha, R = B, stype = "i")
##
##
## Bootstrap Statistics :
##     original     bias    std. error
## t1*     0.83 -0.0000097       0.003
```

9.3 Other Measures of Reliability

For proper estimation and interpretation of coefficient α, the instrument's items have to meet some strong assumptions that often elude educational and psychological measures (Yanyun & Green, 2011). There are a few alternative measures of internal consistency (e.g., Revelle & Zinbarg, 2009), but one that is directly related to factor analysis is McDonald's (1999) ω. The only major assumption for is that all the items are measuring the same construct (i.e., unidimensional). If the instrument's items meet the assumptions to use α, then α and ω will be the same. If the instrument's items do not meet the assumptions, then ω is a more accurate measure of the score's reliability.

An example of an unidimensional factor model with k items is shown in Figure 9.1.[3] If items one through k are summed to produce Y, then Y's variance can be decomposed into the portion due to the common factor and the portion due to error variance. ω is calculated by finding the proportion of Y's variance due to the factor.

Variables are usually standardized before factor analyzing them in an EFA, which makes the factor's variance equal to one as well as standardizes the factor loadings, λ. The amount of Y's variance due to the factor is calculated by summing the standardized loadings and then squaring

[3] For information on calculating ω when there is more than one factor, including higher order factors, see Brunner, Nagy, and Wilhelm (2012). In such cases, there are two versions of ω: total ω and hierarchical ω. The `omega()` function in the *psych* package can estimate both versions of ω.

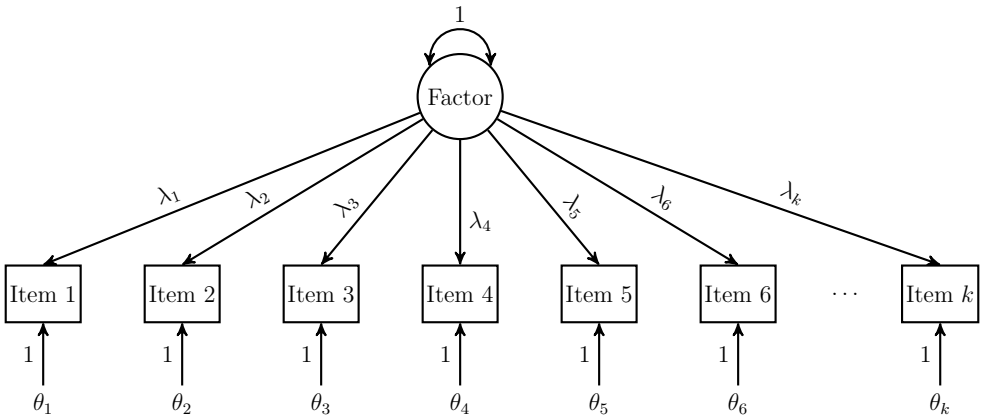


Figure 9.1 Single-factor/unidimensional model with standardized variables.

that sum

$$\left(\sum_{i=1}^{k} \lambda_i\right)^2$$

Likewise, the amount of the Y's variance due to error is calculated by summing all the error variances, θ_i,[4]

$$\sum_{i=1}^{k} \theta_i$$

Consequently,

$$\sigma_Y^2 = \left(\sum_{i=1}^{k} \lambda_i\right)^2 + \sum_{i=1}^{k} \theta_i$$

Thus,

$$\omega = \frac{\left(\sum_{i=1}^{k} \lambda_i\right)^2}{\sigma_Y^2} = \frac{\left(\sum_{i=1}^{k} \lambda_i\right)^2}{\left(\sum_{i=1}^{k} \lambda_i\right)^2 + \sum_{i=1}^{k} \theta_i}$$

To calculate ω in **R**, use the `type="omega"` argument in the `ci.reliability()` function.

```
# coefficient omega for SDQ parent data
ci.reliability(sdq.parent.data, type = "omega")


## $est
## [1] 0.84
##
## $se
## [1] 0.0021
##
## $ci.lower
## [1] 0.83
##
## $ci.upper
```

[4] In the single-factor model with standardized variables, $\theta_i = 1 - \lambda_i^2$.

```
## [1] 0.84
##
## $conf.level
## [1] 0.95
##
## $est.type
## [1] "omega"
##
## $analysis.type
## [1] "factor"
##
## $interval.type
## [1] "normal-theory"
##
## $call
## ci.reliability(data = sdq.parent.data, type = "omega")
##
## $boot
## NULL
```

```
# coefficient omega for GDS data
ci.reliability(gds.data, type = "omega")
```

```
## $est
## [1] 0.89
##
## $se
## [1] 0.0064
##
## $ci.lower
## [1] 0.87
##
## $ci.upper
## [1] 0.9
##
## $conf.level
## [1] 0.95
##
## $est.type
## [1] "omega"
##
## $analysis.type
## [1] "factor"
##
## $interval.type
## [1] "normal-theory"
##
## $call
## ci.reliability(data = gds.data, type = "omega")
##
## $boot
## NULL
```

References

Beaujean, A. A. (2013). Factor analysis using **R**. *Practical Assessment, Research, and Evaluation*, *18*(4), 1-11. Retrieved from http://pareonline.net/pdf/v18n4.pdf.

Beaujean, A. A. (2014). *Latent variable modeling using **R**: A step by step guide*. New York, NY: Routledge/Taylor and Francis.

Beaujean, A. A., Parkin, J., & Parker, S. (2014). Comparing Cattell-Horn-Carroll models for predicting language achievement: Differences between bi-factor and higher-order factor models. *Psychological Assessment*. doi: 10.1037/a0036745

R Development Core Team. (2014). ***R**: A language and environment for statistical computing* [Computer program]. Vienna, Austria: R Foundation for Statistical Computing.

Bock, R. D., Gibbons, R., & Muraki, E. (1988). Full-information item factor analysis. *Applied Psychological Measurement*, *12*, 261-280. doi: 10.1177/014662168801200305

Browne, M. W. (2001). An overview of analytic rotation in exploratory factor analysis. *Multivariate Behavioral Research*, *36*, 111-150. doi: 10.1207/s15327906mbr3601_05

Brunner, M., Nagy, G., & Wilhelm, O. (2012). A tutorial on hierarchically structured constructs. *Journal of Personality*, *80*, 796-846. doi: 10.1111/j.1467-6494.2011.00749.x

Canty, A. J. (2002). Resampling methods in **R**: The boot package. *R News*, *2/3*, 2-10. Retrieved from http://cran.r-project.org/doc/Rnews/Rnews_2002-3.pdf.

Carroll, J. B. (1993). *Human cognitive abilities: A survey of factor-analytic studies*. New York, NY: Cambridge University Press.

Cronbach, L. J. (1951). Coefficient alpha and the internal structure of tests. *Psychometrika*, *16*, 297-334. doi: 10.1007/bf02310555

Gorsuch, R. L. (1983). *Factor analysis* (2nd ed.). Hillsdale, NJ: Lawrence Erlbaum.

Guttman, L. (1945). A basis for analyzing test-retest reliability. *Psychometrika*, *10*, 255-282. doi: 10.1007/BF02288892

Herche, J., & Engelland, B. (1996). Reversed-polarity items and scale unidimensionality. *Journal of the Academy of Marketing Science*, *24*, 366-374. doi: 10.1177/0092070396244007

Holgado-Tello, F. P., Chacón-Moscoso, S., Barbero-García, I., & Vila-Abad, E. (2010). Polychoric versus Pearson correlations in exploratory and confirmatory factor analysis of ordinal variables. *Quality & Quantity*, *44*, 153-166. doi: 10.1007/s11135-008-9190-y

Holzinger, K., & Swineford, F. (1937). The bi-factor method. *Psychometrika*, *2*, 41-54. doi: 10.1007/BF02287965

Jennrich, R. I., & Bentler, P. (2011). Exploratory bi-factor analysis. *Psychometrika*, *76*, 537-549. doi: 10.1007/s11336-011-9218-4

Jennrich, R. I., & Bentler, P. M. (2012). Exploratory bi-factor analysis: The oblique case. *Psychometrika*, *77*, 442-454. doi: 10.1007/s11336-012-9269-1

Kelley, K. (2007). Methods for the Behavioral, Educational, and Social Sciences: An R package. *Behavior Research Methods*, *39*, 979-984. doi: 10.3758/BF03192993

Lai, K., & Kelley, K. (2011). Accuracy in parameter estimation for targeted effects in structural equation modeling: Sample size planning for narrow confidence intervals. *Psychological Methods*, *16*, 127-148. doi: 10.1037/a0021764

MacCallum, R. C., Browne, M. W., & Sugawara, H. M. (1996). Power analysis and determination of sample size for covariance structure modeling. *Psychological Methods*, *1*, 130-149. doi: 10.1037/1082-989X.1.2.130

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

Muthén, L. K., & Muthén, B. O. (2002). How to use a Monte Carlo study to decide on sample size and determine power. *Structural Equation Modeling: A Multidisciplinary Journal*, *9*, 599-620. doi: 10.1207/S15328007SEM0904_8

Osborne, J. W. (2014). *Best practices in exploratory factor analysis*. Louisville, NY: CreateSpace Independent Publishing Platform.

Revelle, W. (2014). *psych: Procedures for psychological, psychometric, and personality research* [Computer program]. Evanston, IL: Northwestern University.

Revelle, W., & Zinbarg, R. E. (2009). Coefficients alpha, beta, omega, and the glb: Comments on Sijtsma. *Psychometrika*, *74*, 145-154. doi: 10.1007/s11336-008-9102-z

Schmid, J., & Leiman, J. (1957). The development of hierarchical factor solutions. *Psychometrika*, *22*, 53-61. doi: 10.1007/bf02289209

Schmiedek, F., & Li, S.-C. (2004). Toward an alternative representation for disentangling age-associated differences in general and specific cognitive abilities. *Psychology and Aging*, *19*, 40-56. doi: 10.1037/0882-7974.19.1.40

Sliter, K. A., & Zickar, M. J. (2014). An IRT examination of the psychometric functioning of negatively worded personality items. *Educational and Psychological Measurement*, *74*, 214-226. doi: 10.1177/0013164413504584

Tanzer, N. K., Gittler, G., & Ellis, B. B. (1995). Cross-cultural validation of item complexity in a LLTM-calibrated spatial ability test. *European Journal of Psychological Assessment*, *11*, 170-183. doi: 10.1027/1015-5759.11.3.170

Venables, W. N., Smith, D. M., & R Development Core Team. (2014). *An introduction to **R***. R Development Core Team. Retrieved from http://cran.r-project.org/doc/manuals/R-intro.pdf.

Wirth, R. J., & Edwards, M. C. (2007). Item factor analysis: Current approaches and future directions. *Psychological Methods*, *12*, 58-79. doi: 10.1037/1082-989x.12.1.58

Yanyun, Y., & Green, S. B. (2011). Coefficient alpha: A reliability coefficient for the 21st century? *Journal of Psychoeducational Assessment*, *29*, 377-392. doi: 10.1177/0734282911406668