Getting Started with Mokken Scale Analysis in R

Andries van der Ark

July 9, 2010

Chapter 1

Getting started

1.1 Introduction

This report aims at researchers who have Windows installed on their computer and who wish to conduct Mokken scale analysis using the freeware R-package `mokken` (Van der Ark, 2007) but who do not know anything about R. It is a step by step guide from scratch to actually performing Mokken scale analysis. A more elaborate book on learning R for SPSS and SAS users is Muenchen (2008). The report is organized as follows. In this chapter (chap. 1), I discuss the preparations that are needed before `mokken` can be used. In section 1.2, I discuss the installation of R and all the necessary packages. In section 1.3, I discuss how SPSS, SAS, STATA, and Splus data sets should be converted to R. In section 1.4, I show a few R commands that come in handy for data manipulation (e.g., variable selection). In chapter 2, I explain `mokken`. In section 2.1, I give an overview of `mokken`'s most important commands (known as *functions* in R) illustrated by examples. In section 3, I explain the use of `mokken` by showing the code for most analyses in the book *Introduction to nonparametric item response theory* (Sijtsma & Molenaar, 2003).

1.2 Installation

1.2.1 What is R?

R (R Development Core Team, 2006) is a language and environment for statistical computing and graphics. It is something in between a statistical package such as SPSS, STATA, or SAS and a programming language such as C++, PASCAL, or FORTRAN. It has two big advantages: It is for free and it has open source. Because it is for free it is accessible for anyone at any time, and because it has an open source researchers can add *packages* to R. Currently, over 2100 packages are added — the package `mokken` is one of

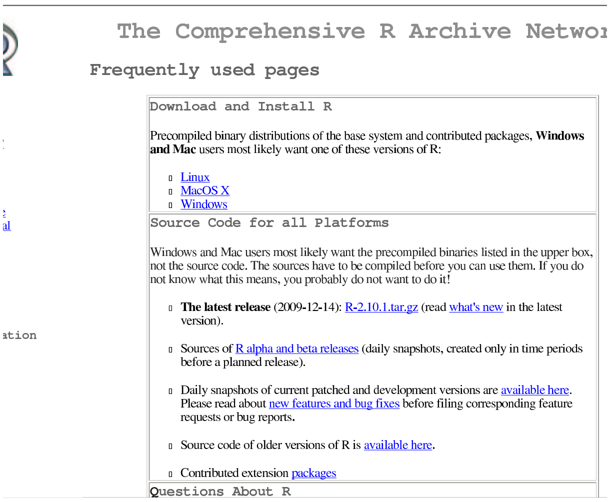


Figure 1.1: *R website. Click on Windows.*

them — allowing the user to conduct almost every possible statistical procedure ranging from common statistical procedures, which are also available in commercial software packages, to statistical techniques such as item response theory, spectral analysis, marginal modelling, Bayesian analysis, and latent class analysis. Every two months or so R releases a new version. At the time of writing version R-2.10.1 was the most recent version. Although, it is good to have a recent version of R, I only update a new version once a year or so. The major R website is `http://cran.r-project.org/`. It contains an abundance of information. A special page is devoted to packages that are of interest to psychometricians (`http://cran.r-project.org/web/views/Psychometrics.html`).

1.2.2 Installing R

Installing R requires the following steps

1. Go to `http://cran.r-project.org/`.
2. Click on Windows (See Figure 1.1)
3. Click on base (See Figure 1.2)
4. Click on Download R 2.10.1 for Windows (or a more recent version; see Figure 1.3)
5. Save the file `R-2.10.1-win32.exe` on your computer (e.g., on `C:/`).
6. Run the file `R-2.10.1-win32.exe` from your computer. You can choose all the default values in the installation Wizard.

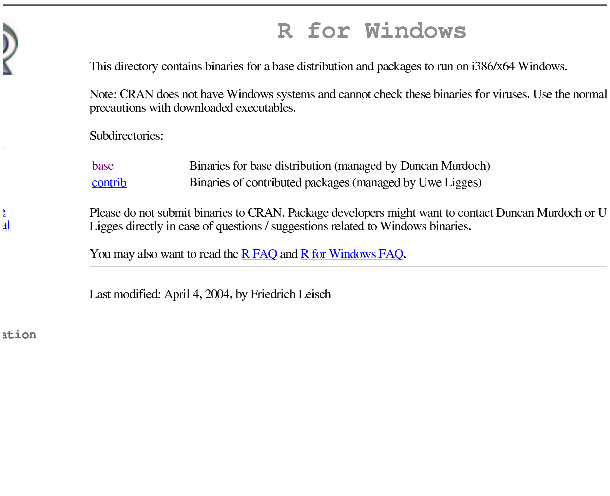


Figure 1.2: *R website. Click on base.*

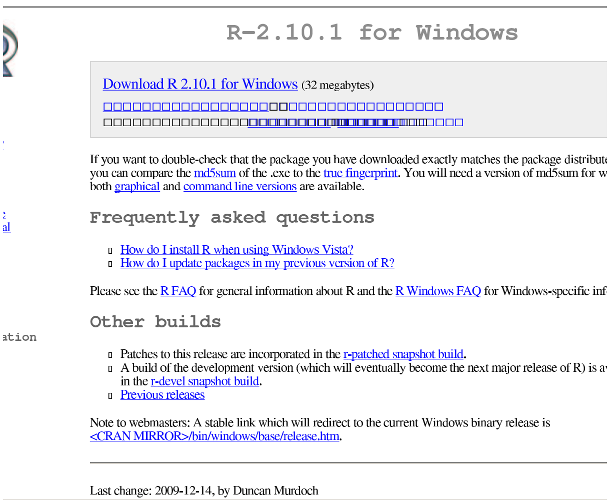


Figure 1.3: *R website. Click on Download R 2.10.1 for Windows.*

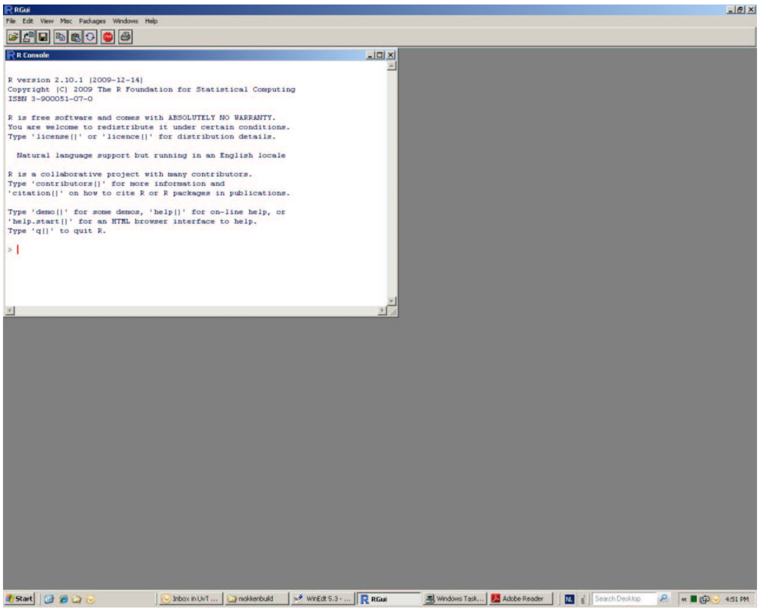


Figure 1.4: *R console.*

7. R will be available from the desk top icon, and from the programme's menu.

1.2.3 Installing the package `mokken`

1. Open R (Figure 1.4 shows the *R console*).

2. In the pull down menu choose `Packages`, `Install package(s)`, choose a location nearby you, and choose the package `mokken` (Figure 1.5).

The package `mokken` is now installed on your computer and need not be installed anymore. It may be noted that the same procedure applies to the installation of all packages.

1.2.4 Working with R

Except for loading packages (section 1.2.3), almost everything in R is conducted by typing (or pasting) code in the R console (Figure 1.4) just after the prompt (>) and end with hard return (Enter). Code to be typed is printed in *`slanted typewriter text`* and preceded by a > (a prompt). The resulting output is printed exactly as it is printed on the screen in `typewrite text`. Note that R is case sensitive. Some examples.

```
> 6 - 3
```

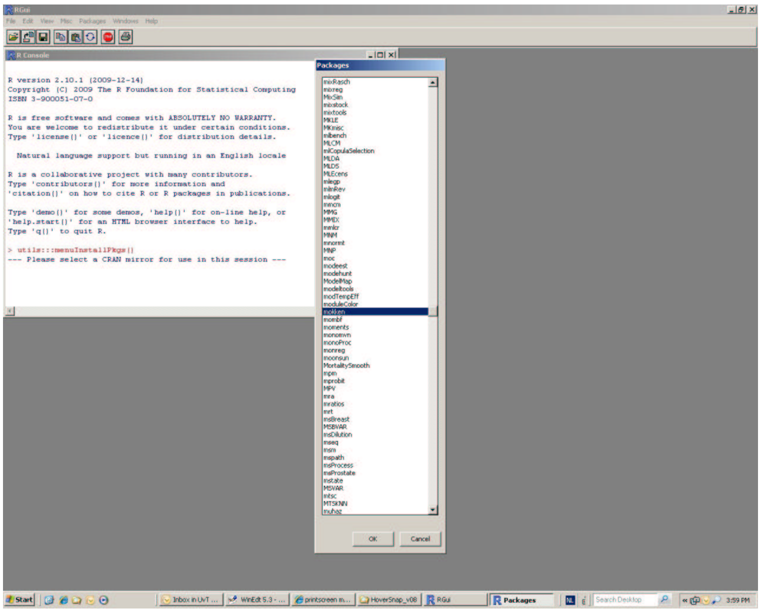


Figure 1.5: *R console*. Choose `mokken`.

```
[1] 3
```

```
> options(width = 60)
> x <- sqrt(2)
> x
```

```
[1] 1.414214
```

R assigns the value $\sqrt{2}$ to variable x (line 1) and displays the value of x (line 2)

To load the procedures of `mokken` into the memory of R type

```
> library(mokken)
```

To quit R, type

```
> q()
```

Free of charge introductions to R are available on the Internet.

- R Development Core Team (2009). *An Introduction to R*. Retrieved from http://cran.r-project.org/doc/manuals/R-intro.html (html) or http://cran.r-project.org/doc/manuals/R-intro.pdf (pdf).

- Paradis, E. (2005). *R for beginners*. Retrieved from http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf
- Baron, J., & Li, Y. (2007). *Notes on the use of R for psychology experiments and questionnaires*. Retrieved from http://www.psych.upenn.edu/~baron/rpsych/rpsych.html

Many more sources are available from http://cran.r-project.org/ (Contributed Documentation).

1.3 Converting data to R and back again

Converting a data set from a commercial package to R is the Achilles Heel of Mokken scale analysis in R. Commercial packages have no interest in free software that can easily read their data sets and these companies put no effort making their data files compatible with R. As a result, small things that you may not be aware of (e.g., whether your computer uses a point or a comma as a decimal separator, whether or not the rows in your data set have labels) may affect the conversion. An elaborated manual for converting many types of files in to files that can be read by R is available from http://cran.r-project.org/doc/manuals/R-data.html. Here only conversions to and from SPSS, SAS, STATA, and Splus are briefly discussed. The fasted strategy is to read the SPSS, SAS, STATA, or Splus file directly in R. Direct reading may occasionally go wrong and an alternative option is to save the SPSS, SAS, STATA, or Splus file as a text-only file (ASCII file), and read the ASCII file into R. In the latter procedure, the variable names may get lost.

1.3.1 SPSS files

Converting SPSS files directly

I assume that an SPSS data set named `ExampleSPSS.sav` has been saved on `C:/` [1] .

1. Type the following code in the R console

```
> library(foreign)
> ExampleR <- data.frame(read.spss("C:/ExampleSPSS.sav"))
> fix(ExampleR)
```

[1] `ExampleSPSS.sav` is a completely arbitrary name and your data set probably has a different name and may be located on another drive than `C:/` . Therefore, you should replace `C:/ExampleSPSS.sav` by your complete path and file name. The SPSS-file is not included in the Mokken package.

Note that data.frame() is an R function; it saves the data in a matrix-like manner, allowing different measurement levels for the scores in each column. Most data sets in R belong to the class data.frame. The data file is now stored in the memory of R under the name ExampleR[2]. The last command is not necessary. It opens the R data in a spread sheet in another window in R; the spreadsheet can be used to check whether the transformation went well. If necessary, the spread sheet may be modified. If the spreadsheet window is closed (by clicking the close button in the upper right-hand corner, see Figure 1.6) the changes are saved. Note that library(foreign) may be omitted, if it has been typed in before during the same R session.

2. If R is closed, ExampleR are lost. Therefore, the data should be saved in an R format that can be retrieved easily. To save the data (in the file C:/ExampleR.Rdata) type

   ```
   > save(ExampleR, file = "C:/ExampleR.Rdata")
   ```

   To get the data back into R type

   ```
   > load("C:/ExampleR.Rdata")
   ```

Saving SPSS files as ASCII files and read the ASCII files

Save the data as a tab delimited ASCII file (.dat file) This format can be read easily by R. The SPSS syntax is

```
SAVE TRANSLATE OUTFILE='C:\ExampleSPSS2.dat'
  /TYPE=TAB
  /MAP
  /REPLACE
  /FIELDNAMES
  /CELLS=LABELS.
```

Converting R data to SPSS

To convert R data sets to SPSS directly is impossible. R creates an SPSS syntax file and an ASCII data file. The SPSS syntax file should be run within SPSS. To create the syntax file "ExampleSPSS.SPS" and the data file "ExampleSSPS.txt" from the R data ExampleR, type

```
> library(foreign)
> write.foreign(ExampleR, datafile = "C:/ExampleSPSS.txt",
+     codefile = "C:/ExampleSPSS.SPS", package = "SPSS")
```

[2] Again ExampleR is a completely arbitrary name and you may decide to name it differently, for example, NKSPdata2008.

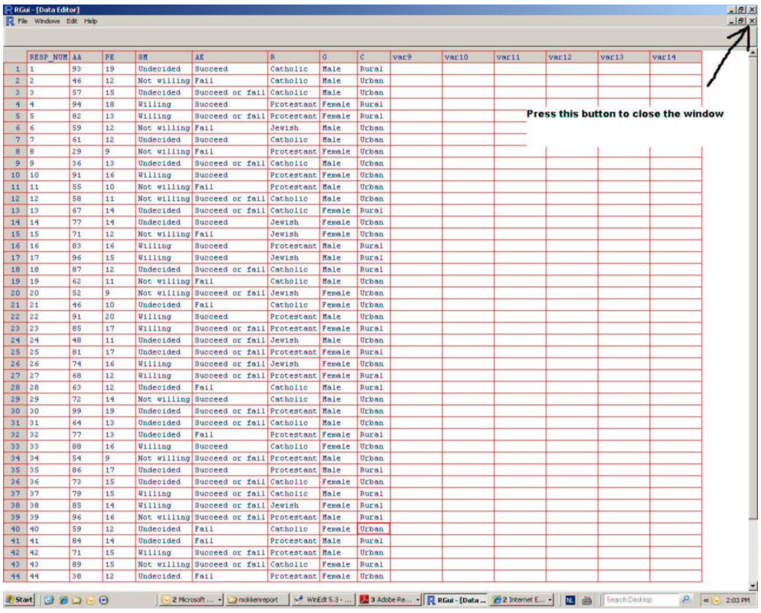


Figure 1.6: *Close the spreadsheet by clicking the button in the upper right-hand corner*

1.3.2 SAS XPORT files

Converting SAS XPORT files directly

I assume that the SAS data set `ExampleSAS.xpt` has been saved on `C:/`.

1. Type the following code in the R console

   ```
   > library(foreign)
   > ExampleR <- data.frame(read.xport("C:/ExampleSAS.xpt"))
   > fix(ExampleR)
   ```

   The data file is now stored in the memory of R under the name `ExampleR`. The last command is not necessary. It opens the R data in a spread sheet, which can be used to check whether the transformation went well. If necessary, the spread sheet may be modified. If the spread-sheet window is closed the changes are saved. Note that `library(foreign)` may be omitted, if it has been typed in before during the same R session.

2. If R is closed, `ExampleR` are lost. Therefore, the data should be saved in an R format that can be retrieved easily. To save the data (in the file `C:/ExampleR.Rdata`) type

```
> save(ExampleR, file = "C:/ExampleR.Rdata")
```

To get the data back into R type

```
> load("C:/ExampleR.Rdata")
```

Converting R data to SAS

To convert R data sets to SAS directly is impossible. R creates a SAS syntax file and an ASCII data file. The SAS syntax file should be run within SAS. To create the syntax file "ExampleSAS.XXX" and the data file "ExampleSAS.txt" from the R data ExampleR, type

```
> library(foreign)
> write.foreign(ExampleR, datafile = "C:/ExampleSAS.txt",
+     codefile = "C:/ExampleSAS.XXX", package = "SAS")
```

1.3.3 STATA files

Converting STATA files directly

I assume that the STATA data set ExampleSTATA.dta has been saved on C:/.

1. Type the following code in the R console

   ```
   > library(foreign)
   > ExampleR <- data.frame(read.dta("C:/ExampleSTATA.dta"))
   > fix(ExampleR)
   ```

   The data file is now stored in the memory of R under the name ExampleR. The last command is not necessary. It opens the R data in a spread sheet, which can be used to check whether the transformation went well. If necessary, the spread sheet may be modified. If the spread-sheet window is closed the changes are saved. Note that library(foreign) may be omitted, if it has been typed in before during the same R session.

2. If R is closed, ExampleR are lost. Therefore, the data should be saved in an R format that can be retrieved easily. To save the data (in the file C:/ExampleR.Rdata) type

   ```
   > save(ExampleR, file = "C:/ExampleR.Rdata")
   ```

   To get the data back into R type

   ```
   > load("C:/ExampleR.Rdata")
   ```

Converting R data to STATA

To convert R data sets to STATA directly is impossible. R creates a STATA syntax file and an ASCII data file. The STATA syntax file should be run within STATA. To create the syntax file `"ExampleSTATA.do"` and the data file `"ExampleSTATA.dat"` from the R data `ExampleR`, type

```
> library(foreign)
> write.foreign(ExampleR, datafile = "C:/ExampleSTATA.dat",
+     codefile = "C:/ExampleSTATA.do", package = "Stata")
```

1.3.4 Splus files

Converting Splus files directly

I assume that the Splus data set `ExampleSplus.ssc` has been saved on `C:/`.

1. Type the following code in the R console

   ```
   > library(foreign)
   > ExampleR <- data.frame(read.s("C:/ExampleSplus.ssc"))
   > fix(ExampleR)
   ```

   The data file is now stored in the memory of R under the name `ExampleR`. The last command is not necessary. It opens the R data in a spread sheet, which can be used to check whether the transformation went well. If necessary, the spread sheet may be modified. If the spread-sheet window is closed the changes are saved. Note that `library(foreign)` may be omitted, if it has been typed in before during the same R session.

2. If R is closed, `ExampleR` are lost. Therefore, the data should be saved in an R format that can be retrieved easily. To save the data (in the file `C:/ExampleR.Rdata`) type

   ```
   > save(ExampleR, file = "C:/ExampleR.Rdata")
   ```

   To get the data back into R type

   ```
   > load("C:/ExampleR.Rdata")
   ```

Converting Splus objects to R objects

I assume that the you have an Splus object `ExampleSplus` in Splus, and that all data can be stored in `C:/`. Type in the Splus console

```
> dump(ExampleSplus, "C:/Example.dmp")
```

Next, type in the R console

```
> ExampleR <- dget("C:/Example.dmp")
```

1.4 R commands required for `mokken`

Rather than typing commands in the R console, I advice to type the commands in a plain text file, save the file, and paste a command or a series of commands into R. In this way the commands will not be lost.

- If `mokken` is used, then one should start each R session with

  ```
  > library(mokken)
  ```

- If help is required at any stage use the command `help()`. For example,

  ```
  > help(mokken)
  ```

  The help file contains examples of `mokken`. It can be instructive to paste these examples into the R console.

- A hash (`#`) indicates that everything beyond it on the same line is a comment.

  does not do anything.

- There are three data sets included in `mokken`: `acl`, `cavalini`, and `transreas`. Typing

  ```
  > data(acl)
  > data(cavalini)
  > data(transreas)
  ```

  makes them available in R. Note that without these `data()` commands, the data sets are are not available.

  ```
  > help(acl)
  ```

  will give all the information on `acl`

  ```
  > fix(cavalini)
  ```

  will show `cavalini` in a spreadsheet.

- An arrow `<-` is used for assignment. Examples

```
> X <- acl
> Y <- 3
> Z <- c(1, 2, 3, 8:11)
```

The value of `X` is the data matric `acl` (`X` and `acl` are now equivalent). The value of `Y` is 3. The value of `Z` is the vector (1,2,3,8,9,10,11). It can be verified by typing

```
> Y

[1] 3

> Z

[1]  1  2  3  8  9 10 11
```

- To select columns and rows from the data matrix brackets are used.

```
> X1 <- acl[, 1]
```

`X1` are the scores on the first item 'Reliable')

```
> X2 <- acl[, 11:20]
```

`X2` are the scores on items 11 to 20 (i.e., only the scores on the 10 items of the scale 'Achievement')

```
> X3 <- acl[1:10, ]
```

`X3` are the scores of the first 10 respondents items on all items

```
> X4 <- acl[232, 133]
```

`X4` is the score of respondent 232 on item 133

```
> scale.1 <- c(1, 2, 4)
> X5 <- acl[c(1:100, 201:300), scale.1]
```

`X5` are the scores of respondents 1-100 and 201-300, on items 1, 2, and 4

```
> X6 <- acl[acl[, 1] == 2, ]
```

`X6` are the scores of those respondents who had a score 2 on item 1.

Note that in data matrices `X3` to `X6`, the cases (rows) not selected are thrown away, and case numbers are not available. Case numbers can be made through the following commands. If you want to identify the them, you can create case numbers for `acl`.

```
> dimnames(acl)[[1]] <- 1:nrow(acl)
```

If you repeat the analyses above, you may observe that the case numbers have been preserved.

Chapter 2

The R package mokken

2.1 An overview of the functions

The package mokken consists of the following functions

2.1.1 aisp

Function aisp performs Mokken's (1971) automated item selection algorithm. In the example, the scores on the first ten items from ACL are used; these are the items of the scale Communality. Mokken's automated item selection algorithm is applied to the ten items. The output (in blue) shows that items unscrupulous* and unintelligent* are unscalable, that items reliable, honest, deceitful*, and dependable are in scale 1, and items obnoxious*, thankless*, unfriendly*, and cruel* are in scale 2.

```
> data(acl)
> Communality <- acl[, 1:10]
> scale <- aisp(Communality, verbose = FALSE)
> scale
```

```
               Scale
reliable           1
honest             1
unscrupulous*      0
deceitful*         1
unintelligent*     0
obnoxious*         2
thankless*         2
unfriendly*        2
dependable         1
cruel*             2
```

Variations of aisp (output not shown) are the following

- Use a genetic algorithm (Straat, van der Ark, & Sijtsma, 2010) rather than Mokken's algorithm.

```
> scale <- aisp(Communality, search = "ga")
```

- Use different values for the lower bound (default `lowerbound = .3`) and or the nominal type I error rate (default `alpha = .05`)

```
> scale <- aisp(Communality, lowerbound = 0.2, alpha = 0.1)
```

- Output on the screen during the item selection (default `verbose = TRUE`)

```
> aisp(Communality, verbose = TRUE)
```

- For more information type

```
> help(aisp)
```

Note that `search = "extend"` has not yet been implemented.

2.1.2 coefH

Computes scalability coefficients H_{ij}, H_i, and H for a set of items. In the example, the scores on the first ten items from ACL are used; these are the items of the scale Communality. First, scalability coefficients H_{ij}, H_i, and H are computed (no output given here because it is rather voluminous). Second, only the item scalability coefficients are extracted. Third, the item scalability coefficients are extracted but rounded to two integers. Fourth, H is extracted.

```
> data(acl)
> Communality <- acl[, 1:10]
> coefficients <- coefH(Communality)
> coefficients$Hi

     reliable        honest  unscrupulous*     deceitful*
    0.3038656     0.2651096      0.2360455      0.3191367
unintelligent*    obnoxious*     thankless*    unfriendly*
    0.1160265     0.2879527      0.2454761      0.3085198
   dependable        cruel*
    0.2994265     0.2522276

> round(coefficients$Hi, 2)
```

```
     reliable         honest  unscrupulous*     deceitful*
         0.30           0.27           0.24           0.32
unintelligent*    obnoxious*     thankless*    unfriendly*
         0.12           0.29           0.25           0.31
   dependable         cruel*
         0.30           0.25
```

```
> coefficients$H
```

```
[1] 0.2635151
```

2.1.3 `check.iio`

Investigates invariant item ordering (IIO) using method *Manifest IIO* (MIIO; Ligtvoet, Van der Ark, Te Marvelde, & Sijtsma, 2010) and methods *Manifest Scale - Cumulative Probability Model* (MS-CPM) and *Increasingness in Transposition* (IT) (Ligtvoet, Van der Ark, Bergsma, & Sijtsma, 2010). Method Manifest IIO is the default. First, all result with respect to IIO are saved in `iio.results`. In the example, the scores on the first ten items from ACL are used; these are the items of the scale Communality. Simply typing `iio.results` produces a list with lots of output for each item. `summary()` reduces this output by giving a summary of the results. The output shows the method used (i.e., Manifest IIO), the violations of manifest IIO, the items selected using the backward selection algorithm, and scalability coefficient H^T for the final scale (items `unfriendly*` and `deceitful*` excluded).

```
> data(acl)
> Communality <- acl[, 1:10]
> iio.results <- check.iio(Communality)
> summary(iio.results)
```

```
$method
[1] "MIIO"

$item.summary
               mean #ac #vi #vi/#ac maxvi  sum sum/#ac tmax
cruel*         3.48  36   0    0.00  0.00 0.00    0.00 0.00
unintelligent* 3.32  35   2    0.06  0.15 0.29    0.01 2.17
unscrupulous*  3.32  35   1    0.03  0.14 0.14    0.00 1.21
unfriendly*    3.30  36   1    0.03  0.15 0.15    0.00 2.17
thankless*     3.26  36   1    0.03  0.12 0.12    0.00 1.50
dependable     3.25  36   0    0.00  0.00 0.00    0.00 0.00
obnoxious*     3.25  36   1    0.03  0.12 0.12    0.00 1.50
reliable       3.09  36   0    0.00  0.00 0.00    0.00 0.00
```

```
honest          3.02  36   2    0.06  0.18 0.31    0.01 2.08
deceitful*      2.94  34   2    0.06  0.18 0.31    0.01 2.08
                #tsig
cruel*              0
unintelligent*      1
unscrupulous*       0
unfriendly*         1
thankless*          0
dependable          0
obnoxious*          0
reliable            0
honest              1
deceitful*          1

$backward.selection
                step 1 step 2 step 3
cruel*               0      0      0
unintelligent*       1      1      0
unscrupulous*        0      0      0
unfriendly*          1      1     NA
thankless*           0      0      0
dependable           0      0      0
obnoxious*           0      0      0
reliable             0      0      0
honest               1      0      0
deceitful*           1     NA     NA

$HT
[1] 0.05468516
```

Variations of `check.iio` (output not shown) are the following.

- Other values for `minvi` and `minsize` (Molenaar & Sijtsma, 2000, pp. 45-46) .

```
> check.iio(Communality, minvi = 0, minsize = 50)
```

- Using methods MS-CPM and IT (Ligtvoet, Van der Ark, Bergsma, & Sijtsma, 2010)

```
> summary(check.iio(Communality, method = "MS-CPM"))
> summary(check.iio(Communality, method = "IT"))
```

- Different nominal Type I error rate for t-test (method MIIO), z-test (Method MS-CPM), and McNemar test (method IT).

```
> summary(check.iio(Communality, alpha = 0.01))
```

- Without backward selection algorithm, and with information screen

```
> summary(check.iio(Communality, item.selection = FALSE))
> summary(check.iio(Communality, verbose = TRUE))
```

- For more information type

```
> help(check.iio)
```

2.1.4 `check.monotonicity` (a.k.a. `check.single`)

Investigates the monotonicity assumption using the observable property manifest monotonicity (Molenaar & Sijtsma, 2000, pp. 70-77). In the example, the scores on the first ten items from ACL are used; these are the items of the scale Communality. First, all result with respect to manifest monotonicity are saved in `monotonicity.results`. Simply typing `monotonicity.results` produces a list with lots of output for each item. `summary()` and `plot()` reduce this output by giving a summary of the results and graphically displaying the estimated item (step) response functions, respectively. For interpretation of the output see Molenaar and Sijtsma (2000, chap. 6, chap. 7). Without further specifications `plot()` displays 10 graphs (1 for each item) in a separate R Window, and requires a hard return to go to the next graph. Figure 2.1.4 (p. 20) shows the 10 graphs.

```
> data(acl)
> Communality <- acl[, 1:10]
> monotonicity.results <- check.monotonicity(Communality)
> summary(monotonicity.results)
> plot(monotonicity.results, items = c(1, 2))
```

```
               ItemH #ac #vi #vi/#ac maxvi  sum sum/#ac
reliable        0.30  24   0    0.00  0.00 0.00       0
honest          0.27  24   0    0.00  0.00 0.00       0
unscrupulous*   0.24  24   0    0.00  0.00 0.00       0
deceitful*      0.32  24   0    0.00  0.00 0.00       0
unintelligent*  0.12  24   1    0.04  0.07 0.07       0
obnoxious*      0.29  24   0    0.00  0.00 0.00       0
thankless*      0.25  24   0    0.00  0.00 0.00       0
unfriendly*     0.31  24   0    0.00  0.00 0.00       0
dependable      0.30  24   0    0.00  0.00 0.00       0
cruel*          0.25  24   0    0.00  0.00 0.00       0
               zmax #zsig
reliable       0.00     0
```

```
honest          0.00     0
unscrupulous*   0.00     0
deceitful*      0.00     0
unintelligent*  0.85     0
obnoxious*      0.00     0
thankless*      0.00     0
unfriendly*     0.00     0
dependable      0.00     0
cruel*          0.00     0
```

Variations of `check.monotonicity` (output not shown) are the following.

- Other values for `minvi` and `minsize` (Molenaar & Sijtsma, 2000, pp. 45-46) .

```
> check.monotonicity(Communality, minvi = 0, minsize = 50)
```

- Plot the results for items 1 and 2 only

```
> plot(check.monotonicity(Communality), item = c(1,
+     2))
```

- Save graphs in a pdf file. `ask=FALSE` assures that no hard return is required between subsequent graphs. The functions `pdf()` and `dev.off()` are not part of `mokken`.

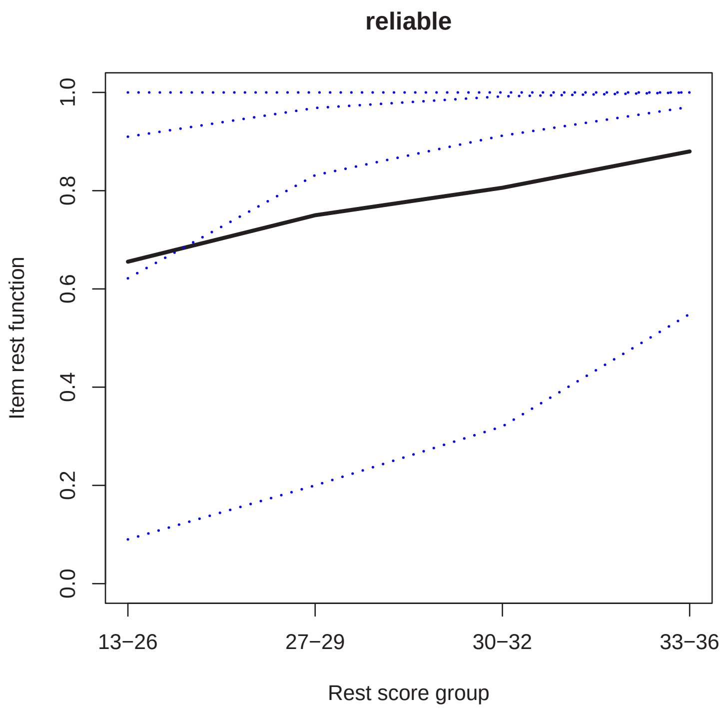
```
> pdf("monotonicity.pdf")
> plot(monotonicity.results, ask = FALSE)
> dev.off()
```

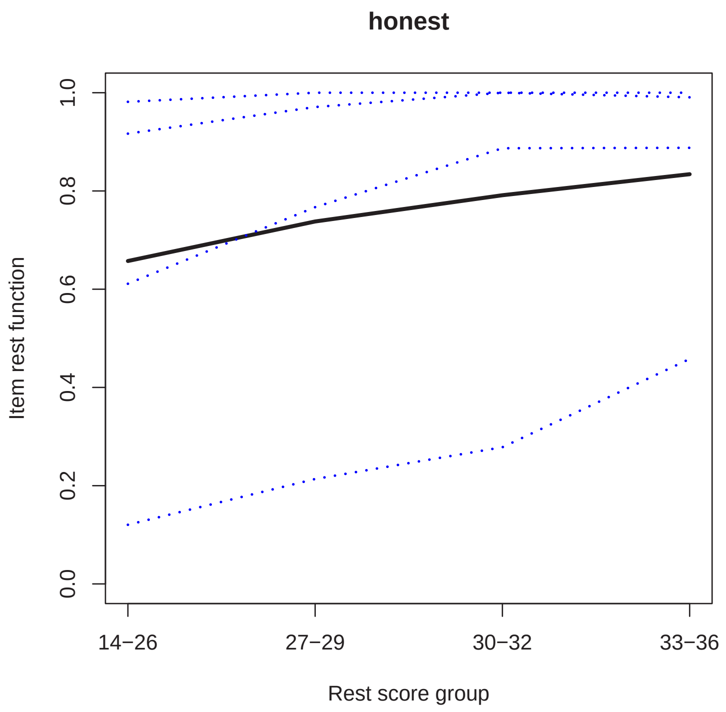
- For more information type

```
> help(check.monotonicity)
```

2.1.5 `check.pmatrix`

Investigates the assumption of nonintersecting item step response functions using the P++ and P−− matrix (Molenaar & Sijtsma, 2000, pp. 80-85). In the example, the scores on the first ten items from ACL are used; these are the items of the scale Communality. First, all result with respect to the P++ (indicted by `ppp`) and P−− (indicated by `pmm`) matrix are saved in `pmatrix.results`. Simply typing `pmatrix.results` produces a list with lots of output for each item. `summary()` and `plot()` reduce this output by giving a summary of the results and graphically displaying the estimated item (step) response functions, respectively. For interpretation of the output


reliable
Item rest function
0.0
0.2
0.4
0.6
0.8
1.0
13−26
27−29
30−32
33−36
Rest score group


honest
Item rest function
0.0
0.2
0.4
0.6
0.8
1.0
14−26
27−29
30−32
33−36
Rest score group

see Molenaar and Sijtsma (2000, pp. 80-85). Without further specifications `plot()` (no output shown) displays 20 graphs (2 for each item) in a separate R Window, and requires a hard return to go to the next graph.

```
> data(acl)
> Communality <- acl[, 1:10]
> pmatrix.results <- check.pmatrix(Communality)
> summary(pmatrix.results)
> plot(pmatrix.results)
```

```
> pmatrix.results <- check.pmatrix(Communality)
> summary(pmatrix.results)
```

```
$ppp.summary.matrix
               ItemH #ac #vi #vi/#ac maxvi  sum sum/#ac
reliable        0.30 144   6   0.001  0.08 0.39   0.003
honest          0.27 144  11   0.000  0.05 0.43   0.003
unscrupulous*   0.24 144   0   0.000  0.00 0.00   0.000
deceitful*      0.32 144   0   0.000  0.00 0.00   0.000
unintelligent*  0.12 144   0   0.000  0.00 0.00   0.000
obnoxious*      0.29 144   0   0.000  0.00 0.00   0.000
thankless*      0.25 144   1   0.000  0.05 0.05   0.000
unfriendly*     0.31 144   1   0.000  0.04 0.04   0.000
dependable      0.30 144  10   0.000  0.05 0.35   0.002
cruel*          0.25 144   1   0.000  0.03 0.03   0.000

$pmm.summary.matrix
               ItemH #ac #vi #vi/#ac maxvi  sum sum/#ac
reliable        0.30 144  10       0  0.07 0.44   0.003
honest          0.27 144   6       0  0.05 0.23   0.002
unscrupulous*   0.24 144   0       0  0.00 0.00   0.000
deceitful*      0.32 144   0       0  0.00 0.00   0.000
unintelligent*  0.12 144   0       0  0.00 0.00   0.000
obnoxious*      0.29 144   0       0  0.00 0.00   0.000
thankless*      0.25 144   0       0  0.00 0.00   0.000
unfriendly*     0.31 144   0       0  0.00 0.00   0.000
dependable      0.30 144   5       0  0.05 0.21   0.001
cruel*          0.25 144   3       0  0.03 0.10   0.001
```

Variations of `check.pmatrix` are the following.

- Other values for `minvi` (Molenaar & Sijtsma, 2000, pp. 45-46) .

  ```
  > check.pmatrix(Communality, minvi = 0)
  ```

- Plot the results for P++, for items 1 and 2 only, and plot the results for P−− for item 5.

```
> plot(check.pmatrix(Communality), pmatrix = "ppp",
+     item = c(1, 2))
> plot(check.pmatrix(Communality), pmatrix = "pmm",
+     item = 5)
```

- Save graphs in a pdf file. `ask=FALSE` assures that no hard return is required between subsequent graphs. The functions `pdf()` and `dev.off()` are not part of `mokken`.

```
> pdf("pmatrix.pdf")
> plot(pmatrix.results, ask = FALSE)
> dev.off()
```

- For more information type

```
> help(check.pmatrix)
```

2.1.6 `check.reliability`

Computes reliability coefficients ρ (a.k.a., the MS statistic; Molenaar & Sijtsma, 1984, 1988; Sijtsma & Molenaar, 1987; Van der Ark, 2010), Cronbach's (1951) alpha, and lambda-2 (Guttman, 1945). In the example, the scores on the first ten items from ACL are used; these are the items of the scale Communality.

```
> data(acl)
> Communality <- acl[, 1:10]
> check.reliability(Communality)

$MS
[1] 0.75766

$alpha
[1] 0.7465871

$lambda.2
[1] 0.7568063
```

2.1.7 `check.restscore`

Investigates the assumption of nonintersecting item step response functions using method restscore (Molenaar & Sijtsma, 2000, pp. 77-80). In the example, the scores on the first ten items from ACL are used; these are the items of the scale Communality. First, all result with respect to method restscore are saved in `restscore.results`. Simply typing `restscore.results` produces a list with lots of output for each item pair. `summary()` and `plot()` reduce this output by giving a summary of the results and plotting the estimated item (step) response functions, respectively. For interpretation of the output see Molenaar and Sijtsma (2000, pp. 77-80). Without further specifications `plot()` displays $\frac{1}{2} \times 10 \times 9 = 45$ graphs (1 for each item pair) in a separate R Window, and requires a hard return to go to the next graph. Figure 2.1.7 (p. 24) shows the rest score plots for the first two item pairs.

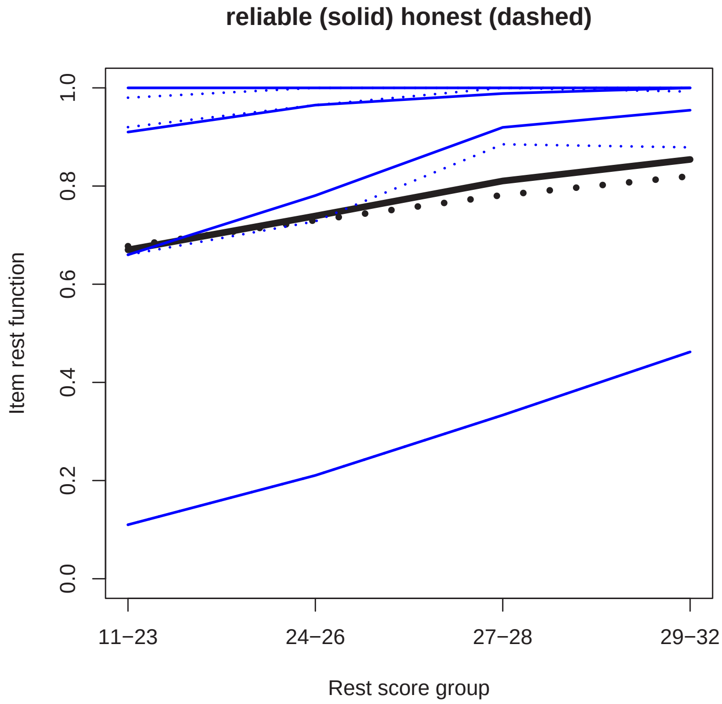
```
> data(acl)
> Communality <- acl[, 1:10]
> restscore.results <- check.restscore(Communality)
> summary(restscore.results)
> plot(restscore.results, item.pairs = c(1, 2))
```

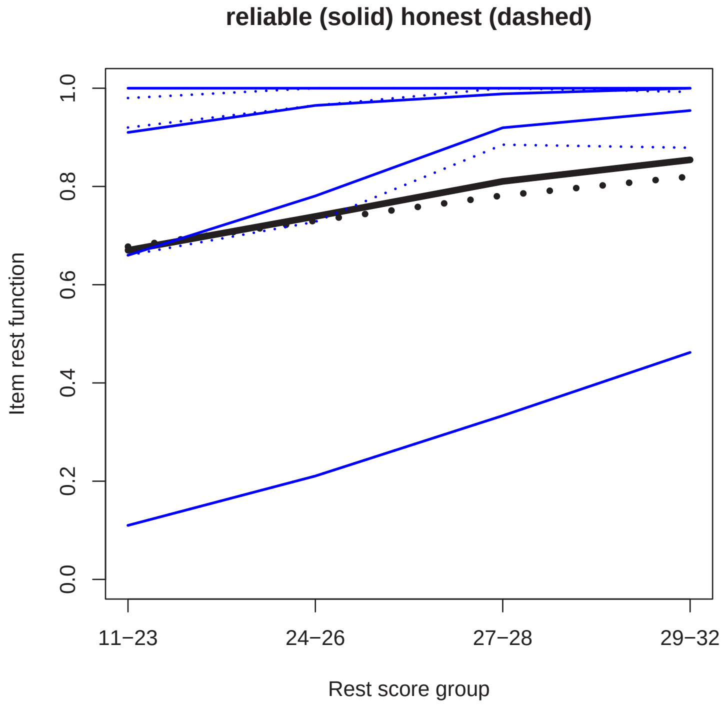
```
               ItemH #ac #vi #vi/#ac maxvi  sum sum/#ac
reliable        0.30 432   7    0.02  0.09 0.31       0
honest          0.27 432   5    0.01  0.07 0.25       0
unscrupulous*   0.24 416   6    0.01  0.11 0.42       0
deceitful*      0.32 400   8    0.02  0.09 0.40       0
unintelligent*  0.12 416  14    0.03  0.11 0.86       0
obnoxious*      0.29 432   7    0.02  0.11 0.52       0
thankless*      0.25 432   7    0.02  0.08 0.38       0
unfriendly*     0.31 432   8    0.02  0.11 0.48       0
dependable      0.30 432   9    0.02  0.09 0.48       0
cruel*          0.25 432   3    0.01  0.04 0.12       0
               zmax #zsig
reliable       1.43     0
honest         1.25     0
unscrupulous*  1.46     0
deceitful*     1.22     0
unintelligent* 1.99     2
obnoxious*     1.83     1
thankless*     1.14     0
unfriendly*    1.99     1
dependable     1.22     0
cruel*         0.67     0
```

Variations of `check.restscore` are the following.

- Other values for `minvi` and `minsize` (Molenaar & Sijtsma, 2000, pp. 45-46) .

  ```
  > check.restscore(Communality, minvi = 0, minsize = 50)
  ```


reliable (solid) honest (dashed)
Item rest function
0.0
0.2
0.4
0.6
0.8
1.0
11−23
24−26
27−28
29−32
Rest score group


reliable (solid) honest (dashed)
Item rest function
0.0
0.2
0.4
0.6
0.8
1.0
11−23
24−26
27−28
29−32
Rest score group

- Plot the results for all item pairs.

```
> plot(check.restscore(Communality))
```

- Save graphs in a pdf file. `ask=FALSE` assures that no hard return is required between subsequent graphs. The functions `pdf()` and `dev.off()` are not part of `mokken`.

```
> pdf("restscore.pdf")
> plot(restscore.results, ask = FALSE)
> dev.off()
```

- For more information type

```
> help(check.restscore)
```

2.1.8 check.groups

The package `mokken` does not yet have a function `check.groups`, which—in analogy to the function`CHECK=GROUPS` in the software program MSP (Molenaar & Sijtsma, 2000, pp. 85-88)—may have been expected.

Some analyses can be conducted using standard R commands. The example shows how scalability coefficient H is computed for the first ten items from ACL, constituting the scale Communality, for respondents having scores, 0 or 1, 2, 3, and 4, respectively, on item 11 (Achievement). Also, see section 1.4.

```
> data(acl)
> Communality <- acl[, 1:10]
> Group <- acl[, 11]
> coefH(Communality[Group == 0 | Group == 1, ])$H

[1] 0.1963215

> coefH(Communality[Group == 2, ])$H

[1] 0.3038342

> coefH(Communality[Group == 3, ])$H

[1] 0.2569371

> coefH(Communality[Group == 4, ])$H

[1] 0.2465098
```

Chapter 3

Examples of Mokken scale analysis in R

This chapter shows the code for producing the tables in Sijtsma and Molenaar (2003).

3.1 Table 3.1

Get the transitive reasoning data, and split them into the grades (first column of the data matrix), and the items scores (the remaining columns in the data matrix).

```
> library(mokken)
> data(transreas)
> grades <- transreas[, 1]
> item.scores <- transreas[, -1]
```

Obtaining the overall mean scores, and the mean scores per grade

```
> apply(item.scores, 2, mean)

     T09L      T12P      T10W      T11P      T04W      T05W
0.3011765 0.4752941 0.5200000 0.6423529 0.7835294 0.8023529
     T02L      T07L      T03W      T01L      T08W      T06A
0.8094118 0.8447059 0.8847059 0.9411765 0.9670588 0.9741176
```

```
> apply(item.scores[grades == 2, ], 2, mean)

     T09L      T12P      T10W      T11P      T04W      T05W
0.2093023 0.6627907 0.4069767 0.4883721 0.8372093 0.7093023
     T02L      T07L      T03W      T01L      T08W      T06A
0.8139535 0.7209302 0.8023256 0.8488372 0.9302326 0.9534884
```

```
> apply(item.scores[grades == 3, ], 2, mean)

     T09L      T12P      T10W      T11P      T04W      T05W
0.3294118 0.5058824 0.5764706 0.6000000 0.6470588 0.8000000
     T02L      T07L      T03W      T01L      T08W      T06A
0.7294118 0.8352941 0.9294118 0.9882353 0.9647059 0.9529412

> apply(item.scores[grades == 4, ], 2, mean)

     T09L      T12P      T10W      T11P      T04W      T05W
0.3048780 0.3902439 0.3902439 0.6707317 0.8170732 0.7804878
     T02L      T07L      T03W      T01L      T08W      T06A
0.8902439 0.8658537 0.9146341 0.9756098 0.9756098 0.9878049

> apply(item.scores[grades == 5, ], 2, mean)

     T09L      T12P      T10W      T11P      T04W      T05W
0.2588235 0.5176471 0.5529412 0.7058824 0.8235294 0.8235294
     T02L      T07L      T03W      T01L      T08W      T06A
0.7882353 0.8705882 0.8470588 0.9294118 0.9764706 0.9764706

> apply(item.scores[grades == 6, ], 2, mean)

     T09L      T12P      T10W      T11P      T04W      T05W
0.4022989 0.2988506 0.6666667 0.7471264 0.7931034 0.8965517
     T02L      T07L      T03W      T01L      T08W      T06A
0.8275862 0.9310345 0.9310345 0.9655172 0.9885057 1.0000000
```

Construction of Table 3.1 (advanced R code).

```
> Total.group <- round(apply(item.scores, 2, mean),
+     2)
> for (i in 2:6) assign(paste("Grade.", i, sep = ""),
+     round(apply(item.scores[grades == i, ], 2,
+         mean), 2))
> Task <- c(9, 12, 10, 11, 4, 5, 2, 7, 3, 1, 8,
+     6)
> Property <- attributes(transreas)$property
> Format <- attributes(transreas)$format
> Objects <- attributes(transreas)$objects
> Measures <- attributes(transreas)$measures
> Table.3.1 <- data.frame(Task, Property, Format,
+     Objects, Measures, Total.group, Grade.2, Grade.3,
+     Grade.4, Grade.5, Grade.6)
> Table.3.1
```

	Task	Property	Format	Objects	Measures	Total.group	Grade.2	Grade.3	Grade.4	Grade.5	Grade.6
T09L	9	length	YA = YB < YC = YD	sticks	12.5, 12.5, 13, 13 (cm)	0.30	0.21	0.33	0.30	0.26	0.40
T12P	12	pseudo				0.48	0.66	0.51	0.39	0.52	0.30
T10W	10	weight	YA = YB < YC = YD	balls	60, 60, 100, 100 (g)	0.52	0.41	0.58	0.39	0.55	0.67
T11P	11	pseudo				0.64	0.49	0.60	0.67	0.71	0.75
T04W	4	weight	YA = YB = YC = YD	cubes	65 (g)	0.78	0.84	0.65	0.82	0.82	0.79
T05W	5	weight	YA < YB < YC	balls	40, 50, 70 (cm)	0.80	0.71	0.80	0.78	0.82	0.90
T02L	2	length	YA = YB = YC = YD	tubes	12 (cm)	0.81	0.81	0.73	0.89	0.79	0.83
T07L	7	length	YA > YB = YC	sticks	28.5, 27.5, 27.5 (cm)	0.84	0.72	0.84	0.87	0.87	0.93
T03W	3	weight	YA > YB > YC	tubes	45, 25, 18 (g)	0.88	0.80	0.93	0.91	0.85	0.93
T01L	1	length	YA > YB > YC	sticks	12, 11.5, 11 (cm)	0.94	0.85	0.99	0.98	0.93	0.97
T08W	8	weight	YA > YB = YC	balls	65, 40, 40 (g)	0.97	0.93	0.96	0.98	0.98	0.99
T06A	6	area	YA > YB > YC	discs	7.5, 7, 6.5 (diameter; cm)	0.97	0.95	0.95	0.99	0.98	1.00

3.2 Table 3.2

To get the data, see Table 3.1.

Obtain scalability coefficients and Z coefficients for items and total scale.

```
> coefH(item.scores)$Hi

       T09L        T12P        T10W        T11P        T04W
 0.17075112 -0.13516424  0.13762914 -0.02534203  0.04791651
       T05W        T02L        T07L        T03W        T01L
 0.09166970  0.08126425  0.17766194  0.18791305  0.20624790
       T08W        T06A
 0.28245200  0.39559408

> coefH(item.scores)$H

[1] 0.090514

> coefZ(item.scores)$Zi

     T09L      T12P      T10W      T11P      T04W      T05W
 4.357153 -4.650737  4.949901 -0.967124  1.948629  3.746486
     T02L      T07L      T03W      T01L      T08W      T06A
 3.307606  6.868759  6.582933  5.684119  6.326821  8.019603

> coefZ(item.scores)$Z

[1] 7.395366
```

Obtain scalability coefficients and Z coefficients for items and total scale, when the pseudo items (2 and 4) are deleted

```
> coefH(item.scores[, -c(2, 4)])$Hi

      T09L       T10W       T04W       T05W       T02L
0.28926218 0.28083167 0.03625403 0.13900577 0.07160711
      T07L       T03W       T01L       T08W       T06A
0.24977935 0.29285894 0.28657074 0.38746871 0.48272634

> coefH(item.scores[, -c(2, 4)])$H

[1] 0.2048305

> coefZ(item.scores[, -c(2, 4)])$Zi

    T09L     T10W     T04W     T05W     T02L     T07L
5.548097 7.506324 1.347390 5.285802 2.725229 9.192316
    T03W     T01L     T08W     T06A
9.922119 7.821882 8.732480 9.881951
```

```
> coefZ(item.scores[, -c(2, 4)])$Z

[1] 13.54854
```

Construction of Table 3.2 (advanced R code).

```
> Task <- c("9", "12", "10", "11", "4", "5", "2",
+     "7", "3", "1", "8", "6", "Total item set")
> Property <- c(attributes(transreas)$property,
+     "")
> Format <- c(attributes(transreas)$format, "")
> Table.3.2 <- data.frame(Task, Property, Format,
+     matrix(NA, 13, 8))
> analysis <- list(c(1:12), c(1, 3, 5:12), c(1,
+     3, 6, 8:12), c(1, 3, 8:12))
> k <- 3
> for (i in 1:4) for (j in 1:2) {
+     k <- k + 1
+     Table.3.2[c(analysis[[i]], 13), k] <- c(round(coefH(item.scores[,
+         analysis[[i]]])$Hi, 2), round(coefH(item.scores[,
+         analysis[[i]]])$H, 2))
+ }
> dimnames(Table.3.2)[[2]][4:11] <- paste(c("k=12",
+     "k=12", "k=10", "k=10", "k=8", "k=8", "k=7",
+     "k=7"), c("Hi", "Zi"))
> Table.3.2
```

```
             Task Property            Format k=12 Hi
1               9   length YA = YB < YC = YD    0.17
2              12   pseudo                     -0.14
3              10   weight YA = YB < YC = YD    0.14
4              11   pseudo                     -0.03
5               4   weight YA = YB = YC = YD    0.05
6               5   weight      YA < YB < YC    0.09
7               2   length YA = YB = YC = YD    0.08
8               7   length      YA > YB = YC    0.18
9               3   weight      YA > YB > YC    0.19
10              1   length      YA > YB > YC    0.21
11              8   weight      YA > YB = YC    0.28
12              6     area      YA > YB > YC    0.40
13 Total item set                               0.09
   k=12 Zi k=10 Hi k=10 Zi k=8 Hi k=8 Zi k=7 Hi k=7 Zi
1     0.17    0.29    0.29   0.44   0.44   0.50   0.50
2    -0.14      NA      NA     NA     NA     NA     NA
```

```
3      0.14    0.28    0.28   0.46   0.46   0.52   0.52
4     -0.03      NA      NA     NA     NA     NA     NA
5      0.05    0.04    0.04     NA     NA     NA     NA
6      0.09    0.14    0.14   0.20   0.20     NA     NA
7      0.08    0.07    0.07     NA     NA     NA     NA
8      0.18    0.25    0.25   0.39   0.39   0.51   0.51
9      0.19    0.29    0.29   0.45   0.45   0.53   0.53
10     0.21    0.29    0.29   0.42   0.42   0.46   0.46
11     0.28    0.39    0.39   0.51   0.51   0.55   0.55
12     0.40    0.48    0.48   0.57   0.57   0.59   0.59
13     0.09    0.20    0.20   0.40   0.40   0.52   0.52
```

3.3 Table 5.1

To get the data, see Table 3.1.

Automated item selection algorithm

```
> options(width = 60)
> scale <- aisp(item.scores, verbose = FALSE)
```

Construction of Table 5.1 (advanced R code).

```
> options(width = 60)
> scale.1 <- c(12, 8, 1, 11, 9, 3, 10)
> scale.2 <- c(7, 5)
> Hi.top <- matrix(NA, 8, 6)
> for (i in 1:6) Hi.top[1:(i + 1), i] <- round(coefH(item.scores[,
+     scale.1[1:(i + 1)]])$Hi, 2)
> for (i in 1:6) Hi.top[8, i] <- round(coefH(item.scores[,
+     scale.1[1:(i + 1)]])$H, 2)
> dimnames(Hi.top)[[2]] <- paste("Step", 1:6)
> Table.5.1.top <- data.frame(Task = c(Task[scale.1],
+     "Total H"), Property = c(Property[scale.1],
+     ""), Format = c(Format[scale.1], ""), Pi = c(round(apply(item.scores[,
+     scale.1], 2, mean), 2), NA))
> Table.5.1.top <- cbind(Table.5.1.top, Hi.top)
> Table.5.1.top
```

```
        Task Property            Format   Pi Step 1 Step 2
T06A       6     area      YA > YB > YC 0.97   0.78   0.76
T07L       7   length      YA > YB = YC 0.84   0.78   0.59
T09L       9   length YA = YB < YC = YD 0.30     NA   0.53
T08W       8   weight      YA > YB = YC 0.97     NA     NA
```

```
T03W       3   weight      YA > YB > YC 0.88     NA     NA
T10W      10   weight YA = YB < YC = YD 0.52     NA     NA
T01L       1   length      YA > YB > YC 0.94     NA     NA
     Total H                             NA   0.78   0.60
     Step 3 Step 4 Step 5 Step 6
T06A   0.66   0.64   0.64   0.59
T07L   0.56   0.50   0.51   0.51
T09L   0.60   0.57   0.51   0.50
T08W   0.59   0.62   0.61   0.55
T03W     NA   0.51   0.53   0.53
T10W     NA     NA   0.52   0.52
T01L     NA     NA     NA   0.46
       0.59   0.55   0.53   0.52
```

3.4 Table 5.2

Get the data, and dichotomize the scores, compute the P-values

```
> data(cavalini)
> X <- cavalini
> X[cavalini < 2] <- 0
> X[cavalini > 1] <- 1
> apply(X, 2, mean)
```

```
     Item1      Item2      Item3      Item4      Item5
0.60024155 0.43478261 0.59057971 0.42632850 0.20289855
     Item6      Item7      Item8      Item9     Item10
0.08937198 0.04710145 0.16062802 0.06280193 0.51932367
    Item11     Item12     Item13     Item14     Item15
0.21618357 0.28623188 0.13647343 0.22342995 0.16304348
    Item16     Item17
0.16666667 0.68599034
```

Make the table (advanced R code)

```
> Table.5.2 <- data.frame(1:17, attributes(X)$labels,
+     round(apply(X, 2, mean), 2))
> dimnames(Table.5.2)[[2]] <- c("Item.number", "Item.text",
+     "Pi")
> rownames(Table.5.2) <- NULL
> Table.5.2
```

```
   Item.number                        Item.text   Pi
1            1            Keep windows closed 0.60
```

```
2            2           no laundry outside 0.43
3            3     search source of malodour 0.59
4            4          no blankets outside 0.43
5            5     try to find out solutions 0.20
6            6     go elsewhere to fresh air 0.09
7            7       call environment agency 0.05
8            8       think of something else 0.16
9            9    file complaint at producer 0.06
10          10  acquiesce in odour annoyance 0.52
11          11 do something to get rid of it 0.22
12          12 say: it might have been worse 0.29
13          13             experience unrest 0.14
14          14    talk to friends and family 0.22
15          15                seek diversion 0.16
16          16          avoid nose breathing 0.17
17          17     try to adapt to situation 0.69
```

3.5 Table 5.3

Get the data, and dichotomize the scores, see previous example

Automated item selection algorithm with different values for the lower bound.

```
> aisp(X, lowerbound = 0, verbose = FALSE)
> aisp(X, lowerbound = 0.05, verbose = FALSE)
> aisp(X, lowerbound = 0.1, verbose = FALSE)
```

Make the table (advanced R code)

```
> lower.bound <- seq(0, 0.6, by = 0.05)
> scaling.results <- matrix(NA, length(lower.bound),
+     ncol(X))
> for (i in 1:length(lower.bound)) scaling.results[i,
+     ] <- aisp(X, lowerbound = lower.bound[i],
+     verbose = FALSE)
> equal <- function(x, n) which(x == n)
> scale.1 <- sapply(apply(scaling.results, 1, "equal",
+     1), paste, collapse = " ")
> scale.2 <- sapply(apply(scaling.results, 1, "equal",
+     2), paste, collapse = " ")
> scale.3 <- sapply(apply(scaling.results, 1, "equal",
+     3), paste, collapse = " ")
> scale.4 <- sapply(apply(scaling.results, 1, "equal",
```

```
+     4), paste, collapse = " ")
> scale.5 <- sapply(apply(scaling.results, 1, "equal",
+     5), paste, collapse = " ")
> Table.5.3 <- data.frame(lower.bound, scale.1,
+     scale.2, scale.3, scale.4, scale.5)
> Table.5.3
```

```
   lower.bound                          scale.1       scale.2
1         0.00 1 2 3 4 5 6 7 8 9 11 13 14   10 12 15 17
2         0.05 1 2 3 4 5 6 7 8 9 11 13 14   10 12 15 17
3         0.10   1 2 3 4 5 6 7 9 11 13 14 8 10 12 15 17
4         0.15   1 2 3 4 5 6 7 9 11 13 14 8 10 12 15 17
5         0.20   1 2 3 4 5 6 7 9 11 13 14 8 10 12 15 17
6         0.25       1 3 5 6 7 9 11 13 14           2 4
7         0.30              3 5 6 7 9 11      1 2 4 13
8         0.35                3 5 7 9 11         1 2 4
9         0.40                3 5 7 9 11         1 2 4
10        0.45                3 5 7 9 11         1 2 4
11        0.50                3 5 7 9 11         1 2 4
12        0.55                    7 9 11           2 4
13        0.60                      9 11         3 5 7
         scale.3 scale.4 scale.5
1
2
3
4
5
6  8 10 12 15 17
7        8 15 17   10 12
8        8 15 17   13 14   10 12
9           8 17   13 14   10 12
10          8 17   13 14   10 12
11          8 17   13 14
12           3 5     1 6
13           2 4
```

3.6 Table 5.4

Get the data, and dichotomize the scores, see previous example

Automated item selection algorithm with two different values for the lower bound.

```
> scale.3 <- aisp(X, lowerbound = 0.3)
```

```
> scale.35 <- aisp(X, lowerbound = 0.35)
```

Make the table (advanced R code)

```
> scale.30 <- aisp(X, lowerbound = 0.3, verbose = F)
> max.scale <- max(scale.30)
> Table.5.4.left <- data.frame()
> for (i in 1:max.scale) {
+     max.item <- max(length(scale.30[scale.30 ==
+         i]))
+     Scale <- c(i, rep("", max.item - 1))
+     Item.30 <- which(scale.30 == i)
+     Hi.30 <- round(coefH(X[, scale.30 == i])$Hi,
+         2)
+     H.30 <- c(rep("", max.item - 1), round(coefH(X[,
+         scale.30 == i])$H, 2))
+     Table.5.4.left <- rbind(Table.5.4.left, data.frame(Scale = Scale,
+         Item = Item.30, Hi = Hi.30, H = H.30),
+         c("", "", "", ""))
+ }
> rownames(Table.5.4.left) <- NULL
> Table.5.4.left
```

```
   Scale Item   Hi    H
1      1    3 0.57
2           5 0.51
3           6  0.3
4           7 0.53
5           9  0.5
6          11 0.45 0.47
7
8      2    1 0.52
9           2 0.61
10          4 0.61
11         13  0.3 0.55
12
13     3    8 0.41
14         15 0.38
15         17 0.49 0.42
16
17     4   10 0.47
18         12 0.47 0.47
19
```

```
> scale.35 <- aisp(X, lowerbound = 0.35, verbose = F)
> max.scale <- max(scale.35)
> Table.5.4.right <- data.frame()
> for (i in 1:max.scale) {
+     max.item <- max(length(scale.35[scale.35 ==
+         i]))
+     Scale <- c(i, rep("", max.item - 1))
+     Item.35 <- which(scale.35 == i)
+     Hi.35 <- round(coefH(X[, scale.35 == i])$Hi,
+         2)
+     H.35 <- c(rep("", max.item - 1), round(coefH(X[,
+         scale.35 == i])$H, 2))
+     Table.5.4.right <- rbind(Table.5.4.right,
+         data.frame(Scale = Scale, Item = Item.35,
+             Hi = Hi.35, H = H.35), c("", "", "",
+             ""))
+ }
> rownames(Table.5.4.right) <- NULL
> Table.5.4.right
```

```
   Scale Item   Hi    H
1      1    3  0.6
2           5 0.53
3           7  0.6
4           9 0.63
5          11 0.49 0.55
6
7      2    1 0.54
8           2 0.68
9           4 0.67 0.64
10
11     3    8 0.41
12         15 0.38
13         17 0.49 0.42
14
15     4   13 0.53
16         14 0.53 0.53
17
18     5   10 0.47
19         12 0.47 0.47
20
```

3.7 Table 6.1

Get the data. The two pseudo task. Item 2 (column 3) and item 4 (column5) were not considered. Also, the first column (Group) is removed from the data. Tasks 3 and 4 (items 5 and 9) were investigated in detail. This is the item pair number 21.

```
> library(mokken)
> data(transreas)
> X <- transreas[, -c(1, 3, 5)]
> check.restscore(X, minsize = 2)$results[[21]]

[[1]]
[[1]][[1]]
[1] "T04W"

[[1]][[2]]
[1] "T03W"


[[2]]
     Group Lo Hi   N     E(X3)     E(X7)  P(X3>=1)
[1,]     1  1  1   2 0.5000000 0.0000000 0.5000000
[2,]     2  2  2   4 0.5000000 0.0000000 0.5000000
[3,]     3  3  3   8 0.7500000 0.3750000 0.7500000
[4,]     4  4  4  27 0.7407407 0.6296296 0.7407407
[5,]     5  5  5  75 0.7600000 0.8266667 0.7600000
[6,]     6  6  6 127 0.8110236 0.9370079 0.8110236
[7,]     7  7  7 117 0.7863248 0.9487179 0.7863248
[8,]     8  8  8  65 0.8000000 0.9846154 0.8000000
      P(X7>=1)
[1,] 0.0000000
[2,] 0.0000000
[3,] 0.3750000
[4,] 0.6296296
[5,] 0.8266667
[6,] 0.9370079
[7,] 0.9487179
[8,] 0.9846154

[[3]]
                    #ac #vi   #vi/#ac     maxvi       sum
P(X3>=1) P(X7>=1)    7   4 0.5714286 0.5000000 1.4861111
E(X3) E(X7)          7   4 0.5714286 0.1846154 0.5396595
```

```
Total               7   4 0.5714286 0.5000000 1.4861111
                      sum/#ac      zmax #zsig
P(X3>=1) P(X7>=1) 0.21230159 0.8913385     0
E(X3) E(X7)       0.07709421 3.7574830     3
Total             0.21230159 0.8913385     0

[[4]]
[[4]][[1]]
[1] 0

[[4]][[2]]
[1] 0

[[4]][[3]]
[1] 0

[[4]][[4]]
[1] 0

[[4]][[5]]

        Welch Two Sample t-test

data:  X[member == gg, i] and X[member == gg, j]
t = -1.0049, df = 145.896, p-value = 0.1583
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
       -Inf 0.04315280
sample estimates:
mean of x mean of y
0.7600000 0.8266667


[[4]][[6]]

        Welch Two Sample t-test

data:  X[member == gg, i] and X[member == gg, j]
t = -3.0693, df = 210.514, p-value = 0.001214
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
        -Inf -0.05817007
sample estimates:
mean of x mean of y
```

```
0.8110236 0.9370079


[[4]][[7]]

        Welch Two Sample t-test

data:  X[member == gg, i] and X[member == gg, j]
t = -3.7575, df = 177.982, p-value = 0.0001162
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
        -Inf -0.09093293
sample estimates:
mean of x mean of y
0.7863248 0.9487179


[[4]][[8]]

        Welch Two Sample t-test

data:  X[member == gg, i] and X[member == gg, j]
t = -3.529, df = 76.011, p-value = 0.0003558
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
       -Inf -0.0975059
sample estimates:
mean of x mean of y
0.8000000 0.9846154

> check.restscore(X, minsize = 40)$results[[21]]

[[1]]
[[1]][[1]]
[1] "T04W"

[[1]][[2]]
[1] "T03W"


[[2]]
     Group Lo Hi  N     E(X3)     E(X7)  P(X3>=1)
[1,]     1  1  4 41 0.7073171 0.4878049 0.7073171
[2,]     2  5  5 75 0.7600000 0.8266667 0.7600000
```

```
[3,]     3  6  6 127 0.8110236 0.9370079 0.8110236
[4,]     4  7  7 117 0.7863248 0.9487179 0.7863248
[5,]     5  8  8  65 0.8000000 0.9846154 0.8000000
      P(X7>=1)
[1,] 0.4878049
[2,] 0.8266667
[3,] 0.9370079
[4,] 0.9487179
[5,] 0.9846154

[[3]]
                  #ac #vi #vi/#ac     maxvi       sum
P(X3>=1) P(X7>=1)   4   1    0.25 0.2195122 0.2195122
E(X3) E(X7)         4   1    0.25 0.2195122 0.2195122
Total               4   1    0.25 0.2195122 0.2195122
                     sum/#ac     zmax #zsig
P(X3>=1) P(X7>=1) 0.05487805 1.679342     1
E(X3) E(X7)       0.05487805 2.053960     0
Total             0.05487805 1.679342     1

[[4]]
[[4]][[1]]

        Welch Two Sample t-test

data:  X[member == gg, i] and X[member == gg, j]
t = 2.054, df = 79.303, p-value = 0.9784
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
    -Inf 0.39738
sample estimates:
mean of x mean of y
0.7073171 0.4878049


[[4]][[2]]
[1] 0

[[4]][[3]]
[1] 0

[[4]][[4]]
[1] 0
```

```
[[4]][[5]]
[1] 0

> plot(check.restscore(X, minsize = 2), item.pairs = 21)
> plot(check.restscore(X, minsize = 40), item.pairs = 21)
> R <- apply(X[, -c(3, 7)], 1, sum)
> table(X[, 3], X[, 7], R)

, , R = 1


     0  1
  0  1  0
  1  1  0

, , R = 2


     0  1
  0  2  0
  1  2  0

, , R = 3


     0  1
  0  1  1
  1  4  2

, , R = 4


     0  1
  0  1  6
  1  9 11

, , R = 5


     0  1
  0  0 18
  1 13 44

, , R = 6
```

```
    0  1
  0  1 23
  1  7 96

, , R = 7


    0  1
  0  2 23
  1  4 88

, , R = 8


    0  1
  0  0 13
  1  1 51

> as.numeric(table(X[, 3][R < 5], X[, 7][R < 5]))

[1]  5 16  7 13
```

3.8 Table 6.2

Get the data. The two pseudo task. Item 2 (column 3) and item 4 (column5) were not considered. Also, the first column (Group) is removed from the data.

```
> library(mokken)
> data(transreas)
> X <- transreas[, -c(1, 3, 5)]
> Task <- c(9, 10, 4, 5, 2, 7, 3, 1, 8, 6)
> ppp <- check.pmatrix(X)$Ppp
> dimnames(ppp) <- list(Task, Task)
> round(ppp, 2)

      9   10    4    5    2    7    3    1    8    6
9    NA 0.22 0.23 0.25 0.24 0.28 0.28 0.29 0.30 0.30
10 0.22   NA 0.39 0.44 0.42 0.48 0.49 0.51 0.51 0.52
4  0.23 0.39   NA 0.64 0.69 0.67 0.69 0.73 0.76 0.76
5  0.25 0.44 0.64   NA 0.66 0.69 0.73 0.77 0.78 0.79
2  0.24 0.42 0.69 0.66   NA 0.68 0.72 0.76 0.78 0.80
```

```
7   0.28 0.48 0.67 0.69 0.68   NA 0.79 0.82 0.83 0.84
3   0.28 0.49 0.69 0.73 0.72 0.79   NA 0.86 0.88 0.88
1   0.29 0.51 0.73 0.77 0.76 0.82 0.86   NA 0.92 0.93
8   0.30 0.51 0.76 0.78 0.78 0.83 0.88 0.92   NA 0.96
6   0.30 0.52 0.76 0.79 0.80 0.84 0.88 0.93 0.96   NA
```

```
> pmm <- check.pmatrix(X)$Pmm
> dimnames(pmm) <- list(Task, Task)
> round(pmm, 2)
```

```
       9   10    4    5    2    7    3    1    8    6
9     NA 0.40 0.14 0.15 0.13 0.13 0.10 0.05 0.03 0.02
10  0.40   NA 0.08 0.12 0.09 0.12 0.09 0.04 0.03 0.02
4   0.14 0.08   NA 0.05 0.09 0.04 0.02 0.01 0.01 0.01
5   0.15 0.12 0.05   NA 0.04 0.04 0.04 0.02 0.01 0.01
2   0.13 0.09 0.09 0.04   NA 0.03 0.02 0.01 0.01 0.01
7   0.13 0.12 0.04 0.04 0.03   NA 0.06 0.03 0.02 0.02
3   0.10 0.09 0.02 0.04 0.02 0.06   NA 0.04 0.02 0.02
1   0.05 0.04 0.01 0.02 0.01 0.03 0.04   NA 0.01 0.01
8   0.03 0.03 0.01 0.01 0.01 0.02 0.02 0.01   NA 0.01
6   0.02 0.02 0.01 0.01 0.01 0.02 0.02 0.01 0.01   NA
```


Acknowledgements

Thanks are due to Wybrandt van Schuur for comments on the first draft of this report.

References

Baron, J., & Li, Y. (2007). *Notes on the use of R for psychology experiments and questionnaires*. Unpublished manuscript. Retrieved from http://www.psych.upenn.edu/~baron/rpsych/rpsych.html

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

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

Ligtvoet, R., Van der Ark, L. A., Te Marvelde, J. M., & Sijtsma, K. (2010). Investigating an invariant item ordering for polytomously scored items. *Educational and Psychological Measurement.*

Ligtvoet, R, van der Ark, L. A. , Bergsma, W. P., & Sijtsma, K. (2010). Polytomous latent scales for the investigation of the ordering of items. Manuscript submitted for publication.

Mokken, R. J. (1971). *A Theory and Procedure of Scale Analysis.* Berlin, Germany: De Gruyter.

Molenaar, I. W. and K. Sijtsma (1984). Internal consistency and reliability in Mokken's nonparametric item response model. *Tijdschrift voor onderwijsresearch, 9*, 257–268.

Molenaar, I. W. and K. Sijtsma (1988). Mokken's approach to reliability estimation extended to multicategory items. *Kwantitatieve methoden, 9(28)*, 115-126.

Molenaar, I.W., & Sijtsma, K. (2000). User's Manual MSP5 for Windows [Software manual]. Groningen, The Netherlands: IEC ProGAMMA.

Muenchen, R. A. (2008). *R for SAS and SPSS Users.* Berlin: Springer.

Paradis, E. (2005). *R for beginners.* Unpublished manuscript. Retrieved from http://cran.r-project.org/doc/contrib/Paradis-rdebuts_en.pdf

R Development Core Team (2006). *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing.

R Development Core Team (2009). *An Introduction to R.* Unpublished manuscript. Retrieved from http://cran.r-project.org/doc/manuals/R-intro.html

Sijtsma, K. and I. W. Molenaar (1987). Reliability of test scores in nonparametric item response theory. *Psychometrika, 52*,79-97.

Sijtsma, K., & Molenaar, I. W. (2002). *Introduction to nonparametric item response theory.* Thousand Oaks, CA: Sage.

Straat, J. H., Van der Ark, L. A., & Sijtsma, K. (2008). Comparing optimization algorithms for item selection in Mokken scale analysis. Paper submitted for publication.

Van der Ark, L. A. (2007). Mokken scale analysis in R. *Journal of Statistical Software, 20* (11), 1-19.

Van der Ark, L. A. (2010). Computation of the Molenaar Sijtsma statistic. In A. Fink, B. Lausen, W. Seidel, & A. Ultsch (Eds.), *Advances in data analysis, data handling and business intelligence* (pp. 775-784). Berlin: Springer.