Statistical Practice in Epidemiology with R
Computer exercises

University of Tartu, Estonia
23–28 August 2019
http://bendixcarstensen.com/SPE
Compiled Tuesday 30 May, 2023, 12:48
from: /home/runner/work/SPE/SPE/build/pracs.tex

Krista Fischer — Estonian Genome Center, University of Tartu, Estonia
Krista.Fischer@ut.ee

Martyn Plummer — University of Warwick, UK
martyn.plummer@warwick.ac.uk

Janne Pitkäniemi — Finnish Cancer Registry, Helsinki, Finland
janne.pitkaniemi@cancer.fi

Bendix Carstensen — Steno Diabetes Center Copenhagen, Gentofte, Denmark
& Dept. of Biostatistics, University of Copenhagen, Denmark
b@bxc.dk
http://BendixCarstensen.com

Damien Geroges — International Agency for Research on Cancer, Lyon, France
gerogesd@iarc.fr

Esa Läärä — Department of Mathematical Sciences, University of Oulu, Finland
Esa.Laara@oulu.fi
http://www.oulu.fi/university/researcher/esa-laara

Program

Daily timetable

9:00 – 9:30	Recap of yesterday's practicals
9:30 – 10:30	Lecture
10:30 – 11:00	Coffee break
11:00 – 13:00	Practical
13:00 – 14:00	Lunch
14:00 – 14:30	Recap of morning's practical
14:30 – 15:30	Lecture
15:30 – 16:00	Tea break
16:00 – 18:00	Practical

Friday 2 June – Day 1

8:30 – 9:00	Registration
9:00 – 9:15	Welcome (KF)
9:15 – 10:15	R History and Ecology
10:15 – 10:30	Data manipulation with `dplyr` (DG)
10:30 – 11:00	Coffeee break
11:00 – 13:00	Practical—choose one: 1: Practice with basic R & Simple data input 2: Data manipulation with `dplyr`
13:00 – 14:00	Lunch
14:00 – 14:30	Recap of morning practical
14:30 – 15:30	Practical: Tabulation & base graphics (KF)
15:30 – 16:00	Tea break
16:00 – 16:30	Simple Poisson and binary regression (JP)
16:30 – 18:00	Practical: Incidence rates and proportions and their contrasts by `glm`
18:00 –	Welcome reception: at ??

Saturday 3 June – Day 2

9:00 – 9:30	Recap of yesterday's practicals.
9:30 – 10:00	Introduction to splines (MP)
10:00 – 10:45	Linear and generalized linear models (EL)
10:45 – 11:15	Coffeee break
11:15 – 13:00	Practical: Linear regression & Poisson regression with estimation and reporting of linear and curved effects
13:00 – 14:00	Lunch
14:00 – 14:30	Recap of morning practical
14:30 – 15:30	Causal inference 1: basic concepts
15:30 – 16:00	Tea break
16:00 – 18:00	Practical: Causal inference

Sunday 4 June – Day 3

9:00 – 9:30	Recap of yesterday's practicals
9:30 – 10:30	More advanced graphics in R, including `ggplot2` (DG)
10:30 – 11:00	Coffee break.
11:00 – 13:00	Practical: Graphical meccano

Monday 5 June – Day 4

9:00 – 9:30	Recap of yesterday's practicals
9:30 – 10:30	Survival analysis: Kaplan Meier & basic Cox-model, and basic analysis of competing risks (JP)
10:30 – 11:00	Coffee break
11:00 – 13:00	Practical: Survival and competing risks in oral cancer
13:00 – 14:00	Lunch
14:00 – 14:30	Recap of morning practical
14:30 – 15:30	Dates in R; follow-up representation in `Lexis` objects, time-splitting, and SMR (BxC)
15:30 – 16:00	Tea break.
16:00 – 18:00	Practical: Time-splitting and SMR in Danish diabetes patients

Tuesday 6 June – Day 5

9:00 – 9:30	Recap of yesterday's practicals
9:30 – 10:30	Nested and matched cc-studies & Case-cohort studies (KF)
10:30 – 11:00	Coffee break.
11:00 – 13:00	Practical: CC study: Risk factors for coronary heart disease
13:00 – 14:00	Lunch
14:00 – 14:30	Recap of morning practical
14:30 – 15:30	Causal inference 2: model-based estimation of causal contrasts (EL)
15:30 – 16:00	Tea break.
16:00 – 18:00	Practical: Model-based estimation of causal contrasts (EL)
19:00 –	Course dinner, venue TBA

Wednesday 7 June – Day 6

9:00 – 9:30	Recap of yesterday's practicals
9:30 – 10:30	Multistate models, Poisson models for rates and simulation of `Lexis` objects (BxC)
10:30 – 11:00	Coffee break.
11:00 – 12:15	Practical: Multistate-model: Renal complications
12:15 – 12:45	Recap of morning practical
12:45 – 13:00	Wrap-up and farewell.
13:00 – 14:00	Lunch

Further material will appear at this year's course website:
http://bendixcarstensen.com/SPE

Chapter 1

Exercises

Data sets

Datasets for the practicals in this course, as well as some useful Rscripts, will be available on the course homepage, in https://spe-r.github.io/. This is where you will also find the "housekeeping" scripts designed to save you typing.

To get all files in one go, download the zip file https://github.com/SPE-R/SPE/raw/gh-spe-material/SPE-all-material.zip.

Graphical User Interfaces to R

When running the exercises it is a good idea to use a text editor instead of typing your commands directly at the R prompt. On Windows and macOS, R comes with a basic graphical user interface including a built-in text editor. Many people like to use the RStudio interface to R, which includes a very powerful syntax-highlighting editor.

Keyboard shortcuts

In the past we have found that some participants have had difficulty finding keys for symbols that are commonly used in the R language. In particular, the tilde symbol ~ is used in all modelling functions but not directly available on some keyboards. If this affects you then please consult the Wikipedia page: http://en.wikipedia.org/wiki/Tilde#Keyboards for advice on the combination of key presses you will need to get tilde.

Recaps

The R-scripts used during the course for the recaps will be available in http://BendixCarstensen.com/SPE/recap.

Ask for help

The faculty are here to help you. Ask them for help.

1.1 Practice with basic R

The main purpose of this session is to give participants who have not had much (or any) experience with using R a chance to practice the basics and to ask questions. For others, it should serve as a reminder of some of the basic features of the language.

R can be installed on all major platforms (*i.e.* Windows, macOS, Linux). We do not assume in this exercise that you are using any particular platform. Many people like to use the RStudio graphical user interface (GUI), which gives the same look and feel across all platforms.

1.1.1 The working directory

A key concept in R is the *working directory* (or *folder* in the terminology of Windows). The working directory is where R looks for data files that you may want to read in and where R will write out any files you create. It is a good idea to keep separate working directories for different projects. In this course we recommend that you keep a separate working directory for each exercise.

If you are working on the command line in a terminal, then you can change to the correct working directory and then launch R by typing "R".

If you are using a GUI then you will typically need to change to the correct working directory after starting R. In RStudio, you can change directory from the "Session" menu. However it is much more useful to create a new *project* to keep your source files and data. When you open a project in the RStudio GUI, your working directory is automatically changed to the directory associated with the project.

You can quit R by typing

```
q()
```

at the R command prompt. You will be asked if you want to save your workspace. We recommend that you answer "no" to this question. If you answer "yes" then R will write a file named `.RData` into the working directory containing all the objects you created during your session so that they will be available next time you start R. This may seem convenient but you will soon find that your workspace becomes cluttered with old objects.

You can display the current working directory with the `getwd()` function and set it with the `setwd()` function. The function `dir()` can be used to see what files you have in the working directory.

1.1.2 Read-evaluate-print

The simplest use of R is interactively. R will read in the command you type, evaluate them, then print out the answer. This is called the *read-eval-print loop*, or REPL for people who don't like words. In this exercise, we recommend that you work interactively. As the course evolves you will find that you need to switch to *script files*. We come back to this issue at the end of the exercise.

It is important to remember that R is case sensitive, so that `A` is different from `a`. Commands in R are generally separated by a newline, although a semi-colon can also be used.

1.1.3 Using R as a calculator

Try using R as a calculator by typing different arithmetic expressions on the command line. Note that R allows you to recall previous commands using the vertical arrow key. You can edit a recalled command and then resubmit it by pressing the return key. Keeping that in mind, try the following:

```
12+16
(12+16)*5
sqrt((12+16)*5)  # square root
round(sqrt((12+16)*5),2)  # round to two decimal places
```

The hash symbol # denotes the start of a *comment*. Anything after the hash is ignored by R.

Round braces are used a lot in R. In the above expressions, they are used in two different ways. Firstly, they can be used to establish the order of operations. In the example

```
> (12+16)*5
[1] 140
```

they ensure that 12 is added to 16 before the result is multiplied by 5. If you omit the braces then you get a different answer

```
> 12+16*5
[1] 92
```

because multiplication has higher *precedence* than addition. The second use of round braces is in a function call (e.g. `sqrt`, `round`). To call a function in R, type the name followed by the arguments inside round braces. Some functions take multiple arguments, and in this case they are separated by commas.

You can see that complicated expressions in R can have several levels of nested braces. To keep track of these, it helps to use a syntax-highlighting editor. For example, in RStudio, when you type an opening bracket "(", RStudio will automatically add a closing bracket ")", and when the cursor moves past a closing bracket, RStudio will automatically highlight the corresponding opening bracket (in grey). Features like this can make it much easier to write R code free from syntax errors.

Instead of printing the result you can store it in an object, say

```
a <-  round(sqrt((12+16)*5),2)
```

In this case R does not print anything to the screen. You can see the results of the calculation, stored in the object `a`, by typing `a` and also use `a` for further calculations, e.g:

```
exp(a)
log(a)     # natural logarithm
log10(a)   # log to the base 10
```

The left arrow expression `<-`, pronounced "gets", is called the assignment operator, and is obtained by typing < followed by - (with no space in between). It is also possible to use the equals sign = for assignment. Note that some R experts do not like this and recommend to use only "gets" for assignment, reserving = for function arguments,

You can also use a right arrow, as in

```
round(sqrt((12+16)*5),2) -> a
```

1.1.4 Vectors

All commands in R are *functions* which act on *objects*. One important kind of object is a *vector*, which is an ordered collection of numbers, or character strings (e.g. "Charles Darwin") , or logical values (`TRUE` or `FALSE`). The components of a vector must be of the same type (numeric, character, or logical). The combine function `c()`, together with the assignment operator, is used to create vectors. Thus

```
> v <- c(4, 6, 1, 2.2)
```

creates a vector `v` with components 4, 6, 1, 2.2 and assigns the result to the vector `v`.

A key feature of the R language is that many operations are *vectorized*, meaning that you can carry out the same operation on each element of a vector in a single operation. Try

```
> v
> 3+v
> 3*v
```

and you will see that R understands what to do in each case.

R extends ordinary arithmetic with the concept of a *missing value* represented by the symbol `NA` (Not Available). Any operation on a missing value creates another missing value. You can see this by repeating the same operations on a vector containing a missing value:

```
> v <- c(4, 6, NA)
> 3 + v
> 3 * v
```

The fact that every operation on a missing value produces a missing value can be a nuisance when you want to create a summary statistic for a vector:

```
> mean(v)
[1] NA
```

While it is true that the mean of `v` is unknown because the value of the third element is missing, we normally want the mean of the non-missing elements. Fortunately the `mean` function has an optional argument called `na.rm` which can be used for this.

```
> mean(v, na.rm=TRUE)
[1] 5
```

Many functions in R have optional arguments that can be omitted, in which case they take their default value (in this case `na.rm=FALSE`), or can be explicitly given in the function call to override the default behaviour.

You can get a description of the structure of any object using the function `str()`. For example, `str(v)` shows that `v` is numeric with 4 components. If you just want to know the length of a vector then it is much easier to use the `length` function.

```
> length(v)
```

1.1.5 Sequences

There are short-cut functions for creating vectors with a regular structure. For example, if you want a vector containing the sequence of integers from 1 to 10, you can use

```
> 1:10
```

The `seq()` function allows the creation of more general sequences. For example, the vector (15, 20, 25, ... ,85) can be created with

```
> seq(from=15, to=85, by=5)
```

The objects created by the ":" operator and the `seq()` function are ordinary vectors, and can be combined with other vectors using the combine function:

```
> c(5, seq(from=20, to=85, by=5))
```

You can learn more about functions by typing ? followed by the function name. For example `?seq` gives information about the syntax and usage of the function `seq()`.

1. Create a vector `w` with components 1, -1, 2, -2
2. Display this vector
3. Obtain a description of `w` using `str()`
4. Create the vector `w+1`, and display it.
5. Create the vector `v` with components (5, 10, 15, ... , 75) using seq().
6. Now add the components 0 and 1 to the beginning of `v` using c().
7. Find the length of this vector.

1.1.6 Displaying and changing parts of a vector (indexing)

Square brackets in R are used to extract parts of vectors. So `x[1]` gives the first element of vector `x`. Since R is vectorized you can also supply a vector of integer index values inside the square brackets. Any expression that creates an integer vector will work.

Try the following commands:

```
> x <- c(2, 7, 0, 9, 10, 23, 11, 4, 7, 8, 6, 0)
> x[4]
> x[3:5]
> x[c(1,5,8)]
> x[(1:6)*2]
> x[-1]
```

Negative subscripts mean "drop this element". So `x[-1]` returns every element of `x` except the first.

Trying to extract an element that is beyond the end of the vector is, surprisingly, not an error. Instead, this returns a missing value

```
> N <- length(x)
> x[N + 1]
[1] NA
```

There is a reason for this behaviour, which we will discuss in the recap.

R also allows *logical subscripting*. Try the following

```
> x > 10
> x[x > 10]
```

The first expression creates a logical vector of the same length as `x`, where each element has the value `TRUE` or `FALSE` depending on whether or not the corresponding element of `x` is greater than 10. If you supply a logical vector as an index, R selects only those elements for which the conditions is `TRUE`.

You can combine two logical vectors with the operators `&` ("logical and") and `|` ("logical or"). For example, to select elements of x that are between 10 and 20 we combine two one-sided logical conditions for $x \geq 10$ **and** $x \leq 20$:

```
> x[x >= 10 & x <= 20]
```

The remaining elements of `x` that are **either** less than 10 **or** greater than 20 are selected with

```
> x[x < 10 | x > 20]
```

Indexing can also be used to replace parts of a vector:

```
> x[1] <- 1000
> x
```

This replaces the first element of `x`. Logical subscripting is useful for replacing parts of a vector that satisfy a certain condition. For example to replace all elements that take the value 0 with the value 1:

```
> x[x==0] <- 1
> x
```

If you want to replace parts of a vector then you need to make sure that the replacement value is either a single value, as in the example above, or a vector equal in length to the number of elements to be replaced. For example, to replace elements 2, 3, and 4 we need to supply a vector of replacement values of length 3.

```
> x[2:4] <- c(0, 8, 1)
> x
```

It is important to remember this when you are using logical subscripting because the number of elements to be replaced is not given explicitly in the R code, and it is easy to get confused about how many values need to be replaced. If we want to add 3 to every element that is less than 3 then we can break the operation down into 3 steps:

```
> y <- x[x < 3]
> y <- y + 3
> x[x < 3] <- y
> x
```

First we extract the values to be modified, then we modify them, then we write back the modified values to the original positions. R experts will normally do this in a single expression.

```
> x[x < 3] <- x[x < 3] + 3
```

Remember, if you are confused by a complicated expression you can usually break it down into simpler steps.

If you want to create an entirely new vector based on some logical condition then use the `ifelse()` function. This function takes three arguments: the first is a logical vector; the second is the value taken by elements of the logical vector that are `TRUE`; and the third is the value taken by elements that are `FALSE`.

In this example, we use the remainder operator `%%` to identify elements of `x` that have value 0 when divided by 2 (i.e. the even numbers) and then create a new character vector with the labels "even" and "odd":

```
> x %% 2
> ifelse(x %% 2 == 0,"even","odd")
```

Now try the following:

8. Display elements that are less than 10, but greater than 4
9. Modify the vector x, replacing by 10 all values that are greater than 10
10. Modify the vector x, multiplying by 2 all elements that are smaller than 5 (Remember you can do this in steps).

1.1.7 Lists

Collections of components of different types are called *lists*, and are created with the `list()` function. Thus

```
> m <- list(4, TRUE, "name of company")
> m
```

creates a list with 3 components: the first is numeric, the second is logical and the third is character. A list element can be any object, including another list. This flexibility means that functions that need to return a lot of complex information, such as statistical modelling functions, often return a list.

As with vectors, single square brackets are used to take a subset of a list, but the result will always be another list, even if you select only one element

```
> m[1:2] #A list containing first two elements of m
> m[3]   #A list containing the third element of m
```

If you just want to extract a single element of a list then you must use double square braces:

```
> m[[3]] #Extract third element
```

Lists are more useful when their elements are named. You can name an element by using the syntax `name=value` in the call to the `list` function:

```
> mylist <- list(name=c("Joe","Ann","Jack","Tom"),
+                age=c(34,50,27,42))
> mylist
```

This creates a new list with the elements "name", a character vector of names, and "age" a numeric vector of ages. The components of the list can be extracted with a dollar sign `$`

```
> mylist$name
> mylist$age
```

1.1.8 Data frames

Data frames are a special structure used when we want to store several vectors of the same length, and corresponding elements of each vector refer to the same record. For example, here we create a simple data frame containing the names of some individuals along with their age in years, their sex (coded 1 or 2) and their height in cm.

```
> mydata <- data.frame(name=c("Joe","Ann","Jack","Tom"),
+                      age=c(34,50,27,42),sex=c(1,2,1,1),
+                      height=c(185,170,175,182))
```

The construction of a data frame is just like a named list (except that we use the constructor function `data.frame` instead of `list`). In fact data frames are lists so, for example, you can extract vectors using the dollar sign or other extraction methods for lists:

```
> mydata$height
> mydata[[4]]
```

On the other hand, data frames are also two dimensional objects:

```
> mydata
  name age sex height
1  Joe  34   1    185
2  Ann  50   2    170
3 Jack  27   1    175
4  Tom  42   1    182
```

When you print a data frame, each variable appears in a separate column. You can use square brackets with two comma-separated arguments to take subsets of rows or columns.

```
> mydata[1,]
> mydata[,c("age", "height")]
> mydata[2,4]
```

We will look into indexing of data frames in more detail below.

Now let's create another data frame with more individuals than the first one:

```
> yourdata <- data.frame(name=c("Ann","Peter","Sue","Jack","Tom","Joe","Jane"),
+                        weight=c(67,81,56,90,72,79,69))
```

This new data frame contains the weights of the individuals. The two data sets can be joined together with the `merge` function.

```
> newdata <- merge(mydata, yourdata)
> newdata
```

The `merge` function uses the variables common to both data frames – in this case the variable "name" – to uniquely identify each row. By default, only rows that are in both data frames are preserved, the rest are discarded. In the above example, the records for Peter, Sue, and Jane, which are not in `mydata` are discarded. If you want to keep them, use the optional argument `all=TRUE`.

```
> newdata <- merge(mydata, yourdata, all=TRUE)
> newdata
```

This keeps a row for all individuals but since Peter, Sue and Jane have no recorded age, height, or sex these are missing values.

1.1.9 Working with built-in data frames

We shall use the births data which concern 500 mothers who had singleton births (i.e. no twins) in a large London hospital. The outcome of interest is the birth weight of the baby, also dichotomised as normal or low birth weight. These data are available in the Epi package:

```
> library(Epi)
> data(births)
> objects()
```

The function `objects()` shows what is in your workspace. To find out a bit more about `births` try

```
help(births)
```

11. The dataframe `"diet"` in the Epi package contains data from a follow-up study with coronary heart disease as the end-point. Load these data with

    ```
    > data(diet)
    ```

    and print the contents of the data frame to the screen..

12. Check that you now have two objects, `births`, and `diet` in your workspace.

13. Get help on the object `diet`.

14. Remove the object `diet` with the command

    ```
    > remove(diet)
    ```

    Check that the object `diet` is no longer in your workspace.

1.1.10 Referencing parts of the data frame (indexing)

Typing `births` will list the entire data frame – not usually very helpful. You can use the `head` function to see just the first few rows of a data frame

```
> head(births)
```

Now try

```
> births[1,"bweight"]
```

This will list the value taken by the first subject for the `bweight` variable. Alternatively

```
> births[1,2]
```

will list the value taken by the first subject for the second variable (which is `bweight`). Similarly

```
> births[2,"bweight"]
```

will list the value taken by the second subject for `bweight`, and so on. To list the data for the first 10 subject for the `bweight` variable, try

```
> births[1:10, "bweight"]
```

and to list all the data for this variable, try

```
> births[, "bweight"]
```

To list the data for the first subject try

```
> births[1, ]
```

An empty index before the comma means "all rows" and an empty index after the comma means "all columns".

15. Display the data on the variable `gestwks` for row 7 in the `births` data frame.
16. Display all the data in row 7.
17. Display the first 10 rows for the variable `gestwks`.

The subset function is another way of getting subsets from a data frame.To select all subjects with height less than 180 cm from the data frame `mydata` we use

```
> subset(mydata, height < 180)
```

The subset function is usually clearer than the equivalent code using []:

```
> mydata[mydata$height < 180, ]
```

Another advantage of subset is that it will drop observations with missing values. Compare the following

```
> newdata[newdata$height < 180, ]
> subset(newdata, height < 180)
```

If `height` is missing then `subset` will drop that row. But `[]` will do something you might not expect. It will include the rows with missing `height`, but will replace every element in those rows with the missing value `NA`.

1.1.11 Summaries

A good way to start an analysis is to ask for a summary of the data by typing

```
> summary(births)
```

This prints some summary statistics (minimum, lower quartile, mean, median, upper quartile, maximum). For variables with missing values, the number of `NA`s is also printed.

To see the names of the variables in the data frame try

```
> names(births)
```

Variables in a data frame can be referred to by name, but to do so it is necessary also to specify the name of the data frame. Thus `births$hyp` refers to the variable `hyp` in the `births` data frame, and typing `births$hyp` will print the data on this variable. To summarize the variable `hyp` try

```
> summary(births$hyp)
```

Alternatively you can use

```
> with(births, summary(hyp))
```

1.1.12 Generating new variables

New variables can be produced using assignment together with the usual mathematical operations and functions. For example

```
> logbw <- log(births$bweight)
```

produces the variable `logbw` in your workspace, while

```
> births$logbw <- log(births$bweight)
```

produces the variable `logbw` in the `births` data frame.

You can also replace existing variables. For example `bweight` measures birth weight in grams. To convert the units to kilograms we replace the original variable with a new one:

```
> births$bweight <- births$bweight/1000
```

1.1.13 Turning a variable into a factor

In R categorical variables are known as *factors*, and the different categories are called the *levels* of the factor. Variables such as `hyp` and `sex` are originally coded using integer codes, and by default R will interpret these codes as numeric values taken by the variables. Factors will become very important later in the course when we study modelling functions, where factors and numeric variables are treated very differently. For the moment, you can think of factors as "value labels" that are more informative than numeric codes.

For R to recognize that the codes refer to categories it is necessary to convert the variables to be factors, and to label the levels. To convert the variable `hyp` to be a factor, try

```
> births$hyp <- factor(births$hyp, labels=c("normal", "hyper"))
```

This takes the original numeric codes (0, 1) and replaces them with informative labels "normal" and "hyper" for normal blood pressure and hypertension, respectively.

18. Convert the variable `sex` into a factor with labels `"M"` and `"F"` for values 1 and 2, respectively

1.1.14 Frequency tables

When starting to look at any new data frame the first step is to check that the values of the variables make sense and correspond to the codes defined in the coding schedule. For categorical variables (factors) this can be done by looking at one-way frequency tables and checking that only the specified codes (levels) occur. The most useful function for making simple frequency tables is `table`. The distribution of the factor `hyp` can be viewed using

```
> with(births, table(hyp))
```

or by specifying the data frame as in

```
> table(births$hyp)
```

For simple expressions the choice is a matter of taste, but `with` is shorter for more complicated expressions.

19. Find the frequency distribution of `sex`.

20. If you give two or more arguments to the `table` function then it produces cross-tabulations. Find the two-way frequency distribution of `sex` and `hyp`.

21. Create a logical variable called `early` according to whether `gestwks` is less than 30 or not. Make a frequency table of `early`.

1.1.15 Grouping the values of a numeric variable

For a numeric variable like `matage` it is often useful to group the values and to create a new factor which codes the groups. For example we might cut the values taken by `matage` into the groups 20–29, 30–34, 35–39, 40–44, and then create a factor called `agegrp` with 4 levels corresponding to the four groups. The best way of doing this is with the function `cut`. Try

```
> births$agegrp  <- cut(births$matage, breaks=c(20,30,35,40,45), right=FALSE)
> with(births, table(agegrp))
```

By default the factor levels are labelled `[20-25)`, `[25-30)`, etc., where `[20-25)` refers to the interval which includes the left hand end (20) but not the right hand end (25). This is the reason for `right=FALSE`. When `right=TRUE` (which is the default) the intervals include the right hand end but not the left hand.

Observations which are not inside the range specified by the `breaks` argument result in missing values for the new factor. Hence it is important that the first element in `breaks` is smaller than the smallest value in your data, and the last element is larger than the largest value.

22. Summarize the numeric variable `gestwks`, which records the length of gestation for the baby, and make a note of the range of values.

23. Create a new factor `gest4` which cuts `gestwks` at 20, 35, 37, 39, and 45 weeks, including the left hand end, but not the right hand. Make a table of the frequencies for the four levels of `gest4`.

1.1.16 Saving and loading data

As noted in section 1.1.1, at the end of the session, R will offer to save your workspace and, if you accept, it will create a file `.RData` in your working directory. In fact you can save any R object to disc. For example, to save the data frame `births` try

```
> save(births, file="births.RData")
```

which will save the births data frame in the file births.RData. If you send this file to a colleague then they can read the data back into R with

```
> load("births.RData")
```

The commands `save()` and `load()` can be used with any R objects, but they are particularly useful when dealing with large data frames. The binary format created by the `save()` functions is the same across all platforms and between R versions.

1.1.17 The search path

When you load a package with the `library()` function, the functions in that package become available for you to use via a mechanism called the *search path*. The command

```
> search()
```

shows the positions on the search path. The first position is ".GlobalEnv". This is the global environment, which is another name for your workspace. The second entry on the search path is the Epi package, the third is a package of commands called methods, the fourth is a package called stats, and so on. To see what is in the workspace try

```
> objects()
```

You should see the objects that you have created in this session. To see what is in the Epi package, try

```
> objects(2)
```

You can also refer to a package by name, not position

```
> objects("package:Epi")
```

When you type the name of an object R looks for it in the order of the search path and will return the first object with this name that it finds.

1.1.18 Attaching a data frame

The search path can also be modified by attaching a data frame. For example:

```
> attach(births)
```

This places a copy of the variables in the `births` data frame in position 2 of the search path. You can verify this with

```
> search()
> objects(2)
```

which shows the objects in this position are the variables from the `births` data frame.

Attaching a data frame makes the variables in it directly accessible. For example, when you type the command:

```
> hyp
```

you should get the variable `hyp` from the `births` data set without having to use the dollar sign. The `detach()` function removes the data frame from the search path.

```
> detach()
```

when no arguments are given, the `detach()` function removes the second entry on the search path (after the global environment).

This seems like an attractive feature, especially for people who are used to other statistical software (e.g. SAS, Stata) in which the variables in the "current working dataset" are directly accessible in this way. However, attaching data frames causes more problems than it solves and should be avoided. In particular:

- Since the attached data frame appears second in the search path, it comes after the global environment. If you have an object `hyp` in the global environment then you will get this, instead of the variable from the `births` data frame. This is called *masking*. R will warn you about masking, but only once for each variable.
- Attaching a data frame creates a copy of all the variables in it. Subsequent changes to the data frame (e.g. selecting rows or recoding variables) are not reflected in the attached copy, which is a snapshot of the data frame when it was attached.
- If you forget to `detach()` the data frame when you are finished with it, then you may create multiple attached copies on your search path, especially when using a script.

It is best to stick to using the dollar sign to select variables in a data frame, or to use the `with()` function. Many R functions (but not all of them) have a `data` argument which can be used to specify a data frame that should be searched before the search path.

1.1.19 Interactive use vs scripting

You can work with R simply by typing function calls at the command prompt and reading the results as they are printed. This is OK for simple use but rapidly becomes cumbersome. If the results of one calculation are used to feed into the next calculation, it can be difficult to go back if you find you have made a mistake, or if you want to repeat the same commands with different data.

When working with R it is best to use a text editor to prepare a batch file (or script) which contains R commands and then to run them from the script. If you are using a GUI then you can use the built-in script editor, or you can use your favourite text editor instead if you prefer.

One major advantage of running all your R commands from a script is that you end up with a record of exactly what you did which can be repeated at any time. This will also help you redo the analysis in the (highly likely) event that your data changes before you have finished all analyses.

1.2 Reading data into R

1.2.1 Introduction

> "If you want to have rabbit stew, first catch the rabbit" – Old saying, origin unknown

R is a language and environment for data analysis. If you want to do something interesting with it, you need data.

For teaching purposes, data sets are often embedded in R packages. The base R distribution contains a whole package dedicated to data which includes around 100 data sets. This is attached towards the end of the search path, and you can see its contents with

```
> objects("package:datasets")
```

A description of all of these objects is available using the `help()` function. For example

```
> help(Titanic)
```

gives an explanation of the `Titanic` data set, along with references giving the source of the data.

The Epi package also contains some data sets. These are not available automatically when you load the Epi package, but you can make a copy in your workspace using the `data()` function. For example

```
> library(Epi)
> data(bdendo)
```

will create a data frame called `bdendo` in your workspace containing data from a case-control study of endometrial cancer. Datasets in the Epi package also have help pages: type `help(bdendo)` for further information.

To go back to the cooking analogy, these data sets are the equivalent of microwave ready meals, carefully packaged and requiring minimal work by the consumer. Your own data will never be able in this form and you must work harder to read it in to R.

This exercise introduces you to the basics of reading external data into R. It consists of reading the same data from different formats. Although this may appear repetitive, it allows you to see the many options available to you, and should allow you to recognize when things go wrong.

> **getting the data:** You will need to download the zip file `data.zip` from the course web site https://spe-r.github.io/data.zip and unpack this in your working directory. This will create a sub-directory data containing (among other things) the files `fem.dat`, `fem-dot.dat`, `fem.csv`, and `fem.dta` (Reminder: use `setwd()` to set your working directory).

1.2.2 Data sources

Sources of data can be classified into three groups:

1. Data in human readable form, which can be inspected with a text editor.

2. Data in binary format, which can only be read by a program that understands that format (SAS, SPSS, Stata, Excel, ...).

3. Online data from a database management system (DBMS)

This exercise will deal with the first two forms of data. Epidemiological data sets are rarely large enough to justify being kept in a DBMS. If you want further details on this topic, you can consult the "R Data Import/Export" manual that comes with R.

1.2.3 Data in text files

Human-readable data files are generally kept in a rectangular format, with individual records in single rows and variables in columns. Such data can be read into a data frame in R.

Before reading in the data, you should inspect the file in a text editor and ask three questions:

1. How are columns in the table separated?

2. How are missing values represented?

3. Are variable names included in the file?

The file `fem.dat` contains data on 118 female psychiatric patients. The data set contains nine variables.

ID	Patient identifier
AGE	Age in years
IQ	Intelligence Quotient (IQ) score
ANXIETY	Anxiety (1=none, 2=mild, 3=moderate,4=severe)
DEPRESS	Depression (1=none, 2=mild, 3=moderate or severe)
SLEEP	Sleeping normally (1=yes, 2=no)
SEX	Lost interest in sex (1=yes, 2=no)
LIFE	Considered suicide (1=yes, 2=no)
WEIGHT	Weight change (kg) in previous 6 months

Inspect the file `fem.dat` with a text editor to answer the questions above.

The most general function for reading in free-format data is `read.table()`. This function reads a text file and returns a data frame. It tries to guess the correct format of each variable in the data frame (integer, double precision, or text).

Read in the table with:

```
> fem <- read.table("./data/fem.dat", header=TRUE)
```

Note that you must assign the result of `read.table()` to an object. If this is not done, the data frame will be printed to the screen and then lost.

You can see the names of the variables with

```
> names(fem)
```

The structure of the data frame can be seen with

```
> str(fem)
```

You can also inspect the top few rows with

```
> head(fem)
```

Note that the IQ of subject 9 is -99, which is an illegal value: nobody can have a negative IQ. In fact -99 has been used in this file to represent a missing value. In R the special value `NA` ("Not Available") is used to represent missing values. All R functions recognize `NA` values and will handle them appropriately, although sometimes the appropriate response is to stop the calculation with an error message.

You can recode the missing values with

```
> fem$IQ[fem$IQ == -99] <- NA
```

Of course it is much better to handle special missing value codes when reading in the data. This can be done with the `na.strings` argument of the `read.table()` function. See below.

1.2.4 Things that can go wrong

Sooner or later when reading data into R, you will make a mistake. The frustrating part of reading data into R is that most mistakes are not fatal: they simply cause the function to return a data frame that is *not what you wanted.* There are three common mistakes, which you should learn to recognize.

1.2.4.1 Forgetting the headers

The first row of the file `fem.dat` contains the variable names. The `read.table()` function does not assume this by default so you have to specify this with the argument `header=TRUE`. See what happens when you forget to include this option:

```
> fem2 <- read.table("data/fem.dat")
> str(fem2)
> head(fem2)
```

and compare the resulting data frame with `fem`. What are the names of the variables in the data frame? What is the class of the variables?

> **Explanation:** Remember that `read.table()` tries to guess the mode of the variables in the text file. Without the `header=TRUE` option it reads the first row, containing the variable names, as data, and guesses that all the variables are character, not numeric. By default, all character variables are coerced to factors by `read.table`. The result is a data frame consisting entirely of factors. (You can prevent the conversion of character variables to factors with the argument `as.is=TRUE`).

If the variable names are not specified in the file, then they are given default names `V1, V2, ...`. You will soon realise this mistake if you try to access a variable in the data frame by, for example

```
> fem2$IQ
```

as the variable will not exist

There is one case where omitting the `header=TRUE` option is harmless (apart from the situation where there is no header line, obviously). When the first row of the file contains **one less** value than subsequent lines, `read.table()` infers that the first row contains the variable names, and the first column of every subsequent row contains its **row name**.

1.2.4.2 Using the wrong separator

By default, `read.table` assumes that data values are separated by any amount of white space. Other possibilities can be specified using the `sep` argument. See what happens when you assume the wrong separator, in this case a tab, which is specified using the escape sequence `"\t"`

```
> fem3 <- read.table("data/fem.dat", sep="\t")
> str(fem3)
```

How many variables are there in the data set?

> **Explanation:** If you mis-specify the separator, `read.table()` reads the whole line as a single character variable. Once again, character variables are coerced to factors, so you get a data frame with a single factor variable.

1.2.4.3 Mis-specifying the representation of missing values

The file `fem-dot.dat` contains a version of the FEM dataset in which all missing values are represented with a dot. This is a common way of representing missing values, but is not recognized by default by the `read.table()` function, which assumes that missing values are represented by "NA".

Inspect the file with a text editor, and then see what happens when you read the file in incorrectly:

```
> fem4 <- read.table("data/fem-dot.dat", header=TRUE)
> str(fem4)
```

You should have enough clues by now to work out what went wrong.

You can read the data correctly using the `na.strings` argument

```
> fem4 <- read.table("data/fem-dot.dat", header=TRUE, na.strings=".")
```

1.2.5 Spreadsheet data

Spreadsheets have become a common way of exchanging data. All spreadsheet programs can save a single sheet in *comma-separated variable* (CSV) format, which can then be read into R. There are two functions in R for reading in CSV data: `read.csv()` and `read.csv2()`.

Both of these are wrappers around the `read.table()` function, *i.e.* the `read.table()` function is still doing the work of reading in the data but the `read.csv()` function provides default argument values for reading in CSV file so all you need to do is specify the file name.

You can see what these default arguments are with the `args()` function.

```
> args(read.csv)
> args(read.csv2)
```

See if you can spot the difference between `read.csv` and `read.csv2`.

> **Explanation:** The CSV format is not a single standard. The file format depends on the *locale* of your computer – the settings that determine how numbers are represented. In some countries, the decimal separator is a point "." and the variable separator in a CSV file is a comma ",". In other countries, the decimal

separator is a comma "," and the variable separator is a semi-colon ";". This is reflected in the different default values for the arguments `sep` and `dec`. The `read.csv()` function is used for the first format and the `read.csv2()` function is used for the second format.

The file `fem.csv` contains the FEM dataset in CSV format. Inspect the file to work out which format is used, and read it into R.

1.2.6 Reading data from the Internet

You can also read in data from a remote web site. The `file` argument of `read.table()` does not need to be a local file on your computer; it can be a Uniform Resource Locator (URL), *i.e.* a web address.

A copy of the file `fem.dat` is held at https://www.bendixcarstensen.com/SPE/data/fem.dat. You can read it in with

```
> fem6 <- read.table("http://www.bendixcarstensen.com/SPE/data/fem.dat",
+                    header=TRUE)
> str(fem6)
```

1.2.7 Reading from the clipboard

On Microsoft Windows, you can copy values directly from an open Excel spreadsheet using the clipboard. Highlight the cells you want to copy in the spread sheet and select copy from the pull-down edit menu. Then type `read.table(file="clipboard")` to read the data in.

There are two reasons why this is a bad idea

- It is not reproducible. In order to read the data in again you need to complete exactly the same sequence of mouse moves and clicks, and there is no record of what you did before.
- Copying from the clipboard loses precision. If you have a value 1.23456789 in your spreadsheet, but have formatted the cell so it is displayed to two decimal places, then the value read into R will be the truncated value 1.23.

1.2.8 Binary data

The foreign package allows you to read data in binary formats used by other statistical packages. Since R is an open source project, it can only read binary formats that are themselves "open", in the sense that the standards for reading and writing data are well-documented. For example, there is a function in the foreign package for reading SAS XPORT files, a format that has been adopted as a standard by the US Food and Drug Administration (http://www.sas.com/govedu/fda/faq.html). However, there is no function in the foreign package for reading native SAS binaries (SAS7BDAT files). Other packages are available from CRAN (http://cran.r-project.org) that offer the possibility of reading SAS binary files: see the haven and sas7bdat packages.

The file `fem.dta` contains the FEM dataset in the format used by Stata. Read it into R with

```
> library(foreign)
> fem5 <- read.dta("data/fem.dta")
> head(fem5)
```

The Stata data set contains value and variable labels. Stata variables with value labels are automatically converted to factors.

There is no equivalent of variable labels in an R data frame, but the original variable labels are not lost. They are still attached to the data frame as an invisible *attribute*, which you can see with

```
> attr(fem5, "var.labels")
```

A lot of *meta-data* is attached to the data in the form of attributes. You can see the whole list of attributes with

```
> attributes(fem5)
```

or just the attribute names with

```
> names(attributes(fem5))
```

The `read.dta()` function can only read data from Stata versions 5–12. The R Core Team has not been able to keep up with changes in the Stata format. You may wish to try the haven package and the readstata13 package, both available from CRAN.

1.2.9 Summary

In this exercise we have seen how to create a data frame in R from an external text file. We have also reviewed some common mistakes that result in garbled data.

The capabilities of the foreign package for reading binary data have also been demonstrated with a sample Stata data set.

1.3 Tabulation

1.3.1 Introduction

R and its add-on packages provide several different tabulation functions with different capabilities. The appropriate function to use depends on your goal. There are at least three different uses for tables.

The first use is to create simple summary statistics that will be used for further calculations in R. For example, a two-by-two table created by the `table` function can be passed to `fisher.test`, which will calculate odds ratios and confidence intervals. The appearance of these tables may, however, be quite basic, as their principal goal is to create new objects for future calculations.

A quite different use of tabulation is to make "production quality" tables for publication. You may want to generate reports from for publication in paper form, or on the World Wide Web. The package `xtable` provides this capability, but it is not covered by this course.

An intermediate use of tabulation functions is to create human-readable tables for discussion within your work-group, but not for publication. The `Epi` package provides a function `stat.table` for this purpose, and this practical is designed to introduce this function.

1.3.2 The births data

We shall use the births data which concern 500 mothers who had singleton births in a large London hospital. The outcome of interest is the birth weight of the baby, also dichotomised as normal or low birth weight. These data are available in the Epi package:

```
> library(Epi)
> data(births)
> names(births)
> head(births)
```

In order to work with this data set we need to transform some of the variables into factors. This is done with the following commands:

```
> births$hyp <- factor(births$hyp,labels=c("normal","hyper"))
> births$sex <- factor(births$sex,labels=c("M","F"))
> births$agegrp <- cut(births$matage,breaks=c(20,25,30,35,40,45),right=FALSE)
> births$gest4 <- cut(births$gestwks,breaks=c(20,35,37,39,45),right=FALSE)
```

Now use `str(births)` to examine the modified data frame. We have transformed the binary variables `hyp` and `sex` into factors with informative labels. This will help when displaying the tables. We have also created grouped variables `agegrp` and `gest4` from the continuous variables `matage` and `gestwks` so that they can be tabulated.

1.3.3 One-way tables

The simplest table one-way table is created by

```
> stat.table(index = sex, data = births)
```

This creates a count of individuals, classified by levels of the factor `sex`. Compare this table with the equivalent one produced by the `table` function. Note that `stat.table` has a `data` argument that allows you to use variables in a data frame without specifying the frame.

You can display several summary statistics in the same table by giving a list of expressions to the `contents` argument:

```
> stat.table(index = sex, contents = list(count(), percent(sex)), data=births)
```

Only a limited set of expressions are allowed: see the help page for `stat.table` for details.

You can also calculate marginal tables by specifying `margin=TRUE` in your call to `stat.table`. Do this for the above table. Check that the percentages add up to 100 and the total for `count()` is the same as the number of rows of the data frame `births`. To see how the mean birth weight changes with `sex`, try

```
> stat.table(index = sex, contents = mean(bweight), data=births)
```

Add the count to this table. Add also the margin with `margin=TRUE`. As an alternative to `bweight` we can look at `lowbw` with

```
> stat.table(index = sex, contents = percent(lowbw), data=births)
```

All the percentages are 100! To use the `percent` function the variable `lowbw` must also be in the index, as in

```
> stat.table(index = list(sex,lowbw), contents = percent(lowbw), data=births)
```

The final column is the percentage of babies with low birth weight by different categories of gestation.

1. Obtain a table showing the frequency distribution of `gest4`.
2. Show how the mean birth weight changes with `gest4`.
3. Show how the percentage of low birth weight babies changes with `gest4`.

Another way of obtaining the percentage of low birth weight babies by gestation is to use the ratio function:

```
> stat.table(gest4,ratio(lowbw,1,100),data=births)
```

This only works because `lowbw` is coded 0/1, with 1 for low birth weight.

Tables of odds can be produced in the same way by using `ratio(lowbw, 1-lowbw)`. The `ratio` function is also very useful for making tables of rates with (say) `ratio(D,Y,1000)` where `D` is the number of failures, and `Y` is the follow-up time. We shall return to rates in a later practical.

1.3.4 Improving the Presentation of Tables

The `stat.table` function provides default column headings based on the `contents` argument, but these are not always very informative. Supply your own column headings using *tagged* lists as the value of the `contents` argument, within a `stat.table` call:

```
> stat.table(gest4,contents = list( N=count(),
+      "(%)" = percent(gest4)),data=births)
```

This improves the readability of the table. It remains to give an informative title to the index variable. You can do this in the same way: instead of giving `gest4` as the `index` argument to `stat.table`, use a named list:

```
> stat.table(index = list("Gestation time" = gest4),data=births)
```

1.3.5 Two-way Tables

The following call gives a 2×2 table showing the mean birth weight cross-classified by `sex` and `hyp`.

```
> stat.table(list(sex,hyp), contents=mean(bweight), data=births)
```

Add the count to this table and repeat the function call using `margin = TRUE` to calculate the marginal tables. Use `stat.table` with the ratio function to obtain a 2×2 table of percent low birth weight by `sex` and `hyp`. You can have fine-grained control over which margins to calculate by giving a logical vector to the `margin` argument. Use `margin=c(FALSE, TRUE)` to calculate margins over `sex` but not `hyp`. This might not be what you expect, but the `margin` argument indicates which of the index variables are to be **marginalized out**, not which index variables are to remain.

1.3.6 Printing

Just like every other R function, `stat.table` produces an object that can be saved and printed later, or used for further calculation. You can control the appearance of a table with an explicit call to `print()`

There are two arguments to the print method for `stat.table`. The `width` argument which specifies the minimum column width, and the `digits` argument which controls the number of digits printed after the decimal point. This table

```
> odds.tab <- stat.table(gest4, list("odds of low bw" = ratio(lowbw,1-lowbw)),
+                data=births)
> print(odds.tab)
```

shows a table of odds that the baby has low birth weight. Use `width=15` and `digits=3` and see the difference.

1.4 Data manipulation with dplyr

1.4.1 Introduction

In this chapter we will reproduce *more or less* the same outputs as in chapter 1.3 using `tidyverse` packages suit.

The main goal is to familiarize you with some of the main `dplyr` and `tidyverse` features. This is an optional practical for participants that have already good basic R basis. All the rest of the course can be done without knowing `dplyr`.

1.4.2 The `births` data

A quick description of the `births` data-set (from `Epi` package) can be found in chapter 1.3.2. First of all, load the `Epi` and `dplyr` packages. Then we should (re)load the births data-set.

```
> library(Epi)
> suppressPackageStartupMessages(library(tidyverse))
> data(births)
```

1.4.3 `tibble` vs `data.frame`

Most `dplyr` functions outputs return `tibble` object instead of `data.frame`.
Inspect the class and characteristics of the `births` object.

```
> class(births)
> head(births)
```

Note: This can be summarized using `str` function

```
> str(births)
```

`births` object is a 500 x 8 data.frame.
Let's convert `births` to `tibble` format with `as_tibble` function.

```
> births_tbl <- as_tibble(births)
> class(births_tbl)
> births_tbl
> ## another way to visualize data set is to use glimpse function
> glimpse(births_tbl)
```

As you can see `tibble` inherits from `data.frame` which implies that all functions working with `data.frame` objects will work with `tibble` objects. The opposite is not necessary true. `tibble` has a couple of extra features compared to classical `data.frame`. One of them is a slightly more user-friendly console print. The main differences being that `tibble` objects supports grouping/nesting features. Some examples we be done will see latter on.

1.4.4 Piping functions

This is one of the most popular features of `tidyverse` grammar. It enables function chaining in `R`. Function outputs are passed as input to the following function and so on. It can help to make the code more readable. Here is an example of classic vs piped functions.

```
> head(births, 4)
> births %>% head(4)
```

Note: By default the chained object is given as the first argument to the following function. You can use . if this is not the case.
Here is a dummy example where we do not give the first argument to a function but the second one.

```
> 4 %>% head(births, .)
```

1.4.5 `mutate` columns

`mutate` will allow you to add and or modify columns in a `data.frame`.
Let's create 2 new variables :

- `agegrp` (5 years mother's age group)
- `gest4` (gestation time split in 4 categories)

And modify 2 others:

- `hyp` (factor version of `hyp`; `normal` vs `hyper`)
- `sex` (factor version of `sex`; `M` vs `F`)

```
> births_tbl <-
+   births_tbl %>%
+   mutate(
+     ## modify hyp varible (conversion into factor)
+     hyp = factor(hyp, levels = c(0, 1), labels = c("normal", "hyper")),
+     ## creating a new variable aggrep
+     agegrp = cut(matage, breaks = c(20, 25, 30, 35, 40, 45), right = FALSE),
+     ## modify sex variable (conversion into factor)
+     sex = factor(sex, levels = c(1, 2), labels = c("M", "F")),
+     ## creating a new variable gest4 with case_when instead of cut
+     gest4 =
+       case_when(
+         gestwks < 25 ~ 'less than 25 weeks',
+         gestwks >= 25 & gestwks < 30  ~ '25-30 weeks',
+         gestwks >= 30 & gestwks < 35  ~ '30-35 weeks',
+         gestwks >= 35 & gestwks < 40  ~ '35-40 weeks',
+         gestwks >= 40  ~ 'more than 40 weeks'
+       )
+   )
> births_tbl
```

You can see in the table columns header the type of data contained in each column. For instance `<dbl>` stands for double (i.e. numeric value) and `fct` stands for factor. In `R` `data.frame` (/ `tibble`) data type have to be the same within a column (e.g. numeric only) but can be of different type across columns.

Notes than `case_when` function do not return a `factor` but a `character` variable in this case. You will have to force the conversion to `factor` if needed.

1.4.6 `select` columns, `filter` and `arrange` rows

`select` is used for column sub-setting while filter is for row sub-setting. They are equivalent to the `[]` in `R` base language.
Let's display a table where only babies' `id`, `sex`, `bweight` and mothers' `agegrp` are kept for babies with a `bweight` grater than 4000g.

```
> births_tbl %>%
+   ## select only id, women age group, sex and birth weight of the baby
+   select(id, agegrp, sex, bweight) %>%
+   ## keep only babies weighing more than 4000g
+   filter(bweight > 4000)
```

`select` can also be useful to reorder and rename columns.
`arrange` is a nice feature to reorder observations according to chosen attributes.
Let's rename `agegrp`, `sex` and `bweight` with more explicit labels and reorder the table according to babies' decreasing birth weight.

```
> births_tbl %>%
+   ## select only id, women age group, sex and birth weight of the baby
+   select(
+     id,
+     'Age group' = agegrp,
+     Sex = sex,
+     'Birth weight' = bweight
+   ) %>%
+   ## rearrange rows to put the heaviest newborn on top
+   arrange(desc(`Birth weight`))
```

Note: `tibble` supports blank spaces in the column names which can be handy for final table rendering. When you want to work with columns with blank spaces, do not forget to use the ` (back-quote).
Let's do the same but arranging the table by decreasing birth weights within each sex.

```
> births_tbl %>%
+   ## select only id, women age group, sex and birth weight of the baby
+   select(
+     id,
+     'Age group' = agegrp,
+     Sex = sex,
+     'Birth weight' = bweight
+   ) %>%
+   ## rearrange rows to put the heaviest newborn on top
+   arrange(Sex, desc(`Birth weight`))
```

1.4.7 `group_by` and `summarise` data

One of the most valuable features of `dplyr` is the ability to aggregate data sharing a common attribute to process by group operations.
Here we want to compute the number of boys and girls in the data-set.
The idea here is to split the `births` table in 2. One for boys, the other for girls and to count the number of rows of each sub-table.

```
> births.01 <-
+   births_tbl %>%
+   ## group the data according to the sex attribute
+   group_by(sex) %>%
+   ## count the number of rows/individuals in each group
+   summarise(
+     count = n()
+   )
> births.01
```

Note: `n` function is equivalent to `nrow`
Now we have the number of boys and girls, we can compute the percentage of newborns in each sex group.

```
> births.02 <-
+   births.01 %>%
+   mutate(
+     percent = count / sum(count) * 100
+   )
```

Trick: most of `dplyr` functions can be applied with an embedded condition using `across` function. This can be very handy in some cases.
As an illustration if you want to calculate the `sum` of every `birth.02` numerical columns

```
> births.03 <-
+   births_tbl %>%
+   select(gest4, sex, gestwks, bweight, matage) %>%
+   group_by(gest4, sex) %>%
+   summarise(
+     across(
+       where(is.numeric),
+       ~ mean(.x, na.rm = TRUE)
+     ),
+     .groups = 'drop'
+   )
> births.03
```

`across` function supports the purrr-style lambda format, e.g. `mean(.x, na.rm = TRUE)` where `.x` refers to the values from the data set to be passed to the function.

Some other functions ending by `_with` can be used conditionally within `dplyr`. As an example we can rename only columns which are not numeric at once (here we want to make all column names using upper characters) using the combination of `rename_with` and `where`.

```
> births.03 %>%
+   rename_with(toupper, where(~ !is.numeric(.x)))
```

Let's now compute the number of births and the mean birth weight according to newborn gender.

```
> births.05 <-
+   births_tbl %>%
+   group_by(sex) %>%
+   summarise(
+     count = n(),
+     bweight.mean = mean(bweight)
+   )
> births.05
```

With `births.05` table, compute the global mean new birth weight.
Note: with such a table the mean baby's birth weight have to be weighted by number of boys and girls.

```
> births.05 %>%
+   summarise(
+     count.tot = sum(count),
+     bweight.mean.tot = weighted.mean(bweight.mean, count)
+   )
> # this is equivalent to
> births_tbl %>%
+   summarise(
+     count.tot = n(),
+     bweight.mean.tot = mean(bweight)
+   )
```

1.4.8 Multiple grouping

In some cases, we can be interested in looking at more than a single strata. This can be achieved using multiple grouping.
Let's count the number of people per gender and birth weight class (low vs not low)

```
> births.06 <-
+   births_tbl %>%
+   group_by(sex, lowbw) %>%
+   summarise(
+     count = n()
+   )
> births.06
```

Try then to compute the percentage of people in each group.
Look at the difference between the 2 following command lines:

```
> births.06 %>%
+   mutate(
+     percent = count / sum(count) * 100
+   )
> births.06 %>%
+   ungroup() %>%
+   mutate(
+     percent = count / sum(count) * 100
+   )
```

Note: summarizing a data-set will remove the latest level of grouping but not the deeper ones if multiple grouping has been done. In some cases you might have to explicitly ungroup your `data.frame` before doing calculations. In the previous examples, if you do not ungroup the data-set, percentages are computed per gender. Ungrouping will let you compute the overall percentages.

Trick: a good practice is to ungroup all the summarized dataset in order to prevent form confusion. You can do it using the `.group = 'drop'` option in `summarize()`.

```
> ## this tibble will still be grouped by sex
> births_tbl %>%
+   group_by(sex, lowbw) %>%
+   summarise(
+     count = n()
+   )
> ## this tibble will be group free
> births_tbl %>%
+   group_by(sex, lowbw) %>%
+   summarise(
+     count = n(),
+     .groups = 'drop'
+   )
>
>
```

The same exercise can be done using gestation time group (`gest4`) as stratifying variable. Lets compute number and mean birth weights according to gestation time category

```
> births_tbl %>%
+   group_by(gest4) %>%
+   summarise(
+     count = n(),
+     bweight.mean = mean(bweight)
+   )
```

Birth weight increases with gestation time.
We can also spot that in our data-set the gestation time is missing for 10 newborns. We will remove this observation for the rest of the practical session.
Lets cross-tabulate the birth weight category and the gestation time groups.

```
> births_tbl %>%
+   ## keep only the newborn with defined gesational time category
+   filter(
+     !is.na(gest4)
+   ) %>%
+   group_by(lowbw, gest4) %>%
+   ## compute the number of babies in each cross category
+   summarise(
+     count = n()
+   ) %>%
+   ## compute the percentage of babies in each gestational time category per
+   ## birth weight category
+   mutate(
+     percent = count / sum(count, na.rm = TRUE)
+   )
```

Similarly we can be interested in the birth weight distribution per gestational time.

```
> births_tbl %>%
+   filter(
+     !is.na(gest4)
+   ) %>%
+   group_by(gest4, lowbw) %>%
+   summarise(
+     count = n()
+   ) %>%
```

```
+   ## compute the percentage of babies in each birth weight category per gestational
+   ## time category
+   mutate(
+     percent = count / sum(count, na.rm = TRUE)
+   )
```

Note: grouping order matters! and can be confusing so think about ungrouping intermediate tables.

1.4.9 Bind and join tables

Another nice feature of `dplyr` is tables binding and joining functions. To practice we will create 2 fake `tibble`:

- `age` an individual database which contains `pid` (unique individuals id) and `age` in year
- `center` an study center database which contains `pid` (unique individuals id) and `center` (the center where an individual is registered)

```
> age <-
+   tibble(
+     pid = 1:6,
+     age = sample(15:25, size = 6, replace = TRUE)
+   )
> center <-
+   tibble(
+     pid = c(1, 2, 3, 4, 10),
+     center = c('A', 'B', 'A', 'B', 'C')
+   )
> age
> center
```

Now the tables are define we will try to make the linkage between individuals ages and the center they belong to.
First of all let's have a look to `bind_rows` function.

```
> bind_rows(age, center)
```

Is it useful?
Here not really but that can be in other situations (e.g. several individuals data base to merge..).
Note: in `bind_rows`, if columns names do not match, they are fill with `NA`.

Here we want to join the 2 `tibble` according to the common attribute `pid`. Depending on the context you can be interested in joining tables differently. Have a look at the differences between `left_join`, `full_join` and `inner_join`.

```
> ## all individuals from ages are kept
> left_join(age, center, by = c('pid'))
> ## everithing is kept
> full_join(age, center, by = c('pid'))
> ## only the individuals present in both dataset are kept
> inner_join(age, center, by = c('pid'))
```

Can you spot the differences between the commands above?
As an exercise, you can try to compute the individuals' mean age per center.

```
> inner_join(age, center, by = c('pid')) %>%
+   group_by(center) %>%
+   summarise(
+     mean_age = mean(age)
+   )
```

Note: the `by` argument indicates which column should be use to make the join. In some cases, you might have to uses several columns to match (e.g. per sex and age group), this can be easily done specifying a vector of column names.

1.4.10 Data Visualization with ggplot2

One of the package that have contributed to `tidyverse` success is for sure `ggplot2`. We will go more into the details on how to produce graphs with `ggplot2` in another practical. Just a quick example of graphic using `ggplot2`.

Let's Create a bar plot to visualize the number of births by women age group.

First you have to create a table with the number of birth per age group.

```
> birth_per_ageg <- births_tbl %>% group_by(agegrp) %>% summarise(total_births = n())
```

```
> (gg.01 <-
+    ggplot(birth_per_ageg, aes(x = agegrp, y = total_births)) +
+    geom_bar(stat = "identity"))
```

This graph can be customize adding labels and title to the plot:

```
> (gg.02 <-
+    gg.01 +
+    xlab("Women Age Group") +
+    ylab("Total Births") +
+    ggtitle("Number of Births per Women Age Group"))
```

As you can see, plots from `ggplot` family are built sequentially using the `+` operator for each additional element.

1.4.11 String manipulation with stringr

Another popular `tidyverse` popular package is `stringr` package. This package is specialized in the string manipulation. Here are couple of examples.

Let's create a character vector with the following elements representing country names: "Estonia", "Finland", "Denmark", "United Kingdom".

```
> countries <- c("Estonia", "Finland", "Denmark", "United Kingdom")
```

Extract the first three characters from each country name:

```
> country_initials <- str_sub(countries, start = 1, end = 3)
```

Convert all country names to uppercase:

```
> countries_upper <- str_to_upper(countries)
```

Replace "United" with "Utd" in each country name:

```
> countries_modified <- str_replace(countries, "United", "Utd")
```

Find the positions of the letter "n" in each country name:

```
> a_positions <- str_locate_all(countries, "n")
```

As you can see, the output of `str_locate_all` is a list (one element per character string) containing a 2 column table with one line for each match. The first column (start) being the position of the beginning of the match and the second one (end) being the end of the match. In our case, since we are searching for a single character match, this 2 indexes are always the same.

Count the number of characters in each country name:

```
> character_counts <- str_length(countries)
```

These examples demonstrate various string manipulation operations using the `stringr` package. You can modify the exercises, combine several operations or explore other string manipulation functions provided by `stringr` to further practice and enhance your skills in manipulating and analyzing text data.

1.4.12 purrrr package to apply functions to list

Among my favorite `tidyverse` packages, you will find `purrr`. This package contains several functions that are very similar to `lapply` function.

The simplest example could be to apply the same function to a list of elements. As an example let's add "country: " in front of all our

Apply a function to each element of the vector using map(). Here producing the mean of some grades per class:

```
> grades <-
+   list(
+     c1 = c(80, 85, 90),
+     c2 = c(75, 70, 85, 88),
+     c3 = c(90, 85, 95)
+   )
> mean_grades <- map(grades, mean)
```

By default `map()` return a list. One of the nice feature of `purrr` functions is to be able to specify the type of output you want (e.g. `_dbl` for numeric, `_chr` for characters, ...). Check and try to explain the differences between the following command lines:

```
> map(grades, mean)
> map_dbl(grades, mean)
> map_chr(grades, mean)
> map_df(grades, mean)
```

Other nice features of `map` like functions is he availability to support more than one argument. `map2()` for 2 arguments and `pmap()` for more than 2. This can be very handy in some conditions. If you are interested you can have a look to this function help file and play with the examples.

`purrr` package has also a set of functions that can be used to apply iteratively a function using `reduce` and/or `accumulate`. The 2 function behave he same way, it takes the 2 first element of a list, apply a function. The results of the first operation is combine with the third element of the list in the same function and so on.. The only difference is that accumulate return intermediate results while reduce return only the final results.

Here an example of the cumulative product of the 10 first numbers.

```
> 1:10 %>% reduce(`*`)
> 1:10 %>% accumulate(`*`)
```

1.4.13 Bonus: Rendering tables

Once you have produced a nice data-set we can be interested in rendering it in a nice format that can meet presentation/publication expectations. The `kableExtra` table can be useful to achieve this goal.

```
> # if(!require(kableExtra)) install.packages('kableExtra')
> library(kableExtra)
> births.08 <-
+   births_tbl %>%
+   filter(
+     !is.na(gest4)
+   ) %>%
+   group_by(gest4) %>%
+   summarise(
+     N = n()
+   ) %>%
+   mutate(
+     `(%)` = (N / sum(N)) %>% scales::percent()
+   )
> ## default
> births.08
> ## markdown flavor (useful fo automatic report production with knitr)
> # births.08 %>%
> #   knitr::kable(fromat = 'markdown')
>
> ## create an html version of the table and save it on the hard drive
> births.08 %>%
+   kable() %>%
+   kable_styling(
+     bootstrap_options = c("striped", "hover", "condensed", "responsive"),
+     full_width = FALSE
+   ) %>%
+   save_kable(file = 'births.08.html', self_contained = TRUE)
```

1.5 Graphics in R

There are three kinds of plotting functions in R:

1. Functions that generate a new plot, e.g. `hist()` and `plot()`.
2. Functions that add extra things to an existing plot, e.g. `lines()` and `text()`.
3. Functions that allow you to interact with the plot, e.g. `locator()` and `identify()`.

The normal procedure for making a graph in R is to make a fairly simple initial plot and then add on points, lines, text etc., preferably in a script.

1.5.1 Simple plot on the screen

Load the births data and get an overview of the variables:

```
> library( Epi )
> data( births )
> str( births )
```

Now look at the birthweight distribution with

```
> hist(births$bweight)
```

The histogram can be refined – take a look at the possible options with

```
> help(hist)
```

and try some of the options, for example:

```
> hist(births$bweight, col="gray", border="white")
```

To look at the relationship between birthweight and gestational weeks, try

```
> with(births, plot(gestwks, bweight))
```

You can change the plot-symbol by the option `pch=`. If you want to see all the plot symbols try:

```
> plot(1:25, pch=1:25)
```

4. Make a plot of the birth weight versus maternal age with

   ```
   > with(births, plot(matage, bweight) )
   ```

5. Label the axes with

   ```
   > with(births, plot(matage, bweight, xlab="Maternal age", ylab="Birth weight (g)") )
   ```

1.5.2 Colours

There are many colours recognized by R. You can list them all by `colours()` or, equivalently, `colors()` (R allows you to use British or American spelling). To colour the points of birthweight versus gestational weeks, try

```
> with(births, plot(gestwks, bweight, pch=16, col="green") )
```

This creates a solid mass of colour in the centre of the cluster of points and it is no longer possible to see individual points. You can recover this information by overwriting the points with black circles using the `points()` function.

```
> with(births, points(gestwks, bweight, pch=1) )
```

Note: when the number of data points on a scatter plot is large, you may also want to decrease the point size: to get points that are 50% of the original size, add the parameter `cex=0.5` (or another number <1 for different sizes).

1.5.3 Adding to a plot

The `points()` function just used is one of several functions that add elements to an existing plot. By using these functions, you can create quite complex graphs in small steps.

Suppose we wish to recreate the plot of birthweight *vs* gestational weeks using different colours for male and female babies. To start with an empty plot, try

```
> with(births, plot(gestwks, bweight, type="n"))
```

Then add the points with the `points` function.

```
> with(births, points(gestwks[sex==1], bweight[sex==1], col="blue"))
> with(births, points(gestwks[sex==2], bweight[sex==2], col="red"))
```

To add a legend explaining the colours, try

```
> legend("topleft", pch=1, legend=c("Boys","Girls"), col=c("blue","red"))
```

which puts the legend in the top left hand corner.

Finally we can add a title to the plot with

```
> title("Birth weight vs gestational weeks in 500 singleton births")
```

1.5.3.1 Using indexing for plot elements

One of the most powerful features of R is the possibility to index vectors, not only to get subsets of them, but also for repeating their elements in complex sequences.

Putting separate colours on males and female as above would become very clumsy if we had a 5 level factor instead of sex.

Instead of specifying one color for all points, we may specify a vector of colours of the same length as the `gestwks` and `bweight` vectors. This is rather tedious to do directly, but R allows you to specify an expression anywhere, so we can use the fact that `sex` takes the values 1 and 2, as follows:

First create a colour vector with two colours, and take look at `sex`:

```
> c("blue","red")
> births$sex
```

Now see what happens if you index the colour vector by sex:

```
> c("blue","red")[births$sex]
```

For every occurrence of a `1` in `sex` you get `"blue"`, and for every occurrence of `2` you get `"red"`, so the result is a long vector of `"blue"`s and `"red"`s corresponding to the males and females. This can now be used in the plot:

```
> with(births, plot( gestwks, bweight, pch=16, col=c("blue","red")[sex]) )
```

The same trick can be used if we want to have a separate symbol for mothers over 40 say. We first generate the indexing variable:

```
> births$oldmum <- ( births$matage >= 40 ) + 1
```

Note we add 1 because `( matage >= 40 )` generates a logic variable, so by adding 1 we get a numeric variable with values 1 and 2, suitable for indexing:

```
> with(births, plot( gestwks, bweight, pch=c(16,3)[oldmum], col=c("blue","red")[sex] ))
```

so where `oldmum` is 1 we get `pch=16` (a dot) and where `oldmum` is 2 we get `pch=3` (a cross).

R will accept any kind of complexity in the indexing as long as the result is a valid index, so you don't need to create the variable `oldmum`, you can create it on the fly:

```
> with(births, plot( gestwks, bweight, pch=c(16,3)[(matage>=40 )+1], col=c("blue","red")[sex] ),
```

1.5.3.2 Generating colours

R has functions that generate a vector of colours for you. For example,

```
> rainbow(4)
```

produces a vector with 4 colours (not immediately human readable, though). There are a few other functions that generates other sequences of colours, type `?rainbow` to see them. The `color` function (or `colour` function if you prefer) returns a vector of the colour names that R knows about. These names can also be used to specify colours.

Gray-tones are produced by the function `gray` (or `grey`), which takes a numerical argument between 0 and 1; `gray(0)` is black and `gray(1)` is white. Try:

```
> plot( 0:10, pch=16, cex=3, col=gray(0:10/10) )
> points( 0:10, pch=1, cex=3 )
```

1.5.4 Saving your graphs for use in other documents

If you need to use the plot in a report or presentation, you can save it in a graphics file. Once you have generated the script (sequence of R commands) that produce the graph (and it looks ok on screen), you can start a non-interactive graphics device and then re-run the script. Instead of appearing on the screen, the plot will now be written directly to a file. After the plot has been completed you will need to close the device again in order to be able to access the file. Try:

```
> pdf(file="bweight_gwks.pdf", height=4, width=4)
> with(births, plot( gestwks, bweight, col=c("blue","red")[sex]) )
> legend("topleft", pch=1, legend=c("Boys","Girls"), col=c("blue","red"))
> dev.off()
```

This will give you a portable document file `bweight_gwks.pdf` with a graph which is 4 inches tall and 4 inches wide.

Instead of *pdf*, other formats can be used (*jpg, png, tiff, ...*). See `help(Devices)` for the available options.

In window-based environments (R GUI for Windows, R-Studio) you may also use the menu (File→Save as ...or Export) to save the active graph as a file and even copy-paste may work (from R graphics window to Word, for instance) – however, writing it manually into the file is recommended for reproducibility purposes (in case you need to redraw your graph with some modifications).

1.5.5 The `par()` command

It is possible to manipulate any element in a graph, by using the graphics options. These are collected on the help page of `par()`. For example, if you want axis labels always to be horizontal, use the command `par(las=1)`. This will be in effect until a new graphics device is opened.

Look at the typewriter-version of the help-page with

```
> help(par)
```

or better, use the the html-version through Help→Html help→Packages→graphics→P→par.

It is a good idea to take a print of this (having set the text size to "smallest" because it is long) and carry it with you at any time to read in buses, cinema queues, during boring lectures etc. Don't despair, few R-users can understand what all the options are for.

`par()` can also be used to ask about the current plot, for example `par("usr")` will give you the exact extent of the axes in the current plot.

If you want more plots on a single page you can use the command

```
> par( mfrow=c(2,3) )
```

This will give you a layout of 2 rows by 3 columns for the next 6 graphs you produce. The plots will appear by row, i.e. in the top row first. If you want the plots to appear columnwise, use `par( mfcol=c(2,3) )` (you still get 2 rows by 3 columns).

To restore the layout to a single plot per page use

```
> par(mfrow=c(1,1))
```

If you want a more detailed control over the layout of multiple graphs on a single page look at `?layout`.

1.5.6 Interacting with a plot

The `locator()` function allows you to interact with the plot using the mouse. Typing `locator(1)` shifts you to the graphics window and waits for one click of the left mouse button. When you click, it will return the corresponding coordinates.

You can use `locator()` inside other graphics functions to position graphical elements exactly where you want them. Recreate the birth-weight plot,

```
> with(births, plot(gestwks, bweight,  col = c("blue", "red")[sex]) )
```

and then add the legend where you wish it to appear by typing

```
> legend(locator(1), pch=1, legend=c("Boys","Girls"), col=c("blue","red") )
```

The `identify()` function allows you to find out which records in the data correspond to points on the graph. Try

```
> with(births, identify(gestwks, bweight))
```

When you click the left mouse button, a label will appear on the graph identifying the row number of the nearest point in the data frame `births`. If there is no point nearby, R will print a warning message on the console instead. To end the interaction with the graphics window, right click the mouse: the `identify` function returns a vector of identified points.

1. Use `identify()` to find which records correspond to the smallest and largest number of gestational weeks and view the corresponding records:

   ```
   > with(births, births[identify(gestwks, bweight), ])
   ```

1.6 Simple simulation

Monte Carlo methods are computational procedures dealing with simulation of artificial data from given probability distributions with the purpose of learning about the behaviour of phenomena involving random variability. These methods have a wide range of applications in statistics as well as in several branches of science and technology. By solving the following exercises you will learn to use some basic tools of statistical simulation.

1. Whenever using a *random number generator* (RNG) for a simulation study (or for another purpose, such as for producing a randomization list to be used in a clinical trial or for selecting a random sample from a large cohort), it is a good practice to set first the *seed*. It is a number that determines the initial state of the RNG, from which it starts creating the desired sequence of pseudo-random numbers. Explicit specification of the seed enables the reproducibility of the sequence. – Instead of the number 5462319 below you may use your own seed of choice.

   ```
   > set.seed(5462319)
   ```

2. Generate a random sample of size 20 from a normal distribution with mean 100 and standard deviation 10. Draw a histogram of the sampled values and compute the conventional summary statistics

   ```
   >  x <- rnorm(20, 100, 10)
   >  hist(x)
   >  c(mean(x), sd(x))
   ```

   Repeat the above lines and compare the results.

3. Now replace the sample size 20 by 1000 and run again twice the previous command lines with this size but keeping the parameter values as before. Compare the results between the two samples here as well as with those in the previous item.

4. Generate 500 observations from a Bernoulli(p) distribution, or Bin$(1, p)$ distribution, taking values 1 and 0 with probabilities p and $1 - p$, respectively, when $p = 0.4$:

   ```
   > X <- rbinom(500, 1, 0.4)
   > table(X)
   ```

5. Now generate another 0/1 variable Y, being dependent on previously generated X, so that $P(Y = 1|X = 1) = 0.2$ and $P(Y = 1|X = 0) = 0.1$.

   ```
   > Y <- rbinom(500,1,0.1*X+0.1)
   > table(X,Y)
   > prop.table(table(X,Y),1)
   ```

6. Generate data obeying a simple linear regression model $y_i = 5 + 0.1x_i + \varepsilon_i$, $i = 1, \ldots 100$, in which $\varepsilon_i \sim N(0, 10^2)$, and x_i values are integers from 1 to 100. Plot the (x_i, y_i)-values, and estimate the parameters of that model.

```
> x <- 1:100
> y <- 5 + 0.1*x + rnorm(100,0,10)
> plot(x,y)
> abline(lm(y~x))
> summary(lm(y~x))$coef
```

Are your estimates consistent with the data-generating model? Run the code a couple of times to see the variability in the parameter estimates.

1.7 Analysis of hazard rates, their ratios and differences

This exercise is *very* prescriptive, so you should make an effort to really understand everything you type into R. Consult the relevant slides of the lecture on "Poisson regression for rates ..."

1.7.1 Hand calculations for a single rate

Let λ be the true **hazard rate** or theoretical incidence rate of a given outcome event. Its estimator is the empirical **incidence rate** $\widehat{\lambda} = D/Y$ = no. cases/person-years. Recall that the standard error of the empirical rate is $\mathrm{SE}(\widehat{\lambda}) = \widehat{\lambda}/\sqrt{D}$.

The simplest approximate 95% confidence interval (CI) for λ is given by

$$\widehat{\lambda} \pm 1.96 \times \mathrm{SE}(\widehat{\lambda})$$

An alternative approach is based on logarithmic transformation of the empirical rate. The standard error of the empirical log-rate $\widehat{\theta} = \log(\widehat{\lambda})$ is $\mathrm{SE}(\widehat{\theta}) = 1/\sqrt{D}$. Thus, a simple approximate 95% confidence interval for the log-hazard $\theta = \log(\lambda)$ is obtained from

$$\widehat{\theta} \pm 1.96/\sqrt{D} = \log(\widehat{\lambda}) \pm 1.96/\sqrt{D}$$

When taking the exponential from the above limits, we get another approximate confidence interval for the hazard λ itself:

$$\exp\{\log(\widehat{\lambda}) \pm 1.96/\sqrt{D}\} = \widehat{\lambda} \stackrel{\times}{\div} \mathrm{EF},$$

where $\mathrm{EF} = \exp\{1.96 \times \mathrm{SE}[\log(\widehat{\lambda})]\}$ is the *error factor* associated with the 95% interval. This approach provides a more accurate approximation with small numbers of cases. (However, both these methods fail when $D = 0$, in which case an *exact* method or one based on *profile-likelihood* is needed.)

1. Suppose 15 outcome events are observed during 5532 person-years in a given study cohort. Let's use R as a simple desk calculator to estimate the underlying hazard rate λ (in 1000 person-years; therefore 5.532) and to get the first version of an approximate confidence interval:

```
> library( Epi )
> options(digits=4)  #  to cut down decimal points in the output


> D <- 15
> Y <- 5.532    # thousands of years!
> rate <- D / Y
> SE.rate <- rate/sqrt(D)
> c(rate, SE.rate, rate + c(-1.96, 1.96)*SE.rate )
```

2. Compute now the approximate confidence interval using the method based on log-transformation and compare the result with that in the previous item.

```
> SE.logr <- 1/sqrt(D)
> EF <- exp( 1.96 * SE.logr )
> c(log(rate), SE.logr)
> c( rate, EF, rate/EF, rate*EF )
```

1.7.2 Poisson model for a single rate with logarithmic link

You are able to estimate the hazard rate λ and compute its CI with a **Poisson regression model**, as described in the relevant slides in the lecture handout.

Poisson regression is a **generalized linear model** in which the **family**, *i.e.* the distribution of the response variable, is assumed to be the Poisson distribution. The most commonly applied **link function** in Poisson regression is the natural logarithm; log for short.

There are several ways of fitting a Poisson regression with these specifications in R.

3. The traditional way, described many textbooks and applied in most software, is based on using the number of events D as the response and the logarithm of person-years Y as an **offset** term.

   – Fit now to the given data a simple Poisson model containing only the intercept term, save the results into a model object `m`, and print a summary of the results.

   ```
   > m <- glm( D ~ 1, family=poisson(link=log), offset=log(Y) )
   > summary( m )
   ```

   Compare the figures below "`Coefficients`" to those you obtained from your hand calculations for `log(rate)` in section 1.6.2, item 2. – What did you get?

4. The summary method produces too much output for our purposes. You can extract CIs for the model parameters directly from the fitted model on the scale determined by the link function with function `ci.lin()`. Thus, the estimate, its SE, and confidence limits for the log-rate $\theta = \log(\lambda)$ are obtained by:

   ```
   > ci.lin( m )
   ```

   However, to get the point estimate and the confidence limits for the hazard rate $\lambda = \exp(\theta)$ itself on the original scale, the results must be exp-transformed:

   ```
   > ci.lin( m, Exp=TRUE)
   ```

   To get just the point estimate and CI for λ from log-transformed quantities you are recommended to use function `ci.exp()`, which is actually a wrapper of `ci.lin()`:

   ```
   > ci.exp( m)
   > ci.lin( m, Exp=TRUE)[, 5:7]
   ```

   Both functions are found from `Epi` package. – Note that the test statistic and P-value are rarely interesting quantities for a single rate.

5. There is another method for fitting Poisson regression, which overcomes some of the limitations in the traditional method. A family object `poisreg`, a modified version of the original `poisson` family object, is available in package `Epi`. When using this, the response is defined as a *matrix* of two columns: numbers of cases D and person-years Y, these being combined into a matrix by `cbind(D,Y)`. No specification of `offset` is needed.

   ```
   > mreg <- glm( cbind(D, Y) ~ 1, family=poisreg(link=log) )
   > ci.exp( mreg )
   ```

   In this course we endorse the use of family `poisreg` because of its advantages in more general settings.

1.7.3 Poisson model for a single rate with identity link

The approach leaning on having the number of cases D as the response and $\log(Y)$ as an offset, is limited only to models with log link. A major advantage of the `poisreg` family is that it allows a straighforward use of the *identity* link, too. With this link the response variable is the same, but the parameters to be directly estimated are now the rates itself and their differences, not the log-rates and their differences as with the log link.

6. Fit a Poisson model with identity link to our simple data, and use `ci.lin()` to produce the estimate and the confidence interval for the hazard rate from this model:

```
> mid <- glm( cbind(D,Y) ~ 1, family=poisreg(link=identity) )
> ci.lin( mid )
> ci.lin( mid )[, c(1,5,6)]
```

   How is the coefficient of this model interpreted? Verify that you actually get the same rate estimate and CI as in section 1.6.1, item 1.

1.7.4 Poisson model assuming the same rate for several periods

Now, suppose the events and person years are collected over three distinct periods.

7. Read in the data and compute period-specific rates

```
> Dx <- c(3,7,5)
> Yx <- c(1.412,2.783,1.337)
> Px <- 1:3
> rates <- Dx/Yx
> rates
```

8. Using these data, fit the same model with log link as in section 1.6.2, assuming a common single hazard λ for the separate periods. Compare the result from the previous ones

```
> m3 <- glm( cbind(Dx,Yx) ~ 1, family=poisreg(link=log) )
> ci.exp( m3 )
```

9. Now test whether the rates are the same in the three periods: Try to fit a model with the period as a factor in the model:

```
> mp <- glm( cbind(Dx,Yx) ~ factor(Px), family=poisreg(link=log) )
> ci.exp(mp)
```

   Compare the goodness-of-fit of the two models using `anova()` with the argument `test="Chisq"`:

```
> anova( m3, mp, test="Chisq" )
```

   Compare the test statistic to the deviance of the model `mp`. – What is the deviance indicating?

1.7.5 Analysis of rate ratio

We now switch to comparison of two rates λ_1 and λ_0, i.e. the hazard in an exposed group vs. that in an unexposed one.

Consider first estimation of the **hazard ratio** or the underlying "true" rate ratio $\rho = \lambda_1/\lambda_0$ between the groups. Suppose we have pertinent empirical data (cases and person-times) from both groups, (D_1, Y_1) and (D_0, Y_0). The point estimate of ρ is the empirical **incidence rate ratio**

$$\widehat{\rho} = \mathrm{RR} = \frac{\widehat{\lambda}_1}{\widehat{\lambda}_0} = \frac{D_1/Y_1}{D_0/Y_0}$$

The variance of $\log(\mathrm{RR})$, that is, the difference of the log of the empirical rates, $\log(\widehat{\lambda}_1) - \log(\widehat{\lambda}_0)$, is commonly estimated as

$$\begin{aligned} \mathrm{var}(\log(\mathrm{RR})) &= \mathrm{var}\{\log(\widehat{\lambda}_1/\widehat{\lambda}_0)\} = \mathrm{var}\{\log(\widehat{\lambda}_1)\} + \mathrm{var}\{\log(\widehat{\lambda}_0)\} \\ &= 1/D_1 + 1/D_0 \end{aligned}$$

Based on a similar argument as before, an approximate 95% CI for the true rate ratio λ_1/λ_0 is then:

$$\mathrm{RR} \stackrel{\times}{\div} \exp\left(1.96\sqrt{\frac{1}{D_1} + \frac{1}{D_0}}\right)$$

Suppose you have 15 events during 5532 person-years in an unexposed group and 28 events during 4783 person-years in an exposed group:

10. Calculate the incidence rates in the two groups, their ratio, and the CI of the true hazard ratio ρ by direct application of the above formulae:

```
> D0 <- 15   ; D1 <- 28
> Y0 <- 5.532 ; Y1 <- 4.783
> R1 <- D1/Y1; R0 <- D0/Y0
> RR <- R1/R0
> SE.lrr <- sqrt(1/D0+1/D1)
> EF <- exp( 1.96 * SE.lrr)
> c( R1, R0, RR, RR/EF, RR*EF )
```

11. Now achieve this using a Poisson model. For that we first combine the group-specific numbers into pertinent vectors and specify a factor to represent the contrast between the exposed and the unexposed group

```
> D <- c(D0,D1) ; Y <- c(Y0,Y1); expos <- 0:1
> mm <- glm( cbind(D,Y) ~ factor(expos), family=poisreg(link=log) )
```

    What do the parameters mean in this model?

12. You can extract the estimation results for exponentiated parameters in two ways, as before:

```
> ci.exp( mm )
> ci.lin( mm, Exp=TRUE ) [,5:7]
```

1.7.6 Analysis of rate difference

For the **hazard difference** $\delta = \lambda_1 - \lambda_0$, the natural estimator is the **incidence rate difference**

$$\widehat{\delta} = \widehat{\lambda}_1 - \widehat{\lambda}_0 = D_1/Y_1 - D_0/Y_0 = \mathrm{RD}.$$

Its variance is just the sum of the variances of the two rates

$$\begin{aligned} \mathrm{var}(\mathrm{RD}) &= \mathrm{var}(\widehat{\lambda}_1) + \mathrm{var}(\widehat{\lambda}_0) \\ &= D_1/Y_1^2 + D_0/Y_0^2 \end{aligned}$$

13. Use this formula to compute the point estimate of the rate difference λ and a 95% confidence interval for it:

```
> RD <- diff( D/Y )    ##  or RD <- R1 - R0
> SED <- sqrt( sum( D/Y^2 ) )
> c( R1, R0, RD, SED, RD+c(-1,1)*1.96*SED )
```

14. Verify that this is the confidence interval you get when you fit an additive model (obtained by identity link) with exposure as a factor:

```
> ma <- glm( cbind(D,Y) ~ factor(expos),
+            family=poisreg(link=identity) )
> ci.lin( ma )[, c(1,5,6)]
```

1.7.7 Optional/Homework: Identity link with weighting

Do this only after you have done the other exercises of this session, i.e. those in section 1.7.

15. There is yet another way of fitting Poisson regression models in R. It is based on using the original family `poisson` but having the empirical rate $\widehat{\lambda} = D/Y$ as a **scaled** Poisson response, and the person-years Y as a **weight** variable instead of an offset term. – When applying this procedure, it will give you a warning about a non-integer response in a Poisson model, but you can ignore this warning

```
> mwei <- glm( D/Y ~ 1, family=poisson(link=log), weight=Y )
> ci.exp( mwei )
```

Verify that this gave the same results as above.

16. Repeat the model fitting parts in sections 1.6.5, and 1.6.6. using this approach based on weighting, in which the response is `D/Y`, the family is `poisson`, and `weight = Y`. Verify that you got similar results.

1.7.8 Optional/Homework: Calculations using matrix tools

NB. This subsection requires some familiarity with matrix algebra. Do this only after you have done the other exercises of this session.

17. Explore the function `ci.mat()`, which lets you use matrix multiplication (operator `'%*%'` in R) to produce a confidence interval from an estimate and its standard error (or CIs from whole columns of estimates and SEs):

```
> ci.mat
> ci.mat()
```

As you see, this function returns a 2×3 matrix (2 rows, 3 columns) containing familiar numbers.

18. When you combine the single rate and its standard error into a row vector of length 2, i.e. a 1×2 matrix, and multiply this by the 2×3 matrix above, the computation returns a 1×3 matrix containing the point estimate and the confidence limit. – Apply this method to the single rate calculations in 1.6.1; first creating the 1×2 matrix and then performing the matrix multiplication.

```
> rateandSE <- c( rate, SE.rate )
> rateandSE
> rateandSE %*% ci.mat()
```

19. When the confidence interval is based on the log-rate and its standard error, the result is obtained by appropriate application of the exp-function on the pertinent matrix product

```
> lograndSE <- c( log(rate), SE.logr )
> lograndSE
> exp( lograndSE %*% ci.mat() )
```

20. For computing the rate ratio and its CI as in 1.6.5, matrix multiplication with `ci.mat()` should give the same result as there:

```
> exp( c( log(RR), SE.lrr ) %*% ci.mat() )
```

21. The main argument in function `ci.mat()` is `alpha`, which sets the confidence level: $1 - \alpha$. The default value is `alpha = 0.05`, corresponding to the level $1 - 0.05 = 95$ %. If you wish to get the confidence interval for the rate ratio at the 90 % level ($= 1 - 0.1$), for instance, you may proceed as follows:

```
> ci.mat( alpha=0.1 )
> exp( c( log(RR), SE.lrr ) %*% ci.mat(alpha=0.1) )
```

22. Look again to the model used to analyse the rate ratio in 1.7.5. Often one would like to get simultaneously both the rates and the ratio between them. This can be achieved in one go using the *contrast matrix* argument `ctr.mat` to `ci.lin()` or `ci.exp()`. Try:

```
> CM <- rbind( c(1,0), c(1,1), c(0,1) )
> rownames( CM ) <- c("rate 0","rate 1","RR 1 vs. 0")
> CM
> mm <- glm( D ~ factor(expos),
+            family=poisson(link=log),  offset=log(Y) )
> ci.exp( mm, ctr.mat=CM )
```

23. Use the same machinery to the additive model to get the rates and the rate-difference in one go. Note that the annotation of the resulting estimates are via the column-names of the contrast matrix.

```
> rownames( CM ) <- c("rate 0","rate 1","RD 1 vs. 0")
> ma <- glm( cbind(D,Y) ~ factor(expos),
+                  family=poisreg(link=identity) )
> ci.lin( ma, ctr.mat=CM )[, c(1,5,6)]
```

1.8 Logistic regression (GLM)

1.8.1 Malignant melanoma in Denmark

In the mid-80s a case-control study on risk factors for malignant melanoma was conducted in Denmark (Østerlind et al. The Danish case-control study of cutaneous malignant melanoma I: Importance of host factors. *Int J Cancer* 1988; 42: 200-206).

The cases were patients with skin melanoma (excluding lentigo melanoma), newly diagnosed from 1 Oct, 1982 to 31 March, 1985, aged 20-79, from East Denmark, and they were identified from the Danish Cancer Registry.

The controls (twice as many as cases) were drawn from the residents of East Denmark in April, 1984, as a random sample stratified by sex and age (within the same 5 year age group) to reflect the sex and age distribution of the cases. This is called group matching, and in such a study, it is necessary to control for age and sex in the statistical analysis. (Yes indeed: In spite of the fact that stratified sampling by sex and age removed the statistical association of these variables with melanoma from the final case-control data set, the analysis must control for variables which determine the probability of selecting subjects from the base population to the study sample.)

The population of East Denmark is a dynamic one. Sampling the controls only at one time point is a rough approximation of *incidence density sampling*, which ideally would spread out over the whole study period. Hence the exposure odds ratios calculable from the data are estimates of the corresponding hazard rate ratios between the exposure groups.

After exclusions, refusals etc., 474 cases (92% of eligible cases) and 926 controls (82%) were interviewed. This was done face-to-face with a structured questionnaire by trained interviewers, who were not informed about the subject's case-control status.

For this exercise we have selected a few host variables from the study in an ascii-file, `melanoma.dat`. The variables are listed in table 1.1.

Table 1.1: *Variables in the melanoma dataset.*

Variable	**Units or Coding**	**Type**	**Name**
Case-control status	1=case, 0=control	numeric	`cc`
Sex	1=male, 2=female	numeric	`sex`
Age at interview	age in years	numeric	`age`
Skin complexion	0=dark, 1=medium, 2=light	numeric	`skin`
Hair colour	0=dark brown/black, 1=light brown, 2=blonde, 3=red	numeric	`hair`
eye colour	0=brown, 1=grey, green, 2=blue	numeric	`eyes`
Freckles	1=many, 2=some, 3=none	numeric	`freckles`
Naevi, small	no. naevi < 5mm	numeric	`nvsmall`
Naevi, largs	no. naevi ≥ 5mm	numeric	`nvlarge`

1.8.2 Reading the data

Start R and load the `Epi` package using the function `library()`. Read the data set from the file `melanoma.dat` found in the course website to a data frame with name `mel` using the `read.table()` function. Remember to specify that missing values are coded ".", and that variable names are in the first line of the file. View the overall structure of the data frame, and list the first 20 rows of `mel`.

```
> library(Epi)
> mel <- read.table("http://bendixcarstensen.com/SPE/data/melanoma.dat", header=TRUE, na.string
> str(mel)
> head(mel, n=20)
```

1.8.3 House keeping

The structure of the data frame `mel` tells us that all the variables are numeric (integer), so first you need to do a bit of house keeping. For example the variables `sex, skin, hair, eye` need to be converted to factors, with labels, and `freckles` which is coded 4 for none down to 1 for many (not very intuitive) needs to be recoded, and relabelled.

To avoid too much typing and to leave plenty of time to think about the analysis, these house keeping commands are in a script file called `melanoma-house.r`. You should study this script carefully before running it. The coding of `freckles` can be reversed by subtracting the current codes from 4. Once recoded the variable needs to be converted to a factor with labels "none", etc. Age is currently a numeric variable recording age to the nearest year, and it will be convenient to group these values into (say) 10 year age groups, using `cut`. In this case we choose to create a new variable, rather than change the original.

```
> source("http://bendixcarstensen.com/SPE/data/melanoma-house.r")
```

Look again at the structure of the data frame `mel` and note the changes. Use the command `summary(mel)` to look at the univariate distributions.

This is enough housekeeping for now - let's turn to something a bit more interesting.

1.8.4 One variable at a time

As a first step it is a good idea to start by looking at the numbers of cases and controls by each variable separately, ignoring age and sex. Try

```
> with(mel, table(cc,skin))
> stat.table(skin, contents=ratio(cc,1-cc), data=mel)
```

to see the numbers of cases and controls, as well as the odds of being a case by skin colour

Now use `effx()` to get crude estimates of the hazard ratios for the effect of skin colour.

```
> effx(cc, type="binary", exposure=skin, data=mel)
```

- Look at the crude effect estimates of `hair`, `eyes` and `freckles` in the same way.

1.8.5 Generalized linear models with binomial family and logit link

The function `effx()` is just a wrapper for the `glm()` function, and you can show this by fitting the glm directly with

```
> mf <- glm(cc ~ freckles, family="binomial", data=mel)
> round(ci.exp( mf ),2)
>
```

Comparison with the output from `effx()` shows the results to be the same.

Note that in `effx()` the type of response is "binary" whereas in `glm()` the family of probability distributions used to fit the model is "binomial". There is a 1-1 relationship between type of response and family:

metric	gaussian
binary	binomial
failure/count	poisson

1.8.6 Controlling for age and sex

Because the probability that a control is selected into the study depends on age and sex it is necessary to control for age and sex. For example, the effect of freckles controlled for age and sex is obtained with

```
> effx(cc, typ="binary", exposure=freckles, control=list(age.cat,sex),data=mel)
```

or

```
> mfas <- glm(cc ~ freckles + age.cat + sex, family="binomial", data=mel)
> round(ci.exp(mfas), 2)
```

Do the adjusted estimates differ from the crude ones that you computed with `effx()`?

1.8.7 Likelihood ratio tests

There are 2 effects estimated for the 3 levels of `freckles`, and `glm()` provides a test for each effect separately, but to test for no effect at all of `freckles` you need a likelihood ratio test. This involves fitting two models, one without `freckles` and one with, and recording the change in deviance. Because there are some missing values for freckles it is necessary to restrict the first model to those subjects who have values for freckles.

```
> mas <- glm(cc ~ age.cat + sex, family="binomial", data=subset(mel, !is.na(freckles)) )
> anova(mas, mfas, test="Chisq")
```

The change in residual deviance is $1785.9 - 1737.1 = 48.8$ on $1389 - 1387 = 2$ degrees of freedom. The P-value corresponding to this change is obtained from the upper tail of the cumulative distribution of the χ^2-distribution with 2 df:

```
> 1 - pchisq(48.786, 2)
```

• There are 3 effects for the 4 levels of hair colour (`hair`). To obtain adjusted estimates for the effect of hair colour and to test the pertinent null hypothesis fit the relevant models, print the and use `anova` to test for no effects of hair colour. Compare the estimates with the crude ones and assess the evidence against the null hypothesis.

1.8.8 Relevelling

From the above you can see that subjects at each of the 3 levels light-brown, blonde, and red, are at greater risk than subjects with dark hair, with similar odds ratios. This suggests creating a new variable `hair2` which has just two levels, dark and the other three. The `Relevel()` function in `Epi` has been used for this in the house keeping script.

• Use `effx()` to compute the odds-ratio of melanoma between persons with red, blonde or light brown hair versus those with dark hair. Reproduce these results by fitting an appropriate glm. Use also a likelihood ratio test to test for the effect of `hair2`.

1.8.9 Controlling for other variables

When you control the effect of an exposure for some variable you are asking a question about what would the effect be if the variable is kept constant. For example, consider the effect of `freckles` controlled for `hair2`. We first stratify by `hair2` with

```
> effx(cc, type="binary", exposure=freckles,
+                   control=list(age.cat,sex), strata=hair2, data=mel)
```

The effect of freckles is still apparent in each of the two strata for hair colour. Use `effx()` to control for `hair2`, too, in addition to `age.cat` and `sex`.

```
> effx(cc, type="binary", exposure=freckles,
+                   control=list(age.cat,sex,hair2), data=mel)
```

It is tempting to control for variables without thinking about the question you are thereby asking. This can lead to nonsense.

1.8.10 Stratification using `glm()`

We shall reproduce the output from

```
> effx(cc, type="binary", exposure=freckles,
+             control=list(age.cat,sex), strata=hair2,data=mel)
```

using `glm()`. To do this requires a nested model formula:

```
> mfas.h <- glm(cc ~ hair2/freckles + age.cat + sex, family="binomial", data=mel)
> ci.exp(mfas.h )
```

In amongst all the other effects you can see the two effects of freckles for dark hair (1.61 and 2.84) and the two effects of freckles for other hair (1.42 and 3.15).

1.8.11 Naevi

The distributions of `nvsmall` and `nvlarge` are very skew to the right. You can see this with

```
> with(mel, stem(nvsmall))
> with(mel, stem(nvlarge))
```

Because of this it is wise to categorize them into a few classes

– small naevi into four: 0, 1, 2-4, and 5+;

– large naevi into three: 0, 1, and 2+.

This has been done in the house keeping script.

• Look at the joint frequency distribution of these new variables using `with(mel, table( ))`. Are they strongly associated?

• Compute the sex- and age-adjusted OR estimates (with 95% CIs) associated with the number of small naevi first by using `effx()`, and then by fitting separate glms including `sex`, `age.cat` and `nvsma4` in the model formula.

• Do the same with large naevi `nvlar3`.

• Now fit a glm containing `age.cat, sex, nvsma4` and `nvlar3`. What is the interpretation of the coefficients for `nvsma4` and `nvlar3`?

1.8.12 Treating freckles as a numeric exposure

The evidence for the effect of `freckles` is already convincing. However, to demonstrate how it is done, we shall perform a linear trend test by treating freckles as a numeric exposure.

```
> mel$fscore<-as.numeric(mel$freckles)
> effx(cc, type="binary", exposure=fscore, control=list(age.cat,sex), data=mel)
```

You can check for linearity of the log odds of being a case with `fscore` by comparing the model containing `freckles` as a factor with the model containing `freckles` as numeric.

```
> m1 <- glm(cc ~ freckles + age.cat + sex, family="binomial", data=mel)
> m2 <- glm(cc ~ fscore + age.cat + sex, family="binomial", data=mel)
> anova(m2, m1, test="Chisq")
```

There is no evidence against linearity ($p = 0.22$).

It is sometimes helpful to look at the linearity in more detail with

```
> m1 <- glm(cc ~ C(freckles, contr.cum) + age.cat + sex, family="binomial",data=mel)
> round(ci.exp(m1 ), 2)
> m2 <- glm(cc ~ fscore + age.cat + sex, family="binomial",data=mel)
> round(ci.exp(m2), 2)
```

The use of `C(freckles, contr.cum)` makes each odds ratio to compare the odds at that level versus the previous level; not against the baseline (except for the 2nd level). If the log-odds are linear then these odds ratios should be the same (and the same as the odds ratio for `fscore` in m2.

1.8.13 Graphical displays

The odds ratios (with CIs) can be graphically displayed using function `plotEst()` in `Epi`. It uses the value of `ci.lin()` evaluated on the fitted model object. As the intercept and the effects of age and sex are of no interest, we shall drop the corresponding rows (the 7 first ones) from the matrix produced by `ci.lin()`, and the plot is based just on the 1st, 5th and the 6th column of this matrix:

```
> m <- glm(cc ~ nvsma4 + nvlar3 + age.cat + sex, family="binomial",data=mel)
> plotEst( exp( ci.lin(m)[ 2:5, -(2:4)] ), xlog=T, vref=1 )
```

The `xlog` argument makes the OR axis logarithmic.

1.9 Estimation of effects: simple and more complex

This exercise deals with analysis of metric or continuous response variables. We start with simple estimation of effects of a binary, categorical or a numeric explanatory variable, the exposure variable of interest. Then evaluation of potential modification confounding and/or by other variables is considered by stratification by and adjustment or control for these variables. Use of function `effx()` for such tasks is introduced together with functions `lm()` and `glm()` that can be used for more general linear and generalized linear models. Finally, more complex polynomial models for the effect of a numeric exposure variable are illustrated.

1.9.1 Response and explanatory variables

Identifying the *response* or *outcome variable* correctly is the key to analysis. The main types are:

- Metric (a measurement taking many values, usually with units)
- Binary (two values coded 1/0)
- Failure (does the subject fail at end of follow-up, coded 1/0, and how long was follow-up, measurement of time)
- Count (aggregated data on failures in a group)

All these response variable are numeric.

Variables on which the response may depend are called *explanatory variables*. They can be categorical factors or numeric variables. A further important aspect of explanatory variables is the role they will play in the analysis.

- Primary role: exposure.
- Secondary role: confounder and/or modifier.

The word *effect* is a general term referring to ways of comparing the values of the response variable at different levels of an explanatory variable. The main measures of effect are:

- Differences in means for a metric response.
- Ratios of odds for a binary response.
- Ratios of rates for a failure or count response.

Other measures of effect include ratios of geometric means for positive-valued metric outcomes, differences and ratios between proportions (risk difference and risk ratio), and differences between failure rates.

1.9.2 Data set `births`

We shall use the `births` data to illustrate different aspects in estimating effects of various exposures on a metric response variable `bweight` = birth weight, recorded in grams.

1. Load the `Epi` package and the data set and look at its content

```
> library(Epi)
> data(births)
> str(births)
```

2. Because all variables are numeric we need first to do a little housekeeping. Two of them are directly converted into factors, and categorical versions are created of two continuous variables by function `cut()`.

```
> births$hyp <- factor(births$hyp, labels = c("normal", "hyper"))
> births$sex <- factor(births$sex, labels = c("M", "F"))
> births$agegrp <- cut(births$matage,
+     breaks = c(20, 25, 30, 35, 40, 45), right = FALSE)
> births$gest4 <- cut(births$gestwks,
+     breaks = c(20, 35, 37, 39, 45), right = FALSE)
```

3. Have a look at univariate summaries of the different variables in the data; especially the location and dispersion of the distribution of `bweight`.

```
> summary(births)
> with(births, sd(bweight) )
```

1.9.3 Simple estimation with `effx()`, `lm()` and `glm()`

We are ready to analyze the "effect" of `sex` on `bweight`. A binary exposure variable, like `sex`, leads to an elementary two-group comparison of group means for a metric response.

4. Comparison of two groups is commonly done by the conventional t-test and the associated confidence interval.

```
> with( births, t.test(bweight ~ sex, var.equal=T) )
```

   The P-value refers to the test of the null hypothesis that there is no effect of `sex` on birth weight (quite an uninteresting null hypothesis in itself!). However, `t.test()` does not provide the point estimate for the effect of sex; only the test result and a confidence interval.

5. The function `effx()` in `Epi` is intended to introduce the estimation of effects in epidemiology, together with the related ideas of stratification and controlling, i.e. adjustment for confounding, without the need for familiarity with statistical modelling. It is in fact a wrapper of function `glm()` that fits generalized linear models. – Now, do the same analysis with `effx()`

```
> effx(response=bweight, type="metric", exposure=sex, data=births)
```

The estimated effect of sex on birth weight, measured as a difference in means between girls and boys, is −197 g. Either the output from `t.test()` above or the command

```
> stat.table(sex, mean(bweight), data=births)
```

confirms this ($3032.8 - 3229.9 = -197.1$).

6. The same task can easily be performed by `lm()` or by `glm()`. The main argument in both is the *model formula*, the left hand side being the response variable and the right hand side after ∼ defines the explanatory variables and their joint effects on the response. Here the only explanatory variable is the binary factor `sex`. With `glm()` one specifies the `family`, i.e. the assumed distribution of the response variable, but in case you use `lm()`, this argument is not needed, because `lm()` fits only models for metric responses assuming Gaussian distribution.

   ```
   > m1 <- glm(bweight ~ sex, family=gaussian, data=births)
   > summary(m1)
   ```

   Note the amount of output that `summary()` method produces. The point estimate plus confidence limits can, though, be concisely obtained by `ci.lin()`.

   ```
   > round( ci.lin(m1)[ , c(1,5,6)] , 1)
   ```

7. Now, use `effx()` to find the effect of `hyp` (maternal hypertension) on `bweight`.

1.9.4 Factors on more than two levels

The variable `gest4` became as the result of cutting `gestwks` into 4 groups with left-closed and right-open boundaries [20,35) [35,37) [37,39) [39,45).

8. We shall find the effects of `gest4` on the metric response `bweight`.

   ```
   > effx(response=bweight,typ="metric",exposure=gest4,data=births)
   ```

   There are now 3 effect estimates:

   ```
   [35,37) vs [20,35)  857
   [37,39) vs [20,35) 1360
   [39,45) vs [20,35) 1668
   ```

   The command

   ```
   > stat.table(gest4,mean(bweight),data=births)
   ```

   confirms that the effect of `agegrp` (level 2 vs level 1) is $2590 - 1733 = 857$, etc.

9. Compute these estimates by `lm()` and find out how the coefficients are related to the group means

   ```
   > m2 <- lm(bweight ~ gest4, data = births)
   > round( ci.lin(m2)[ , c(1,5,6)] , 1)
   ```

1.9.5 Stratified effects and interaction or effect modification

We shall now examine whether and to what extent the effect of `hyp` on `bweight` varies by `gest4`.

10. The following "interaction plot" shows how the mean `bweight` depends jointly on `hyp` and `gest4`

```
> par(mfrow=c(1,1))
> with( births, interaction.plot(gest4, hyp, bweight) )
```

    It appears that the mean difference in `bweight` between normotensive and hypertensive mothers is inversely related to gestational age.

11. Let us get numerical values for the mean differences in the different `gest4` categories:

```
> effx(bweight, type="metric", exposure=hyp, strata=gest4,data=births)
```

    The estimated effects of `hyp` in the different strata defined by `gest4` thus range from about -100 g among those with ≥ 39 weeks of gestation to about -700 g among those with < 35 weeks of gestation. The error margin especially around the latter estimate is quite wide, though. The P-value 0.055 from the test for *effect modification* indicates weak evidence against the null hypothesis of "no interaction between `hyp` and `gest4`". On the other hand, this test may well be not very sensitive given the small number of preterm babies in these data.

12. Stratified estimation of effects can also be done by `lm()`, and you should get the same results:

```
> m3 <- lm(bweight ~ gest4/hyp, data = births)
> round( ci.lin(m3)[ , c(1,5,6)], 1)
```

13. An equivalent model with an explicit *interaction term* between `gest4` and `hyp` is fitted as follows

```
> m3I <- lm(bweight ~ gest4 + hyp + gest4:hyp, data = births)
> round( ci.lin(m3I)[ , c(1,5,6)], 1)
```

    From this output you would find a familiar estimate -673 g for those < 35 gestational weeks. The remaining coefficients are estimates of the interaction effects such that e.g. $515 = -158 - (-673)$ g describes the contrast in the effect of `hyp` on `bweight` between those 35 to < 37 weeks and those < 35 weeks of gestation.

14. Perhaps a more appropriate reference level for the categorized gestational age would be the highest one. Changing the reference level, here to be the 4th category, can be done by `Relevel()` function in the `Epi` package, after which an equivalent interaction model is fitted, now using a shorter expression for it in the model formula:

```
> births$gest4b <- Relevel( births$gest4, ref = 4)
> m3Ib <- lm(bweight ~ gest4b*hyp, data = births)
> round( ci.lin(m3Ib)[ , c(1,5,6)], 1)
```

Notice now the coefficient −91.6 for `hyp`. It estimates the effect of `hyp` on `bweight` among those with ≥ 39 weeks of gestation. The estimate −88.5 g = −180.1 − (−91.6) g describes the additional effect of `hyp` in the category 37 to 38 weeks of gestation upon that in the reference class.

15. At this stage it is interesting to compare the results from the interaction models to those from the corresponding *main effects* model, in which the effect of `hyp` is assumed not to be modified by `gest4`:

```
> m3M <- lm(bweight ~ gest4 + hyp, data = births)
> round( ci.lin(m3M)[ , c(1,5,6)], 1)
```

The estimate −201 g describing the overall effect of `hyp` is obtained as a weighted average of the stratum-specific estimates obtained by `effx()` above. It is a meaningful estimate adjusting for `gest4` insofar as it is reasonable to assume that the effect of `hyp` is not modified by `gest4`. This assumption or the "no interaction" null hypothesis can formally be tested by a common deviance test.

```
> anova(m3I, m3M)
```

The P-value is practically the same as before when the interaction was tested in `effx()`. However, in spite of obtaining a "non-significant" result from this test, the possibility of a real interaction should not be ignored in this case.

16. Now, use `effx()` to stratify (i) the effect of `hyp` on `bweight` by `sex` and then (ii) perform the stratified analysis using the two ways of fitting an interaction model with `lm`.

Look at the results. Is there evidence for the effect of `hyp` being modified by `sex`?

1.9.6 Controlling or adjusting for the effect of hyp for sex

The effect of `hyp` is *controlled for* – or *adjusted for* – `sex` by first looking at the estimated effects of `hyp` in the two stata defined by `sex`, and then combining these effects if they seem sufficiently similar. In this case the estimated effects were −496 and −380 which look quite similar (and the P-value against "no interaction" was quite large, too), so we can perhaps combine them, and control for `sex`.

17. The combining is done by declaring `sex` as a control variable:

```
> effx(bweight, type="metric", exposure=hyp, control=sex, data=births)
```

18. The same is done with `lm()` as follows:

```
> m4 <- lm(bweight ~ sex + hyp, data = births)
> ci.lin(m4)[ , c(1,5,6)]
```

The estimated effect of `hyp` on `bweight` controlled for `sex` is thus −448 g. There can be more than one control variable, e.g `control=list(sex,agegrp)`.

Many people go straight ahead and control for variables which are likely to confound the effect of exposure without bothering to stratify first, but usually it is useful to stratify first.

1.9.7 Numeric exposures

If we wished to study the effect of gestation time on the baby's birth weight then `gestwks` is a numeric exposure.

19. Assuming that the relationship of the response with `gestwks` is roughly linear (for a metric response) we can estimate the linear effect of `gestwks`, both with `effx()` and with `lm()` as follows:

```
> effx(response=bweight, type="metric", exposure=gestwks,data=births)
> m5 <- lm(bweight ~ gestwks, data=births) ; ci.lin(m5)[ , c(1,5,6)]
```

We have fitted a simple linear regression model and obtained estimates of the two regression coefficient: `intercept` and `slope`. The linear effect of `gestwks` is thus estimated by the slope coefficient, which is 197 g per each additional week of gestation.

20. You cannot stratify by a numeric variable, but you can study the effects of a numeric exposure stratified by (say) `agegrp` with

```
> effx(bweight, type="metric", exposure=gestwks, strata=agegrp, data=births)
```

You can control/adjust for a numeric variable by putting it in the control list.

1.9.8 Checking the assumptions of the linear model

At this stage it will be best to make some visual check concerning our model assumptions using `plot()`. In particular, when the main argument for the *generic function* `plot()` is a fitted `lm` object, it will provide you some common diagnostic graphs.

21. To check whether `bweight` goes up linearly with `gestwks` try

```
> with(births, plot(gestwks,bweight))
> abline(m5)
```

22. Moreover, take a look at the basic diagnostic plots for the fitted model.

```
> par(mfrow=c(2,2))
> plot(m5)
```

What can you say about the agreement with data of the assumptions of the simple linear regression model, like linearity of the systematic dependence, homoskedasticity and normality of the error terms?

1.9.9 Third degree polynomial of `gestwks`

A common practice to assess possible deviations from linearity is to compare the fit of the simple model with models having higher order polynomial terms. In perinatal epidemiology a popular model for describing the relationship between gestational age and birth weight is a 3rd degree polynomial.

23. For fitting a third degree polynomial of `gestwks` we can update our previous simple linear model by adding the quadratic and cubic terms of `gestwks` using the *insulate* operator `I()`

```
> m6 <- update(m5, . ~ .  + I(gestwks^2) + I(gestwks^3))
> round(ci.lin(m6)[, c(1,5,6)], 1)
```

The intercept and linear coefficients are really spectacular – but don't make any sense!

24. A more elegant way of fitting polynomial models is to utilize *orthogonal polynomials*, which are linear transformations of the original polynomial terms such that they are mutually uncorrelated. However, they are scaled in such a way that the estimated regression coefficients are also difficult to interpret, apart from the intercept term.

25. As function `poly()` creating orthogonal polynomials does not accepet missing values, we shall only include babies whose value of `gestwks` is not missing. Let us also perform an F test for the null hypothesis of simple linear effect against the 3rd degree polynomial model

```
> births2 <- subset(births, !is.na(gestwks))
> m.ortpoly <- lm(bweight ~ poly(gestwks, 3), data= births2 )
> round(ci.lin(m.ortpoly)[, c(1,5,6)], 1)
> anova(m5, m.ortpoly)
```

Note that the estimated intercept 3138 g has the same value as the mean birth weight among all those babies who are included, i.e. whose gestational age was known.

There seems to be strong evidence against simple linear regression; addition of the quadratic and the cubic term appears to have reduced the residual sum of squares "highly significantly".

26. Irrespective of whether the polynomial terms were orthogonalized or not, the fitted or predicted values for the response variable remain the same. As the next step we shall present graphically the fitted polynomial curve together with 95 % confidence limits for the expected responses as well as 95 % prediction intervals for individual observations in new data comprising gestational weeks from 24 to 45 in steps of 0.25 weeks.

```
> nd <- data.frame(gestwks = seq(24, 45, by = 0.25) )
> fit.poly <- predict( m.ortpoly, newdata=nd, interval="conf" )
> pred.poly <- predict( m.ortpoly, newdata=nd, interval="pred" )
> par(mfrow=c(1,1))
> with( births, plot( bweight ~ gestwks, xlim = c(23, 46), cex.axis= 1.5, cex.lab = 1.5 )
> matlines( nd$gestwks, fit.poly, lty=1, lwd=c(3,2,2), col=c('red','blue','blue') )
> matlines( nd$gestwks, pred.poly, lty=1, lwd=c(3,2,2), col=c('red','green','green') )
```

The fitted curve fits nicely within the range of observed values of the regressor. However, the tail behaviour in polynomial models tends to be problematic.

We shall continue the analysis in the next practical, in which the apparently curved effect of `gestwks` is modelled by a *penalized spline*. Also, key details in fitting linear regression models and spline models are covered in the lecture of this afternoon.

1.9.10 Extra (if you have time): Frequency data

Data from very large studies are often summarized in the form of frequency data, which records the frequency of all possible combinations of values of the variables in the study. Such data are sometimes presented in the form of a contingency table, sometimes as a data frame in which one variable is the frequency. As an example, consider the `UCBAdmissions` data, which is one of the standard R data sets, and refers to the outcome of applications to 6 departments in the graduate school at Berkeley by gender.

27. Let us have a look at the data

    ```
    > UCBAdmissions
    ```

    You can see that the data are in the form of a $2 \times 2 \times 6$ contingency table for the three variables `Admit` (admitted/rejected), `Gender` (male/female), and `Dept` (A/B/C/D/E/F). Thus in department A 512 males were admitted while 312 were rejected, and so on. The question of interest is whether there is any bias against admitting female applicants.

28. The next command coerces the contingency table to a data frame, and shows the first 10 lines.

    ```
    > ucb <- as.data.frame(UCBAdmissions)
    > head(ucb)
    ```

    The relationship between the contingency table and the data frame should be clear.

29. Let us turn `Admit` into a numeric variable coded 1 for rejection, 0 for admission

    ```
    > ucb$Admit <- as.numeric(ucb$Admit)-1
    ```

    The effect of `Gender` on `Admit` is crudely estimated by

    ```
    > effx(Admit,type="binary",exposure=Gender,weights=Freq,data=ucb)
    ```

    The odds of rejection for female applicants thus appear to be 1.84 times the odds for males (note the use of `weights` to take account of the frequencies). A crude analysis therefore suggests there is a strong bias against admitting females.

30. Continue the analysis by stratifying the crude analysis by department - does this still support a bias against females? What is the effect of gender controlled for department?

1.10 Estimation and reporting of curved effects

This exercise deals with modelling of curved effects of continuous explanatory variables both on a metric response assuming the Gaussian distribution and on a count or rate outcome based on the Poisson family.

In the first part we continue our analysis on gestational age on birth weight focussing on fitting spline models, both unpenalized and a penalized one.

In the second part we analyse the `testisDK` data found in the `Epi` package. It contains the numbers of cases of testis cancer and mid-year populations (person-years) in 1-year age groups in Denmark during 1943–96. In this analysis we apply Poisson regression on the incidence rates treating age and calendar time, first as categorical but then fitting a penalized spline model.

1.10.1 Data `births`: Simple linear regression and 3rd degree polynomial

Recall what was done in items 17 to 24 of the Exercise on simple estimation of effects, in which a simple linear regression and a 3rd degree polynomial were fitted. The main results are also shown on slides 6, 8, 9, and 20 of the lecture on linear models.

1. Make a basic scatter plot and draw the fitted line from a simple linear regression on it.

```
> library(Epi)
> data(births)
> par(mfrow=c(1,1))
> with(births, plot(gestwks, bweight))
> mlin <- lm( bweight ~ gestwks, data = births )
> abline( mlin )
```

2. Repeat also the diagnostic plots of this simple model

```
> par( mfrow=c(2,2) )
> plot( mlin )
```

   Some deviation from the linear model is apparent.

1.10.2 Fitting a natural cubic spline

A popular approach for flexible modelling is based on natural regression splines, which have more reasonable tail behaviour than polynomial regression.

3. By the following piece of code you can fit a *natural cubic spline* with 5 pre-specified knots to be put at 28, 34, 38, 40 and 43 weeks of gestation, respectively, determining the degree of smoothing.

```
> library(splines)
> mNs5 <- lm( bweight ~ Ns( gestwks, knots = c(28,34,38,40,43)),
+             data = births )
> round( ci.exp( mNs5, Exp=FALSE ), 1)
```

These regression coefficients are even less interpretable than those in the polynomial model.

4. A graphical presentation of the fitted curve together with the confidence and prediction intervals is more informative. As this kind of presentation will be repeated, let us write a short function script to facilitate the task. We utilize function `matshade()` in `Epi`, which creates shaded areas, and function `matlines()` which draws lines joining the pertinent end points over the x-values for which the predictions are computed.

```
> plotFitPredInt <- function( xval, fit, pred, ...)
+ {
+     matshade( xval, fit,  lwd=2, alpha=0.2)
+     matshade( xval, pred, lwd=2, alpha=0.2)
+     matlines( xval, fit,  lty=1, lwd=c(3,2,2), col=c("red","blue","blue") )
+     matlines( xval, pred, lty=1, lwd=c(3,2,2), col=c("red","green","green") )
+ }
```

5. Now, create a vector of x-values and compute the fitted/predicted values as well as the interval limits at these points from the fitted model object utilizing function `predict()`. This function creates a matrix of three columns: (1) fitted/predicted values, (2) lower limits, (3) upper limits

```
> nd <- data.frame(gestwks = seq(24, 45, by = 0.25) )
> fit.Ns5 <- predict( mNs5, newdata=nd, interval="conf" )
> pred.Ns5 <- predict( mNs5, newdata=nd, interval="pred" )
> par(mfrow=c(1,1))
> with( births, plot(bweight ~ gestwks, xlim=c(23, 46),
+                    cex.axis=1.5, cex.lab=1.5 ) )
> plotFitPredInt(xval=nd$gestwks, fit=fit.Ns5, pred=pred.Ns5)
```

Compare this with the 3rd order curve previously fitted (see slide 20 of the lecture). In a natural spline the curve is constrained to be linear beyond the extreme knots.

6. Take a look at the basic diagnostic plots from the spline model.

```
> par(mfrow=c(2,2))
> plot(mNs5)
```

How would you interpret these plots?

The choice of the number of knots and their locations can be quite arbitrary, and the results are often sensitive to these choices.

7. To illustrate arbitrariness and associated problems with specification of knots, you may now fit another natural spline model like the one above but now with 10 knots at the following sequence of points: `seq(25, 43, by = 2)`. Display graphically the results

The behaviour of the curve is really wild for small values of `gestwks`!

1.10.3 Penalized spline model

One way to go around the arbitrariness in the specification of knots is to fit a *penalized spline* model, which imposes a "roughness penalty" on the curve. Even though a big number of knots are initially allowed, the resulting fitted curve will be optimally smooth.

You cannot fit a penalized spline model with `lm()` or `glm()`, Instead, function `gam()` in package `mgcv` can be used for this purpose.

8. You must first load package `mgcv`.

9. When calling `gam()`, the model formula contains expression '`s(X)`' for any explanatory variable `X`, for which you wish to fit a smooth function

```
> library(mgcv)
> mPs <- gam( bweight ~ s(gestwks), data = births)
> summary(mPs)
```

From the output given by `summary()` you find that the estimated intercept is here, too, equal to the overall mean birth weight in the data. The estimated residual variance is given by "`Scale est.`" or from subobject `sig2` of the fitted `gam` object. Taking square root you will obtain the estimated residual standard deviation: 445.2 g.

```
> mPs$sig2
> sqrt(mPs$sig2)
```

The degrees of freedom in this model are not computed as simply as in previous models, and they typically are not integer-valued. However, the fitted spline seems to consume only a little more degrees of freedom as the 3rd degree polynomial above.

10. As in previous models we shall plot the fitted curve together with 95 % confidence intervals for the mean responses and 95 % prediction intervals for individual responses. Obtaining the prediction intervals from the fitted `gam` object requires a bit more work than with `lm` objects.

```
> pr.Ps <- predict( mPs, newdata=nd, se.fit=TRUE )
> str(pr.Ps) # with se.fit=TRUE, only two columns: fitted value and its SE
> fit.Ps <- cbind(pr.Ps$fit,
+                 pr.Ps$fit - 2*pr.Ps$se.fit,
+                 pr.Ps$fit + 2*pr.Ps$se.fit)
> pred.Ps <- cbind(pr.Ps$fit,  # must add residual variance to se.fit^2
+                  pr.Ps$fit - 2*sqrt( pr.Ps$se.fit^2 + mPs$sig2),
+                  pr.Ps$fit + 2*sqrt( pr.Ps$se.fit^2 + mPs$sig2))
> par(mfrow=c(1,1))
> with(births, plot(bweight ~ gestwks, xlim=c(24, 45),
+                   cex.axis=1.5, cex.lab=1.5) )
> plotFitPredInt(nd$gestwks, fit.Ps, pred.Ps)
```

The fitted curve is indeed clearly more reasonable than the polynomial.

1.10.4 Testis cancer: Data input and housekeeping

We shall now switch to analyzing the incidence of testis cancer in Denmark during 1943–1998 by age and calendar time or period.

11. Load the data and inspect its structure:

```
>  library( Epi )
>    data( testisDK )
>     str( testisDK )
> summary( testisDK )
>    head( testisDK )
```

12. There are nearly 5000 observations from 90 one-year age groups and 54 calendar years. To get a clearer picture of what's going one we do some housekeeping. The age range will be limited to 15–79 years, and age and period are both categorised into 5-year intervals – according to the time-honoured practice in epidemiology.

```
> tdk <- subset(testisDK, A > 14 & A < 80)
> tdk$Age <- cut(tdk$A, br = 5*(3:16), include.lowest=TRUE, right=FALSE)
>    nAge <- length(levels(tdk$Age))
> tdk$Per <- cut(tdk$P, br = seq(1943,1998,by=5),
+                include.lowest=TRUE, right=FALSE)
>    nPer <- length(levels(tdk$Per))
```

1.10.5 Some descriptive analysis

Computation and tabulation of incidence rates

13. Tabulate numbers of cases and person-years, and compute the incidence rates (per 100,000 y) in each 5 y × 5 y cell using `stat.table()`

```
> tab <- stat.table(  index = list(Age, Per),
+                   contents = list(D = sum(D),
+                                   Y = sum(Y/1000),
+                                rate = ratio(D, Y, 10^5) ),
+                    margins = TRUE,
+                       data = tdk )
> print(tab, digits=c(sum=0, ratio=1))
```

Look at the incidence rates in the column margin and in the row margin. In which age group is the marginal age-specific rate highest? Do the period-specific marginal rates have any trend over time?

14. From the saved table object `tab` you can plot an age-incidence curve for each period separately, after you have checked the structure of the table, so that you know the relevant dimensions in it. There is a function `rateplot()` in `Epi` that does default plotting of tables of rates (see the help page of `rateplot`)

```
> str(tab)
> par(mfrow=c(1,1))
> rateplot( rates=tab[3, 1:nAge, 1:nPer], which="ap", ylim=c(1,30),
+           age=seq(15, 75, 5), per=seq(1943, 1993, 5),
+           col=heat.colors(16), ann=TRUE )
```

Is there any common pattern in the age-incidence curves across the periods?

1.10.6 Age and period as categorical factors

We shall first fit a Poisson regression model with log link on age and period model in the traditional way, in which both factors are treated as categorical. The model is additive on the log-rate scale. It is useful to scale the person-years to be expressed in 10^5 y.

15. ```
    > tdk$Y <- tdk$Y/100000
    > mCat <- glm( cbind(D,Y) ~ Age + Per,
    +              family=poisreg(link=log), data= tdk )
    > round( ci.exp( mCat ), 2)
    ```

    What do the estimated rate ratios tell about the age and period effects?

16. A graphical inspection of point estimates and confidence intervals can be obtained as follows. In the beginning it is useful to define shorthands for the pertinent mid-age and mid-period values of the different intervals

    ```
    > aMid <- seq(17.5, 77.5, by = 5)
    > pMid <- seq(1945, 1995, by = 5)
    > par(mfrow=c(1,2))
    > plot( c(15,80), c(0.6, 6), type="n", log="y",
    +       cex.lab = 1.5, cex.axis = 1.5,
    +    xlab = "Age (years)", ylab = "Rate ratio")
    > lines( aMid,  c( 1, ci.exp(mCat)[2:13, 1] ), type = "o", pch = 16 )
    > segments( aMid[-1],  ci.exp(mCat)[2:13, 2],
    +           aMid[-1], ci.exp(mCat)[2:13, 3] )
    > plot( c(1943,1998), c(0.6, 6), type="n", log="y",
    +       cex.lab = 1.5, cex.axis = 1.5,
    +    xlab = "Calendar year - 1900", ylab = "Rate ratio")
    > lines( pMid,c( 1, ci.exp(mCat)[14:23, 1] ), type = 'o', pch = 16 )
    > segments( pMid[-1],  ci.exp(mCat)[14:23, 2],
    +           pMid[-1],  ci.exp(mCat)[14:23, 3] )
    ```

17. In the fitted model the reference category for each factor was the first one. As age is the dominating factor, it may be more informative to remove the intercept from the model. As a consequence the age effects describe fitted rates at the reference level of the period factor. For the latter one could choose the middle period 1968-72.

    ```
    > tdk$Per70 <- Relevel(tdk$Per, ref = 6)
    > mCat2 <- glm( cbind(D,Y) ~ -1 + Age +Per70,
    +               family=poisreg(link=log), data= tdk )
    > round( ci.exp( mCat2 ), 2)
    ```

    We shall plot just the point estimates from the latter model

    ```
    > par(mfrow=c(1,2))
    > plot( c(15,80), c(2, 20), type="n", log="y", cex.lab = 1.5, cex.axis = 1.5,
    +    xlab = "Age (years)", ylab = "Incidence rate (per 100000 y)")
    > lines( aMid,  c(ci.exp(mCat2)[1:13, 1] ), type = "o", pch = 16)
    > plot( c(1943,1998), c(0.4, 2), type="n", log="y",
    +       cex.lab = 1.5, cex.axis = 1.5,
    +    xlab = "Calendar year", ylab = "Rate ratio")
    > lines( pMid, c(ci.exp(mCat2)[14:18, 1], 1, ci.exp(mCat2)[19:23, 1]),
    +    type = "o", pch = 16 )
    > abline(h=1, col="gray")
    ```
    ```

1.10.7 Generalized additive model with penalized splines

It is obvious that the age effect on the log-rate scale is highly non-linear. Yet, it is less clear whether the true period effect deviates from linearity. Nevertheless, there are good indications to try fitting smooth continuous functions for both.

18. As the next task we fit a generalized additive model for the log-rate on continuous age and period applying penalized splines with default settings of function `gam()` in package `mgcv`. In this fitting an "optimal" value for the penalty parameter is chosen based on an AIC-like criterion known as UBRE.

```
> library(mgcv)
> mPen <- gam( cbind(D, Y) ~ s(A) + s(P),
+             family = poisreg(link=log), data = tdk)
> summary(mPen)
```

The summary is quite brief, and the only estimated coefficient is the intercept, which sets the baseline level for the log-rates, against which the relative age effects and period effects will be contrasted. On the rate scale the baseline level 5.53 per 100000 y is obtained by `exp(1.7096)`.

19. See also the default plot for the fitted curves (solid lines) describing the age and the period effects which are interpreted as contrasts to the baseline level on the log-rate scale.

```
> par(mfrow=c(1,2))
> plot(mPen, seWithMean=TRUE)
```

The dashed lines describe the 95 % confidence band for the pertinent curve. One could get the impression that year 1968 would be some kind of reference value for the period effect, like period 1968-72 chosen as the reference in the categorical model previously fitted. This is not the case, however, because `gam()` by default parametrizes the spline effects such that the reference level, at which the spline effect is nominally zero, is the overall "grand mean" value of the log-rate in the data. This corresponds to the principle of *sum contrasts* (`contr.sum`) for categorical explanatory factors.

From the summary you will also find that the degrees of freedom value required for the age effect is nearly the same as the default dimension $k - 1 = 9$ of the part of the model matrix (or basis) initially allocated for each smooth function. (Here k refers to the relevant argument that determines the basis dimension when specifying a smooth term by `s()` in the model formula). On the other hand the period effect takes just about 3 df.

20. It is a good idea to do some diagnostic checking of the fitted model

```
> par(mfrow=c(2,2))
> gam.check(mPen)
```

The four diagnostic plots are analogous to some of those used in the context of linear models for Gaussian responses, but not all of them may be as easy to interpret. – Pay attention to the note given in the printed output about the value of `k`.

21. Let us refit the model but now with an increased `k` for age:

```
> mPen2 <- gam( cbind(D,Y) ~ s(A, k=20) + s(P),
+             family = poisreg(link=log), data = tdk)
> summary(mPen2)
> par(mfrow=c(2,2))
> gam.check(mPen2)
```

With this choice of `k` the df value for age became about 11, which is well below $k - 1 = 19$. Let us plot the fitted curves from this fitting, too

```
> par( mfrow=c(1,2) )
> plot( mPen2, seWithMean=TRUE )
> abline( v=1968, h=0, lty=3 )
```

There does not seem to have happened any essential changes from the previously fitted curves, so maybe 8 df could, after all, be quite enough for the age effect.

22. Graphical presentation of the effects using `plot.gam()` can be improved. For instance, we may present the age effect to describe the "mean" incidence rates by age, averaged over the whole time span of 54 years. This is obtained by adding the estimated intercept to the estimated smooth curve for the age effect and showing the antilogarithms of the ordinates of the curve. For that purpose we need to extract the intercept and modify the labels of the y-axis accordingly. The estimated period curve can also be expressed in terms of relative indidence rates in relation to the fitted baseline rate, as determined by the model intercept.

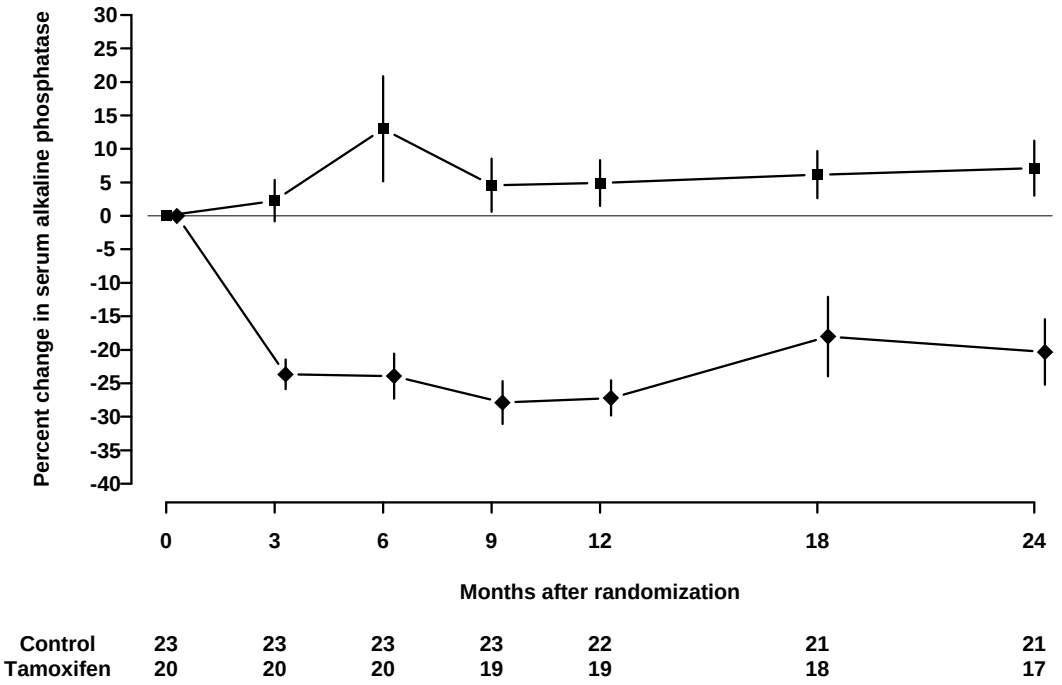
```
> par( mfrow=c(1,2) )
> icpt <- coef(mPen2)[1]  #  estimated intecept
> plot( mPen2, seWithMean=TRUE, select=1, rug=FALSE,
+       yaxt="n", ylim= c(log(1),log(20)) - icpt,
+       xlab="Age (y)", ylab="Mean rate (/100000 y)" )
> axis(2, at = log( c(1, 2, 5, 10, 20)) - icpt, labels=c(1, 2, 5, 10, 20) )
> plot( mPen2, seWithMean=TRUE, select=2, rug=FALSE,
+       yaxt="n", ylim=c( log(0.4), log(2) ),
+       xlab="Calendat year", ylab="Relative rate")
> axis(2, at=log( c(0.5, 0.75, 1, 1.5, 2)), labels = c(0.5, 0.75, 1, 1.5, 2))
> abline( v=1968, h=0, lty=3 )
```

One could continue the analysis of these data by fitting an age-cohort model as an alternative to the age-period model, as well as an age-cohort-period model.

There will also be some extras added in the solution part of this exercise.

1.11 Graphics meccano

The plot below is from a randomized study of the effect of Tamoxifen treatment on bone mineral metabolism, in a group of patients who were treated for breast cancer.



The data are available in the file `alkfos.csv` (using comma as separator, so `read.csv` will read it).

```
> alkfos <- read.csv("./data/alkfos.csv") # change filename as needed
```

The purpose of this exercise is to show you how to build a similar graph using base graphics in R. This will take you through a number of fundamental techniques. The exercise will also walk you through creating the graph using `ggplot2`.

To get started, run the code in the housekeeping script `alkfos-house.r` . You probably should not study the code in too much detail at this point. The script create the following objects in your workspace.

- `times`, a vector of length 7 giving the observation times
- `means`, a 2×7 matrix giving the mean percentage change at each time point. Each group has its own row.
- `sems`, a 2×7 matrix giving standard errors of the means, used to create the error bars.
- `available`, a 2×7 matrixing giving the number of participants still available

Use the `objects()` to see the objects created function to see them.

1.11.1 Base graphics

Now we start building the plot. It is important that you use some form of script to hold the R code since you will frequently have to modify and rerun previously entered code.

1. First, plot the means for group 1 (i.e. `means[1,]`) against `times`, using `type="b"` (look up what this does)

2. Then *add* a similar curve for group 2 to the plot using `points` or `lines`. Notice that the new points are below the y scale of the plot, so you need to revise the initial plot by setting a suitable `ylim` value.

3. Add the error bars using `segments`. (You can calculate the endpoints using `upper <- means + sems` etc.). You may have to adjust the `ylim` again.

4. Add the horizontal line at $y = 0$ using `abline`

5. Use `xlab` and `ylab` in the initial `plot` call to give better axis labels.

6. We need a nonstandard x axis. Use `xaxt="n"` to avoid plotting it at first, then add a custom axis with `axis`

7. The counts below the x axis can be added using `mtext` on lines 5 and 6 below the bottom of the plot, but you need to make room for the extra lines. Use `par(mar=.1 + c(8,4,4,2))` *before* plotting anything to achieve this.

You now have quite a good reconstruction of the original plot. There are some additional steps you can take to reproduce the published plot exactly. These are advanced exercises so feel free to come back to them later.

8. It is not too important here (it was for some other variables in the study), but the S-PLUS plot has the points for the second group offset horizontally by a small amount (.25) to prevent overlap. Redo the plot with this modification.

9. Further things to fiddle with: Get rid of the bounding box. Add Control/Tamoxifen labels to the lines of counts. Perhaps use different plotting symbols. Rotate the y axis values. Modify the line widths or line styles.

10. Finally, try plotting to the `pdf()` device and view the results using a PDF viewer (*e.g.* Adobe Acrobat Reader). You may need to change the `pointsize` option and/or the plot dimensions for optimal appearance. You might also try saving the plot as a metafile and include it in a Word document.

1.11.2 Using ggplot2

The housekeeping script `alkfos-house.r` also creates a data frame `ggdata` containing the variables in long format. The code for generating the data frame is shown below, but you do not need to repeat it if you have run the script.

```
> ggdata <- data.frame(
+     times = rep(times, 2),
+     means = c(means[1,], means[2,]),
+     sds = c(sds[1,], sds[2,]),
+     available = c(available[1,], available[2,]),
+     treat = rep(c("placebo","tamoxifen"), each=7)
+     )
> ggdata <- transform(ggdata, sems = sds/sqrt(available))
```

To create a first approximation to the plot in `ggplot2` we use the `qplot` function (short for "quick plot"). First you must install the `ggplot2` package from CRAN and then load it:

```
> library(ggplot2)
> qplot(x=times, y=means, group=treat, geom=c("point", "line") , data=ggdata)
```

The first arguments to `qplot` are called "aesthetics" in the grammar of graphics. Here we want to plot `y=means` by `x=times` grouped by `group=treat`. The aesthetics are used by the "geometries", which are specified with the `geom` argument. Here we want to plot both points and lines. The `data` argument tells `qplot` to get the aesthetics from the data frame `ggdata`.

To add the error bars, we add a new geometry "linerange" which uses the aesthetics `ymin` and `ymax`

```
> p <- qplot(x=times, y=means, group=treat,
+            ymin=means-sems, ymax=means+sems,
+            yintercept=0, geom=c("point", "line", "linerange"),
+            data=ggdata)
> print(p)
```

In this case we are saving the output of `qplot` to an R object `p`. This means the plot will not appear automatically when we call `qplot`. Instead, we must explicitly print it.

Note how the y axes are automatically adjusted to include the error bars. This is because they are included in the call to `qplot` and not added later (as was the case with base graphics).

It remains to give informative axis labels and put the right tick marks on the x-axis. This is done by adding scales to the plot

```
> p <- p +
+     scale_x_continuous(name="Months after randomization",
+                        breaks=ggdata$times) +
+     scale_y_continuous(name="% change in alkaline phosphatase")
> print(p)
```

We can also change the look and feel of the plot by adding a theme (in this case the black and white theme).

```
> p + theme_bw()
```

As an alternative to `qplot`, we can use the `ggplot` function to define the data and the common aesthetics, then add the geometries with separate function calls. All the grobs (*graphical objects*) created by these function calls are combined with the `+` operator:

```
> p <- ggplot(data=ggdata,
+             aes(x=times, y=means, ymin=means-sems, ymax=means+sems,
+                 group=treat)) +
+     geom_point() +
+     geom_line() +
```

```
+     geom_linerange() +
+     geom_hline(yintercept=0, colour="darkgrey") +
+     scale_x_continuous(breaks=ggdata$times) +
+     scale_y_continuous(breaks=seq(-35,25,5))
> print(p)
```

This call adds another geometry "hline" which uses the aesthetic `yintercept` to add a horizontal line at 0 on the y-axis. Note that this alternate syntax allows each geometry to have its own aesthetics: here we draw the horizontal line in darkgrey instead of the default black.

1.11.3 Grid graphics

As a final, advanced topic, this subsection shows how viewports from the `grid` package may be used to display the plot and the table in the same graph. First we create a text table:

```
> tab <- ggplot(data=ggdata, aes(x=times, y=treat, label=available)) +
+     geom_text(size=3) + xlab(NULL) + ylab(NULL) +
+     scale_x_continuous(breaks=NULL)
> tab
```

Then we create a layout that will contain the graph above the table. Most of the space is taken by the graph. The `grid.show.layout` allows you to preview the layout.

```
> library(grid)
> Layout <- grid.layout(nrow = 2, ncol = 1, heights = unit(c(2, 0.25),
+                                                  c("null", "null")))
> grid.show.layout(Layout)
```

The units are relative ("null") units. You can specify exact sizes in centimetres, inches, or lines if you prefer.

We then print the graph and the table in the appropriate viewports

```
> grid.newpage() #Clear the page
> pushViewport(viewport(layout=Layout))
> print(p, vp=viewport(layout.pos.row=1, layout.pos.col=1))
> print(tab, vp=viewport(layout.pos.row=2, layout.pos.col=1))
```

Notice that the left margins do not match. One way to get the margins to match is to use the `plot_grid` function from the cowplot package.

```
> library(cowplot)
> plot_grid(p, tab, align="v", ncol=1, nrow=2, rel_heights=c(5,1))
```

The cowplot package has a theme that is useful for publications. It is a black and white theme with no grid:

```
> theme_set(theme_cowplot())
> plot_grid(p, tab, align="v", ncol=1, nrow=2, rel_heights=c(5,1))
```

The `theme_set` function changes the default ggplot2 theme so that all subsequent displays will use the given theme.

1.12 Survival analysis: Oral cancer patients

1.12.1 Description of the data

File `oralca2.txt`, that you may access from a url address to be given in the practical, contains data from 338 patients having an oral squamous cell carcinoma diagnosed and treated in one tertiary level oncological clinic in Finland since 1985, followed-up for mortality until 31 December 2008. The dataset contains the following variables:

`sex`	=	sex, a factor with categories `1 = "Female", 2 = "Male"`,
`age`	=	age (years) at the date of diagnosing the cancer,
`stage`	=	TNM stage of the tumour (factor): `1 = "I", ..., 4 = "IV", 5 = "unkn"`,
`time`	=	follow-up time (in years) since diagnosis until death or censoring,
`event`	=	event ending the follow-up (numeric): 0 = censoring alive, 1 = death from oral cancer, 2 = death from other causes.

1.12.2 Loading the packages and the data

11. Load the R packages `Epi, mstate`, and `survival` needed in this exercise.

```
> library(Epi)
> library(popEpi)
> library(data.table)
> library(knitr)
> library(survival)
```

12. Read the datafile `oralca2.txt` from a website, whose precise address will be given in the practical, into an R data frame named `orca`. Look at the head, structure and the summary of the data frame. Using function `table()` count the numbers of censorings as well as deaths from oral cancer and other causes, respectively, from the `event` variable.

```
> orca <- read.table("pracs/data/oralca2.txt", header=T)
> head(orca) ; str(orca) ; summary(orca)
```

1.12.3 Total mortality: Kaplan–Meier analyses

1. We start our analysis of total mortality pooling the two causes of death into a single outcome. First, construct a *survival object* `orca$suob` from the event variable and the follow-up time using function `Surv()`. Look at the structure and summary of `orca$suob` .

```
> orca$suob <- Surv(orca$time, 1*(orca$event > 0) )
> str(orca$suob)
> summary(orca$suob)
```

2. Create a `survfit` object `s.all`, which does the default calculations for a Kaplan–Meier analysis of the overall (marginal) survival curve.

```
> s.all <- survfit(suob ~ 1, data=orca)
```

See the structure of this object and apply `print()` method on it, too. Look at the results; what do you find?

```
> s.all
> str(s.all)
```

3. The `summary` method for a `survfit` object would return a lengthy life table. However, the `plot` method with default arguments offers the Kaplan–Meier curve for a conventional illustration of the survival experience in the whole patient group. Alternatively, instead of graphing survival proportions, one can draw a curve describing their complements: the cumulative mortality proportions. This curve is drawn together with the survival curve as the result of the second command line below.

```
> plot(s.all)
> lines(s.all, fun = "event", mark.time=F, conf.int=F)
```

The effect of option `mark.time=F` is to omit marking the times when censorings occurred.

1.12.4 Total mortality by stage

Tumour stage is an important prognostic factor in cancer survival studies.

1. Plot separate cumulative mortality curves for the different stage groups marking them with different colours, the order which you may define yourself. Also find the median survival time for each stage.

```
> s.stg  <- survfit(suob ~ stage, data= orca)
> col5 <- c("green", "blue", "black", "red", "gray")
> plot(s.stg, col= col5, fun="event", mark.time=F )
> s.stg
```

2. Create now two parallel plots of which the first one describes the cumulative hazards and the second one graphs the log-cumulative hazards against log-time for the different stages. Compare the two presentations with each other and with the one in the previous item.

```
> par(mfrow=c(1,2))
> plot(s.stg, col= col5, fun="cumhaz", main="cum. hazards" )
> plot(s.stg, col= col5, fun="cloglog", main = "cloglog: log cum.haz"  )
```

3. If the survival times were *exponentially* distributed in a given (sub)population the corresponding cloglog-curve should follow an approximately linear pattern. Could this be the case here in the different stages?

4. Also, if the survival distributions of the different subpopulations would obey the *proportional hazards* model, the vertical distance between the cloglog-curves should be approximately constant over the time axis. Do these curves indicate serious deviation from the proportional hazards assumption?

5. In the lecture handouts (p. 34, 37) it was observed that the crude contrast between males and females in total mortality appears unclear, but the age-adjustment in the Cox model provided a more expected hazard ratio estimate. We shall examine the confounding by age somewhat closer. First categorize the continuous age variable into, say, three categories by function `cut()` using suitable breakpoints, like 55 and 75 years, and cross-tabulate sex and age group:

```
> orca$agegr <- cut(orca$age, br=c(0,55,75, 95))
> stat.table( list( sex, agegr), list( count(), percent(agegr) ),
+              margins=T, data = orca )
```

Male patients are clearly younger than females in these data.

Now, plot Kaplan–Meier curves jointly classified by sex and age.

```
> s.agrx <- survfit(suob ~ agegr + sex, data=orca)
> par(mfrow=c(1,1))
> plot(s.agrx, fun="event", mark.time=F, xlim = c(0,15),
+              col=rep(c("red", "blue"),3), lty=c(2,2, 1,1, 5,5))
```

In each ageband the mortality curve for males is on a higher level than that for females.

1.12.5 Event-specific cumulative mortality curves

We move on to analysing cumulative mortalities for the two causes of death separately, first overall and then by prognostic factors.

1. Use the `survfit`-function in `survival` package with option `type="mstate"`.

```
> library(survival)
> cif1 <- survfit( Surv( time, event, type="mstate") ~ 1,
+                  data = orca)
> str(cif1)
```

2. One could apply here the plot method of the survfit object to plot the cumulative incidences for each cause. However, we suggest that you use instead a simple function `plotCIF()` found in the `Epi` package. The main arguments are

$$\texttt{data} = \text{data frame created by function } \texttt{survfit()}, \tag{1.1}$$
$$\texttt{event} = \text{indicator for the event: values 1 or 2.} \tag{1.2}$$

Other arguments are like in the ordinary `plot()` function.

3. Draw two parallel plots describing the overall cumulative incidence curves for both causes of death

```
> par(mfrow=c(1,2))
> plotCIF(cif1, 1, main = "Cancer death")
> plotCIF(cif1, 2, main= "Other deaths")
```

4. Compute the estimated cumulative incidences by stage for both causes of death. Now you have to add variable `stage` to survfit-function.

   See the structure of the resulting object, in which you should observe strata variable containing the stage grouping variable. Plot the pertinent curves in two parallel graphs. Cut the y-axis for a more efficient graphical presentation

```
> col5 <- c("green", "blue", "black", "red", "gray")
> cif2 <- survfit( Surv( time, event, type="mstate") ~ stage,
+                  data = orca)
> str(cif2)
> par(mfrow=c(1,2))
> plotCIF(cif2, 1, main = "Cancer death by stage",
+         col = col5, ylim = c(0, 0.7) )
> plotCIF(cif2, 2, main= "Other deaths by stage",
+         col=col5, ylim = c(0, 0.7) )
```

   Compare the two plots. What would you conclude about the effect of stage on the two causes of death?

5. Using another function `stackedCIF()` in `Epi` you can put the two cumulative incidence curves in one graph but stacked upon one another such that the lower curve is for the cancer deaths and the upper curve is for total mortality, and the vertical difference between the two curves describes the cumulative mortality from other causes. You can also add some colours for the different zones:

```
> par(mfrow=c(1,1))
> stackedCIF(cif1, colour = c("gray70", "gray85"))
```

1.12.6 Regression modelling of overall mortality.

1. Fit the semiparametric proportional hazards regression model, a.k.a. the Cox model, on all deaths including sex, age and stage as covariates. Use function `coxph()` in package `survival`. It is often useful to center and scale continuous covariates like `age` here. The estimated rate ratios and their confidence intervals can also here be displayed by applying `ci.lin()` on the fitted model object.

```
> options(show.signif.stars = F)
> m1 <- coxph(suob ~ sex + I((age-65)/10) + stage, data= orca)
> summary( m1 )
> round( ci.exp(m1 ), 4 )
```

   Look at the results. What are the main findings?

2. Check whether the data are sufficiently consistent with the assumption of proportional hazards with respect to each of the variables separately as well as globally, using the `cox.zph()` function.

```
> cox.zph(m1)
```

3. No evidence against proportionality assumption could apparently be found. Moreover, no difference can be observed between stages I and II in the estimates. On the other hand, the group with stage unknown is a complex mixture of patients from various true stages. Therefore, it may be prudent to exclude these subjects from the data and to pool the first two stage groups into one. After that fit a model in the reduced data with the new stage variable.

```
> orca2 <- subset(orca, stage != "unkn")
> orca2$st3 <- Relevel( orca2$stage, list(1:2, 3, 4:5) )
> levels(orca2$st3) = c("I-II", "III", "IV")
> m2 <- update(m1, . ~ . - stage + st3, data=orca2 )
> round( ci.exp(m2 ), 4)
```

4. Plot the predicted cumulative mortality curves by stage, jointly stratified by sex and age, focusing only on 40 and 80 year old patients, respectively, based on the fitted model `m2`. You need to create a new artificial data frame containing the desired values for the covariates.

```
> newd <- data.frame( sex = c( rep("Male", 6), rep("Female", 6) ),
+                     age = rep( c( rep(40, 3), rep(80, 3) ), 2 ),
+                     st3 = rep( levels(orca2$st3), 4) )
> newd
> col3 <- c("green", "black", "red")
> par(mfrow=c(1,2))
> plot( survfit(m2, newdata= subset(newd, sex=="Male" & age==40)),
+      col=col3, fun="event", mark.time=F)
> lines( survfit(m2, newdata= subset(newd, sex=="Female" & age==40)),
+       col= col3, fun="event", lty = 2, mark.time=F)
> plot( survfit(m2, newdata= subset(newd, sex=="Male" & age==80)),
+      ylim = c(0,1), col= col3, fun="event", mark.time=F)
> lines( survfit(m2, newdata= subset(newd, sex=="Female" & age==80)),
+       col=col3, fun="event", lty=2, mark.time=F)
>
```

1.12.7 Modelling event-specific hazards and hazards of the subdistribution

1. Fit the Cox model for the cause-specific hazard of cancer deaths with the same covariates as above. In this case only cancer deaths are counted as events and deaths from other causes are included into censorings.

```
> m2haz1 <- coxph( Surv( time, event==1)  ~ sex + I((age-65)/10) + st3 , data=orca2 )
> round( ci.exp(m2haz1 ), 4)
> cox.zph(m2haz1)
```

   Compare the results with those of model `m2`. What are the major differences?

2. Fit a similar model for deaths from other causes and compare the results.

```
> m2haz2 <- coxph( Surv( time, event==2)  ~ sex + I((age-65)/10) + st3 , data=orca2 )
> round( ci.exp(m2haz2 ), 4)
> cox.zph(m2haz2)
```

3. Finally, fit the Fine–Gray model for the hazard of the subdistribution for cancer deaths with the same covariates as above. For this you have to first load package `cmprsk`, containing the necessary function `crr()`, and attach the data frame.

```
> library(cmprsk)
> attach(orca2)
> m2fg1 <- crr(time, event, cov1 = model.matrix(m2), failcode=1)
> summary(m2fg1, Exp=T)
```

   Compare the results with those of model `m2` and `m2haz1`.

4. Fit a similar model for deaths from other causes and compare the results.

```
> m2fg2 <- crr(time, event, cov1 = model.matrix(m2), failcode=2)
> summary(m2fg2, Exp=T)
```

1.12.8 Lexis object with multi-state set-up

Before entering to analyses of cause-specific mortality it might be instructive to apply some Lexis tools to illustrate the competing-risks set-up. More detailed explanation of these tools will be given by Bendix in this afternoon.

1. Form a `Lexis` object from the data frame and print a summary of it. We shall name the main (and only) time axis in this object as `stime`.

```
> orca.lex <- Lexis(exit = list(stime = time),
+            exit.status = factor(event,
+    labels = c("Alive", "Oral ca. death", "Other death")),
+                   data = orca)
> summary(orca.lex)
```

2. Draw a box diagram of the two-state set-up of competing transitions. Run first the following command line

```
boxes( orca.lex )
```

   Now, move the cursor to the point in the graphics window, at which you wish to put the box for "Alive", and click. Next, move the cursor to the point at which you wish to have the box for "Oral ca. death", and click. Finally, do the same with the box for "Other death". If you are not happy with the outcome, run the command line again and repeat the necessary mouse moves and clicks.

1.12.9 Poisson regression as an alternative to Cox model

It can be shown that the Cox model with an unspecified form for the baseline hazard $\lambda_0(t)$ is mathematically equivalent to the following kind of Poisson regression model. Time is treated as a categorical factor with a dense division of the time axis into disjoint intervals or *timebands* such that only one outcome event occurs in each timeband. The model formula contains this time factor plus the desired explanatory terms.

A sufficient division of time axis is obtained by first setting the break points between adjacent timebands to be those time points at which an outcome event has been observed to occur. Then, the pertinent `lexis` object is created and after that it will be split according to those breakpoints. Finally, the Poisson regression model is fitted on the splitted `lexis` object using function `glm()` with appropriate specifications.

We shall now demonstrate the numerical equivalence of the Cox model `m2haz1` for oral cancer mortality that was fitted above, and the corresponding Poisson regression.

1. First we form the necessary `lexis` object by just taking the relevant subset of the already available `orca.lex` object. Upon that the three-level stage factor `st3` is created as above.

```
> orca2.lex <- subset(orca.lex, stage != "unkn" )
> orca2.lex$st3 <- Relevel( orca2$stage, list(1:2, 3, 4:5) )
> levels(orca2.lex$st3) = c("I-II", "III", "IV")
```

   Then, the break points of time axis are taken from the sorted event times, and the `lexis` object is split by those breakpoints. The `timeband` factor is defined according to the splitted survival times stored in variable `stime`.

```
> cuts <- sort(orca2$time[orca2$event==1])
> orca2.spl <- splitLexis( orca2.lex, br = cuts, time.scale="stime" )
> orca2.spl$timeband <- as.factor(orca2.spl$stime)
```

   As a result we now have an expanded `lexis` object in which each subject has several rows; as many rows as there are such timebands during which he/she is still at risk. The outcome status `lex.Xst` has value 0 in all those timebands, over which the subject stays alive, but assumes the value 1 or 2 at his/her last interval ending at the time of death. – See now the structure of the splitted object.

```
> str(orca2.spl)
> orca2.spl[ 1:20, ]
```

2. We are ready to fit the desired Poisson model for oral cancer death as the outcome. The splitted person-years are contained in `lex.dur`, and the explanatory variables are the same as in model `m2haz1`. – This fitting may take some time ...

```
> m2pois1 <- glm( 1*(lex.Xst=="Oral ca. death")  ~
+       -1 + timeband + sex + I((age-65)/10) + st3,
+       family=poisson, offset = log(lex.dur), data = orca2.spl)
```

   We shall display the estimation results graphically for the baseline hazard (per 1000 person-years) and numerically for the rate ratios associated with the covariates. Before doing that it is useful to count the length `ntb` of the block occupied by baseline hazard in the whole vector of estimated parameters. However, owing to how the splitting to timebands was done, the last regression coefficient is necessarily zero and better be omitted when displaying the results. Also, as each timeband is quantitatively named accoding to its leftmost point, it is good to compute the midpoint values `tbmid` for the timebands

```
> tb <- as.numeric(levels(orca2.spl$timeband)) ; ntb <- length(tb)
> tbmid <- (tb[-ntb] + tb[-1])/2   # midpoints of the intervals
> round( ci.exp(m2pois1 ), 3)
> par(mfrow=c(1,1))
> plot( tbmid, 1000*exp(coef(m2pois1)[1:(ntb-1)]),
+             ylim=c(5,3000), log = "xy", type = "l")
```

Compare the regression coefficients and their error margins to those model `m2haz1`. Do you find any differences? How does the estimated baseline hazard look like?

3. The estimated baseline looks quite ragged when based on 71 separate parameters. A smoothed estimate may be obtained by spline modelling using the tools contained in package `splines` (see the practical of Saturday 25 May afternoon). With the following code you will be able to fit a reasonable spline model for the baseline hazard and draw the estimated curve (together with a band of the 95% confidence limits about the fitted values). From the same model you should also obtain quite familiar results for the rate ratios of interest.

```
> library(splines)
> m2pspli <- update(m2pois1, . ~  ns(stime, df = 6, intercept = F) +
+         sex + I((age-65)/10) + st3)
> round( ci.exp( m2pspli ), 3)
> news <- data.frame( stime = seq(0,25, length=301), lex.dur = 1000, sex = "Female",
+                     age = 65, st3 = "I-II")
> blhaz <- predict(m2pspli, newdata = news, se.fit = T, type = "link")
> blh95 <- cbind(blhaz$fit, blhaz$se.fit) %*% ci.mat()
> par(mfrow=c(1,1))
> matplot( news$stime, exp(blh95), type = "l", lty = c(1,1,1), lwd = c(2,1,1) ,
+       col = rep("black", 3),  log = "xy", ylim = c(5,3000)  )
```

1.13 Time-splitting, time-scales and SMR

This exercise is about mortaity among Danish Diabetes patients. It is based on the dataset `DMlate`, a random sample of 10,000 patients from the Danish Diabetes Register (scrambeled dates), all with date of diagnosis after 1994.

1. First load the data and take a look at the data:

```
library( Epi)
library(popEpi)
library( mgcv)
library(tidyverse)
sessionInfo()
data( DMlate)
str( DMlate)
```

   You can get a more detailed explanation of the data by referring to the help page:

```
?DMlate
```

2. Set up the dataset as a `Lexis` object with age, calendar time and duration of diabetes as timescales, and date of death as event. Make sure that you know what each of the arguments to `Lexis` mean:

```
LL <- Lexis(entry = list(A = dodm-dobth,
                         P = dodm,
                       dur = 0),
             exit = list(P = dox),
      exit.status = factor(!is.na(dodth),
                           labels = c("Alive","Dead")),
             data = DMlate)
```

   Take a look at the first few lines of the resulting dataset, for example using `head()`.

3. Get an overview of the mortality by using `stat.table` to tabulate no. deaths, person-years (`lex.dur`) and the crude mortality rate by sex. Try:

```
stat.table(sex,
           list(D = sum(lex.Xst == "Dead"),
                Y = sum(lex.dur),
             rate = ratio(lex.Xst == "Dead", lex.dur, 1000)),
           margins = TRUE,
           data = LL)
```

4. If we want to assess how mortality depends on age, calendar time and duration or how it relates to population mortality, we should in principle split the follow-up along all three time scales. In practice it is sufficient to split it along one of the time-scales and then use the value of each of the time-scales at the left endpoint of the intervals. Use `splitLexis` (or `splitMulti` from the `popEpi` package) to split the follow-up along the age-axis in sutiable intervals (here set to 1/2 year, but really immaterial as long as it is small):

```
SL <- splitLexis(LL, breaks = seq(0,125,1/2), time.scale = "A")
summary(SL)
```

   How many records are now in the dataset? How many person-years? Compare to the original `Lexis`-dataset.

Age-specific mortality

5. Now estimate age-specific mortality curves for men and women separately, using splines as implemented in `gam`. We use `k = 20` to be sure to catch any irregularities by age.

```
r.m <- gam(cbind(lex.Xst == "Dead", lex.dur) ~ s(A, k = 20),
           family = poisreg,
           data = subset(SL, sex == "M"))
gam.check(r.m)
```

   Make sure you understand all the components on this modeling statement. Fit the same model for women. There is a convenient wrapper for this, exploiting the `Lexis` structure of data, but which does not have an update

```
r.m <- gam.Lexis(subset(SL, sex == "M"), ~ s(A, k = 20))
r.f <- gam.Lexis(subset(SL, sex == "F"), ~ s(A, k = 20))
gam.check(r.m)
gam.check(r.f)
```

6. Now, extract the estimated rates by using the wrapper function `ci.pred` that computes predicted rates and confidence limits for these. However, when using the `glm.Lexis` or `gam.Lexis` we aviod this; they rely on the `poisreg` family that will return the rates in the (inverse) units in which the person-years were given; that is the units of `lex.dur`.

```
nd <-  data.frame(A = seq(20,90,0.5))
p.m <- ci.pred(r.m, newdata = nd)
p.f <- ci.pred(r.f, newdata = nd)
str(p.m)
```

7. Plot the predicted rates for men and women together - using for example `matplot` or `matshade`.

```
p.f <- ci.pred(r.f, newdata = nd)
matplot(nd$A, cbind(p.m,p.f) * 1000,
        type = "l", col = rep(c("blue","red"),each = 3), lwd = c(3,1,1), lty = 1,
        log = "y", xlab = "Age", ylab = "Mortality of DM ptt per 1000 PY")
```

Further time scales: period and duration

8. We now want to model the mortality rates among diabetes patients also including current date and duration of diabetes, using penalized splines. Use the argument `bs = "cr"` to `s()` to get cubic splines indstead of thin plate (`"tp"`) splines which is the default. Check if you have a reasonable fit using `gam.check`.

```
Mtp <- gam((lex.Xst == "Dead") ~ s(  A, bs = "cr", k = 10) +
                                 s(  P, bs = "cr", k = 10) +
                                 s(dur, bs = "cr", k = 10),
           offset = log(lex.dur),
           family = poisson,
             data = subset(SL, sex == "M"))
```

An easier specification of the model exploits the `Lexis` class of the dataset, try:

```
Mcr <- gam.Lexis(subset(SL, sex == "M"),
                 ~ s(  A, bs = "cr", k = 10) +
                   s(  P, bs = "cr", k = 10) +
                   s(dur, bs = "cr", k = 10))
summary(Mcr)
gam.check(Mcr)
```

Fit the same model for women as well. Are the models reasonably fitting?

9. Plot the estimated effects, using the default plot method for `gam` objects. Remember that there are three effects estimated, so it is useful set up a multi-panel display, and for the sake of comparability to set ylim to the same for men and women:

```
par(mfrow = c(2,3))
plot(Mcr, ylim = c(-3,3))
plot(Fcr, ylim = c(-3,3))
```

What is the absolute scale for these effects?

10. Compare the fit of the naive model with just age and the three-factor models, using `anova`, e.g.:

```
anova(Mcr, r.m, test = "Chisq")
```

What do you conclude?

11. The model we fitted has three time-scales: current age, current date and current duration of diabetes, so the effects that we report are not immediately interpretable, as they are (as in any kind of multiple regressions) to be interpreted as "all else equal" which they are not, as the three time scales advance simultaneously at the same pace. The reporting would therefore more naturally be on the mortality scale as a function of age, but showing the mortality for persons diagnosed in different ages, using separate displays for separate years of diagnosis. This is most easily done using the `ci.pred` function with the `newdata =` argument. So a person diagnosed in age 50 in 1995 will have a mortality measured in cases per 1000 PY as:

```
pts <- seq(0, 12, 1/4)
nd <- data.frame(A =   50 + pts,
                 P = 1995 + pts,
               dur =        pts)
m.pr <- ci.pred(Mcr, newdata = nd)
```

Note that because we used `gam.Lexis` which uses the`poisreg` family we need not specify `lex.dur` as a variable in the prediction data frame `nd`. Predictions will be rates in the same units as `lex.dur`. Now take a look at the result from the `ci.pred` statement and construct prediction of mortality for men and women diagnosed in a range of ages, say 50, 60, 70, and plot these together in the same graph:

```
cbind(nd, ci.pred(Mcr, newdata = nd))
```

12. From figure it seems that the duration effect is over-modeled, so refit constraining the d.f. to 5:

```
Mcr <- gam.Lexis(subset(SL, sex == "M"),
                 ~ s(  A, bs = "cr", k = 10) +
                   s(  P, bs = "cr", k = 10) +
                   s(dur, bs = "cr", k = 5))
summary(Mcr)
Fcr <- gam.Lexis(subset(SL, sex == "F"),
                 ~ s(  A, bs = "cr", k = 10) +
                   s(  P, bs = "cr", k = 10) +
                   s(dur, bs = "cr", k = 5))
```

How does `gam.check()` look for these models? Plot the estimated rates from the revised models. What do you conclude from the plots?

SMR

The SMR is the **S**tandardized **M**ortality **R**atio, which is the mortality rate-ratio between the diabetes patients and the general population. In real studies we would subtract the deaths and the person-years among the diabetes patients from those of the general population, but since we do not have access to these, we make the comparison to the general population at large, *i.e.* also including the diabetes patients. So we now want to include the population mortality rates as a fixed variable in the split dataset; for each record in the split dataset we attach the value of the population mortality for the relevant sex, and and calendar time. This can be achieved in two ways: Either we just use the current split of follow-up time and allocate the population mortality rates for some suitably chosen (mid-)point of the follow-up in each, or we make a second split by date, so that follow-up in the diabetes patients is in the same classification of age and data as the population mortality table.

13. We will use the former approach, using the dataset split in 6 month intervals, and then include as an extra variable the population mortality as available from the data set `M.dk`. First create the variables in the diabetes dataset that we need for matching with the population mortality data, that is sex and age and date at the midpoint of each of the intervals (or rater at a point 3 months after the left endpoint of the interval — recall we split the follow-up in 6 month intervals). We need to have variables of the same type when we merge, so we must transform the sex variable in `M.dk` to a factor, and must for each follow-up interval in the `SL` data have an age and a period variable that can be used in merging with the population data.

```
str(SL)
SL$Am <- floor(SL$A + 0.25)
SL$Pm <- floor(SL$P + 0.25)
data(M.dk)
str(M.dk)
M.dk <- transform(M.dk, Am = A,
                        Pm = P,
                       sex = factor(sex, labels = c("M","F")))
str(M.dk)
```

Then match the rates from `M.dk` into `SL` — `sex`, `Am` and `Pm` are the common variables, and therefore the match is on these variables:

```
SLr <- merge(SL, M.dk[,c("sex", "Am", "Pm", "rate")])
dim(SL)
dim(SLr)
```

This merge (remember to `?merge`!) only takes rows that have information from both datasets, hence the slightly fewer rows in `SLr` than in `SL`.

14. Compute the expected number of deaths as the person-time multiplied by the corresponding population rate, and put it in a new variable, `E`, say (Expected). Use `stat.table` to make a table of observed, expected and the ratio (SMR) by age (suitably grouped, look for `cut`) and sex.

15. Fit a poisson model with sex as the explanatory varable and log-expected as offset to derive the SMR (and c.i.). Some of the population mortality rates are 0, so you need to exclude those records from the analysis.

```
msmr <- glm((lex.Xst == "Dead") ~ sex - 1 + offset(log(E)),
            family = poisson,
            data = subset(SLr,E>0))
ci.exp(msmr)
```

Recogninze the numbers?

16. The same model can be fitted a bit simpler by the `poisreg` family, try:

```
msmr <- glm(cbind(lex.Xst == "Dead", E) ~ sex - 1,
           family = poisreg,
             data = subset(SLr, E > 0))
ci.exp(msmr)
```

We can assess the ratios of SMRs between men and women by using the `ctr.mat` argument which should be a matrix:

```
(CM <- rbind(M = c(1,0),
             W = c(0,1),
         'M/F' = c(1,-1)))
round(ci.exp(msmr, ctr.mat = CM), 2)
```

What do you conclude?

1.13.1 SMR modeling

17. Now model the SMR using age and date of diagnosis and diabetes duration as explanatory variables, including the expected-number instead of the person-years, using separate models for men and women. You cannot use `gam.Lexis` from the code you used for fitting models for the rates, you need to use `gam` with the `poisreg` family. And remember to exclude those units where no deaths in the population occur (that is where the rate is 0). Plot the estimated smooth effects for both men and women using e.g. `plot.gam`. What do you see?

18. Plot the predicted SMRs from the models for men and women diagnosed in ages 50, 60 and 70 as you dif for the rates. What do you see?

19. Try to simplify the model to one with a simple sex effect, separate linear effects of age and date of follow-up for each sex, and a smooth effect of duration common for both sexes, giving an estimate of the change in SMR by age and calendar time. How much does SMR change by each year of age? And by each calendar year?

20. Use your previous code to plot the predicted mortality from this model too. Are the predicted SMR curves credible?

1.14 Causal inference

1.14.1 Proper adjustment for confounding in regression models

The first exercise of this session will ask you to simulate some data according to pre-specified causal structure (don't take the particular example too seriously) and see how you should adjust the analysis to obtain correct estimates of the causal effects.

Suppose one is interested in the effect of beer-drinking on body weight. Let's *assume* that in addition to the potential effect of beer on weight, the following is true in reality:

- Men drink more beer than women
- Men have higher body weight than women
- People with higher body weight tend to have higher blood pressure
- Beer-drinking increases blood pressure

The task is to simulate a dataset in accordance with this model, and subsequently analyse it to see, whether the results would allow us to conclude the true association structure.

1. Sketch a causal graph (not necessarily with R) to see, how should one generate the data
2. Suppose the actual effect sizes are following:
   - The probability of beer-drinking is 0.2 for females and 0.7 for males
   - Men weigh on average $10kg$ more than women
   - One kg difference in body weight corresponds in average to $0.5mmHg$ difference in (systolic) blood pressures
   - Beer-drinking increases blood pressure by $10mmHg$ in average.
   - Beer-drinking has **no** effect on body weight

   The R commands to generate the data are:

```
> bdat= data.frame(sex = c(rep(0,500),rep(1,500))  )
>                    # a data frame with 500 females, 500 males
> bdat$beer <- rbinom(1000,1,0.2+0.5*bdat$sex)
> bdat$weight <- 60 + 10*bdat$sex + rnorm(1000,0,7)
> bdat$bp <- 110 + 0.5*bdat$weight + 10*bdat$beer + rnorm(1000,0,10)
```

3. Now fit the following models for body weight as dependent variable and beer-drinking as independent variable. Look, what is the estimated effect size:
   (a) Unadjusted (just simple linear regression)
   (b) Adjusted for sex
   (c) Adjusted for sex and blood pressure
4. What would be the conclusions on the effect of beer on weight, based on the three models? Do they agree? Which (if any) of the models gives an unbiased estimate of the actual causal effect of interest?

5. How can the answer be seen from the graph?

6. Now change the data-generation algorithm so, that in fact beer-drinking does increase the body weight by 2kg. Look, what are the conclusions in the above models now. Thus the data is generated as before, but the weight variable is computed as:

```
> bdat$weight <- 60 + 10*bdat$sex + 2*bdat$beer + rnorm(1000,0,7)
```

7. Suppose one is interested in the effect of beer-drinking on blood pressure instead, and is fitting a) an unadjusted model for blood pressure, with beer as an only covariate; b) a model with beer, weight and sex as covariates. Would either a) or b) give an unbiased estimate for the effect? (You may double-check whether the simulated data is consistent with your answer).

1.14.2 Instrumental variables estimation, Mendelian randomization and assumptions

In the lecture slides it was shown that in a model for blood glucose level (associated with the risk of diabetes), both BMI and FTO genotype were significant. Seeing such result in a real dataset may misleadingly be interpreted as an evidence of a direct effect of FTO genotype on glucose. Conduct a simulation study to verify that one may see a significant genotype effect on outcome in such model if in fact the assumptions for Instrumental Variables estimation (Mendelian Randomization) are valid – genotype has a direct effect on the exposure only, whereas exposure-outcome association is confounded.

1. Start by generating the genotype variable as *Binomial(2,p)*, with $p = 0.2$:

```
>  n <- 10000
>  mrdat <- data.frame(G = rbinom(n,2,0.2))
>  table(mrdat$G)
```

2. Also generate the confounder variable U

```
> mrdat$U <- rnorm(n)
```

3. Generate a continuous (normally distributed) exposure variable BMI so that it depends on G and U. Check with linear regression, whether there is enough power to get significant parameter estimates. For instance:

```
> mrdat$BMI <- with(mrdat, 25 + 0.7*G + 2*U + rnorm(n) )
```

4. Finally generate Y ("Blood glucose level") so that it depends on BMI and U (but not on G).

```
> mrdat$Y <- with(mrdat, 3 + 0.1*BMI - 1.5*U + rnorm(n,0,0.5) )
```

5. Verify, that simple regression model for Y, with BMI as a covariate, results in a biased estimate of the causal effect (parameter estimate is different from what was generated) How different is the estimate from 0.1?

6. Estimate a regression model for Y with two covariates, G and BMI. Do you see a significant effect of G? Could you explain analytically, why one may see a significant parameter estimate for G there?

7. Find an IV (instrumental variables) estimate, using G as an instrument, by following the algorithm in the lecture notes (use two linear models and find a ratio of the parameter estimates). Does the estimate get closer to the generated effect size?

```
> mgx<-lm(BMI ~ G, data=mrdat)
> ci.lin(mgx)  # check the instrument effect
> bgx<-mgx$coef[2]   # save the 2nd coefficient (coef of G)
> mgy<-lm(Y ~ G, data=mrdat)
> ci.lin(mgy)
> bgy<-mgy$coef[2]
> causeff <- bgy/bgx
> causeff    # closer to 0.1?
```

8. A proper simulation study would require the analysis to be run several times, to see the extent of variability in the parameter estimates. A simple way to do it here would be using a `for`-loop. Modify the code as follows (exactly the same commands as executed so far, adding a few lines of code to the beginning and to the end):

```
> n <- 10000
> # initializing simulations:
> # 30 simulations (change it, if you want more):
> nsim<-30
> mr<-rep(NA,nsim)   # empty vector for the outcome parameters
> for (i in 1:nsim) { # start the loop
+ ### Exactly the same commands as before:
+ mrdat <- data.frame(G = rbinom(n,2,0.2))
+ mrdat$U <- rnorm(n)
+ mrdat$BMI <- with(mrdat, 25 + 0.7*G + 2*U + rnorm(n) )
+ mrdat$Y <- with(mrdat, 3 + 0.1*BMI - 1.5*U + rnorm(n,0,0.5) )
+ mgx<-lm(BMI ~ G, data=mrdat)
+ bgx<-mgx$coef[2]
+ mgy<-lm(Y ~ G, data=mrdat)
+ bgy<-mgy$coef[2]
+ # Save the i'th parameter estimate:
+ mr[i]<-bgy/bgx
+ }   # end the loop
```

   Now look at the distribution of the parameter estimate:

```
> summary(mr)
```

9. (*optional*) Change the code of simulations so that the assumptions are violated: add a weak direct effect of the genotype G to the equation that generates Y:

```
> mrdat$Y <- with(mrdat, 3 + 0.1*BMI - 1.5*U + 0.05*G + rnorm(n,0,0.5) )
```

   Repeat the simulation study to see, what is the bias in the average estimated causal effect of BMI on Y.

10. (*optional*) Using library `sem` and function `tsls`, obtain a two-stage least squares estimate for the causal effect. Do you get the same estimate as before?

```
> library(sem)
> summary(tsls(Y ~ BMI, ~G, data=mrdat))
```

Why are simulation exercises useful for causal inference?

If we simulate the data, we know the data-generating mechanism and the "true" causal effects. So this is a way to check, whether an analysis approach will lead to estimates that correspond to what is generated. One could expect to see similar phenomena in real data analysis, if the data-generation mechanism is similar to what was used in simulations.

1.15 Nested case-control study and case-cohort study: Risk factors of coronary heart disease

In this exercise we shall apply both the nested case-control (NCC) design and the case-cohort (CC) design in sampling control subjects from a defined cohort or closed study population. The case group comprises those cohort members who die from coronary heart disease (CHD) during a > 20 years follow-up of the cohort. The risk factors of interest are cigarette smoking, systolic blood pressure, and total cholesterol level.

Our study population is an occupational cohort comprising 1501 men working in blue-collar jobs in one Nordic country. Eligible subjects had no history of coronary heart disease when recruited to the study in the early 1990s. Smoking habits and many other items were inquired at baseline by a questionnaire, and blood pressure was measured by a research nurse, the values being written down on the questionnaire. Serum samples were also taken from the cohort members at the same time and were stored in a freezer. For some reason, the data in the questionnaires were not entered to any computer file, but the questionnaires were kept in a safe storehouse for further purposes. Also, no biochemical analyses were initially performed for the sera collected from the participants. However, dates of birth and dates of entry to the study were recorded in an electronic file.

In 2010 the study was suddenly reactivated by those investigators of the original team who were still alive then. As the first step mortality follow-up of the cohort members was executed by record linkage to the national population register, from which the dates of death and emigration were obtained. Another linkage was performed with the national register of causes of death in order to get the deaths from coronary heard disease identified. As a result a data file `occoh.txt` was completed containing the following variables:

`id`	= identification number,
`birth`	= date of birth,
`entry`	= date of recruitment and baseline measurements,
`exit`	= date of exit from mortality follow-up,
`death`	= indicator for vital status at the end of follow-up,
	= 1, if dead from any cause, and = 0, if alive,
`chdeath`	= indicator for death from coronary heart disease,
	= 1, if "yes", and 0, if "no".

This exercise is divided into five main parts:

(1) Description of the study base or the follow-up experience of the whole cohort, identification of the cases and illustrating the risk sets.

(2) Nested case-control study within the cohort: (i) selection of controls by risk set or time-matched sampling using function `ccwc()` in package `Epi`, (ii) collection of exposure data for cases and controls from the pertinent data base of the whole cohort to the case-control data set using function `merge()`, and (iii) analysis of the case-control data set with stratified Cox model using function `clogit()` in package `survival()`,

(3) Case-cohort study within the cohort: (i) selection of a subcohort by simple random sampling from the cohort, (ii) collection of exposure data for subcohort members and cases, and (iii) analysis of the case-cohort data set with Cox model by weighted partial

likelihood including appropriate weighting and correction of estimated covariance matrix for the model coefficients using function `cch()` in package `survival()`.

(4) Comparison of results from all previous analyses, also with those from a full cohort design.

(5) Further tasks and homework.

1.15.1 Reading the cohort data, illustrating the study base and risk sets

11. Load the packages `Epi` and `survival`. Read in the cohort data file and name the resulting data frame as `oc`. See its structure and print the univariate summaries.

```
> library(Epi)
> library(survival)
> url <- "https://raw.githubusercontent.com/SPE-R/SPE/master/pracs/data"
> oc <- read.table( paste(url, "occoh.txt", sep = "/"), header=TRUE)
> str(oc)
> summary(oc)
```

12. It is convenient to change all the dates into fractional calendar years

```
> oc$ybirth <- cal.yr(oc$birth)
> oc$yentry <- cal.yr(oc$entry)
> oc$yexit <- cal.yr(oc$exit)
```

We shall also compute the age at entry and at exit, respectively, as age will be the main time scale in our analyses.

```
> oc$agentry <- oc$yentry - oc$ybirth
> oc$agexit <- oc$yexit - oc$ybirth
```

13. As the next step we shall create a `lexis` object from the data frame along the calendar period and age axes, and as the outcome event we specify the coronary death.

```
> oc.lex <- Lexis( entry = list( per = yentry,
+                                age = yentry - ybirth ),
+                  exit = list( per = yexit),
+            exit.status = chdeath,
+                     id = id, data = oc)
> str(oc.lex)
> summary(oc.lex)
```

14. At this stage it is informative to examine a graphical presentation of the follow-up lines and outcome cases in a conventional Lexis diagram. Make use of the `plot` method for `Lexis` objects. Gray lifelines are drawn and a bullet is put at the exit point of those lifelines that end with the outcome event.

```
> par(mfrow=c(1,1))
> plot( oc.lex, xlim=c(1990, 2010),grid=TRUE )
> points( oc.lex, pch=c(NA, 16)[oc.lex$lex.Xst+1] )
```

15. As age is here the main time axis, we shall graphically illustrate the **study base**, *i.e.* the follow-up lines and outcome events, only along the age scale, being ordered by age at exit. Vertical lines at those ages when new coronary deaths occur are drawn to identify the pertinent **risk sets**. For that purpose it is useful first to sort the data frame and the `Lexis` object jointly by age at exit & age at entry, and to give a new ID number according to that order.

```
> oc.ord <- cbind(ID = 1:1501, oc[ order( oc$agexit, oc$agentry), ] )
> oc.lexord <- Lexis( entry = list( age = agentry ),
+                       exit = list( age = agexit),
+               exit.status = chdeath,
+                        id = ID, data = oc.ord)
> plot(oc.lexord, "age")
> points(oc.lexord, pch=ifelse(oc.lexord$lex.Xst==1, 16, NA) )
> with( subset(oc.lexord, lex.Xst==1),
+       abline( v=agexit, lty=3))
```

16. For a closer look, we now zoom the graphical illustration of the risk sets into event times occurring between 50 to 58 years. – Copy the last four lines from the previous item and add arguments `xlim` and `ylim` to the call of `plot()`.

```
> plot(oc.lexord, "age", xlim=c(50, 58), ylim=c(5, 65))
> points(oc.lexord, "age", pch=ifelse(oc.lexord$lex.Xst==1, 16, NA))
> with( subset(oc.lexord, lex.Xst==1),
+       abline( v=agexit, lty=3))
```

1.15.2 Nested case-control study

We shall now employ the strategy of **risk-set sampling** or **time-matched** sampling of controls, *i.e.* we are conducting a **nested case-control study** within the cohort.

17. The risk sets are defined according to the age at diagnosis of the case. Further matching is applied for age at entry by 1-year agebands. For this purpose we first generate a categorical variable `agen2` for age at entry

```
> oc.lex$agen2 <- cut(oc.lex$agentry, br = seq(40, 62, 1) )
```

Matched sampling from risk sets may be carried out using function `ccwc()` found in the `Epi` package. Its main arguments are the times of `entry` and `exit` which specify the time at risk along the main time scale (here age), and the outcome variable to be given in the `fail` argument. The number of controls per case is set to be two, and the additional matching factor is given. – After setting the RNG seed (with your own number), make a call of this function and see the structure of the resulting data frame `cactrl` containing the cases and the chosen individual controls.

```
> set.seed(98623)
> cactrl <-
+    ccwc(entry=agentry, exit=agexit, fail=chdeath,
+         controls = 2, match= agen2,
+         include = list(id, agentry),
+         data=oc.lex, silent=FALSE)
> str(cactrl)
```

Check the meaning of the four first columns of the case-control data frame from the help page of function `ccwc()`.

18. Now we shall start collecting data on the risk factors for the cases and their matched controls, including determination of the total cholesterol levels from the frozen sera! The storehouse of the risk factor measurements for the whole cohort is file `occoh-Xdata.txt`. It contains values of the following variables.

`id` =	identification number, the same as in `occoh.txt`,
`smok` =	cigarette smoking with categories, 1: "never", 2: "former", 3: "1-14/d", 4: "15+/d",
`sbp` =	systolic blood pressure (mmHg),
`tchol` =	total cholesterol level (mmol/l).

```
> ocX <- read.table( paste(url, "occoh-Xdata.txt", sep = "/"), header=TRUE)
> str(ocX)
```

19. In the next step we collect the values of the risk factors for our cases and controls by merging the case-control data frame and the storehouse file. In this operation we utilize function `merge()` to select columns of two data frames: `cactrl` (all columns) and `ocX` (four columns) and to merge these into a single file (see exercise 1.1, subsection 1.1.8, where `merge()` was introduced). The `id` variable in both files is used as the key to link each individual case or control with his own data on risk factors.

```
> oc.ncc <- merge(cactrl, ocX[, c("id", "smok", "tchol", "sbp")],
+    by = "id")
> str(oc.ncc)
```

20. We shall treat smoking as categorical and total cholesterol and systolic blood pressure as quantitative risk factors, but the values of the latter will be divided by 10 to get more interpretable effect estimates.

    Convert the smoking variable into a factor.

```
> oc.ncc$smok <- factor(oc.ncc$smok,
+     labels = c("never", "ex", "1-14/d", ">14/d"))
```

21. It is useful to start the analysis of case-control data by simple tabulations by the categorized risk factors. Crude estimates of the rate ratios associated with them, in which matching is ignored, can be obtained as follows. We shall focus on smoking

```
> stat.table( index = list( smok, Fail ),
+           contents = list( count(), percent(smok) ),
+            margins = T, data = oc.ncc )
> smok.crncc <- glm( Fail ~ smok, family=binomial, data = oc.ncc)
> round(ci.exp(smok.crncc), 3)
```

22. A proper analysis takes into account matching that was employed in the selection of controls for each case from the pertinent risk set, further restricted to subjects who were about the same age at entry as the case was. Also, adjustment for the other risk factors is desirable. In this analysis function `clogit()` in `survival` package is utilized. It is in fact a wrapper of function `coxph()`.

```
> m.clogit <- clogit( Fail ~ smok + I(sbp/10) + tchol +
+        strata(Set), data = oc.ncc )
> summary(m.clogit)
> round(ci.exp(m.clogit), 3)
```

Compare these with the crude estimates obtained above.

1.15.3 Case-cohort study

Now we start applying the second major outcome-selective sampling strategy for collecting exposure data from a big study population

23. The subcohort is selected as a simple random sample ($n = 260$) from the whole cohort. The `id`-numbers of the individuals that are selected will be stored in vector `subcids`, and `subcind` is an indicator for inclusion to the subcohort.

```
> N <- 1501; n <- 260
> set.seed(15792)
> subcids <- sample(N, n )
> oc.lexord$subcind <- 1*(oc.lexord$id %in% subcids)
```

24. We form the data frame `oc.cc` to be used in the subsequent analysis selecting the union of the subcohort members and the case group from the data frame of the full cohort. After that we collect the data of the risk factors from the data storehouse for the subjects in the case-cohort data

```
> oc.cc <- subset( oc.lexord, subcind==1 | chdeath ==1)
> oc.cc <- merge( oc.cc, ocX[, c("id", "smok", "tchol", "sbp")],
+    by ="id")
> str(oc.cc)
```

25. We shall now create a graphical illustration of the lifelines contained in the case-cohort data. Lines for the subcohort non-cases are grey without bullet at exit, those for subcohort cases are blue with blue bullet at exit, and for cases outside the subcohort the lines are red and dotted with red bullets at exit.

```
> plot( subset(oc.cc, chdeath==0), "age")
> lines( subset(oc.cc, chdeath==1 & subcind==1), col="blue")
> lines( subset(oc.cc, chdeath==1 & subcind==0), col="red")
> points(subset(oc.cc, chdeath==1), pch=16,
+        col=c("blue", "red")[oc.cc$subcind+1])
```

26. Define the categorical smoking variable again.

```
> oc.cc$smok <- factor(oc.cc$smok,
+     labels = c("never", "ex", "1-14/d", ">14/d"))
```

A crude estimate of the hazard ratio for the various smoking categories k vs. non-smokers ($k = 1$) can be obtained by tabulating cases (D_k) and person-years (y_k) in the subcohort by smoking and then computing the relevant exposure odds ratio for each category:

$$\mathrm{HR}_k^{\mathrm{crude}} = \frac{D_k/D_1}{y_k/y_1}$$

```
> sm.cc <- stat.table( index = smok,
+    contents = list( Cases = sum(lex.Xst), Pyrs = sum(lex.dur) ),
+          margins = T, data = oc.cc)
> print(sm.cc, digits = c(sum=0, ratio=1))
> HRcc <- (sm.cc[ 1, -5]/sm.cc[ 1, 1])/(sm.cc[ 2, -5]/sm.cc[2, 1])
> round(HRcc, 3)
```

Do these estimates resemble those obtained from nested case-control data?

27. To estimate the rate ratios associated with smoking and adjusted for the other risk factors we now fit the pertinent Cox model applying the method of *weighted partial likelihood* as presented by Ling & Ying (1993) and Barlow (1994). This analysis can be done using function `cch()` in package `survival` with `method = "LinYing"`

```
> oc.cc$survobj <- with(oc.cc, Surv(agentry, agexit, chdeath) )
> cch.LY <- cch( survobj ~  smok + I(sbp/10)  + tchol, stratum=NULL,
+    subcoh = ~subcind, id = ~id,  cohort.size = N, data = oc.cc,
+     method ="LinYing" )
> summary(cch.LY)
```

1.15.4 Full cohort analysis and comparisons

Finally, suppose the investigators after all could afford to collect the data on risk factors from the storehouse for the whole cohort.

28. Let us form the data frame corresponding to the full cohort design and convert again smoking to be categorical.

```
> oc.full <- merge( oc.lex, ocX[, c("id", "smok", "tchol", "sbp")],
+    by.x = "id", by.y = "id")
> oc.full$smok <- factor(oc.full$smok,
+     labels = c("never", "ex", "1-14/d", ">14/d"))
```

Juts for comparison with the corresponding analysis in case-cohort data perform a similar crude estimation of hazard ratios associated with smoking.

```
> sm.coh <- stat.table( index = smok,
+    contents = list( Cases = sum(lex.Xst), Pyrs = sum(lex.dur) ),
+          margins = T, data = oc.full)
> print(sm.coh, digits = c(sum=0, ratio=1))
> HRcoh <- (sm.coh[ 1, -5]/sm.coh[ 1, 1])/(sm.coh[ 2, -5]/sm.coh[2, 1])
> round(HRcoh, 3)
```

29. Fit now the ordinary Cox model to the full cohort. There is no need to employ extra tricks upon the ordinary `coxph()` fit.

```
> cox.coh <- coxph( Surv(agentry, agexit, chdeath) ~
+         smok + I(sbp/10)  + tchol, data = oc.full)
> summary(cox.coh)
```

30. Lastly, a comparison of the point estimates and standard errors between the different designs, including variants of analysis for the case-cohort design, can be performed.

```
> betas <- cbind( coef(cox.coh), coef(m.clogit), coef(cch.LY) )
> colnames(betas) <-  c("coh", "ncc", "cch.LY")
> round(betas, 3)
> SEs <- cbind( sqrt( diag( cox.coh$var ) ),
+               sqrt( diag( m.clogit$var ) ),
+               sqrt( diag( cch.LY$var ) ) )
> colnames(SEs) <- colnames(betas)
> round(SEs, 3)
```

You will notice that the point estimates of the coefficients obtained from the full cohort, nested case-control, and case-cohort analyses, respectively, are somewhat variable. However, the standard errors from the NCC and CC analyses should be quite similar when the numbers of cases and non-cases are similar.

1.15.5 Further exercises and homework

31. If you have time, you could run both the NCC study and CC study again but now with a larger control group or subcohort; for example 4 controls per case in NCC and $n = 520$ as the subcohort size in CC. Remember resetting the seed first. Pay attention in the results to how much closer will be the point estimates and the proper SEs to those obtained from the full cohort design.

32. Instead of simple linear terms for `sbp` and `tchol` you could try to fit spline models to describe their effects.

33. A popular alternative to weighted partial likelihood in the analysis of case-cohort data is the *pseudo-likelihood method* (Prentice 1986), which is based on "late entry" to follow-up of the case subjects not belonging to the subcohort. The way to do this is provided by function `cch()` which you can apply directly to the case-cohort data `oc.cc` as before but now with `method = "Prentice"`. – Try this and compare the results with those obtained by weighted partial likelihood in model `cch.LY`.

34. Yet another computational solution for maximizing weighted partial likelihood is provided by a combination of functions `twophase()` and `svycoxph()` of the `survey` package. The approach is illustrated with an example in a vignette "Two-phase designs in epidemiology" by Thomas Lumley (see `http://cran.r-project.org/web/packages/survey/vignettes/epi.pdf`). – You can try this at home and check that you would obtain similar results as with model `cch.LY`.

1.16 Time-dependent variables and multiple states

The following practical exercise is based on the data from paper:

P Hovind, L Tarnow, P Rossing, B Carstensen, and HH Parving: Improved survival in patients obtaining remission of nephrotic range albuminuria in diabetic nephropathy. *Kidney Int*, **66**(3):1180–1186, Sept 2004.

You can find a `.pdf`-version of the paper here:
http://BendixCarstensen.com/~bxc/AdvCoh/papers/Hovind.2004.pdf

1.16.1 The renal failure dataset

The dataset `renal.dta` contains data on follow up of 125 patients from Steno Diabetes Center. They enter the study when they are diagnosed with nephrotic range albuminuria (NRA). This is a condition where the levels of albumin in the urine is exceeds a certain level as a sign of kidney disease. The levels may however drop as a consequence of treatment, this is called remission. Patients exit the study at death or kidney failure (dialysis or transplant).

Table 1.2: *Variables in* `renal.dta`.

`id`	Patient id
`sex`	1=male, 2=female
`dob`	Date of birth
`doe`	Date of entry into the study (2.5 years after NRA)
`dor`	Date of remission. Missing if no remission has occurred
`dox`	Date of exit from study
`event`	Exit status: 1,2,3=event (death, ESRD), 0=censored

1. The dataset is in Stata-format, so you must read the dataset using `read.dta` from the `foreign` package (which is part of the standard R-distribution). At the same time, convert `sex` to a proper factor. Choose where to read the dataset.

```
library( Epi ) ; clear()
library( foreign )
renal <- read.dta(
         "https://raw.githubusercontent.com/SPE-R/SPE/master/pracs/data/renal.dta")
# renal <- read.dta( "http://BendixCarstensen.com/SPE/data/renal.dta" )
# renal <- read.dta( "./data/renal.dta" )
renal$sex <- factor( renal$sex, labels=c("M","F") )
head( renal )
```

2. Use the `Lexis` function to declare the data as survival data with age, calendar time and time since entry into the study as timescales. Label any event > 0 as "ESRD", i.e. renal death (death of kidney (transplant or dialysis), or person). Note that you must make sure that the "alive" state (here `NRA`) is the first, as `Lexis` assumes that everyone starts in this state (unless of course `entry.status` is specified):

```
Lr <- Lexis( entry = list( per=doe,
                           age=doe-dob,
                           tfi=0 ),
             exit = list( per=dox ),
       exit.status = factor( event>0, labels=c("NRA","ESRD") ),
             data = renal )
str( Lr )
summary( Lr )
```

Make sure you know what the variables in `Lr` stand for.

3. Visualize the follow-up in a Lexis-diagram, by using the `plot` method for `Lexis` objects.

```
plot( Lr, col="black", lwd=3 )
subset( Lr, age<0 )
```

What is wrong here? List the data for the person with negative entry age.

4. Correct the data and make a new plot, for example by:

```
Lr <- transform( Lr, dob = ifelse( dob>2000, dob-100, dob ),
                     age = ifelse( dob>2000, age+100, age ) )
subset( Lr, id==586 )
plot( Lr, col="black", lwd=3 )
```

5. (*Optional, esoteric*) We can produce a slightly more fancy Lexis diagram. Note that we have a x-axis of 40 years, and a y-axis of 80 years, so when specifying the output file adjust the *total* width of the plot so that the use of `mai` (look up the help page for `par`) to specify the margins of the plot so that it leaves a plotting area twice as high as wide. The `mai` argument to `par` gives the margins in inches, so the total size of the horizontal and vertical margins is 1 inch each, to which we add 80/5 in the height, and 40/5 in the horizontal direction, each giving exactly 5 years per inch in physical size.

6. Now make a Cox-regression analysis of the enpoint ESRD with the variables sex and age at entry into the study, using time since entry to the study as time scale.

```
library( survival )
mc <- coxph( Surv( lex.dur, lex.Xst=="ESRD" ) ~
             I(age/10) + sex, data=Lr )
summary( mc )
```

What is the The hazard ratio between males and females? Between two persons who differ 10 years in age at entry?

7. The main focus of the paper was to assess whether the occurrence of remission (return to a lower level of albumin excretion, an indication of kidney recovery) influences mortality. "Remission" is a time-dependent variable which is initially 0, but takes the value 1 when remission occurs. In order to handle this, each person who sees a remission must have two records:

   - One record for the time before remission, where entry is `doe`, exit is `dor`, remission is 0, and event is 0.

- One record for the time after remission, where entry is `dor`, exit is `dox`, remission is 1, and event is 0 or 1 according to whether the person had an event at `dox`.

This is accomplished using the `cutLexis` function on the `Lexis` object, where we introduce a remission state "Rem". You must declare the "NRA" state as a precursor state, i.e. a state that is *less* severe than "Rem" in the sense that a person who see a remission will stay in the "Rem" state unless he goes to the "ESRD" state. Also use `split.state=TRUE` to have different ESRD states according to whether a person had had remission or not prioer to ESRD. The statement to do this is:

```
Lc <- cutLexis( Lr, cut = Lr$dor, # where to cut follow up
                timescale = "per",  # what timescale are we referring to
                new.state = "Rem",  # name of the new state
              split.state = TRUE,   # different states sepending on previous
         precursor.states = "NRA" ) # which states are less severe
summary( Lc )
```

List the records from a few select persons (choose values for `lex.id`, using for example `subset( Lc, lex.id %in% c(5,7,9) )`, or other numbers).

8. Now show how the states are connected and the number of transitions between them by using `boxes`. This is an interactive command that requires you to click in the graph window:

```
boxes( Lc )
```

It has a couple of fancy arguments, try:

```
boxes( Lc, boxpos=TRUE, scale.R=100, show.BE=TRUE, hm=1.5, wm=1.5 )
```

You may even be tempted to read the help page for `boxes.Lexis` ...

9. Plot a Lexis diagram where different coloring is used for different segments of the follow-up. The `plot.Lexis` function draws a line for each record in the dataset, so you can index the coloring by `lex.Cst` and `lex.Xst` as appropriate — indexing by a factor corresponds to indexing by the *index number* of the factor levels, so you must be know which order the factor levels are in:

```
par( mai=c(3,3,1,1)/4, mgp=c(3,1,0)/1.6 )
plot( Lc, col=c("red","limegreen")[Lc$lex.Cst],
      xlab="Calendar time", ylab="Age",
      lwd=3, grid=0:20*5, xlim=c(1970,2010), ylim=c(0,80), xaxs="i", yaxs="i", las=1 )
points( Lc, pch=c(NA,NA,16)[Lc$lex.Xst],
            col=c("red","limegreen","transparent")[Lc$lex.Cst])
points( Lc, pch=c(NA,NA,1)[Lc$lex.Xst],
            col="black", lwd=2 )
```

10. Make Cox-regression of mortality (i.e. endpoint "ESRD" or "ESRD(Rem)") with sex, age at entry and remission as explanatory variables, using time since entry as timescale, and include `lex.Cst` as time-dependent variable, and indicate that each record represents follow-up from `tfi` to `tfi+lex.dur`. Make sure that you know why what goes where here in the call to `coxph`.

```
( EP <- levels(Lc)[3:4] )
m1 <- coxph( Surv( tfi,                    # from
                   tfi+lex.dur,            # to
                   lex.Xst %in% EP ) ~     # event
             sex + I((doe-dob-50)/10) +    # fixed covariates
             (lex.Cst=="Rem"),             # time-dependent variable
             data = Lc )
summary( m1 )
```

What is the effect of of remission on the rate of ESRD?

1.16.2 Splitting the follow-up time

In order to explore the effect of remission on the rate of ESRD, we shall split the data further into small pieces of follow-up. To this end we use the function `splitLexis`. The rates can then be modeled using a Poisson-model, and the shape of the underlying *rates* be explored. Furthermore, we can allow effects of both time since NRA and current age. To this end we will use splines, so we need the `splines` and also the `mgcv` packages.

11. Now split the follow-up time every month after entry, and verify that the number of events and risk time is the same as before and after the split:

```
sLc <- splitLexis( Lc, "tfi", breaks=seq(0,30,1/12) )
summary( Lc, scale=100 )
summary(sLc, scale=100 )
```

12. Try to fit the Poisson-model corresponding to the Cox-model we fitted previously. The function `Ns()` produces a model matrix corresponding to a piece-wise cubic function, modeling the baseline hazard explicitly (think of the `ns` terms as the baseline hazard that is not visible in the Cox-model). Use teh wrapper function `glm.Lexis`

```
mp <- glm.Lexis(sLc, ~ Ns(tfi, knots = c(0,2,5,10)) +
                       sex + I((doe-dob-40)/10) + I(lex.Cst=="Rem"))
ci.exp( mp )
```

How does the effects of sex change from the Cox-model?

13. Try instead using the `gam` function from the `mgcv` package. There is convenience wrapper for this for `Lexis` objects as well:

```
library( mgcv )
mx <- gam.Lexis(sLc,
                ~ s(tfi, k=10) + sex + I((doe-dob-40)/10) + I(lex.Cst=="Rem"))
ci.exp( mp, subset=c("Cst","doe","sex") )
ci.exp( mx, subset=c("Cst","doe","sex") )
```

We see that there is virtually no difference between the two approaches in terms of the regression parameters.

14. Extract the regression parameters from the models using `ci.exp` and compare with the estimates from the Cox-model:

```
ci.exp( mx, subset=c("sex","dob","Cst"), pval=TRUE )
ci.exp( m1 )
round( ci.exp( mp, subset=c("sex","dob","Cst") ) / ci.exp( m1 ), 2 )
```

How lare is the difference in estimated regression parameters?

15. The model has the same assumptions as the Cox-model about proportionality of rates, but there is an additional assumption that the hazard is a smooth function of time since entry. It seems to be a sensible assumption (well, restriction) to put on the rates that they vary smoothly by time. No such restriction is made in the Cox model. The `gam` model optimizes the shape of the smoother by general cross-validation. Try to look at the shape of the estimated effect of `tfi`:

    ```
    plot( mx )
    ```

    Is this a useful plot?

16. However, `plot` does not give you the *absolute* level of the underlying rates because it bypasses the intercept. So try to predict the rates as a function of `tfi` and the covariates, by setting up a prediction data frame. Note that age in the model specification is entered as `doe-dob`, hence the prediction data frame must have these two variables and not the age, but it is onlythe difference that matters for the prediction:

    ```
    nd <- data.frame(tfi = seq(0,20,0.1),
                     sex = "M",
                     doe = 1990,
                     dob = 1940,
                 lex.Cst = "NRA")
    str(nd)
    matshade(nd$tfi, cbind(ci.pred(mp, newdata = nd),
                           ci.pred(mx, newdata = nd)) * 100, plot = TRUE,
             type="l", lwd = 3:4, col = c("black", "forestgreen"),
             log = "y", xlab = "Time since entry (years)",
             ylab = "ESRD rate (per 100 PY) for 50 year man")
    ```

    Try to overlay with the corresponding prediction from the `glm` model using `Ns`.

1.16.3 Prediction from the multistate model

If we want to make proper statements about the survival and disease probabilities we must know not only how the occurrence of remission influences the rate of death/ESRD, but we must also model the occurrence rate of remission itself.

17. The rates of ESRD were modelled by a Poisson model with effects of age and time since NRA — in the models `mp` and `mx`. But if we want to model whole process we must also model the remission rates transition from "NRA" to "Rem", but the number of events is rather small so we restrict covariates in this model to only time since NRA and sex. Note that only the records that represent follow-up in the "NRA" state should be used; this is most easily done using the `gam.Lexis` function

```
mr <- gam.Lexis(sLc, ~ s( tfi, k=10 ) + sex,
                     from = "NRA",
                       to = "Rem")
ci.exp(mr, pval = TRUE)
```

What is the remission rate-ration between men and women?

18. If we want to predict the probability of being in each of the three states using these estimated rates, we may resort to analytical calculations of the probabilities from the estimated rates, which is actually doable in this case, but which will be largely intractable for more complicated models. Alternatively we can *simulate* the life course for a large group of (identical) individuals through a model using the estimated rates. That will give a simulated cohort (in the form of a `Lexis` object), and we can then just count the number of persons in each state at each of a set of time points. This is accomplished using the function `simLexis`. The input to this is the initial status of the persons whose life-course we shall simulate, and the transition rates in suitable form:

    - Suppose we want predictions for men aged 50 at NRA. The input is in the form of a `Lexis` object (where `lex.dur` and `lex.Xst` will be ignored). Note that in order to carry over the `time.scales` and the `time.since` attributes, we construct the input object using `subset` to select columns, and `NULL` to select rows (see the example in the help file for `simLexis`):

      ```
      inL <- subset( sLc, select=1:11 )[NULL,]
      str( inL )
      timeScales(inL)
      inL[1,"lex.id"] <- 1
      inL[1,"per"] <- 2000
      inL[1,"age"] <- 50
      inL[1,"tfi"] <- 0
      inL[1,"lex.Cst"] <- "NRA"
      inL[1,"lex.Xst"] <- NA
      inL[1,"lex.dur"] <- NA
      inL[1,"sex"] <- "M"
      inL[1,"doe"] <- 2000
      inL[1,"dob"] <- 1950
      inL <- rbind( inL, inL )
      inL[2,"sex"] <- "F"
      inL
      str( inL )
      ```

    - The other input for the simulation is the transitions, which is a list with an element for each transient state (that is "NRA" and "Rem"), each of which is again a list with names equal to the states that can be reached from the transient state. The content of the list will be `glm` objects, in this case the models we just fitted, describing the transition rates:

      ```
      Tr <- list( "NRA" = list( "Rem"  = mr,
                                "ESRD" = mx ),
                  "Rem" = list( "ESRD(Rem)" = mx ) )
      ```

    With this as input we can now generate a cohort, using `N=5` to simulate life course of 10 persons (5 for each set of starting values in `inL`):

```
( iL <- simLexis( Tr, inL, N=10 ) )
summary( iL, by="sex" )
```

What type of object have you got as `iL`. Simulate a couple of thousand persons.

19. Now generate the life course of 5,000 persons, and look at the summary. The `system.time` command is just to tell you how long it took, you may want to start with 1000 just to see how long that takes.

```
system.time(
sM <- simLexis( Tr, inL, N = 5000, t.range = 12 ) )
summary( sM, by="sex" )
```

Why are there so many ESRD-events in the resulting data set?

20. Now count how many persons are present in each state at each time for the first 10 years after entry (which is at age 50). This can be done by using `nState`. Try:

```
nStm <- nState( subset(sM,sex=="M"), at=seq(0,10,0.1), from=50, time.scale="age" )
nStf <- nState( subset(sM,sex=="F"), at=seq(0,10,0.1), from=50, time.scale="age" )
head( nStf )
```

What is tn the object `nStf`?

21. With the counts of persons in each state at the designated time points (in `nStm`), compute the cumulative fraction over the states, arranged in order given by `perm`:

```
ppm <- pState( nStm, perm=c(2,1,3,4) )
ppf <- pState( nStf, perm=c(2,1,3,4) )
head( ppf )
tail( ppf )
```

What do the entries in `ppf` represent?

22. Try to plot the cumulative probabilities using the `plot` method for `pState` objects:

```
plot( ppf )
```

Is this useful?

23. Now try to improve the plot so that it is easier to read, and easier to comapre men and women:

```
par( mfrow=c(1,2) )
plot( ppm, col=c("limegreen","red","#991111","forestgreen") )
lines( as.numeric(rownames(ppm)), ppm[,"Rem"], lwd=4 )
text( 59.5, 0.95, "Men", adj=1, col="white", font=2, cex=1.2 )
axis( side=4, at=0:10/10 )
axis( side=4, at=1:99/100, labels=NA, tck=-0.01 )
plot( ppf, col=c("limegreen","red","#991111","forestgreen"), xlim=c(60,50) )
lines( as.numeric(rownames(ppf)), ppf[,"Rem"], lwd=4 )
text( 59.5, 0.95, "Women", adj=0, col="white", font=2, cex=1.2 )
axis( side=2, at=0:10/10 )
axis( side=2, at=1:99/100, labels=NA, tck=-0.01 )
```

What is the 10-year risk of remission for men and women respectively?