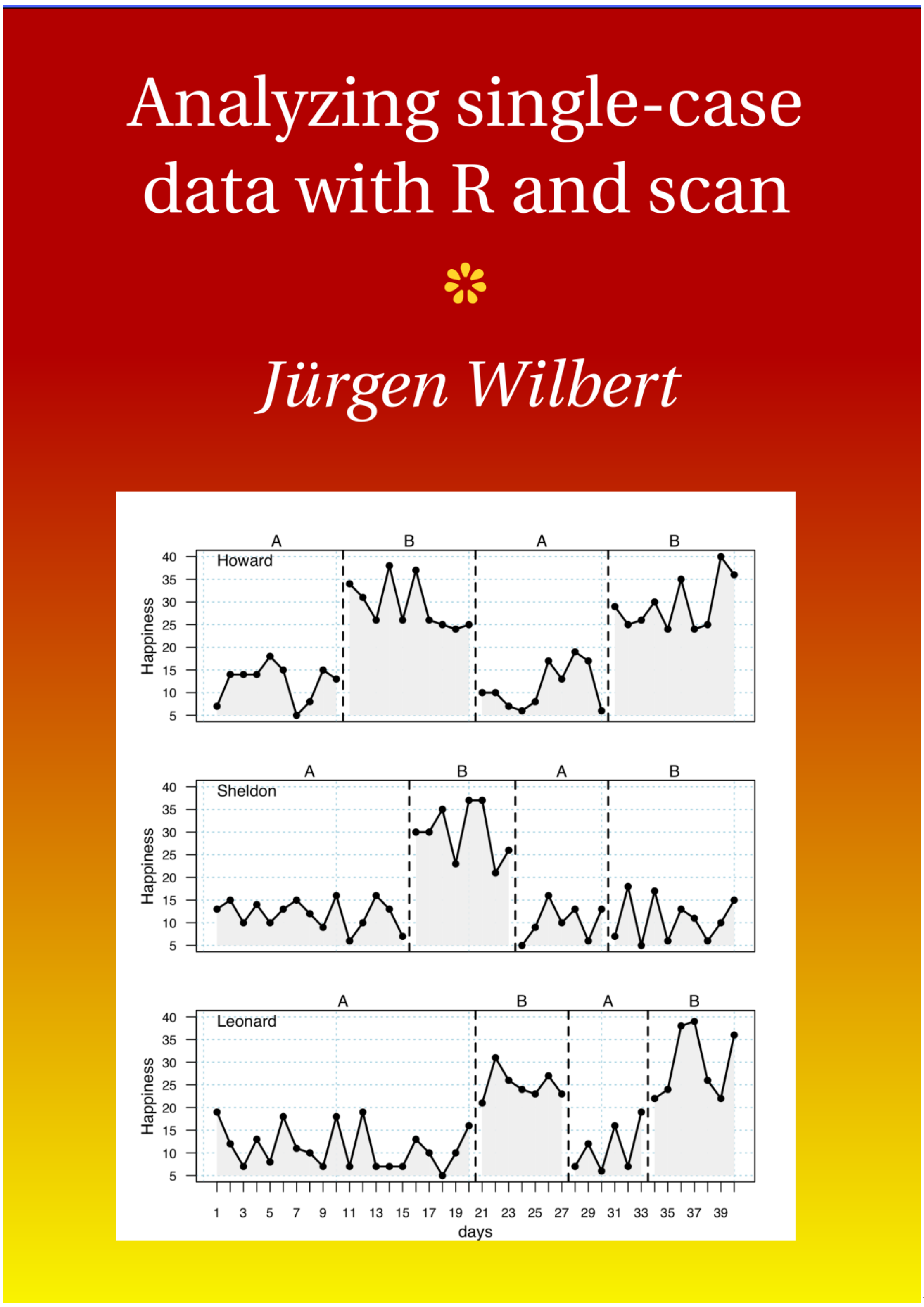
Analyzing single-case data with R and scan

Jürgen Wilbert

2019-08-02

Contents

Welcome



{width = 400px}

Preface

Hello!

I am glad your found your way to this book as is tells me you are beginning to use the scan package.
While `scan` is quiet thoroughly developed, this book is at an early stage. I am continuously working on it and extending it.
At this point in time there is no release of this book available. Only this draft which is full of errors (code and typos).

Thank you!

Jürgen

02 August 2019

Chapter 1

Introduction

Single case research has become an important and broadly accepted method for gaining insight into educational processes. Especially the field of special education has adopted single-case research as a proper method for evaluating the effectiveness of an intervention or the developmental processes underlying problems in acquiring academic skills. Single-case studies are also popular among teachers and educators who are interested in evaluating the learning progress of their students. The resulting information of a single-case research design provide helpful information for pedagogical decision processes regarding further teaching processes of an individual student but also help to decide, whether or how to implement certain teaching methods into a classroom.

This book has been created using the `Rmarkdown` (Allaire et al., 2019) and `bookdown` (Xie, 2019) environment. The analyses have been conducted with the **R** package `scan` at version 0.4.0 (Wilbert & Lueke, 2019).

R version 3.6.1 (2019-07-05) was used (R Core Team, 2019).

Chapter 2

Some things about R

R is a programming language optimized for statistical purposes. It was created in 1992 by Ross Ihaka and Robert Gentleman at the University of Auckland. Since then it has been developed continuously and became one of the leading statistical software programs. R is unmatched in its versatility. It is used for teaching introductory courses into statistics up to doing the most sophisticated mathematical analysis. It has become the defacto standard in many scientific disciplines from the natural to the social sciences.
R is completely community driven. That is, it is developed and extended by anybody who likes to participate. It comes at no costs and can be downloaded for free for all major and many minor platforms at www.r-project.org. Yet, it is as reliable as other proprietary software like Mplus, STATA, SPSS etc. You can tell from my writing that is hard not to become an R-fan when you are into statistics :-)
R can be used in at least two ways:

1. You can use it for applying data analyses. In that way it functions like most other statistical programs. You have to learn the specific syntax of R and it will compute the data analysis you need. For example `mean(x)` will return the mean of the variable `x`; `lm(y ~ x)` will calculate a linear regression with the criteria `y` and the predictor `x` for you or `plot(x, y)` will return a scatter-plot of the variables `x` and `y`.
2. You can use R to program new statistical procedures, or extend previous ones.

It is the second function that is the origin of R's huge success and versatility. New statistical procedures and functions can be published to be used for everyone in so called packages. A package usually contains several functions, help files and example data-sets. Hundreds of such packages are available to help in all kinds of specialized analyses. The basic installation of R comes with a large variety of packages per installed. New packages can most of the times be easily installed from within R. Admittedly, if you must have the latest developmental version of a new package installation sometimes can get a bit more complex. But with a bit of help and persistence it is not to difficult to accomplish.

The book at hand describes the use of such an additional package named *scan* providing specialized functions for single-case analyses. *scan* comes in two versions: A "stable" version and a developmental version. Both versions can be installed directly from within R. The stable version is much older and only provides a limited functionality. Therefore, I will refer to the developmental version in this book.

2.1 Basic R

R is a script language. That is, you type in text and let R execute the commands you wrote down. Either you work in a *console* or a *textfile*. In a *console* the command will be executed every time you press the RETURN-key. In n a *textfile* you type down your code, mark the part you like to be executed, and run that code (with a click or a certain key). The latter text files can be saved and reused for later R sessions. Therefore, usually you will work in a text file.

A value is assigned to a variable with the `<-` operator. Which should be read as an arrow rather than a less sign and a minus sign. A `#` is followed by a comment to make your code more understandable. So, what follows a `#` is not interpreted by R. A vector is a chain of several values. With a vector you could describe the values of a measurement series. The `c` function is used to build a vector (e.g., `c(1, 2, 3, 4)`). If you like to see the content of a variable you could use the `print` function. `print(x)` will display the content of the variable `x`. A shortcut for this is just to type `x`.

```r
# x is assigned the value 10:
x <- 10

# See what's inside of x:
x
```

```
## [1] 10
```

```r
# x is assigned a vector with three values:
x <- c(10, 11, 15)

# ... and display the content of x:
x
```

```
## [1] 10 11 15
```

Two important concepts in **R** are *functions* and *arguments*. A *function* is the name for a procedure that does something with the *arguments* that are provided by you. For example, the function `mean` calculated the mean. `mean` takes the argument `x` with is the vector (a series of values) from which it will calculate the mean. `mean( x = c(1, 3, 5) )` will compute the mean of the values 1, 3, and 5 and return the result 3. Some *functions* can take several arguments. `mean` for example also takes the argument `trim`. For calculating a trimmed mean. `mean( x = c(1, 1, 3, 3, 5, 6, 7, 8, 9, 9), trim = 0.1)` will calculate the 10% trimmed mean of the provided values. The name of the first argument could be dropped. That is, `mean( c(1, 3, 5) )` will be interpreted by **R** as `mean( x = c(1, 3, 5) )`. You could also provide a variable to an argument.

```r
y <- c(1, 4, 5, 6, 3, 7, 7, 5)
mean(x = y)
```

```
## [1] 4.75
```

```r
# or shorter:
mean(y)
```

```
## [1] 4.75
```

The return value of a function can be assigned to a new variable instead:

```r
y <- c(1, 4, 5, 6, 3, 7, 7, 5)
res <- mean(y)
#now res contains the mean of y:
res
```

```
## [1] 4.75
```

Every function in R has a help page written by the programmers. You can retrieve these pages with the `help` function or its short cut `?`. `help("mean")` will display the help page for the `mean` function. The quotation marks are necessary here because you do not provide a variable with the name *mean* but a word 'mean'. The shortcut works `?mean`. A bit confusingly, you do not need the quotation marks here.

2.2 Installing the *scan* package

You can use the `install.packages` function to install *scan*.

1. `install.packages("scan", repos="http://R-Forge.R-project.org", type = "source")` will install the developmental version and
2. `install.packages("scan")` will install the stable version.

At some time in the future I will release a new stable version. But as long as the stable version is `0.20` it is deprecated and you need to install the developmental version. Please look at the beginning of the Chapter "Introduction" which version of *scan* has been used for creating this book and make sure you have this version or a newer one installed.

R contains many packages and it would significantly slow down if all packages would be loaded into the computer memory at the beginning of each R session. Therefore, after installing *scan* it needs to be activated at the beginning of each session you use R. Usually a session starts when you start the R program and ends with closing R.

For activating a package you need the `library` function. In this case `library(scan)`. You should get something like

```
scan 0.4.0 (development version)
Single-Case Data Analysis for Single and Multiple Baseline Designs
```

indicating that everything went smoothly and *scan* is ready for the job.

Chapter 3

Managing single-case data

3.1 A *single-case data frame*

Scan provides its own data-class for encoding single-case data: the *single-case data frame* (short *scdf*). An *scdf* is an object that contains one or multiple single-case data sets and is optimized for managing and displaying these data. Think of an scdf as a file including a separate datasheet for each single case. Each datasheet is made up of at least three variables: The measured **values**, the **phase** identifier for each measured value, and the measurement time (**mt**) of each measure. Optionally, scdfs could include further variables for each single-case (e.g., control variables), and also a name for each case.

Technically, an scdf object is a list containing data frames. It is of the class `c("scdf","list")`. Additionally, an *scdf* entails some attributes. `dvar`, `pvar`, and `mvar` contain the names of the `values`, `phase`, and the `measurement time` variable. By default, these are set to `values`, `phase`, and `mt`.

Several functions are available for creating, transforming, merging, and importing/exporting *scdfs*.

3.2 Creating scdfs

The `scdf` function is the basic tool for creating a single-case data frame. Basically, you have to provide the measurement *values* and the *phase* structure and a scdf object is build. There are three different ways of defining the phase structure. First, defining the beginning of the B-phase with the `B.start` argument, second, defining a design with the `phase.design` argument and third, setting paramters in a named vector of the dependent variable.

```
### Three ways to code the same scdf
scdf(values = c(A = 2,2,4,5, B = 8,7,6,9,8,7))
scdf(values = c(2,2,4,5,8,7,6,9,8,7),
     B.start = 5)
```

```r
scdf(values = c(2,2,4,5,8,7,6,9,8,7),
     phase.design = c(A = 4, B = 6))
```

The `B.start` argument is only applicable when the single-case has one A-phase followed by one B-phase. The number assigned to `B.start` indicates the measurement-time as defined in the `mt` argument. That is, when the vector for the measurement times is `mt = c(1,3,7,10,15,17,18,20)` and `B.start = 15` the first measurement of the B-phase will start with the fifth measurement at which $mt = 15$.
The `phase.design` argument is a named vector with the name and length of each phase. The names of each set can be set arbitrary, although I recommend to follow the rule of using capital letters (A, B, C, ...) for each phase and, when indicated, a number if the phases repeat (A1, B1, A2, B2, ...). Although it is possible to give the same name to more than one phase (A, B, A, B) this might lead to some confusion and errors when coding analyzes with *scan*.
When the vector of the dependent variable includes named values, a phase.design structure is created automatically. Each named value sets the beginning of a new phase. For exmaple `c(A = 3,2,4, B = 5,4,3, C = 6,7,6,5)` will create an ABC-phase design with 3, 3, and 4 values per phase.
Use only one of the three methods at a time and I recommend to use the `phase.design` argument or the named vector method as they are the most versatile.
If no measurement times are given, scdf automatically adds them numbered sequentially 1, 2, 3, ..., N where N is the number of measurements. in some circumstances it might be useful to define individual measurement times for each measurement. For example, if you want to include the days since the beginning of the study as time intervals between measurements are widely varying you might get more valid results this way when analyzing the data in a regression approach.

```r
# example of a more complex design
scdf(values = c(2,2,4,5, 8,7,6,9,8,7, 12,11,13),
     mt = c(1,2,3,6, 8,9,11,12,16,18, 27,28,29),
     phase.design = c(A = 4, B = 6, C = 3))
```

```
## #A single-case data frame with one case
##
##  Case1: values mt phase
##              2  1     A
##              2  2     A
##              4  3     A
##              5  6     A
##              8  8     B
##              7  9     B
##              6 11     B
##              9 12     B
##              8 16     B
##              7 18     B
##             12 27     C
##             11 28     C
##             13 29     C
```

Missing values could be coded using `NA`.

Table 3.1: Arguments of the scdf function

Argument	What it does ...
B.start	The first measurement of phase B (simple coding if design is strictly AB).
phase.design	A vector defining the length and label of each phase.
name	A name for the case.
dvar	The name of the dependent variable. By default this is 'values'.
pvar	The name of the variable containing the phase information. By default this is 'phase'.
mvar	The name of the variable with the measurement-time. The default is 'mt'.
...	Any number of variables with a vector asigned to them.

```
scdf(values = c(A = 2,2,NA,5, B = 8,7,6,9,NA,7))
```

Further variables could be implemented by defining new variable names with a vector containing the values. Please consider that these new variable must never have the same name as one of the arguments of the function (B.start, phase.design, name, dvar, pvar, mvar).

```
scdf(values = c(A = 2,2,3,5, B = 8,7,6,9,7,7),
     teacher = c(0,0,1,1,0,1,1,1,0,1),
     hour = c(2,3,4,3,3,1,6,5,2,2))
```

```
## #A single-case data frame with one case
##
##  Case1: values teacher hour mt phase
##              2       0    2  1     A
##              2       0    3  2     A
##              3       1    4  3     A
##              5       1    3  4     A
##              8       0    3  5     B
##              7       1    1  6     B
##              6       1    6  7     B
##              9       1    5  8     B
##              7       0    2  9     B
##              7       1    2 10     B
```

Table 3.1 shows a complete list of arguments that could be passed to the function.

If you want to create a data-set comprising several single-cases the easiest way is to first create an scdf for each case and then join them into a new scdf with the `c` command:

```
case1 <- scdf(values = c(5, 7, 10, 5, 12, 7, 10, 18, 15, 14, 19),
              B.start = 6, name = "Charlotte")
case2 <- scdf(values = c(3, 4, 3, 5, 7, 4, 7, 9, 8, 10, 12),
              B.start = 5, name = "Theresa")
case3 <- scdf(values = c(9, 8, 8, 7, 5, 7, 6, 14, 15, 12, 16),
              B.start = 7, name = "Antonia")
mbd <- c(case1, case2, case3)
```

If you like to use other than the default variable names (“values”, “phase”, and “mt”) you could define these with the `dvar` (for the dependent variable), `pvar` (the variable indicating the phase), and `mvar` (the measurement-time variable) arguments.

```
# Example: Using a different name for the dependent variable
case <- scdf(score = c(5, 7, 10, 5, 12, 7, 10, 18, 15, 14, 19),
     B.start = 6, dvar = "score")

# Example: Using new names for the dependent and the phase variables
case <- scdf(score = c(3, 4, 3, 5, 7, 4, 7, 9, 8, 10, 12),
     B.start = 5, dvar = "score", pvar = "section")

# Example: Using new names for dependent, phase, and measurement-time variables
case <- scdf(score = c(9, 8, 8, 7, 5, 7, 6, 14, 15, 12, 16),
     B.start = 7, name = "Antonia",
     dvar = "score", pvar = "section", mvar = "day")

summary(case)
```

```
## #A single-case data frame with one case
##
##         Measurements Design
## Antonia           11    A B
##
## Variable names:
## score <dependent variable>
## day <measurement-time variable>
## section <phase variable>
```

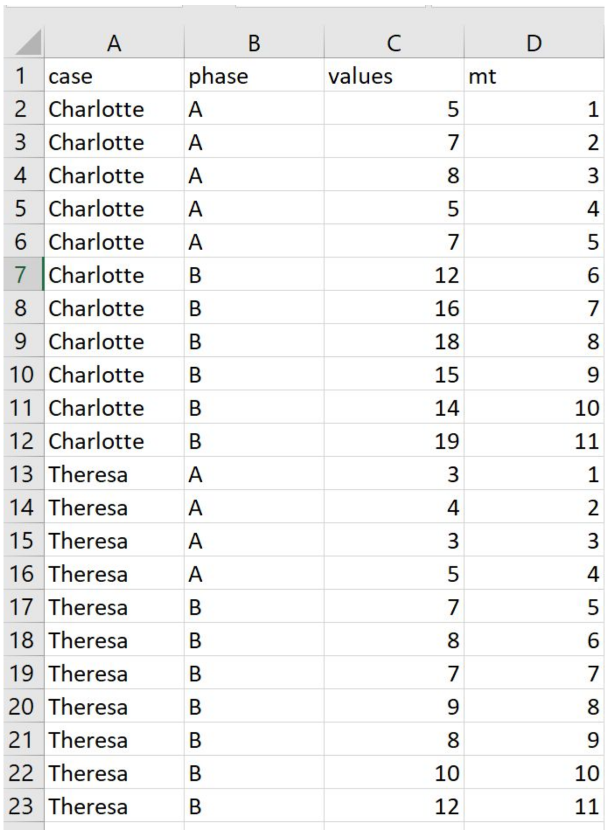
3.3 Saving and reading *single-case data frames*

Usually, it is not needed to save an scdf to a separate file on your computer. In most of the cases you could keep the coding of the *scdf* as described above and rerun it every time that you are working with your data. But sometimes it is more convenient to separately save the data to a file for later use or to send them to a colleague.
The simplest way is to use the base *R* functions `saveRDS` and `readRDS` for this purpose. `saveRDS` takes at least two arguments: the first is the object you like to save and the second is a file name for the resulting file. If you have an *scdf* with the name `study1` the line `saveRDS(study1, "study1.rds")` will save the *scdf* to your drive. You could later read this file with `study1 <- readRDS("study1.rds")`. `getwd()` will return the current active folder that you are working in.

3.4 Import and export *single-case data frames*

When you are working with other programs besides **R** you need to export and import the *scdf* into a common file format. `readSC` imports a comma-separated-variable (*csv*) file and converts it into an *scdf* object. By default, the csv-file has to contain the columns *case*, *phase*, and *values*. Optionally, a further column named *mt* could be provided. And should be build up like this:

In case your variables names differ from the standard (i.e. “case”, “values”, “phase”, and “mt”



	A	B	C	D
1	case	phase	values	mt
2	Charlotte	A	5	1
3	Charlotte	A	7	2
4	Charlotte	A	8	3
5	Charlotte	A	5	4
6	Charlotte	A	7	5
7	Charlotte	B	12	6
8	Charlotte	B	16	7
9	Charlotte	B	18	8
10	Charlotte	B	15	9
11	Charlotte	B	14	10
12	Charlotte	B	19	11
13	Theresa	A	3	1
14	Theresa	A	4	2
15	Theresa	A	3	3
16	Theresa	A	5	4
17	Theresa	B	7	5
18	Theresa	B	8	6
19	Theresa	B	7	7
20	Theresa	B	9	8
21	Theresa	B	8	9
22	Theresa	B	10	10
23	Theresa	B	12	11

Figure 3.1: How to format a single-case file in a spreadsheet program for importing into scan

), you could set additional arguments to fit your file. `readSC("example.csv", var.case = "name", ivar = "wellbeing", pvar = "intervention", mvar = "time")` for example will set the variables attributes of the resulting scdf. Cases will be split by the `"name"`, values are in the variable `"wellbeing"`, phase information in the variable `"intervention"`, and measurement times in the variable `"time"`. You could also reassign the phase names within the phase variable by setting the argument `phase.names`. Assume for example your file contains the values 0 and 1 to identify the two phases I recommend to set them to “A” and “B” with `readSC("example.csv", phase.names = c("A", "B"))`.

```
dat <- readSC.excel("example2.xlsx",cvar = "name", pvar = "intervention", dvar = "wellbeing",
```

```
## Imported 20 cases.
```

```
summary(dat)
```

```
## #A single-case data frame with 20 cases
##
##           Measurements Design
## Charles             20    A B
## Kolten              20    A B
## Annika              20    A B
## Kaysen              20    A B
## Urijah              20    A B
## Leila               20    A B
## Leia                20    A B
## Aleigha             20    A B
## Greta               20    A B
## Alijah              20    A B
## Ricardo             20    A B
## Dallas              20    A B
## Edith               20    A B
## Braylee             20    A B
## Giovanni            20    A B
## Ismael              20    A B
## Grady               20    A B
## Raina               20    A B
## Cambria             20    A B
## Lincoln             20    A B
##
## Variable names:
## intervention <phase variable>
## wellbeing <dependent variable>
## time <measurement-time variable>
## age
## gender
## gym
```

For some reasons, computer systems with a German (and some other) language setups export csv-files by default with a comma as a decimal point and a semicolon as a separator between values. In these cases you have to set to extra arguments to import the data:

```
readSC("example.csv", dec = ",", sep = ";")
```

`readSC.excel` allows for directly importing *Microsoft Excel* `.xlsx` or `.xls` files. You need to have the library `readxl` installed in your R setup for this to work.

`writeSC` exports an scdf object as a comma-separated-variables file (*csv*) which can be imported into any other software for data analyses (MS OFFICE, Libre Office etc.). The scdf object is converted into a single data frame with a *case* variable identifying the rows for each subject. The first argument of the command identifies the scdf to be exported and the second argument (`file`) the name of the resulting csv-file. If no file argument is provided a dialog box is opened to choose a file interactively. By default, writeSC exports into a standard csv-format with a dot as the decimal point and a comma for separating variables. If your system expects a comma instead of a point for decimal numbers you may use the `dec` and the `sep` arguments. For example, `writeSC(example, file = "example.csv", dec = ",", sep = ";")` exports a csv variation usually used for example in Germany.

3.5 Displaying and manipulating scdf-files

scdf are displayed by just typing the name of the object.

```
#Beretvas2008 is an example scdf included in scan
Beretvas2008
```

```
## #A single-case data frame with one case
##
##  Case1: values mt phase
##             0.7  1     A
##             1.6  2     A
##             1.4  3     A
##             1.6  4     A
##             1.9  5     A
##             1.2  6     A
##             1.3  7     A
##             1.6  8     A
##              10  9     B
##            10.8 10     B
##            11.9 11     B
##              11 12     B
##              13 13     B
##            12.7 14     B
##              14 15     B
```

The `print` command allows for specifying the output. Some possible arguments are `cases` (the number of cases to be displayed; Three by default), `rows` (the maximum number of rows to be displayed; Fifteen by default), and `digits` (number of digits). `cases = 'all'` and `rows = 'all'` prints all cases and rows.

```
#Huber2014 is an example scdf included in scan
print(Huber2014, cases = 2, rows = 10)
```

```
## #A single-case data frame with 4 cases
```

```
##
##  Adam: mt compliance phase | Berta: mt compliance phase |
##         1         25     A |          1         25     A |
##         2       20.8     A |          2       20.8     A |
##         3       39.6     A |          3       39.6     A |
##         4         75     A |          4         75     A |
##         5         45     A |          5         45     A |
##         6       39.6     A |          6       14.6     A |
##         7       54.2     A |          7       45.8     A |
##         8         50     A |          8       33.3     A |
##         9       28.1     A |          9       31.3     A |
##        10         40     A |         10       32.5     A |
## # ... up to 66 more rows
## #  2 more cases
```

The argument `long = TRUE` prints each case one after the other instead side by side (e.g., `print(exampleAB, long = TRUE)`).

`summary()` gives a very concise overview of the *scdf*

```
summary(Huber2014)
```

```
## #A single-case data frame with 4 cases
##
##            Measurements Design
## Adam                 37    A B
## Berta                29    A B
## Christian            76    A B
## David                76    A B
##
## Variable names:
## mt <measurement-time variable>
## compliance <dependent variable>
## phase <phase variable>
##
##
## Note:  Behavioral data (compliance in percent).
## Author of data:  Christian Huber
```

You can extract one or more single-cases from an *scdf* with multiple cases in two ways. If the case has a name, you can address it with the $ operator.

```
Huber2014$David
```

or you can use squared brackets

```
Huber2014[1] #extracts case 1
Huber2014[2:3] #extracts cases 2 and 3
```

```
new.huber2014 <- Huber2014[c(1, 4)] #extracts cases 1 and 4
print(new.huber2014)
```

```
## #A single-case data frame with 2 cases
```

```
##
##  Adam: mt compliance phase | David: mt compliance phase |
##         1         25     A |          1       65.6     A |
##         2       20.8     A |          2       37.5     A |
##         3       39.6     A |          3       58.3     A |
##         4         75     A |          4       72.9     A |
##         5         45     A |          5       33.3     A |
##         6       39.6     A |          6       59.4     A |
##         7       54.2     A |          7       77.1     A |
##         8         50     A |          8       54.2     A |
##         9       28.1     A |          9       68.8     A |
##        10         40     A |         10       43.8     A |
##        11       52.1     B |         11       62.5     B |
##        12       31.3     B |         12       64.6     B |
##        13       15.6     B |         13       60.4     B |
##        14       29.2     B |         14       81.3     B |
##        15       43.8     B |         15       79.2     B |
## # ... up to 61 more rows
```

Chapter 4

Creating a single-case data plot

Plotting the data is a first important approach of analyzing. After you build an *scdf* the `plot` command helps to visualize the data. When the `scdf` includes more than one case a multiple baseline figure is provided. Labels of the x- and y-axes can be changed with the `xlab` and `ylab` argument. The names of the phases can be changed with the `phase.names` argument. The x- and y-scaling of the graphs are by default calculated as the minimum and the maximum of all included single cases. The `xlim` and the `ylim` argument are used to set specific values. The argument takes a vector of two numbers. The first for the lower and the second for the upper limit of the scale. In case of multiple single cases an `NA` sets the individual minimum or maximum for each case. Assume for example the study contains three single cases `ylim = c(0, NA)` will set the lower limit for all three single cases to `0` and the upper limit individually at the maximum of each case.

Table 4.1 gives an overview of all available arguments.

Extra lines can be added to the plot using the `lines` argument. The lines argument takes several separate sub-arguments which have to be provided in a list. In its most simple form this list contains one element. `lines = list('median')` adds a line with the median of each phase to the plot. Additional arguments like `col` or `lwd` help to format these lines. For adding red thick median lines use the command `lines = list('median', col = 'red', lwd = '2')`.

Specific data points can be highlighted using the `marks` argument. A `list` defines the measurement times to be marked, the marking color and the size of the marking. `marks = list(position = c(1,5,6))` marks the first, fifth, and sixth measurement time. If the *scdf* contains more than one data-set marking would be the same for all data sets in this example. In case you define a `list` Containing vectors, marking can be individually defined for each data set. Assume, for example, we have an *scdf* comprising three data sets, then `marks = list(position = list(c(1,2), c(3,4), c(5,6)))` will highlight measurement times one and two for the first data set, three and four for the second and five and six for the third. `pch`, `col` and `cex` define symbol, colour and size of the markings.

```r
# plot with marks in a red circles 2.5 times larger than the standard symbol
# size. exampleAB is an example scdf included in the scan package
marks <- list(
  positions = list( c(8, 9), c(17, 19), c(7, 18) ),
  col = 'red', cex = 2.5, pch = 1
```

Table 4.1: Arguments of the plot function

Argument	What it does ...
data	A single-case data frame.
ylim	Lower and upper limits of the y-axis
xlim	Lower and upper limits of the x-axis.
style	A specific design for displaying the plot.
lines	A character or list defining one or more lines or curves to be plotted.
marks	A list of parameters defining markings of certain data points.
main	A figure title
phase.names	By default phases are labeled as given in the phase variable. Use this argument to specify different labels: ‘phase.names = c('Baseline', 'Intervention')‘.
case.names	Case names. If not provided, names are taken from the scdf or left blank if the scdf does not contain case names.
xlab	The label of the x-axis. The default is taken from the name of the measurement variable as provided by the scdf.
ylab	The labels of the y-axis. The default is taken from the name of the dependent variable as provided by the scdf.
xinc	An integer. Increment of the x-axis. 1 : each mt value will be printed, 2 : every other value, 3 : every third values etc.
...	Further arguments passed to the generic plot command.

```
)
plot(exampleAB, marks = marks, style = "sienna")
```

4.1 Graphical styles of a plot

The `style` argument of the plot function allows to specify a specific design of a plot. By default, the `grid` style is applied. `scan` includes some further predefined styles. `default, yaxis, tiny, small, big, chart, ridge, annotate, grid, grid2, dark, nodot, and sienna`. The name of a style is provided as a character string (e.g., `style = "grid"`).
Some styles only address specific elements (e.g., “small” or “tiny” just influence text and line sizes). These styles lend themselves to be combined with other style. This could be achieved by providing several style names to the plot argument: `style = c("grid", "annotate", "small")`.
Beyond predefined styles, styles can be individually modified and created. New styles are provided as a `list` of several design parameters that are passed to the `style` argument of the `plot` function. Table 4.3 shows all design parameter that could be defined.
To define a new style, first create a list containing a plain design. The `style_plotSC` function returns such a list with the default values for a plain design (e.g., `mystyle <- style_plotSC()`). Single design parameters can now be set by assigning a specific value within the list. For example, `newstyle$fill <- "grey90"` will set the `fill` parameter to `"grey90"`. Alternatively, changes to the plain design can already by defined within the `style_plotSC` function. To set a light-blue background color and also an orange grid, create the style `style_plotSC(fill.bg`

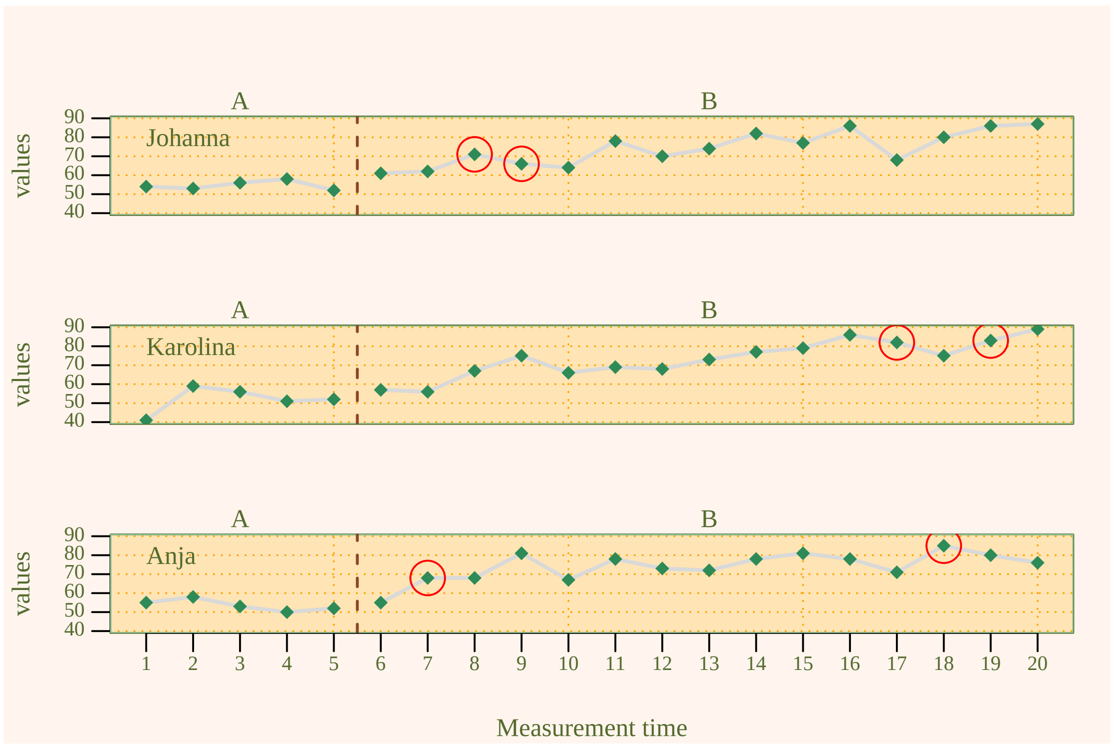


Figure 4.1: Example of a plot with highlighted data-points

Table 4.2: Values of the lines argument

Value	What it does ...
median	separate lines for the medians of each phase
mean	separate lines for the means of each phase. By default it is 10%-trimmed. Other trims can be set using a second parameter (e.g., 'lines = list(mean = 0.2)' draws a 20%-trimmed mean line).
trend	Separate lines for the trend of each phase.
trendA	Trend line for phase A, extrapolated throughout the other phases
maxA	Line at the level of the highest phase A score.
minA	Line at the level of the lowest phase A score.
medianA	Line at the phase A median score.
meanA	Line at the phase A 10%-trimmed mean score. Apply a different trim, by using the additional argument (e.g., 'lines = list(meanA = 0.2)').
movingMean	Draws a moving mean curve, with a specified lag: ''lines = list(movingMean = 2)''. Default is a lag 1 curve.
movingMedian	Draws a moving median curve, with a specified lag: ''lines = list(movingMedian = 3).' Default is a lag 1 curve.
loreg	Draws a non-parametric local regression line. The proportion of data influencing each data point can be specified using 'lines = list('loreg' = 0.66)'. The default is 0.5.
lty	Line type. Examples are: 'solid','dashed', 'dotted'.
lwd	Line thickness, e.g., 'lwd = 4'.
col	Line colour, e.g., 'col = 'red''.

`= "lightblue", grid = "orange")`. If you do not want to start with the plain design but a different of the predefined styles, set the `style` argument. If, for example, you like to have the `grid` combined with the `big` style but want to change the color of the grid to orange type `style_plotSC(style = c("grid", "big"), col.grid = "orange")`. `plot(mydata, style = mystyle)` will apply the new style in a plot. Please note that the new style is not passed in quotation marks.

The width of the lines are set with the `lwd` argument, `col` is used to set the line colour and `pch` sets the symbol for a data point. The `pch` argument can take several values for defining the symbol in which data points are plotted.

Here is an example customizing a plot with several additional graphic parameters

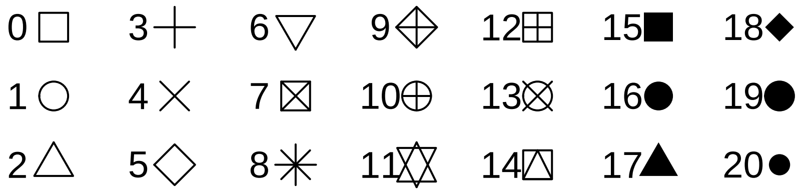


Figure 4.2: Some of the possible symbols and their pch values.

Table 4.3: Arguments of the style plotSC function

Argument	What it does ...
fill	If TRUE area under the line is filled.
col.fill	Sets the color of the area under the line.
grid	If TRUE a grid is included.
col.grid	Sets the color of the grid.
lty.grid	Sets the line type of the grid.
lwd.grid	Sets the line thikness of the grid.
fill.bg	If not NA the backgorund of the plot is filled with the given color.
annotations	A list of parameters defining annotations to each data point. This adds the score of each MT to your plot. ‘’pos’‘ Position of the annotations: 1 = below, 2 = left, 3 = above, 4 = right. ‘’col’‘ Color of the annotations. ‘’cex’‘ Size of the annotations. ‘’round’‘ rounds the values to the specified decimal. ‘annotations = list(pos = 3, col = ’brown’, round = 1)‘ adds scores rounded to one decimal above the data point in brown color to the plot.
text.ABlag	By default a vertical line separates phases A and B in the plot. Alternatively, you could print a character string between the two phases using this argument: ‘text.ABlag = ’Start’‘.
lwd	Width of the plot line. Default is ‘lwd = 2‘.
pch	Point type. Default is ‘pch = 17‘ (triangles). Other options are for example: 16 (filled circles) or ’A’ (uses the letter A).
col.lines	The color of the lines. If set to an empty string no lines are drawn.
col.dots	The color of the dots. If set to an empty string no dots are drawn.
mai	Sets the margins of the plot.
...	Further arguments passed to the plot command.

```
newstyle <- style_plotSC(fill = "grey95", grid = "lightblue", pch = 16)

plot(exampleABAB, style = newstyle)
```

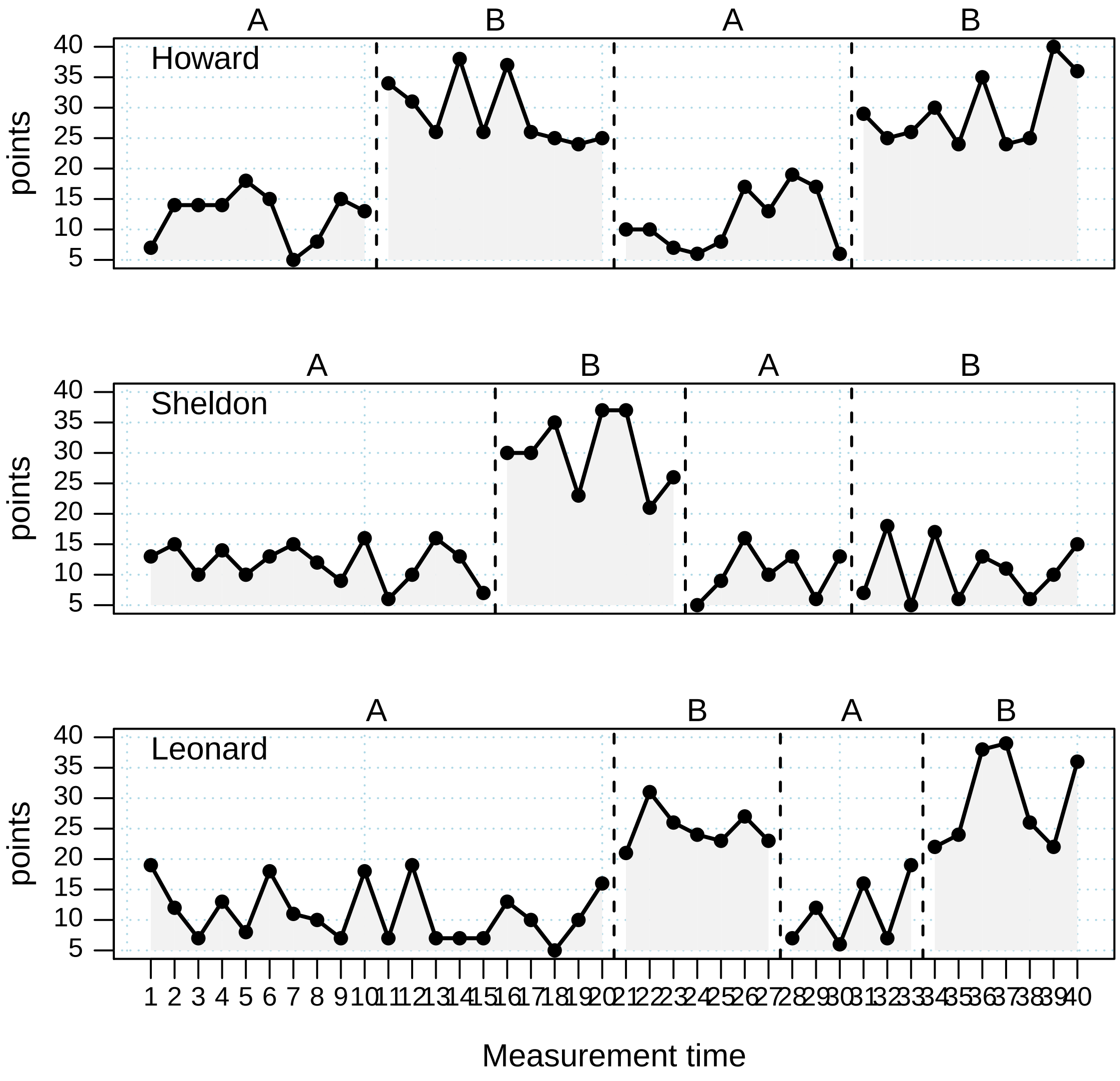


Figure 4.3: A plot with a customized style.

Chapter 5

Describe and manipulate single-case data frames

5.1 Describing and summarizing

A short description of the *scdf* is provided by the `summary` command. The results are pretty much self explaining

```
summary(Huber2014)
```

```
## #A single-case data frame with 4 cases
##
##            Measurements Design
## Adam                 37    A B
## Berta                29    A B
## Christian            76    A B
## David                76    A B
##
## Variable names:
## mt <measurement-time variable>
## compliance <dependent variable>
## phase <phase variable>
##
##
## Note:  Behavioral data (compliance in percent).
## Author of data:  Christian Huber
```

`describeSC` is the basic command to get an overview on descriptive statistics. As an argument it only takes the name of the *scdf* object. For each case of the *scdf* and each phase within a case descriptive statistics are provided. The output table contains statistical indicators followed by a dot and the name of the phase (e.g., `n.A` for the number of measurements of phase A).

Table 5.1: Statistics of the describeSC command

Parameter	What it means …
n	Number of measurements.
mis	Number of missing values.
m	Mean values.
md	Median of values.
sd	Standard deviation of values.
mad	Median average deviation of values.
min/max	Min and max of values.
trend	Slope of a regression line through values by time.

```
describeSC(exampleABC)
```

```
## Describe Single-Case Data
##
## Design:  A B C
##
##          Marie Rosalind Lise
##      n.A    10       15   20
##      n.B    10        8    7
##      n.C    10        7    3
##    mis.A     0        0    0
##    mis.B     0        0    0
##    mis.C     0        0    0
##
##          Marie Rosalind   Lise
##      m.A 52.00    52.27  52.35
##      m.B 72.10    73.25  73.57
##      m.C 68.00    66.43  71.33
##     md.A 53.50    52.00  52.00
##     md.B 72.50    72.00  73.00
##     md.C 69.00    68.00  76.00
##     sd.A  8.29     8.15  10.87
##     sd.B 11.37    13.13  10.64
##     sd.C 12.70    10.49  21.39
##    mad.A 11.12     7.41  10.38
##    mad.B 10.38    10.38  16.31
##    mad.C 17.79    11.86  20.76
##    min.A 39.00    37.00  35.00
##    min.B 47.00    54.00  60.00
##    min.C 51.00    52.00  48.00
##    max.A 63.00    65.00  74.00
##    max.B 85.00    97.00  87.00
##    max.C 87.00    78.00  90.00
##  trend.A -1.92     0.50  -0.09
##  trend.B -0.61     0.64   1.93
##  trend.C -0.19    -2.93 -14.00
```

The resulting table could be exported into a csv file to be used in other software (e.g., to inserted in a word processing document). Therefore, first write the results of the `describeSC` command into an R object and then use the `write.csv` (or `write.csv2` for a German OS system setup) to export the `descriptives` element of the object.

```
# write the results into a new R object named `res`
res <- describeSC(exampleABC)
# create a new file containing the descriptives on your harddrive
write.csv(res$descriptives, file = "descriptive data.csv")
```

The file is written to the currently active working directory. If you are not sure where that is, type `getwd()` (you can use the `setwd()` command to define a different working directory. To get further details type `help(setwd)` into R).

5.2 Autoregression and trendanalyses

The `autocorrSC` function calculates autocorrelations within each phase and across all phases. The `lag.max` argument defines the lag up to which the autocorrelation will be computed.

```
autocorrSC(exampleABC, lag.max = 4)
```

```
## Autocorrelations
##
##       case phase lag_1 lag_2 lag_3 lag_4
##      Marie     A  0.29 -0.11  0.10  0.12
##      Marie     B -0.28 -0.10 -0.14 -0.09
##      Marie     C  0.00 -0.33 -0.14 -0.25
##      Marie   all  0.21  0.10  0.25  0.12
##   Rosalind     A  0.37 -0.29 -0.33 -0.34
##   Rosalind     B -0.34  0.24 -0.40  0.04
##   Rosalind     C -0.07 -0.32  0.27  0.02
##   Rosalind   all  0.49  0.38  0.22  0.17
##       Lise     A  0.04 -0.32 -0.05 -0.09
##       Lise     B -0.63  0.50 -0.40  0.31
##       Lise     C -0.38 -0.12    NA    NA
##       Lise   all  0.33  0.36  0.23  0.27
```

The `trendSC` function provides an overview of linear trends in single-case data. By default, it gives you the intercept and slope of a linear and a squared regression of measurement-time on scores. Models are computed separately for each phase and across all phases. For a more advanced application, you can add regression models using the R specific formula class.

```
# Simple example
trendSC(exampleABC[1])
```

```
## Trend for each phase
##
##             Intercept      B   Beta
## Linear.ALL     55.159  0.612  0.392
## Linear.A       60.618 -1.915 -0.700
## Linear.B       74.855 -0.612 -0.163
## Linear.C       68.873 -0.194 -0.046
## Squared.ALL    59.135  0.017  0.330
## Squared.A      57.937 -0.208 -0.712
## Squared.B      73.217 -0.039 -0.098
## Squared.C      68.490 -0.017 -0.038
##
## Note. Measurement-times of phase B start at 0
```

```
# Complex example
trendSC(exampleAB$Johanna, offset = 0, model = c("Cubic" = values ~ I(mt^3), "Log Time" = valu
```

```
## Trend for each phase
##
##               Intercept      B   Beta
## Linear.ALL       50.484  1.787  0.908
## Linear.A         54.300  0.100  0.066
## Linear.B         61.133  1.625  0.813
## Squared.ALL      57.879  0.079  0.871
## Squared.A        54.747 -0.013 -0.054
## Squared.B        66.343  0.094  0.775
## Cubic.ALL        60.886  0.004  0.816
## Cubic.A          54.959 -0.008 -0.169
## Cubic.B          68.368  0.006  0.732
## Log Time.ALL     43.532 12.149  0.848
## Log Time.A       54.032  0.593  0.156
## Log Time.B       57.300  9.051  0.791
##
## Note. Measurement-times of phase B start at 1
```

5.3 Missing values

There are two kinds of missing values in single-case data series. First, missings that were explicitly recorded as `NA` and assigned to a phase and measurement-time as in the following example:

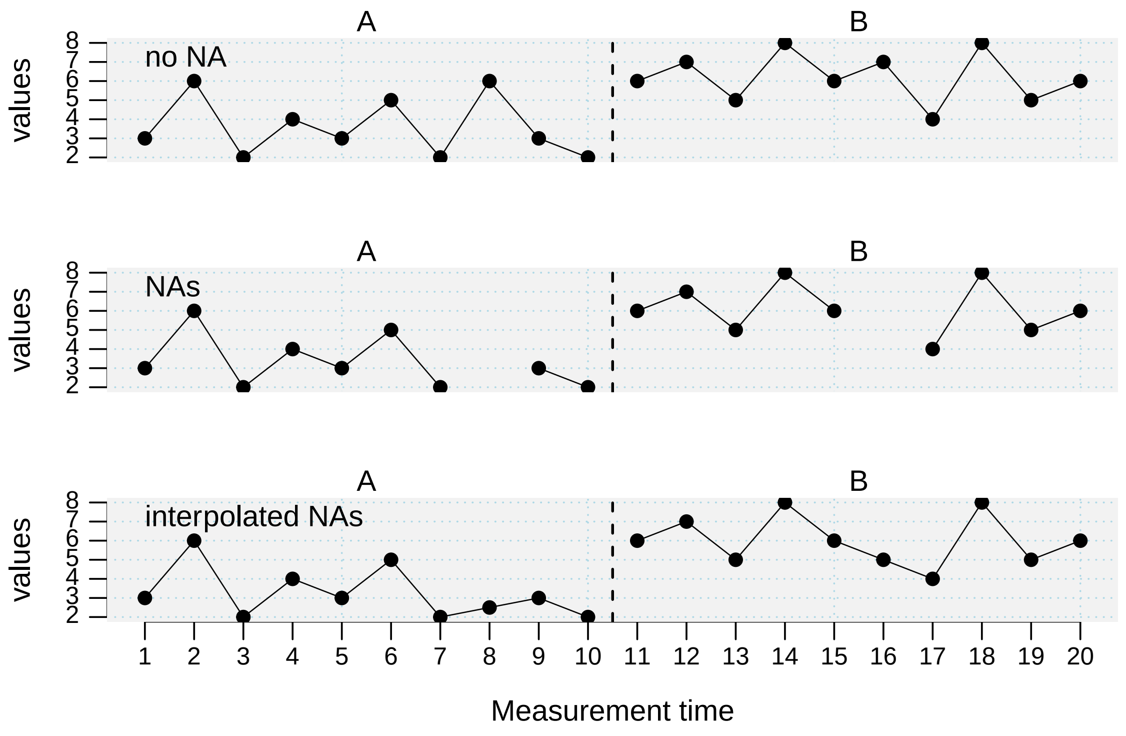
```
scdf(c(5, 3, 4, 6, 8, 7, 9, 7, NA, 6), phase.design = c(A = 4, B = 6))
```

The second type of missing occures when there are gaps between measuremnt-times that are not explicitly coded as in the following example:

```
scdf(c(5, 3, 4, 6, 8, 7, 9, 7, 6), phase.design = c(A = 4, B = 5),
     mt = c(1, 2, 3, 4, 5, 6, 7, 8, 10))
```

In both cases, missing values pose a threat to the internal validity of overlap indices. Randomization tests are more robust against the first type of missing values but are affected by the second type. Regression approaches are less impacted by both types as they take the measurement-time into account.

```
case1  <- scdf(c(3,6,2,4,3,5,2,6,3,2, 6,7,5,8,6,7,4,8,5,6),
               phase.design = c(A = 10, B = 10), name = "no NA")
case2 <- scdf(c(3,6,2,4,3,5,2,NA,3,2, 6,7,5,8,6,NA,4,8,5,6),
               phase.design = c(A = 10, B = 10), name = "NAs")
case3 <- fillmissingSC(case2)
names(case3) <- "interpolated NAs"
ex <- c(case1, case2, case3)
plot(ex, style = "grid")
```



```
overlapSC(ex)
```

```
## Overlap Indices
##
## Design:  A B
## Comparing phase 1 against phase 2
##
##                 no NA    NAs interpolated NAs
## PND             40.00  33.33            30.00
## PEM            100.00 100.00           100.00
## PET            100.00 100.00           100.00
## NAP             88.50  91.36            91.50
## NAP.rescaled    77.00  82.72            83.00
## PAND            72.50  80.56            80.00
## TAU_U            0.42   0.39             0.47
## Base_Tau         0.59   0.64             0.64
## Diff_mean        2.60   2.78             2.75
## Diff_trend       0.02   0.11             0.12
## SMD              1.65   1.96             2.02
```

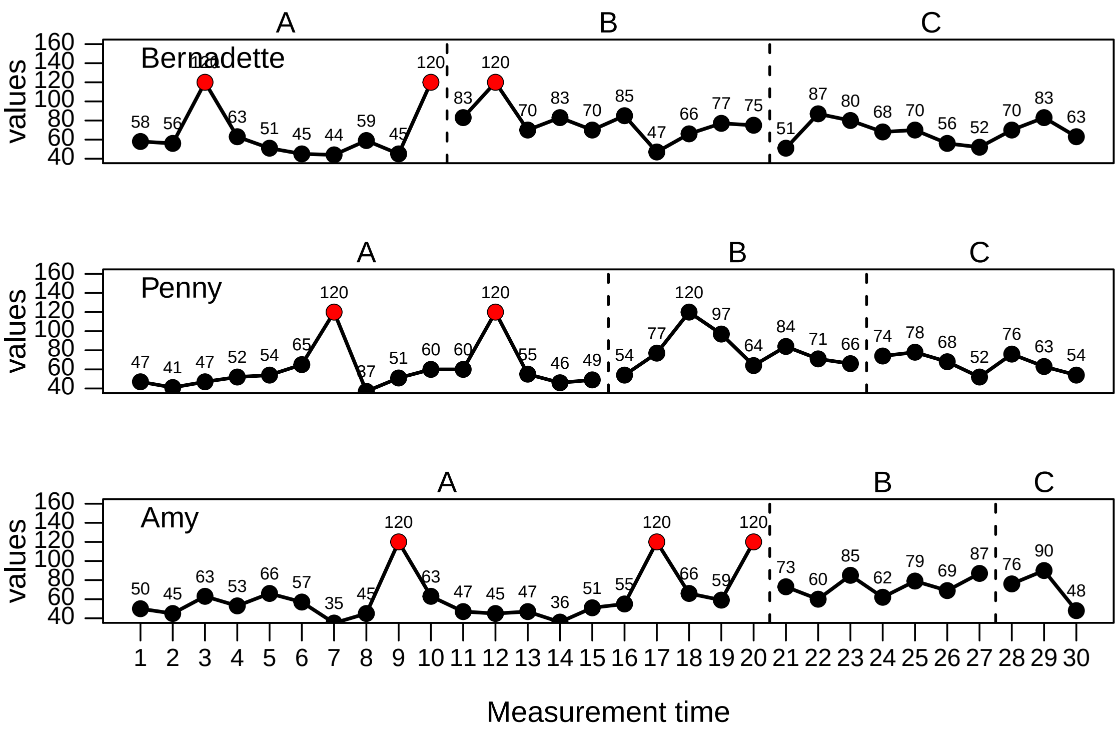
5.4 Outlieranalysis

scan provides several methods for analyzing outliers. All of them are implemented in the `outliersSC` function. Available methods are the **standard deviation**, **mean average deviation**, **confidence intervalls**, and **Cook's distance**. The criteria argument takes a vector with two information, the first defines the analyzing method ("SD", "MAD", CI","Cook") and the second the criteria. For"SD" the criteria is the number of standard deviations (**sd**) from the mean of each phase for which a value is not considered to be an outlier. For example, `criteria = c("SD",2)` would identify every value exceeding two **sd** above or below the mean as an outlier whereas **sd** and mean refer to phase of a value. As this might be misleading particularly for small samples Iglewicz and Hoaglin Iglewicz & Hoaglin (1993) recommend the use the much more robust median average deviation (**MAD**) instead. The **MAD** is is constructed similar to the **sd** but uses the median instead of the mean. Multiplying the **MAD** by 1.4826 approximates the **sd** in a normal distributed sample. This corrected MAD is applied in the `outlierSC` function. A deviation of 3.5 times the corrected **MAD** from the median is suggested to be an outlier. To use this criterion set `criteria = c("MAD", 3.5)`. `criteria = c("CI", 0.95)` takes exceeding the 95% confidence interval as the criteria for outliers. The Cook's distance method for calculation outliers can be applied with a strict AB-phase design. in that case, the Cook's distance analyses are based on a piecewise-regression model. Most commonly, Cook's distance exceeding 4/n is used as a criteria. This could be implemented setting 'criteria = c("Cook","4/n").

```
outlierSC(exampleABC_outlier, criteria = c("MAD", 3.5))
```

```
## Outlier Analysis for Single-Case Data
##
## Criteria: Exceeds 3.5 Mean Average Deviations
##
## $Bernadette
##   phase md mad    lower    upper
## 1     A 57   9 10.2981 103.7019
## 2     B 76   7 39.6763 112.3237
## 3     C 69  12  6.7308 131.2692
##
## $Penny
##   phase md mad    lower    upper
## 1     A 52   6 20.8654  83.1346
## 2     B 74  10 22.1090 125.8910
## 3     C 68   8 26.4872 109.5128
##
## $Amy
##   phase md mad    lower    upper
## 1     A 54   9  7.2981 100.7019
## 2     B 73  11 15.9199 130.0801
## 3     C 76  14  3.3526 148.6474
##
## Case Bernadette : Dropped 3
## Case Penny : Dropped 2
## Case Amy : Dropped 3
```

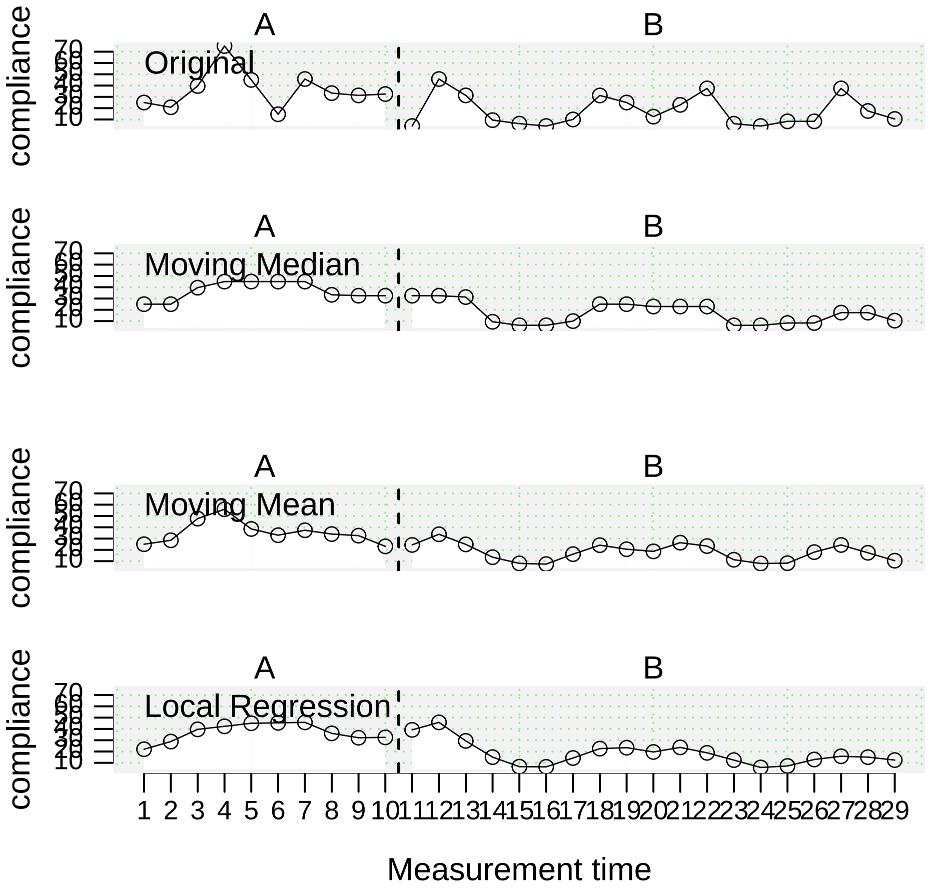
```r
# Visualizing outliers with the plot function
res <- outlierSC(exampleABC_outlier, criteria = c("MAD", 3.5))
plot(exampleABC_outlier, marks = res,
     style = "annotate", ylim = c(40,160))
```



5.5 Smoothing data

The smoothSC function provides procedures to smooth single-case data and eliminate noise. A moving average function (mean- or median-based) replaces each data point by the average of the surrounding data points step-by-step. A *lag* defines the number of measurements before and after the calculation is based on. So a lag-1 will take the average of the proceeding and following value and lag-2 the average of the two proceeding and two following measurements. With a local regression function, each data point is regressed by its surrounding data points. Here, the proportion of measurements surrounding a value is usually defined. So an intensity of 0.2 will take the surrounding 20% of data as the basis for a regression.
The function returns am scdf with smoothed data points.

```r
## Use the three different smoothing functions and compare the results
berta_mmd <- smoothSC(Huber2014$Berta)
berta_mmn <- smoothSC(Huber2014$Berta, FUN = "movingMean")
berta_lre <- smoothSC(Huber2014$Berta, FUN = "localRegression")
new.study <- c(Huber2014$Berta, berta_mmd, berta_mmn, berta_lre)
names(new.study) <- c("Original","Moving Median","Moving Mean","Local Regression")
plotSC(new.study, style = "grid2")
```


A
B
Original
Moving Median
Moving Mean
Local Regression
compliance
10
20
30
40
50
60
70
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
Measurement time

Chapter 6

Overlapping indices

`overlapSC` provides a table with some of the most important overlap indices for each case of an *scdf*. For calculating overlap indicators is is important to know if a decrease or an increase of values is expected between phases. By default `overlapSC` assumes an increase in values. If the argument `decreasing = TRUE` is set, calculation will be based on the assumption of decreasing values.

```
overlapSC(exampleAB)
```

```
## Overlap Indices
##
## Design:  A B
## Comparing phase 1 against phase 2
##
##              Johanna Karolina   Anja
## PND           100.00    86.67  93.33
## PEM           100.00   100.00 100.00
## PET           100.00    93.33 100.00
## NAP           100.00    96.67  98.00
## NAP.rescaled  100.00    93.33  96.00
## PAND          100.00    90.00  90.00
## TAU_U           0.76     0.78   0.63
## Base_Tau        0.63     0.59   0.61
## Diff_mean      19.53    21.67  20.47
## Diff_trend      1.53     0.54   2.50
## SMD             8.11     3.17   6.71
```

Overlap measures refer to a comparison of two phases within a single-case data-set. By default, overlapSC compares a Phase A to a Phase B. The `phases` argument is needed if the phases of the *scdf* do not include phases named A and B or a comparison between other phases in wanted. The `phases` argument takes a list with two elements. One element for each of the two phases that should be compared. The elements could contain either the name of the two phases or the number of the position within the *scdf*. If you want to compare the first to the third phase you can set `phases = list(1,3)`. If the phases of your case are named 'A', 'B', and 'C' you could alternatively set `phases = list("A","C")`.

It is also possible to compare a combination of several cases against a combination of other phases. Each of the two list-elements could contain more than one phase which are concatenated with the `c` command. For example if you have an ABAB-Design and like to compare the two A-phases against the two B-phases `phases = list( c(1,3), c(2,4) )` will do the trick.

```
overlapSC(exampleA1B1A2B2, phases = list( c("A1","A2"), c("B1","B2")))
```

```
## Overlap Indices
##
## Design:  A1 B1 A2 B2
## Comparing phases 1 + 3 against phases 2 + 4
##
##                Pawel Moritz Jannis
## PND            55.00  77.78  71.43
## PEM           100.00 100.00 100.00
## PET           100.00 100.00 100.00
## NAP            94.50  96.84  98.35
## NAP.rescaled   89.00  93.69  96.70
## PAND           82.50  85.00  90.00
## TAU_U           0.44   0.45   0.37
## Base_Tau        0.65   0.67   0.68
## Diff_mean      12.25  13.58  15.27
## Diff_trend     -0.05   0.00  -0.54
## SMD             2.68   3.27   3.62
```

6.1 Percentage non-overlapping data (PND)

The percentage of non-overlapping data (PND) effect size measure was described by Scruggs, Mastropieri, & Casto (1987) . It is the percentage of all data-points of the second phase of a single-case study exceeding the maximum value of the first phase. In case you have a study where you expect a decrease of values in the second phase, PND is calculated as the percentage of data-point of the second phase below the minimum of the first phase.

The function `pnd` provides the PND for each case as well as the mean of all PNDs of that *scdf*. When you expect decreasing values set `decreasing = TRUE`. When there are more than two phases or phases are not named A and B, use the `phases` argument as described at the beginning of this chapter.

```
pnd(exampleAB)
```

```
## Percent Non-Overlapping Data
##
##      Case    PND Total Exceeds
##   Johanna   100%    15      15
##  Karolina 86.67%    15      13
##      Anja 93.33%    15      14
##
## Mean  : 93.33 %
```

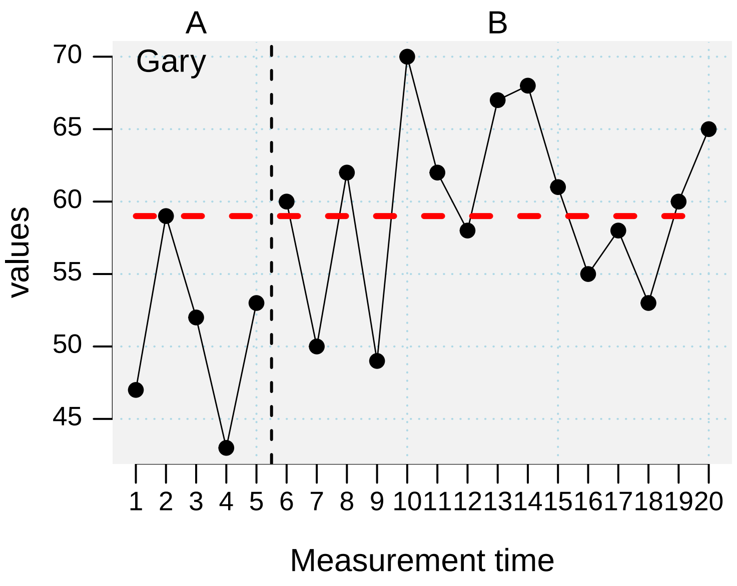


Figure 6.1: Illustration of PND. PND is 60% as 9 out of 15 datapoints of phase B are higher than the maximum of phase A.

6.2 Percentage exceeding the median (PEM)

The pem function returns the percentage of phase B data exceeding the phase A median. Additionally, a binomial test against a 50/50 distribution is computed. Different measures of central tendency can be addressed for alternative analyses.

```
pem(exampleAB)
```

```
## Percent Exceeding the Median
##
##          PEM positives total binom.p
## Johanna  100        15    15       0
## Karolina 100        15    15       0
## Anja     100        15    15       0
##
## Alternative hypothesis: true probability > 50%
```

6.3 Percentage exceeding the regression trend (PET)

The pet function provides the percentage of phase B data points exceeding the prediction based on the phase A trend. A binomial test against a 50/50 distribution is computed. Furthermore, the percentage of phase B data points exceeding the upper (or lower) 95 percent confidence interval of the predicted progress is computed.

```
pet(exampleAB)
```

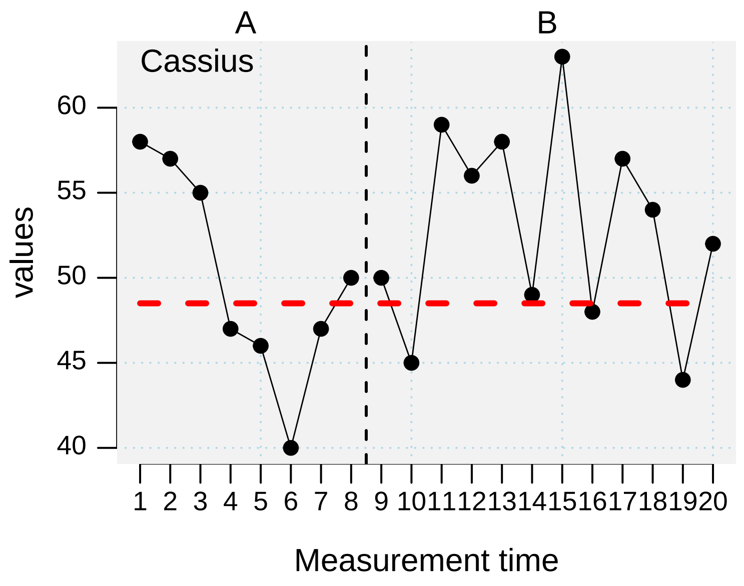


Figure 6.2: Illustration of PEM. PEM is 75% as 9 out of 12 datapoints of phase B are higher than the median of phase A.

```
## Percent Exceeding the Trend
##
## N cases =  3
##
##                PET binom.p  PET CI
## Johanna  100.000       0  86.667
## Karolina  93.333       0   0.000
## Anja     100.000       0 100.000
##
## Binom.test: alternative hypothesis: true probability > 50%
## PET CI: Percent of values greater than upper 95% confidence threshold (greater 1.645*se ab
```

6.4 Percentage of all non-overlapping data (PAND)

The `pand` function calculates the percentage of all non-overlapping data (Parker, Hagan-Burke, & Vannest, 2007), an index to quantify a level increase (or decrease) in performance after the onset of an intervention. The argument `correction = TRUE` makes `pand` use a frequency matrix, which is corrected for ties. A tie is counted as the half of a measurement in both phases. Set `correction = FALSE` to use the uncorrected matrix, which is not recommended.

```
pand(exampleAB)
```

```
## Percentage of all non-overlapping data
##
## PAND =  93.3 %
## Φ =  0.822  ; Φ² =  0.676
```

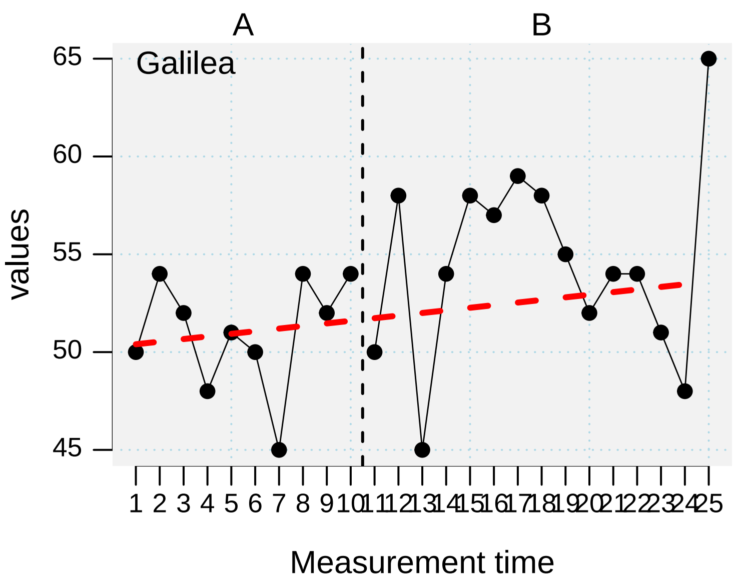


Figure 6.3: Illustration of PET. PET is 66.7% as 10 out of 15 datapoints of phase B are higher than the median of phase A.

```
##
## Number of Cases: 3
## Total measurements: 60  (in phase A: 15; in phase B: 45)
## n overlapping data per case: 0, 2, 2
## Total overlapping data: n = 4 ; percentage = 6.7
##
## 2 x 2 Matrix of proportions
##  % expected
##  A   B   total
## %    A   21.7    3.3 25
## real B   3.3 71.7    75
##  total   25  75
##
## 2 x 2 Matrix of counts
##  expected
##  A   B   total
##      A   13  2   15
## real B   2   43  45
##  total   15  45
##
##
## Note. Matrix is corrected for ties
##
## Correlation based analysis:
##
## z = 6.316, p = 0.000,   = 0.822
```

PAND indicates nonoverlap between phase A and B data (like PND), but uses all data and is therefore not based on one single (probably unrepresentative) datapoint. Furthermore, PAND allows the comparison of real and expected associations (Chi-square test) and estimation of the effect size Phi, which equals Pearsons r for dichotomous data. Thus, phi-Square is the amount of explained variance. The original procedure for computing PAND does not account for ambivalent datapoints (ties). The newer NAP overcomes this problem and has better precision-power (Richard I Parker et al., 2011).

6.5 Nonoverlap of all pairs (NAP)

The `nap` function calculates the nonoverlap of all pairs (Parker & Vannest, 2009). NAP summarizes the overlap between all pairs of phase A and phase B data points. If an increase of phase B scores is expected, a non-overlapping pair has a higher phase B data point. The NAP equals number of pairs showing no overlap / number of pairs. Because NAP can only take values between 50 and 100 percent, a rescaled and therefore more intuitive NAP (0-100%) is also displayed. NAP is equivalent to the the U-test and Wilcox rank sum test. Thus, a Wilcox test is conducted and reported for each case.

```
nap(exampleAB)
```

```
## Nonoverlap of All Pairs
##
##      Case NAP Rescaled Pairs Positives Ties   W       p
##   Johanna 100      100    75        75    0 0.0 0.00062
##  Karolina  97       93    75        72    1 2.5 0.00129
##      Anja  98       96    75        73    1 1.5 0.00095
```

6.6 Tau-U

The `tauUSC` function takes a *scdf* and returns Tau-U calculations for each single-case within that file. Additionally, an overall Tau-U value is calculated for all cases. The overall Tau-U value is the average of all Tau-U values weighted by their standard error. This procedure has been proposed by Richard I. Parker et al. (2011).
The `tauUSC` function provides two methods for calculation of Tau-U. By setting the argument `method = "parker"`, Tau-U is calculated as described in Parker et al. (2011). This procedure could lead to Tau-U values above 1 and below -1 which are difficult to interpret. `method = "complete`, which is the default, applies a correction that keeps the values within the -1 to 1 range and should be more appropriate.
In the method proposed by Parker et al. (2011) data of the same value (ties) are ignored and treated as if these data had not been available. Alternatively, ties might be considered in favor of the intervention or not. For the latter, the `ties.method` argument could be set to `ties.method = "positive"` or `ties.method = "negative"`. The default is `"omit"` following Parker et al. (2011).
The standard return of the `tauUSC` function does not display all calculations. If you like to have more details, apply the `print` function with the additional argument `complete = TRUE`.

```
dat <- scdf(c(2,0,1,4,3,5,9,7,8), phase.design = c(A = 4, B = 5))
res <- tauUSC(dat)
print(res, complete = TRUE)
```

```
## Tau-U
## Method:  complete
##
## Overall Tau-U:
## A vs. B + Trend B - Trend A           A vs. B - Trend A
##                   0.6111111                   0.6153846
##
## $Case1
##                             pairs pos neg ties  S      D         b    SD
## A vs. B                        20  19   1    0 18 20.000 0.900 0.900 8.165
## Trend A                         6   4   2    0  2  6.000 0.333 0.333 2.944
## Trend B                        10   8   2    0  6 10.000 0.600 0.600 4.082
## Trend B - Trend A              16  10   6    0  4 33.941 0.250 0.118 9.381
## A vs. B - Trend A              26  21   5    0 16 30.594 0.615 0.523 8.679
## A vs. B + Trend B              30  27   3    0 24 32.863 0.800 0.730 9.129
## A vs. B + Trend B - Trend A    36  29   7    0 22 36.000 0.611 0.611 9.592
##                                VAR     Z     p SE.Tau.b
## A vs. B                     66.667 2.205 0.027    0.408
## Trend A                      8.667 0.679 0.497    0.491
## Trend B                     16.667 1.470 0.142    0.408
## Trend B - Trend A           88.000 0.426 0.670    0.276
## A vs. B - Trend A           75.333 1.843 0.065    0.284
## A vs. B + Trend B           83.333 2.629 0.009    0.278
## A vs. B + Trend B - Trend A 92.000 2.294 0.022    0.266
```

```
tauUSC(exampleAB)
```

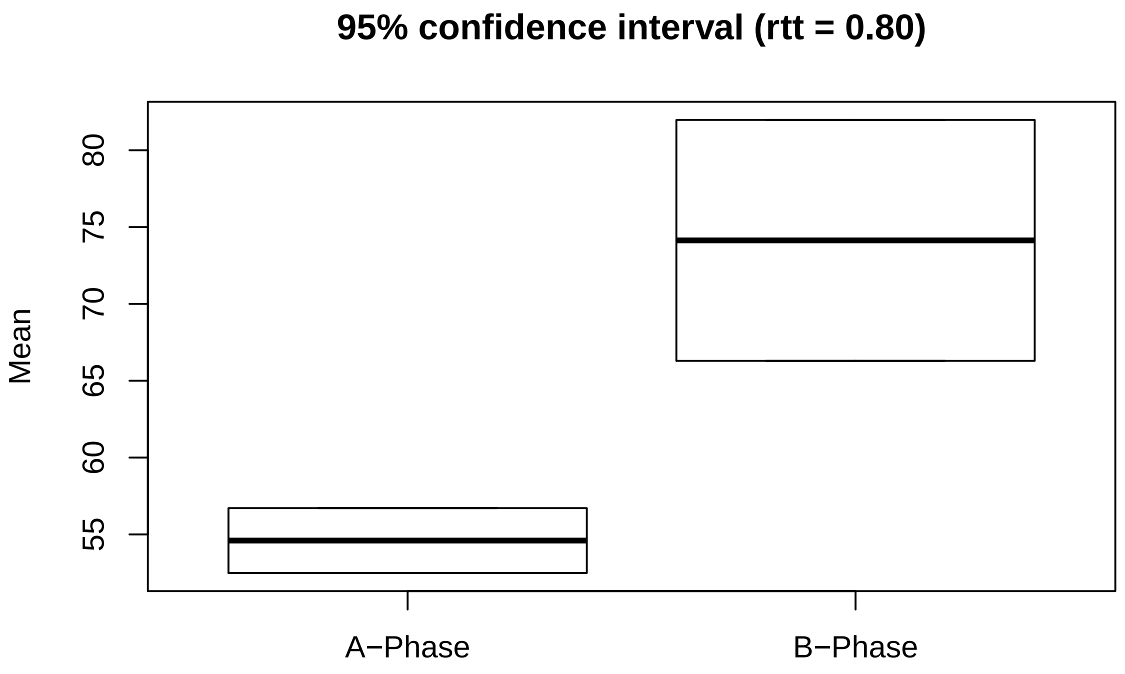
```
## Tau-U
## Method:  complete
##
## Overall Tau-U:
## A vs. B + Trend B - Trend A           A vs. B - Trend A
##                   0.7243947                   0.8745661
##
## $Johanna
##                               S       D         b     Z     p
## A vs. B - Trend A            75 126.748 0.882 0.592 3.224 0.001
## A vs. B + Trend B           145 184.445 0.806 0.786 4.749 0.000
## A vs. B + Trend B - Trend A 145 189.499 0.763 0.765 4.707 0.000
##
## $Karolina
##                               S       D         b     Z     p
## A vs. B - Trend A            70 126.412 0.824 0.554 3.010 0.003
## A vs. B + Trend B           148 183.957 0.822 0.805 4.849 0.000
## A vs. B + Trend B - Trend A 148 188.997 0.779 0.783 4.807 0.000
```

```
##
## $Anja
##                              S       D         b     Z     p
## A vs. B - Trend A           78 125.060 0.918 0.624 3.361 0.001
## A vs. B + Trend B          114 181.989 0.633 0.626 3.744 0.000
## A vs. B + Trend B - Trend A 120 186.976 0.632 0.642 3.907 0.000
```

6.7 Reliable change index

The ‘rciSC function computes three indices of reliable change (Wise, 2004) and corresponding descriptive statistics.

```
rciSC(exampleAB$Johanna, graph = TRUE)
```



```
## Reliable Change Index
##
## Mean Difference =  19.53333
## Standardized Difference =  1.678301
##
## Descriptives:
##          n     mean       SD       SE
## A-Phase  5 54.60000 2.408319 1.077033
## B-Phase 15 74.13333 8.943207 3.999524
##
## Reliability =  0.8
##
## 95 % Confidence Intervals:
```

```
##             Lower    Upper
## A-Phase 52.48905 56.71095
## B-Phase 66.29441 81.97226
##
## Reliable Change Indices:
##                                RCI
## Jacobson et al.         18.13624
## Christensen and Mendoza 12.82426
## Hageman and Arrindell   18.49426
```

Chapter 7

Piecewise linear regressions

In a piecewise regression analysis (sometimes called segmented regression) a data-set is split at a specific break point and regression parameters (intercept and slopes) are calculated separately for data before and after the break point. This is done because we assume that at the break point a qualitative change happens affecting intercept and slope. This approach lends itself perfectly to analyze single-case data which are from a statistical point of view time-series data segmented into phases. A general model for single-case data based on the piecewise regression approach has been suggested by Huitema and McKean Huitema & Mckean (2000). They refer to two phase single-case designs with a pre-intervention phase containing some measurements before the start of the intervention (A-phase) and an intervention phase containing measurements beginning at the intervention's start and lasting throughout the intervention (B-phase). In this model, four factors predict the outcome at a specific measurement point: The performance at the beginning of the study, a developmental effect leading to a continuous increase throughout all measurements (trend effect), an intervention effect leading to an immediate and constant increase in performance (level effect), and a second intervention effect that evolves continuously with the beginning of the intervention (slope effect). *scan* provides an implementation based of this piecewise regression approach. Though the original model is extended by several factors:

- multiple phase designs
- additional (control) variables
- autoregression modelling
- logistic, binomial, and poisson distributed dependent variables and error terms
- multivariate analyzes for analyzing the effect of an intervention on more than one outcome variable.

7.1 The basic plm function

The basic function for applying a regression analyzes to a single-case dataset is `plm`. This function analyzes one single-case. In its simplest way, `plm` takes one argument with an *scdf* object and it returns a full piecewise regression analyzes.

```
plm(exampleAB$Johanna)
```

```
## Piecewise Regression Analysis
##
## Dummy model:  B&L-B
##
## Fitted a gaussian distribution.
## F(3, 16) = 28.69; p = 0.000; R² = 0.843; Adjusted R² = 0.814
##
##                     B   2.5%  97.5%    SE      t     p    ΔR²
## Intercept      54.300 43.978 64.622 5.267 10.310 0.000
## Trend mt        0.100 -3.012  3.212 1.588  0.063 0.951 0.0000
## Level phase B  6.333 -2.979 15.646 4.751  1.333 0.201 0.0174
## Slope phase B  1.525 -1.642  4.692 1.616  0.944 0.359 0.0087
##
## Autocorrelations of the residuals
##  lag    cr
##    1 -0.32
##    2 -0.13
##    3 -0.01
##
## Formula: values ~ 1 + mt + phaseB + interB
```

7.1.1 Dummy model

The `model` argument is used to code the *dummy variable*. This *dummy variable* is used to compute the slope and level effects of the *phase* variable.
The *phase* variable is categorical, identifying the phase of each measurement. Typically, categorical variables are implemented by means of dummy variables. In a piecewise regression model two phase effects have to be estimated: a level effect and a slope effect. The level effect is implemented quite straight forward: for each phase beginning with the second phase a new dummy variable is created with values of zero for all measurements except the measurements of the phase in focus where values of one are set.

phase	values	level_B
A	3	0
A	6	0
A	4	0
A	7	0
B	5	1
B	3	1
B	4	1
B	6	1
B	3	1

For estimating the *slope effect* of each phase, another kind of dummy variables have to be created. Like the dummy variables for level effects the values are set to zero for all measurements except the ones of the phase in focus. Here, values start to increase with every measurement until the end of the phase.
Various suggestions have been made regarding the way in which these values increase. The *B&L-B* model starts with a one at the first measurement of the phase and increases with every

measurement while the *H-M* model starts with a zero.

phase	values	level	slope B&L-M	slope H-M
A	3	0	0	0
A	6	0	0	0
A	4	0	0	0
A	7	0	0	0
B	5	1	1	0
B	3	1	2	1
B	4	1	3	2
B	6	1	4	3
B	3	1	5	4

With single-case studies with more than two phases it gets a bit more complicated. Applying the a fore described models to three phases would result in a comparison of each phase to the first phase (usually the A Phase). That is, regression weights and significance tests will depict differences of each phase to the values of phase A. This might be OK depending on what you are interested in. But in a lot of cases we are more interested in analyzing the effects of a phase compared to the previous one.
This is achieved applying the *JW* dummy model. In this model, the dummy variable for the level effect is set to zero for all phases preceding the phase in focus and set to one for all remaining measurements. Similar, the dummy variable for the slope effect is set to zero for all phases preceding the one in focus and starts with one for the first measurement of the target phase and increases until the last measurement of the case.

phase	values	level_B	level_C	slope_B	slope_C
A	3	0	0	0	0
A	6	0	0	0	0
A	4	0	0	0	0
A	7	0	0	0	0
B	5	1	0	1	0
B	3	1	0	2	0
B	4	1	0	3	0
B	6	1	0	4	0
B	3	1	0	5	0
B	7	1	1	6	1
B	5	1	1	7	2
B	6	1	1	8	3
B	4	1	1	9	4
B	8	1	1	10	5

7.2 Different modells

```r
plm(exampleAB$Karolina)
```

```
## Piecewise Regression Analysis
##
## Dummy model:  B&L-B
```

```
##
## Fitted a gaussian distribution.
## F(3, 16) = 33.73; p = 0.000; R² = 0.863; Adjusted R² = 0.838
##
##                     B   2.5%  97.5%    SE     t     p    ΔR²
## Intercept      47.600 36.724 58.476 5.549 8.578 0.000
## Trend mt        1.400 -1.879  4.679 1.673 0.837 0.415 0.0060
## Level phase B   3.352 -6.459 13.164 5.006 0.670 0.513 0.0038
## Slope phase B   0.539 -2.798  3.877 1.703 0.317 0.756 0.0009
##
## Autocorrelations of the residuals
##  lag    cr
##    1  0.03
##    2 -0.40
##    3 -0.22
##
## Formula: values ~ 1 + mt + phaseB + interB
```

The piecewise regression reveals a significant level effect and two non significant effects for trend and slope. In a further analyses we would like to put the slope effect out of the equation. There are several ways to do this. The easiest way is the to unset the `slope` argument: `slope = FALSE`.

```
plm(exampleAB$Karolina, slope = FALSE)
```

```
## Piecewise Regression Analysis
##
## Dummy model:  B&L-B
##
## Fitted a gaussian distribution.
## F(2, 17) = 53.38; p = 0.000; R² = 0.863; Adjusted R² = 0.846
##
##                     B  2.5%  97.5%    SE      t     p    ΔR²
## Intercept      46.038 41.187 50.889 2.475 18.602 0.000
## Trend mt        1.921  1.328  2.513 0.302  6.352 0.000 0.3261
## Level phase B   2.460 -5.432 10.351 4.026  0.611 0.549 0.0030
##
## Autocorrelations of the residuals
##  lag    cr
##    1  0.06
##    2 -0.39
##    3 -0.24
##
## Formula: values ~ 1 + mt + phaseB
```

In the resulting estimations the trend and level effects are now significant. The model estimated a trend effect of 0.96 points per measurement time and a level effect of 14.3 points. That is, with the beginning of the intervention (the B-phase) the score increases by 14.3 points.

7.3 Modelling autoregression

```
autocorrSC(Grosche2011)
```

```
## Autocorrelations
##
##   case phase lag_1 lag_2 lag_3
##    Eva     A -0.04 -0.56 -0.01
##    Eva     B  0.46  0.10  0.16
##    Eva   all  0.48  0.13  0.24
##  Georg     A  0.51 -0.01 -0.13
##  Georg     B -0.01 -0.02 -0.14
##  Georg   all  0.40  0.15 -0.12
##   Olaf     A  0.64  0.29 -0.24
##   Olaf     B -0.45 -0.20  0.16
##   Olaf   all  0.35  0.12 -0.09
```

7.4 Adding additional predictors

In more complex analyses additional predictors can be included in the piecewise regression model.

To do this, we have to change the regression formula ‘manually’ by applying the `update` argument. The `update` argument allows to change the underlying regression formula. To add a new variable named for example `newVar`, set `update = .~. + newVar`. The `.~.` part takes the internally build formula and `+ newVar` adds a variable named `newVar` to the equation.

```
exampleAB_add
```

```
## #A single-case data frame with one case
##
##  Rolf: day wellbeing cigarrets depression phase
##          1        46         2          7  Base
##          2        49         5          6  Base
##          3        49         4          1  Base
##          4        49         1          4  Base
##          5        50         2          7  Base
##          6        47         4          2  Base
##          7        45         4          6  Base
##          8        59         0          0  Base
##          9        58         2          3  Base
##         10        59         3          6  Base
##         11        59         2          8  Base
##         12        43         1          7  Base
##         13        46         4          9  Base
##         14        52         5          6  Base
##         15        55         5          3  Base
## # ... up to 25 more rows
```

```r
plm(exampleAB_add, update = .~. + cigarrets)
```

```
## Piecewise Regression Analysis
##
## Dummy model:  B&L-B
##
## Fitted a gaussian distribution.
## F(4, 35) = 5.87; p = 0.001; R² = 0.402; Adjusted R² = 0.333
##
##                              B   2.5%  97.5%    SE      t     p    ΔR²
## Intercept               48.579 42.539 54.618 3.081 15.765 0.000
## Trend day                0.392 -0.221  1.005 0.313  1.253 0.218 0.0269
## Level phase Medication   3.753 -2.815 10.321 3.351  1.120 0.270 0.0214
## Slope phase Medication  -0.294 -0.972  0.384 0.346 -0.850 0.401 0.0124
## cigarrets               -0.221 -1.197  0.755 0.498 -0.443 0.660 0.0034
##
## Autocorrelations of the residuals
##  lag    cr
##    1  0.20
##    2 -0.19
##    3 -0.16
##
## Formula: wellbeing ~ day + phaseMedication + interMedication + cigarrets
```

The formula has two parts divided by a tilde. Left of the tilde is the variable to be predicted and right of it the predictors. A `1` indicates the intercept, the variable `mt` estimates the trend effect, `phaseB` the level effect of the B-phase and the variable `interB` the slope effect of the B-phase. If `formula` is not explicitly defined it is set to `formula = values ~ 1 + mt + phaseB + interB` (assuming an AB-design) to estimate the full piecewise regression model.

Chapter 8

Multilevel regression analyses

Multilevel analyses can take the piecewise-regression approach even further. It allows for

- analyzing the effects between phases for multiple single-cases at once
- describing variability between subjects regarding these effects, and
- introducing variables and factors for explaining the differences.

The basic function for applying a multilevel piecewise regression analysis is `hplm`. The `hplm` function is similar to the `plm` function, so I recommend that you get familar with `plm` before applying an `hplm`.

Here is a simple example:

```r
hplm(exampleAB_50)
```

```
## Hierarchical Piecewise Linear Regression
##
## Estimation method ML
## Slope estimation method: B&L-B
## 50 Cases
##
## ICC = 0.195; L = 192.0; p = 0.000
##
## Fixed effects (values ~ 1 + mt + phaseB + interB)
##
##                     B    SE   df      t p
## Intercept      51.614 1.282 1281 40.274 0
## Trend mt        0.671 0.115 1281  5.844 0
## Level phase B  12.938 0.590 1281 21.942 0
## Slope phase B   0.859 0.119 1281  7.236 0
##
## Random effects (~1 | case)
##
##           EstimateSD
## Intercept      8.179
## Residual       5.335
```

Here is an example inlcuding random slopes:

```
hplm(exampleAB_50, random.slopes = TRUE)
```

```
## Hierarchical Piecewise Linear Regression
##
## Estimation method ML
## Slope estimation method: B&L-B
## 50 Cases
##
## ICC = 0.195; L = 192.0; p = 0.000
##
## Fixed effects (values ~ 1 + mt + phaseB + interB)
##
##                     B    SE   df      t p
## Intercept      51.720 1.348 1281 38.378 0
## Trend mt        0.637 0.125 1281  5.080 0
## Level phase B  13.090 0.762 1281 17.176 0
## Slope phase B   0.887 0.129 1281  6.856 0
##
## Random effects (~1 + mt + phaseB + interB | case)
##
##                EstimateSD
## Intercept           8.691
## Trend mt            0.343
## Level phase B       3.654
## Slope phase B       0.358
## Residual            4.994
```

8.1 Adding additional L2-variables

In some analyses researchers want to investigate whether attributes of the individuals contribute to the effectiveness of an intervention. For example might an intervention on mathematical abilities be less effective for student with a migration background due to too much language related material within the training. Such analyses can also be conducted with *scan*. Therefore, we need to define a new *data frame* including the relevant information of the subjects of the single-case studies we want to analyze. This *data frame* consists of a variable labeled `case` which has to correspond to the case names of the *scfd* and further variables with attributes of the subjects. To build a *data frame* we can use the R function `data.frame`.

```
L2 <- data.frame(
  case = c("Antonia","Theresa", "Charlotte", "Luis", "Bennett", "Marie"),
  age = c(16, 13, 13, 10, 5, 14),
  sex = c("f","f","f","m","m","f")
)
L2
```

```
##        case age sex
## 1   Antonia  16   f
## 2   Theresa  13   f
## 3 Charlotte  13   f
## 4      Luis  10   m
## 5   Bennett   5   m
## 6     Marie  14   f
```

Multilevel analyses require a high number of Level 2 units. The exact number depends on the complexity of the analyses, the size of the effects, the number of level 1 units, and the variability of the residuals. But surely we need at least about 30 level 2 units. In a single-case design that is, we need at least 30 single-cases (subjects) within the study. After setting the level 2 data frame we use the `data.l2` argument of the `hplm` function to include it into the analysis. Then we have to specify the regression function using the `update.fixed` argument. The level 2 variables can be added just like any other additional variable. For example, we have added a level 2 data-set with the two variables `sex` and `age`. `update` could be construed of the level 1 piecewise regression model `.~.` plus the additional level 2 variables of interest `+ sex + age`. The complete argument is `update.fixed = .~. + sex + age`. This analyses will estimate a main effect of sex and age on the overall performance. In case we want to analyze an interaction between the intervention effects and for example the sex of the subject we have to add an additional interaction term (a cross-level interaction). An interaction is defined with a colon. So `sex:phase` indicates an interaction of sex and the level effect in the single case study. The complete formula now is `update.fixed = .~. + sex + age + sex:phase`.

scan includes an example single-case study with 50 subjects `example50` and an additional level 2 data-set `example50.l2`. Here are the first 10 cases of `example50.l2`.

case	sex	age
Jeremiah	m	10
Leon	m	12
Pablo	m	10
Jamal	m	10
Giovanny	m	11
Callum	m	10
Dwayne	m	11
Neymar	m	10
Krish	m	11
Caiden	m	12

Analyzing the data with `hplm` could look like this:

```
hplm(exampleAB_50, data.l2 = exampleAB_50.l2, update.fixed = .~. + sex + age)
```

```
## Hierarchical Piecewise Linear Regression
##
## Estimation method ML
## Slope estimation method: B&L-B
## 50 Cases
##
## ICC = 0.195; L = 192.0; p = 0.000
##
## Fixed effects (values ~ mt + phaseB + interB + sex + age)
##
```

```
##                     B     SE   df      t     p
## Intercept      71.464 13.076 1281  5.465 0.000
## Trend mt        0.667  0.115 1281  5.808 0.000
## Level phase B  12.951  0.590 1281 21.951 0.000
## Slope phase B   0.863  0.119 1281  7.264 0.000
## sexm           -2.130  2.336   47 -0.912 0.367
## age            -1.710  1.157   47 -1.478 0.146
##
## Random effects (~1 | case)
##
##           EstimateSD
## Intercept      7.975
## Residual       5.335
```

sex is a factor with the levels f and m. So sexm is the effect of being male on the overall performance. age does not seem to have any effect. So we drop age out of the equation and add an interaction of sex and phase to see whether the sex effect is due to a weaker impact of the intervention on males.

```
hplm(exampleAB_50, data.l2 = exampleAB_50.l2, update.fixed = .~. + sex + sex:phaseB)
```

```
## Hierarchical Piecewise Linear Regression
##
## Estimation method ML
## Slope estimation method: B&L-B
## 50 Cases
##
## ICC = 0.195; L = 192.0; p = 0.000
##
## Fixed effects (values ~ mt + phaseB + interB + sex + phaseB:sex)
##
##                          B    SE   df       t     p
## Intercept           49.801 1.735 1280  28.711 0.000
## Trend mt             0.638 0.109 1280   5.847 0.000
## Level phase B       16.762 0.644 1280  26.034 0.000
## Slope phase B        0.889 0.113 1280   7.884 0.000
## sexm                 3.983 2.368   48   1.682 0.099
## Level phase B:sexm  -7.463 0.622 1280 -11.994 0.000
##
## Random effects (~1 | case)
##
##           EstimateSD
## Intercept      8.144
## Residual       5.059
```

Now the interaction phase:sexm is significant and the main effect is no longer relevant. It looks like the intervention effect is 6.9 points lower for male subjects. While the level-effect is 17.3 points for female subjects it is 17.3 - 6.9 = 10.4 for males.

Chapter 9

Randomization tests

The randSC function computes a randomization test for single or multiple baseline single-case data. The function is based on an algorithm from the SCRT package (Bulte & Onghena, 2009, 2012), but rewritten and extended for the use in AB designs.

The `statsitics` argument defines the statistic on which the comparison of the phases is based on. The following comparisons are possible:

- "Mean A-B": Uses the difference between the mean of phase A and the mean of phase B. * This is appropriate if a decrease of scores is expected for phase B.
- "Mean B-A": Uses the difference between the mean of phase B and the mean of phase A. This is appropriate if an increase of scores is expected for phase B.
- "Mean |A-B|": Uses the absolute value of the difference between the means of phases A and B.
- "Median A-B": The same as "Mean A-B", but based on the median.
- "Median B-A": The same as "Mean B-A", but based on the median.

number
Sample size of the randomization distribution. The exactness of the p-value can not exceed 1/number (i.e., number = 100 results in p-values with an exactness of one percent). Default is number = 500. For faster processing use number = 100. For more precise p-values set number = 1000.

complete
If TRUE, the distribution is based on a complete permutation of all possible starting combinations. This setting overwrites the number Argument. The default setting is FALSE. limit
Minimal number of data points per phase in the sample. The first number refers to the A-phase and the second to the B-phase (e.g., limit = c(5,3)). If only one number is given, this number is applied to both phases. Default is limit = 5.

startpoints
Alternative to the limit-parameter startpoints exactly defines the possible start points of phase B (e.g., startpoints = 4:9 restricts the phase B start points to measurements 4 to 9. startpoints overruns the limit-parameter.

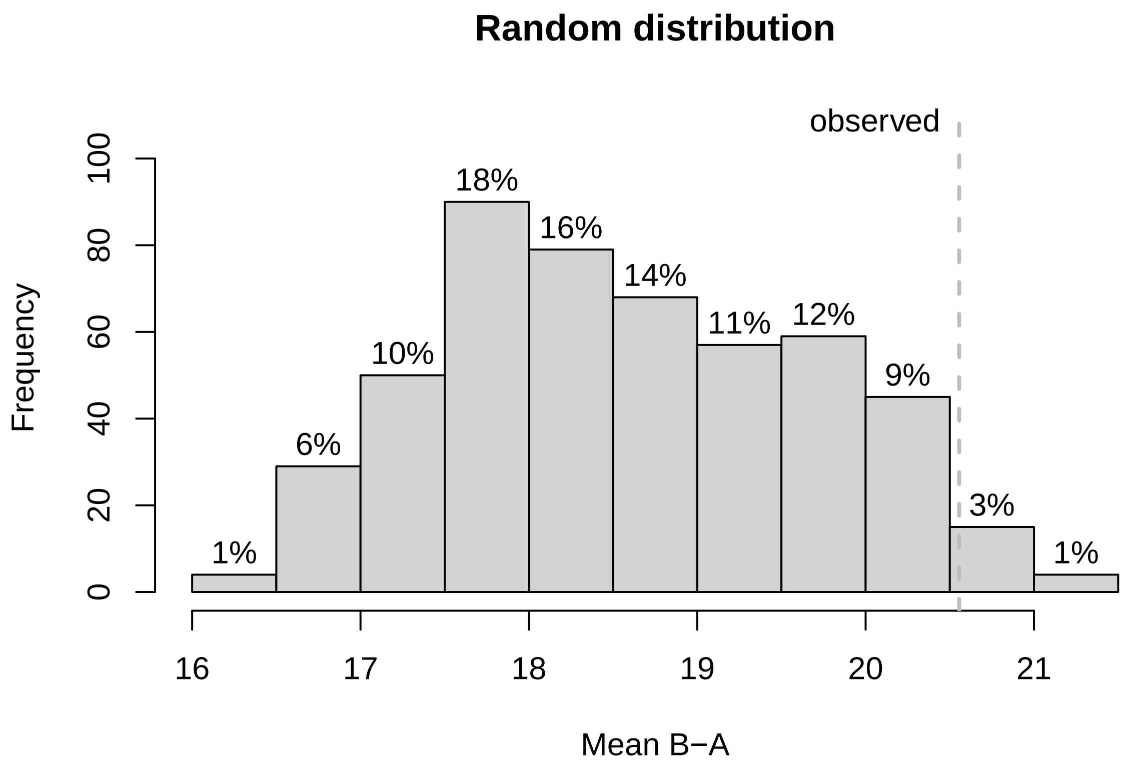
exclude.equal
If set to exclude.equal = FALSE, which is the default, random distribution values equal to

the observed distribution are counted as null-hypothesis conform. That is, they decrease the probability of rejecting the null-hypothesis (increase the p-value). exclude.equal should be set to TRUE if you analyse one single-case design (not a multiple baseline data set) to reach a sufficient power. But be aware, that it increases the chance of an alpha-error.

graph
If graph = TRUE, a histogram of the resulting distribution is plotted. It's FALSE by default.

phases
A vector of two characters or numbers indicating the two phases that should be compared. E.g., phases = c("A","C") or phases = c(2,4) for comparing the second and the fourth phase. Phases could be combined by providing a list with two elements. E.g., phases = list(A = c(1,3), B = c(2,4)) will compare phases 1 and 3 (as A) against 2 and 4 (as B). Default is phases = c("A","B").

```
randSC(exampleAB, graph = TRUE)
```



```
## Randomization Test
##
## Test for 3 cases.
##
## Comparing phase 1 against phase 2
## Statistic:  Mean B-A
##
## Minimal length of each phase:  5 5
## Observed statistic =  20.55556
##
##
```

```
## Distribution based on a random sample of all 1331 possible combinations.
## n   =  500
## M   =  18.57825
## SD  =  1.11305
## Min =  16.05185
## Max =  21.34493
##
## p   =  0.034
##
## Shapiro-Wilk Normality Test: W = 0.980; p = 0.000  (Hypothesis of Normality rejected)
## z = 1.7765, p = 0.0378 (single sided)
```

Chapter 10

Exporting *scan* results

Caution!! The exporting function is still in an experimental state.

The `export` function will make it easier to convert the results of your `scan` analyses to tables and descriptions for your documents and presentations.
Basically, `export` takes a `scan` object and converts it to a html table or latex output.

`export` it build on top of the `knitr` and `kableextra` packages. The list provided in the `kable_options` argument is implemented in the `kable` function of `knitr` and the list provided to the `kable_styling_options` is implemented in the `kable_styling` command of the `kableExtra` package. `export` sets some defaults for these functions but you can play around and overwrite them.

`export` works best when used within an rmarkdown file and/or within `RStudio`.
In `RStudio`

[xxx to be continued!]

10.1 Single case data files

```
export(exampleA1B1A2B2_zvt, kable_options = list(booktab = TRUE))
```

Tick				Trick				Track			
zvt	d2	day	part	zvt	d2	day	part	zvt	d2	day	part
47	131	1	A1	51	100	1	A1	54	89	1	A1
58	134	2	A1	58	126	2	A1	57	116	2	A1
76	141	3	A1	70	130	3	A1	51	114	3	A1
63	141	4	B1	65	130	4	B1	61	131	4	B1
71	140	5	B1	67	137	5	B1	57	132	5	B1
59	140	6	B1	63	133	6	B1	53	130	6	B1
64	138	7	A2	64	136	7	A2	58	128	7	A2
69	140	8	A2	70	137	8	A2	57	131	8	A2
72	141	9	A2	70	135	9	A2	60	130	9	A2
77	140	10	B2	68	128	10	B2	55	129	10	B2
76	138	11	B2	69	137	11	B2	58	118	11	B2
73	140	12	B2	70	138	12	B2	58	131	12	B2

10.2 Descriptive stats

```
res <- describeSC(GruenkeWilbert2014)
export(res, digits = 1, kable_options = list(booktab = TRUE))
```

Descriptive statistics.

	n		Missing		M		Median		SD		MAD		Min		Max		Trend	
	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B
Anton	4	14	0	0	5.00	9.14	5	9	0.82	0.77	0.74	1.48	4	8	6	10	-0.40	0.03
Bob	7	11	0	0	3.00	8.82	3	9	0.82	0.87	1.48	0.00	2	7	4	10	0.04	0.04
Paul	6	12	0	0	3.83	8.83	4	9	0.75	0.72	0.74	0.74	3	8	5	10	-0.26	0.02
Robert	8	10	0	0	4.12	8.90	4	9	0.83	0.99	1.48	1.48	3	7	5	10	-0.06	-0.14
Sam	5	13	0	0	4.60	9.08	5	9	0.55	0.86	0.00	1.48	4	8	5	10	0.10	0.03
Tim	4	14	0	0	3.00	9.00	3	9	0.82	0.96	0.74	1.48	2	7	4	10	-0.60	0.00

n = Number of measurements; Missing = Number of missing values; M = Mean; Median = Median; SD = Standard deviation; MAD = Median average deviation; Min = Minimum; Max = Maximum; Trend = Slope of dependent variable regressed on measurement-time.

10.3 Overlap indices

```
res <- overlapSC(exampleA1B1A2B2_zvt, phases = list(c(1,3),c(2,4)))
export(res, digits = 1, flip = TRUE, kable_options = list(booktab = TRUE))
```

Overlap indices.Comparing phases A1 + A2 against phases B1 + B2.

	Tick	Trick	Track
PND	16.67	0.00	16.67
PEM	66.67	50.00	50.00
PET	66.67	33.33	33.33
NAP	68.06	51.39	58.33
NAP-R	36.11	2.78	16.67
PAND	66.67	50.00	54.17
Tau-U	0.14	0.03	-0.03
Base_Tau	0.27	-0.25	0.13
Delta M	5.50	3.17	0.83
Delta Trend	-0.31	-1.10	-0.74
SMD	0.52	0.40	0.26

PND = Percentage Non-Overlapping Data; PEM = Percentage Exceeding the Median; PET = Percentage Exceeding the Trend; NAP = Nonoverlap of all pairs; NAP-R = NAP rescaled; PAND = Percentage all nonoverlapping data;Tau U = Parker's Tau-U; Base Tau = Baseline corrected Tau; Delta M = Mean difference between phases; Delta Trend = Trend difference between phases; SMD = Standardized Mean Difference.

10.4 Piecewise linear models

```r
res <- plm(exampleA1B1A2B2$Pawel)
export(res, kable_options = list(booktab = TRUE))
```

Piecewise-regression model for variable ".

		CI(95%)					
Parameter	B	2.5%	97.5%	SE	t	p	Delta R^2
Intercept	12.47	4.90	20.03	3.86	3.23	<.01	
Trend mt	0.22	-0.99	1.44	0.62	0.36	.72	.00
Level phase B1	17.69	7.71	27.67	5.09	3.48	<.01	.14
Level phase A2	2.58	-16.96	22.12	9.97	0.26	.79	.00
Level phase B2	12.54	-18.46	43.54	15.82	0.79	.43	.01
Slope phase B1	-1.41	-3.13	0.32	0.88	-1.60	.11	.03
Slope phase A2	-1.10	-2.83	0.62	0.88	-1.25	.21	.02
Slope phase B2	-1.08	-2.81	0.64	0.88	-1.23	.22	.02

Note:
F(7, 32) = 7.86; p <.001; $R^2 = 0.632$; Adjusted $R^2 = 0.552$

10.5 Hierarchical piecewise regressions

```
res <- hplm(exampleAB_50, data.l2 = exampleAB_50.l2, lr.test = TRUE, random.slopes = TRUE)
export(res, digits = 2, kable_options = list(booktab = TRUE))
```

Hierarchical Piecewise Linear Regression for variable ".

Parameter	B	SE	df	t	p
Intercept	51.72	1.35	1281	38.38	<.001
Trend mt	0.64	0.13	1281	5.08	<.001
Level phase B	13.09	0.76	1281	17.18	<.001
Slope phase B	0.89	0.13	1281	6.86	<.001
Random effects					
	SD	**L**	**df**	**p**	
Intercept	8.69	194.8	4	<.001	
Trend mt	0.34	4.8	4	.30	
Level phase B	3.65	28.25	4	<.001	
Slope phase B	0.36	4.81	4	.30	
Residual	4.99	NA	NA	NA	
AIC	8403.5				
BIC	8481.5				
ICC	0.2	L = 192	p <.001		

Note:
Estimation method ML; Slope estimation method: B&L-B; 50 cases

References

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., … Iannone, R. (2019). *Rmarkdown: Dynamic documents for r*. Retrieved from https://CRAN.R-project.org/package=rmarkdown

Huitema, B. E., & Mckean, J. W. (2000). Design specification issues in time-series intervention models. *Educational and Psychological Measurement*, *60*(1), 38–58. Retrieved from http://epm.sagepub.com/content/60/1/38.short

Iglewicz, B., & Hoaglin, D. C. (1993). *How to detect and handle outliers*. Milwaukee, Wis. : ASQC Quality Press.

Parker, R. I., Hagan-Burke, S., & Vannest, K. (2007). Percentage of All Non-Overlapping Data (PAND) An Alternative to PND. *The Journal of Special Education*, *40*(4), 194–204. Retrieved from http://sed.sagepub.com/content/40/4/194.short

Parker, R. I., & Vannest, K. (2009). An improved effect size for single-case research: Nonoverlap of all pairs. *Behavior Therapy*, *40*(4), 357–367. Retrieved from http://www.sciencedirect.com/science/article/pii/S0005789408000816

Parker, R. I., Vannest, K. J., & Davis, J. L. (2011). Effect Size in Single-Case Research: A Review of Nine Nonoverlap Techniques. *Behavior Modification*, *35*(4), 303–322. https://doi.org/10.1177/0145445511399147

Parker, R. I., Vannest, K. J., Davis, J. L., & Sauber, S. B. (2011). Combining Nonoverlap and Trend for Single-Case Research: Tau-U. *Behavior Therapy*, *42*(2), 284–299. https://doi.org/10.1016/j.beth.2010.08.006

R Core Team. (2019). *R: A language and environment for statistical computing*. Retrieved from https://www.R-project.org/

Scruggs, T. E., Mastropieri, M. A., & Casto, G. (1987). The Quantitative Synthesis of Single-Subject Research Methodology and Validation. *Remedial and Special Education*, *8*(2), 24–33. https://doi.org/10.1177/074193258700800206

Wilbert, J., & Lueke, T. (2019). *Scan: Single-case data analyses for single and multiple baseline designs*.

Wise, E. A. (2004). Methods for analyzing psychotherapy outcomes: A review of clinical significance, reliable change, and recommendations for future directions. *Journal of Personality Assessment*, *82*(1), 50–59. Retrieved from http://www.tandfonline.com/doi/abs/10.1207/s15327752jpa8201_10

Xie, Y. (2019). *Bookdown: Authoring books and technical documents with r markdown.* Retrieved from https://CRAN.R-project.org/package=bookdown