# FAKIN Documentation Index

*Hauke Sonnenberg*

*2018-06-04*

# Contents

# Part I

# FAKIN-Dokumentation

# Chapter 1

# Einführung

Hier wollen wir die Ergebnisse unseres Forschungsprojekts FAKIN (Forschungsdatenmanagement an kleinen Instituten) veröffentlichen.

# Chapter 2

# Fallbeispiele

Da wir viele, zum Teil ganz unterschiedliche Projekte bearbeiten, sind auch unterschiedliche Bereiche des Forschungsdatenmanagements betroffen.

Wir wollen hier zunächst versuchen, Fallbeispiele herauszuarbeiten, in die sich die verschiedenen Projekte einordnen lassen. Für jedes Fallbeispiel wollen wir dann Gemeinsamkeiten in Bezug auf Anforderungen und Problemstellungen herausarbeiten und für diese dann Lösungen entwickeln. Diese Lösungen sollen dann in zukünftigen Projekten, die sich den jeweiligen Fallbeispielen zuordnen lassen, angewendet werden.

# Chapter 3

# Projektbausteine

Wir können “Projektbausteine” definieren, die in verschiedenen Projekten vorkommen. Diese werden im Folgenden einzeln betrachtet.

## 3.1 Monitoring

In vielen Projekten führen wir eigene Messungen durch. Dazu verwenden wir eigene Messgeräte.

Problemstellungen:

- verschiedene Geräte verschiedener Hersteller -> verschiedene, meist nicht standardisierte Dateiformate
- Zeitstempel der Messung
  - Stellt das Gerät auf Sommerzeit um? -> Problembereich Zeitstempel
  - Geht die interne Uhr immer richtig? Wie gewährleiste ich das?

Beispielprojekte:

- MIACSO
- KURAS
- Flusshygiene

## 3.2 Kanalnetzmodellierung

Beispielprojekte:

- MIACSO
- KURAS
- Flusshygiene

## 3.3 Datenimport

## 3.4 Fallbeispiel: LCA (Life Cycle Assessment = Lebenszyklusbewertung)

# Chapter 4

# Problembereiche

## 4.1 Zeitstempel

**Frage:**

Wie stellen wir die Messgeräte ein? Sollen sie die Uhr automatisch umstellen oder nicht?

**Diskussion:**

Aus Sicht der Datenverarbeitung bereitet uns die Umstellung von Winterzeit (= Normalzeit) auf Sommerzeit und von Sommerzeit auf Winterzeit immer wieder Schwierigkeiten. Warum?

**Umstellung von Winterzeit auf Sommerzeit**

Die Uhr wird um eine Stunde von 02:00 CET (*Central European Time*) auf 03:00 CEST (*Central European Summer Time*) **vorgestellt**. Merke: Die Stühle eines Cafés werden **vor** den Laden gestellt.

In einer Zeitreihe ergibt sich daraus am Tag der Zeitumstellung (z.B. am 25.03.2018) eine Lücke in den Zeitstempeln:

```
2018-03-25 01:45:00 CET
2018-03-25 01:50:00 CET
2018-03-25 01:55:00 CET
2018-03-25 03:00:00 CEST
2018-03-25 03:05:00 CEST
2018-03-25 03:10:00 CEST
```

Wenn die als Text vorliegenden Zeitstempel in einen numerischen Zeitstempel umgewandelt werden, gibt es, auch wenn die Angabe `CET` oder `CEST` fehlt, im Grunde kein Problem, da sie eindeutig sind. Voraussetzung ist, dass bei der Umwandlung angegeben wird, dass diese Zeitstempel in der Zeitzone “Europe/Berlin” (in der es Sommer- und Winterzeit gibt) aufgenommen wurden.

In der Programmiersprache R ist das auf unseren Rechnern das Standardverhalten, da die Zeitzone vom Betriebssystem abgefragt wird:

```r
as.POSIXct("2018-03-25 01:55:00")
## [1] "2018-03-25 01:55:00 CET"
as.POSIXct("2018-03-25 03:00:00")
## [1] "2018-03-25 03:00:00 CEST"
```

Interessant an dieser Stelle ist das Verhalten, wenn ein Zeitstempel, der in der Zeitzone “Europe/Berlin” nicht existiert, angegeben wird:

```
as.POSIXct("2018-03-25 02:30:00")
## [1] "2018-03-25 01:30:00 CET"
```

Leider tritt an dieser Stelle kein Fehler auf!

**Umstellung von Sommerzeit auf Winterzeit**

Die Uhr wird um 03:00 CEST wieder um eine Stunde auf die Normalzeit 02:00 CET **zurückgestellt**. Merke: Die Stühle werden wieder in den Laden **zurück** gestellt.

In einer Zeitreihe ergeben sich daraus am Tag der Zeitumstellung (z.B. am 28.10.2018) Duplikate in den Zeitstempeln:

```
2018-10-28 02:15:00 CEST
2018-10-28 02:30:00 CEST
2018-10-28 02:45:00 CEST
2018-10-28 02:00:00 CET
2018-10-28 02:15:00 CET
2018-10-28 02:30:00 CET
```

Ohne die Information `CEST` bzw. `CEST` sind die Zeitstempel (für sich genommen) nicht eindeutig. In Dateien, die wir von den BWB bekommen (z.B. Regendaten) würden die Zeitstempel zum Beispiel in dieser Form vorkommen:

```
28.10.2018 02:15:00
28.10.2018 02:30:00
28.10.2018 02:45:00
28.10.2018 02:00:00
28.10.2018 02:15:00
28.10.2018 02:30:00
```

Erst aus der Reihenfolge (erstes Auftreten ist CEST, zweites Auftreten ist CET) lässt sich die Zuordnung rekonstruieren. Dies müssen wir beachten, wenn wir die Daten bearbeiten und ich denke, dass wir dies in den seltensten Fällen tun!

Wir sollten eine einheitliche, eindeutige Vorgehensweise entwickeln.

**TODO:** Ggf. weitere Beschreibungen in R-Workshop, den ich mal gegeben habe…

**Vorschlag:**

- Nach Möglichkeit die Messgeräte so einstellen, dass der ausgegebene Zeitstempel den UTC-Offset enthält. Am besten wäre ein Zeitstempel in einem Format, das die Differenz gegenüber *Greenich Mean Time (GMT)* bzw. *Coordinated Universal Time (UTC)* enthält, wie wir es ggf. aus unseren E-Mail-Programmen oder Terminkalendern kennen: `2018-06-01 23:18 GMT+02:00`.
- Manchmal gibt es auch die Möglichkeit, dass zusätzlich zum lokalen Zeitstempel der UTC-Offset (+01 im Winter und +02 im Sommer) als extra Spalte ausgegeben wird. Von dieser Möglichkeit sollte dann Gebrauch gemacht werden.
- Ansonsten die Geräte so einstellen, dass sie die Zeitumstellung nicht mitmachen (also immer die Normalzeit, also Winterzeit aufzeichnen). Dann sind die Zeitstempel eindeutig. Aufzupassen ist dann allerdings bei der Verarbeitung und Interpretation der Daten, dazu mehr unter Datenimport

**Werkzeuge:**

KWB-eigenes R-Paket `kwb.datetime`. Dieses enthält unter anderem die Funktionen

- `date_range_CEST()` zur Ermittlung der Tage, an denen die Zeitumstellung stattfindet.
- **TODO:** Wichtigste Funktionen (z.B. zur Ermittlung der vollständigen Zeitstempel inklusive UTC-Offset aus fortlaufenden lokalen Zeitstempeln) dokumentieren und hier referenzieren. Oder welche Pakete sollten wir benutzen? Was bietet z.B. `lubridate`? Es scheint mir allerdings, dass unsere Problematik im WWW etwas unterbeleuchtet ist.

# Part II

# Anhang: Extended R Workshop

- Part I: Motivation or “Why should I use R”?
- Part II: Writing functions, documentation, packages
- Part III: Database Access conventional and with package “kwb.base”

Everything behind the # character is treated as a comment and not considered as R code.

```r
# Set working directory
wdir <- "~/R-Development_SVN/RTraining/2012_01_17_Workshop/"
```

# Chapter 5

# Motivation or “Why should I use R”?

1. R is able to handle large data sets.
2. R uses vector calculation instead of
   - copying a formula (Excel)
   - loops (other programming languages) –> clear code, less error-prone
3. Easy sub-setting, filtering of data sets
4. Easy, nice plotting
5. Statistics (should be number one of this ranking but I did not yet use statistics much.)

## 5.1 Large Data Sets

```r
# Create a large matrix with 10 rows and 300 columns
mtx <- matrix(rnorm(3000, 50), ncol = 300)

# Create a data frame
dfr <- data.frame(x = 1:10, y = mtx)

# Define the path to a CSV file
file <- file.path(wdir, "fields_300.csv")

# Write the data to the file
write.table(dfr, file, sep = ";", row.names = FALSE)

# Read back the first lines of the file (shortened to 60 characters)
substr(readLines(file, 5), 1, 60)
```

```
## [1] "\"x\";\"y.1\";\"y.2\";\"y.3\";\"y.4\";\"y.5\";\"y.6\";\"y.7\";\"y.8\";\"y.9\";\"y"
## [2] "1;51.0449084639268;50.9032206697848;48.5240705459872;48.7744"
## [3] "2;50.4556741271269;49.4940970902596;50.3763482539745;50.9742"
## [4] "3;46.7974600720794;50.3186088824857;50.165919647488;51.66524"
## [5] "4;50.7393312360715;49.3030574790128;50.7176778343797;50.9580"
```

## 5.2 Vector calculation

This is a real strength of R!

```
# Simple: create a sequence of numbers 1:20
1:20
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

Is there an Excel equivalent? …

Ok, "easy" also in Excel.

What about the following?

```
x <- 1:100000

head(x)
```

```
## [1] 1 2 3 4 5 6
```

```
tail(x)
```

```
## [1]  99995  99996  99997  99998  99999 100000
```

Is there an Excel equivalent? … ???

1. Slower (is there a fast method to do?)
2. limit at 65536

But look, Excel (version 2010) shows the number of values, mean and sum!

In R, we can do the same, using the functions `length()`, `mean()`, and `sum()`, respectively:

```
rng <- 1:65536

length(rng)
```

```
## [1] 65536
```

```
mean(rng)
```

```
## [1] 32768.5
```

```
sum(rng) # [1] NA
```

```
## Warning in sum(rng): Integer-Überlauf - nutze sum(as.numeric(.))
```

```
## [1] NA
```

What's up, R, are you losing?

But look, R speaks to you (see the warning above). Does Excel speak to you? R tells us what to do. So let's do, what R proposes:

```
sum(as.numeric(rng)) # [1] 2147516416
```

```
## [1] 2147516416
```

Aha, like that it works!

Let's say it's a draw - as you want…

What about the sum of elements 1, 3, 5, …, 65536?

```
sum(seq(1, 65535, by = 2)) # 1073741824
```

```
## [1] 1073741824
```

The same in Excel?

Not so easy, but possible.

Again, what happens if by mistake I delete/change numbers in between?

Multiply the whole column with 1000, e.g. for unit conversion

```
rng1000 <- rng * 1000
```

What about multiplying only every second number with 1000?

```
rng1000.2 <- rng
indices <- seq(1, 65535, by = 2)
rng1000.2[indices] <- rng[indices] * 1000
```

This is even possible in Excel…

But again: every cell could contain a different formula…

## 5.3 Easy sub-setting, filtering of data sets

## 5.4 Statistics

## 5.5 Timestamps

```
# As easy it is possible to create vectors of timestamps, e.g.
t1 <- as.POSIXct("2010-01-01", tz = "UTC")
t2 <- as.POSIXct("2011-01-01", tz = "UTC")
tstamps <- seq(t1, t2, by = "60 min")

# How many timestamps did we get?
length(tstamps) # 8761
```

```
## [1] 8761
```

```
# What number did we expect?
365 * 24 # 105120
```

```
## [1] 8760
```

Who wants to explain the difference of one?

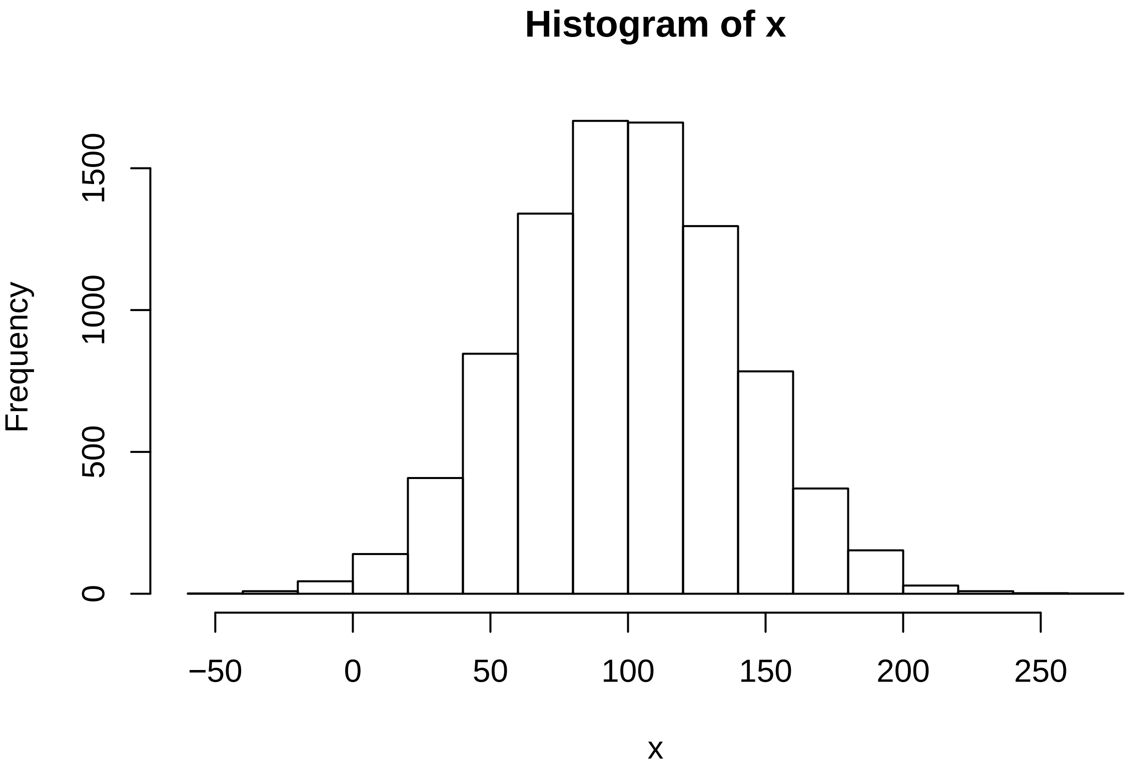
Let's write the timestamps to a database table, together with random numbers, normally distributed around 100 with a standard deviation of 40

```
x <- rnorm(n = length(tstamps), mean = 100, sd = 40)
```

## 5.6 Easy, nice plotting

Let's have a look at the distribution

```
hist(x)
```


Histogram of x
Frequency
0
500
1000
1500
−50
0
50
100
150
200
250
x

# Chapter 6

# Writing functions, documentation, packages

Example from MIA-CSO project: How big are InfoWorks simulation result files to be expected?

E.g. simulation of one year in 5 minutes result timestep with a timestamp column of format "dd/mm/yyyy HH:MM:SS" and 20 data columns of 10 characters size each?

```
#bytesRow <- (colWidth + 1) * colNum + nchar(tsFormat) + 21 + 2
#nRows <- nDays * (24 * 60 * 60)/timestep + 1
#    nRows * bytesRow + bytesHeader

bytesRow <- (10 + 1) * 20 + nchar("dd/mm/yyyy HH:MM:SS") + 21 + 2
nRows <- 365 * (24 * 60 * 60)/300 + 1
nRows * bytesRow + bytesRow # assuming bytesRow for header row

nRows <- 365 * 24 * 12
bytesFile <- nRows * bytesRow
bytesFile

# MBytes?
bytesFile / (1024*1024) #  23.15781
```

Do you want to check this with the windows calculator…?

```
system("calc")
```

Same result. So, what is the advantage of using R?

- We are able to store the results in variables with the arrow operator "<-"

```
spd <- 60 * 24 * 24 # seconds per day
```

# Chapter 7

# Database Access conventional and with package “kwb.base”

## 7.1 Database Access with package RODBC

```r
# and now put it to the table
mdb <- file.path(wdir, "RTraining.mdb")
mydata <- data.frame(tstamps, x)
```

## 7.2 Using RODBC package (only on Windows!)

```r
# What functions does the package offer?
library(RODBC)
help(package = RODBC)

channel <- odbcConnectAccess(mdb)

tbl1 <- "tbl_TS_Rnd_2010_1h_RODBC"
sqlSave(channel, mydata, tablename = tbl1)

# Error:
# Fehler in sqlSave(channel, mydata, tablename = "tblRnd2010_5min_RODBC") :
#   [RODBC] Ausf?hren in Aktualisierung fehlgeschlagen
# 22018 39 [Microsoft][ODBC Microsoft Access Driver]Ung?ltiger Zeichenwert f?r  Konvertierungsangabe. (
#
# The table has been created but it is empty.
#
# What is the problem?
# Maybe we find out by using the "test" argument (see ?sqlSave for explanation)
sqlSave(channel, mydata, tablename = tbl1, test = TRUE)

# Binding: 'rownames' DataType 12, ColSize 255
# Binding: 'tstamps' DataType 8, ColSize 53
# Binding: 'x' DataType 8, ColSize 53
# Without knowing what DataType 8 means, it cannot be true that timestamp and
# random value will be stored as the same data types in the database.

# To make this run we have to tell the sqlSave function what MS Access
# data types to use.
datatypes <- c("DATETIME", "DOUBLE")
names(datatypes) <- names(mydata)
# we have to set "rownames" to true in order to prevent sqlSave from sending
# the row numbers as an extra column
sqlSave(channel, mydata, tablename = tbl1, varTypes = datatypes, rownames = FALSE)

# Do not forget to close...
odbcClose(channel)
```

## 7.3 Database Access with package “kwb.base”

```r
# 1. Install the package kwb.base from
# \\moby\miacso$\REvaluation\RPackages\kwb.base_0.1.1.zip
# by selecting "Pakete -> Installiere Paket(e) aus lokalen zip-Dateien..."
# from the R Console window:
#
# Paket 'kwb.base' erfolgreich ausgepackt und MD5 Summen abgeglichen

# Load the package
library(kwb.base)

# What does the package offer?
help(package = "kwb.base")

# Let's test the hsPutTable function...
# What does the function need?
?hsPutTable

# Use the function
tbl2 <- "tbl_TS_Rnd_2010_1h_hsPut"
tbl2 <- hsPutTable(mdb, mydata, tbl2)

# Is there a difference?
# Tables seem to be identical.

# I have defined a function to set the primary key, we should
# do that:
??"primary key"

# What does the function need?
?hsSetPrimaryKey

# Let's try it out
hsSetPrimaryKey(mdb, tbl2, "tstamps")

# Did this work? Check in mdb... Yes, it worked!
# Let's try to get the data back. First using the RODBC functions
# Again open...
channel <- odbcConnectAccess(mdb)

# ... get data ...
myDataBack <- sqlFetch(channel, tbl1)

# Let's look at the first rows:
head(myDataBack)
#       tstamps         x
#  1 2010-01-01  72.19366
#  2 2010-01-01  89.55628
#  3 2010-01-01 132.70388
#  4 2010-01-01 136.82242
#  5 2010-01-01  99.67992
#  6 2010-01-01  73.66911

# What happened to the timestamp? The time information got lost!
# Is it a problem of the size?
myDataBack2 <- sqlFetch(channel, tbl1, max = 12*24)
tail(myDataBack2)

# Aha! the timestamp is back again!

# We find that we can get data of 86 days but not of 87:
tail(myDataBack2 <- sqlFetch(channel, tbl1, max = 24*86))
tail(myDataBack2 <- sqlFetch(channel, tbl1, max = 24*87))

# Any idea where this strange behaviour comes from?
# Hint: 28 March 2010 is the last Sunday in March its the
# day of switching from normal to summer time:
tail(myDataBack2 <- sqlFetch(channel, tbl1, max = 24*86 + 2))
tail(myDataBack2 <- sqlFetch(channel, tbl1, max = 24*86 + 3))

# Solution: R tries to convert the timestamp that it gets from MS Access in
# primarily in text format to a date time object. R asks the Windows system
# for the time zone being currently active on the computer. As daylight saving
# time is active R is clever and does not accept timestamps that do not exist
# in summer time as it is the case for the times between 2:00 (inclusive)
# and 3:00 (exclusive) of March 28, 2010.

# Knowing that we can apply a trick:
tz <- Sys.getenv("tz") # Get current time zone from system environment
Sys.setenv(tz = "UTC") # Set time zone to UTC
tail(myDataBack2 <- sqlFetch(channel, tbl1, max = 24*86 + 3))
# Voila! the timestamp is back!
Sys.setenv(tz = tz)    # Set time zone back

# Let's close the database
odbcClose(channel)

# The dealing with theses specialties I have hidden in the function
# hsMdbTimeSeries
```

```
?hsMdbTimeSeries
head(myDataBack3 <- hsMdbTimeSeries(mdb, tbl1, "tstamps"))

# The function returns POSIXct timestamps in "UTC" timezone:
time.ct <- myDataBack3$tstamps
attributes(time.ct)

# POSIXt is a "parent" class of the two classes "POSIXlt" and "POSIXct"
# They differ in the form the time information is internally stored.
# While POSIXlt stores it as number of seconds, POSIXct stores it as a
# vector of numbers specifying year, month, day, etc.
# Objects of both classes can be easily converted from one to another:
time.lt <- as.POSIXlt(time.ct)
attributes(time.lt)

# And here is the difference:
as.integer(time.ct[1]) # 1262304000 = number of seconds since...
as.integer(time.lt[1]) # 0   0   0   1   0 110   5   0   0
```

## 7.4 Example: Checking and repairing a time series

```
# I have prepared a time series with data from some timestamps
# missing: I forgot the name, let's look it up:
hsTables(mdb)

# Ok, it's "tbl_Ex_Hyd_TS_lacking", let's get it
# For hsMdbTimeSeries we need to know the name of the
# timestamp field
hsFields(mdb, "tbl_Ex_Hyd_TS_lacking") # [1] "Zeitst" "Q"      "v"

# get the time series
tsh.lack <- hsMdbTimeSeries(mdb, "tbl_Ex_Hyd_TS_lacking", "Zeitst")
tsh.lack

# Recording timestep seems to be one minute. Some timestamps
# are missing. How can I find out where they are?
# Volunteers to find them with Excel?
# My idea? Make events if the time step difference is above 1 min.
# If we have a complete time series there should only be exactly
# one event from the very first to the very last timestamp of the
# table.
# The kwb.base package contains a function hsEvents.
# How does it work?
?hsEvents
hsEvents(tsh.lack$Zeitst, evtSepTime = 60, signalWidth = 60)

# We get a list of three events. Let's check
# The "time gaps" are the time intervals between the events.
# So, there are two gaps
# Can we plot this?
# Let's prepare a 2x1 plot
```

```
par(mfrow = c(2,1))
plot(tsh.lack$Zeitst, tsh.lack$Q, pch = 16)

# Yes, we can see the gaps.
# I prepared a function to fill these timestamp gaps but I
# forgot the name...
??fill  # ah, it is hsFillUp
?hsFillUp
tsh.rep <- hsFillUp(tsh.lack, "Zeitst", step_s = 60)
tsh.rep

# We get 21 rows between 2:30 and 2:50, seems to be ok
# What happened?
# For the missing timestamps the function interpolated the
# values for Q and v. The original columns are shown in order
# to see where the gaps have been.
# Let's have a look to the plot:
lines(tsh.rep$Zeitst, tsh.rep$Q, type = "b", col = "red")

# Ok, we cut the maximum but what else can we do...

# Now let's assume that we have water quality data in another
# table, I have prepared something in the example database:
hsTables(mdb)
hsFields(mdb, "tbl_Ex_Qua_TS_lacking")
tsq.lack <- hsMdbTimeSeries(mdb, "tbl_Ex_Qua_TS_lacking", "myDateTime")
tsq.lack

# Again we can fill-up first
tsq.rep <- hsFillUp(tsq.lack) # As we see there are default values...
tsq.rep

# Three rows have been added for 2:32 to 2:34.
# We see that not only the time gaps have been filled but also
# the missing CSB value of 2:40 has been calculated by interpolation.
# Let's check the graphs:
plot(tsq.lack$myDateTime, tsq.lack$CSB, pch = 16)
lines(tsq.rep$myDateTime, tsq.rep$CSB, type = "b", col = "red")

# Look's nice, what do you think? And we did not even miss a
# maximum.

# From the plot we already see that the time intervals of which
# we have data are not exactly the same. Nevertheless we would
# like to join hydraulic and water quality data into one and
# the same data frame.
# How can we do this? Ask R!
??join # -> we find R's function merge

# It's so easy! just give the two (repaired) data frames
# and the names of the columns which need to be synchronized
# With [,1:3] we select only the first three columns (we are not
# interested any more in the original (missing) values
```

```
merge(tsh.rep[,1:3], tsq.rep[,1:3], by.x = "Zeitst", by.y = "myDateTime")

# The result is a time series from 2:30 to 2:47. If we want to
# see all data from both tables we need to specify "all = TRUE":
tshq <- merge(tsh.rep[,1:3], tsq.rep[,1:3], by.x = "Zeitst", by.y = "myDateTime", all = TRUE)

# We see that rows have been appended to the beginning and to
# the end of the data frame.
# Again we could try to "repair" but as hsFillUp will not extra-
# polate nothing changes:
hsFillUp(tshq)

# However, if we give first values for Q and v...
row1 <- data.frame(Zeitst = hsToPosix("2010-07-23 02:28:00"), Q = 1.0, v = 0.4, CSB = 30.0, CSBf = 20.8
rown <- data.frame(Zeitst = hsToPosix("2010-07-23 02:51:00"), Q = 1.4, v = 0.5, CSB = 90.0, CSBf = 32.1
tshq.ext <- rbind(row1, tshq[,1:5], rown)
tshq.ext

# Fill up again...
tshq.rep <- hsFillUp(tshq.ext)
tshq.rep

# Unfortunately the comments have been stripped off when creating
# the package but I will fix this...

# The last step could be to write the new table back into the
# database:
##### hsDropTable(mdb, "tbl_hyd_qua_repaired")
tbl <- hsPutTable(mdb, tshq.rep[,1:5], "tbl_hyd_qua_repaired")

# Check by reading again:
hsMdbTimeSeries(mdb, tbl, "Zeitst")

# That's the story. Should be enough for today.

# If you are interested in how the functions work, have a look at
# the R code by typing the name of the function without parentheses, e.g.
hsFillUp
```

### 7.4.1 Random numbers

```
# Task: Generate 10 random numbers between 1 and 10
sample(1:10, 10)
```

```
##  [1]  3  1  7  5  8  9  2  4 10  6
```

```
# How would you do that without R? Excel? Let's try:
# = Zufallsbereich()
# Ok, it works, so why use R?
# 1. R speeks English. In Excel, function names change with the language of
#    the Excel version. That makes things difficult. E.g. the English version of
#    Zufallsbereich() is RandBetween(); and in French?
# 2. In R it is one function call, in Excel it is a copy of
#    function calls. What happens if, by mistake, you press a number key
#    when having selected a random number cell? How do you know that this
#    is not a random number any more?

# Task: Create a complete sequence of timestamps between tBegin and tEnd
```

### 7.4.2 Functions

```
# Let's come back to the example of calculating the size of
# a simulation result file:
bytesRow <- nchar("dd/mm/yyyy HH:MM:SS") + 20 * 10 + 10 * nchar(";") + 2
nRows <- 365 * 24 * 12
bytesFile <- nRows * bytesRow
bytesFile / (1024*1024) # MByte
```

```
## [1] 23.15781
```

### 7.4.3 Getting help

If you work with function it is essential to know which arguments a function takes and what they mean. This information is contained in help files that can be inspected by the help command or by typing a question mark in front of the function name for which help is required. Let's look for help about the `read.csv` function:

```
help(read.csv)
```

### 7.4.4 Data import from CSV

Task:

File iwExample_Q1.csv contains flows through the outlet sewers as they were simulated by InfoWorks. I would like to know the overflow volume for each outlet and then to have a ranking of the three most important outlets in terms of overflow volume.

```
#21/04/2010 12:40:00
#iwq <- csv.get("iwExample_Q.csv", dateformat="%d/%m/%Y HH:MM:SS")
#iwq <- read.table("iwExample_Q.csv", header = TRUE)
iwq <- read.csv("iwExample_Q1.csv", header = TRUE, sep = ";",
    blank.lines.skip = TRUE, as.is = FALSE, fill = FALSE)

# Convert data columns to matrix so that we can calculate
iwqm <- as.matrix(iwq[,2:ncol(iwq)])

# For each column calculate the sum over all rows
cs <- colSums(iwqm)

# That was the sum of flows. Multiply with timestep (60s) to gain total volume
cs <- cs * 60

# At how many outlets (each column represents an outlet) the total volume was
# above 1000 m3?
sum(cs > 1000)
```

```
# The three main outlets and their volume
cs[order(cs, decreasing = TRUE)[1:3]]

#X19204903.1 X17255907.1 X22225703.1
#  170256.88    81728.78    78501.65

## Import inother CSV file in another format:
iwq2 <- read.csv("iwExample_Q2_DE.csv", header = TRUE, sep = ";")

# Converting text to double by replacing "," with "." in each data column
for (i in 3:ncol(iwq2))
  iwq2[,i] <- as.numeric(gsub(",", "\\.", iwq2[,i]))
```

## 7.5 What about date conversion problems?

```
# Let's open file "dateConfusion.csv" with Excel...
# -> Excel interpreted the date format as dd/mm/yyyy which only worked
# for the first datasets; if this was a longer dataset you may not have been
# aware of the problem...
dcf <- read.csv("dateConfusion.csv", header = TRUE)
datestr <- gsub("(\\d\\d)/(\\d\\d)/(\\d\\d\\d\\d)", "\\3-\\1-\\2", dcf[,1])

# Convert string to date
dates <- as.Date(datestr)

# Print date in format you want
format.Date(dates, "%d.%m.%Y")

#
# --> YOU have the control over data type conversion AND NOT Excel!!!
#
```