

# Analyzing single-case data with R and scan

Jürgen Wilbert

2021-05-28



# Contents

<b>Welcome</b>	<b>5</b>
<b>Preface</b>	<b>7</b>
Software reference . . . . .	7
<b>1 Introduction</b>	<b>9</b>
1.1 A teaser . . . . .	9
<b>2 Some things about R</b>	<b>17</b>
2.1 Basic R . . . . .	18
<b>3 The scan package</b>	<b>21</b>
3.1 Installing the <i>scan</i> package . . . . .	21
3.2 Development version of scan . . . . .	21
3.3 Reporting issues with <i>scan</i> and suggesting enhancements . . . . .	22
3.4 Functions overview . . . . .	22
3.5 Changes with version 0.50 . . . . .	22
<b>4 Managing single-case data</b>	<b>25</b>
4.1 A <i>single-case data frame</i> . . . . .	25
4.2 Creating scdfs . . . . .	25
4.3 Saving and reading <i>single-case data frames</i> . . . . .	29
4.4 Import and export <i>single-case data frames</i> . . . . .	29
4.5 Displaying and manipulating scdf-files . . . . .	31
<b>5 Creating a single-case data plot</b>	<b>35</b>
5.1 Graphical styles of a plot . . . . .	36
<b>6 Describe and manipulate single-case data frames</b>	<b>41</b>
6.1 Describing and summarizing . . . . .	41
6.2 Autoregression and trendanalyses . . . . .	43
6.3 Missing values . . . . .	44
6.4 Outlieranalysis . . . . .	46
6.5 Smoothing data . . . . .	48
<b>7 Overlapping indices</b>	<b>49</b>
7.1 Percentage non-overlapping data (PND) . . . . .	50
7.2 Percentage exceeding the median (PEM) . . . . .	51

7.3	Percentage exceeding the regression trend (PET) . . . . .	51
7.4	Percentage of all non-overlapping data (PAND) . . . . .	52
7.5	Nonoverlap of all pairs (NAP) . . . . .	54
7.6	Tau-U . . . . .	54
7.7	Reliable change index . . . . .	56
<b>8</b>	<b>Piecewise linear regressions</b>	<b>59</b>
8.1	The basic plm function . . . . .	60
8.2	[to be written] Multivariate piecewise regression . . . . .	65
8.3	Multilevel plm analyses . . . . .	65
<b>9</b>	<b>Randomization tests</b>	<b>71</b>
<b>10</b>	<b>Default settings</b>	<b>75</b>
<b>11</b>	<b>Exporting <i>scan</i> results</b>	<b>77</b>
11.1	Single case data files . . . . .	77
11.2	Descriptive stats . . . . .	77
11.3	Overlap indices . . . . .	77
11.4	Piecewise linear models . . . . .	80
11.5	Hierarchical piecewise regressions . . . . .	80
	<b>About the author</b>	<b>83</b>
	<b>References</b>	<b>85</b>

# Welcome



---

<sup>1</sup>The cover has been designed by Tony Wilbert and Henry Ritter. Thanx for that!



# Preface

Hello!

I am glad you found your way to this book as it tells me you are beginning to use the `scan` package. While `scan` is quite 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).

If you have any suggestions how to enhance the book or would like to report errors, comments, feedback etc. you can do so by posting an issue to the gitHub repository of this book. You can find the repository at <https://github.com/jazznbass/scan-Book>.

Thank you!

Jürgen

28 May 2021

## Software reference

This book has been created using the **Rmarkdown** (Allaire et al., 2021) and **bookdown** (Xie, 2021) packages within the RStudio (RStudio Team, 2018) environment. The analyses have been conducted with the **R** package `scan` at version 0.53 (Wilbert and Lueke, 2021). R version 4.1.0 (2021-05-18) was used (R Core Team, 2021).





# 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. Despite its usefulness, standards on how to conduct single-case studies, how to analyze the data, and how to present the results is less well developed compared to group based research designs. Moreover, while there is ample software helping to analyse data, most of the software is designed towards analyzing group based data sets. Visualizing single-case data sets oftentimes means to tinker with spreadsheet programs and analyzing becomes a cumbersome endeavor. This book addresses this gap. It has been written around a specialized software tool for managing, visualizing, and analyzing single-case data. This tool is an extension package for the software R (R Core Team, 2021) named **scan**, an acronym for **single-case analyses**.

### 1.1 A teaser

Before I go into the details on how **scan** exactly works, I like to provide an example of what you can do with **scan**. It is meant to be a teaser to get you motivated to tackle the steep learning curve associated with the use of R (but there is a land of milk and honey behind this curve!). So, do not mind if you do not understand every detail of this example, it will all be explained and obvious to you once you get familiar with **scan**.

Let us set a fictional context. Let us assume you are researching on a method to foster the calculation abilities of struggling fourth grade students. You developed an intervention program named *KUNO*. In a pilot study you like to get some evidence on the effectiveness of that new method and you set up a multi-baseline single-case study comprising three students that take part in the *KUNO* program across a period of ten weeks. Throughout that course you regularly measured the calculation abilities of each student 20 times with a reliable test. You also implemented a follow up after eight weeks with additional five measures. The calculation test gives you the number of correctly solved calculation tasks within ten minutes.

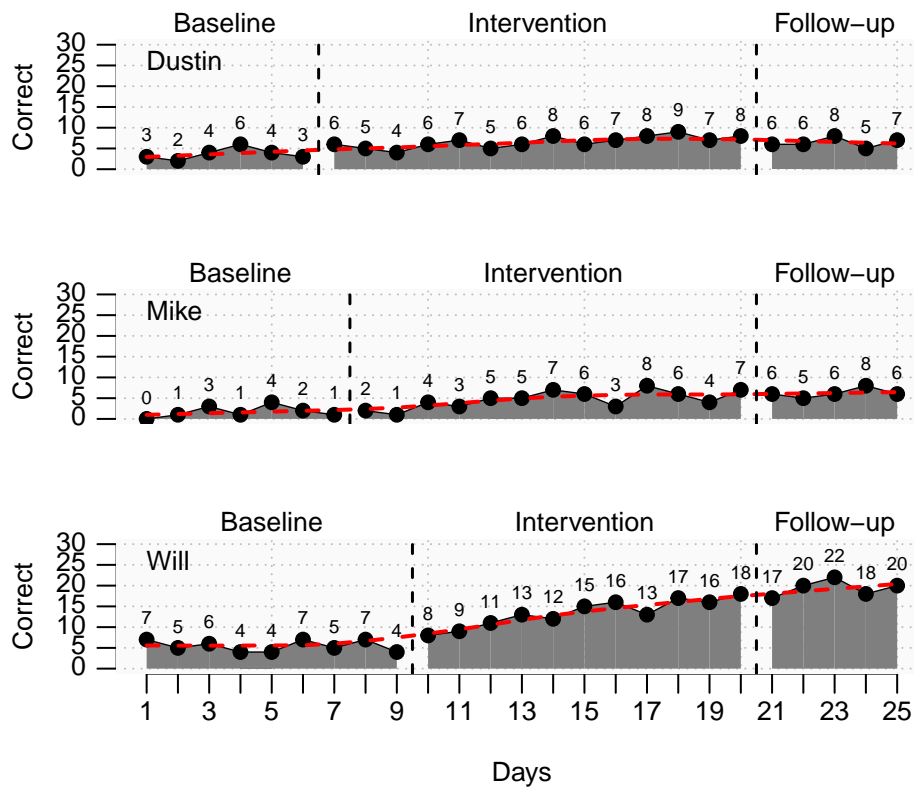
Now, I invent some data for this fictitious *KUNO* study as it would be too laborious to conduct a real study and actually to evolve a real intervention method.

We use the `scan` package to code the data. Each case consists of 25 measurements. We have three phases: pre intervention (A), during the intervention (B), and follow-up (C). Phases A and B have different lengths. The cases are named and combined into a single object called `strange_study`.

```
case1 <- scdf(
  c(A = 3, 2, 4, 6, 4, 3,
     B = 6, 5, 4, 6, 7, 5, 6, 8, 6, 7, 8, 9, 7, 8,
     C = 6, 6, 8, 5, 7),
  name = "Dustin"
)
case2 <- scdf(
  c(A = 0, 1, 3, 1, 4, 2, 1,
     B = 2, 1, 4, 3, 5, 5, 7, 6, 3, 8, 6, 4, 7,
     C = 6, 5, 6, 8, 6),
  name = "Mike"
)
case3 <- scdf(
  c(A = 7, 5, 6, 4, 4, 7, 5, 7, 4,
     B = 8, 9, 11, 13, 12, 15, 16, 13, 17, 16, 18,
     C = 17, 20, 22, 18, 20),
  name = "Will"
)
strange_study <- c(case1, case2, case3)
```

Now we visualize the cases:

```
plot(
  strange_study,
  ylab = "Correct",
  xlab = "Days",
  lines = c("loreg", col = "red"),
  phase.names = c("Baseline", "Intervention", "Follow-up"),
  style = "chart",
  ylim = c(0, 30),
  xinc = 2
)
```



Now we need some descriptive statistics:

```
describe(strange_study)
```

```
\begin{table}
```

```
\caption{(\#tab:describe_strange_study)Descriptive statistics}
```

Parameter	Dustin	Mike	Will
n A	6.00	7.00	9.00
n B	14.00	13.00	11.00
n C	5.00	5.00	5.00
Missing A	0.00	0.00	0.00
Missing B	0.00	0.00	0.00
Missing C	0.00	0.00	0.00
m A	3.67	1.71	5.44
m B	6.57	4.69	13.45
m C	6.40	6.20	19.40
md A	3.50	1.00	5.00
md B	6.50	5.00	13.00
md C	6.00	6.00	20.00
sd A	1.37	1.38	1.33
sd B	1.40	2.10	3.27
sd C	1.14	1.10	1.95
mad A	0.74	1.48	1.48
mad B	1.48	2.97	4.45
mad C	1.48	0.00	2.97
Min A	2.00	0.00	4.00
Min B	4.00	1.00	8.00
Min C	5.00	5.00	17.00
Max A	6.00	4.00	7.00
Max B	9.00	8.00	18.00
Max C	8.00	8.00	22.00
Trend A	0.23	0.21	-0.08
Trend B	0.25	0.36	0.91
Trend C	0.10	0.30	0.40

*Note:*

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.

\end{table}

Single-case data are oftentimes analyzed with overlap indices. Let us get an overview comparing phases A and B:

```
overlap(strange_study)
```

\begin{table}

\caption{(#tab:overlapAB\_strange\_study)Overlap indices. Comparing phase A against phase B}

	Dustin	Mike	Will
PND	50.00	53.85	100.00
PEM	100.00	92.31	100.00
PET	71.43	61.54	100.00
NAP	92.86	87.91	100.00
NAP-R	85.71	75.82	100.00
PAND	90.00	80.00	100.00
Tau-U	0.66	0.56	0.80
Base_Tau	0.60	0.55	0.74
Delta M	2.90	2.98	8.01
Delta Trend	0.02	0.14	0.99
SMD	2.13	2.16	6.01
Hedges g	2.04	2.07	5.75

*Note:*

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; Hedges g = Corrected SMD.

\end{table}

How do the changes hold up against the follow-up? Let us compare phases A and C:

```
overlap(strange_study, phases = c("A", "C"))
```

\begin{table}

\caption{(\#tab:overlapAC\_strange\_study)Overlap indices. Comparing phase A against phase C}

	Dustin	Mike	Will
PND	40.00	100.00	100.00
PEM	100.00	100.00	100.00
PET	0.00	60.00	100.00
NAP	93.33	100.00	100.00
NAP-R	86.67	100.00	100.00
PAND	81.82	100.00	100.00
Tau-U	0.46	0.51	0.61
Base_Tau	0.67	0.76	0.74
Delta M	2.73	4.49	13.96
Delta Trend	-0.13	0.09	0.48
SMD	2.00	3.25	10.47
Hedges g	1.83	3.00	9.80

*Note:*

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; Hedges g = Corrected SMD.

\end{table}

Finally, we conduct regression analyses for each cases with a piecewise regression model:

```
plm(strange_study$Dustin)
plm(strange_study$Mike)
plm(strange_study$Will)
```

\begin{table}

\caption{(#tab:plm\_strange\_study)Piecewise-regression model predicting variable ‘values’}

Parameter	B	CI(95%)		SE	t	p	Delta R <sup>2</sup>
		2.5%	97.5%				
Intercept	2.87	0.77	4.97	1.07	2.68	<.05	
Trend mt	0.23	-0.31	0.77	0.28	0.83	.41	.01
Level phase B	0.49	-1.58	2.56	1.06	0.46	.64	.00
Level phase C	-1.34	-10.59	7.91	4.72	-0.28	.77	.00
Slope phase B	0.02	-0.54	0.58	0.29	0.06	.95	.00
Slope phase C	-0.13	-1.02	0.77	0.46	-0.28	.78	.00

*Note:*

$F(5, 19) = 7.88$ ;  $p < .001$ ;  $R^2 = 0.675$ ; Adjusted  $R^2 = 0.589$

\end{table}

\begin{table}

\caption{(#tab:plm\_strange\_study) Piecewise-regression model predicting variable ‘values’}

Parameter	B	CI(95%)		SE	t	p	Delta R <sup>2</sup>
		2.5%	97.5%				
Intercept	0.86	-1.65	3.37	1.28	0.67	.51	
Trend mt	0.21	-0.35	0.78	0.29	0.75	.46	.01
Level phase B	-0.16	-2.84	2.51	1.36	-0.12	.90	.00
Level phase C	0.16	-9.41	9.73	4.88	0.03	.97	.00
Slope phase B	0.14	-0.46	0.75	0.31	0.46	.64	.00
Slope phase C	0.09	-1.01	1.18	0.56	0.15	.87	.00

*Note:*

$F(5, 19) = 8.00$ ;  $p < .001$ ;  $R^2 = 0.678$ ; Adjusted  $R^2 = 0.593$

\end{table}

\begin{table}

\caption{(#tab:plm\_strange\_study) Piecewise-regression model predicting variable ‘values’}

Parameter	B	CI(95%)		SE	t	p	Delta R <sup>2</sup>
		2.5%	97.5%				
Intercept	5.86	3.71	8.01	1.10	5.35	<.001	
Trend mt	-0.08	-0.46	0.30	0.19	-0.43	.67	.00
Level phase B	2.89	0.25	5.53	1.35	2.15	<.05	.01
Level phase C	14.01	7.42	20.59	3.36	4.17	<.001	.05
Slope phase B	0.99	0.52	1.47	0.24	4.10	<.001	.05
Slope phase C	0.48	-0.53	1.49	0.52	0.94	.35	.00

*Note:*

$F(5, 19) = 68.16$ ;  $p < .001$ ;  $R^2 = 0.947$ ; Adjusted  $R^2 = 0.933$

\end{table}





## Chapter 2

# Some things about R

In this chapter you will get a brief introduction to R. If you are familiar with R you might like to go directly to the next chapter.

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](http://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 too 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 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 variable name (and press return) `x`.

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

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

```
[1] 10
```

```
# 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` has an argument `x` which “expects” that you provide a 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.

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

```
[1] 4.75
```

```
# or shorter:  
mean(values)
```

```
[1] 4.75
```

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

```
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 the 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.



## Chapter 3

# The scan package

### 3.1 Installing the *scan* package

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

`install.packages("scan")` will install the stable version.

The current stable release is version 0.51. Please look at Section *Software reference* for 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 0.51 (2021-02-11)
Single-Case Data Analysis for Single and Multiple Baseline Designs
indicating that everything went smoothly and scan is ready for the job.
```

### 3.2 Development version of scan

Alternatively, you can compile the development version of *scan* yourself. This might be necessary if the stable version has some bugs or missing functions which has been fixed.

You may need some computer expertise to get the development version running. It is hosted on GitHub at <<https://github.com/jazznbass/scan>>.

For installation, you can apply the `install_github` function from the `devtools` package (make sure you have installed the `devtools` package before):

```
devtools::install_github("jazznbass/scan", dependencies = TRUE)
```

When you are running a Windows operating system you will probably have to install Rtools before. Rtools contains additional programs (e.g. compilers) that are needed to compile R source packages.

You can find Rtools here: <https://cran.r-project.org/bin/windows/Rtools/>

### 3.3 Reporting issues with *scan* and suggesting enhancements

The *scan* gitHub repository at <https://github.com/jazznbass/scan> is the ideal place to report bugs, problems, or ideas for enhancing *scan*. Please use the issue tool (direct link: <https://github.com/jazznbass/scan/issues>).

We are very thankful for any feedback, corrections, or whatever helps to improve *scan*!

### 3.4 Functions overview

The functions of the *scan* package can be divided into the following categories:

*Manage data, analyze, manipulate, simulate, and depict.*

Table 3.1 gives an overview of all functions.

### 3.5 Changes with version 0.50

#### 3.5.1 New function names

With version 0.50 *scan* introduced new names for its functions. The old function names are still usable but they will return a “deprecated” warning telling you to use the new function names.

Table 3.2 shows the changes.

#### 3.5.2 Change target variables in functions

All functions in R that analyze data now allow for temporarily changing dependent, phase, and measurement-time variables by adding three argument:

**dvar** sets the dependent variable.

**pvar** sets the phase variable.

**mvar** sets the measurement-time variable.

For example, `overlap(exampleAB_add, dvar = "depression")` will report overlap parameters for the variable *depression* while `overlap(exampleAB_add)` will take *wellbeing* as the dependent variable (as defined in the *scdf*).

After finishing the analysis, the variables are set back to their original values as defined in the *scdf*.

Table 3.1: Functions in scan

Function	What it does ...	Category
<b>scdf</b>	Creates a single-case data-frame	Manage da
<b>select_cases</b>	Selects specific cases of an scdf	Manage da
<b>subset</b>	Selects specific measurements or variables of an scdf	Manage da
<b>readSC</b>	Loads external data into an scdf	Manage da
<b>writeSC</b>	Writes scdf into an external file	Manage da
<b>scdf_attr</b>	Gets and sets attributes of an scdf	Manage da
<b>plot/plot.scdf</b>	Creates plots of single cases	Depict
<b>style_plot</b>	Defines single-case plot graphical styles	Depict
<b>export</b>	Creates html or latex tables from the output of various can functions	Depict
<b>print/print.sc</b>	Prints the results of various scan outputs	Depict
<b>print/print.scdf</b>	Prints an scdf	Depict
<b>summary/summary.scdf</b>	Summaizes an scdf	Depict
<b>autocorr</b>	Autocorrelations for each phase of each case	Analyze
<b>corrected_tau</b>	Baseline corrected tau	Analyze
<b>describe</b>	Descriptive statistics for each phase of each case	Analyze
<b>overlap</b>	An overview of overlap indeces for each case	Analyze
<b>rci</b>	Reliable change index	Analyze
<b>rand_test</b>	Randomization test	Analyze
<b>tau_u</b>	Tau-U for each case and all cases	Analyze
<b>trend</b>	Trend analyses for each case	Analyze
<b>plm</b>	Piecewise linear regression model	Analyze
<b>mplm</b>	Multivariate piecewise linear regression model	Analyze
<b>hplm</b>	Hierarchical piecewise linear regression model	Analyze
<b>nap</b>	Non-overlap of all pairs for each case	Analyze
<b>pnd</b>	Percentage of non overlapping data for each case	Analyze
<b>pand</b>	Percentage of all non overlapping data for all cases	Analyze
<b>pem</b>	Percentage exceeding the mean for each case	Analyze
<b>pet</b>	Percentage exceeding the trend for each case	Analyze
<b>outlier</b>	Detect outliers for all cases	Analyse
<b>fill_missing</b>	Interpolate missign values or missing measurement times	Manipulat
<b>ranks</b>	Covert data into ranked data	Manipulat
<b>shift</b>	Shifts values by a constant	Manipulat
<b>smooth_cases</b>	Smoothes time series data	Manipulat
<b>truncate_phase</b>	Deletes measuremts of phases	Manipulat
<b>standardize</b>	Standardizes or centres variables	Manipulat
<b>design_rSC</b>	Defines a deisgn of one or multiple single-cases	Simulate
<b>estimate_design</b>	Extracts a deisgn of one or multiple single-cases from an scdf	Simulate
<b>power_test</b>	Calculates power and alpha error of a specific analyzes for a specific single-case design	Simulate
<b>rSC</b>	Creats random single-case studies from a single-case design	Simulate

Table 3.2: scan previous and current function names.

Current function name	Previous function name
<b>autocorr</b>	autocorrSC
<b>corrected_tau</b>	corrected_tauSC
<b>describe</b> [since v0.52]	describeSC
<b>fill_missing</b>	fillmissingSC
<b>outlier</b>	outlierSC
<b>overlap</b>	overlapSC
<b>power_test</b>	power_testSC
<b>rand_test</b>	randSC; rand.test
<b>ranks</b>	rankSC
<b>rci</b>	rCi; rciSC
<b>shift</b>	shiftSC
<b>smooth_cases</b>	smoothSC
<b>style_plot</b>	style.plotSC; style_plotSC
<b>tau_u</b>	tauUSC
<b>trend</b>	trendSC
<b>truncate_phase</b>	truncateSC



## Chapter 4

# Managing single-case data

### 4.1 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 an attribute `scdf` with a list with further attributes. `var.values`, `var.phase`, and `var.mt` contain the names of the **values**, **phase**, and the **measurement time** variable. By default, these names are set to **values**, **phase**, and **mt**.

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

### 4.2 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 parameters 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)
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 consists of a single A-phase followed by a B-phase. It is a remnant from the time when `scan` could only handle sign-case designs with two phases. The number assigned to `B.start` indicates the measurement-time as defined in the `mt` argument. That is, assume a vector for the measurement times `mt = c(1,3,7,10,15,17,18,20)` and `B.start = 15` then 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 phase names can be set arbitrary, although I recommend to use capital letters (A, B, C, ...) for each phase followed by, 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 example `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.

```
# 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 4.1: Arguments of the scdf function

Argument	What it does ...
<b>values</b>	The default vector with values for the dependent variable. It can be changed with the dvar argument.
<b>phase</b>	Usually, this variable is not defined manually and will be created by the function. It is the default vector with values for the phase variable. It can be changed with the pvar argument.
<b>mt</b>	The default vector with values for the measurement-time variable. It can be changed with the mvar argument.
<b>phase.design</b>	A vector defining the length and label of each phase.
<b>B.start</b>	The first measurement of phase B (simple coding if design is strictly AB).
<b>name</b>	A name for the case.
<b>dvar</b>	The name of the dependent variable. By default this is 'values'.
<b>pvar</b>	The name of the variable containing the phase information. By default this is 'phase'.
<b>mvar</b>	The name of the variable with the measurement-time. The default is 'mt'.
<b>...</b>	Any number of variables with a vector assigned to them.

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

More variables are implemented by adding new variable names with a vector containing the values. Please be aware that a new variable must never have the same name as one of the arguments of the function (i.e. 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 4.1 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(A = 5, 7, 10, 5, 12, B = 7, 10, 18, 15, 14, 19),
  name = "Charlotte"
)
case2 <- scdf(
  values = c(A = 3, 4, 3, 5, B = 7, 4, 7, 9, 8, 10, 12),
  name = "Theresa"
)
case3 <- scdf(
  values = c(A = 9, 8, 8, 7, 5, 7, B = 6, 14, 15, 12, 16),
  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(A = 5, 7, 10, 5, 12, B = 7, 10, 18, 15, 14, 19),
  dvar = "score"
)

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

# Example: Using new names for dependent, phase, and measurement-time variables
case <- scdf(
  score = c(A = 9, 8, 8, 7, 5, 7, B = 6, 14, 15, 12, 16),
  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>

### 4.3 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.

### 4.4 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. The csv file should be build up like this:

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

In case your variables names differ from the standard (i.e. “case”, “values”, “phase”, and “mt”), you could set additional arguments to fit your file. `readSC("example.csv", cvar = "name", dvar = "wellbeing", pvar = "intervention", mvar = "time")` for example will set the variables attributes of the resulting scdf. Cases will be split by the variable “name”, “wellbeing” is set as the dependent variable (default is *values*), phase information are 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", mvar = "time", phase.names = c("A","B")
)
```

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 two 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.

## 4.5 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 5

# 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. Various arguments can be set to customize the appearance of the plot. Table 5.1 gives an overview of all available arguments.

Labels of the axes and for the phases can be changed with the `xlab`, `ylab`, and the `phase.names` arguments. 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.

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.

```
# 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 5.1: Arguments of the plot function

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

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

## 5.1 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 5.3 shows all design parameter that could be defined.

To define a new style, first create a list containing a plain design. The **style\_plot** function returns such a list with the default values for a plain design (e.g., **mystyle <- style\_plot()**).

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 be defined within the **style\_plot** function. To set a light-blue background color and also an orange grid, create the style **style\_plot(fill.bg =**

Table 5.2: Values of the lines argument

Argument	What it does ...
<b>median</b>	separate lines for the medians of each phase
<b>mean</b>	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).
<b>trend</b>	Separate lines for the trend of each phase.
<b>trendA</b>	Trend line for phase A, extrapolated throughout the other phases
<b>maxA</b>	Line at the level of the highest phase A score.
<b>minA</b>	Line at the level of the lowest phase A score.
<b>medianA</b>	Line at the phase A median score.
<b>meanA</b>	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)').
<b>movingMean</b>	Draws a moving mean curve, with a specified lag: 'lines = list(movingMean = 2)'. Default is a lag 1 curve.
<b>movingMedian</b>	Draws a moving median curve, with a specified lag: 'lines = list(movingMedian = 3)'. Default is a lag 1 curve.
<b>loreg</b>	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.
<b>lty</b>	Line type. Examples are: 'solid', 'dashed', 'dotted'.
<b>lwd</b>	Line thickness, e.g., 'lwd = 4'.
<b>col</b>	Line colour, e.g., 'col = 'red''.

Table 5.3: Arguments of the style plot function

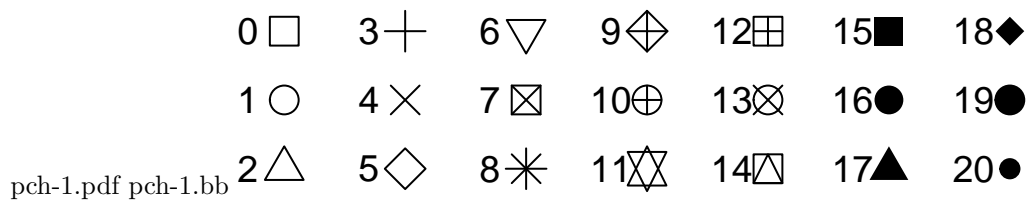
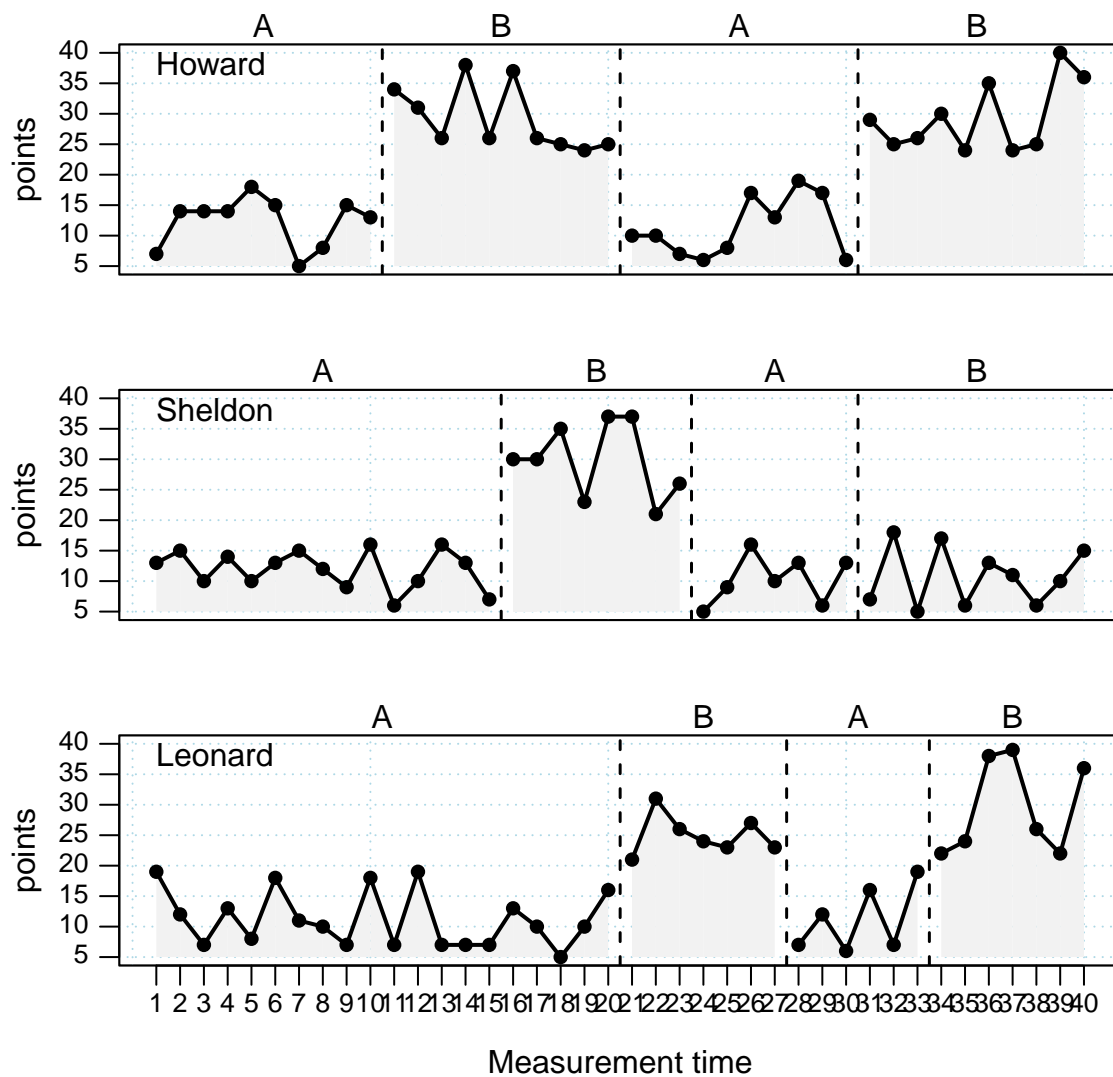
Argument	What it does ...
<b>fill</b>	If TRUE area under the line is filled.
<b>col.fill</b>	Sets the color of the area under the line.
<b>grid</b>	If TRUE a grid is included.
<b>col.grid</b>	Sets the color of the grid.
<b>lty.grid</b>	Sets the line type of the grid.
<b>lwd.grid</b>	Sets the line thickness of the grid.
<b>fill.bg</b>	If not NA the background of the plot is filled with the given color.
<b>annotations</b>	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.
<b>text.ABlag</b>	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’.
<b>lwd</b>	Width of the plot line. Default is ‘lwd = 2’.
<b>pch</b>	Point type. Default is ‘pch = 17’ (triangles). Other options are for example: 16 (filled circles) or ‘A’ (uses the letter A).
<b>col.lines</b>	The color of the lines. If set to an empty string no lines are drawn.
<b>col.dots</b>	The color of the dots. If set to an empty string no dots are drawn.
<b>mai</b>	Sets the margins of the plot.
<b>...</b>	Further arguments passed to the plot command.

"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_plot(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

```
newstyle <- style_plot(fill = "grey95", grid = "lightblue", pch = 16)
plot(exampleABAB, style = newstyle)
```

Figure 5.1: (`#fig:symbols, pch`)Some of the possible symbols and their pch values.Figure 5.2: (`#fig:custom_style_example`)A plot with a customized style.





## Chapter 6

# Describe and manipulate single-case data frames

### 6.1 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

`describe` 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).

```
describe(exampleABC)
```

```
Describe Single-Case Data
```

Table 6.1: Statistics of the describe command

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

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 <- describe(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).



### Conflicting function names

Sometimes R packages include the same function names. For example, the `describe()` function is also part of the `psych` package. Now, if you have loaded the `psych` package with `library(psych)` after `scan` the `describe()` function of `scan` will be masked (`describe()` would now call the corresponding function of the `psych` package).

There are two solutions to this problem:

1. activate the `psych` library before the `scan` library (now the `psych describe()` function will be masked) or
2. include the package name into the function call with the prefix `scan::`  
`scan::describe()`.

## 6.2 Autoregression and trendanalyses

The `autocorr` 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.

```
autocorr(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
trend(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
trend(exampleAB$Johanna, offset = 0,
      model = c("Cubic" = values ~ I(mt^3), "Log Time" = values ~ log(mt))
)
```

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

## 6.3 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:

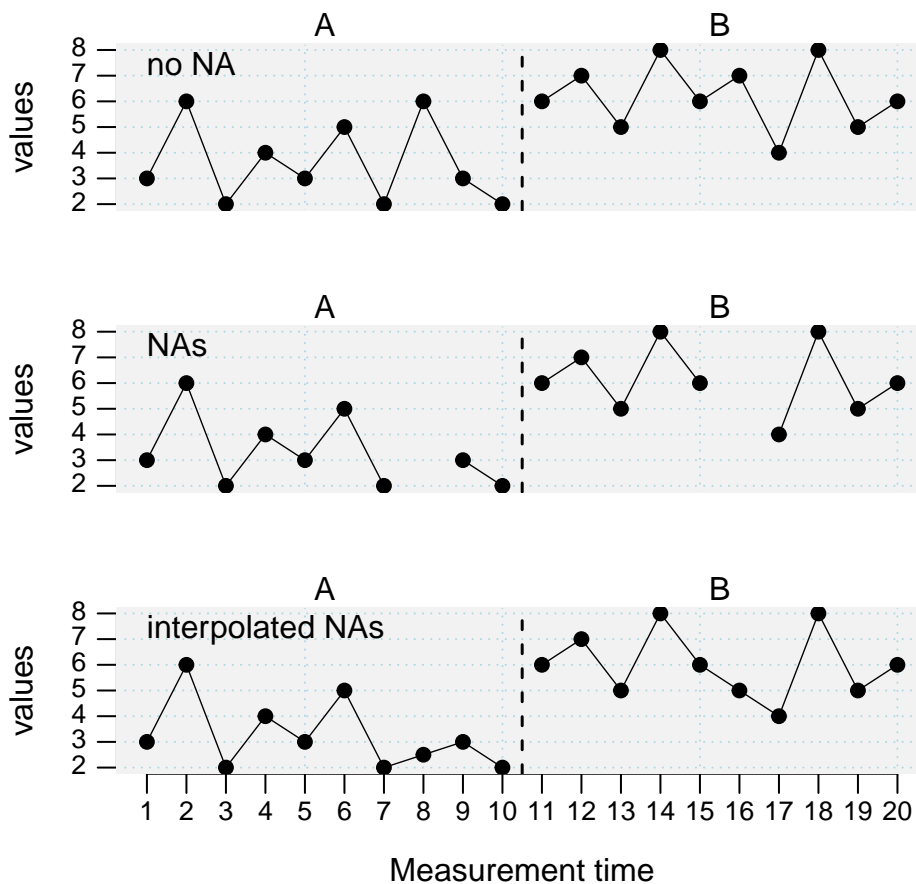
```
scdf(c(5, 3, 4, 6, 8, 7, 9, 7, NA, 6), phase.design = c(A = 4, B = 6))
```

The second type of missing occurs when there are gaps between measurement-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 interval between measurement-times 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)
```



```
overlap(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	0.45	0.51	0.50
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
Hedges_g	1.58	1.87	1.94

## 6.4 Outlieranalysis

*scan* provides several methods for analyzing outliers. All of them are implemented in the `outliers` function. Available methods are the **standard deviation**, **mean average deviation**, **confidence intervals**, 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 and Hoaglin (1993) recommend the use the much more robust median average deviation (**MAD**) instead.

The **MAD** 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 `outlier` 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")`.

```
outlier(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
```

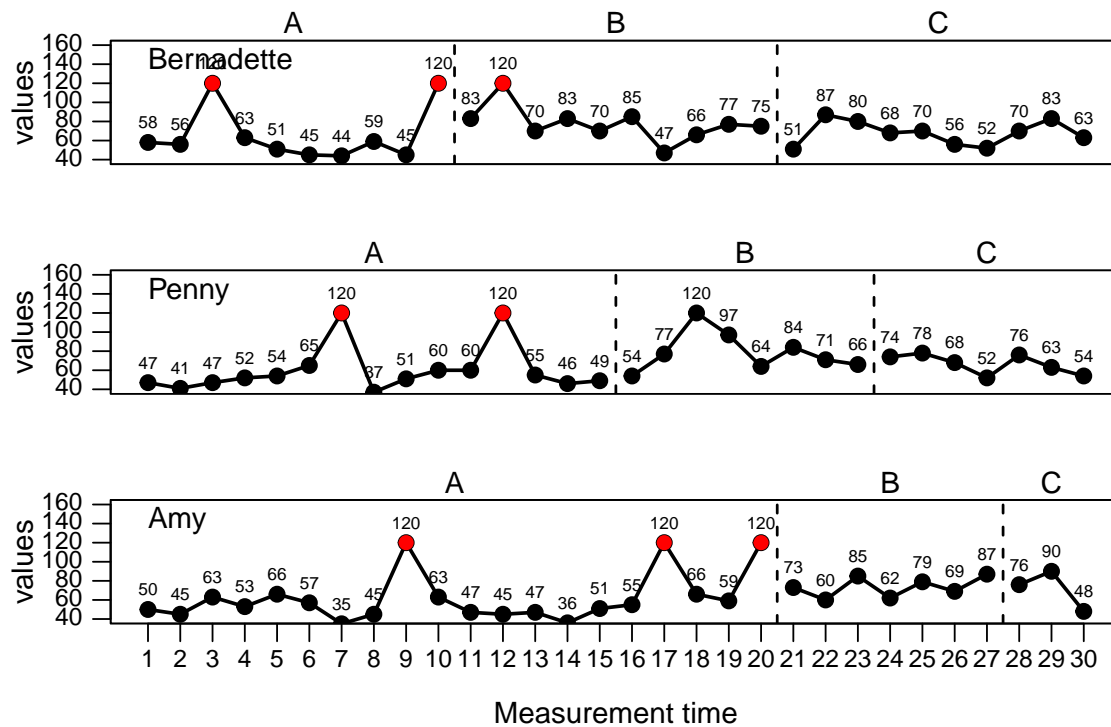
```
Case Penny : Dropped 2
```

```
Case Amy : Dropped 3
```

```
# Visualizing outliers with the plot function
```

```
res <- outlier(exampleABC_outlier, criteria = c("MAD", 3.5))
```

```
plot(exampleABC_outlier, marks = res, style = "annotate", ylim = c(40,160))
```



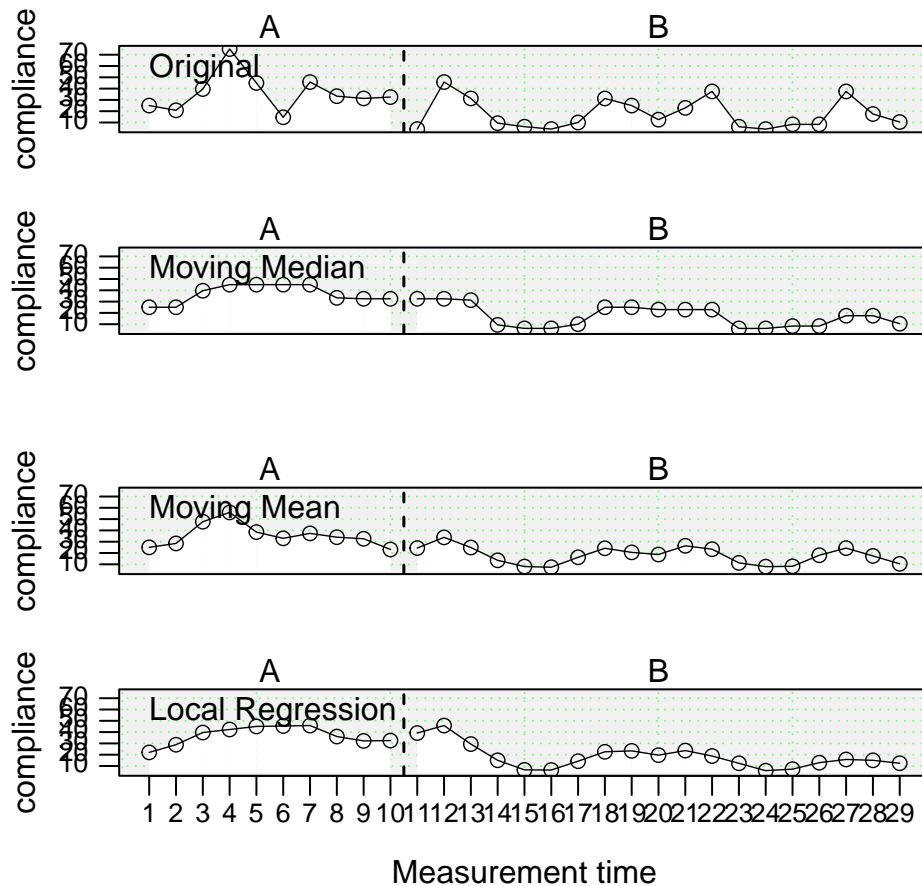
## 6.5 Smoothing data

The `smooth_cases` 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 an `scdf` with smoothed data points.

```
## Use the three different smoothing functions and compare the results
berta_mmd <- smooth_cases(Huber2014$Berta)
berta_mmn <- smooth_cases(Huber2014$Berta, FUN = "movingMean")
berta_lre <- smooth_cases(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")
plot(new_study, style = "grid2")
```





## Chapter 7

# Overlapping indices

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

```
overlap(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	0.77	0.78	0.64
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
Hedges_g	7.77	3.04	6.43

Overlap measures refer to a comparison of two phases within a single-case data-set. By default, `overlap` 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 is 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.

```
overlap(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	0.45	0.46	0.38
Base_Tau	0.65	0.68	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
Hedges_g	2.63	3.21	3.55

## 7.1 Percentage non-overlapping data (PND)

The percentage of non-overlapping data (PND) effect size measure was described by Scruggs et al. (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

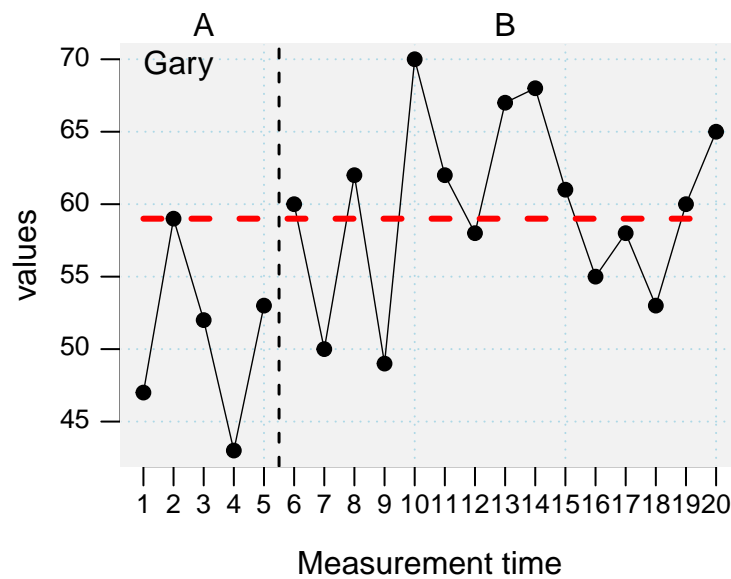


Figure 7.1: Illustration of PND. PND is 60% as 9 out of 15 datapoints of phase B are higher than the maximum of phase A.

Mean : 93.33 %

## 7.2 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%

## 7.3 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.

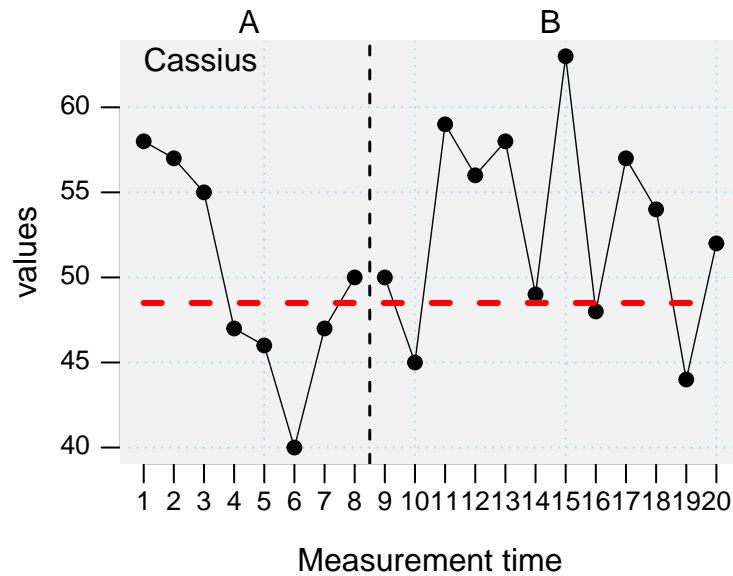


Figure 7.2: Illustration of PEM. PEM is 75% as 9 out of 12 datapoints of phase B are higher than the median of phase A.

```
pet(exampleAB)
```

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 above

## 7.4 Percentage of all non-overlapping data (PAND)

The `pand` function calculates the percentage of all non-overlapping data (Parker et al., 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 %

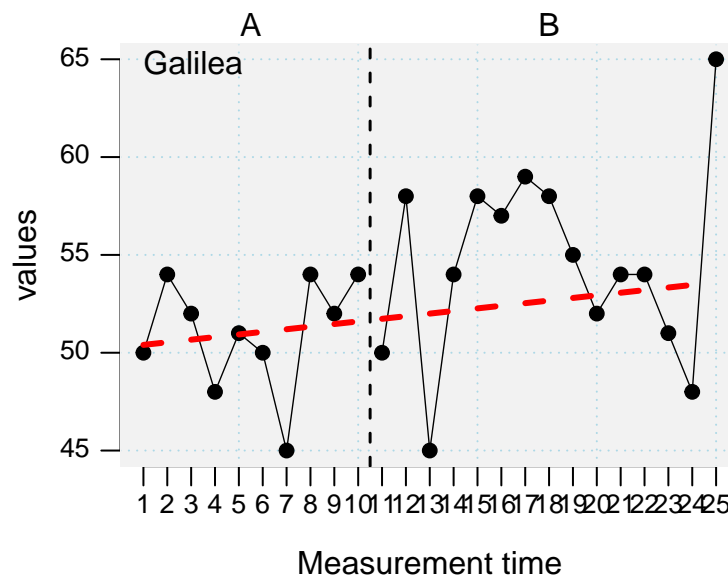


Figure 7.3: Illustration of PET. PET is 66.7% as 10 out of 15 datapoints of phase B are higher than the projected trend-line of phase A.

$$\Phi = 0.822 \quad ; \quad \Phi^2 = 0.676$$

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
%	A	21.7	3.3
real B	B	3.3	71.7
	total	25	75

2 x 2 Matrix of counts

		expected	
		A	B
A	A	13	2
real B	B	2	43
	total	15	45

Note. Matrix is corrected for ties

Correlation based analysis:

$$z = 6.316, \quad p = 0.000, \quad \Phi = 0.822$$

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 (Parker et al., 2011a).

## 7.5 Nonoverlap of all pairs (NAP)

The `nap` function calculates the nonoverlap of all pairs (Parker and 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 Wilcoxon rank sum test. Thus, a Wilcoxon 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

## 7.6 Tau-U

The `tau_u` 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 Parker et al. (2011b).

The `tau_u` 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. (2011b). 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. (2011b) 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. (2011b).

The standard return of the `tau_u` 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 <- tau_u(dat)
print(res, complete = TRUE)
```

Tau-U

Method: complete

Applied Kendall's Tau-b

Case: Case1

	pairs	pos	neg	ties	S	D	Tau	SD_S	VAR_S	SE_Tau
A vs. B	20	19	1	0	18	20.0	0.900	8.16	66.67	0.408
Trend A	6	4	2	0	2	6.0	0.333	2.94	8.67	0.491
Trend B	10	8	2	0	6	10.0	0.600	4.08	16.67	0.408
A vs. B - Trend A	26	21	5	0	16	30.6	0.523	8.68	75.33	0.284
A vs. B + Trend B	30	27	3	0	24	32.9	0.730	9.13	83.33	0.278
A vs. B + Trend B - Trend A	36	29	7	0	22	36.0	0.611	9.59	92.00	0.266
	Z	p								
A vs. B	2.205	0.02749								
Trend A	0.679	0.49691								
Trend B	1.470	0.14164								
A vs. B - Trend A	1.843	0.06527								
A vs. B + Trend B	2.629	0.00856								
A vs. B + Trend B - Trend A	2.294	0.02181								

tau\_u(exampleAB)

Tau-U

Method: complete

Applied Kendall's Tau-b

Overall Tau-U:

	Model	Tau_U	se	z	p
A vs. B	0.969	0.1772	5.47	4.54e-08	
A vs. B - Trend A	0.590	0.1064	5.54	3.04e-08	
A vs. B + Trend B	0.740	0.0960	7.71	1.29e-14	
A vs. B + Trend B - Trend A	0.731	0.0942	7.75	9.09e-15	

Case: Johanna

	Tau	SE_Tau	Z	p
A vs. B	1.000	0.306	3.27	0.001
A vs. B - Trend A	0.592	0.184	3.22	0.001
A vs. B + Trend B	0.786	0.166	4.75	0.000
A vs. B + Trend B - Trend A	0.765	0.163	4.71	0.000

Case: Karolina

	Tau	SE_Tau	Z	p
A vs. B	0.940	0.308	3.06	0.002
A vs. B - Trend A	0.554	0.184	3.01	0.003
A vs. B + Trend B	0.805	0.166	4.85	0.000
A vs. B + Trend B - Trend A	0.783	0.163	4.81	0.000

Case: Anja

	Tau	SE_Tau	Z	p
A vs. B	0.966	0.308	3.14	0.002

A vs. B - Trend A	0.624	0.186	3.36	0.001
A vs. B + Trend B	0.626	0.167	3.74	0.000
A vs. B + Trend B - Trend A	0.642	0.164	3.91	0.000

## 7.7 Reliable change index

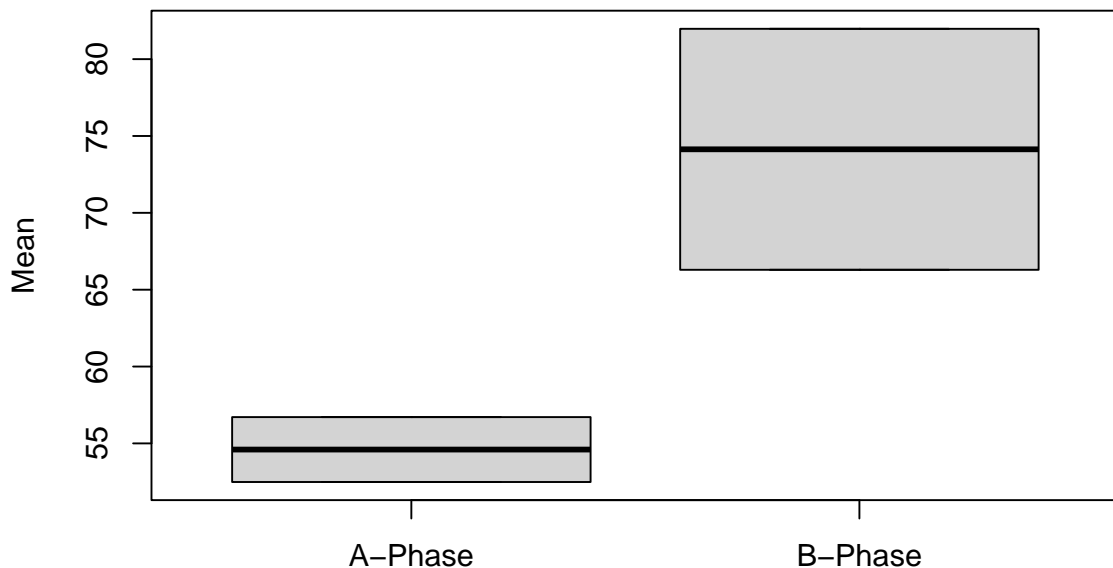
Basically, the reliable change index (rci) depicts if a post-test is above a pre-test value. Based on the reliability of the measurements and the standard-deviation the standard error is calculated. The mean difference between phase-A and phase-B is divided by the standard-error.

Several authors proposed refined methods for calculating the rci.

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

```
rci(exampleAB$Johanna, rel = 0.8, graph = TRUE)
```

### 95% confidence interval (rtt = 0.80)



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



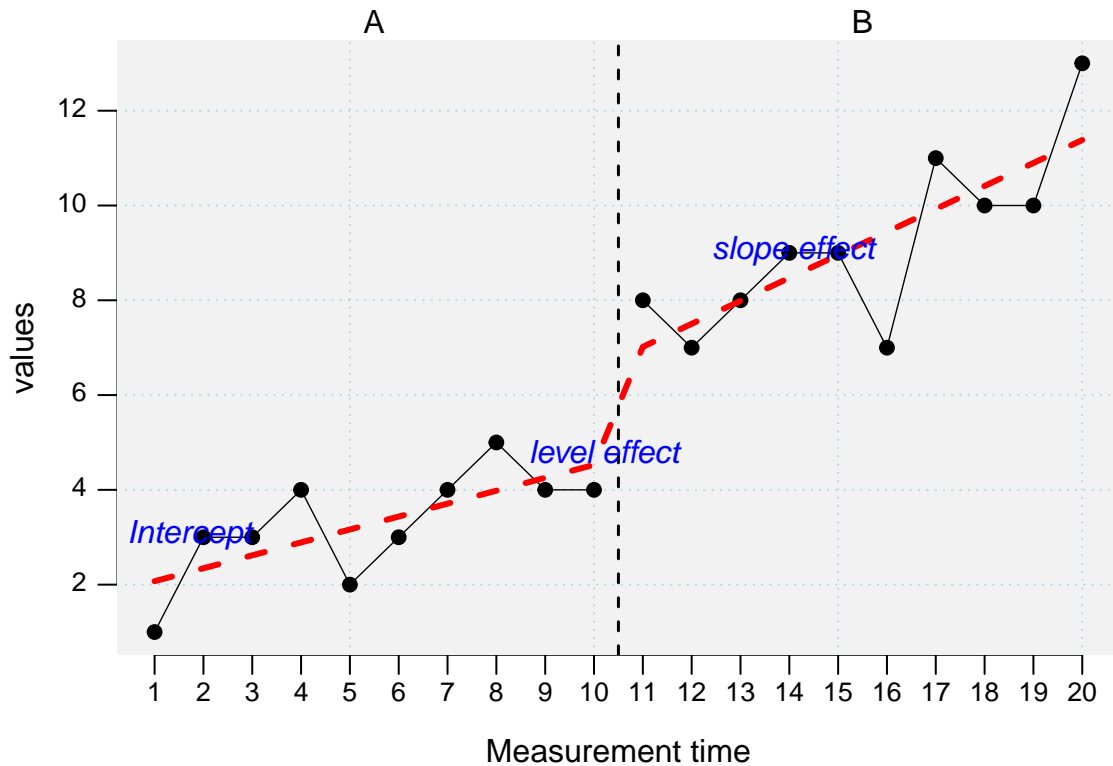
## Chapter 8

# 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 and 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 parameters predict the outcome at a specific measurement point:

1. The performance at the beginning of the study (**intercept**),
2. a developmental effect leading to a continuous increase throughout all measurements (**trend effect**),
3. an intervention effect leading to an immediate and constant increase in performance (**level effect**), and
4. a second intervention effect that evolves continuously with the beginning of the intervention (**slope effect**).



*scan* provides an implementation based on this piecewise regression approach. Though the original model is extended by several factors:

- multiple phase designs
- additional (control) variables
- autoregression modeling
- 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.

## 8.1 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.

$F(3, 16) = 28.69$ ;  $p = 0.000$ ;  $R^2 = 0.843$ ; Adjusted  $R^2 = 0.814$

	B	2.5%	97.5%	SE	t	p	delta R <sup>2</sup>
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

### 8.1.1 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 *BEL-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
C	7	1	1	6	1
C	5	1	1	7	2
C	6	1	1	8	3
C	4	1	1	9	4
C	8	1	1	10	5

### 8.1.2 Adjusting the model

```
example <- scdf(
  values = c(55, 58, 53, 50, 52, 55, 68, 68, 81, 67, 78, 73, 72, 78, 81, 78, 71, 85, 80, 76)
  phase.design = c(A = 5, B = 15)
)

plm(example)
```

Piecewise Regression Analysis

Dummy model: B&L-B

Fitted a gaussian distribution.

$F(3, 16) = 21.36$ ;  $p = 0.000$ ;  $R^2 = 0.800$ ; Adjusted  $R^2 = 0.763$

	B	2.5%	97.5%	SE	t	p	delta	$R^2$
Intercept	57.800	46.521	69.079	5.755	10.044	0.000		
Trend mt	-1.400	-4.801	2.001	1.735	-0.807	0.432	0.0081	
Level phase B	14.467	4.291	24.642	5.192	2.786	0.013	0.0970	
Slope phase B	2.500	-0.961	5.961	1.766	1.416	0.176	0.0250	

Autocorrelations of the residuals

lag	cr
1	-0.28
2	0.05
3	-0.11

Formula:  $\text{values} \sim 1 + \text{mt} + \text{phaseB} + \text{interB}$

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(example, slope = FALSE)
```

Piecewise Regression Analysis

Dummy model: B&L-B

Fitted a gaussian distribution.

$F(2, 17) = 29.30$ ;  $p = 0.000$ ;  $R^2 = 0.775$ ; Adjusted  $R^2 = 0.749$

	B	2.5%	97.5%	SE	t	p	delta	$R^2$
Intercept	50.559	45.239	55.878	2.714	18.627	0.000		
Trend mt	1.014	0.364	1.664	0.332	3.057	0.007	0.1236	
Level phase B	10.329	1.674	18.983	4.416	2.339	0.032	0.0724	

Autocorrelations of the residuals

lag	cr
1	-0.07
2	0.06
3	-0.17

Formula:  $\text{values} \sim 1 + \text{mt} + \text{phaseB}$

In the resulting estimations the trend and level effects are now significant. The model estimated a trend effect of 1.01 points per measurement time and a level effect of 2.46 points.

That is, with the beginning of the intervention (the B-phase) the score increases by 15.38 points ( $5 \times 1.01 + 10.33$ ).

### 8.1.3 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.

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

Piecewise Regression Analysis

Dummy model: B&L-B

Fitted a gaussian distribution.

$F(4, 35) = 5.87$ ;  $p = 0.001$ ;  $R^2 = 0.402$ ; Adjusted  $R^2 = 0.333$

	B	2.5%	97.5%	SE	t	p	delta	$R^2$
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.

### 8.1.4 [to be written] Modelling autoregression

```
autocorr(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

```

## 8.2 [to be written] Multivariate piecewise regression

```
mplm(exampleAB_add, dvar = c("wellbeing", "depression"))
```

Registered S3 methods overwritten by 'car':

```

method          from
influence.merMod lme4
cooks.distance.influence.merMod lme4
dfbeta.influence.merMod lme4
dfbetas.influence.merMod lme4

```

Multivariate piecewise linear model

Dummy model: B&L-B

Coefficients:

```

                wellbeing depression
(Intercept)    48.0380952  4.0857143
day             0.3785714  0.1142857
Level Phase Medication 3.8633333 -0.7800000
Slope Phase Medication -0.2754945 -0.1650549

```

Formula: y ~ 1 + day + phaseMedication + interMedication

Type III MANOVA Tests: Pillai test statistic

```

                Df test stat approx F num Df den Df Pr(>F)
(Intercept)      1  0.89679  152.053      2    35 <2e-16 ***
day              1  0.05450   1.009      2    35  0.3750
Level Phase Medication 1  0.03846   0.700      2    35  0.5034
Slope Phase Medication 1  0.03911   0.712      2    35  0.4975

```

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

## 8.3 Multilevel plm 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 familiar with `plm` before

applying an `hplm`.

Here is a simple example:

```
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 including 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

### 8.3.1 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.12` 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 `example50.12`. Here are the first 10 cases of `example50.12`.

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 7.5 points lower for male subjects. While the level-effect is 16.8 points for female subjects it is  $16.8 - 7.5 = 9.3$  for males.



## Chapter 9

# Randomization tests

The `rand_test` 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 overwrite the limit-parameter.

*exclude.equal*

If set to 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.

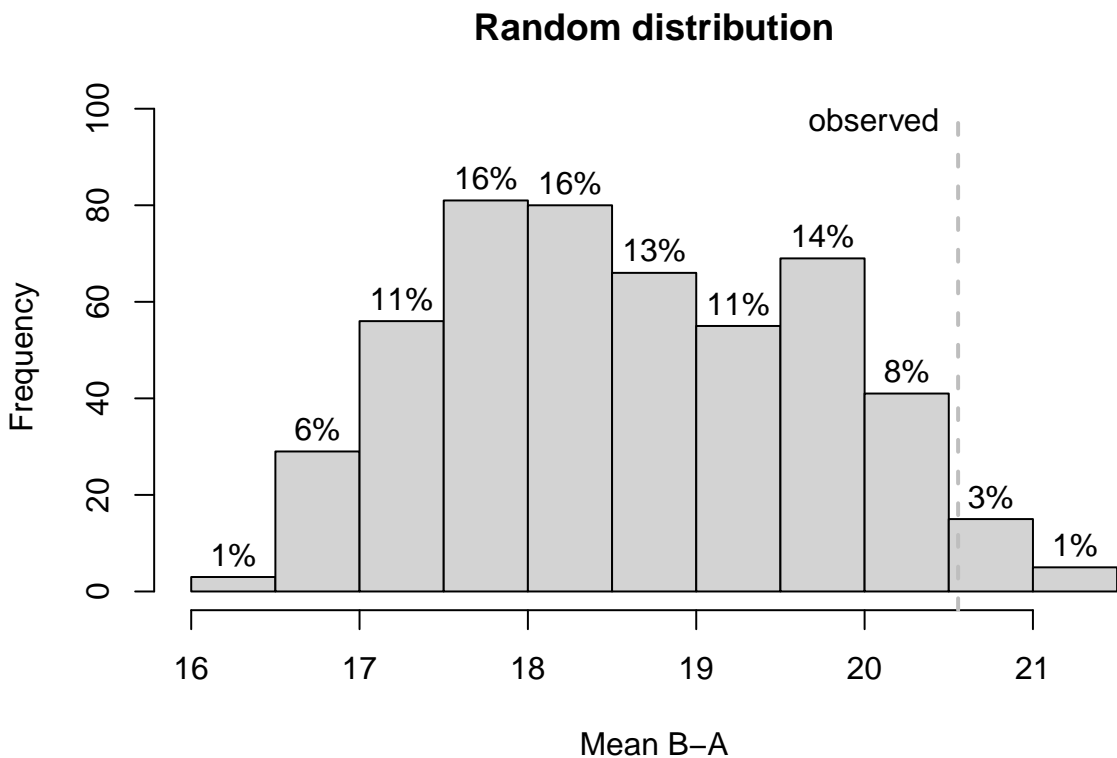
*graph*

If set TRUE, a histogram of the resulting distribution is plotted.

*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")`.

```
rand_test(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 = 18.59305  
SD = 1.114388  
Min = 16.05185  
Max = 21.34493

p = 0.036

Shapiro-Wilk Normality Test:  $W = 0.979$ ;  $p = 0.000$  (Hypothesis of Normality rejected)  
 $z = 1.7611$ ,  $p = 0.0391$  (single sided)



## Chapter 10

# Default settings

Some of the default settings of `scan` can be changed with the `options()` argument. Table 10.1 shows a complete list of options and their default values.

```
# get the current value of an option  
getOption("scan.print.rows")
```

```
[1] 15
```

```
# set option to a different value  
options(scan.print.rows = 5, scan.print.scdf.name = FALSE)  
print(exampleAB)
```

```
#A single-case data frame with 3 cases
```

phase	values	mt	phase	values	mt	phase	values	mt
A	54	1	A	41	1	A	55	1
A	53	2	A	59	2	A	58	2
A	56	3	A	56	3	A	53	3
A	58	4	A	51	4	A	50	4
A	52	5	A	52	5	A	52	5

```
# ... up to 15 more rows
```

```
options(scan.print.rows = 15, scan.print.scdf.name = TRUE)  
print(exampleAB)
```

```
#A single-case data frame with 3 cases
```

Johanna: phase	values	mt	Karolina: phase	values	mt	Anja: phase	values	mt
A	54	1	A	41	1	A	55	1
A	53	2	A	59	2	A	58	2
A	56	3	A	56	3	A	53	3
A	58	4	A	51	4	A	50	4
A	52	5	A	52	5	A	52	5
B	61	6	B	57	6	B	55	6
B	62	7	B	56	7	B	68	7

Table 10.1: Scan Options

Option	Default	What it does ...
<code>scan.print.cases</code>	"fit"	Max number of cases printed for scdf
<code>scan.print.rows</code>	15	Max number of rows printed for scdf
<code>scan.print.cols</code>	"all"	Max number of columns printed for scdf
<code>scan.print.digits</code>	1	Max number of digits printed for scdf
<code>scan.print.long</code>	FALSE	If TRUE, prints scdf objects in long
<code>scan.print.scdf.name</code>	TRUE	If TRUE, prints case names of scdf
<code>scan.deprecated.warning</code>	FALSE	When TRUE returns information on
<code>scan.export.kable</code>	list(digits = 2, linesep = "", booktab = TRUE)	List with default arguments for the k
<code>scan.export.kable__styling</code>	list(bootstrap_options = c("bordered", "condensed"), full_width = FALSE, position = "center")	List with default arguments for the k
<code>scan.plot.style</code>	"grid"	NA

```

      B      71  8 |      B      67  8 |      B      68  8 |
      B      66  9 |      B      75  9 |      B      81  9 |
      B      64 10 |      B      66 10 |      B      67 10 |
      B      78 11 |      B      69 11 |      B      78 11 |
      B      70 12 |      B      68 12 |      B      73 12 |
      B      74 13 |      B      73 13 |      B      72 13 |
      B      82 14 |      B      77 14 |      B      78 14 |
      B      77 15 |      B      79 15 |      B      81 15 |
# ... up to 5 more rows

```

## Chapter 11

# Exporting *scan* results

The `export` function will make it easier to convert the results of your `scan` analyses into tables and descriptions you can add to your documents and presentations. Basically, `export` takes a `scan` object and converts it to an 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!]

### 11.1 Single case data files

```
export(exampleA1B1A2B2_zvt)
```

### 11.2 Descriptive stats

```
res <- describe(GruenkeWilbert2014)
export(res)
```

### 11.3 Overlap indices

```
res <- overlap(exampleA1B1A2B2_zvt, phases = list(c(1,3),c(2,4)))
export(res, flip = TRUE)
```

Table 11.1: Single case data frame with 3 cases

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

Table 11.2: 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

*Note:*

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 dependent variable regressed on measurement-time.

Table 11.3: 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	0.02	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
Hedges g	0.48	0.37	0.24

*Note:*

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; Hedges g = Corrected SMD.

Table 11.4: Piecewise-regression model predicting variable 'values'

Parameter	B	CI(95%)		SE	t	p	Delta R <sup>2</sup>
		2.5%	97.5%				
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  $R^2 = 0.552$

## 11.4 Piecewise linear models

```
res <- plm(exampleA1B1A2B2$Pawel)
export(res)
```

## 11.5 Hierarchical piecewise regressions

```
res <- hplm(exampleAB_50, data.12 = exampleAB_50.12, lr.test = TRUE, random.slopes = TRUE)
export(res)
```



Table 11.5: Hierarchical Piecewise Linear Regression predicting variable 'values'

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
<b>Random effects</b>					
	<b>SD</b>	<b>L</b>	<b>df</b>	<b>p</b>	
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&amp;L-B; 50 cases



# About the author

Currently, I am a professor for research methods and diagnostics at the department of inclusive education at the University of Potsdam in Germany. I studied education sciences at the University of Cologne where I also did my PhD in psychology. Thereafter, I got a tenured position as a senior researcher at the department of special education (also University of Cologne). Later I did my habilitation on “Pedagogic and psychology in learning disabilities” at the *Carl von Ossietzky University* Oldenburg.

My current work focuses on:

- Single-case research designs, analyzing single case data, and reporting single-case based results.
- Social inclusion and social participation in classrooms.
- Implementation of Open Science and Data Science concepts into special education research.

My work can be found at:

- University of Potsdam: <https://www.uni-potsdam.de/en/inklusion/researchmethodsanddiagnostics/wilbert.html>
- Researchgate: [https://www.researchgate.net/profile/Juergen\\_Wilbert2](https://www.researchgate.net/profile/Juergen_Wilbert2)
- github: <https://github.com/jazznbass>
- orcid: <https://orcid.org/0000-0002-8392-2873>
- Googel scholar: <https://scholar.google.de/citations?user=RF7Siq4AAAAJ&hl=en>
- Empirische Sonderpädagogik: <http://www.psychologie-aktuell.com/index.php?id=255>
- OSF (Open science framework): <https://osf.io/q7gme/>



# References



# Bibliography

- Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., and Iannone, R. (2021). *rmarkdown: Dynamic Documents for R*. R package version 2.8.
- Huitema, B. E. and McKean, J. W. (2000). Design specification issues in time-series intervention models. *Educational and Psychological Measurement*, 60(1):38–58.
- Iglewicz, B. and Hoaglin, D. C. (1993). *How to detect and handle outliers*. Milwaukee, Wis. : ASQC Quality Press.
- Parker, R. I., Hagan-Burke, S., and Vannest, K. (2007). Percentage of All Non-Overlapping Data (PAND) An Alternative to PND. *The Journal of Special Education*, 40(4):194–204.
- Parker, R. I. and Vannest, K. (2009). An improved effect size for single-case research: Nonoverlap of all pairs. *Behavior Therapy*, 40(4):357–367.
- Parker, R. I., Vannest, K. J., and Davis, J. L. (2011a). Effect Size in Single-Case Research: A Review of Nine Nonoverlap Techniques. *Behavior Modification*, 35(4):303–322.
- Parker, R. I., Vannest, K. J., Davis, J. L., and Sauber, S. B. (2011b). Combining Nonoverlap and Trend for Single-Case Research: Tau-U. *Behavior Therapy*, 42(2):284–299.
- R Core Team (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.
- RStudio Team (2018). *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA.
- Scruggs, T. E., Mastropieri, M. A., and Casto, G. (1987). The Quantitative Synthesis of Single-Subject Research Methodology and Validation. *Remedial and Special Education*, 8(2):24–33.
- Wilbert, J. and Lueke, T. (2021). *scan: Single-Case Data Analyses for Single and Multiple Baseline Designs*. <https://github.com/jazznbass/scan>, <https://jazznbass.github.io/scan-Book>.
- 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.
- Xie, Y. (2021). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.22.