Getting started with Mokken scale analysis in R

Andries van der Ark

May 26, 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 2.2, 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).

The report can be read best from a computer screen because it contains colored text: internet links, R code, R results, and Links as they appear on the R websites. The R code may be selected from the document and pasted into the R console.

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 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.
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)` (Figure 1.5), choose a location nearby you (Figure 1.6), and choose the package `mokken` (Figure 1.7).

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.

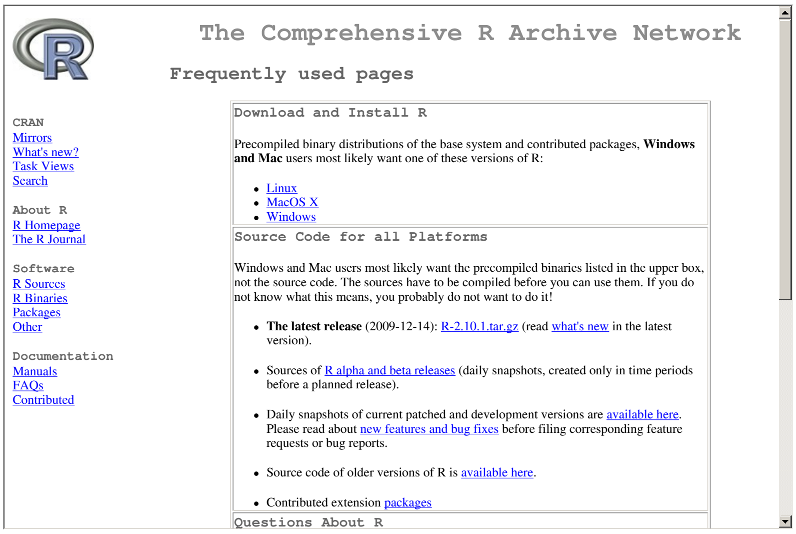


Figure 1.1: *R website. Click on Windows.*

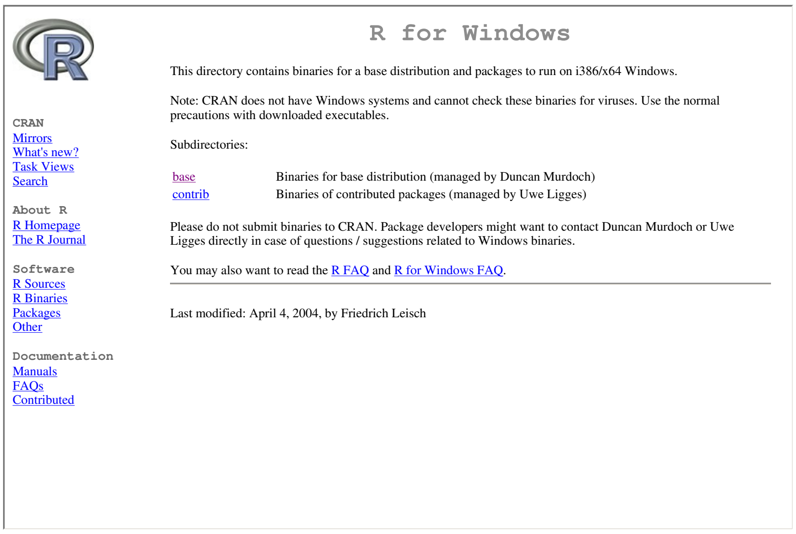


Figure 1.2: *R website. Click on base.*

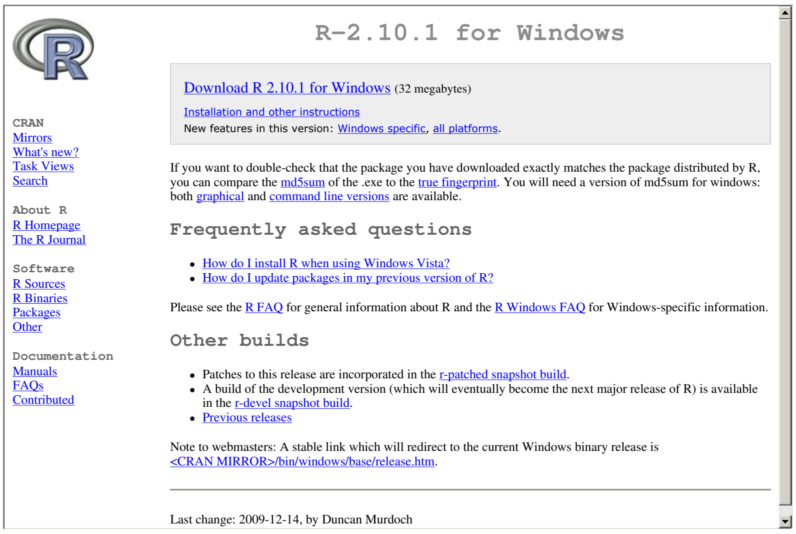


Figure 1.3: *R website. Click on Download R 2.10.1 for Windows.*

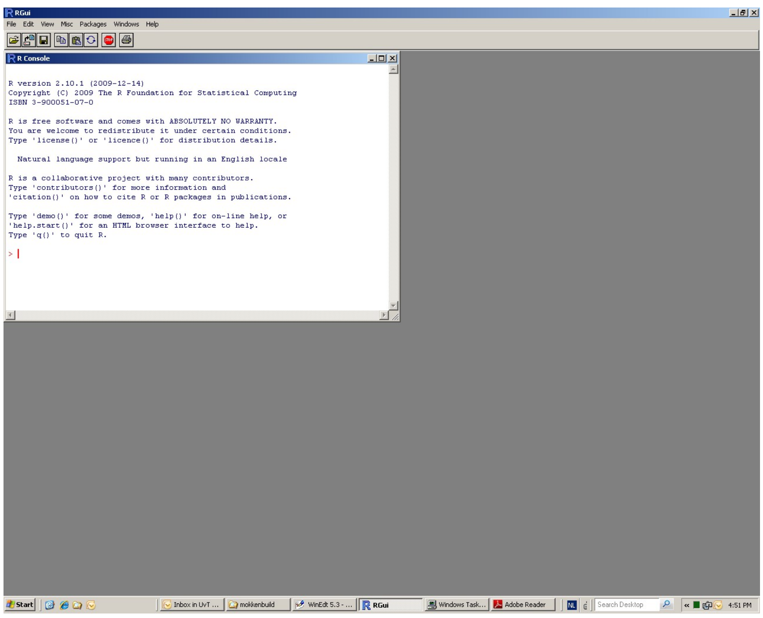


Figure 1.4: *R console.*

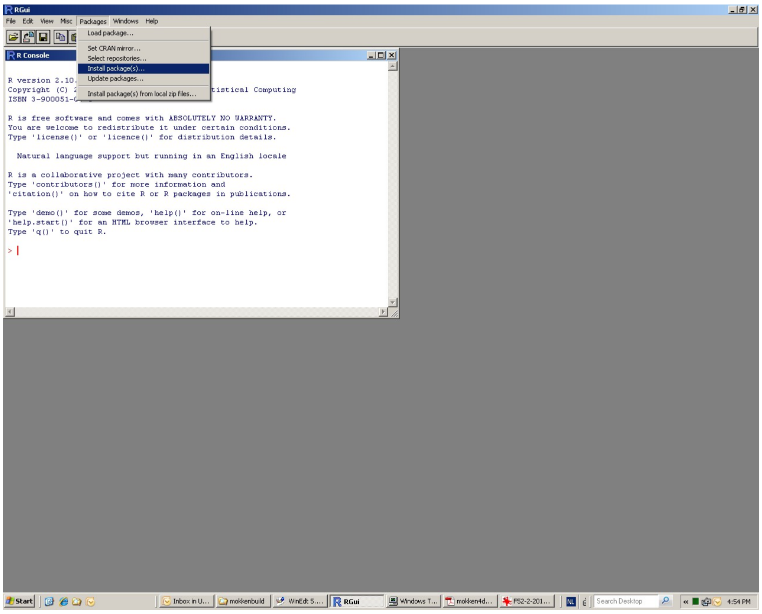


Figure 1.5: *R console*. Choose `Packages`, `Install package(s)`.

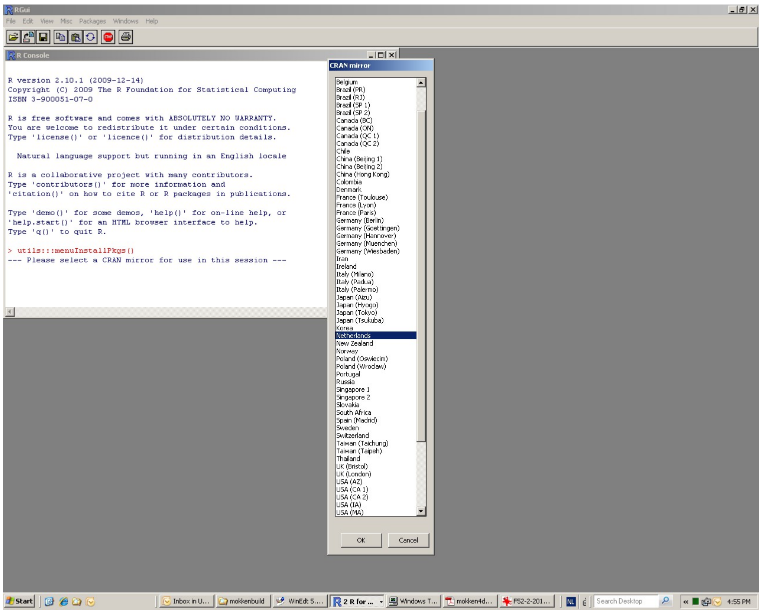


Figure 1.6: *R console*. Choose a location nearby you.

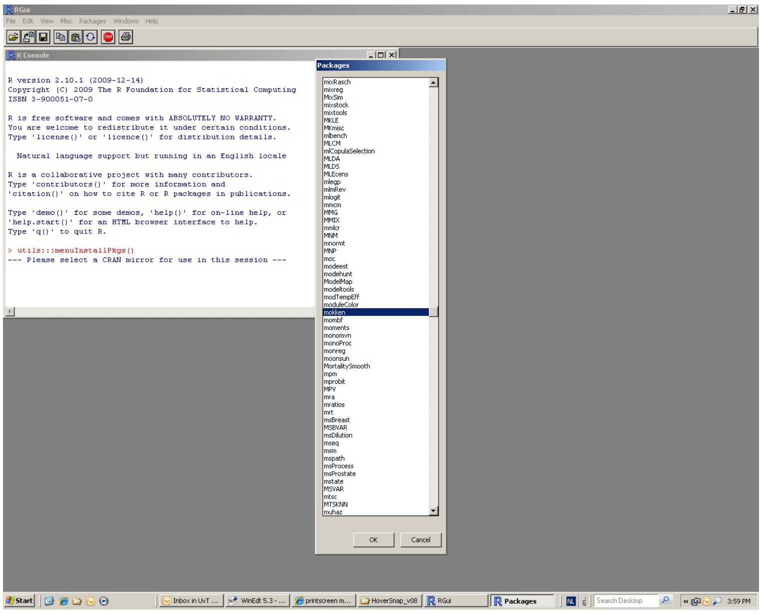


Figure 1.7: *R console*. Choose `mokken`.

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 red, the resulting output on the screen produced by R is printed in blue. Note that R is case sensitive. Some examples.

If you type

```
6 - 3
```

R returns $6 - 3 =$

```
[1] 3
```

If you type

```
x <- sqrt(2)
x
```

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

```
[1] 1.414214
```

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] `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.

1. Type the following code in the R console

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

   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 1.8) 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.
```

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

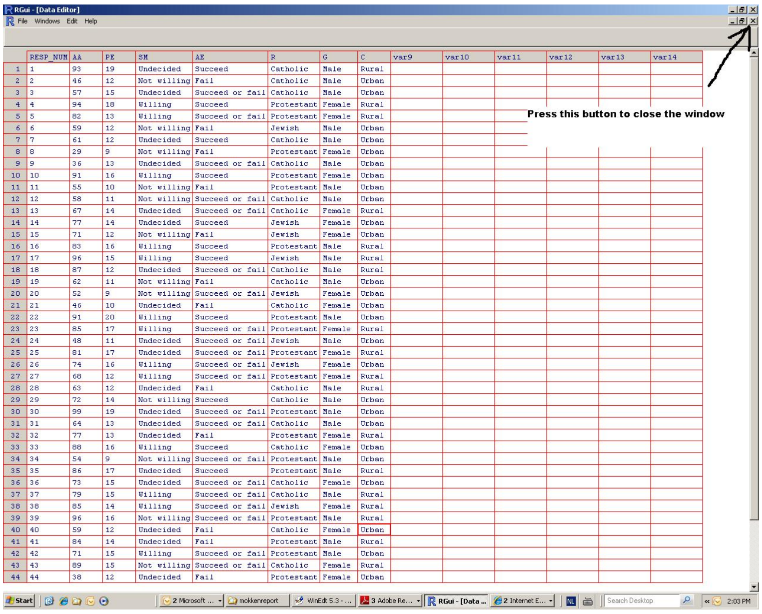


Figure 1.8: *Close the spreadsheet by clicking the button in the upper right-hand corner*

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")
```

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.

  ```
  # help(mokken)
  ```

  does not do anything.

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

  ```
  data(acl)
  data(cavalini)
  data(trans.reas)
  ```

  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

  ```
  X # lots of output
  Y
  Z
  ```

- 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 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.

Example 1 *labelE1*

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

```
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` are the following (for more information type `help(aisp)`).

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

  ```
  scale2 <- 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`)

  ```
  scale3 <- aisp(Communality, lowerbound = .2, alpha =.10)
  ```

- Direct no output to the screen during the item selection (default `verbose = TRUE`)

  ```
  scale4 <- aisp(Communality, verbose=FALSE)
  ```

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 Example 2 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 computed. Third, the item scalability coefficients are computed but rounded to two integers.

Example 2

```
data(acl)
Communality <- acl[,1:10]
coefH(Communality)
coefH(Communality)$Hi
round(coefH(Communality)$Hi,2)
```

The output for the last two commands is as follows.

```
  reliable         honest   unscrupulous*      deceitful* unintelligent*
 0.3038656      0.2651096       0.2360455       0.3191367      0.1160265
obnoxious*     thankless*     unfriendly*     dependable           cruel*
```

```
 0.2879527       0.2454761       0.3085198       0.2994265       0.2522276

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

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 Example 3, 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).

Example 3

```
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 #tsig
cruel*         3.48  36   0    0.00  0.00 0.00    0.00 0.00     0
unintelligent* 3.32  35   2    0.06  0.15 0.29    0.01 2.17     1
unscrupulous*  3.32  35   1    0.03  0.14 0.14    0.00 1.21     0
unfriendly*    3.30  36   1    0.03  0.15 0.15    0.00 2.17     1
thankless*     3.26  36   1    0.03  0.12 0.12    0.00 1.50     0
dependable     3.25  36   0    0.00  0.00 0.00    0.00 0.00     0
obnoxious*     3.25  36   1    0.03  0.12 0.12    0.00 1.50     0
reliable       3.09  36   0    0.00  0.00 0.00    0.00 0.00     0
```

```
honest          3.02  36    2     0.06   0.18 0.31     0.01 2.08      1
deceitful*      2.94  34    2     0.06   0.18 0.31     0.01 2.08      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` are the following (for more information type `help(check.iio)`).

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

```
check.iio(Communality, minvi=0.00, minsize=50)
```

- Using methods MS-CPM and IT

```
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=.01))
```

- Without backward selection algorithm, and with information screen

```
summary(check.iio(Communality, item.selection=FALSE))
x <- summary(check.iio(Communality, verbose=TRUE))
```

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 Example 4 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 shows the 10 graphs.

Example 4

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

The output for the last two commands is as follows.

```
               ItemH #ac #vi #vi/#ac maxvi  sum sum/#ac zmax #zsig
reliable        0.30  24   0    0.00  0.00 0.00       0 0.00     0
honest          0.27  24   0    0.00  0.00 0.00       0 0.00     0
unscrupulous*   0.24  24   0    0.00  0.00 0.00       0 0.00     0
deceitful*      0.32  24   0    0.00  0.00 0.00       0 0.00     0
unintelligent*  0.12  24   1    0.04  0.07 0.07       0 0.85     0
obnoxious*      0.29  24   0    0.00  0.00 0.00       0 0.00     0
thankless*      0.25  24   0    0.00  0.00 0.00       0 0.00     0
unfriendly*     0.31  24   0    0.00  0.00 0.00       0 0.00     0
dependable      0.30  24   0    0.00  0.00 0.00       0 0.00     0
cruel*          0.25  24   0    0.00  0.00 0.00       0 0.00     0

Waiting to confirm page change...
Waiting to confirm page change...
Waiting to confirm page change...
Waiting to confirm page change...
Waiting to confirm page change...
Waiting to confirm page change...
Waiting to confirm page change...
```

```
Waiting to confirm page change...
Waiting to confirm page change...
Waiting to confirm page change...
```

Variations of `check.monotonicity` are the following (for more information type `help(check.monotonicity)`).

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

  ```
  check.monotonicity(Communality, minvi=0.00, 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()
  ```

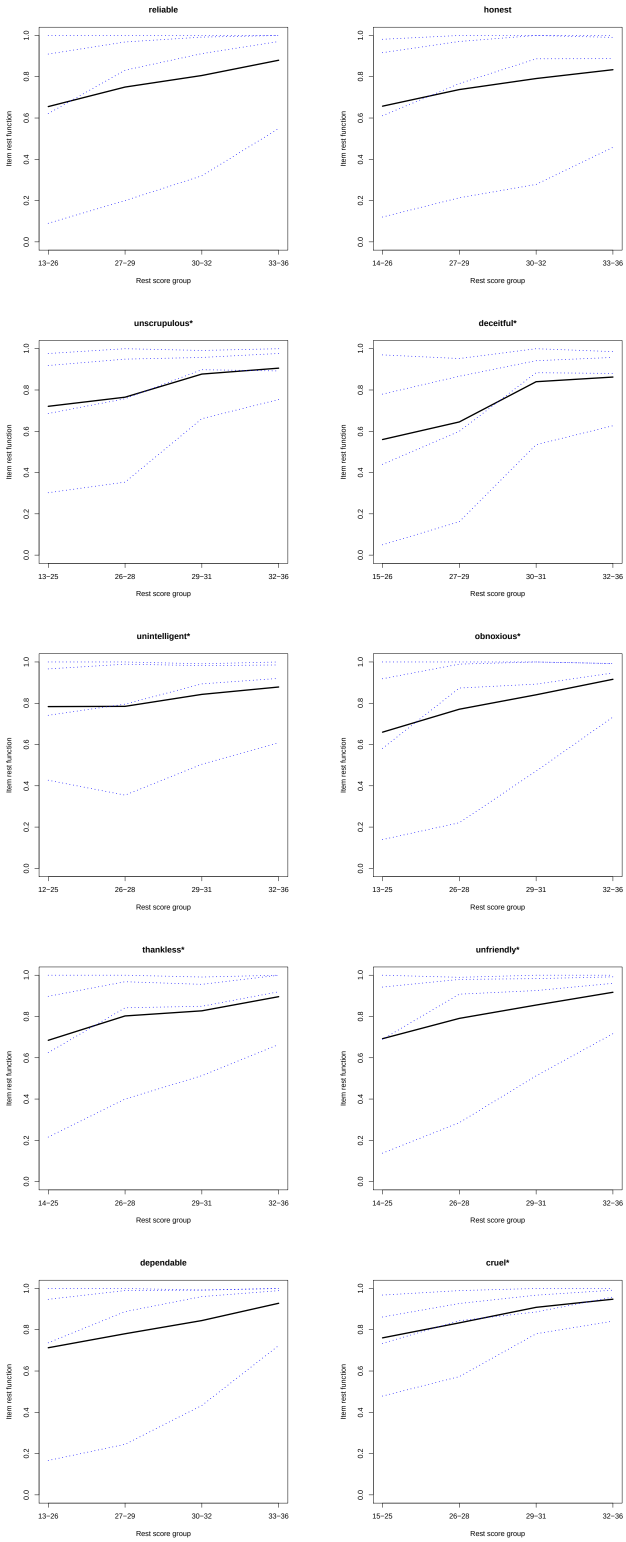
2.1.5 `check.pmatrix`

Investigates the assumption of nonintersecting item step response functions using the P++ and P−− matrix (Molenaar & Sijtsma, 2000, pp. 80-85). In Example 5 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 see Molenaar and Sijtsma (2000, pp. 80-85). Without further specifications `plot()` displays 20 graphs (2 for each item) in a separate R Window, and requires a hard return to go to the next graph.

Example 5

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

Figure 2.1: *Plots in Example 4.*



```
pmatrix.results
summary(pmatrix.results)
plot(pmatrix.results)
```

The output for `summary(pmatrix.results)` is as follows.

```
$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 (for more information type `help(check.pmatrix)`).

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

  ```
  check.pmatrix(Communality, minvi=0.00)
  ```

- 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()
```

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 Example 6 the scores on the first ten items from ACL are used; these are the items of the scale Communality.

Example 6

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

The output for `summary(restscore.results)` is as follows.

```
$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 Example 7 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.

Example 7

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

The output for `summary(restscore.results)` is as follows.

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

Variations of `check.restscore` are the following (for more information type `help(check.restscore)`).

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

  ```
  check.restscore(Communality, minvi=0.00, minsize=50)
  ```

- Plot the results for the first item pair (item 1 and item 2) and the second item pair (item 1 and item 3) only.

  ```
  plot(check.restscore(Communality), item.pairs=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("restscore.pdf")
  plot(restscore.results, ask=FALSE)
  dev.off()
  ```

2.1.8 `check.groups`

The package `mokken` does not 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. The reason is that Mokken scale analysis for different subgroups can be done easily using standard R commands. Example 8 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.

Example 8

```
data(acl)
Communality <- acl[,1:10]
Group <- acl[,11]
coefH(Communality[Group==0|Group==1,])$H
coefH(Communality[Group==2,])$H
coefH(Communality[Group==3,])$H
coefH(Communality[Group==4,])$H
```

The output is

```
[1] 0.1963215
[1] 0.3038342
[1] 0.2569371
[1] 0.2465098
```

2.2 Examples of Mokken scale analysis in R

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

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)
apply(item.scores[grades==2,],2,mean)
apply(item.scores[grades==3,],2,mean)
apply(item.scores[grades==4,],2,mean)
apply(item.scores[grades==5,],2,mean)
apply(item.scores[grades==6,],2,mean)
```

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
```

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)$Hi
coefH(item.scores)$H
coefZ(item.scores)$Zi
coefZ(item.scores)$Z
```

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)])$Hi
coefH(item.scores[,-c(2,4)])$H
coefZ(item.scores[,-c(2,4)])$Zi
coefZ(item.scores[,-c(2,4)])$Z
```

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
```

Table 5.1 To get the data, see Table 3.1.

Automated item selection algorithm

```
scale <- aisp(item.scores)
```

Construction of Table 5.1 (advanced R code).

```
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
```

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)
```

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
```

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=0.00)
aisp(X,lowerbound=0.05)
aisp(X,lowerbound=0.10)
# etc.
```

Make the table (advanced R code)

```
lower.bound <- seq(0,.6,by=.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
```

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=0.30)
scale.35 <- aisp(X,lowerbound=0.35)
```

Make the table (advanced R code)

```
scale.30 <- aisp(X,lowerbound=0.30,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.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
```

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)$results[[21]]
check.restscore(X,minsize=40)$results[[21]]
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)
as.numeric(table(X[,3][R < 5],X[,7][R < 5]))
```

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)

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


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.