Analyzing single-case data with R and scan

Jürgen Wilbert

2022-05-18

Contents

Welcome

Note: The cover has been designed by Tony Wilbert and Henry Ritter.

Thanx for that!

Preface

Hello!

I am glad your found your way to this book as is tells me you are beginning to use the scan package. While `scan` is quiet thoroughly developed, this book is at an early stage (about 30% is done). 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

18 May 2022

Software reference

This book has been created using the `Rmarkdown` (**?**) and `bookdown` (**?**) packages within the RStudio (**?**) environment. The analyses have been conducted with the **R** package `scan` at version 0.54.3 (**?**). R version 4.2.0 (2022-04-22) was used (**?**).

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` (**?**) 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`.

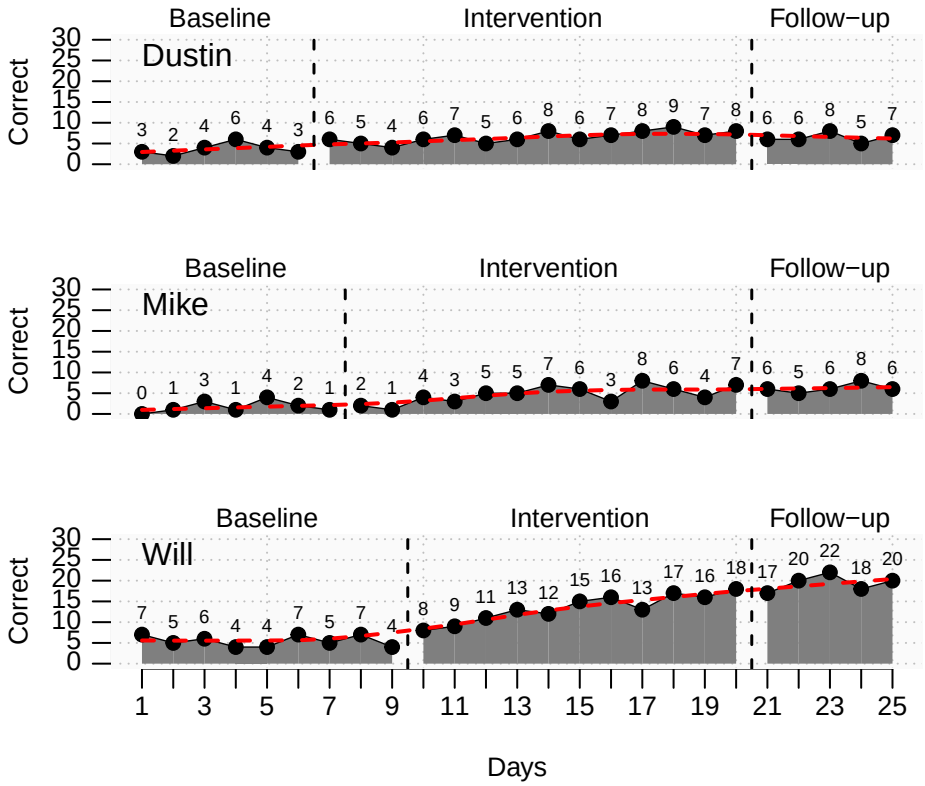
Let us set a fictional context. Let us assume you are researching on a method to foster the calculation abilities of struggling fourth grade students. You developed an intervention program named *KUNO*. In a pilot study you like to get some evidence on the effectiveness of that new method and you set 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 to 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)
```

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

```
overlap(strange_study)
```

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

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

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)
```

Table 1.1: Descriptive statistics

Parameter	Dustin	Mike	Will
Design	A-B-C	A-B-C	A-B-C
n A	6	7	9
n B	14	13	11
n C	5	5	5
Missing A	0	0	0
Missing B	0	0	0
Missing C	0	0	0
m A	3.67	1.71	5.44
m B	6.57	4.69	13.45
m C	6.4	6.2	19.4
md A	3.5	1.0	5.0
md B	6.5	5.0	13.0
md C	6	6	20
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	0	4
Min B	4	1	8
Min C	5	5	17
Max A	6	4	7
Max B	9	8	18
Max C	8	8	22
Trend A	0.23	0.21	-0.08
Trend B	0.25	0.36	0.91
Trend C	0.1	0.3	0.4

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.

Table 1.2: Overlap indices. Comparing phase 1 against phase 2

	Dustin	Mike	Will
Design	A-B-C	A-B-C	A-B-C
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	80	100
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	2.00	1.51	2.96

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 1.3: Overlap indices. Comparing phase A against phase C

	Dustin	Mike	Will
Design	A-B-C	A-B-C	A-B-C
PND	40	100	100
PEM	100	100	100
PET	0	60	100
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	1.97	3.25	8.34

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 1.4: Piecewise-regression model predicting variable 'values'

		CI(95%)					
Parameter	B	2.5%	97.5%	SE	t	p	Delta R^2
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 $R^2 = 0.589$

Table 1.5: Piecewise-regression model predicting variable 'values'

		CI(95%)					
Parameter	B	2.5%	97.5%	SE	t	p	Delta R^2
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 $R^2 = 0.593$

Table 1.6: Piecewise-regression model predicting variable 'values'

		CI(95%)					
Parameter	B	2.5%	97.5%	SE	t	p	Delta R^2
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 $R^2 = 0.933$

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. Yet, it is as reliable as other proprietary software like Mplus, STATA, SPSS etc . You can tell from my writing that is hard not to become an R-fan when you are into statistics :-)
R can be used in at least two ways:

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

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

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

2.1 Basic R

R is a script language. That is, you type in text and let R execute the commands you wrote down. Either you work in a *console* or a *textfile*. In a *console* the command will be executed every time you press the RETURN-key. In 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 `0.54.1`. 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 0.54.1 (2022-04-03)
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 **??** gives an overview of all functions. Furthermore, you can see the current life cycle stage of a function. The life cycle stage categorization is based on the tidyverse package and described in detail here https://lifecycle.r-lib.org/articles/stages.html.

Table 3.1: Functions in scan.

Function	What it does ...	Category
scdf	Creates a single-case data-frame	Manage da
select_cases	Selects specific cases of an scdf	Manage da
select_phases	Selects and/or recombines phases	Manage da
subset	Selects specific measurements or variables of an scdf	Manage da
read_scdf	Loads external data into an scdf	Manage da
write_scdf	Writes scdf into an external file	Manage da
convert	Converts an scdf object into R syntax	Manage da
set_var	(Re)sets dependent, measurement, and phase variable of an scdf	Manage da
scdf_attr	Gets and sets attributes of an scdf	Manage da
add_l2	Adds level-two data to an scdf	Manage da
as.data.frame/as.data.frame.scdf	Transforms an scdf into a data frame	Manage da
plot/plot.scdf	Creates plots of single cases	Depict
style_plot	Defines single-case plot graphical styles	Depict
export	Creates html or latex tables from the output of various can functions	Depict
print/print.sc	Prints the results of various scan outputs	Depict
print/print.scdf	Prints an scdf	Depict
summary/summary.scdf	Summaizes an scdf	Depict
plot_rand	Create a distribution plot from a randomization test obejct	Depict
autocorr	Autocorrelations for each phase of each case	Analyze
corrected_tau	Baseline corrected tau	Analyze
describe	Descriptive statistics for each phase of each case	Analyze
overlap	An overview of overlap indeces for each case	Analyze
smd	Various standardized mean differences between phase A and B	Analyze
rci	Reliable change index	Analyze
rand_test	Randomization test	Analyze
tau_u	Tau-U for each case and all cases	Analyze
trend	Trend analyses for each case	Analyze
plm	Piecewise linear regression model	Analyze
mplm	Multivariate piecewise linear regression model	Analyze
hplm	Hierarchical piecewise linear regression model	Analyze
nap	Non-overlap of all pairs for each case	Analyze
pnd	Percentage of non overlapping data for each case	Analyze
pand	Percentage of all non overlapping data for all cases	Analyze
pem	Percantage exceeding the mean for each case	Analyze
pet	Percentage exceeding the trend for each case	Analyze
cdc	Conservative dual-criterion test	Analyze
outlier	Detect outliers for all cases	Analyse
fill_missing	Interpolate missign values or missing measurement times	Manipulat
ranks	Covert data into ranked data across all cases	Manipulat
transform	Change and create new variabes	Manipulat
smooth_cases	Smoothes time series data	Manipulat
truncate_phase	Deletes measurements of phases	Manipulat
standardize	Standardizes or centers variables across cases	Manipulat
design	Defines a design of one or multiple single-cases	Simulate
power_test	Calculates power and alpha error of a specific analyzes for a specific single-case design	Simulate
estimate_design	Extraxt a deisgn template from an existing scdf	Simulate
random_scdf	Creats random single-case studies from a single-case design	Simulate

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` (not available).

```
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 **??** 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)
```

Table 4.1: Arguments of the scdf function

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

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. `read_scdf` 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:

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. `read_scdf("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 `read_scdf("example.csv", phase.names = c("A", "B"))`.

```
dat <- read_scdf(
  "example2.xlsx", cvar = "name", pvar = "intervention",
  dvar = "wellbeing", mvar = "time", phase.names = c("A","B")
)
```

```
Loaded 20 cases.
```

	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

```
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:

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

`read_scdf` also 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. Excel files will be automatically detected by the filename extension `xls`or `xlsx` or by explicitly setting the `type` argument (e.g. `type = "xlsx"`).

`write_scdf()` 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, `write_scdf(example, file = "example.csv", dec = ",", sep = ";")` exports a csv variation usually used for example in Germany.

4.5 Convert an scdf object back to scan syntax

You can also reconvert an scdf object back to "raw" scan syntax. This is a convenient way when you imported data from an Excel or csv file and want to keep everything clean and transparent within your R syntax files.

Here is an example:

```
convert(exampleABC)
```

```
case1 <- scdf(
   values = c(58, 56, 60, 63, 51, 45, 44, 59, 45, 39, 83, 65, 70, 83, 70, 85, 47, 66, 77, 75,
   phase_design = c(A = 10, B = 10, C = 10),
   name = "Marie"
)

case2 <- scdf(
   values = c(47, 41, 47, 52, 54, 65, 55, 37, 51, 60, 60, 65, 55, 46, 49, 54, 77, 73, 97, 64,
   phase_design = c(A = 15, B = 8, C = 7),
   name = "Rosalind"
)

case3 <- scdf(
   values = c(50, 45, 63, 53, 66, 57, 35, 45, 74, 63, 47, 45, 47, 36, 51, 55, 35, 66, 59, 55,
   phase_design = c(A = 20, B = 7, C = 3),
   name = "Lise"
)

study <- c(case1, case2, case3)
```

Now you can copy and past the output into your R file or you set the `file` argument to save the output into an R file `convert(exampleABC, file = "scdf.R")`.

4.6 Displaying 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
```

4.7 Selecting cases and measurements

4.7.1 Subsetting cases with base R syntax

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
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
```

4.7.2 Select cases

Since version 0.53 scan includes some functions to work with pipe-operators. Therefore, we will provide syntax examples with and without pipe operators.

The `select_cases()` function takes case-names and/or numbers for selecting cases:

```
# With pipes:
Huber2014 %>%
  select_cases("Adam", "Berta", 4) %>%
  summary()
```

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

       Measurements Design
 Adam            37    A-B
 Berta           29    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
```

```
# Without pipes:

# new_huber <- select_cases(Huber2014, "Adam", "Berta", 4)
# summary(new_huber)
```

4.7.3 Select measurements

The `subset()` function helps with extracting measurements (or rows) by a specific criteria from a scdf.

Subset takes a scdf as its first argument and a logical expression as the second argument (`filter`). Only measurements for which the logical argument is evaluated to be TRUE are inlcuded in the returning scdf object.

For example, the scdf `Huber2014` has a variable `compliance` and we like to keep measurements where `compliance` is larger than 10 because we assume the others to be outliers:

```
Huber2014 %>%
  subset(compliance > 10) %>%
  summary()
```

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

           Measurements Design
 Adam                37    A-B
 Berta               20    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
```

In an more complex example, we only like to keep values lower than 60 when they are in phase A or values equal or larger than 60 when they are in phase B:

```
exampleAB %>%
  subset((values < 60 & phase == "A") | (values >= 60 & phase == "B")) %>%
  summary()
```

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

          Measurements Design
 Johanna            20    A-B
 Karolina           18    A-B
 Anja               19    A-B

Variable names:
values <dependent variable>
mt <measurement-time variable>
phase <phase variable>


Note:  Randomly created data with normal distributed dependent variable.
```

4.8 Change and create variables

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 **??** gives an overview of all available arguments.

```
plot(exampleA1B1A2B2_zvt)
```

5.1 Plot axis

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. The argument `xinc` sets the incremental steps for the x-axis ticks with corresponding values. For example `xinc = 1` will set a tick for every measurement time increase of 1 while `xinc = 5` will only set every ffith tick.

```
plot(
  exampleABC,
  phase.names = c("Baseline", "Intervention", "Follow-Up"),
  case.names = c("First", "Second", "Third"),
  ylab = "Frequency",
  xlab = "Days",
  main = "An example",
  ylim = c(0, 120),
  xinc = 2
)
```

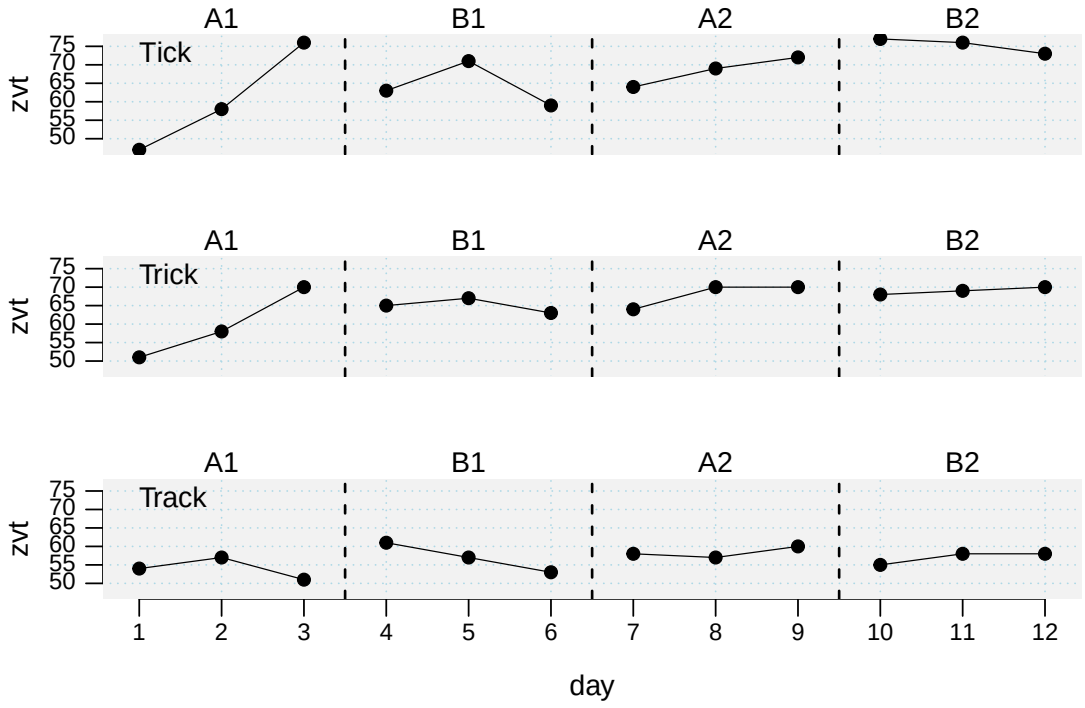


Figure 5.1: A simple plot does not need much.

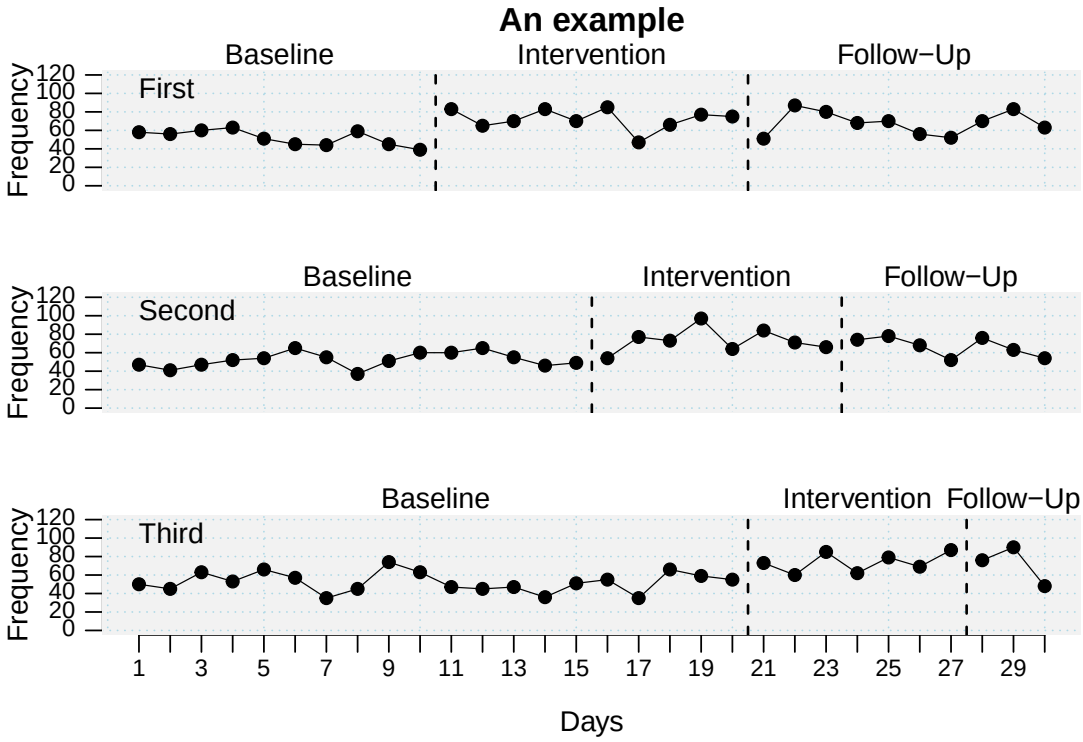


Figure 5.2: A plot with various axis specidications.

Table 5.1: Arguments of the plot function

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

5.2 Adding lines

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(type = '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(type = 'median', col = 'red', lwd = '2')`.

```
plot(
  exampleAB,
  lines = list(
    list(type = "median", col = "red", lwd = 0.5),
    list(type = "trend", col = "blue", lty = "dashed", lwd = 2),
    list(type = "loreg", f = 0.2, col = "green", lty = "solid", lwd = 1)
  )
)
```

5.3 Mark data points

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 =`

Table 5.2: Values of the lines argument

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

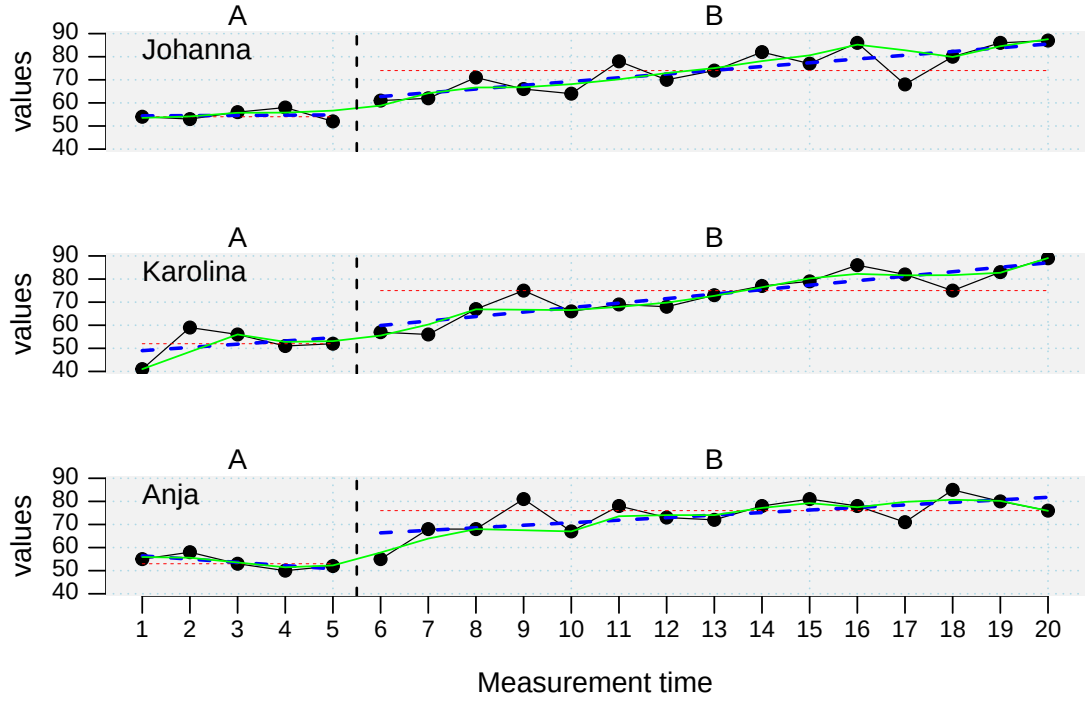


Figure 5.3: A plot with various visual aids

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

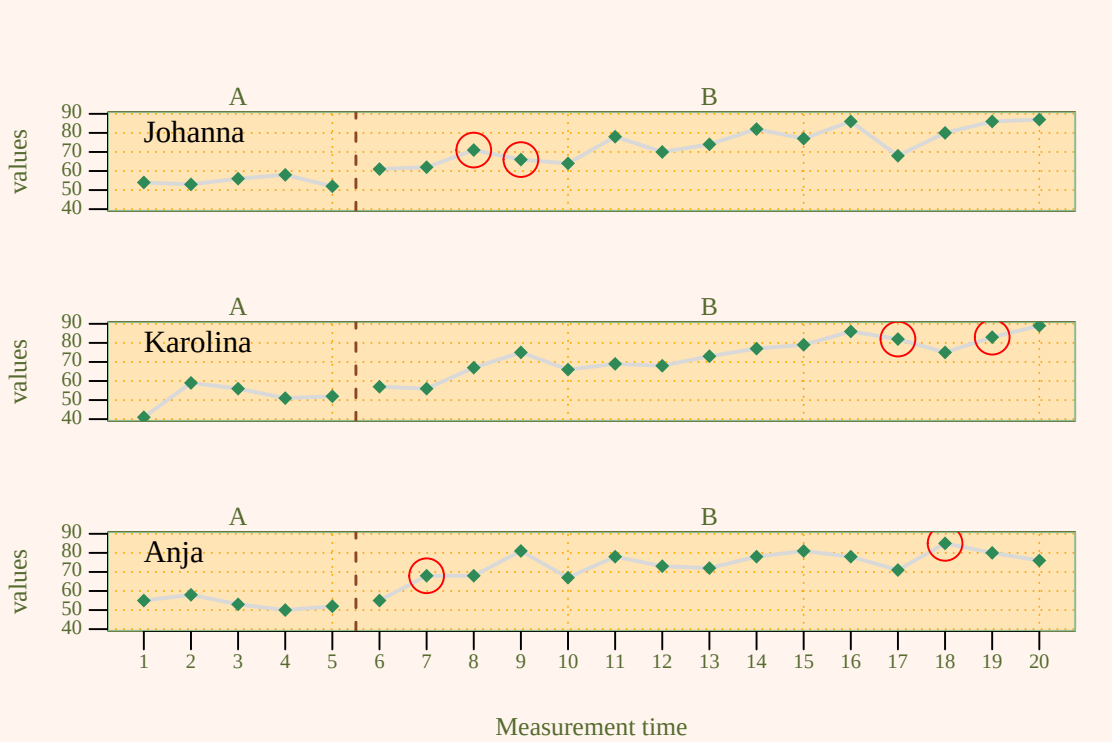


Figure 5.4: A plot with highlighted data-points

5.4 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 styles. This could be achieved by providing several style names to the plot argument: `style = c("grid", "annotate", "small")`. A style overwrites the settings of all previously included style.
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 **??** 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 by 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 = "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.

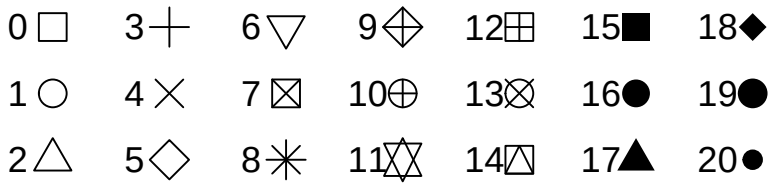


Figure 5.5: (#fig:symbols, pch)Some of the possible symbols and their pch values.

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

Table 5.3: Arguments of the style plot function

Argument	What it does ...
fill	If TRUE area under the line is filled.
col.fill	Sets the color of the area under the line.
grid	If TRUE a grid is included.
col.grid	Sets the color of the grid.
lty.grid	Sets the line type of the grid.
lwd.grid	Sets the line thikness of the grid.
fill.bg	If not NA the backgorund of the plot is filled with the given color. If multiple colours are provided, the colours change with phases (e.g., 'fill.bg = c('aliceblue', 'mistyrose1', 'honeydew')'
annotations	A list of parameters defining annotations to each data point. This adds the score of each MT to your plot. ''pos'' Position of the annotations: 1 = below, 2 = left, 3 = above, 4 = right. ''col'' Color of the annotations. ''cex'' Size of the annotations. ''round'' rounds the values to the specified decimal. 'annotations = list(pos = 3, col = 'brown', round = 1)' adds scores rounded to one decimal above the data point in brown color to the plot.
text.ABlag	By default a vertical line separates phases A and B in the plot. Alternatively, you could print a character string between the two phases using this argument: 'text.ABlag = 'Start''.
lwd	Width of the plot line. Default is 'lwd = 2'.
pch	Point type. Default is 'pch = 17' (triangles). Other options are for example: 16 (filled circles) or 'A' (uses the letter A).
col.lines	The color of the lines. If set to an empty string no lines are drawn.
col.dots	The color of the dots. If set to an empty string no dots are drawn.
mai	Sets the margins of the plot.
...	Further arguments passed to the plot command.

```
newstyle <- style_plot(
  fill = "grey95",
  fill.bg = c('aliceblue', 'mistyrose1', 'honeydew'),
  names = list(col = "brown", cex = 2, font = 3, side = 3),
  annotations = list(col = "brown"),
  col.dots = "blue",
  grid = "lightblue",
  pch = 16)

plot(exampleABAB, style = newstyle)
```

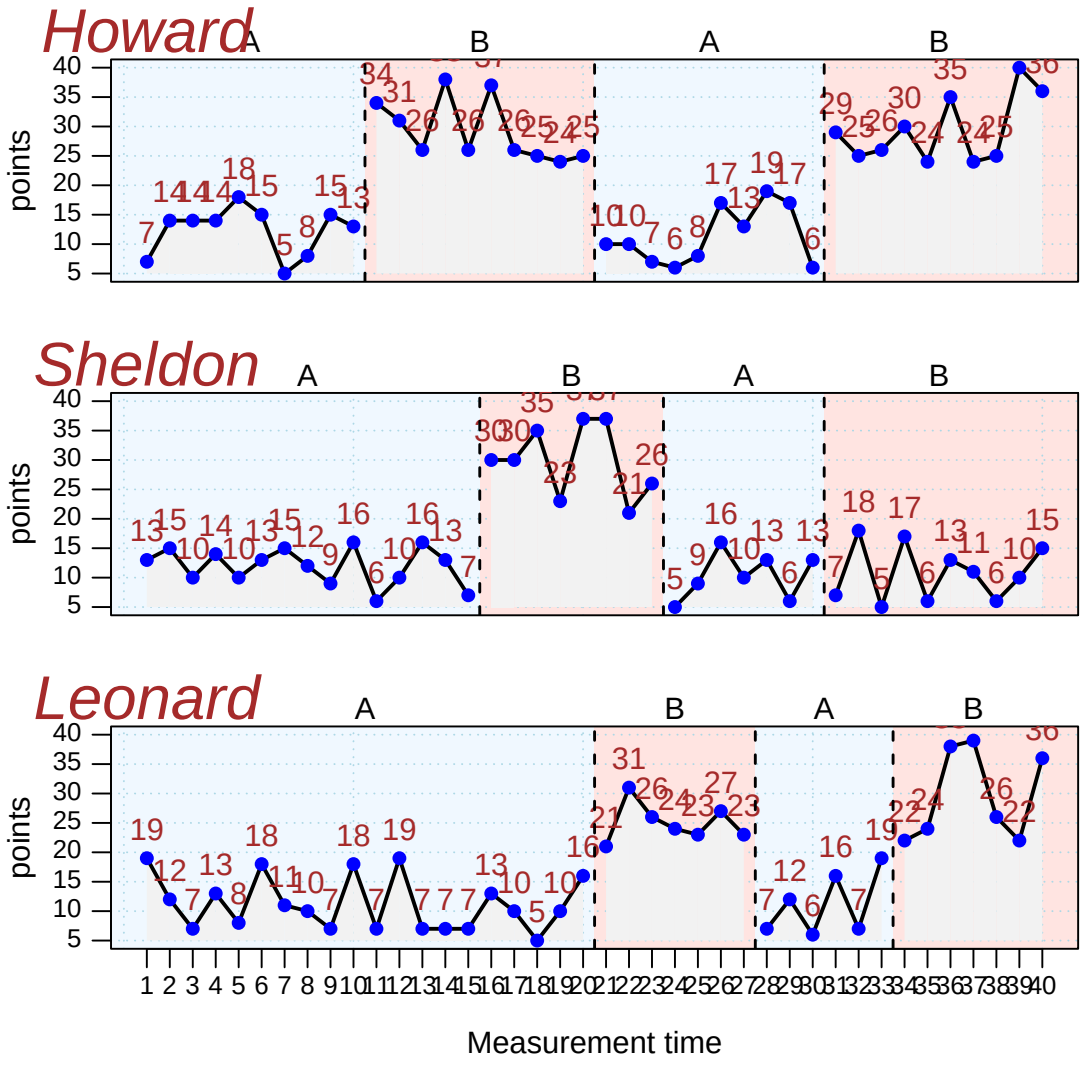


Figure 5.6: (#fig:custom_style_example)A plot with a customized style.

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).

Table 6.1: Statistics of the describe command

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

```
describe(exampleABC)
```

```
Describe Single-Case Data

       Marie Rosalind  Lise
Design A-B-C    A-B-C A-B-C
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      52.000   52.267  52.350
m.B      72.100   73.250  73.571
m.C      68.000   66.429  71.333
md.A       53.5     52.0    52.0
md.B       72.5     72.0    73.0
md.C         69       68      76
sd.A      8.287    8.146  10.869
sd.B     11.367   13.134  10.644
sd.C     12.702   10.486  21.385
mad.A    11.119    7.413  10.378
mad.B    10.378   10.378  16.309
mad.C    17.791   11.861  20.756
min.A        39       37      35
min.B        47       54      60
min.C        51       52      48
max.A        63       65      74
max.B        85       97      87
max.C        87       78      90
trend.A  -1.915    0.500  -0.088
trend.B  -0.612    0.643   1.929
```

```
trend.C  -0.194   -2.929 -14.000
```

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 `describe` 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

Marie
 Phase Lag 1 Lag 2 Lag 3 Lag 4
     A  0.29 -0.11  0.10  0.12
     B -0.28 -0.10 -0.14 -0.09
     C  0.00 -0.33 -0.14 -0.25
   all  0.21  0.10  0.25  0.12

Rosalind
```

```
 Phase Lag 1 Lag 2 Lag 3 Lag 4
     A  0.37 -0.29 -0.33 -0.34
     B -0.34  0.24 -0.40  0.04
     C -0.07 -0.32  0.27  0.02
   all  0.49  0.38  0.22  0.17

Lise
 Phase Lag 1 Lag 2 Lag 3 Lag 4
     A  0.04 -0.32 -0.05 -0.09
     B -0.63  0.50 -0.40  0.31
     C -0.38 -0.12    NA    NA
   all  0.33  0.36  0.23  0.27
```

The `trend` 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 start at 0  for each phase
```

```
# 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 start at 1  for each phase
```

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:

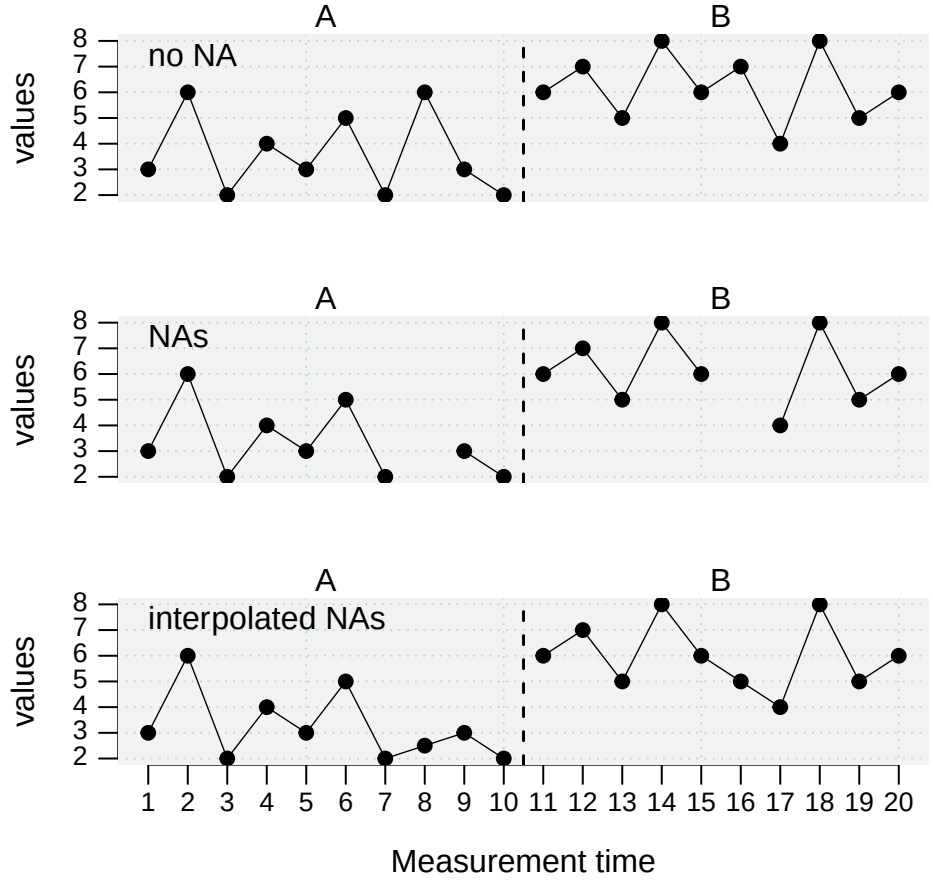
```
scdf(c(5, 3, 4, 6, 8, 7, 9, 7, NA, 6), phase_design = c(A = 4, B = 6))
```

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 <- fill_missing(case2)
names(case3) <- "interpolated NAs"
ex <- c(case1, case2, case3)
plot(ex)
```




```
overlap(ex)
```

```
Overlap Indices

Comparing phase 1 against phase 2

              no NA  NAs interpolated NAs
Design          A-B  A-B               A-B
PND              40   33                30
PEM             100  100               100
PET             100  100               100
NAP              88   91                92
NAP rescaled     77   83                83
PAND             72   81                80
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      1.71 1.90            1.96
```

6.4 Outlieranalysis

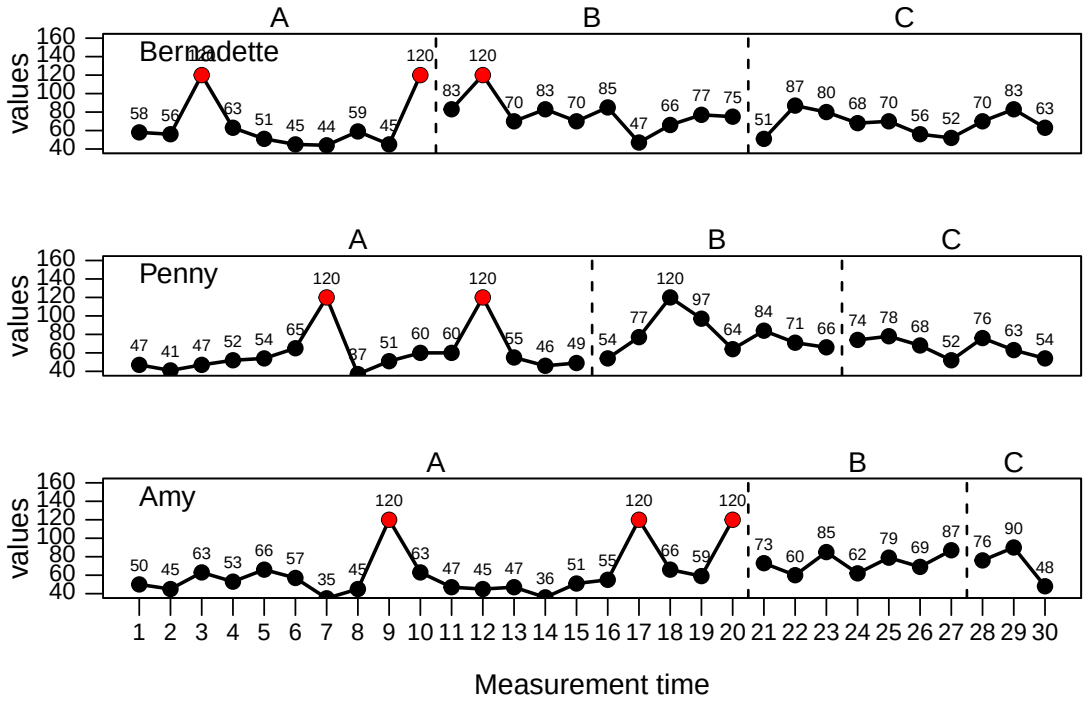
scan provides several methods for analyzing outliers. All of them are implemented in the `outliers` function. Available methods are the **standard deviation**, **mean average deviation**, **confidence intervals**, and **Cook's distance**. The criteria argument takes a vector with two information, the 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 **?** recommend the use the much more robust median average deviation (**MAD**) instead. The **MAD** is is constructed similar to the **sd** but uses the median instead of the mean. Multiplying the **MAD** by 1.4826 approximates the **sd** in a normal distributed sample. This corrected MAD is applied in the `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
```

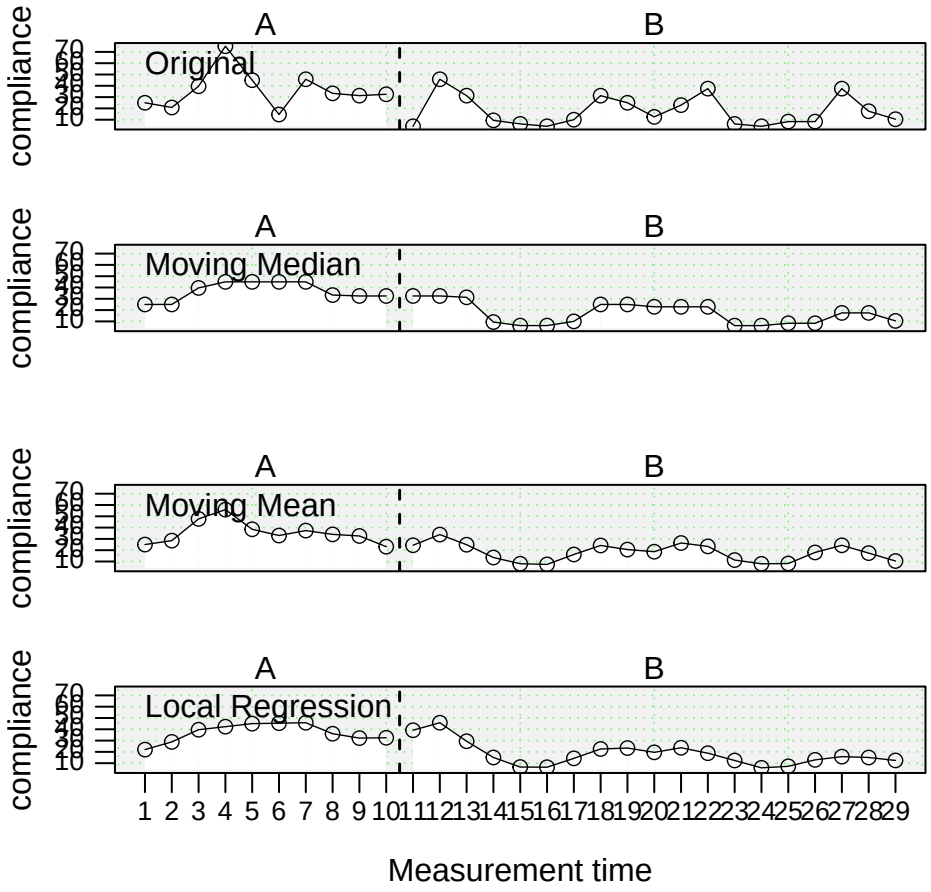
```
# Visualizing outliers with the plot function
res <- outlier(exampleABC_outlier, criteria = c("MAD", 3.5))
plot(exampleABC_outlier, marks = res, style = "annotate", ylim = c(40,160))
```



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 am 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 is 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

Comparing phase 1 against phase 2

             Johanna Karolina  Anja
Design           A-B      A-B   A-B
PND              100       87    93
PEM              100      100   100
PET              100       93   100
NAP              100       97    98
NAP rescaled     100       93    96
PAND             100       90    90
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        2.35     2.26  2.87
```

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 in wanted.
The `phases` argument takes a list with two elements. One element for each of the two phases that should be compared. The elements could contain either the name of the two phases or the

number of the position within the *scdf*. If you want to compare the first to the third phase you can set `phases = list(1,3)`. If the phases of your case are named 'A', 'B', and 'C' you could alternatively set `phases = list("A","C")`.
It is also possible to compare a combination of several cases against a combination of other phases. Each of the two list-elements could contain more than one phase which are concatenated with the `c` command. For example if you have an ABAB-Design and like to compare the two A-phases against the two B-phases `phases = list( c(1,3), c(2,4) )` will do the trick.

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

```
Overlap Indices

Comparing phases A1 + A2 against phases B1 + B2

                   Pawel      Moritz      Jannis
Design       A1-B1-A2-B2 A1-B1-A2-B2 A1-B1-A2-B2
PND                   55          78          71
PEM                  100         100         100
PET                  100         100         100
NAP                   94          97          98
NAP rescaled          89          94          97
PAND                  82          85          90
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            2.07        2.72        2.98
```

7.1 Standardized mean differences

Standardized mean differences can be calculated in various ways. They refer to the difference in the means of two phases. The `smd` function provides an overview of the most common parameters for each single-case:

```
smd(exampleAB_score)
```

```
Standardized mean differences

                           Christiano Lionel Neymar
mA                               2.70   3.10   2.30
mB                              15.35  15.35  15.60
sdA                              1.42   1.59   1.49
sdB                              2.13   1.60   2.19
sd cohen                         1.81   1.60   1.87
sd hedges                        1.93   1.60   1.99
```

```
Glass' delta                      8.92   7.68   8.90
Hedges' g                         6.54   7.67   6.68
Hedges' g correction              6.37   7.46   6.50
Hedges' g durlak correction       6.15   7.21   6.28
Cohen's d                         6.98   7.67   7.10
```

7.2 Percentage non-overlapping data (PND)

The percentage of non-overlapping data (PND) effect size measure was described by **?** . 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.

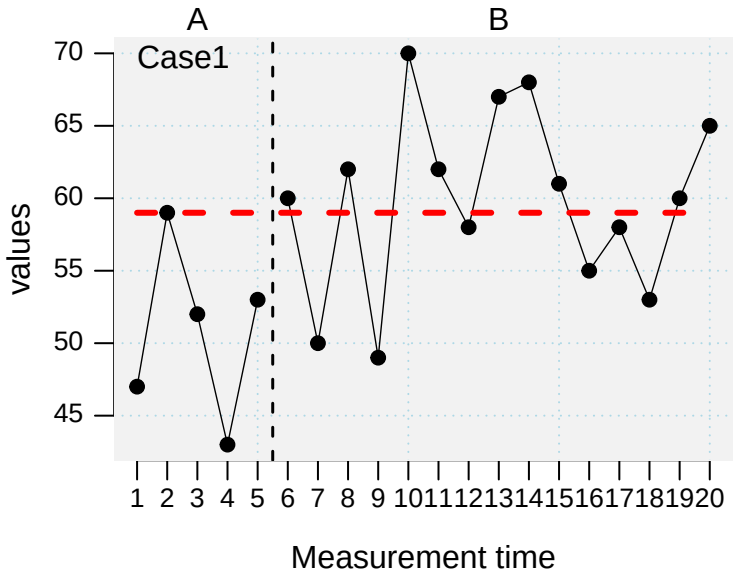


Figure 7.1: Illustration of PND. PND is 60% as 9 out of 15 datapoints of phase B are higher than the maximum of phase A.

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

```
pnd(exampleAB)
```

```
Percent Non-Overlapping Data

     Case    PND Total Exceeds
  Johanna   100%    15      15
```

```
 Karolina 86.67%    15       13
     Anja 93.33%    15       14

Mean   : 93.33 %
```

7.3 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.

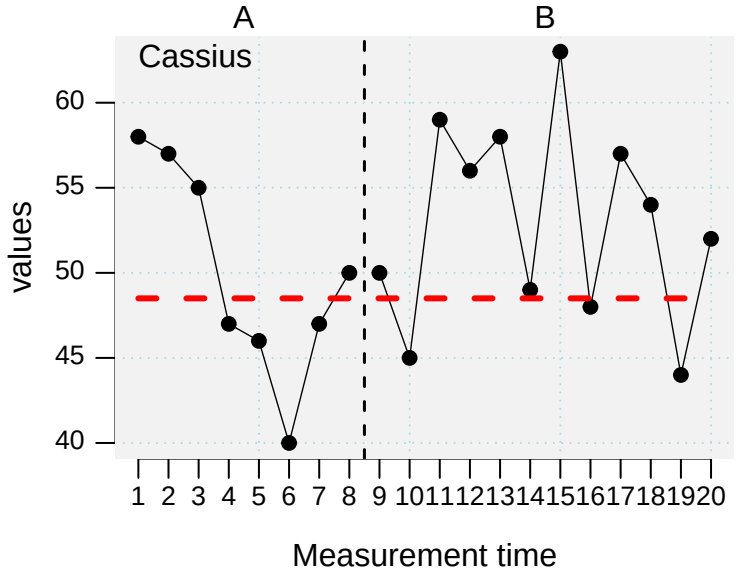


Figure 7.2: Illustration of PEM. PEM is 75% as 9 out of 12 datapoints of phase B are higher than the median of phase A.

```
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%
```

7.4 Percentage exceeding the regression trend (PET)

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

```
pet(exampleAB)
```

```
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 predic
```

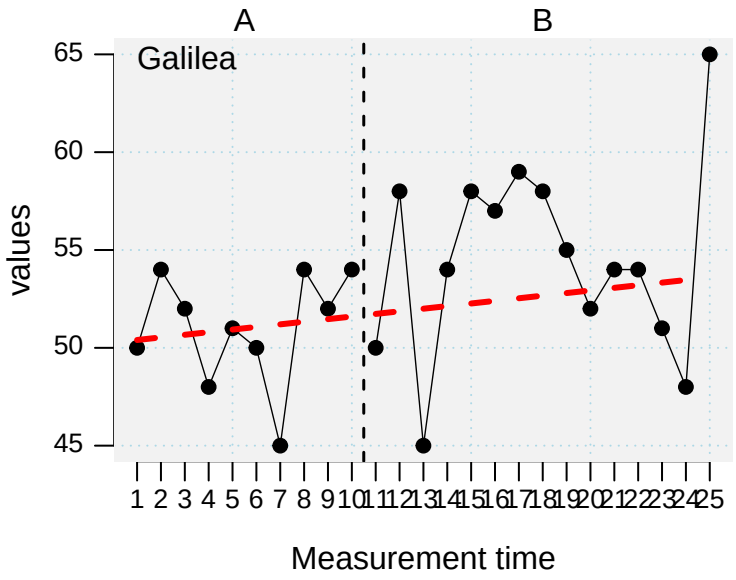


Figure 7.3: Illustration of PET. PET is 66.7% as 10 out of 15 datapoints of phase B are higher than the projected trend-line of phase A.

7.5 Percentage of all non-overlapping data (PAND)

The `pand` function calculates the percentage of all non-overlapping data (**?**), an index to quantify a level increase (or decrease) in performance after the onset of an intervention. The argument

`correction = TRUE` makes `pand` use a frequency matrix, which is corrected for ties. A tie is counted as the half of a measurement in both phases. Set `correction = FALSE` to use the uncorrected matrix, which is not recommended.

```
pand(exampleAB)
```

```
Percentage of all non-overlapping data

PAND =  93.3 %
Φ =  0.822  ; Φ² =  0.676

Number of cases: 3
Total measurements: 60  (in phase A: 15; in phase B: 45)
n overlapping data per case: 0, 2, 2
Total overlapping data: n = 4 ; percentage = 6.7

2 x 2 Matrix of proportions
    % expected
    A   B   total
%    A  21.7    3.3 25
real B  3.3 71.7    75
 total  25  75

2 x 2 Matrix of counts
    expected
    A   B   total
     A  13  2   15
real B  2   43  45
 total  15  45


Note. Matrix is corrected for ties

Correlation based analysis:

z = 6.316, p = 0.000,   = 0.822
```

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

7.6 Nonoverlap of all pairs (NAP)

The `nap` function calculates the nonoverlap of all pairs (**?**). NAP summarizes the overlap between all pairs of phase A and phase B data points. If an increase of phase B scores is expected, a

non-overlapping pair has a higher phase B data point. The NAP equals number of pairs showing no overlap / number of pairs. Because NAP can only take values between 50 and 100 percent, a rescaled and therefore more intuitive NAP (0-100%) is also displayed. NAP is equivalent to the the U-test and Wilcox rank sum test. Thus, a Wilcox test is conducted and reported for each case.

```
nap(exampleAB)
```

```
Nonoverlap of All Pairs

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

7.7 Tau-U

The *Tau-U* statistic has been proposed by **?** and is one of the more broadly used approach for reporting effect sizes of single case data. Unfortunately, various and ambiguous implementations of Tau-U exist (**??**). The `tau_u` function tries to cover several of these implementation. It 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 based on a meta-analysis.

Several arguments an be set to define how Tau-U should be calculated. By setting the argument `method = "parker"`, Tau-U is calculated as described in **?**. 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 original method proposed by **?** data, calculations are based on Kendall's Tau A which does not correct for ties. Alternatively, Kendall's Tau B has a correction for Tau in the presence of ties. The `tau_method`‘ can be set to decide on the tau method to use `"a"` for Kendall's Tau A and `"b"`‘ for Kendall's Tau B.

Here is an example with setting that reconstruct the values from the original example in **?** :

```
tau_u(Parker2011, method = "parker", tau_method = "a", continuity_correction = FALSE, ci = NA)
```

```
Tau-U
Method: parker
Applied Kendall's Tau-a

Case: Case1
                          pairs pos neg ties  S  D   Tau CI lower CI upper
A vs. B                      20  17   1    2 16 20 0.800       NA       NA
Trend A                       6   4   1    1  3  6 0.500       NA       NA
Trend B                      10   8   1    1  7 10 0.700       NA       NA
A vs. B - Trend A            20  18   5    3 13 20 0.650       NA       NA
A vs. B + Trend B            30  25   2    3 23 30 0.767       NA       NA
```

```
A vs. B + Trend B - Trend A    36  26   6    4 20 36 0.556       NA       NA
                           SD_S VAR_S SE_Tau    Z     p
A vs. B                    8.16 66.67  0.408 1.96 0.050
Trend A                    2.94  8.67  0.491 1.02 0.308
Trend B                    4.08 16.67  0.408 1.71 0.086
A vs. B - Trend A          9.59 92.00  0.480 1.36 0.175
A vs. B + Trend B          9.59 92.00  0.320 2.40 0.016
A vs. B + Trend B - Trend A 9.59 92.00  0.266 2.09 0.037
```

A different implementation of the method (provided at http://www.singlecaseresearch.org/calculators/tau-u)) uses Kendall's Tau B:

```
tau_u(exampleAB$Johanna, method = "parker", tau_method = "b", continuity_correction = FALSE)
```

```
Tau-U
Method: parker
Applied Kendall's Tau-b

Case: Johanna
                           pairs pos neg ties   S   D   Tau CI lower CI upper
A vs. B                       75  75   0    0  75  75 1.000    0.401    1.599
Trend A                       10   5   5    0   0  10 0.000      NaN      NaN
Trend B                      105  87  17    1  70 104 0.670    0.291    1.049
A vs. B - Trend A             75  80   5    0  75 127 0.592    0.232    0.951
A vs. B + Trend B            180 162  17    1 145 184 0.786    0.462    1.111
A vs. B + Trend B - Trend A  190 167  22    1 145 189 0.765    0.447    1.084
                            SD_S VAR_S SE_Tau    Z     p
A vs. B                    22.91 525.0  0.306 3.27 0.001
Trend A                     4.08  16.7    NaN 0.00 1.000
Trend B                    20.21 408.3  0.193 3.46 0.001
A vs. B - Trend A          23.26 541.2  0.184 3.22 0.001
A vs. B + Trend B          30.53 932.4  0.166 4.75 0.000
A vs. B + Trend B - Trend A 30.81 949.0  0.163 4.71 0.000
```

A different online calculator created by Rumen Manolov is available at https://manolov.shinyapps.io/Overlap/ it applies an R code developed by Kevin Tarlow for caluclating Tau-U. This setting will replicated results from this approach:

```
tau_u(exampleAB$Johanna, method = "complete", tau_method = "a", continuity_correction = FALSE)
```

```
Tau-U
Method: complete
Applied Kendall's Tau-a

Case: Johanna
                           pairs pos neg ties   S   D   Tau CI lower CI upper
A vs. B                       75  75   0    0  75  75 1.000    0.401     1.60
```

```
Trend A                          10   5   5    0   0  10 0.000      NaN      NaN
Trend B                         105  87  17    1  70 105 0.667    0.289     1.04
A vs. B - Trend A                85  80   5    0  75  85 0.882    0.172     1.59
A vs. B + Trend B               180 162  17    1 145 180 0.806    0.470     1.14
A vs. B + Trend B - Trend A     190 167  22    1 145 190 0.763    0.445     1.08
                              SD_S VAR_S SE_Tau    Z     p
A vs. B                      22.91 525.0  0.306 3.27 0.001
Trend A                       4.08  16.7    NaN 0.00 1.000
Trend B                      20.21 408.3  0.192 3.46 0.001
A vs. B - Trend A            30.82 950.0  0.363 2.43 0.015
A vs. B + Trend B            30.82 950.0  0.171 4.70 0.000
A vs. B + Trend B - Trend A 30.82 950.0  0.162 4.70 0.000
```

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`.

```
tau_u(exampleAB$Johanna) %>% print(complete = TRUE)
```

```
Tau-U
Method: complete
Applied Kendall's Tau-b
95% CIs for tau are reported.

Case: Johanna
                              pairs pos neg ties   S   D   Tau CI lower CI upper
A vs. B                          75  75   0    0  75  75 1.000    0.401    1.599
Trend A                          10   5   5    0   0  10 0.000      NaN      NaN
Trend B                         105  87  17    1  70 104 0.670    0.291    1.049
A vs. B - Trend A                85  80   5    0  75 127 0.592    0.232    0.951
A vs. B + Trend B               180 162  17    1 145 184 0.786    0.462    1.111
A vs. B + Trend B - Trend A     190 167  22    1 145 189 0.765    0.447    1.084
                              SD_S VAR_S SE_Tau    Z     p
A vs. B                      22.91 525.0  0.306 3.27 0.001
Trend A                       4.08  16.7    NaN 0.00 1.000
Trend B                      20.21 408.3  0.193 3.46 0.001
A vs. B - Trend A            23.26 541.2  0.184 3.22 0.001
A vs. B + Trend B            30.53 932.4  0.166 4.75 0.000
A vs. B + Trend B - Trend A 30.81 949.0  0.163 4.71 0.000
```

When you provide multiple single-cases to the `tau-u`‘ function, it will calculate a Tau-U table for each case and an overall calculation. The overall Tau-U value is the average of all Tau-U values weighted by their standard error. You can choose between a random- and a fixed-effect approach for the meta-analyses (`meta_method = "random"` or `"fixed"`).

```
tau_u(exampleAB)
```

```
Tau-U
```

```
Method: complete
Applied Kendall's Tau-b
95% CIs for tau are reported.

Overall Tau-U
Meta-anlysis model: random effect

                      Model Tau_U     se CI lower CI upper    z        p
                  A vs. B 0.969 0.1772    0.622    1.316 5.47 4.54e-08
          A vs. B - Trend A 0.590 0.1064    0.381    0.798 5.54 3.04e-08
          A vs. B + Trend B 0.740 0.0960    0.552    0.928 7.71 1.29e-14
 A vs. B + Trend B - Trend A 0.731 0.0942    0.546    0.915 7.75 9.09e-15

Case: Johanna
                            pairs pos neg ties   S   D   Tau CI lower CI upper
A vs. B                        75  75   0    0  75  75 1.000    0.401    1.599
Trend A                        10   5   5    0   0  10 0.000      NaN      NaN
Trend B                       105  87  17    1  70 104 0.670    0.291    1.049
A vs. B - Trend A              85  80   5    0  75 127 0.592    0.232    0.951
A vs. B + Trend B             180 162  17    1 145 184 0.786    0.462    1.111
A vs. B + Trend B - Trend A   190 167  22    1 145 189 0.765    0.447    1.084
                             SD_S VAR_S SE_Tau    Z     p
A vs. B                     22.91 525.0  0.306 3.27 0.001
Trend A                      4.08  16.7    NaN 0.00 1.000
Trend B                     20.21 408.3  0.193 3.46 0.001
A vs. B - Trend A           23.26 541.2  0.184 3.22 0.001
A vs. B + Trend B           30.53 932.4  0.166 4.75 0.000
A vs. B + Trend B - Trend A 30.81 949.0  0.163 4.71 0.000

Case: Karolina
                            pairs pos neg ties   S     D   Tau CI lower
A vs. B                        75  72   2    1  70  74.5 0.940    0.337
Trend A                        10   5   5    0   0  10.0 0.000      NaN
Trend B                       105  91  13    1  78 104.5 0.746    0.367
A vs. B - Trend A              85  77   7    1  70 126.4 0.554    0.193
A vs. B + Trend B             180 163  15    2 148 184.0 0.805    0.479
A vs. B + Trend B - Trend A   190 168  20    2 148 189.0 0.783    0.464
                            CI upper  SD_S VAR_S SE_Tau    Z     p
A vs. B                        1.542 22.91 525.0  0.308 3.06 0.002
Trend A                          NaN  4.08  16.7    NaN 0.00 1.000
Trend B                        1.125 20.21 408.3  0.193 3.86 0.000
A vs. B - Trend A              0.914 23.25 540.8  0.184 3.01 0.003
A vs. B + Trend B              1.130 30.52 931.4  0.166 4.85 0.000
A vs. B + Trend B - Trend A    1.102 30.79 948.0  0.163 4.81 0.000

Case: Anja
                            pairs pos neg ties   S     D    Tau CI lower
A vs. B                        75  73   1    1  72  74.5  0.966   0.3636
Trend A                        10   2   8    0  -6  10.0 -0.600  -1.4002
```

```
Trend B                          105  71  29    5  42 102.5  0.410   0.0234
A vs. B - Trend A                 85  81   3    1  78 125.1  0.624   0.2600
A vs. B + Trend B                180 144  30    6 114 182.0  0.626   0.2985
A vs. B + Trend B - Trend A      190 152  32    6 120 187.0  0.642   0.3198
                               CI upper  SD_S VAR_S SE_Tau     Z     p
A vs. B                         1.569 22.91 525.0  0.308  3.14 0.002
Trend A                         0.200  4.08  16.7  0.408 -1.47 0.142
Trend B                         0.796 20.21 408.3  0.197  2.08 0.038
A vs. B - Trend A               0.987 23.21 538.6  0.186  3.36 0.001
A vs. B + Trend B               0.954 30.45 927.0  0.167  3.74 0.000
A vs. B + Trend B - Trend A     0.964 30.71 943.3  0.164  3.91 0.000
```

7.8 Baseline corrected tau

This method has been proposed by **?**. The baseline data are checked for a significant autocorrelation (based on Kendalls Tau). If so, a non-parameteric Theil-Sen regression is applied for the baseline data where the dependent values are regressed on the measurement time. The resulting slope information is then used to predict data of the B-phase. The dependent variable is now corrected for this baseline trend and the residuals of the Theil-Sen regression are taken for further calculations. Finally, Kendalls tau is calculated for the dependent variable and the dichotomous phase variable. The function here provides two extensions to this procedure: The more accurate Siegel repeated median regression is applied when `repeated = TRUE` (**?**) and a continuity correction is applied when `continuity = TRUE` (both are the default settings).

```
dat <- scdf(c(A = 33,25,17,25,14,13,15, B = 15,16,16,5,7,9,6,5,3,3,8,11,7))
corrected_tau(dat)
```

```
Baseline corrected tau

Method: Theil-Sen regression
Continuity correction not applied.

                           tau     z     p
Baseline autocorrelation -0.68 -2.13  <.05
Uncorrected tau          -0.57 -2.94  <.01
Baseline corrected tau    0.70  3.61 <.001

Baseline correction should be applied.
```

Here is a replication of an example provided by **?** :

```
dat <- scdf(c(A = 33, 25, 17, 25, 14, 13,14, B = 14, 15, 15, 4, 6, 9, 5 ,4 ,2 ,2 ,8, 11 ,7))
corrected_tau(dat, repeated = FALSE)
```

```
Baseline corrected tau
```

```
Method: Theil-Sen regression
Continuity correction not applied.

                          tau     z     p
Baseline autocorrelation -0.75 -2.31  <.05
Uncorrected tau          -0.58 -2.98  <.01
Baseline corrected tau    0.69  3.57 <.001

Baseline correction should be applied.
```

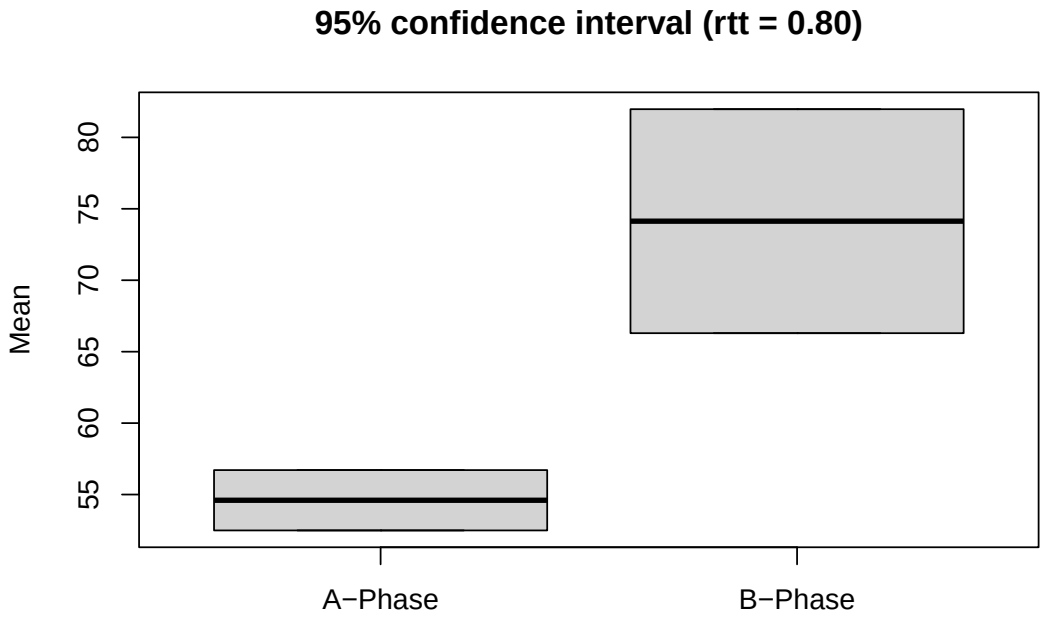
```
model <- plm(dat)
```

7.9 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 (**?**) and corresponding descriptive statistics.

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



```
Reliable Change Index

Mean Difference =  19.53333
Standardized Difference =  1.678301

Descriptives:
         n     mean       SD       SE
A-Phase  5 54.60000 2.408319 1.077033
B-Phase 15 74.13333 8.943207 3.999524

Reliability =  0.8

95 % Confidence Intervals:
           Lower    Upper
A-Phase 52.48905 56.71095
B-Phase 66.29441 81.97226

Reliable Change Indices:
                              RCI
Jacobson et al.          18.13624
Christensen and Mendoza 12.82426
Hageman and Arrindell    18.49426
```

Chapter 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 **?**. 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).

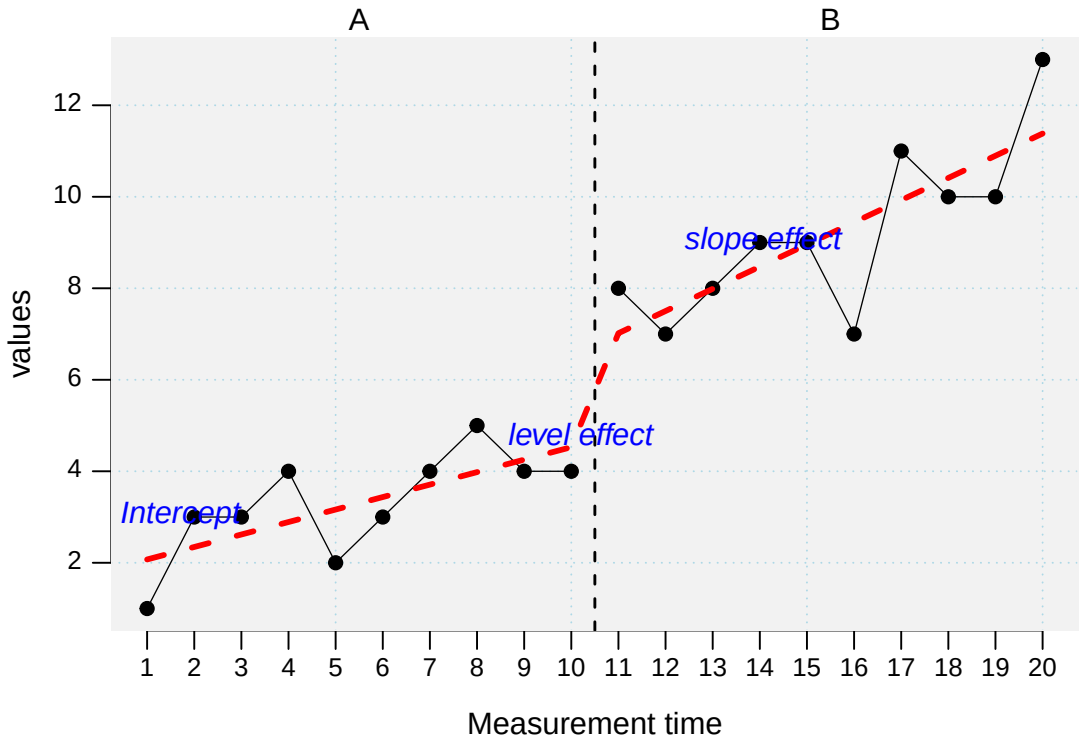
In this model, four parameters predict the outcome at a specific measurement point:

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; R² = 0.843; Adjusted R² = 0.814

                   B   2.5%  97.5%    SE      t     p delta R²
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 *B&L-B* model starts with a one at the first measurement of the phase and increases with every measurement while the *H-M* model starts with a zero.

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

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

phase	values	level_B	level_C	slope_B	slope_C
A	3	0	0	0	0
A	6	0	0	0	0
A	4	0	0	0	0
A	7	0	0	0	0
B	5	1	0	1	0
B	3	1	0	2	0
B	4	1	0	3	0
B	6	1	0	4	0
B	3	1	0	5	0
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; R² = 0.800; Adjusted R² = 0.763

                   B   2.5%  97.5%    SE      t     p delta R²
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: values ~ 1 + mt + phaseB + interB
```

The piecewise regression reveals a significant level effect and two non significant effects for trend and slope. In a further analyses we would like to put the slope effect out of the equation. There are several ways to do this. The easiest way is the to set the `slope` argument to `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; R² = 0.775; Adjusted R² = 0.749

                   B   2.5%  97.5%    SE      t     p delta R²
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: values ~ 1 + mt + 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 10.33 points. That is,

with the beginning of the intervention (the B-phase) the score increases by 15.38 points (5 x 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; R² = 0.402; Adjusted R² = 0.333

                            B   2.5%  97.5%    SE      t     p delta R²
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

tobewritten