Occurrence rates, cumulative risks, competing risks, state probabilities with multiple states and time scales in in Register Research with R and `Epi::Lexis`

Computer practicals

SDCG, Nuuk
1–3 March 2022
http://bendixcarstensen.com/AdvCoh/courses/Nuuk-2022
Version 3

Compiled Monday 28th February, 2022, 13:29
from: `/home/bendix/teach/AdvCoh/courses/Nuuk.2022/pracs/pracs.tex`

Lecturers:

Bendix Carstensen — Steno Diabetes Center Copenhagen, Denmark & Department of Biostatistics, University of Copenhagen
b@bxc.dk bendix.carstensen@regionh.dk
http://BendixCarstensen.com

Dorte Vistisen — Steno Diabetes Center Copenhagen, Denmark & Department of Biostatistics, University of Copenhagen
dorte.vistisen@regionh.dk

Lars Pedersen — Statistics Greenland

Johan Ejstrud — Ejstrud Consulting

Organizers:

Marit Eika Jørgensen — Steno Diabetes Center Greenland

Hjalte Erichsen — University of Greenland

Contents

Preface . . . iii
Program . . . iv

1 Using R **1**
1.1 Installing and using R . . . 1
1.2 Writing code and results . . . 1
1.2.1 Coding style in R . . . 2
1.2.2 R lingo . . . 2
1.3 Simple usage of R . . . 3
1.3.1 Using R as a calculator . . . 3
1.3.2 A functional language . . . 3
1.3.3 Sequences . . . 5
1.3.4 The births data . . . 6
1.3.5 Referencing parts of a data frame . . . 7
1.3.6 Summaries . . . 7
1.3.7 Generating new variables . . . 7
1.3.8 Logical variables . . . 8
1.3.9 Turning a variable into a factor . . . 9
1.3.10 Tables . . . 11
1.3.11 Reading data . . . 13
1.3.12 Saving data . . . 14
1.4 Graphics . . . 15
1.4.1 `ggplot2` . . . 15
1.4.2 Base graphics . . . 15
1.4.3 Simple base graphs . . . 16
1.5 Dates in R . . . 20

2 Survival and rates: `lung` **23**
2.1 Data and simple survival . . . 23
2.2 Rates and rate-ratios: Simple Cox model . . . 25
2.3 Simple Poisson model . . . 26
2.4 Representation of follow-up: `Lexis` object . . . 27
2.5 Estimating the hazard function: splitting time . . . 29

3 Competing risks: `DMlate` **35**

3.1 Data . . . 35
3.2 State probabilities . . . 37
3.3 What not to do . . . 38
3.4 Modeling cause specific rates . . . 39
3.5 Integrals with R . . . 41
3.6 Cumulative risks from parametric models . . . 44
3.7 Expected life time: using simulated objects . . . 46

4 Multistate models: `steno2` 48
4.1 `Lexis` object for `steno2` . . . 48
4.2 Transition rates: multiple time scales . . . 53
4.3 State probabilities . . . 58
4.3.1 Models for transition rates . . . 58
4.3.2 Simulation of state probabilities . . . 61
4.4 State probabilities using the Aalen-Johansen approach from `survival` . . . 71
4.5 Time spent in albuminuria states . . . 77
4.6 Clinical variables . . . 78
4.7 Several transitions from one state: `stack` . . . 81

5 Statistics Greenland 85
5.0.1 api light - saved queries . . . 85
5.0.2 for more control . . . 86
5.0.3 Example 1: pxweb (cran) . . . 87
5.1 Example 6: pxR (cran) . . . 94
5.2 Carlos J. Gil Bellosta <cgb at datanalytics.com> . . . 94

Preface

This course draws on the content of the book "Epidemiology with R" [**?**], (http://bendixcarstensen.com/EwR), and the draft of my new book (which by no means is sure ever to appear as a book) "Practical multistate modeling with R and `Epi:Lexis`". The former is available through Oxford University Press, the latter as a draft (updated at unpredictable times) as http://bendixcarstensen.com/MSbook.pdf.

- The **target audience** is the group of statisticians and epidemiologists working in or with the 5 SDCentres.
- The **prerequisites** are
  1. a very basic knowledge of R(exercises 1 is designed to get you going),
  2. a working installation of `Epi_2.44`
  3. a working installation of `popEpi_0.4.8`
  4. some epidemiological practice
- The **format** of the course will be short lectures closely aligned with the topics in the exercises. The exercises will be run in chunks between the short lectures.

Exercises are given including most of the solutions. You can get the exercise code chunks from the course website http://bendixcarstensen.com/AdvCoh/courses/Nuuk-2022

Program

Tuesday 1 March	
9:00–9:15	Welcome and practical information
9:15–10.15	L: General introduction to R
10:15–10.30	Coffee break
10:30–12:00	P: Exercises in R
12:00–13:00	Lunch
13:00–13:45	Lars Pedersen: Introduction to Statistics Greenland
13:45–14:00	Coffee break
14:00–15:30	P: Exercises in R (cont.)
Wednesday 2 March	
9:00–10:00	L: Introduction to multistate models
10:00–10.30	Coffee break
10:30–12:00	P: Survival analysis
12:00–13:00	Lunch
13:00–13:45	L: Introduction to competing risks
13:45–14:00	Coffee break
14:00–15:30	P: Cause specific rates and competing risks
Thursday 3 March	
9:00–10.00	L: Multistate models in practice
10:00–10.30	Coffee break
10:30–12:00	P: Multistate models
12:00–13:00	Lunch
13:00–13:45	Johan Ejstrud: Survival analysis in the pharmaceutical industry
14:00–15:30	P: State probabilities

Within each of the the chunks of topics (see the table of contents) there will be a short introductury lecture, introducing the practical.

Chapter 1

Using R

This introduction to R is based on chapter 1 of my book "Epidemiology with R",

The best way to learn R is to use it. Start by using it as a simple calculator, and keep on exploring what you get back by inspecting the size, shape and content of what you create.

1.1 Installing and using R

The first thing you should do is to install R on your computer so that you can start doing simple exercises.

R is available from `CRAN`, The `C`omprehensive `R` `A`rchive `N`etwork (Google it), you will find a link to installation there. If it does not work directly it may be because your administrator has placed restrictions on what you are allowed to install on your computer.

A nice interface to R is RStudio (Google it) which is a commercial product, but RStudio has a free open source license that allows you to have a very good and handy interface to R for free, including the possibility of writing reports using `Rmarkdown`, `Sweave` or `knitr`.

1.2 Writing code and results

You have probably repeatedly been told that you should comment your computer code so that you can actually remember what you intended to do with the code. And in some instances did. If you return to un-commented code more than a fortnight after it was written you will most likely be facing the problem of reverse-engineering: trying to deduce from the code what you did (and maybe even what you intended to do). That is not always a pleasant exercise, and some people end up doing the programming from scratch again. This leaves you with a number of different programs that purportedly claim to do approximately the same. But of course never does. So the coding first approach is a recipe for chaos in your code and results.

Therefore it is a good habit *first* in plain text to describe what you want to do, and only subsequently write the code that does it.

1.2.1 Coding style in R

Different people have different coding styles, that is how they place variable name, parentheses and operators relative to each other. There is no particular reason that you should take over precisely the coding style I am (trying to be) using in this book; many will disagree to some or all of my points. But you should give it a good thought because you can make your code more readable.

I have largely adhered to the following general rules in the code you see in this book, mainly for the sake of readability:

- Put spaces around the assignment operator ("`<-`")
- Let any comma be followed by a space.
- Put spaces around all operators such as `+`, `/`, etc., except around "`:`"
- Use fairly short and meaningful names for variables and objects. Very long object names makes it difficult to get the meaning of the code (and increases the likelihood of typos). This is one of the most difficult tasks in programming, but it pays to spend time on it. `long_name_proponents` do exist, though.
- Use short lines of code; a command can be broken across several line at (almost) any point. Normally it is done after a comma.
- Occasionally you may want to put more than one statement on the same line. That can be done by separating statements with a semi colon (;).
- When using braces ("`{}`") let the opening and closing braces be at the same position on the line. Putting them on a separate line each is sometimes useful. The closing brace should always be on a line of its own.
- When putting the arguments of a function on separate lines, place all arguments indented at the opening bracket of the function.
- When calling functions with many arguments, it is sometimes useful to make the equal signs between argument names and argument values vertically aligned (this is in conflict with the previous point).

Finally, keep in mind that when writing a piece of R-code it is only a *secondary* purpose to get the data processing and calculations correct; the *primary* purpose of the code is to document that what you claim to have done is actually what you did do.

1.2.2 R lingo

When talking about R, a couple of words and phrases are used frequently:

gets is the official pronunciation of the assignment operator "`<-`"

of is the official pronunciation of using a function on a argument, "f of x" meaning `f(x)`. So whenever you hear "glm of ..." you should type `glm()` and wait for what goes in between the brackets.

console the window in RStudio where the results are displayed and where you can type the occasional command you do not wish inserted in your document.

script window the window in RStudio where you type your code (or `Rmarkdown` code and text)

arguments are what is supplied to functions inside brackets. Each argument has a *name*

which is placed to the left of an "=", and a *value* which is placed to the right of it. So `name=value`. The argument names are characteristics of the function, you supply the values. These pairs are separated by commas.

package is a collection of functions (and/or datasets) that can be attached to your R-session so that you have access to the functions. `Epi` is one such package. Oddly, a package is attached (loaded) for use by the function `library()`.
Before you can do that you must install the package by `install.packages("Epi")`—that is only needed once, `library()` is needed anew whenever you restart R.

1.3 Simple usage of R

The following is intended for you to try out and also change a bit to get further insight to the objects you are manipulating. It introduces a number of basic features of R that are best demonstrated if you explore them yourself. Therefore, only some of the results of the code are shown; you only get to see the missing ones by running R yourself.

When you start R you will see a ">" at the beginning of the line in the console. When you type code in there (or transfer it from the script using `CTRL-ENTER`) R will know if you have typed a complete expression or not. If you have, you will see the result of it (if any is produced), but if you have not completed the command, the next line will have a `+` at the beginning indicating that R expects more to come.

1.3.1 Using R as a calculator

Typing `2+2` will return the answer 4, typing `2^3` will return the answer 8 (2 to the power of 3), typing `log(10)` will return the natural logarithm of 10, which is 2.3026, and typing `sqrt(25)` will return the square root of 25.

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

```
> a <- 2 + 2
```

. . . and you can actually also do:

```
> 2 + 2 -> a
```

The contents of the object `a` can be printed by typing `a`. Try that.

1.3.2 A functional language

R is a *functional* language; everything you ever do is to call a function that transforms something to something else and possibly assigns it or just prints it, try for example:

```
> x <- 1:10
> x
```

There does not seem to be any functions here? The first statement actually uses the function ":" which takes two arguments, in this case `1` and `10` and returns a sequence of numbers with distance 1 and assigns it to `x` (—you can actually write `":"(1, 10)` if you wish). The second statement implicitly invokes the `print` function to print the vector `x`. Using a function on `x` without assigning it will automatically invoke the print function and print it on your screen (console).

From a practical point of view what you do is that you create a vector of the number 1 to 10 and store it in a so-called *object* called `x`, so you can access it later. For example printing it by just typing its name as above.

A couple of simple functions are:

```
> sum(x)
> sd(x)
> diff(x)
> cumsum(x)
> rev(x)
> prod(x)
> x > 7
> x >= 7
```

Try them and find out what they do.

Exercises:

1. Calculate $\sqrt{3^2 + 4^2}$.
2. Find the probability above 4.3 in a chi-squared distribution on 1 degree of freedom.

Objects and functions

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 an ordered collection of character strings. Examples of vectors are (4, 6, 1, 2.2), which is a numeric vector with 4 components, and ("Charles Darwin", "Alfred Wallace") which is a vector of character strings with 2 components. 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 by first combining the 4 numbers 4, 6, 1, 2.2 in order and then assigning the result to the vector `v`.

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

```
> m <- list(4:7, six = 6, "name of company")
> m
[[1]]
[1] 4 5 6 7

$six
[1] 6

[[3]]
[1] "name of company"
```

creates a `list` with 3 components. `list`s allows elements of different kinds, in this case two numeric vectors (length 4 an 1) and a character vector; and in this case the second element is named.

The main differences between the numbers 4, 6, 1, 2.2 and the vector `v` is that along with `v` is stored information about what sort of object it is and hence how it is printed and how it is combined with other objects, try:

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

and you will see that R understands what to do in each case. This may seem trivial, but remember that unlike most statistical packages there are many different kinds of object in R.

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.

What makes R different: functions

R also gives you the possibility of writing your own functions; they need not be very fancy, nor do they need to have any arguments. In this book we will frequently use probabilities π and *odds*, $\omega = \pi/(1-\pi)$ and so we will want to be able to convert easily from one to another. This can be done by defining functions for the conversions:

```
> p2o <- function(p) p / (1 - p)
> o2p <- function(o) o / (1 + o)
```

These functions will convert between probabilities and odds:

```
> p2o(0.25)
> o2p(8)
```

What do you think you get if you write `o2p(p2o(0.25))`?

A function in R is defined by `function` and the value returned by the function is the value of the *last* statement. To make it a bit more clear how a function is defined we could have written:

```
> p2o <-
+ function(p)
+   {
+   odds <- p / (1 - p)
+   odds
+   }
```

The function is defined by naming the *arguments* (what is between the `()`s—in this case one, `p`), and then defining what to be computed from these in the *body* of the function (what is between the `{}`s). The *value* of the function when called with appropriate argument(s) is the value of the last expression in the function body, in this case just "`odds`".

1.3.3 Sequences

It is not always necessary to type out all the components of a vector to create one. For example, the vector (15, 20, 25, ..., 85) can be created with:

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

and the vector (5, 20, 25, ..., 85) can be created with

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

It is also possible to repeat vectors in complex patterns, try:

```
> rep(c(3, 2, 7), c(1, 4, 3))
> rep(c(3, 2, 7), 5)
> rep(c(3, 2, 7), each = 5)
```

A particularly simple form of a sequence is on where the step length is 1; this is created by ":":

```
> 7:10
> 8:3.5
```

```
> 3.7:8.1
```

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

Exercises:

1. Create a vector `w` with components $1, -1, 2, -2$
2. Print this vector (to the screen)
3. Obtain a description of `w` using `str()`
4. Create the vector `w+1`, and print it.
5. Create the vector (0, 1, 5, 10, 15, ... , 75) using c() and seq().
6. Create a vector with 20 elements equally spaced between 7 and 23

1.3.4 The births data

The most important example of a vector in epidemiology is the data on a variable recorded for a group of subjects. A collection of these can be put side-by-side to form a data set, in R called a `data.frame`. As an example we shall use the births data which concern 500 mothers who had singleton births in a large London hospital. These data are available as an R `data.frame` called `births` in the `Epi` package.

The easiest way to access the births data is first to load the `Epi` package with

```
> library(Epi)
```

and then to load the data with

```
> data(births)
```

You get an overview from the `Epi` package documentation of the data set by:

```
> ?births
```

Some of the variables which make up these data take integer values while others are numeric taking measurements as values. For most variables the integer values are just codes for different categories, such as `"male"` and `"female"` which are coded 1 and 2 for the variable `sex`.

The function

```
> str(births)
```

shows that the object `births` is a data frame with 500 observations of 8 variables. The names and types of the variables are also shown together with the first couple of values of each variable.

Exercises:

1. The data frame `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.
2. Check that you now have two objects, `births`, and `diet` in your work space, using `ls()` or the `lls()` from `Epi`.
3. Obtain a description of the object `diet`.
4. Remove the object `diet` with the command
   ```
   > rm(diet)
   ```
   Check that you only have the object `births` left in your workspace.

1.3.5 Referencing parts of a data frame

Typing `births` will list the entire data frame - not usually very helpful. Now try
```
> births[1, "bweight"]
```
This will list the value taken by the first subject for the `bweight` variable. 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 persons for the `bweight` variable, try
```
> births[1:10, "bweight"]
```
and to list all the data for this variable, try
```
> births[, "bweight"]
```
An alternative way of referring to a variable in a data frame is using the "`$`"
```
> births$bweight
```

Exercises:

1. Print the data on the variable `gestwks` for subject 7 in the `births` data frame.
2. Print all the data for subject 7.
3. Print all the data on the variable `gestwks`.

1.3.6 Summaries

A good way to start an analysis is to ask for a summary of the data by typing
```
> summary(births)
```
To see just the names of the variables in the data frame try
```
> names(births)
```
A bit more information is obtained by
```
> str(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)
```
So you see that `summary` behaves differently when you supply a data frame and vector to it.

In most datasets there will be some missing values. The summary shows the number of missing values for each variable, indicated by `NA` (Not Available).

1.3.7 Generating new variables

New variables can be produced using assignment together with the usual mathematical operations and functions:

`+` `-` `*` `/` `^` `sqrt` `log` `exp`

The sign `^` means "to the power of", `sqrt(x)` means "square root of x", $\sqrt{x}$. `log` means "natural logarithm".

The `transform` function allows you to transform or generate variables in a data frame. For example, try:
```
> births <- transform(births,
+                     num1 = 1,
```

```
+                       logbw = log(bweight),
+                         avg = bweight / gestwks)
```

The variable `logbw` is the natural logarithm of birth weight, and `avg` is the birth weight per gestational week.

`dplyr`

The package `dplyr` provides a slightly different syntax for the same using the pipe operator, `%>%`, to indicate that first we have `births`, and then we subject it to a mutation:

```
> library(dplyr)
> bth <- births %>% mutate( num1 = 1,
+                           logbw = log(bweight),
+                             avg = bweight / gestwks)
```

More logically, we might put the assignment of the result at the end to indicate that the assignment comes after the mutation:

```
> births %>% mutate( num1 = 1,
+                    logbw = log(bweight),
+                      avg = bweight / gestwks) -> bth
```

All three sets of code will produce the same result, namely the births data frame with three extra variables. The `mutate` function is however more versatile; for example, it allows further calculations on variables defined inside `mutate`, which `transform` does not.

1.3.8 Logical variables

Logical variables take the values `TRUE` or `FALSE`, and behave mostly like factors. New variables can be created which are logical functions of existing variables. For example

```
> low <- births$bweight < 2000
> str(low)
```

creates a logical variable `low` with levels `TRUE` and `FALSE`, according to whether `bweight` is less than 2000 or not. The logical expressions which R allow are:

```
!      ==      <      <=      >      >=      !=
```

The first is logical negation, the second equals and the last is logical *not* equals. One common use of logical variables is to restrict a command to a subset of the data. For example, to list the values taken by `bweight` for hypertensive women, try

```
> births$bweight[births$hyp == 1]
```

If you want the entire data frame restricted to hypertensive women try:

```
> births[births$hyp == 1, ]
```

The `subset()` function allows you to take a subset of a data frame. Try

```
> subset(births, hyp == 1)
```

You can check whether birth weight is smaller than 2500 grams among the first 10 births:

```
> births$bweight[1:10] < 3000
 [1]  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

and you can also find out where the `TRUE` values are:

```
> which(births$bweight[1:10] < 3000)
[1] 1 3
```

Caveat: You cannot use `TRUE` or `FALSE` as names of variables. But you can abbreviate `TRUE` and `FALSE` as `T` and `F`, and you *can* use `T` and `F` as variable names. If you do can get almost impenetrable errors or, even worse, undetected misbeaviour, some very hard to find. So: Never call a variable `T` or `F`, and always use the full form `TRUE` and `FALSE`.

Exercises:

1. Create a logical variable called `early` according to whether `gestwks` is less than 30 or not. Make a frequency table of `early` using `table`.
2. Print the `id` numbers of women with `gestwks` less than 30 weeks.

1.3.9 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. But we would never want to do calculations on these numerical values; they would only ever be used to indicate a category.

For R to recognize that the codes refer to categories it is necessary to convert the variables to be factors, and in order to make code and results human readable also to label the levels. To convert the variable `hyp` to be a factor, try

```
> hyp <- factor(births$hyp)
> lls()
```

The latter shows that `hyp` is both in your work space (as a factor), and in the `births` data frame (as a numeric variable). It is better to use the `transform` function on the data frame, so that the `hyp` variable in the data frame is converted to a factor:

```
> births <- transform(births, hyp = factor(hyp))
> str(births)
'data.frame':        500 obs. of  11 variables:
 $ id     : num  1 2 3 4 5 6 7 8 9 10 ...
 $ bweight: num  2974 3270 2620 3751 3200 ...
 $ lowbw  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ gestwks: num  38.5 NA 38.2 39.8 38.9 ...
 $ preterm: num  0 NA 0 0 0 0 0 0 0 0 ...
 $ matage : num  34 30 35 31 33 33 29 37 36 39 ...
 $ hyp    : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1 1 ...
 $ sex    : num  2 1 2 1 1 2 2 1 2 1 ...
 $ num1   : num  1 1 1 1 1 1 1 1 1 1 ...
 $ logbw  : num  8 8.09 7.87 8.23 8.07 ...
 $ avg    : num  77.2 NA 68.7 94.2 82.3 ...
```

which shows that `hyp`, in the `births` data frame, is now a factor with two levels, labeled "`0`" and "`1`"—the original values taken by the variable. It is better to assign labels as (say) `"normal"` and `"hyper"` with:

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

You may want a different order than the numerical defaults of the levels; one way of achieving this is using the `levels` argument:

```
> births <- transform(births,
+                     early = factor(preterm,
+                                    levels = c(1, 0),
+                                    labels = c("Pre", "Norm")))
> with(births, table(preterm, early))
```

The naming of the arguments is a bit odd, `levels` refer to the *in*coming values (of `preterm`) and `labels` to the *out*going values (in `early`). However, if you afterwards want to know

what values the factor assumes, we refer to these as the `levels` of the factor:

```
> levels(births$early)
```

Internally, the factor levels are stored as the integers 1, 2, . . . , and the (names of the) levels of the factor in a separate structure. That way the names of the levels are only stored once, saving space.

Manipulating factor levels

When producing tables you may want to have levels of a factor in a specific order or even combine some of the levels. Using the dataset diet, try:

```
> data(diet)
> table(diet$job)
     Driver   Conductor Bank worker
        102          84         151
> table(relevel(diet$job, 2))
  Conductor      Driver Bank worker
         84         102         151
> table(relevel(diet$job, "Bank worker"))
Bank worker      Driver   Conductor
        151         102          84
> table(Relevel(diet$job, 3:1))
Bank worker   Conductor      Driver
        151          84         102
> table(Relevel(diet$job, list(3, 1:2)))
     Bank worker Driver+Conductor
             151              186
```

The base R function `relevel` (lower case) only has the capability of moving a specific level of the factor up as the first — a facility which is handy in regression modeling. The `Epi` function `Relevel` (capitalized) allows combination of factor levels too.

`Relevel` also allows grouping via a look-up in a table — try

```
> example(Relevel)
```

to see examples of this.

If you take a subset of a data frame, you may end up with a factor that has a levels that is not assumed:

```
> subdiet <- subset(diet, job != "Driver")
> table(subdiet$job)
```

In some contexts this may be impractical; the way to get rid of the non-used levels is by using `factor`:

```
> table(factor(subdiet$job))
```

Exercises:

1. In the `births` data frame, convert the variable `sex` into a factor.
2. Label the levels of `sex` as `"M"` and `"W"`.
3. In the `diet` dataset, combine levels `Driver` and `Conductor` to a level called `Bus employee`.

Grouping values of a quantitative variable

For a numeric variable like `matage` it is occasionally useful to group the values and to create a new factor representing the grouping. This should only be used for exploration of data; modeling of effects of a quantitative variable should *never* be based on a grouping,

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`:

```
> births <- transform(births, agegrp = cut(matage,
+                                          breaks = c(25, 30, 35, 40, 45),
+                                           right = FALSE))
> table(births$agegrp, exclude = NULL)
[25,30) [30,35) [35,40) [40,45)    <NA>
     68     200     194      36       2
```

CODE EXPLAINED: `transform` is used to define a new variable (a factor), `agegrp` in the `births` data frame. The argument `right` is a logical indicating whether the right endpoint should be included in each interval; we want the left endpoint to be included, so we set it to `FALSE`. Persons with a value of `matage` less than 25 or larger than 45 will be transformed to `NA`. `table` will ignore `NA`s, unless instructed to include everything by `exclude=NULL`.

By default the factor levels are labeled [20-25), [25-30), etc., where [20-25) refers to the interval which includes the left end (20) but not the right end (25). This was brought about by using the argument `right=FALSE`. When `right=TRUE` (which is the default) the intervals include the right end but not the left.

It is important to realize that observations which are not inside the range specified in the `breaks()` part of the command result in missing values for the new factor. For example:

```
> births <- transform(births, agegrp=cut(matage,
+                                        breaks = c(20, 30, 35),
+                                         right = FALSE))
> summary(births$agegrp)
```

Only observations from 20 up to, but not including 35, are included. For the rest, `agegrp` is coded missing. This will not immediately show up if you use `table`, but the argument `exclude=NULL` will remedy this; try:

```
> table(births$agegrp)
> table(births$agegrp, exclude = NULL)
> addmargins(table(births$agegrp, exclude = NULL))
```

`addmargins` adds margins to any type of a table; it can be any type of margins, not only sums (which is the default).

Exercises:

1. Summarize the numeric variable `gestwks`, which records the length of gestation for the baby, and make a note of the range of values.
2. 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`.
3. Create a new factor `gest5` which cuts `gestwks` into 5 equal intervals, and make a table of frequencies.

1.3.10 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. A very useful function for making

tables is `stat.table` from the `Epi` package.

The distribution of the factors `hyp` and `sex` can be viewed by typing

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

Their cross-tabulation is obtained by typing

```
> stat.table(list(hyp, sex), data = births)
 ----------------------
      -------sex-------
hyp          1       2
 ----------------------
 0          221     207
 1           43      29
 ----------------------
```

Cross-tabulations are useful when checking for consistency, but because no distinction is drawn between the response variable and any explanatory variables, they are not necessarily useful as a way of presenting data, and as you see, rather meaningless if the variables you tabulate are not properly labeled factors.

Tables of means and other things

To obtain the mean of `bweight` by `sex`, try

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

The headings of the table can be improved with

```
> stat.table(sex,
+            list("Mean birth weight" = mean(bweight)),
+            data = births)
```

To make a two-way table of mean birth weight by sex and hypertension, first convert `sex` and `hyp` to factors for readability.

```
> births <- transform(births, sex = factor(sex, labels = c("M", "W")),
+                             hyp = factor(hyp, labels = c("No", "Yes")))
> stat.table(list(sex, hyp),
+             mean(bweight),
+             margins = TRUE,
+                data = births)
```

and to tabulate the count as well as the mean, including the margins:

```
> stat.table(list(sex, hyp),
+             list(count(),
+                  mean(bweight)),
+             margins = TRUE,
+                data = births)
```

Available functions for the cells of the table are `count, mean, weighted.mean, sum, min, max, quantile, median, IQR`, and `ratio`. The last of these is useful for rates and odds. For example, to make a table of the odds of low birth weight by hypertension, try

```
> stat.table(hyp,
+            list("odds" = ratio(lowbw, 1 - lowbw, 100)),
+            data = births)
```

The scale factor 100 makes the odds per 100, so essentially %. Margins can be added to the tables, as required. For example, you will do

```
> stat.table(sex,
+            mean(bweight),
```

```
+            margins = TRUE,
+               data = births)
```

for a one-way table. For a two-way table, you can try;

```
> stat.table(list(sex, hyp),
+             mean(bweight),
+             margins = c(TRUE, FALSE),
+                data = births)
> stat.table(list(sex, hyp),
+             mean(bweight),
+             margins = TRUE,
+                data = births)
```

Exercises:

1. Make a table of median birth weight by `sex`.
2. Do the same for gestation time, but include `count` as a function to be tabulated along with `median`. Note that when there are missing values for the variable being summarized the count refers to the number of non-missing observations for the row variable, not the summarized variable.
3. Create a table showing the mean gestation time for the baby by `hyp` and `lowbw`, together with margins for both.
4. Make a table showing the odds of hypertension by sex of the baby.

1.3.11 Reading data

R can read data from many different formats, the functions for reading various data formats are found in a number of different packages. So remember to read the documentation, there are many pitfalls, and since this book is not about data no comprehensive overview is given here. Reading data without reading the documentation of the function you use to read data is a prescription of erroneous data.

When reading data, a number of points should be kept in mind that may give rise to funny data if forgotten:

- Variable names — are they in the first line of the data file?
- How are missing values coded?
- How are categorical variables (factors) coded?
- How are dates represented?
- What is the decimal separator?

Different function for reading data will handle these issues differently, and most will have a large number of arguments that control how data is read.

The following functions will cover many needs you may have:

- Plain files with spaces separating variables, use `read.table`, for example:

  ```
  > fem <- read.table("http://bendixcarstensen.com/SPE/data/fem.dat",
  +                   header = TRUE,
  +                 na.strings = c("-99", "NA"))
  ```

  As you see, R will recognize a URL and read directly from it. In the file, the first line contains the variable names, and missing values are represented either by `-99` or `NA`.
- Comma-separated files, `.csv`, use the function `read.csv` or `read.csv2` depending on whether the file is with comma or semicolon as separator.
- Clipboard: A quick and dirty way to get in a small chunk of data is to highlight the

data on your screen (*e.g.* in Excel) and press CTRL-C ("copy"). The data is then placed on your clipboard. You can then just do:
```
> qad <- read.table("clipboard")
```
—but you will still have the all the issues with missing data representation etc.

- Data from other statistical packages such as SAS or Stata: Use the functionalities in the `haven` package:
  ```
  > help(package = haven)
  ```
  The package `haven` also contains facilities to *write* data in formats for other statistics pages.
- Excel files, use the package `xlsx`, see `help(package = xlsx)` to obtain more information.
- SQL databases: use the package `RODBC`, see `help(package = RODBC)` to obtain more information.

1.3.12 Saving data

Saving the work space

When exiting from R you are offered the chance of saving all the objects in your current work space. If you do so, the work space is re-instated next time you start R. It is only occasionally useful to do this, but if you choose to do so it is worth tidying things up, because the work space can fill up with temporary objects, and it is easy to forget what these are when you resume the session.

The general advice is *not* to save the workspace.

Saving R objects in a file

The command `read.table()` is relatively slow because it carries out quite a lot of processing as it reads the data. To avoid doing this more than once you can save the data frame, which includes the R information, and read from this saved file in future. For example,
```
> save(births, file = "births.Rda")
```
will save the births data frame in the file `births.Rda`. By default the data frame is saved as a binary file, but the option `ascii=TRUE` can be used to save it as a text file. You can save more than one object in an R-file, they need not be data frames, they can be fitted models for example:
```
> save(births, p2o, o2p, file = "births.Rda")
```
To load the object(s) from an `.Rda` file, use:
```
> load("births.Rda", verbose=TRUE)
```
The commands `save()` and `load()` can be used with any R objects, but they are particularly useful when dealing with large data frames. The `verbose` argument lists the names of the objects loaded.

Using `with`

It is quite tedious to write `births$` in front of every variable name used. One way of avoiding this is to wrap the expressions in `with`, such as:
```
> with(births, plot(gestwks, bweight))
```

The first argument is a data frame, the second argument is an expression where variable names are assumed to come from the data frame. You can use other variable names too, they will be taken from the global environment,

1.4 Graphics

There are two main graphics systems used in R: Base graphics, which is an integral part of any R distribution, and `ggplot2` (gg referring to grammar of graphics) which is a separate package that you need to install, which has a different syntax, and is not compatible with base graphics. `ggplot2` is part of the `tidyverse` packages.

Besides these two there is also `lattice` graphics that allows quite elaborate graphs of multidimensional structures, however at the price of quite a complicated interface.

1.4.1 ggplot2

The grammar of graphics underlies the package `ggplot2`, which defines graphs as graphical objects (`grob`s) that can be modified by adding different aspects of the graph such as themes.

It is not as easy to master as base graphics, but the graphs (particular multiframe displays) will be more consistent. However, this graphical system is an entire (large) topic of its own, and will not be treated in any detail in this book; a few examples of its use will be shown though. The `ggplot2` package is part of the `tidyverse` environment, see section **??** on page **??**.

1.4.2 Base graphics

The plotting model of base graphics is emulating your pencil (or fountain pen): ink on paper. Each command in base graphics puts something on the graph, and you cannot remove it. If you get it wrong, you will have to start over—which is not so bad, you just run the code again. Unless you are typing along in the console window—do not do that

If you just issue plot commands, the graph will appear on the screen; if you want to put the graph in a particular file, you must open a graphics *device* before the plotting commands, and close it afterwards. For example, if you want a plot in a `pdf`-file you will open the `pdf` device using `pdf()` and close it using `dev.off()`:

```
> pdf("a_graph.pdf")
> x <- seq(1, 5, 0.01)
> plot(x, (x - 2) * (x - 4))
> dev.off()
```

This will create the file `a_graph.pdf` in your current directory (if you do not know which that is, use `getwd()`)

You can get a list of available devices by:

```
> ?Devices
```

(must be a capital `D`).

Sometimes the default graph window in RStudio is too small to hold your graph. You can open another graph window outside of RStudio by:

```
> RStudioGD()
```

(RStudioGgraphicsDevince). Your graphs will then go there and you can just swap to this the usual way (using Alt-Tab, *i.e.* holding down the Alt key and repeatedly pressing the Tab-key, and releasing the Alt once you have found your graph window).

1.4.3 Simple base graphs

There are three kinds of plotting functions in base graphics:

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()`. We will not go into these.

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.

Plot on the screen

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

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

Now attach the data frame and look at the birth-weight distribution with

```
> attach(births)
> hist(bweight)
```

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

```
> ?hist
```

and try some of the options, for example:

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

To look at the relationship between birth-weight and gestational weeks, try

```
> 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)
```

or, using the `rep` function to generate a grid of points:

```
> plot(rep(1:5, 5), rep(1:5, each = 5), pch = 1:25,
+      cex = 5, xlim = c(0, 6), ylim = c(0, 6), lwd = 4)
> text(rep(1:5, 5) + 0.3, rep(1:5, each = 5), 1:25)
```

Exercises:

1. Make a plot of the birth weight versus maternal age with
   ```
   > plot(matage, bweight)
   ```
2. Label the axes with
   ```
   > plot(matage, bweight, xlab="Maternal age", ylab="Birth weight (g)")
   ```

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 birth-weight versus gestational weeks, try

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

This creates a solid mass of colour in the center 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.

```
> points(gestwks, bweight)
```

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

```
> rainbow(4)
```

produces a vector with 4 colours (not immediately human readable, though). There other functions that generates other sequences of colours, type `?rainbow` to see them.

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)
```

Colours can be given explicitly in the RGB-space (red, green, blue) as a character string `"#RRGGBB"` where `R`, `G` and `B` are hexadecimal[1] digits (0–9, A–F).

There is a number of functions in base R to manipulate colours, try for example:

```
> col2rgb("orange")
> rgb(t(col2rgb("orange")), m = 256)
```

There is also the possibility of generating semi-transparent colours, using for example `adjustcolor`. This is used in the function `matshade` that plots confidence bands as shaded areas.

Some thought has been put into constructing functions that generate sequences of colours useful in more advanced graphs; two such packages are `RColorBrewer` and `viridis`. It is left to you to explore these further, try for example

```
> help(package = RColorBrewer)
```

Adding to a plot

As we just saw, `points()` 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 birth weight vs. gestational weeks using different colours for male and female babies. To start with an empty plot, try:

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

Even if nothing is plotted, the axes are constructed so that all points will be contained in the plot.

Then we can add the points with the `points` function:

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

This should put 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] Refers to the base 16 representation of numbers using digits 0–9, A–F, with A representing 10, F representing 15 and, say, 1B representing 16+11=27. A two-digit hexadecimal number can represent the numbers from 0 through 255 ($16^2 - 1$)

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 just two sexes.

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 a look at `sex`:

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

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

```
> c("blue", "red")[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:

```
> 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 under 30 and over 35, say. We first generate the indexing variable as a factor

```
> magr <- cut(matage, c(0, 30, 35, 100))
> table(magr)
```

`magr` is now a factor with 3 levels, and indexing with the variable is the same as indexing with the numerical representation of the factor, 1, 2, 3; so we ask for symbols 15, 16, 17 according to the age-class of the mother. Moreover, in the specification of the legend we can just use the generated levels as text.

```
> plot(gestwks, bweight,
+      pch = (15:17)[magr], col = c("blue", "red")[sex])
> legend("topleft", pch = 15:17, legend = levels(magr), col = 1, bty = "n")
> text(28, 4200+0:1*200, c("Boys", "Girls"),
+      col = c("blue", "red"), adj = 0)
```

Note that we generated the legend for the colors by simply using `text` to write "Boys" resp. "Girls" in blue and red.

R will accept any kind of complexity in the indexing as long as the result is a valid index, including a factor.

Saving graphs for use in other documents

Once you have a graph in the graphics window in RStudioyou can click on `Export` and choose the format you want your graph in. The `pdf` (Acrobat reader) has a button of its own, `.pdf` normally the most economical, and Acrobat reader has good options for viewing in more detail on the screen.

The `win.metafile` format will give you an enhanced metafile `.emf`, which can be imported into a Word document. Metafiles can be re-sized and edited inside Word; they are in a vector graphics format as are `.pdf` and `.eps`, which means they do not get woolly when enlarged, as do bitmap formats `tiff`, `bmp`, `jpg` and `png`.

If you want precise control over the size of your plot-file you can start a graphics device *before* doing the plot. Instead of appearing on the screen, the plot will 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 = "plot1.pdf", height = 3, width = 4)
> plot(gestwks, bweight)
> dev.off()
```

This will give you a `pdf` file `plot1.pdf` with a graph which is 3 inches tall and 4 inches wide. Similarly:

```
> win.metafile(file="plot1.emf", height=3, width=4)
> plot(gestwks, bweight)
> dev.off()
```

will give you a `emf` file `plot1.emf` with a graph which is 3 inches tall and 4 inches wide. This is a vector graphics file that can be inserted in a Word document, and which can be modified in Word.

The `win.metafile` is only available on windows systems, for other systems use the device `emf` from the `devEMF` package.

Same graph on multiple devices

If you want the same graph in different file types (or in slightly different aspect ratios), a simple way is to exploit the function facility in R and put the entire plot code into a function with no arguments, and the call the function when different devices are open as in the following example:

```
> myplfn <- function() # Define the function that does the plot
+ {
+ plot(gestwks, bweight,
+      pch = (15:17)[magr], col = c("blue", "red")[sex])
+ legend("topleft", pch = 15:17, legend = levels(magr), col = 1, bty = "n")
+ text(28, 4200+0:1*200, c("Boys", "Girls"),
+      col = c("blue", "red"), adj = 0)
+ }
> #
> # on the screen
> myplfn()
> #
> # pdf graph
> pdf("plot1.pdf", height = 8, width = 10)
> myplfn()
> dev.off()
> #
> # windows meta file
> win.metafile("plot1.eps", height = 8, width = 10)
> myplfn()
> dev.off()
```

This has the advantage that if you want to change the plot a little, you only edit the code in one place and all plots will be revised accordingly.

The `par()` command

It is possible to manipulate almost any element in a graph, by using the graphics options. These are collected in the function `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. No one promised you that things should be intuitively clear.

It is a good idea to take a print of the help page for `par` (having set the font 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., and perhaps even put under your pillow at night. Do not 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. With logarithmic axes it's not immediately obvious what you get, you need to read the help page for `par`.

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 column-wise, 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))
```

A more versatile machinery for putting multiple graphs on a page in almost arbitrary (rectangular, though) layouts is the function `layout`—not treated further here.

1.5 Dates in R

Epidemiological studies often contain date variables which take values such as 2/11/1962. We shall use the `diet` data to illustrate how to deal with variables whose values are dates.

The important variables in the dataset are `chd`, which takes the value 1 if the subject develops coronary heart disease during the study, and the value 0 if the observation is censored, and the three date variables which are date of birth (`dob`), date of entry (`doe`) and date of exit (`dox`). The command

```
> data(diet)
> str(diet)
```

shows that these three variables are `Date` variables; if you try

```
> head(diet)
```

you will see these variables printed as "real" dates. The variables are internally stored as number of days since 1/1/1970.

To convert a character string (or a character variable or factor) to date format try:

```
>            as.Date("14/07/1952", format = "%d/%m/%Y")
> as.numeric(as.Date("14/07/1952", format = "%d/%m/%Y"))
```

The first statement shows the date form and the latter the number of days since 1/1/1970, which is a negative number for dates prior to 1/1/1970.

The format parts, "`%d`" etc., identify elements of the dates, whereas the "`/`"s are just the separator characters that are in the character string. There is a large number of possibilities for formats, see `?strftime`.

Reading dates from an external file is done by reading the fields as character variables and

then transforming them to date variables by the function `as.Date`, using the the relevant format. It will also work if your date variables accidentally ended as factors.

If you want to enter a fixed date, for example if you want to terminate follow-up at 1st April 1995 you could say:

```
> newx <- pmin(diet$dox, as.Date("1995-4-1", format="%F"))
```

The format `%F` is shorthand for the ISO-standard date representation `%Y-%m-%d`, which is the default, so it can be omitted altogether:

```
> newx <- pmin(diet$dox, as.Date("1975-4-1"))
```

You will get `NA`s if your dates are not correct:

```
> as.Date(c("1997-02-28", "1997-02-29", "1997-13-22"))
```

You can have other separators than "-", even quite silly ones:

```
> as.Date("1995$4$1", format = "%Y$%m$%d")
> as.Date("1995sep4DIV1", format = "%Ysep%mDIV%d")
```

You can print dates in the format you like by using the function `format` (really `format.Date`), try for example:

```
>   bdat <- as.Date("1952-7-14", format = "%F")
>   format(bdat, format = "%A %d %B %Y")
```

In practical epidemiological analyses it is more convenient to use time measured in years than in days, so the `Epi` package has a function `cal.yr` that converts dates to numeric years

```
> (dd <- as.Date(c('1970-1-1',
+                  '1971-1-1',
+                  '1972-1-1',
+                  '1973-1-1',
+                  '1974-1-1',
+                  '1975-1-1')))
[1] "1970-01-01" "1971-01-01" "1972-01-01" "1973-01-01" "1974-01-01"
[6] "1975-01-01"
> cal.yr(dd)
[1] 1970.000 1970.999 1971.999 1973.001 1974.000 1974.999
attr(,"class")
[1] "cal.yr"  "numeric"
```

Because of the leap-years it is only every 4^{th} year 1 January precisely fits with an integer. Formally, the `cal.yr` converts dates (measured in units of days) to units of 365.25 days, and we just choose to call this unit "year". The conventional use of "year" is formally inaccurate, because a year sometimes is 365 and sometimes 366 days.

You can also see that the differences between the dates are not the same, neither measured in days or "years" of course.

```
> diff(dd)
> diff(cal.yr(dd))
```

On the other hand if you take dates that have a given distance in days you get consistency:

```
> (xx <- as.Date("1970-1-17") + 0:5 * 300)
[1] "1970-01-17" "1970-11-13" "1971-09-09" "1972-07-05" "1973-05-01"
[6] "1974-02-25"
> diff(cal.yr(xx))
[1] 0.8213552 0.8213552 0.8213552 0.8213552 0.8213552
attr(,"class")
[1] "cal.yr"  "numeric"
```

In addition, `cal.yr` has the facility that with a data frame as argument it will find all `Date` variables in the data frame and convert them to `cal.yr` format, and return the data frame with the converted variables; try:

```
> data(diet)
```

```
> diet[1:4, 1:4]
   id        doe        dox        dob
1 102 1976-01-17 1986-12-02 1939-03-02
2  59 1973-07-16 1982-07-05 1912-07-05
3 126 1970-03-17 1984-03-20 1919-12-24
4  16 1969-05-16 1969-12-31 1906-09-17
> food <- cal.yr(diet)
> food[1:4, 1:4]
   id      doe      dox      dob
1 102 1976.042 1986.917 1939.164
2  59 1973.537 1982.507 1912.508
3 126 1970.205 1984.215 1919.977
4  16 1969.370 1969.997 1906.709
> str(food[1:4])
'data.frame':        337 obs. of  4 variables:
 $ id : num  102 59 126 16 247 272 268 206 182 2 ...
 $ doe: 'cal.yr' num  1976 1974 1970 1969 1968 ...
 $ dox: 'cal.yr' num  1987 1983 1984 1970 1979 ...
 $ dob: 'cal.yr' num  1939 1913 1920 1907 1919 ...
```

Exercises:

1. Generate a new variable `y` which is the elapsed time in years between the date of entry and the date of exit.
2. Enter your own birthday as a date. Print it using `format.Date()` with the format `"%A %d %B %Y"`. Did you learn anything new?
3. Print your birthday in `cal.yr` format.
4. Enter the birthday of your husband/wife/... as a date too. When will you be (or were you) 100 years old together? (Hint: `mean()` works on vectors of dates as well.)

Chapter 2

Survival and rates: `lung`

Paraphernalia

It is advisable to load all packages needed at the start:

```
> library(survival)
> library(Epi)
> library(popEpi)
> # popEpi::splitMulti returns a data.frame rather than a data.table
> options("popEpi.datatable" = FALSE)
> clear()
```

2.1 Data and simple survival

1. Load the `lung` data from the `survival` package, and convert `sex` to a factor (*always* do that with categorical variables). Also we rescale time from days to months:

```
> data(lung)
> lung$sex <- factor(lung$sex,
+                    levels = 1:2,
+                    labels = c("M", "W"))
> lung$time <- lung$time / (365.25/12)
> head(lung)
  inst      time status age sex ph.ecog ph.karno pat.karno meal.cal wt.loss
1    3 10.053388      2  74   M       1       90       100     1175      NA
2    3 14.948665      2  68   M       0       90        90     1225      15
3    3 33.182752      1  56   M       0       90        90       NA      15
4    5  6.899384      2  57   M       1       90        60     1150      11
5    1 29.010267      2  60   M       0      100        90       NA       0
6   12 33.577002      1  74   M       1       50        80      513       0
```

2. Use `survfit` to construct the Kaplan-Meier estimator of overall survival:

```
> ?Surv
> ?survfit
> km  <- survfit(Surv(time, status == 2) ~ 1, data = lung)
> km
Call: survfit(formula = Surv(time, status == 2) ~ 1, data = lung)

      n  events  median 0.95LCL 0.95UCL
 228.00  165.00   10.18    9.36   11.93
> # summary(km) # very long output
```

The standard print method just prints the number of events and the median survival, while the `summary` prints the entire survival function estimate.

We can plot the survival curve—this is the default plot for a `survfit` object:

```
> plot(km)
```

What is the median survival? What does it mean?

3. Explore if survival patterns between men and women are different:

```
> kms <- survfit(Surv(time, status == 2) ~ sex, data = lung)
> kms
Call: survfit(formula = Surv(time, status == 2) ~ sex, data = lung)

        n events median 0.95LCL 0.95UCL
sex=M 138    112   8.87    6.97    10.2
sex=W  90     53  14.00   11.43    18.1
```

We can plot the two resulting survival curves with confidence limits:

```
>  plot(kms, col = c("blue", "red"), lwd = 1, conf.int = TRUE)
> lines(kms, col = c("blue", "red"), lwd = 3)
```

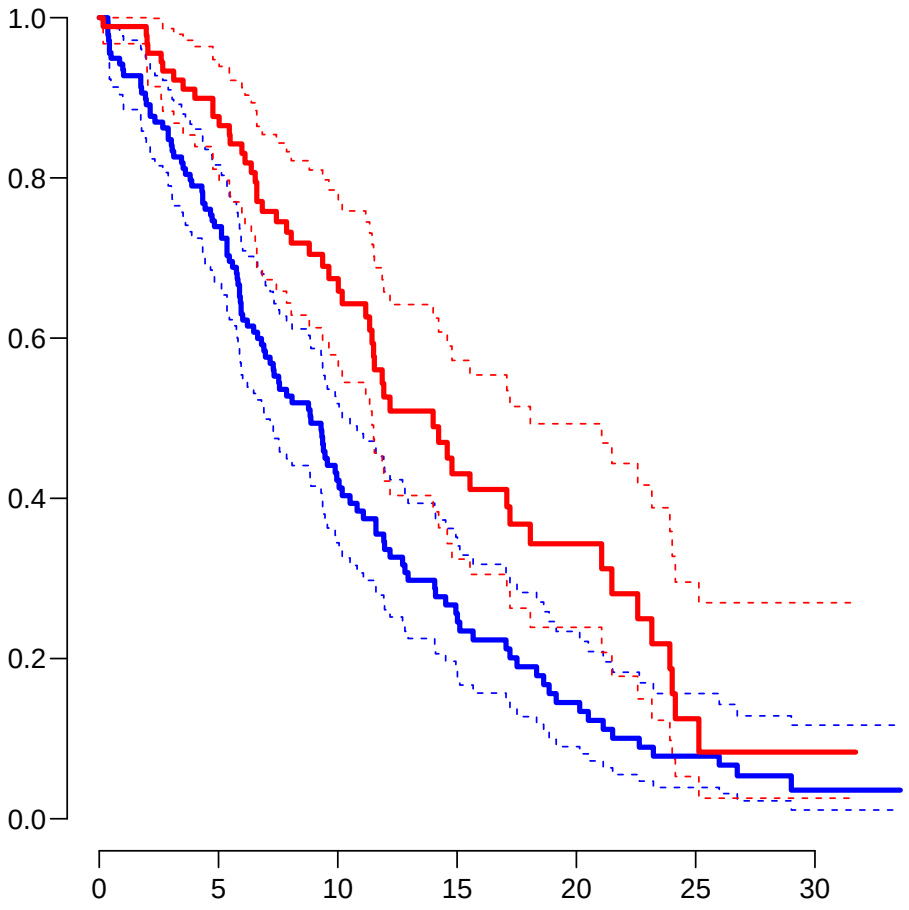


Figure 2.1: *Kaplan-Meier estimators of survival for men (blue) and women (red). W*
`../graph/surv-kms`

e see that men have worse survival than women, but they are also a bit older (`age` is age at diagnosis of lung cancer):

```
> with(lung, tapply(age, sex, mean))
       M        W
63.34058 61.07778
```

Formally there is a significant difference in survival between men and women

```
> ?survdiff
> survdiff(Surv(time, status==2) ~ sex, data = lung)
```

What is the null hypothesis tested here?

2.2 Rates and rate-ratios: Simple Cox model

4. Now explore how sex and age (at diagnosis) influence the mortality—note that we are now addressing the mortality rate and not the survival in a Cox-model:

```
> c0 <- coxph(Surv(time, status == 2) ~ sex       , data = lung)
> c1 <- coxph(Surv(time, status == 2) ~ sex + I(age/10), data = lung)
> summary(c1)
Call:
coxph(formula = Surv(time, status == 2) ~ sex + I(age/10), data = lung)

  n= 228, number of events= 165

               coef exp(coef) se(coef)      z Pr(>|z|)
sexW      -0.51322   0.59857  0.16746 -3.065  0.00218
I(age/10)  0.17045   1.18584  0.09223  1.848  0.06459

           exp(coef) exp(-coef) lower .95 upper .95
sexW          0.5986     1.6707    0.4311    0.8311
I(age/10)     1.1858     0.8433    0.9897    1.4208

Concordance= 0.603  (se = 0.025 )
Likelihood ratio test= 14.12  on 2 df,   p=9e-04
Wald test            = 13.47  on 2 df,   p=0.001
Score (logrank) test = 13.72  on 2 df,   p=0.001
> ci.exp(c0)
     exp(Est.)      2.5%     97.5%
sexW 0.5880028 0.4237178 0.8159848
> ci.exp(c1)
           exp(Est.)      2.5%     97.5%
sexW       0.598566 0.4310936 0.8310985
I(age/10)  1.185842 0.9897335 1.4208086
```

   We see that there is not much confounding by age; the W/M mortality RR (hazard ratio is another word for this) is slightly below 0.6 whether age is included or not. The age effect is formally non-significant, the estimate corresponds to a 1.7% higher mortality rate per year of age at diagnosis (mortality RR or hazard ratio of 1.017). What is the mortality RR for a 10 year age difference?
5. We can check if the assumption of proportional hazards holds, `cox.zph` provides a test, and the plot method shows the Schoenfeld residuals and a smooth of them; interpretable as an estimate of the interaction effect; that is how the W/M (log) rate-ratio depends on time:

```
> ?cox.zph
```

```
> cox.zph(c0)
       chisq df     p
sex     2.86  1 0.091
GLOBAL  2.86  1 0.091
> (z1 <- cox.zph(c1))
          chisq df    p
sex       2.608  1 0.11
I(age/10) 0.209  1 0.65
GLOBAL    2.771  2 0.25
> par(mfrow = c(1, 2)) ; plot(z1)
```

If the proportional hazards model holds, then the resulting lines in he plots should be approximately horizontal.

6. We see that there is no systematic pattern for age, but an increase by sex. The `cox.zph` really gives a test for an *interaction* between each covariate and the time scale. We will keep that in mind so we can assess this through proper modeling of the interaction—the Cox model does not include the estimate of the effect of `time`, and the by that token it is impossible to estimate any interaction with time as well.
7. Before we showed the Kaplan-Meier estimator for each of the two sexes. We can also show the estimated survival curves for the two sexes as derived from the Cox-model. This requires a *prediction* data frame—a data frame with the same variables as in the Cox-model and values of these representing the persons for whom we want predictions:

```
> prs <- survfit(c0, newdata = data.frame(sex = c("M","W")))
> plot(prs, col = c("blue", "red"))
```

How is the shape of the two curves relative to each other?

8. Try to over-plot the Cox-prediction on the Kaplan-Meier curves:

```
>  plot(prs, col = c("blue", "red"), lwd = 1, lty = 1, conf.int = TRUE)
> lines(prs, col = c("blue", "red"), lty = 1, lwd=3)
> lines(kms, col = c("blue", "red"), lty = 2, lwd=2)
```

Do they agree? What does that mean?

2.3 Simple Poisson model

9. But we do not know how the mortality *per se* looks as a function of `time` (since diagnosis). That function is not available from the Cox-model or from the `survfit` object. To that end we must provide a model for the effect of time on mortality; the simplest is of course to assume that it is constant or a simple linear function of time. For a start we assume that the mortality is constant over time, it is so that the likelihood for the model is equivalent to a Poisson likelihood, which can be fitted using the `poisreg` family from the `Epi` package:

```
> ?poisreg
> p1 <- glm(cbind(status == 2, time) ~ sex + age,
+           family = poisreg,
+             data = lung)
> ci.exp(p1) # estimates form Poisson
             exp(Est.)       2.5%     97.5%
(Intercept) 0.03255152 0.01029228 0.1029511
sexW        0.61820515 0.44555636 0.8577537
age         1.01574132 0.99777446 1.0340317
```

```
> ci.exp(c1) # estimates from Cox
             exp(Est.)      2.5%     97.5%
sexW          0.598566 0.4310936 0.8310985
I(age/10)     1.185842 0.9897335 1.4208086
```

We see that the estimates of sex and age effects are quite close between the Poisson and the Cox models, but also that the Poisson model has an intercept term, the estimate of the (assumed) constant underlying mortality. Since we entered the risk time part of the response (second argument in the `cbind`) in units of months (remember we rescaled in the beginning?), the `(Intercept)` (taken from the `ci.exp`) is a rate per 1 person-month.

What age and sex does the `(Intercept)` refer to?

10. The syntax for `poisreg` is a bit different from that for `poisson`, which would be:

```
> px <- glm(status == 2  ~ sex + age + offset(log(time)),
+           family = poisson,
+             data = lung)
> ## or:
> px <- glm(status == 2  ~ sex + age,
+           offset = log(time),
+           family = poisson,
+             data = lung)
> ci.exp(px)
```

The formulation with the `offset` is the reason that papers use the description "... we fitted a Poisson model with log person years as offset".

The drawback of the `poisson` approach is that you need the (risk) `time` (person-years) as a variable in the prediction frame. This is not the case for `poisreg`, where you get the predicted rates per unit in which as you entered the person years when specifying the model.

We shall return to prediction of `rates`.

2.4 Representation of follow-up: `Lexis` object

If we want to see how mortality varies by age we must split the follow-up of each person in small intervals of say, 30 days. This is most easily done using a `Lexis` object. That is basically just taking the `lung` dataset and adding a few features that defines times and states. The point is that it makes life a lot easier when things get more complex than just simple survival.

11. First make a `Lexis` object:

```
> ?Lexis
> Ll <- Lexis(exit = list(tfl = time),
+             exit.status = factor(status,
+                                  levels = 1:2,
+                                  labels = c("Alive","Dead")),
+             data = lung)
NOTE: entry.status has been set to "Alive" for all.
NOTE: entry is assumed to be 0 on the tfl timescale.
> head(Ll)
  tfl   lex.dur lex.Cst lex.Xst lex.id inst      time status age sex ph.ecog ph.karno
1   0 10.053388   Alive    Dead      1    3 10.053388      2  74   M       1       90
2   0 14.948665   Alive    Dead      2    3 14.948665      2  68   M       0       90
```

```
3   0 33.182752   Alive   Alive      3    3 33.182752      1  56   M       0       90
4   0  6.899384   Alive    Dead      4    5  6.899384      2  57   M       1       90
5   0 29.010267   Alive    Dead      5    1 29.010267      2  60   M       0      100
6   0 33.577002   Alive   Alive      6   12 33.577002      1  74   M       1       50
  pat.karno meal.cal wt.loss
1       100     1175      NA
2        90     1225      15
3        90       NA      15
4        60     1150      11
5        90       NA       0
6        80      513       0
```

We see that 5 variables have been added to the dataset:

`tfl:` time from lung cancer *at the time of entry*, therefore it is 0 for all persons; the entry time is 0 from the entry time.

`lex.dur:` the *length* of time a person is in state `lex.Cst`, here measured in months, because `time` is.

`lex.Cst:` Current state, the state in which the `lex.dur` time is spent.

`lex.Xst:` eXit state, the state to which the person moves after the `lex.dur` time in `lex.Cst`.

`lex.id:` a numerical id of each record in the dataset (normally this will be a person id).

This seems a bit of an overkill for keeping track of time and death for the lung cancer patients, but the point is that this generalizes to multistate data too.

It also gives a handy overview of the follow-up:

```
> summary(L1)
Transitions:
     To
From    Alive Dead  Records:  Events: Risk time:  Persons:
  Alive    63  165       228      165    2286.42       228
```

What is the average follow-up time for persons?

For a graphical representation, try:

```
> ?boxes
> boxes(L1, boxpos = TRUE)
```

Explain the numbers in the resulting graph. Redo the graph with risk time counted in years.

12. We can make the Cox-analysis using the `Lexis`-specific variables by:

```
> ?Surv
> cl <- coxph(Surv(tfl,
+                  tfl + lex.dur,
+                  lex.Xst == "Dead") ~ sex + age,
+             data = L1)
```

but even simpler, by using the `Lexis` features:

```
> ?coxph.Lexis
> cL <- coxph.Lexis(L1, tfl ~ sex + age)
survival::coxph analysis of Lexis object L1:
Rates for the transition Alive->Dead
Baseline timescale: tfl
> ci.exp(cL)
     exp(Est.)      2.5%     97.5%
sexW  0.598566 0.4310936 0.8310985
age   1.017191 0.9989686 1.0357467
```

```
> ci.exp(cl)
     exp(Est.)      2.5%     97.5%
sexW  0.598566 0.4310936 0.8310985
age   1.017191 0.9989686 1.0357467
```

13. And we can make the Poisson-analysis by:

```
> pc <- glm(cbind(lex.Xst == "Dead", lex.dur) ~ sex + age,
+           family = poisreg,
+             data = Ll)
```

or even simpler, by using the `Lexis` features:

```
> pL <- glm.Lexis(Ll, ~ sex + age)
stats::glm Poisson analysis of Lexis object Ll with log link:
Rates for the transition: Alive->Dead
> ci.exp(pL)
             exp(Est.)       2.5%     97.5%
(Intercept) 0.03255152 0.01029228 0.1029511
sexW        0.61820515 0.44555636 0.8577537
age         1.01574132 0.99777446 1.0340317
> ci.exp(pc)
             exp(Est.)       2.5%     97.5%
(Intercept) 0.03255152 0.01029228 0.1029511
sexW        0.61820515 0.44555636 0.8577537
age         1.01574132 0.99777446 1.0340317
```

Remember that the Poisson-model fitted is a very brutal approximation to the Cox-model; it assumes that the baseline hazard is constant, whereas the Cox-model allows the baseline hazard to vary arbitrarily by time.

2.5 Estimating the hazard function: splitting time

If we want a more detailed version of the baseline hazard we split follow-up time in small intervals, assume that the hazard is constant in each small interval, and assume the the *size* of the hazard varies smoothly with time, `tfl`:

14. We can subdivide the follow-up in small intervals by `survival:::survSplit`, `Epi:::splitLexis` or `popEpi:::splitMulti` (and possibly many more). The `splitMulti` is by far the easiest to use (and fastest as well). Recall we rescaled time to months, so we split in 1 month intervals:

```
> Sl <- splitMulti(Ll, tfl = 0:36)
```

This will split the follow-up along the time-scale `tfl` at times 0, 1, . . . , 36 months; we see that the follow-up time is the same, but there are now about 10 times as many records:

```
> summary(Ll)
Transitions:
     To
From    Alive Dead  Records:  Events: Risk time:  Persons:
  Alive    63  165       228      165    2286.42       228
> summary(Sl)
Transitions:
     To
From    Alive Dead  Records:  Events: Risk time:  Persons:
  Alive  2234  165      2399      165    2286.42       228
```

We can see how the follow up for person, 10 say, is in the original and the split dataset:

```
> wh <- names(Ll)[1:10] # names of variables in some order
> subset(Ll, lex.id == 10)[,wh]
   tfl  lex.dur lex.Cst lex.Xst lex.id inst     time status age sex
10   0 5.453799   Alive    Dead     10    7 5.453799      2  61   M
> subset(Sl, lex.id == 10)[,wh]
    tfl   lex.dur lex.Cst lex.Xst lex.id inst     time status age sex
163   0 1.0000000   Alive   Alive     10    7 5.453799      2  61   M
164   1 1.0000000   Alive   Alive     10    7 5.453799      2  61   M
165   2 1.0000000   Alive   Alive     10    7 5.453799      2  61   M
166   3 1.0000000   Alive   Alive     10    7 5.453799      2  61   M
167   4 1.0000000   Alive   Alive     10    7 5.453799      2  61   M
168   5 0.4537988   Alive    Dead     10    7 5.453799      2  61   M
```

In `Sl` each record now represents a small interval of follow-up for a person, so each person has many records. The main thing to note here is `tfl`, which represents the time from lung cancer at the beginning of each interval, and `lex.dur` representing the risk time ("person-years", in months though).

15. We can now include a smooth effect of `tfl` in the Poisson-model allowing the baseline hazard to vary by time. That is done by natural splines, `Ns`:

```
> ps <- glm(cbind(lex.Xst == "Dead", lex.dur)
+           ~ Ns(tfl, knots = seq(0, 36, 12)) + sex + age,
+           family = poisreg,
+             data = Sl)
> ci.exp(ps)
                                  exp(Est.)        2.5%       97.5%
(Intercept)                       0.0189837 0.005700814  0.06321569
Ns(tfl, knots = seq(0, 36, 12))1 2.4038681 0.809442081  7.13896863
Ns(tfl, knots = seq(0, 36, 12))2 4.1500822 0.436273089 39.47798357
Ns(tfl, knots = seq(0, 36, 12))3 0.8398973 0.043928614 16.05849662
sexW                              0.5987171 0.431232662  0.83124998
age                               1.0165872 0.998377104  1.03512945
```

or even simpler:

```
> ?glm.Lexis
> ps <- glm.Lexis(Sl, ~ Ns(tfl, knots = seq(0, 36, 12)) + sex + age)
> ci.exp(ps)
```

16. Compare these to the regression estimates from the Cox-model and from the model with constant baseline:

```
> round(cbind(ci.exp(cl),
+             ci.exp(ps, subset = c("sex","age")),
+             ci.exp(pc, subset = c("sex","age"))), 3)
     exp(Est.)  2.5% 97.5% exp(Est.)  2.5% 97.5% exp(Est.)  2.5% 97.5%
sexW     0.599 0.431 0.831     0.599 0.431 0.831     0.618 0.446 0.858
age      1.017 0.999 1.036     1.017 0.998 1.035     1.016 0.998 1.034
```

We see that the smooth parametric Poisson model and the Cox model produce virtually the same estimates, whereas the Poisson model with constant hazard produce slightly different ones.

17. The proportional hazards assumption is the same for the Cox model and the Poisson models: The M/W hazard ratio is the same at any time after diagnosis. What differs is the assumed shape of the hazard (`not` a hazard ratio).

    The Cox model allows the baseline rate to change arbitrarily at every event time time not using the quantitative nature of time, the `ps` Poisson model has a baseline that

varies smoothly by time and the `pc` Poisson model has a baseline that is constant over time. The latter is clearly not tenable, whereas the smooth Poisson model and the Cox model give the same regression estimates.

18. We now have a *parametric* model for the baseline hazard which means that we can show the estimated baseline hazard for a 60-year old woman, by supplying a suitable prediction frame, i.e. a data frame where each row represents a set of covariate values, including the time where we want the predicted mortality:
```
> prf <- data.frame(tfl = seq(0, 30, 0.2),
+                   sex = "W",
+                   age = 60)
```
We can over-plot with the predicted rates from the model where mortality rates are constant, the only change is the model (`pc` instead of `ps`):
```
> matshade(prf$tfl, ci.pred(ps, prf),
+          plot = TRUE, log = "y", lwd = 3)
> matshade(prf$tfl, ci.pred(pc, prf), lty = 2, lwd = 3)
```
What we see from the plot is that mortality rates are increasing during the first 1.5 years after lung cancer and then leveling off.
Put some sensible axis labels on the plot, and rescale the rates to rates per 1 person-year.
19. We can transform the hazard function, $\lambda(t)$, to a survival function, $S(t)$ using the relationship $S(t) = \exp(-\int_0^t \lambda(u)\,\mathrm{d}u)$. This is implemented in the `ci.surv` function, which takes the model and a prediction data frame as arguments; the prediction data frame must correspond to a sequence of equidistant time points, so we can use `prf` for this purpose:
```
> matshade(prf$tfl, ci.surv(ps, prf, intl = 0.2),
+          plot = TRUE, ylim = 0:1, lwd = 3)
```
We can expand this by overlaying the survival function from the model with constant hazard (also known as "exponential(y distributed) survival") and the KM-estimator
```
> matshade(prf$tfl, ci.surv(ps, prf, intl = 0.2),
+          plot = TRUE, ylim = 0:1, lwd = 3)
> lines(prf$tfl, ci.surv(pc, prf, intl = 0.2)[,1])
> lines(survfit(c1, newdata = data.frame(sex = "W", age = 60)),
+       lwd = 2, lty = 1)
```
We see that the survival function from the constant hazard model is quite a bit off, but also a good correspondence between the Cox-model based survival and the survival from the parametric hazard function.
We can bring the plots together in one graph:
```
> par(mfrow = c(1,2))
> # hazard scale
> matshade(prf$tfl, ci.pred(ps, prf),
+          plot = TRUE, log = "y", lwd = 3)
> matshade(prf$tfl, ci.pred(pc, prf), lty = 3, lwd = 3)
> # survival
> matshade(prf$tfl, ci.surv(ps, prf, intl = 0.2),
+          plot = TRUE, ylim = 0:1, lwd = 3)
> matshade(prf$tfl, ci.surv(pc, prf, intl = 0.2),
+          lty = 3, alpha = 0, lwd = 3)
> lines(survfit(c1, newdata = data.frame(sex = "W", age = 60)),
+       col = "forestgreen", lwd = 3)
```

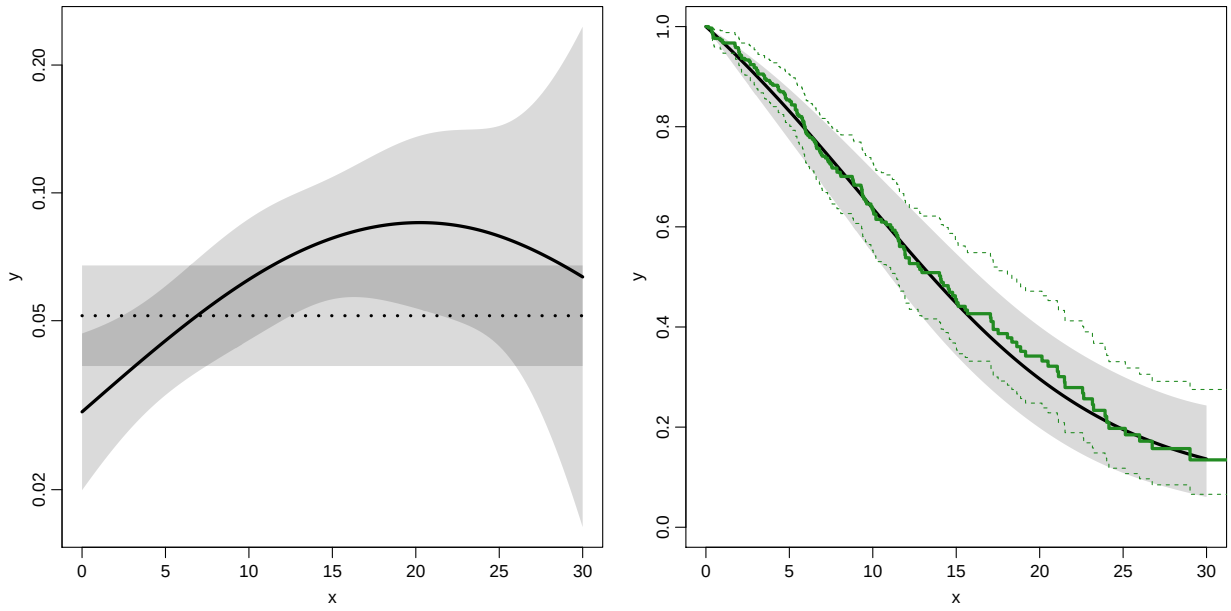


Figure 2.2: *Hazards (left) and survival (right) for 60 year old women. The left hand plot is unavailable from the Cox model.*

`../graph/surv-ratesurv`

20. We have compared the predicted survival curve from a Poisson model with age and sex and time since lung cancer as covariates to that from a Cox-model with age and sex as covariates and time since lung cancer as underlying time scale.
    We now go back to the Kaplan-Meier estimator and compare that to the corresponding Poisson-model, which is one with time (`tfl`) as the only covariate:

```
> par(mfrow=c(1,2))
> pk <- glm(cbind(lex.Xst == "Dead",
+                 lex.dur) ~ Ns(tfl, knots = seq(0, 36, 12)),
+           family = poisreg,
+             data = Sl)
> # hazard
> matshade(prf$tfl, ci.pred(pk, prf),
+          plot = TRUE, log = "y", lwd = 3, ylim = c(0.01,1))
> # survival from smooth model
> matshade(prf$tfl, ci.surv(pk, prf, intl = 0.2) ,
+          plot = TRUE, lwd = 3, ylim = 0:1)
> # K-M estimator
> lines(km, lwd = 2)
```

21. We can explore how the tightness of the knots in the smooth model influence the underlying hazard and the resulting survival function. This is easiest done by setting up a function that does the analysis withe different number of knots

```
> zz <-
+ function(dk)
+ {
+ kn <- seq(0, 36, dk)
+ pk <- glm(cbind(lex.Xst == "Dead",
+                 lex.dur) ~ Ns(tfl, knots = kn),
```

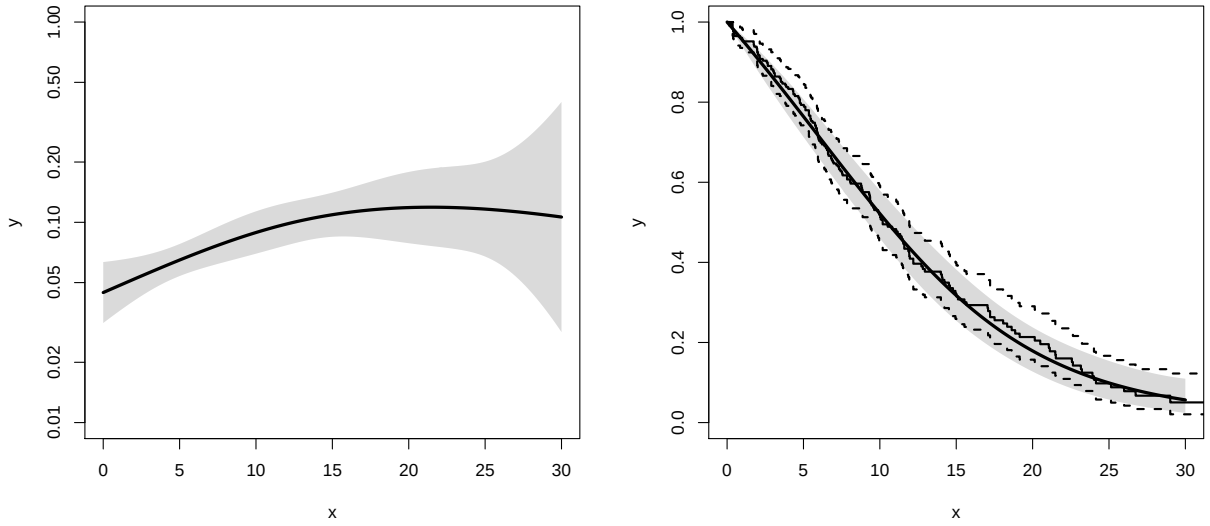


Figure 2.3: *Baseline hazard (left), and corresponding survival function from parametric model and Kaplan-Meier estimator.*

`../graph/surv-parkm`

```
+           family = poisreg,
+             data = S1)
+ matshade(prf$tfl, ci.pred(pk, prf),
+          plot = TRUE, log = "y", lwd = 3, ylim = c(0.01,1))
+ rug(kn, lwd=3)
+
+ plot(km, lwd = 2, col = "limegreen")
+ matshade(prf$tfl, ci.surv(pk, prf, intl = 0.2) ,
+          lwd = 3, ylim = 0:1)
+ }
> par(mfrow=c(1,2))
> zz(12)
> par(mfrow=c(4,2))
> for (nk in c(6, 4, 3, 2)) zz(nk)
```

You will see that the more knots you include, the closer the parametric estimate gets to the Kaplan-Meier estimator. But also that the estimated underlying hazard becomes increasingly silly. The ultimate silliness is of course achieved when we arrive at the Kaplan-Meier estimator.

Fortunately the baseline hazard underlying the Kaplan-Meier and the Breslow estimator is rarely shown.

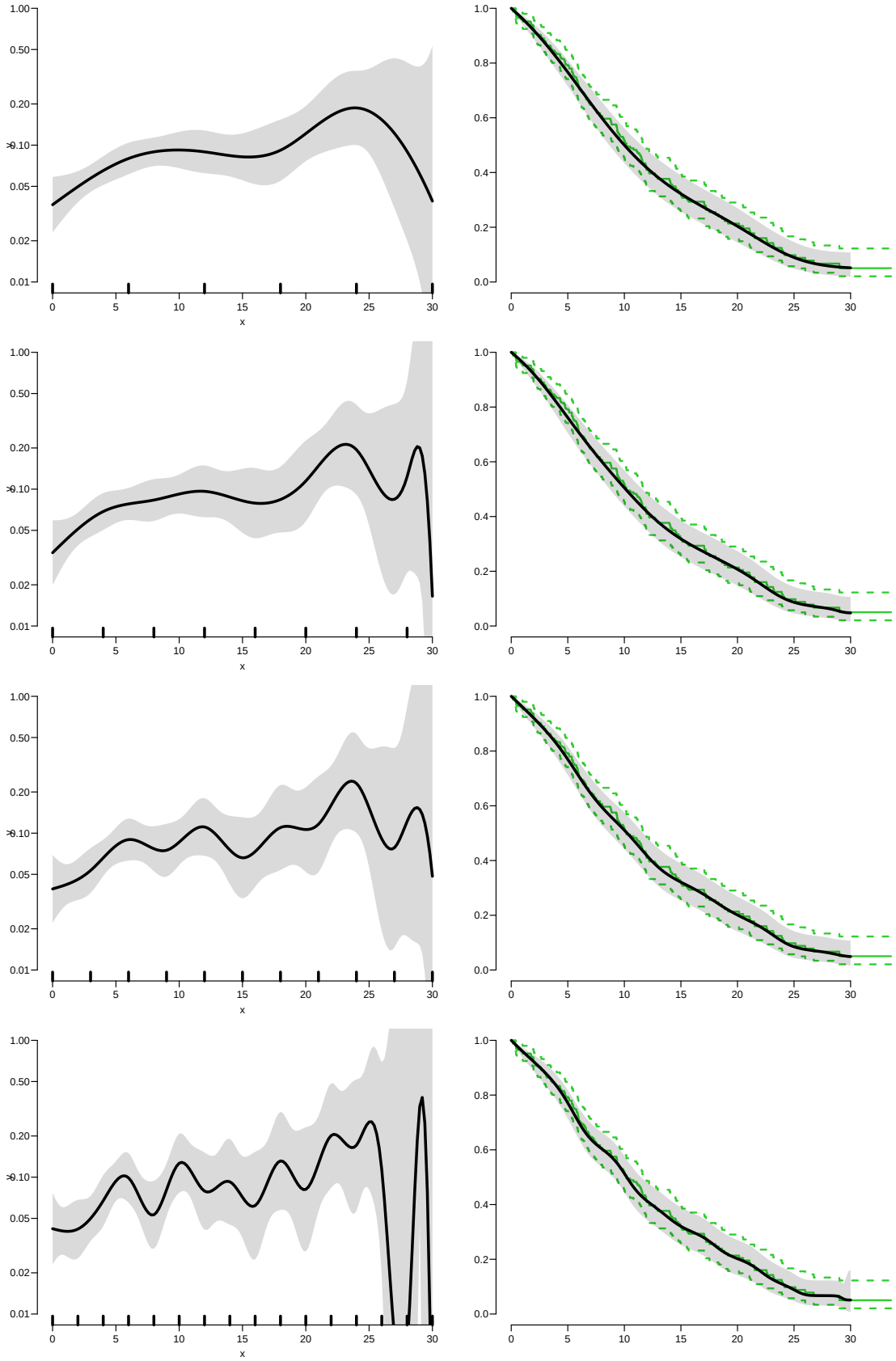


Figure 2.4: *Hazard (left) and survival (right) comparing a parametric model with different number of knots and the Kaplan-Meier estimator.*

`../graph/surv-knots2`

Chapter 3

Competing risks: `DMlate`

Paraphernalia

It is advisable to load all packages needed at the start:

```
> library(survival)
> library(Epi)
> library(popEpi)
> # popEpi::splitMulti returns a data.frame rather than a data.table
> options("popEpi.datatable" = FALSE)
> library(tidyverse)
> clear()
```

3.1 Data

This exercise follows quite closely the section on competing risks in "Epidemiology with R", pp. 207 and 210 ff. With the major exception that we will use the function `ci.Crisk`, which was not available in the *Epi* package when the book was written.

We shall use the `DMlate` dataset which is a random sample of Danish diabetes patients, with dates of birth, diabetes, OAD start, insulin start and death.

We want to look at the event "start of OAD", which occurs at `dooad`, while taking death as competing event into account. This means that we want to address the question of the probability of starting OAD, while taking death into account. Essentially estimating the probability of being in each of the states `DM`, `OAD` and `Dead`, where `OAD` means "started OAD and either alive or dead after this" and `Dead` means "dead without starting OAD".

1. Load the `DMlate` data from the `Epi` package, and for ease of calculation restrict to a random sample of 2000 persons:

```
> data(DMlate)
> # str(DMlate)
> set.seed(1952)
> DMlate <- DMlate[sample(1:nrow(DMlate), 2000),]
> str(DMlate)
'data.frame':        2000 obs. of  7 variables:
 $ sex  : Factor w/ 2 levels "M","F": 2 1 2 1 1 1 1 1 1 1 ...
 $ dobth: num  1964 1944 1957 1952 1952 ...
 $ dodm : num  2003 2006 2008 2007 2003 ...
```

```
 $ dodth: num  NA NA NA NA NA NA NA NA NA NA ...
 $ dooad: num  NA 2006 NA 2007 2006 ...
 $ doins: num  NA NA NA 2008 NA ...
 $ dox  : num  2010 2010 2010 2010 2010 ...
> head(DMlate)
       sex    dobth     dodm dodth    dooad    doins      dox
70126    F 1963.591 2003.481    NA       NA       NA 2009.997
235221   M 1944.127 2005.644    NA 2005.778       NA 2009.997
230872   F 1956.790 2007.886    NA       NA       NA 2009.997
138167   M 1952.355 2006.969    NA 2006.969 2008.026 2009.997
406109   M 1952.240 2003.361    NA 2005.852       NA 2009.997
72438    M 1978.758 2001.948    NA       NA 2001.967 2009.997
```

2. Define a `Lexis` object with the total follow up for each person:

```
> Ldm <- Lexis(entry = list(per = dodm,
+                           age = dodm - dobth,
+                           tfd = 0),
+              exit = list(per = dox),
+       exit.status = factor(!is.na(dodth),
+                            labels = c("DM","Dead")),
+              data = DMlate)
NOTE: entry.status has been set to "DM" for all.
NOTE: Dropping  1  rows with duration of follow up < tol
> summary(Ldm)
Transitions:
     To
From   DM Dead  Records:  Events: Risk time:  Persons:
  DM 1521  478      1999      478   10742.34      1999
```

Then subdivide the follow-up at the date of OAD, using `dooad`:

```
> Cdm <- cutLexis(Ldm,
+                 cut = Ldm$dooad,
+           timescale = "per",
+           new.state = "OAD")
> summary(Cdm)
Transitions:
     To
From   DM  OAD Dead  Records:  Events: Risk time:  Persons:
  DM  685  634  226      1545      860    5414.29      1545
  OAD   0  836  252      1088      252    5328.05      1088
  Sum 685 1470  478      2633     1112   10742.34      1999
```

In this context we are not interested in what goes on after OAD so we only keep follow-up in state `DM` (note that we must use `subset` because `filter` does not have a method for `Lexis` objects):

```
> Adm <- subset(Cdm, lex.Cst == "DM")
> summary(Adm)
Transitions:
     To
From  DM OAD Dead  Records:  Events: Risk time:  Persons:
  DM 685 634  226      1545      860    5414.29      1545
> boxes(Adm, boxpos = TRUE, scale.R = 100, show.BE = TRUE)
```

As shown in figure 3.1 we now have a traditional competing risks set-up, with some 1500 DM patients starting without OAD, and where the quantity of interest is the probability of starting drug treatment, and the `OAD` state here means "having been on oral antidiabetic treatment, disregarding subsequent death". The other event

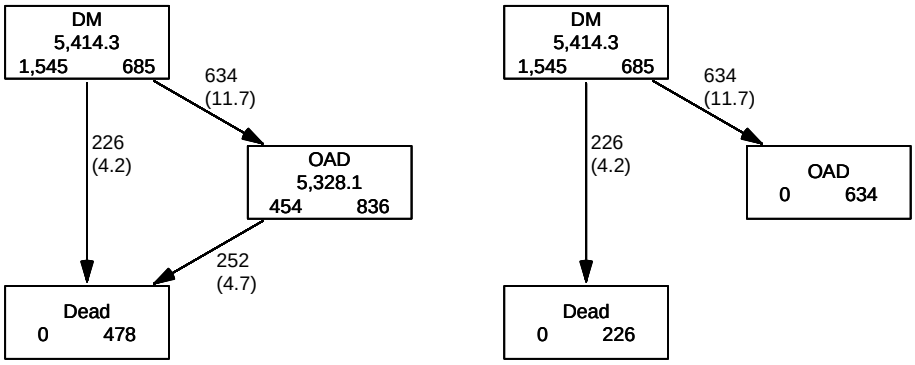


Figure 3.1: *Competing risks set-up for events* `OAD` *and* `Dead`.

`../graph/cmpr-boxCR`

considered is `Dead` which here means "dead without initiating oral antidiabetic treatment".

3.2 State probabilities

We can compute the (correct) counterpart of the survival function for this competing risks setup. The survival function we saw in the previous exercise gives the probability of being alive, and the complement is the probability of being dead.

3. `survfit` can do the corresponding calculation for the three states in the figure; the requirements are: 1) the third argument to the `Surv` function is a factor and 2) an `id` argument is given, pointing to an id variable that links together records belonging to the same person. The latter is superfluous in this case because there is only one record for each person, but even so it is required by the function `survfit`.
   Also note that the initial state (`DM`) must be the first level of the factor `lex.Xst`:

```
> levels(Adm$lex.Xst)
[1] "DM"   "OAD"  "Dead"
> m3 <- survfit(Surv(tfd,
+                    tfd + lex.dur,
+                    lex.Xst) ~ 1,
+                id = lex.id,
+              data = Adm)
> names(m3)
 [1] "n"           "time"        "n.risk"      "n.event"     "n.censor"    "pstate"
 [7] "p0"          "cumhaz"      "std.err"     "sp0"         "logse"       "transition
[13] "conf.int"    "conf.type"   "lower"       "upper"       "conf.type"   "conf.int"
[19] "states"      "type"        "call"
> m3$states
[1] "(s0)" "OAD"  "Dead"
> head(cbind(time = m3$time, m3$pstate))
           time
```

```
[1,] 0.002737851 0.9987055 0.001294498 0.0000000000
[2,] 0.005475702 0.9928803 0.006472492 0.0006472492
[3,] 0.008213552 0.9889968 0.009061489 0.0019417476
[4,] 0.010951403 0.9877023 0.009708738 0.0025889968
[5,] 0.013689254 0.9838188 0.013592233 0.0025889968
[6,] 0.016427105 0.9805825 0.016828479 0.0025889968
```

Because `lex.Xst` is a factor, `survfit` will compute the Aalen-Johansen estimator of being in a given state and place the probabilities in the matrix `m3$pstate`; the times these refer to are in the vector `m3$time`. These are measured in years since diabetes, because `tfd` is in units of years,

Explore the object `m3`; start by using `names(m3)`.

Compare `m3$transitions` to `summary(Adm)`.

4. The `m3$pstate` contains the Aalen-Johansen probabilities of being in the `Alive`, having left to the `OAD`, resp. `Dead` state.
   Plot the three curves in the same graph (use for example `matplot`). Add the confidence limits.
5. These three curves have sum 1, so basically this is a way of distributing the probabilities across states at each time. It is therefore natural to stack the probabilities, which can be done by `stackedCIF`:

```
> par( mfrow=c(1,2) )
> matplot(m3$time, m3$pstate,
+         type="s", lty=1, lwd=4,
+         col=c("ForestGreen","red","black"),
+         xlim=c(0,15), xaxs="i",
+         ylim=c(0,1), yaxs="i" )
> stackedCIF(m3, lwd=3, xlim=c(0,15), xaxs="i", yaxs="i" )
> text( rep(12,3), c(0.9,0.3,0.6), levels(Cdm) )
> box()
```

6. What do you get if you replace "`~ 1`" by "`~ sex`" in the call to `survfit`?

3.3 What not to do

A very common error is to use a *partial* outcome such as `OAD`, when there is a competing type of event, in this case `Dead`. If that is ignored and a traditional survival analysis is made *as if* `OAD` were the only possible event, we will have a substantial *over*estimate of the cumulative probability of going on drug. Here is an illustration of this erroneous approach:

```
> m2 <- survfit(Surv(tfd,
+                    tfd + lex.dur,
+                    lex.Xst == "OAD" ) ~ 1,
+               data = Adm)
> M2 <- survfit(Surv(tfd,
+                    tfd + lex.dur,
+                    lex.Xst == "Dead") ~ 1,
+               data = Adm)
> par(mfrow = c(1,2))
> mat2pol(m3$pstate, c(2,3,1), x = m3$time,
+         col = c("red", "black", "transparent"),
+         xlim=c(0,15), xaxs="i",
+         yaxs = "i", xlab = "time since DM", ylab = "" )
```

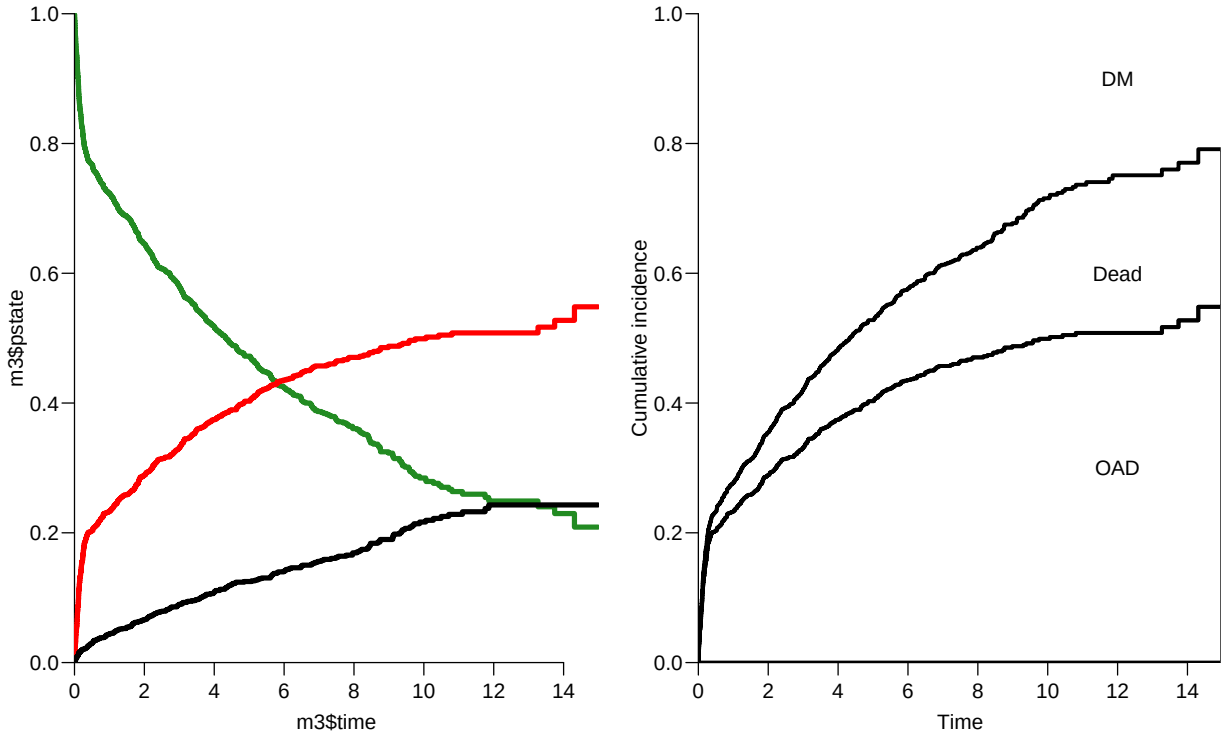


Figure 3.2: *Separate state probabilities (left) and stacked state probabilities (right). In the left panel,* `Alive` *is green,* `OAD` *is red and* `Dead` *is black.* `../graph/cmpr-surv2`

```
>   lines(m2$time, 1 - m2$surv, lwd = 3, col = "red" )
> mat2pol(m3$pstate, c(3,2,1), x = m3$time, yaxs = "i",
+         col = c("black","red","transparent"),
+         xlim=c(0,15), xaxs="i",
+         yaxs = "i", xlab = "time since DM", ylab = "" )
>   lines(M2$time, 1 - M2$surv, lwd = 3, col = "black" )
```

The first two statements calculate the survival as if only `OAD`, respectively `Dead` were the only way of exiting the state `Alive`. The `mat2pol` (`mat`rix to `pol`ygon) takes the columns of state probabilities from the `survfit` object `m3` that contains the correctly modeled probabilities and plot them as coloured areas stacked; the second argument to `mat2pol` is the order in which they should be stacked. The `lines` plot the wrongly computed cumulative risks (from `m2` and `M2`) — in order to find these we fish out the `surv` component from the `survfit` objects.

3.4 Modeling cause specific rates

There is nothing wrong with modeling the cause-specific event-rates, the problem lies in how you transform them into probabilities. The relevant model for a competing risks situation normally consists of separate models for each of the cause-specific rates. Not for technical or statistical reasons, but for *substantial* reasons; it is unlikely that rates of different types of event (OAD initiation and death, say) depend on time in the same way.

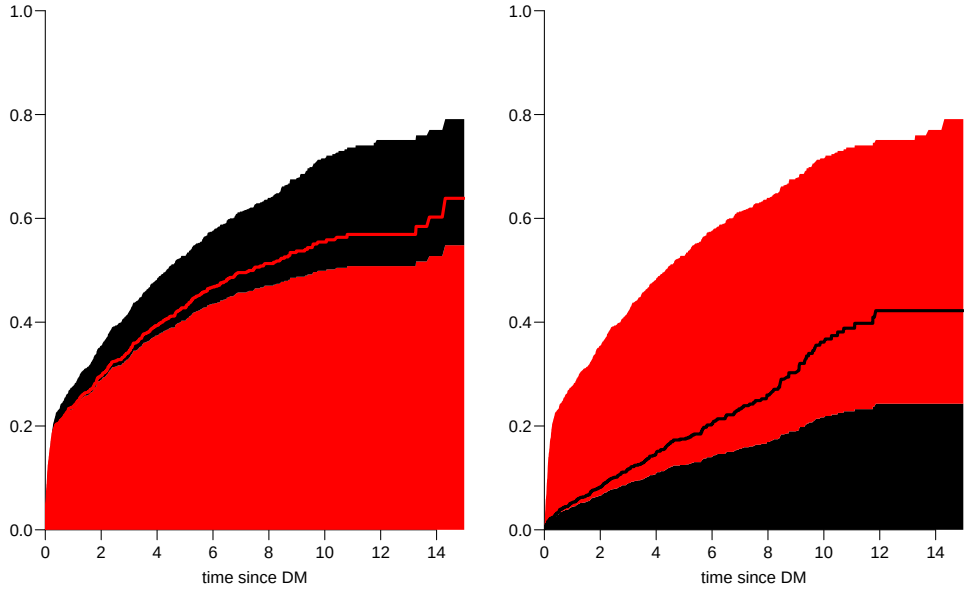


Figure 3.3: *Stacked state probabilities* `Alive` *is white,* `OAD` *is red and* `Dead` *is black. The red line in the left panel is the wrong (but often computed) "cumulative risk" of* `OAD`*, and the black line in the right panel is the wrong (but often computed) "cumulative risk" of* `Death`*. The black and the red areas in the two plots represent the correctly computed probabilities; they have the same size in both panels, only they are stacked differently.* `../graph/cmpr-surv3`

7. Now model the two sets of rates by parametric models; this must be based on a time-split data set:

```
> Sdm <- splitMulti(Adm, tfd = seq(0,20,0.1) )
> summary(Adm)
Transitions:
     To
From  DM OAD Dead  Records:  Events: Risk time:  Persons:
  DM 685 634  226      1545      860    5414.29      1545
> summary(Sdm)
Transitions:
     To
From    DM OAD Dead  Records:  Events: Risk time:  Persons:
  DM 54064 634  226     54924      860    5414.29      1545
```

8. We will use natural splines for the effect of diabetes duration in a model using `glm`. The `Ns` requires a set of pre-specified knots for the time variable, where the specification should be (partially) guided by the location on the times of the events:

```
> round(cbind(
+ with(subset(Sdm, lex.Xst == "OAD" ), quantile(tfd + lex.dur, 0:10/10)),
+ with(subset(Sdm, lex.Xst == "Dead"), quantile(tfd + lex.dur, 0:10/10))),
+ 3)
       [,1]   [,2]
0%    0.003  0.005
10%   0.038  0.129
20%   0.095  0.507
30%   0.142  1.083
40%   0.239  1.730
```

```
50%   0.534  2.552
60%   1.268  3.584
70%   2.199  4.490
80%   3.373  6.196
90%   5.213  8.471
100% 14.311 11.858
```

We see that the `OAD` occur earlier than `Dead`, so we choose the knots a bit earlier:

```
> okn <- c(0,0.5,3,6)
> dkn <- c(0,2.0,5,9)
>  OAD.glm <- glm.Lexis(Sdm, ~ Ns(tfd, knots = okn), from = "DM", to = "OAD" )
stats::glm Poisson analysis of Lexis object Sdm with log link:
Rates for the transition: DM->OAD
> Dead.glm <- glm.Lexis(Sdm, ~ Ns(tfd, knots = dkn), from = "DM", to = "Dead")
stats::glm Poisson analysis of Lexis object Sdm with log link:
Rates for the transition: DM->Dead
```

9. With models for the two rates out of the `DM` state we can derive the estimated rates from the two models for rates by time by using a prediction frame, `nd`:

```
> int <- 0.01
> nd <- data.frame(tfd = seq(0, 15, int))
> l.glm <- ci.pred( OAD.glm, nd)
> m.glm <- ci.pred(Dead.glm, nd)
```

Now plot the estimated rates, in this case the `gam` models with dotted and `glm` models with full lines; mortality with black and OAD rates with red:

```
> matshade(nd$tfd,
+          cbind(l.glm, m.glm) * 100,
+          plot = TRUE,
+          log = "y", ylim = c(2, 20),
+          col = rep(c("red","black"), 2), lwd = 3)
```

3.5 Integrals with R

Based on these parametric models we can estimate the cumulative risks of being in each of the states, but also the expected time time spent in each state. The theory of these involves calculation of integrals of the rate functions. Integrals looks scary to many people, but they are really just areas under curves. So here is a digression showing how to calculate integrals as areas under a curve.

The key is to understand how a curve is represented in R. A curve representing the function μ is just a set of two vectors, one vector of ts and one vector $y = \mu(t)$s. When we have a model such as the `gam` or `glm` above that estimates the mortality as a function of time (`tfd`), we can get a representation of the mortality as a funtion of time by first choosing the timepoints, say from 0 to 15 years in steps of 0.01 year ($\approx$ 4 days). Then put this in a dataframe (`nd`, `newdata`) with the variable name from the model to get the function values at the chosen time points:

```
> t <- seq(0, 15, 0.01)
> nd <- data.frame(tfd = t)
> mu <- ci.pred(Dead.glm, nd)[,1]
> head(cbind(t, mu))
     t         mu
1 0.00 0.06919036
2 0.01 0.06885302
```

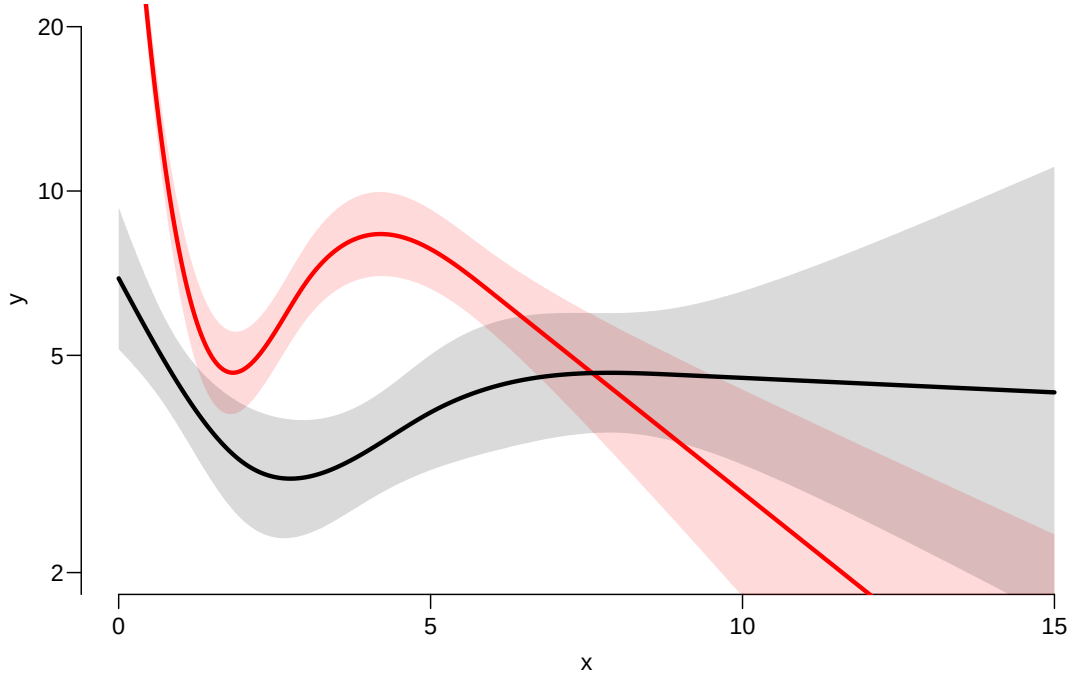


Figure 3.4: *Mortality rates (black) and OAD-rates (red), from a* `glm` *model with natural splines.*

`../graph/cmpr-OAD-mort`

```
3 0.02 0.06851733
4 0.03 0.06818330
5 0.04 0.06785093
6 0.05 0.06752022
> plot(t, mu, type="l", lwd = 3,
+      xlim = c(0, 7), xaxs = "i",
+      ylim = c(0, max(mu)), yaxs = "i")
> polygon(t[c(1:501,501:1)], c(mu[1:501], rep(0, 501)),
+         col = "gray", border = "transparent")
```

This is a representaion of the points $(t, \mu(t))$; if we want the integral of *mu* over the interval $[0, 5]$, say, $M(5) = \int_0^5 \mu(s)\,\mathrm{d}s$, we are just asking for the area under the curve. Each t represents an endpoint of an interval, but what we want in order to compute the area under the curve is the *width* of each interval, `diff(t)`, multiplied by the average of the function values at the ends of each interval (this goes under the name of the "trapezoidal formula"). So we need a small function to compute midpoints between successive values in a vector:

```
> mid <- function(x) x[-1] - diff(x) / 2
> (x <- c(1:5, 7, 10))
[1]  1  2  3  4  5  7 10
> mid(x)
[1] 1.5 2.5 3.5 4.5 6.0 8.5
```

Note that `mid(x)` is a vector that is 1 shorter than the vector `x`, just as `diff(x)` is.

So if we want the integral over the period 0 to 5 years, we want the sum over the first 500 intervals, corresponding to the first 501 interval endpoints:

```
> sum(diff(t[1:501]) * mid(mu[1:501]))
```

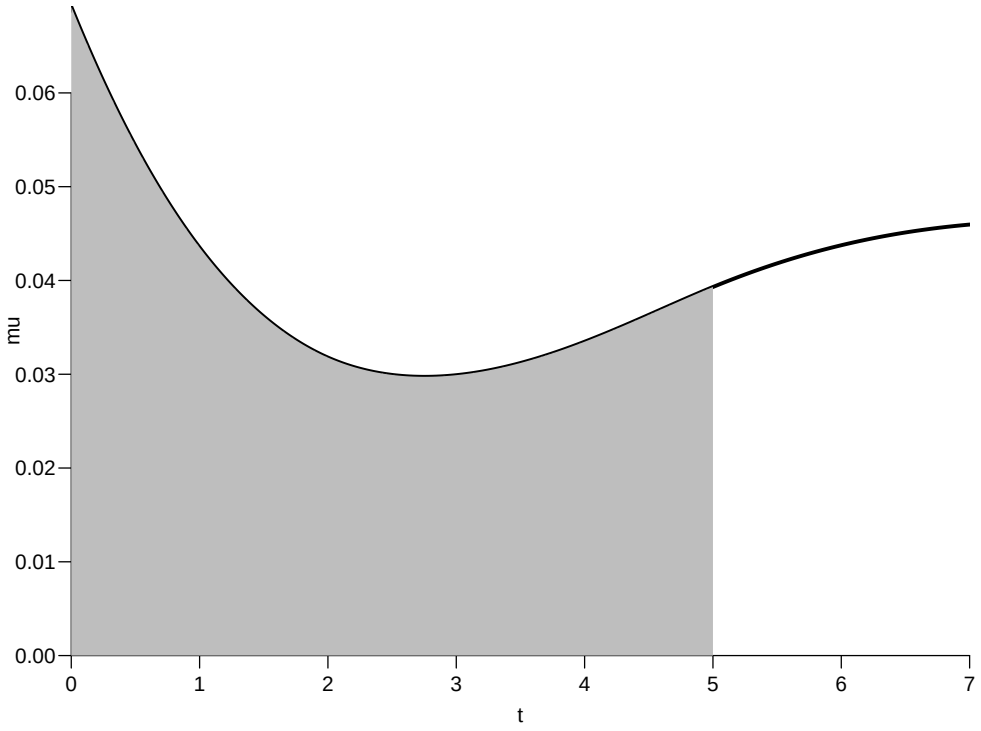


Figure 3.5: *Mortality function and integral from 0 to 5 years.*

`../graph/cmpr-int-ill`

```
[1] 0.1896222
```

So now we have computed $\int_0^5 \mu(s)\,\mathrm{d}(s)$. This is called the cumulative *rate* over the interval $[0, 5]$ years.

It is iportamt to get the units right. In the modeling we entered the risk time ("person-years") in units of 1 year, so the unit of predicted mortality function, `mu`, is events per 1 person-year. Therefore, the units of `t` must be year too; otherwise we will introduce a scaling.

In pratice we will want the integral *function* of μ, so for every t we want $M(t) = \int_0^t \mu(s)\,\mathrm{d}(s)$. This is easily accomplished by the function `cumsum`:

```
> Mu <- c(0, cumsum(diff(t) * mid(mu)))
> head(cbind(t, Mu))
     t           Mu
  0.00 0.0000000000
2 0.01 0.0006902169
3 0.02 0.0013770686
4 0.03 0.0020605718
5 0.04 0.0027407429
6 0.05 0.0034175987
```

Note the first value which is the integral from 0 to 0, so by definition 0.

3.6 Cumulative risks from parametric models

Here is the theory where we need integration: The cumulative risk of `OAD` at time t is:

$$R_{\texttt{OAD}}(t) = \int_0^t \lambda(u) S(u)\,\mathrm{d}u = \int_0^t \lambda(u) \exp\left(-\int_0^u \lambda(s) + \mu(s)\,\mathrm{d}s\right)\mathrm{d}u$$

where λ is the rate of `OAD` (`lam`), and μ the mortality rate (`mrt`). A similar formula is obtained for the cumulative risk of `Dead` (that is "dead without OAD"), by exchanging λ and μ.

The prectical calculation of these quantities are on pages 214–5 of "Epidemiology with R".

10. This means that if we have estimates of λ and μ as functions of time, we can derive the cumulative risks. In practice this will be by numerical integration; compute the rates at closely spaced intervals and evaluate the integrals as sums. This is easy, but what is not so easy is to come up with confidence intervals for the cumulative risks. Confidence intervals are most conveniently produced by simulation ("parametric bootstrap" as some say):
    (a) generate a random vector from the multivariate normal distribution with mean equal to the parameters of the model, and variance-covariance equal to the estimated variance-covariance of the parameter estimates (the Hessian as it is called).
    (b) use this to generate a simulated set of rates ($\lambda(t)$, $\mu(t)$), evaluated a closely spaced times
    (c) use these in numerical integration to derive state probabilities at these times
    (d) repeat 1000 times, say, to obtain 1000 sets of state probabilities at these times
    (e) use these to derive confidence intervals for the state probabilities as the 2.5 and 97.5 percentiles of the state probabilities at each time

    This machinery is implemented in the function `ci.Crisk`

```
> cR <- ci.Crisk(mods = list(OAD =  OAD.glm,
+                            Dead = Dead.glm),
+                nd = nd)
NOTE: Times are assumed to be in the column tfd at equal distances of 0.01
> str(cR)
List of 4
 $ Crisk: num [1:1501, 1:3, 1:3] 1 0.991 0.983 0.975 0.968 ...
  ..- attr(*, "dimnames")=List of 3
  .. ..$ tfd  : chr [1:1501] "0" "0.01" "0.02" "0.03" ...
  .. ..$ cause: chr [1:3] "Surv" "OAD" "Dead"
  .. ..$      : chr [1:3] "50%" "2.5%" "97.5%"
 $ Srisk: num [1:1501, 1:2, 1:3] 0 0.000692 0.001374 0.002048 0.002713 ...
  ..- attr(*, "dimnames")=List of 3
  .. ..$ tfd  : chr [1:1501] "0" "0.01" "0.02" "0.03" ...
  .. ..$ cause: chr [1:2] "Dead" "Dead+OAD"
  .. ..$      : chr [1:3] "50%" "2.5%" "97.5%"
 $ Stime: num [1:1501, 1:3, 1:3] 0 0.00996 0.01983 0.02963 0.03934 ...
  ..- attr(*, "dimnames")=List of 3
  .. ..$ tfd  : chr [1:1501] "0" "0.01" "0.02" "0.03" ...
  .. ..$ cause: chr [1:3] "Surv" "OAD" "Dead"
  .. ..$      : chr [1:3] "50%" "2.5%" "97.5%"
 $ time : num [1:1501] 0 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 ...
 - attr(*, "int")= num 0.01
```

There are 4 components of the results, the three first are simply arrays with 2 or 3 functions of time with confidence intervals.
So now plot the cumulative *risks* of being in each of the states (the `Crisk` component):

```
> matshade(as.numeric(dimnames(cR$Crisk)[[1]]),
+          cbind(cR$Crisk[,1,],
+                cR$Crisk[,2,],
+                cR$Crisk[,3,]), plot = TRUE,
+          lwd = 2, col = c("limegreen","red","black"))
```

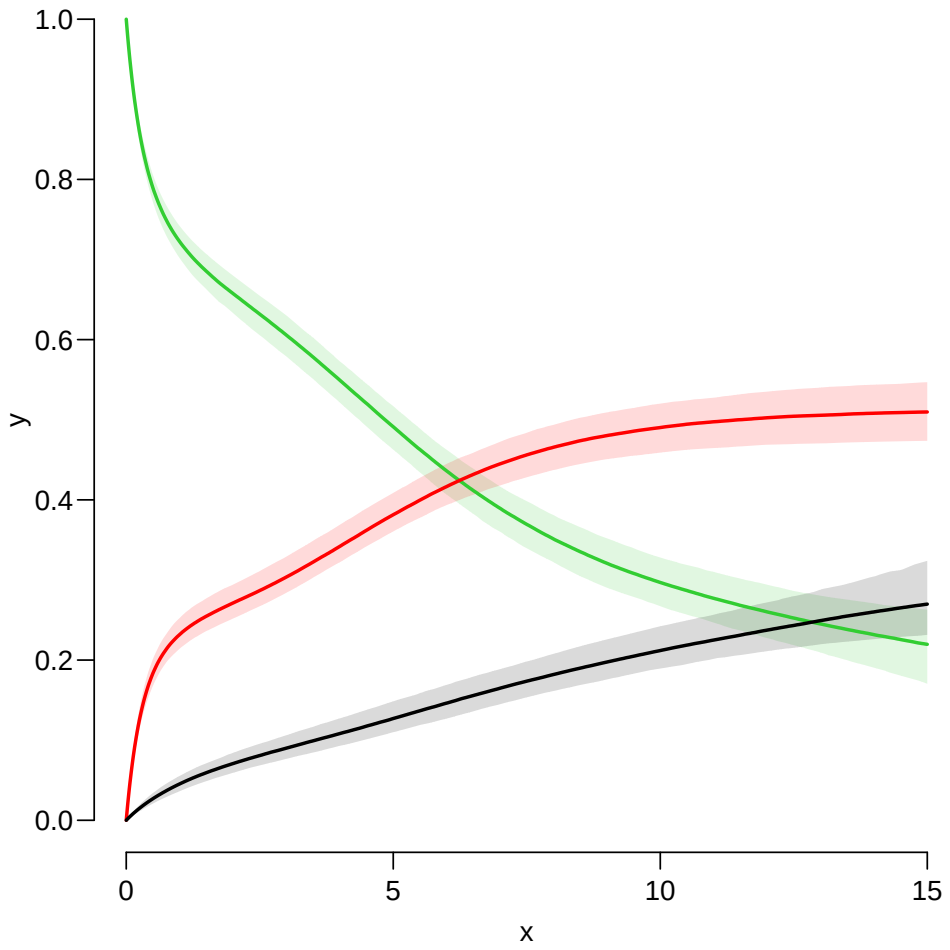


Figure 3.6: *Cumulative risks of being in each of the states* `DM` *(green),* `OAD` *(red) and* `Dead`
`../graph/cmpr-crisk`

11. Plot the stacked probabilities (`matrix` `2` `polygons`):
    `> mat2pol(cR$Crisk[,3:1,1], col = c("forestgreen","red","black")[3:1])`
    The component `Srisk` has the confidence limits of the stacked probabilities, add these to the plot, for example by semi-transparent shades or dotted lines,
    If you are really entrepreneurial, devise a function that will take the `Srisk` component of `cR` and produce a stacked plot with shaded confidence limits; here is the stacked

plot:

```
> matshade(as.numeric(dimnames(cR$Srisk)[[1]]),
+          cbind(cR$Srisk[,1,],
+                cR$Srisk[,2,]), plot = TRUE,
+          lwd = 2, col = c("black","red"),
+          ylim = 0:1, yaxs = "i")
```

Note the `yaxs = "i"`...

You may want to look at `adjustcolor` or `rgb` to see how to make semi-transparent colours.

3.7 Expected life time: using simulated objects

12. It is not only the cumulative risks of being in different states that my be of interest, the *integrals* — area under the cumulative risk curves are of interest too. The cumulative risks are probabilities, so dimensionless, which means that integrals of these along the time-axis will have dimension time; they will represent the expected time spent in each of the states.
    The areas between the lines (up to say 10 years) are `expected sojourn times`, that is:
    - expected years alive without OAD
    - expected years lost to death without OAD
    - expected years after OAD, including years dead after OAD

    Not all of these are of direct relevance; actually only the first may be so. They are available (with simulation-based confidence intervals) in the component of `cR`, `Stime` (Sojourn `time`).
    A relevant quantity would be the expected time alive without OAD during the first 5, 10 and 15 years (remember that the first dimension of `Stime` is in unots of 1/100 year):

```
> str(cR$Stime)
 num [1:1501, 1:3, 1:3] 0 0.00996 0.01983 0.02963 0.03934 ...
 - attr(*, "dimnames")=List of 3
  ..$ tfd  : chr [1:1501] "0" "0.01" "0.02" "0.03" ...
  ..$ cause: chr [1:3] "Surv" "OAD" "Dead"
  ..$      : chr [1:3] "50%" "2.5%" "97.5%"
> round(cR$Stime[1:3*500+1,"Surv",], 1)
tfd  50% 2.5% 97.5%
  5  3.2  3.1   3.3
  10 5.1  4.9   5.3
  15 6.4  6.0   6.8
```

13. We can also compute the expected fraction of the first 5, 10, 15 years alive:

```
> (mY <- matrix(rep(1:3 * 5, 3), 3, 3))
     [,1] [,2] [,3]
[1,]    5    5    5
[2,]   10   10   10
[3,]   15   15   15
> round(100 * cR$Stime[1:3*500+1,"Surv",] / mY, 1)
tfd   50% 2.5% 97.5%
  5  64.7 62.5  66.8
  10 51.3 49.1  53.4
```

```
  15 42.7 40.3  45.0
```

This can also be shown as a function of time; how large a fraction of the first t time can a person expect to be alive, for t ranging from 0 to 15 years:

```
> time <- as.numeric(dimnames(cR$Stime)[[1]]) / 100
> matshade(time, cR$Stime[,"Surv",] /
+                cbind(time,
+                      time,
+                      time) * 100,
+          plot=TRUE,
+          ylim = 0:1*100, yaxs = "i", xaxs = "i")
```

Amend the plot with proper axis labels.

Chapter 4

Multistate models: steno2

Paraphernalia

First we load the relevant packages and set some options:

```
> library(survival)
> library(Epi)
> library(popEpi)
> # popEpi::splitMulti returns a data.frame rather than a data.table
> options("popEpi.datatable" = FALSE)
> library(tidyverse)
> # setwd("/home/bendix/teach/AdvCoh/courses/Aalborg.2022/pracs")
> # setwd("/home/bendix/teach/AdvCoh/courses/Nuuk.2022/pracs")
> getwd()
[1] "C:/Bendix/teach/AdvCoh/courses/Nuuk.2022/pracs"
> clear()
```

For later convenience we devise a function that prints a data frame with all its numerical values rounded—this is particularly useful for `Lexis` objects with time scales calendar time and say, age.

```
> nround <-
+ function(df, dec = 2)
+ {
+ wh.num <- sapply(df, is.numeric)
+ df[,wh.num] <- round(df[,wh.num], dec)
+ print(df)
+ }
```

4.1 Lexis object for steno2

1. Bring in the `steno2` dataset, and convert dates to `cal.yr` to get a natural unit of time (years—365.25 days, that is). Because of the way data were anonymized, the `doEnd` is not perfectly aligned to `doDth`, which we remedy on the fly by resetting `doEnd` if a `doDth` is known.

```
> data(steno2)
> steno2 <- transform(cal.yr(steno2),
+                     doEnd = ifelse(!is.na(doDth),
+                                    doDth,
+                                    doEnd))
```

```
> str(steno2)
'data.frame':        160 obs. of  14 variables:
 $ id      : num  1 2 3 4 5 6 7 8 9 10 ...
 $ allo    : Factor w/ 2 levels "Int","Conv": 1 1 2 2 2 2 2 1 1 1 ...
 $ sex     : Factor w/ 2 levels "F","M": 2 2 2 2 2 2 1 2 2 2 ...
 $ baseCVD : num  0 0 0 0 0 1 0 0 0 0 ...
 $ deathCVD: num  0 0 0 0 1 0 0 0 1 0 ...
 $ doBth   : 'cal.yr' num  1932 1947 1943 1945 1936 ...
 $ doDM    : 'cal.yr' num  1991 1982 1983 1977 1986 ...
 $ doBase  : 'cal.yr' num  1993 1993 1993 1993 1993 ...
 $ doCVD1  : 'cal.yr' num  2014 2009 2002 1995 1994 ...
 $ doCVD2  : 'cal.yr' num  NA 2009 NA 1997 1995 ...
 $ doCVD3  : 'cal.yr' num  NA 2010 NA 2003 1998 ...
 $ doESRD  : 'cal.yr' num  NaN NaN NaN NaN 1998 ...
 $ doEnd   : num  2015 2015 2002 2003 1998 ...
 $ doDth   : 'cal.yr' num  NA NA 2002 2003 1998 ...
```

2. Start by setting up a `Lexis` data frame for the entire observation time for each person; from entry (`doBase`, date of baseline) to exit, `doEnd`. Note that we call the initial state `Mic`(roalbuminuria), because all patients in the Steno2 study had this status at entry—it was one of the inclusion criteria:

```
> L2 <- Lexis(entry = list(per = doBase,
+                          age = doBase - doBth,
+                          tfi = 0),
+             exit = list(per = doEnd),
+       exit.status = factor(deathCVD + !is.na(doDth),
+                            labels=c("Mic","D(oth)","D(CVD)")),
+               id = id,
+             data = steno2)
NOTE: entry.status has been set to "Mic" for all.
> summary(L2, t = TRUE)
Transitions:
     To
From  Mic D(oth) D(CVD)  Records:  Events: Risk time:  Persons:
  Mic  67     55     38       160       93    2420.91       160

Timescales:
per age tfi
 ""  ""  ""
> boxes(L2, boxpos = TRUE, show.BE = TRUE)
```

How many deaths are there in the cohort?
Explain the coding of `exit.status`.
How many person-years?
What are the time scales?

3. In this set-up we can study the CVD and the non-CVD mortality rates, a classical competing risks problem, but we want in particular to see how the mortality rates depend on albuminuria status.
In order to allocate follow-up (person-time and events) to *current* albuminuria status we need to know when the persons change status; this is recorded in the data frame `st2alb`.
We will cut the follow-up at each date of albuminuria measurement allowing the patients to change between states `Norm`oalbuminuria, `Mic`roalbuminuria and

Macroalbuminuria at each of these dates, possibly several times per person. To this end we use the function `rcutLexis` (recurrent `cuts`), which requires a data frame of transitions with columns `lex.id`, `cut` and `new.state` — see `?rcutLexis`.
We change the scale of the date of transition to year by `cal.yr` (to align with the `per` variable in `L2`), and in order to comply with the requirements of `rcutLexis` rename the id variable `id` to `lex.id`, the date variable `doTr` to `cut` and the state variable `state` to `new.state`:

```
> data(st2alb)
> cut2 <- rename(cal.yr(st2alb),
+                lex.id = id,
+                   cut = doTr,
+             new.state = state)
> str(cut2)
'data.frame':        563 obs. of  3 variables:
 $ lex.id   : num  1 1 1 1 1 2 2 2 2 2 ...
 $ cut      : 'cal.yr' num  1993 1995 2000 2002 2007 ...
 $ new.state: Factor w/ 3 levels "Norm","Mic","Mac": 2 1 2 1 2 1 2 3 2 2 ...
> head(cut2)
  lex.id      cut new.state
1      1 1993.444       Mic
2      1 1995.361      Norm
3      1 2000.067       Mic
4      1 2001.984      Norm
5      1 2007.317       Mic
6      2 1993.786      Norm
```

How many persons are in the `cut2` data frame?

```
> with(cut2, addmargins(table(table(lex.id))))
  1   2   3   4   5 Sum
  4  25  40  46  41 156
```

Explain the entries in this table.

4. Now cut at intermediate transition times (note that `rcutLexis` assumes that values in the `cut` column refer to the first timescale by default, and the first of the timescales in `L2` is `per`:

```
> L3 <- rcutLexis(L2, cut2)
> summary(L3)
Transitions:
     To
From   Mic Norm Mac D(oth) D(CVD)  Records:  Events: Risk time:  Persons:
  Mic  299   72  65     27     13       476      177    1383.56       160
  Norm  31   90   5     14      7       147       57     608.75        69
  Mac   20    3  44     14     18        99       55     428.60        64
  Sum  350  165 114     55     38       722      289    2420.91       160
> boxes(L3, boxpos = TRUE, cex = 0.8)
```

5. Note that there are transitions both ways between all three of `Norm`, `Mic` and `Mac`, which is a bit illogical, since we have a natural ordering of states: `Norm` < `Mic` < `Mac`, so transitions from `Norm` to `Mac` (and vice versa) should go through `Mic`
In order to remedy this anomaly we find all transitions `Norm` → `Mac` and provide a transition `Norm` → `Mic` in between. And of course similarly for transitions `Mac` → `Norm`.
The relevant "jump" transitions are easily found:

```
> (jump <-
+ subset(L3, (lex.Cst == "Norm" & lex.Xst == "Mac") |
```

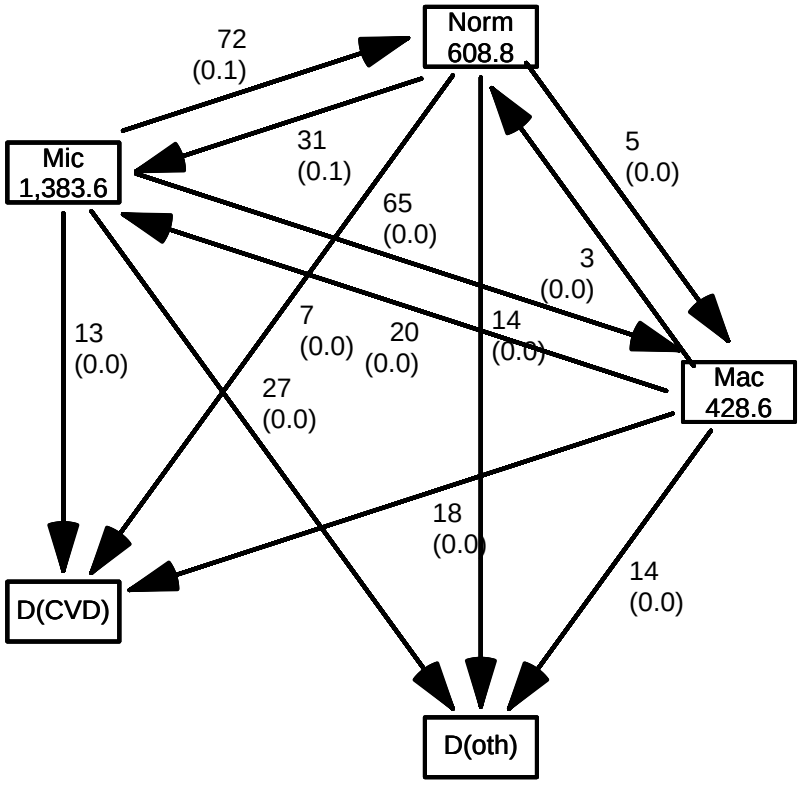


Figure 4.1: *The default lay-out of the 5 boxes placed on a circle, including the jumps directly between* `Norm` *and* `Mac`.

`../graph/ms-boxL3`

```
+            (lex.Xst == "Norm" & lex.Cst == "Mac"))[,
+        c("lex.id", "per", "lex.dur","lex.Cst", "lex.Xst")])
    lex.id      per    lex.dur lex.Cst lex.Xst
291     70 1999.487  2.6748802     Mac    Norm
353     86 2001.759 12.8158795    Norm     Mac
506    130 2000.910  1.8781656     Mac    Norm
511    131 1997.756  4.2354552    Norm     Mac
525    136 1997.214  0.4709103     Mac    Norm
526    136 1997.685  4.2436687    Norm     Mac
654    171 1996.390  5.3388090    Norm     Mac
676    175 2004.585  9.8836413    Norm     Mac
```

6. What we need to do for each of these "jumps" is to provide an extra transition to `Mic` at a time during the stay in either `Norm` or `Mac`, i.e. somewhere between `per` and `per` + `lex.dur` in these records; we choose a random time in the middle 80% between the dates:

```
> set.seed(1952)
> xcut <- select(transform(jump,
+                          cut = per + lex.dur * runif(per, 0.1, 0.9),
+                      new.state = "Mic"),
+                c(lex.id, cut, new.state))
> xcut
    lex.id      cut new.state
291     70 2001.789       Mic
353     86 2012.232       Mic
506    130 2001.488       Mic
511    131 2001.032       Mic
525    136 1997.610       Mic
526    136 2000.780       Mic
654    171 1997.057       Mic
676    175 2013.472       Mic
```

   How many extra records will be generated when cutting the follow-up?

7. Now make extra cuts (transitions to `Mic`) at these dates using `rcutLexis` with `xcut` on the `L3` object:

```
> L4 <- rcutLexis(L3, xcut)
> summary(L4)
Transitions:
     To
From   Mic Norm Mac D(oth) D(CVD)  Records:  Events: Risk time:  Persons:
  Mic  312   72  65     30     14       493      181    1437.39       160
  Norm  35   90   0     13      6       144       54     581.83        66
  Mac   22    0  41     12     18        93       52     401.70        60
  Sum  369  162 106     55     38       730      287    2420.91       160
```

   We see that there are no transitions directly between `Norm` and `Mac` in `L4`, so we can make a more intelligible plot of the transitions:

```
> opar <- par(bg="black",fg="white")
> par(opar)
> boxes(L4, boxpos = list(x = c(20,20,20,80,80),
+                         y = c(50,90,10,75,25)),
+           show.BE = "nz",
+           scale.R = 100, digits.R = 2,
+           cex = 0.9, pos.arr = 0.3)
```

   Explain the arguments used to `boxes`.

   Explain the numbers in the graph.

   Describe the overall effect of albuminuria on the two mortality rates.

With this multistate model (well, there is no model yet) set up we can look at mortality rates and see how they depend on the current albuminuria state, or look at the transition rates between the different albuminuria states and assess how these depend on covariates.

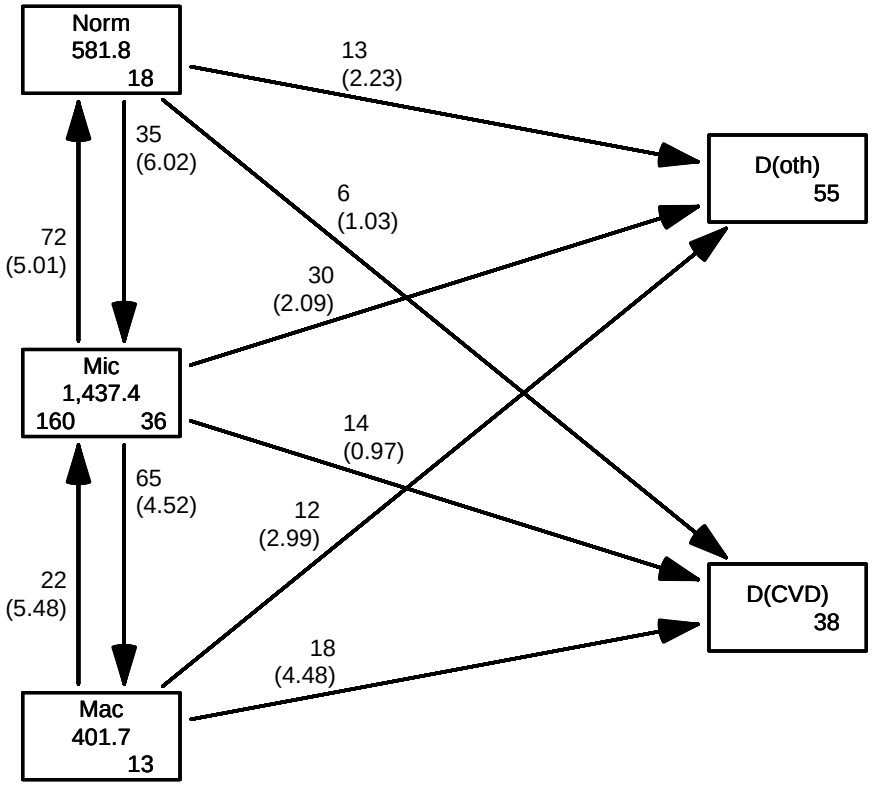


Figure 4.2: *Transitions between states in the Steno2 study.*

`../graph/ms-b4`

4.2 Transition rates: multiple time scales

8. We will model the transition rates with parametric functions, so we need to split the dataset along some time scale; we will use 3 month intervals (they should be sufficiently small to accommodate an assumption of constant rates in each interval):

```
> S4 <- splitMulti(L4, tfi = seq(0, 25, 1/2))
> summary(L4)
Transitions:
     To
From   Mic Norm Mac D(oth) D(CVD)  Records:  Events: Risk time:  Persons:
  Mic  312   72  65     30     14       493      181    1437.39       160
  Norm  35   90   0     13      6       144       54     581.83        66
  Mac   22    0  41     12     18        93       52     401.70        60
  Sum  369  162 106     55     38       730      287    2420.91       160
> summary(S4)
Transitions:
     To
From    Mic Norm Mac D(oth) D(CVD)  Records:  Events: Risk time:  Persons:
  Mic  3107   72  65     30     14      3288      181    1437.39       160
```

```
  Norm   35 1254   0     13      6      1308       54     581.83        66
  Mac    22    0 847     12     18       899       52     401.70        60
  Sum  3164 1326 912     55     38      5495      287    2420.91       160
```

We see that the number of events (transitions) and person-years are the same, in the two `Lexis` objects, but the number of records in `S4` is substantially larger than in `L4`.

9. We can now model the overall mortality rates as functions of age and duration (time since entry) using the defaults for `glm.Lexis` (this function call will trigger a warning):

```
> ma <- glm.Lexis(S4, ~ Ns(tfi, knots = seq( 0, 20, 5)) +
+                       Ns(age, knots = seq(50, 80, 10)) +
+                       lex.Cst)
stats::glm Poisson analysis of Lexis object S4 with log link:
Rates for transitions: Mic->D(oth), Norm->D(oth), Mac->D(oth), Mic->D(CVD), Norm->D(CV
> ci.exp(ma)
                                      exp(Est.)         2.5%        97.5%
(Intercept)                        0.002050421 0.0003671892 1.144975e-02
Ns(tfi, knots = seq(0, 20, 5))1    5.586238327 1.1524085205 2.707899e+01
Ns(tfi, knots = seq(0, 20, 5))2    3.948224386 0.9544630678 1.633219e+01
Ns(tfi, knots = seq(0, 20, 5))3   34.408040078 0.8997125880 1.315879e+03
Ns(tfi, knots = seq(0, 20, 5))4    0.466409150 0.1500745257 1.449530e+00
Ns(age, knots = seq(50, 80, 10))1  3.269829526 1.3358892679 8.003497e+00
Ns(age, knots = seq(50, 80, 10))2 11.582318649 1.4600392048 9.188117e+01
Ns(age, knots = seq(50, 80, 10))3 12.640207886 5.6379476934 2.833919e+01
lex.CstNorm                        1.041469079 0.6062915725 1.789004e+00
lex.CstMac                         1.772156120 1.1036543651 2.845580e+00
```

The warning triggered here just tells you that you are modeling the occurrence of any type of death, which amounts to modeling of the sum of CVD and non-CVD death rates.

The model structure with `lex.Cst` as an additive term is assuming that the overall mortality rates are proportional between states of albuminuria.

What are the mortality rate-ratios (hazard ratios), what ratios do they refer to: rates of what between which groups?

10. The default for `glm.Lexis` is to model all transitions to absorbing states which in this case are the two "dead" states, `D(oth)` and `D(CVD)`.

The `glm.Lexis` above is just a convenience wrapper for:

```
> m1 <- glm(cbind(lex.Xst %in% c("D(oth)", "D(CVD)")
+                 & lex.Cst != lex.Xst,
+                 lex.dur)
+           ~ Ns(tfi, knots = seq( 0, 20, 5)) +
+             Ns(age, knots = seq(50, 80, 10)) +
+             lex.Cst,
+           family = poisreg,
+             data = subset(S4, lex.Cst %in% c("Norm","Mic","Mac")))
```

This will also give the same results as:

```
> m2 <- glm((lex.Xst %in% c("D(oth)", "D(CVD)") & lex.Cst != lex.Xst)
+           ~ Ns(tfi, knots = seq( 0, 20, 5)) +
+             Ns(age, knots = seq(50, 80, 10)) +
+             lex.Cst,
+           offset = log(lex.dur),
+           family = poisson,
+             data = subset(S4, lex.Cst %in% c("Norm","Mic","Mac")))
```

—note the difference between the families `poisreg` and `poisson`: `poisreg` enters events and person-time more logically as part of the outcome, whereas `poisson` enters events as the response and person-years (`lex.dur`) via the `offset`,

11. The parameters from any of the formulations are on the log-scale so we want to see them exponentiated, so on the rate-scale:

```
> round(ci.exp(ma), 2)
                                    exp(Est.) 2.5%   97.5%
(Intercept)                              0.00 0.00    0.01
Ns(tfi, knots = seq(0, 20, 5))1          5.59 1.15   27.08
Ns(tfi, knots = seq(0, 20, 5))2          3.95 0.95   16.33
Ns(tfi, knots = seq(0, 20, 5))3         34.41 0.90 1315.88
Ns(tfi, knots = seq(0, 20, 5))4          0.47 0.15    1.45
Ns(age, knots = seq(50, 80, 10))1        3.27 1.34    8.00
Ns(age, knots = seq(50, 80, 10))2       11.58 1.46   91.88
Ns(age, knots = seq(50, 80, 10))3       12.64 5.64   28.34
lex.CstNorm                              1.04 0.61    1.79
lex.CstMac                               1.77 1.10    2.85
```

We see there is a higher mortality in the `Mac` state but no discernible difference between the `Mic` and the `Norm` states.

It can be formally tested whether the three states carry the same mortality using a Wald test (testing whether the `Norm` and `Mac` parameters both are 0 on the log-scale):

```
> Wald(ma, subset = "lex.Cst")
     Chisq       d.f.          P
6.11103822 2.00000000 0.04709827
```

What is the meaning of this test (i.e. the null hypothesis).

12. Now do the same analysis for the two causes of death separately, using the `to` argument to `glm.Lexis`:

```
> mo <- glm.Lexis(S4, ~ Ns(tfi, knots = seq( 0, 20, 5)) +
+                       Ns(age, knots = seq(50, 80, 10)) +
+                       lex.Cst,
+                   to = "D(oth)")
stats::glm Poisson analysis of Lexis object S4 with log link:
Rates for transitions: Mic->D(oth), Norm->D(oth), Mac->D(oth)
> round(ci.exp(mo), 3)
                                    exp(Est.)  2.5%        97.5%
(Intercept)                             0.000 0.000 7.000000e-03
Ns(tfi, knots = seq(0, 20, 5))1       115.151 2.779 4.770588e+03
Ns(tfi, knots = seq(0, 20, 5))2        30.897 1.466 6.512260e+02
Ns(tfi, knots = seq(0, 20, 5))3     23342.027 4.716 1.155320e+08
Ns(tfi, knots = seq(0, 20, 5))4         1.737 0.302 1.000100e+01
Ns(age, knots = seq(50, 80, 10))1       2.745 0.901 8.360000e+00
Ns(age, knots = seq(50, 80, 10))2       2.053 0.208 2.028900e+01
Ns(age, knots = seq(50, 80, 10))3      12.979 4.637 3.633200e+01
lex.CstNorm                             1.000 0.518 1.929000e+00
lex.CstMac                              0.994 0.503 1.965000e+00
> mC <- glm.Lexis(S4, ~ Ns(tfi, knots = seq( 0, 20, 5)) +
+                       Ns(age, knots = seq(50, 80, 10)) +
+                       lex.Cst,
+                   to = "D(CVD)")
stats::glm Poisson analysis of Lexis object S4 with log link:
Rates for transitions: Mic->D(CVD), Norm->D(CVD), Mac->D(CVD)
> round(ci.exp(mC), 3)
                                    exp(Est.)  2.5%     97.5%
```

```
(Intercept)                              0.001 0.000     0.012
Ns(tfi, knots = seq(0, 20, 5))1          1.079 0.165     7.069
Ns(tfi, knots = seq(0, 20, 5))2          1.932 0.305    12.254
Ns(tfi, knots = seq(0, 20, 5))3          1.143 0.018    73.365
Ns(tfi, knots = seq(0, 20, 5))4          0.129 0.016     1.065
Ns(age, knots = seq(50, 80, 10))1        6.419 1.069    38.564
Ns(age, knots = seq(50, 80, 10))2      417.853 1.795 97264.177
Ns(age, knots = seq(50, 80, 10))3       14.997 3.545    63.443
lex.CstNorm                              1.091 0.416     2.859
lex.CstMac                               3.513 1.719     7.179
```

What is the conclusion w.r.t. the effect of albuminuria state on the two cause-specific mortality rates?

13. Make a formal test of relevant hypotheses using `Wald`.

```
> Wald(mo, subset = "Cst")
       Chisq         d.f.            P
0.0002966161 2.0000000000 0.9998517030
> Wald(mC, subset = "Cst")
       Chisq         d.f.            P
13.764652275  2.000000000  0.001025755
```

What is the formal w.r.t. mortality dependence on albuminuria status?

14. We can show how fitted mortality rates look for persons currently in state `Mic` entering the study at a set of specific ages. The entry ages are in the vector `L2$age`:

```
> summary(L2$age)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  37.39   48.52   56.60   55.13   61.06   67.50
```

Based on this we shall use ages 45, 55 and 65, and show mortality rates for persons entering at these ages. We will show the rates as functions of their current age. We need a prediction data frame, with values for all variables in the model, (current) `age` and time from entry, `tfi`. Here `expand.grid` is our friend:

```
> expand.grid(tfi = c(NA, seq(0, 20, 5)),
+             ain = c(45, 55, 65))
   tfi ain
1   NA  45
2    0  45
3    5  45
4   10  45
5   15  45
6   20  45
7   NA  55
8    0  55
9    5  55
10  10  55
11  15  55
12  20  55
13  NA  65
14   0  65
15   5  65
16  10  65
17  15  65
18  20  65
```

—it will give all combinations of the values in the vectors supplied as a `data.frame`. The `NA`s are there for plotting purposes— we get a break in plotted curves if there is an

`NA` in the data. We want the `tfi` points to be closer than in the illustrative example:

```
> prf <- transform(expand.grid(tfi = c(NA, seq(0, 20, 0.5)),
+                              ain = c(45, 55, 65))[-1,],
+                  age = ain + tfi,
+              lex.Cst = "Mic")
> head(prf)
  tfi ain  age lex.Cst
2 0.0  45 45.0     Mic
3 0.5  45 45.5     Mic
4 1.0  45 46.0     Mic
5 1.5  45 46.5     Mic
6 2.0  45 47.0     Mic
7 2.5  45 47.5     Mic
> prf[40:44,]
    tfi ain  age lex.Cst
41 19.5  45 64.5     Mic
42 20.0  45 65.0     Mic
43   NA  55   NA     Mic
44  0.0  55 55.0     Mic
45  0.5  55 55.5     Mic
> matshade(prf$age, cbind(ci.pred(mo, prf),
+                         ci.pred(mC, prf)) * 100,
+          lwd = 3, col = c("black","blue"),
+          log = "y", ylim = c(0.01,50), plot = TRUE)
```

OTE: `matshade` uses `polygon` internally, and if the polygon—her the confidence limits—are too far outside the plotting area, they will not show up. Increase the `ylim` to see what is the matter.

The rates of death from other causes is very small at the beginning and increases steeply over the first 5 years of follow-up, while the CVD mortality is pretty stable with a foreseeable increase by age.

Give an informal description of the curves, and a possible reason for the shape of the curves.

15. We can show the impact of albuminuria state on the mortality rates in a 3-panel layout:

```
> par(mfrow=c(1,3))
> for(st in c("Norm","Mic","Mac"))
+    {
+ matshade(prf$age, cbind(ci.pred(mo, transform(prf, lex.Cst = st)),
+                         ci.pred(mC, transform(prf, lex.Cst = st))) * 100,
+          lwd = 3, col = c("black","blue"),
+          log = "y", ylim = c(0.05,50), plot = TRUE)
+ text(60, 50, st, adj = 0)
+    }
> # the matshade uses polygon that requires the shades to be inside so
> # we replace small numbers by 0.05 - the 0.05 must be second arg
> for(st in c("Norm","Mic","Mac"))
+    {
+ matshade(prf$age, pmax(
+                   cbind(ci.pred(mo, transform(prf, lex.Cst = st)),
+                         ci.pred(mC, transform(prf, lex.Cst = st))) * 100,
+                         0.05),
+          lwd = 3, col = c("black","blue"),
+          log = "y", ylim = c(0.05,50), plot = TRUE)
```

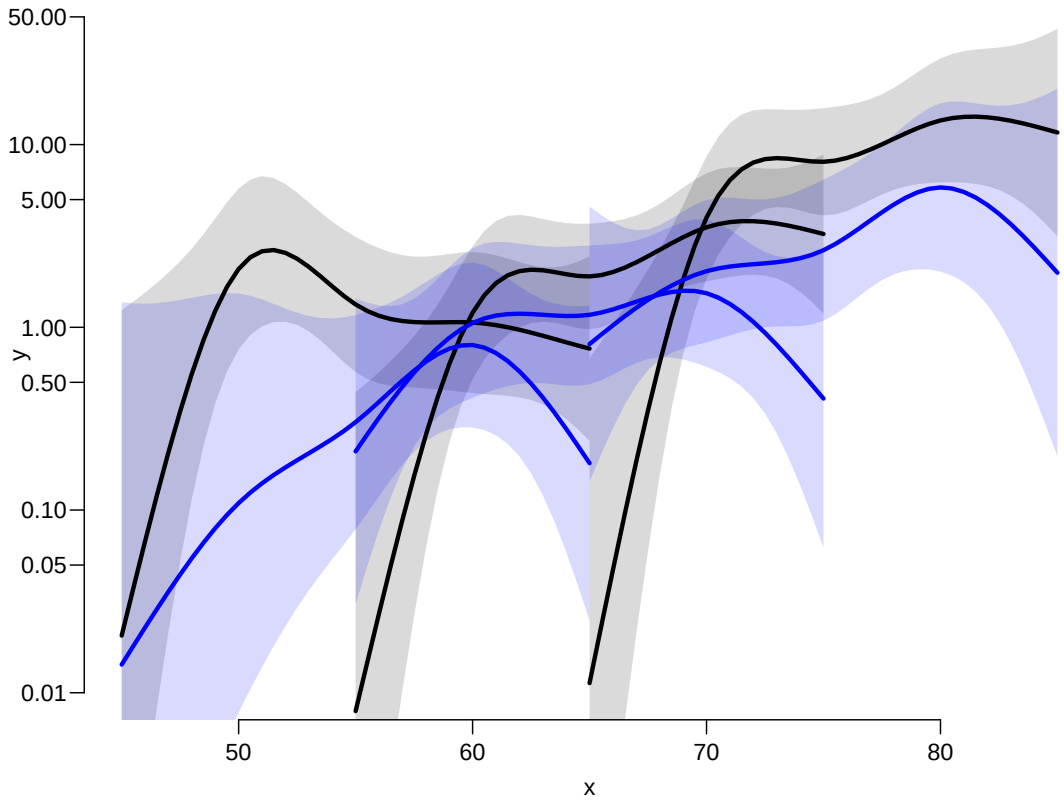


Figure 4.3: *CVD mortality rates (blue) and non-CVD mortality rates (black), with 95% confidence intervals as shades. Curve represent persons entering the study at ages 45, 55 and 65 respectively. N* `../graph/ms-mort1`

```
+ text(60, 50, st, adj = 0)
+    }
```

How are the curves in the three panels related?
Describe the effect of albuminuria status on the two types of mortality.
How can you see this from the model parameters?

4.3 State probabilities

We would like to see how the probabilities of being in each of the states in figure 4.2 look as a function of time since entry, and we will in particular be interested in how this depends on `allo`, the allocation to intensified or standard treatment.

4.3.1 Models for transition rates

Thus we will need models for 1) the cause-specific mortality rates and 2) transition rates between albuminuria states. And of course models which all include the effect of `allo` (treatment allocation).

We already fitted models for the mortality rates, but here we refit them in a slightly different guise.

Mortality rates

16. We first model the mortality rates using a proportional hazards model, but allowing different mortality between the two allocation groups (in `allo`), and the three albuminuria states (in `lex.Cst`):

```
> mix <- glm.Lexis(S4, ~ Ns(tfi, knots = seq( 0, 20, 5)) +
+                        Ns(age, knots = seq(50, 80, 10)) +
+                        lex.Cst * allo,
+                  to = "D(oth)")
stats::glm Poisson analysis of Lexis object S4 with log link:
Rates for transitions: Mic->D(oth), Norm->D(oth), Mac->D(oth)
> round(ci.exp(mix), 3)
                                    exp(Est.)  2.5%        97.5%
(Intercept)                             0.000 0.000 5.000000e-03
Ns(tfi, knots = seq(0, 20, 5))1       138.431 3.177 6.032407e+03
Ns(tfi, knots = seq(0, 20, 5))2        36.322 1.653 7.981850e+02
Ns(tfi, knots = seq(0, 20, 5))3     35690.096 6.479 1.965958e+08
Ns(tfi, knots = seq(0, 20, 5))4         2.183 0.378 1.259800e+01
Ns(age, knots = seq(50, 80, 10))1       2.746 0.909 8.295000e+00
Ns(age, knots = seq(50, 80, 10))2       1.627 0.159 1.666400e+01
Ns(age, knots = seq(50, 80, 10))3      11.953 4.268 3.347400e+01
lex.CstNorm                             1.039 0.388 2.786000e+00
lex.CstMac                              1.686 0.665 4.275000e+00
alloConv                                1.931 0.927 4.022000e+00
lex.CstNorm:alloConv                    0.929 0.244 3.544000e+00
lex.CstMac:alloConv                     0.336 0.086 1.314000e+00
```

We would however like to see the allocation effect on mortality separately for each albuminuria state; this is done by the "`/`" operator in the model formula (pronounced `allo` effect *within* `lex.Cst`):

```
> mox <- glm.Lexis(S4, ~ Ns(tfi, knots = seq( 0, 20, 5)) +
+                        Ns(age, knots = seq(50, 80, 10)) +
+                        lex.Cst / allo,
+                  to = "D(oth)")
stats::glm Poisson analysis of Lexis object S4 with log link:
Rates for transitions: Mic->D(oth), Norm->D(oth), Mac->D(oth)
> round(ci.exp(mox), 3)
                                    exp(Est.)  2.5%        97.5%
(Intercept)                             0.000 0.000 5.000000e-03
Ns(tfi, knots = seq(0, 20, 5))1       138.431 3.177 6.032407e+03
Ns(tfi, knots = seq(0, 20, 5))2        36.322 1.653 7.981850e+02
Ns(tfi, knots = seq(0, 20, 5))3     35690.096 6.479 1.965958e+08
Ns(tfi, knots = seq(0, 20, 5))4         2.183 0.378 1.259800e+01
Ns(age, knots = seq(50, 80, 10))1       2.746 0.909 8.295000e+00
Ns(age, knots = seq(50, 80, 10))2       1.627 0.159 1.666400e+01
Ns(age, knots = seq(50, 80, 10))3      11.953 4.268 3.347400e+01
lex.CstNorm                             1.039 0.388 2.786000e+00
lex.CstMac                              1.686 0.665 4.275000e+00
lex.CstMic:alloConv                     1.931 0.927 4.022000e+00
lex.CstNorm:alloConv                    1.794 0.590 5.455000e+00
lex.CstMac:alloConv                     0.649 0.204 2.065000e+00
> c(deviance(mox), deviance(mix))
```

```
[1] 554.6063 554.6063
```

The use of the deviance gives a good indication that the models fitted actually *are* the same model, just differently parametrized.

What is the meaning of the parameters?

17. We also need a similar model for the CVD-mortality:

```
> mCx <- glm.Lexis(S4, ~ Ns(tfi, knots = seq( 0, 20, 5)) +
+                        Ns(age, knots = seq(50, 80, 10)) +
+                        lex.Cst / allo,
+                  to = "D(CVD)")
stats::glm Poisson analysis of Lexis object S4 with log link:
Rates for transitions: Mic->D(CVD), Norm->D(CVD), Mac->D(CVD)
> round(ci.exp(mCx), 3)
                                    exp(Est.)  2.5%      97.5%
(Intercept)                             0.000 0.000      0.009
Ns(tfi, knots = seq(0, 20, 5))1         0.928 0.141      6.105
Ns(tfi, knots = seq(0, 20, 5))2         2.202 0.357     13.586
Ns(tfi, knots = seq(0, 20, 5))3         1.012 0.016     65.082
Ns(tfi, knots = seq(0, 20, 5))4         0.110 0.012      0.976
Ns(age, knots = seq(50, 80, 10))1       6.836 1.113     41.984
Ns(age, knots = seq(50, 80, 10))2     558.246 2.052 151860.752
Ns(age, knots = seq(50, 80, 10))3      20.881 4.798     90.877
lex.CstNorm                             1.244 0.307      5.044
lex.CstMac                              1.544 0.380      6.272
lex.CstMic:alloConv                     1.684 0.579      4.894
lex.CstNorm:alloConv                    1.392 0.276      7.016
lex.CstMac:alloConv                     4.880 1.372     17.355
```

What is the conclusion for the intervention effect on CVD mortality rates?

Albuminuria state rates

For a complete description of transitions in the model we also need models for the transitions between albuminuria states.

18. We will use different models for `det`erioration and `imp`rovement in albuminuria (arrow up or down in figure 4.2). Again the modeling is a bit simplified by `glm.Lexis`:

```
> det <- glm.Lexis(S4, ~ Ns(tfi, knots = seq( 0, 20, 5)) +
+                        Ns(age, knots = seq(50, 80, 10)) +
+                        lex.Cst / allo,
+                  from = c("Norm","Mic"),
+                    to = c("Mic","Mac"))
stats::glm Poisson analysis of Lexis object S4 with log link:
Rates for transitions: Norm->Mic, Mic->Mac
> imp <- glm.Lexis(S4, ~ Ns(tfi, knots = seq( 0, 20, 5)) +
+                        Ns(age, knots = seq(50, 80, 10)) +
+                        lex.Cst / allo,
+                  from = c("Mic","Mac"),
+                    to = c("Norm","Mic"))
stats::glm Poisson analysis of Lexis object S4 with log link:
Rates for transitions: Mic->Norm, Mac->Mic
> round(ci.exp(det), 3)
                                    exp(Est.)  2.5%  97.5%
(Intercept)                             0.030 0.015  0.060
Ns(tfi, knots = seq(0, 20, 5))1         0.606 0.232  1.584
Ns(tfi, knots = seq(0, 20, 5))2         0.264 0.075  0.931
```

```
Ns(tfi, knots = seq(0, 20, 5))3       0.243 0.041  1.440
Ns(tfi, knots = seq(0, 20, 5))4       0.218 0.061  0.784
Ns(age, knots = seq(50, 80, 10))1     2.029 0.852  4.831
Ns(age, knots = seq(50, 80, 10))2     3.477 0.927 13.042
Ns(age, knots = seq(50, 80, 10))3     2.712 0.762  9.645
lex.CstNorm                           2.560 1.448  4.525
lex.CstMic:alloConv                   1.964 1.178  3.277
lex.CstNorm:alloConv                  0.488 0.221  1.080
> round(ci.exp(imp), 3)
                                   exp(Est.)  2.5% 97.5%
(Intercept)                            0.207 0.131 0.326
Ns(tfi, knots = seq(0, 20, 5))1        0.255 0.079 0.825
Ns(tfi, knots = seq(0, 20, 5))2        0.059 0.009 0.383
Ns(tfi, knots = seq(0, 20, 5))3        0.042 0.007 0.240
Ns(tfi, knots = seq(0, 20, 5))4        0.201 0.039 1.050
Ns(age, knots = seq(50, 80, 10))1      0.825 0.281 2.420
Ns(age, knots = seq(50, 80, 10))2      0.351 0.070 1.763
Ns(age, knots = seq(50, 80, 10))3      0.583 0.070 4.873
lex.CstMac                             1.064 0.469 2.415
lex.CstMic:alloConv                    0.526 0.324 0.855
lex.CstMac:alloConv                    1.338 0.543 3.294
> round(ci.exp(det, subset="al"), 1)
                     exp(Est.) 2.5% 97.5%
lex.CstMic:alloConv        2.0  1.2   3.3
lex.CstNorm:alloConv       0.5  0.2   1.1
> round(ci.exp(imp, subset="al"), 1)
                    exp(Est.) 2.5% 97.5%
lex.CstMic:alloConv       0.5  0.3   0.9
lex.CstMac:alloConv       1.3  0.5   3.3
```

What was the meaning of "different models for `det` and `imp`"?
What do the parameters in the models represent?
What are the assumptions in the models?
Label the transitions in figure 4.2 with the models for each of the transitions.

4.3.2 Simulation of state probabilities

We now have statistical models for all transitions, two models for the cause specific mortality rates, and two models for transitions between albuminuria states.

The state probabilities that in principle can be derived from these are not trivial to compute, essentially they can only be computed by simulation[1].

19. But first we need an explicit specification of what transitions the models refer to, since the simulated transitions will be using predictions from these models. This is specified in a list of lists (remember what a list is??).
    There must be one element in the list for each transient state (of which we have 3):

```
> Tr <- list(Norm = list("Mic" = det,
+                        "D(oth)" = mox,
+                        "D(CVD)" = mCx),
+            Mic = list("Mac" = det,
+                       "Norm" = imp,
```

[1] A detailed description of the use of `simLexis` is available in the vignette in the `Epi` package, also available as http://bendixcarstensen.com/Epi/simLexis.pdf

```
+                     "D(oth)" = mox,
+                     "D(CVD)" = mCx),
+             Mac = list("Mic" = imp,
+                     "D(oth)" = mox,
+                     "D(CVD)" = mCx))
> lapply(Tr, names)
$Norm
[1] "Mic"    "D(oth)" "D(CVD)"

$Mic
[1] "Mac"    "Norm"   "D(oth)" "D(CVD)"

$Mac
[1] "Mic"    "D(oth)" "D(CVD)"
```

For example, the object `Tr$Norm$Mic` is a model for the transition rate `Norm` → `Mic`; we see that there are 10 entries in the specification of `Tr`, corresponding to each of the 10 transitions in the diagram in figure 4.2. Some of the entries in `Tr` point to the same model; all the models fitted were actually joint models for more than one transiton.

20. We can use the estimated rates to simulate the transition between states in a group of people with a given set of covariates.
    The simulated data can the be used to assess the probability of being in each of the states at a given time after entry to the study, separately for the two intervention groups if we wish.
    These probabilities depend on the age at entry to the study (because current age (`age`) and time since entry, (`tfi`) are both in the models).
    We can choose our initial cohort in (at least) two different ways:
    - Use a population with the same age-distribution as the entire study population ("population-averaged")
    - Evaluate the probabilities for a prespecified set of ages at entry ("conditional").

    What is needed for this is a data frame of persons indicating their initial status. `simLexis` will then simulate their individual trajectories through states (what transition takes place when) and produce a simulated cohort of persons in the form of a `Lexis` object. The initial (baseline) data frame should also be a `Lexis` object, but the values of `lex.Xst` and `lex.dur` need not be given, since these will be simulated.

Study population cohort

21. First construct a cohort with the same covariate distribution as the entire study for each of the allocation groups:

```
> ini <- L2[,c("per", "age", "tfi", "lex.Cst")]
> ini <- rbind(transform(ini, lex.Cst = "Mic", allo = "Int"),
+              transform(ini, lex.Cst = "Mic", allo = "Conv"))
> # lex.Cst must be a factor with the relevant set of levels
> ini$lex.Cst <- factor(ini$lex.Cst,
+                       levels = c("Norm","Mic","Mac","D(CVD)","D(oth)"))
> str(ini)
Classes 'Lexis' and 'data.frame':       320 obs. of  5 variables:
 $ per    : 'cal.yr' num  1993 1993 1993 1993 1993 ...
 $ age    : 'cal.yr' num  61.1 46.6 49.9 48.5 57.3 ...
```

```
 $ tfi    : num  0 0 0 0 0 0 0 0 0 0 ...
 $ lex.Cst: Factor w/ 5 levels "Norm","Mic","Mac",..: 2 2 2 2 2 2 2 2 2 2 ...
 $ allo   : chr  "Int" "Int" "Int" "Int" ...
 - attr(*, "breaks")=List of 3
  ..$ per: NULL
  ..$ age: NULL
  ..$ tfi: NULL
 - attr(*, "time.scales")= chr [1:3] "per" "age" "tfi"
 - attr(*, "time.since")= chr [1:3] "" "" ""
```

This will be the initial values in the cohort we follow through states—we have the starting state in `lex.Cst` and the covariates (at start): timescales (`per`, `age`, `tfi`) and the other covariates `allo`

22. First we simulate transitions from a large cohort that looks like the study population, say 10 copies of each person in the original data set (see `?simLexis`):

```
> set.seed(1952)
> system.time(
+ Sorg <- simLexis(Tr = Tr,  # models for each transition
+                 init = ini, # cohort of starters
+                    N = 10,  # how many copies of each person in ini
+              t.range = 20,  # how long should we simulate before censoring
+                n.int = 200))# how many intervals for evaluating rates
  bruger   system forlobet
   19.05     2.42    21.47
> summary(Sorg, t = T)
Transitions:
     To
From   Norm  Mic  Mac D(CVD) D(oth)  Records:  Events: Risk time:  Persons:
  Norm  398  640    0    119    282       1439     1041   11621.26      1310
  Mic  1439  622 1311    279    580       4231     3609   27091.33      3200
  Mac     0  391  302    380    238       1311     1009    7604.01      1217
  Sum  1837 1653 1613    778   1100       6981     5659   46316.59      3200

Timescales:
per age tfi
 ""  ""  ""
> nround(subset(Sorg, lex.id %in% 28:32), 2)
   lex.id     per   age  tfi lex.dur lex.Cst lex.Xst allo    cens
79     28 1993.37 49.94 0.00    0.78     Mic    Norm  Int 2013.37
80     28 1994.15 50.72 0.78    5.57    Norm     Mic  Int 2013.37
81     28 1999.72 56.29 6.35    1.10     Mic    Norm  Int 2013.37
82     28 2000.82 57.39 7.45    5.74    Norm  D(oth)  Int 2013.37
83     29 1993.37 49.94 0.00    2.11     Mic    Norm  Int 2013.37
84     29 1995.48 52.06 2.11    2.79    Norm  D(oth)  Int 2013.37
85     30 1993.37 49.94 0.00    7.15     Mic    Norm  Int 2013.37
86     30 2000.53 57.10 7.15    3.14    Norm  D(CVD)  Int 2013.37
87     31 1993.34 48.50 0.00    5.14     Mic    Norm  Int 2013.34
88     31 1998.47 53.64 5.14   14.86    Norm    Norm  Int 2013.34
89     32 1993.34 48.50 0.00    4.64     Mic     Mac  Int 2013.34
90     32 1997.98 53.14 4.64    0.65     Mac     Mic  Int 2013.34
91     32 1998.62 53.79 5.28   14.18     Mic  D(oth)  Int 2013.34
```

23. Describe in words how the simulated data looks, and what each record represents. What is it really we simulated?

```
> addmargins(table(table(Sorg$lex.id)))
   1    2    3    4    5    6    7    8  Sum
```

```
 874 1397  534  297   70   24    3    1 3200
```

What does this table mean?

24. Now we can just count how many of the original 1600 persons are in each of the states at each of a set of times; this is done by the function `nState`:

```
> system.time(
+ Nst <- nState(Sorg,
+                  at = seq(0, 20, 0.2),
+                from = 0,
+          time.scale = "tfi"))
  bruger   system forlobet
    1.16     0.02     1.17
> str(Nst)
 'table' int [1:101, 1:5] 0 88 168 230 290 336 384 438 488 529 ...
 - attr(*, "dimnames")=List of 2
  ..$ when : chr [1:101] "0" "0.2" "0.4" "0.6" ...
  ..$ State: chr [1:5] "Norm" "Mic" "Mac" "D(CVD)" ...
> head(Nst)
     State
when  Norm  Mic  Mac D(CVD) D(oth)
  0      0 3200    0      0      0
  0.2   88 3077   33      2      0
  0.4  168 2965   60      7      0
  0.6  230 2865   97      8      0
  0.8  290 2780  114     16      0
  1    336 2702  142     19      1
```

This is however not necessarily a relevant summary; we would be interested in seeing how things look in each of the allocation groups, `Int` and `Conv`.

```
> Nint <- nState(subset(Sorg, allo == "Int"),
+                  at = seq(0, 20, 0.1),
+                from = 0,
+         time.scale = "tfi")
> Nconv<- nState(subset(Sorg, allo == "Conv"),
+                  at = seq(0, 20, 0.1),
+                from = 0,
+         time.scale = "tfi")
> head(Nint)
     State
when  Norm  Mic  Mac D(CVD) D(oth)
  0      0 1600    0      0      0
  0.1   24 1569    6      1      0
  0.2   55 1533   11      1      0
  0.3   77 1506   15      2      0
  0.4  105 1472   21      2      0
  0.5  121 1443   34      2      0
> head(Nconv)
     State
when  Norm  Mic  Mac D(CVD) D(oth)
  0      0 1600    0      0      0
  0.1   18 1562   19      1      0
  0.2   33 1544   22      1      0
  0.3   41 1524   31      4      0
  0.4   63 1493   39      5      0
  0.5   76 1471   47      6      0
```

If we divide each of these by 1600, we get the probabilities of being in each if the states

at the different times since entry.

25. If we want the cumulated state probabilities over states we can derive these by `pState`, that yields a matrix with the cumulative state probabilities.

```
> Pint  <- pState(Nint )
> Pconv <- pState(Nconv)
>  str(Pint)
 'pState' num [1:201, 1:5] 0 0.015 0.0344 0.0481 0.0656 ...
 - attr(*, "dimnames")=List of 2
  ..$ when : chr [1:201] "0" "0.1" "0.2" "0.3" ...
  ..$ State: chr [1:5] "Norm" "Mic" "Mac" "D(CVD)" ...
> head(Pint)
     State
when      Norm      Mic      Mac D(CVD) D(oth)
  0   0.000000 1.000000 1.000000      1      1
  0.1 0.015000 0.995625 0.999375      1      1
  0.2 0.034375 0.992500 0.999375      1      1
  0.3 0.048125 0.989375 0.998750      1      1
  0.4 0.065625 0.985625 0.998750      1      1
  0.5 0.075625 0.977500 0.998750      1      1
```

Describe the structure of `Pst`.

26. There is a standard plotting method for a `pState` object, it will plot the stacked state probabilities stacked in the order given by the `perm` argument (not used here because they are already in the order we want):

```
> clr <- c("forestgreen", "orange", "red", "blue", gray(0.4))
> par(mfrow = c(1,2), mar=c(3,3,2,2))
> plot(Pint, col = clr, xlim = c(0, 20))
> # the survival curve
> lines(as.numeric(rownames(Pint)), Pint[,"Mac"], lwd = 4, col = "white")
> lines(as.numeric(rownames(Pint)), Pint[,"Mac"], lwd = 1, col = "black")
> text(rownames(Pint)[150],
+      Pint[150,] - diff(c(0, Pint[150,]))/2,
+      colnames(Pint), col = "white", cex = 0.8)
> plot(Pconv, col = clr, xlim = c(20, 0))
> # the survival curve
> lines(as.numeric(rownames(Pconv)), Pconv[,"Mac"], lwd = 4, col = "white")
> lines(as.numeric(rownames(Pconv)), Pconv[,"Mac"], lwd = 1, col = "black")
> text(rownames(Pconv)[150],
+      Pconv[150,] - diff(c(0, Pconv[150,]))/2,
+      colnames(Pint), col = "white", cex = 0.8)
> mtext(c("Intensive care","Conventional care"),
+       side = 3, at = c(1,3)/4, outer = TRUE, line = -2)
```

Redo the plot with proper labeling of axes, including units where needed.

27. Describe the results and conclude on the probabilities shown.

28. The plot 4.4 may be of limited interest; the probabilities here are really "the probability that a randomly chosen person from the Steno 2 study…". So we are referring to a universe that is not generalizable, the reference is to a particular distribution of ages at entry into the study. The plot is only partially relevant for showing the intervention effect, the absolute sizes of the state probabilities are strictly speaking irrelevant.

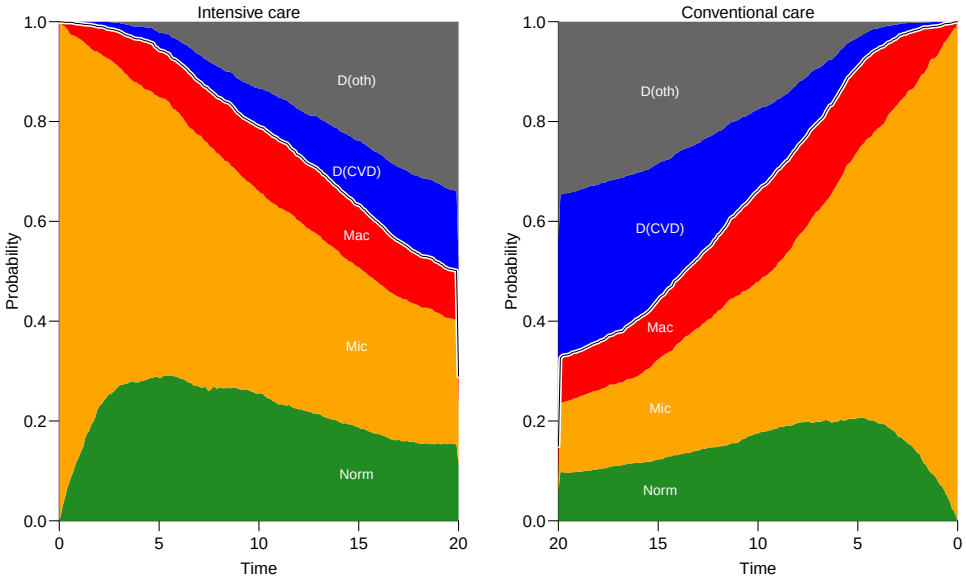


Figure 4.4: *State probabilities for the two intervention groups, for populations of the same structure as the original* total *Steno2 population.*

`../graph/ms-pStates`

Cohort with predefined variables

29. Even if we take the modeling background deeply serious and accept that occurrence rates depend only on current age (`age`), time since entry (`tfi`) and treatment allocation (`allo`), the assumption of age-distribution as in the Steno 2 study is quite absurd; who wants to refer to this? Often this is disguised in terms such as "population averaged". Therefore, it would be more relevant to show the results for a homogeneous population of persons at select ages at entry. This would just require a different `init` data frame:

```
> ini <- S4[1:10,c("lex.id", "per", "age", "tfi", "lex.Cst", "allo")]
> ini[,"lex.id"]  <- 1:10
> ini[,"per"]     <- 1993 # not used but it is a time scale in S4
> ini[,"age"]     <-
+ ini[,"ain"]     <- rep(seq(45,65,5), 2)
> ini[,"tfi"]     <- 0
> ini[,"lex.Cst"] <- factor("Mic",
+                           levels = c("Norm","Mic","Mac","D(CVD)","D(oth)"))
> ini[,"allo"]    <- factor(rep(c("Int","Conv"), each = 5))
> ini
   lex.id  per age tfi lex.Cst allo ain
1       1 1993  45   0     Mic  Int  45
2       2 1993  50   0     Mic  Int  50
3       3 1993  55   0     Mic  Int  55
4       4 1993  60   0     Mic  Int  60
5       5 1993  65   0     Mic  Int  65
6       6 1993  45   0     Mic Conv  45
7       7 1993  50   0     Mic Conv  50
8       8 1993  55   0     Mic Conv  55
9       9 1993  60   0     Mic Conv  60
```

```
10     10 1993  65   0     Mic Conv  65
> str(ini)
Classes 'Lexis' and 'data.frame':        10 obs. of  7 variables:
 $ lex.id : int  1 2 3 4 5 6 7 8 9 10
 $ per    : num  1993 1993 1993 1993 1993 ...
 $ age    : num  45 50 55 60 65 45 50 55 60 65
 $ tfi    : num  0 0 0 0 0 0 0 0 0 0
 $ lex.Cst: Factor w/ 5 levels "Norm","Mic","Mac",..: 2 2 2 2 2 2 2 2 2 2
 $ allo   : Factor w/ 2 levels "Conv","Int": 2 2 2 2 2 1 1 1 1 1
 $ ain    : num  45 50 55 60 65 45 50 55 60 65
 - attr(*, "time.scales")= chr [1:3] "per" "age" "tfi"
 - attr(*, "time.since")= chr [1:3] "" "" ""
 - attr(*, "breaks")=List of 3
  ..$ per: NULL
  ..$ age: NULL
  ..$ tfi: num [1:51] 0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 ...
```

Note that it is important that we enter the variable `lex.Cst` as a factor with the same levels as in the `Lexis` object `S4`, in the order we want the states when reporting results. `allo` must also be entered as a factor, otherwise it is not possible to compute predictions from the models where `allo` were included as a factor.

30. For each of these combinations of age (at entry) and treatment allocation we will simulate 100 persons (note that we are using the same transition rates, the models in `Tr`):

```
> system.time(
+ Sdef <- simLexis(Tr = Tr,
+                  init = ini,
+                     N = 100,
+               t.range = 20,
+                 n.int = 200))
  bruger   system forlobet
    5.21     0.19     5.39
> # str(Sdef)
> summary(Sdef)
Transitions:
     To
From   Norm Mic Mac D(CVD) D(oth)  Records:  Events: Risk time:  Persons:
  Norm  126 210   0     42     76       454      328    3667.94       407
  Mic   454 210 402     89    180      1335     1125    8756.18      1000
  Mac     0 125  87    122     68       402      315    2296.99       365
  Sum   580 545 489    253    324      2191     1768   14721.11      1000
> nround(head(Sdef))
  lex.id     per   age  tfi lex.dur lex.Cst lex.Xst allo ain cens
1      1 1993.00 45.00 0.00    0.06     Mic    Norm  Int  45 2013
2      1 1993.06 45.06 0.06   19.94    Norm    Norm  Int  45 2013
3      2 1993.00 45.00 0.00   20.00     Mic     Mic  Int  45 2013
4      3 1993.00 45.00 0.00   20.00     Mic     Mic  Int  45 2013
5      4 1993.00 45.00 0.00    3.94     Mic  D(oth)  Int  45 2013
6      5 1993.00 45.00 0.00    8.19     Mic     Mac  Int  45 2013
```

In real applications we would use 5000 or 10,000 replicates of each to minimize the simulation error.

31. Now we will repeat the graph above, but for the 10 combinations of age at enrollment (`ain`), and allocation; we start with the 45 year old allocated to `Int`:

```
> P45i <- nState(subset(Sdef, ain == 45 & allo == "Int"),
```

```
+                at = seq(0, 20, 0.1),
+              from = 0,
+         time.scale = "tfi")
> head(P45i)
     State
when  Norm Mic Mac D(CVD) D(oth)
  0      0 100   0      0      0
  0.1    3  97   0      0      0
  0.2    8  92   0      0      0
  0.3   11  89   0      0      0
  0.4   13  87   0      0      0
  0.5   14  86   0      0      0
> head(pState(P45i))
     State
when  Norm Mic Mac D(CVD) D(oth)
  0   0.00   1   1      1      1
  0.1 0.03   1   1      1      1
  0.2 0.08   1   1      1      1
  0.3 0.11   1   1      1      1
  0.4 0.13   1   1      1      1
  0.5 0.14   1   1      1      1
```

This should then be repeated for 4 other ages at enrollment and the two allocations, plus we will only store the state probabilities:

```
> P45c <- pState(nState(subset(Sdef, ain == 45 & allo == "Conv"),
+                at = seq(0, 20, 0.1),
+              from = 0,
+         time.scale = "tfi"))
> P45i <- pState(nState(subset(Sdef, ain == 45 & allo == "Int"),
+                at = seq(0, 20, 0.1),
+              from = 0,
+         time.scale = "tfi"))
> P50c <- pState(nState(subset(Sdef, ain == 55 & allo == "Conv"),
+                at = seq(0, 20, 0.1),
+              from = 0,
+         time.scale = "tfi"))
> P50i <- pState(nState(subset(Sdef, ain == 55 & allo == "Int"),
+                at = seq(0, 20, 0.1),
+              from = 0,
+         time.scale = "tfi"))
> P55c <- pState(nState(subset(Sdef, ain == 55 & allo == "Conv"),
+                at = seq(0, 20, 0.1),
+              from = 0,
+         time.scale = "tfi"))
> P55i <- pState(nState(subset(Sdef, ain == 55 & allo == "Int"),
+                at = seq(0, 20, 0.1),
+              from = 0,
+         time.scale = "tfi"))
> P60c <- pState(nState(subset(Sdef, ain == 55 & allo == "Conv"),
+                at = seq(0, 20, 0.1),
+              from = 0,
+         time.scale = "tfi"))
> P60i <- pState(nState(subset(Sdef, ain == 55 & allo == "Int"),
+                at = seq(0, 20, 0.1),
+              from = 0,
+         time.scale = "tfi"))
```

```
> P65c <- pState(nState(subset(Sdef, ain == 65 & allo == "Conv"),
+                at = seq(0, 20, 0.1),
+              from = 0,
+        time.scale = "tfi"))
> P65i <- pState(nState(subset(Sdef, ain == 65 & allo == "Int"),
+                at = seq(0, 20, 0.1),
+              from = 0,
+        time.scale = "tfi"))
```

32. Then we can plot these in a multiple lay-out:

```
> par(mfrow = c(5,2), mar = c(1,1,0,0),
+       oma = c(3,3,1,0), mgp=c(3,1,0)/1.6)
> plot(P45i, col = clr, xlim = c(0,20))
> plot(P45c, col = clr, xlim = c(20,0))
> plot(P50i, col = clr, xlim = c(0,20))
> plot(P50c, col = clr, xlim = c(20,0))
> plot(P55i, col = clr, xlim = c(0,20))
> plot(P55c, col = clr, xlim = c(20,0))
> plot(P60i, col = clr, xlim = c(0,20))
> plot(P60c, col = clr, xlim = c(20,0))
> plot(P65i, col = clr, xlim = c(0,20))
> plot(P65c, col = clr, xlim = c(20,0))
> mtext(c("Int","Conv"), side = 3, at = c(1,3)/4, outer = TRUE, line = 0)
> mtext(paste(seq(45,65,5)), side = 2, at = (5:1*2-1)/10,
+       outer = TRUE, line = 0)
```

e see that the curves are quite ragged; this is the simulation errors, it would be nicer if we simulated 1000 copies of each instead of only 100.

33. *Digression*: The previous is a lot of hard-coding, we would like to be able to easily get a plot with only a subset of the ages. To this end it is more convenient to collect the state probabilities in an array:

```
> (ain <- seq(45, 65, 5))
[1] 45 50 55 60 65
> (all <- levels(S4$allo))
[1] "Int"  "Conv"
> pdef <- NArray(c(list(ain = ain,
+                       allo = all),
+                  dimnames(P45i)))
> str(pdef)
 logi [1:5, 1:2, 1:201, 1:5] NA NA NA NA NA NA ...
 - attr(*, "dimnames")=List of 4
  ..$ ain  : chr [1:5] "45" "50" "55" "60" ...
  ..$ allo : chr [1:2] "Int" "Conv"
  ..$ when : chr [1:201] "0" "0.1" "0.2" "0.3" ...
  ..$ State: chr [1:5] "Norm" "Mic" "Mac" "D(CVD)" ...
```

But when we stick the results in an array we lose the `pState` class of the results: so we resort to the `mat2pol` function that stacks probabilities and plots them, so we simply take the result from `nState` and divide by the number in the initial state (`Mic`) using `sweep`:

```
> for(aa in ain)
+ for(gg in all)
+    pdef[paste(aa), gg, ,] <-
+    nState(subset(Sdef, ain == aa & allo == gg),
+           at = as.numeric(dimnames(pdef)[["when"]]),
+         from = 0,
```

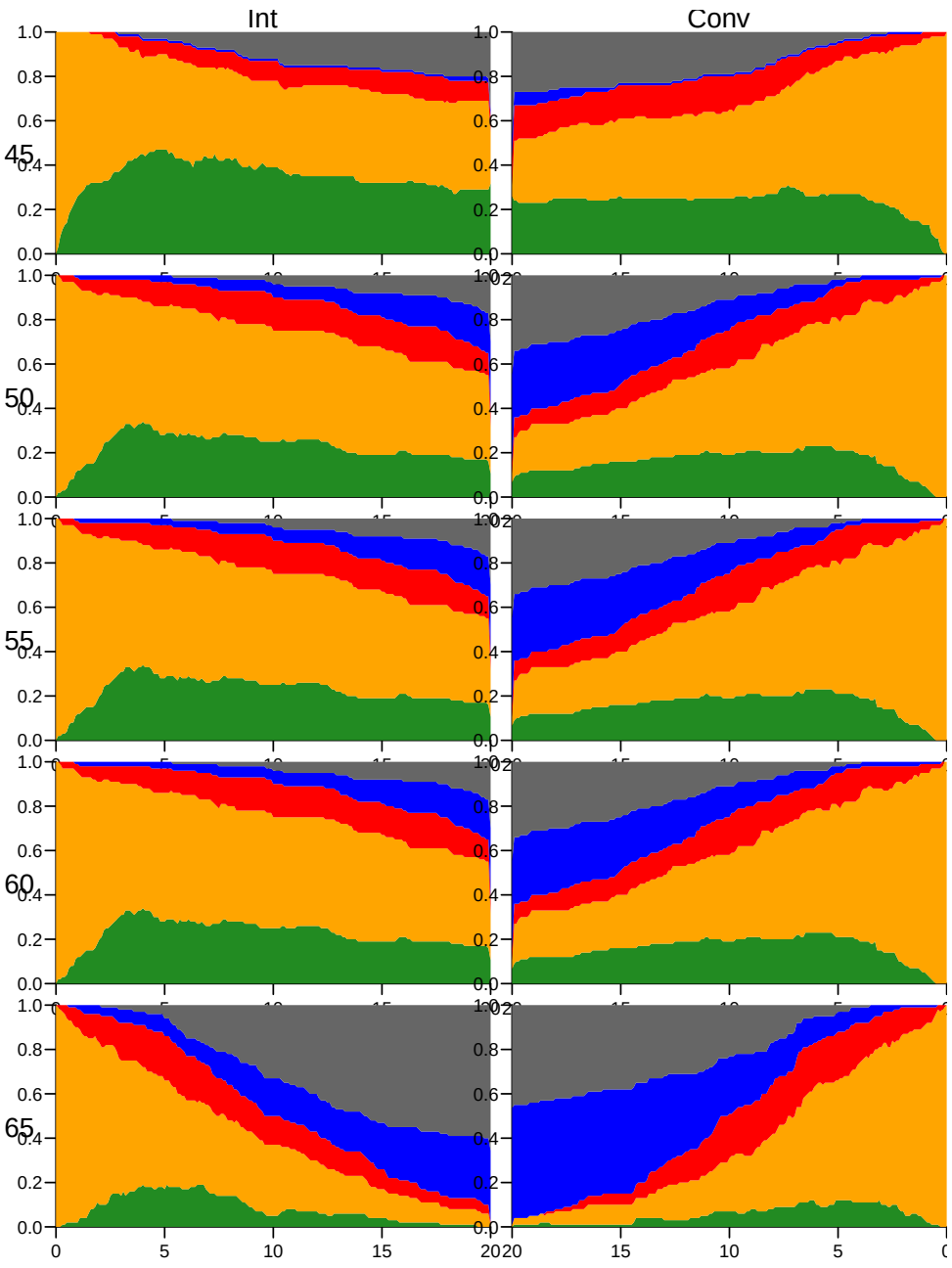


Figure 4.5: *Predicted probabilities of being in each of the states for persons aged 45, 50, 55, 60 and 65 at entry, separately for the two intervention groups. W* ../graph/ms-panel5

```
+   time.scale = "tfi")
> pdef <- sweep(pdef, 1:2, pdef[,,1,"Mic"], "/")
> str(pdef)
 num [1:5, 1:2, 1:201, 1:5] 0 0 0 0 0 0 0 0 0 0 ...
 - attr(*, "dimnames")=List of 4
  ..$ ain  : chr [1:5] "45" "50" "55" "60" ...
  ..$ allo : chr [1:2] "Int" "Conv"
```

```
  ..$ when : chr [1:201] "0" "0.1" "0.2" "0.3" ...
  ..$ State: chr [1:5] "Norm" "Mic" "Mac" "D(CVD)" ...
```

Then we have the state probabilities in the array pdef

```
> ain <- seq(45, 65, 10)
> par(mfrow = c(length(ain),2),
+       mar = c(3,3,1,1),
+       oma = c(0,2,1,0),
+       mgp = c(3,1,0) / 1.6)
> for(aa in ain)
+    {
+ mat2pol(pdef[paste(aa),"Int" ,,], col = clr, xlim = c(0,20))
+ mat2pol(pdef[paste(aa),"Conv",,], col = clr, xlim = c(20,0))
+    }
> mtext(c("Int","Conv"), side = 3, at = c(1,3)/4, outer = TRUE, line = 0)
> mtext(ain, side = 2, at = (length(ain):1 * 2 - 1) / (length(ain) * 2),
+       outer = TRUE, line = 0)
```

4.4 State probabilities using the Aalen-Johansen approach from survival

The survival package allows estimation of state probabilities by the Aalen-Johansen estimator similar to what we did in competing risks.

As mentioned under competing risks, the results will refer to a population of the same structure as the study population, and so the absolute sizes of the state probabilities will not be generalizable to other populations. The results here correspond to the results we derived using the original Steno2 population cohort in section 4.3.2 on page 62 ff.

The estimates of state probabilities in section 4.3.2 are based on parametric models for the transition probabilities, where some of the transition rates depend on age and duration in the same way. The estimates from the Aalen-Johansen approach is non-parametric in the sense that the transition rates can have any shape; the down side is that they cannot depend on more than one time scale (sensibly time since entry) and the shape and size of them are not easily retrievable.

34. A direct application gives the wrong result—transitions are wrong:

```
> AaJ <- survfit(Surv(tfi, tfi + lex.dur, lex.Xst) ~ 1,
+                id = lex.id,
+              data = L4)
> AaJ$transitions
        to
from     Norm Mac D(oth) D(CVD) (censored)
  (s0)     63  55     19      9         14
  Norm     96   5     17     10         16
  Mac       3  46     19     19          6
  D(oth)    0   0      0      0          0
  D(CVD)    0   0      0      0          0
> summary(L4)
Transitions:
     To
From   Mic Norm Mac D(oth) D(CVD)  Records:  Events: Risk time:  Persons:
  Mic  312   72  65     30     14       493      181    1437.39       160
```

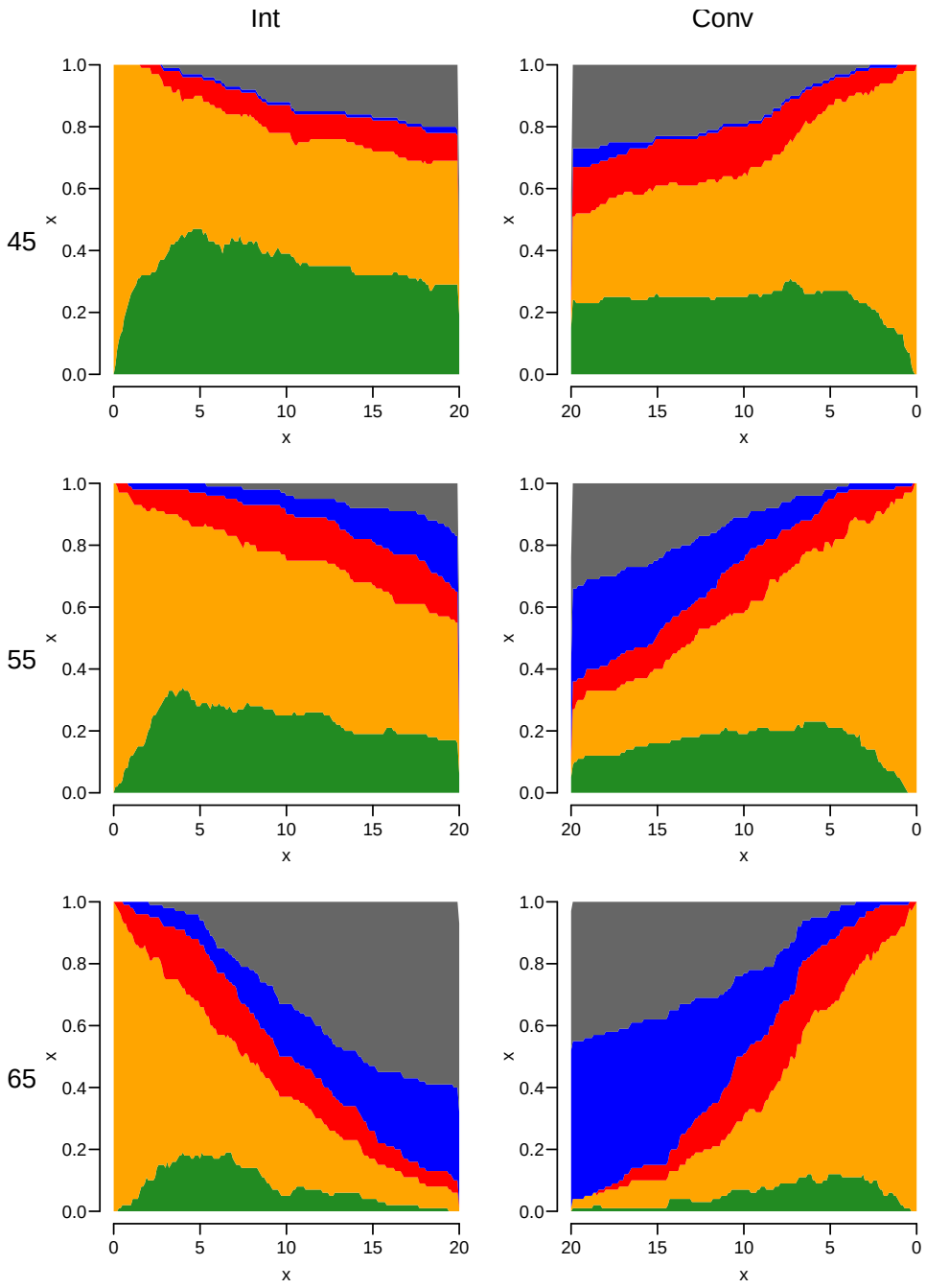


Figure 4.6: *Predicted probabilities of being in each of the states for persons aged 45, 55 and 65 at entry, separately for the two intervention groups.*

`../graph/ms-panel3`

```
Norm  35   90   0     13      6       144       54     581.83       66
Mac   22    0  41     12     18        93       52     401.70       60
Sum  369  162 106     55     38       730      287    2420.91      160
```

Comparing with the summary of `L4` we see that we get the number of transitions wrong; me must use the `istate` argument:

```
> survfit(Surv(tfi, tfi + lex.dur, lex.Xst) ~ 1,
+         id = lex.id,
+     istate = lex.Cst,
+       data = L4)
```

. . . but this will crash. This is because the machinery does not allow records with null transitions, that is records that is just a transition from a given state to the same if it is the *last* record for a person (i.e. censorings in the last state).

35. We therefore must rename these levels of `lex.Xst` to, say, `cens` (for `cens`ored, but any name will do), and this state must then be the `first` level of `lex.Xst`:

```
> R4 <- sortLexis(L4)
> last <- rev(!duplicated(rev(R4$lex.id)))
> R4$lex.Xst <- ifelse(last & R4$lex.Cst == R4$lex.Xst,
+                      "cens",
+                      as.character(R4$lex.Xst))
> R4 <- Relevel(factorize(R4), "cens")
NOTE: lex.Cst and lex.Xst now have levels:
 Mic Norm Mac cens D(CVD) D(oth)
> summary(L4)
Transitions:
     To
From   Mic Norm Mac D(oth) D(CVD)  Records:  Events: Risk time:  Persons:
  Mic  312   72  65     30     14       493      181    1437.39       160
  Norm  35   90   0     13      6       144       54     581.83        66
  Mac   22    0  41     12     18        93       52     401.70        60
  Sum  369  162 106     55     38       730      287    2420.91       160
> summary(R4)
Transitions:
     To
From   cens Mic Norm Mac D(CVD) D(oth)  Records:  Events: Risk time:  Persons:
  Mic    36 276   72  65     14     30       493      217    1437.39       160
  Norm   18  35   72   0      6     13       144       72     581.83        66
  Mac    13  22    0  28     18     12        93       65     401.70        60
  Sum    67 333  144  93     38     55       730      354    2420.91       160
```

Describe how the two Lexis objects are related.

36. As mentioned, we must tell what state each record starts in, this is conveyed in the argument `istate` (initial `state`):

```
> AaJ <- survfit(Surv(tfi, tfi + lex.dur, lex.Xst) ~ 1,
+                id = lex.id,
+            istate = lex.Cst,
+              data = R4)
```

We see that we get the correct number of transitions when we merge the initial state `s(0)` with `Mic`:

```
> AaJ$transitions[,c(6,1:5)]
        to
from     (censored) Mic Norm Mac D(CVD) D(oth)
  Mic            36 276   72  65     14     30
  Norm           18  35   72   0      6     13
  Mac            13  22    0  28     18     12
  D(CVD)          0   0    0   0      0      0
  D(oth)          0   0    0   0      0      0
> summary(R4)
Transitions:
     To
```

```
From   cens Mic Norm Mac D(CVD) D(oth)  Records:  Events: Risk time:  Persons:
  Mic    36 276   72  65     14     30       493      217    1437.39       160
  Norm   18  35   72   0      6     13       144       72     581.83        66
  Mac    13  22    0  28     18     12        93       65     401.70        60
  Sum    67 333  144  93     38     55       730      354    2420.91       160
> summary(L4)
Transitions:
     To
From   Mic Norm Mac D(oth) D(CVD)  Records:  Events: Risk time:  Persons:
  Mic  312   72  65     30     14       493      181    1437.39       160
  Norm  35   90   0     13      6       144       54     581.83        66
  Mac   22    0  41     12     18        93       52     401.70        60
  Sum  369  162 106     55     38       730      287    2420.91       160
```

The predicted state probabilities are in the slot called `pstate`, and the confidence intervals in the corresponding slots `lower` and `upper`.

```
> names(AaJ)
 [1] "n"           "time"        "n.risk"      "n.event"     "n.censor"    "pstate"
 [7] "p0"          "cumhaz"      "std.err"     "sp0"         "logse"       "transition
[13] "lower"       "upper"       "conf.type"   "conf.int"    "states"      "type"
[19] "call"
> AaJ$states
[1] "Mic"    "Norm"   "Mac"    "D(CVD)" "D(oth)"
> head(AaJ$pstate)
        [,1]    [,2]    [,3] [,4] [,5]
[1,] 0.99375 0.00000 0.00625    0    0
[2,] 0.98750 0.00625 0.00625    0    0
[3,] 0.98750 0.00625 0.00625    0    0
[4,] 0.98125 0.01250 0.00625    0    0
[5,] 0.98125 0.01250 0.00625    0    0
[6,] 0.98125 0.01250 0.00625    0    0
> head(AaJ$lower)
          [,1]         [,2]         [,3] [,4] [,5]
[1,] 0.9816133           NA 0.0008858142   NA   NA
[2,] 0.9704340 0.0008858142 0.0008858142   NA   NA
[3,] 0.9704340 0.0008858142 0.0008858142   NA   NA
[4,] 0.9604561 0.0031535032 0.0008858142   NA   NA
[5,] 0.9604561 0.0031535032 0.0008858142   NA   NA
[6,] 0.9604561 0.0031535032 0.0008858142   NA   NA
> head(AaJ$upper)
     [,1]       [,2]       [,3] [,4] [,5]
[1,]    1         NA 0.04409785   NA   NA
[2,]    1 0.04409785 0.04409785   NA   NA
[3,]    1 0.04409785 0.04409785   NA   NA
[4,]    1 0.04954807 0.04409785   NA   NA
[5,]    1 0.04954807 0.04409785   NA   NA
[6,]    1 0.04954807 0.04409785   NA   NA
```

We can now show the Aalen-Johansen estimator of the state probabilities:

```
> mat2pol(AaJ$pstate, perm = c(2,1,3,5,4), x = AaJ$time,
+         col = clr)
> lines(AaJ$time, apply(AaJ$pstate[,1:3], 1, sum), lwd = 5)
```

37. But as above, we are interested in seeing the results from each of the allocation groups, so we do the calculation for each:

```
> AaJ <- survfit(Surv(tfi, tfi + lex.dur, lex.Xst) ~ allo,
+                id = lex.id,
+            istate = lex.Cst,
```

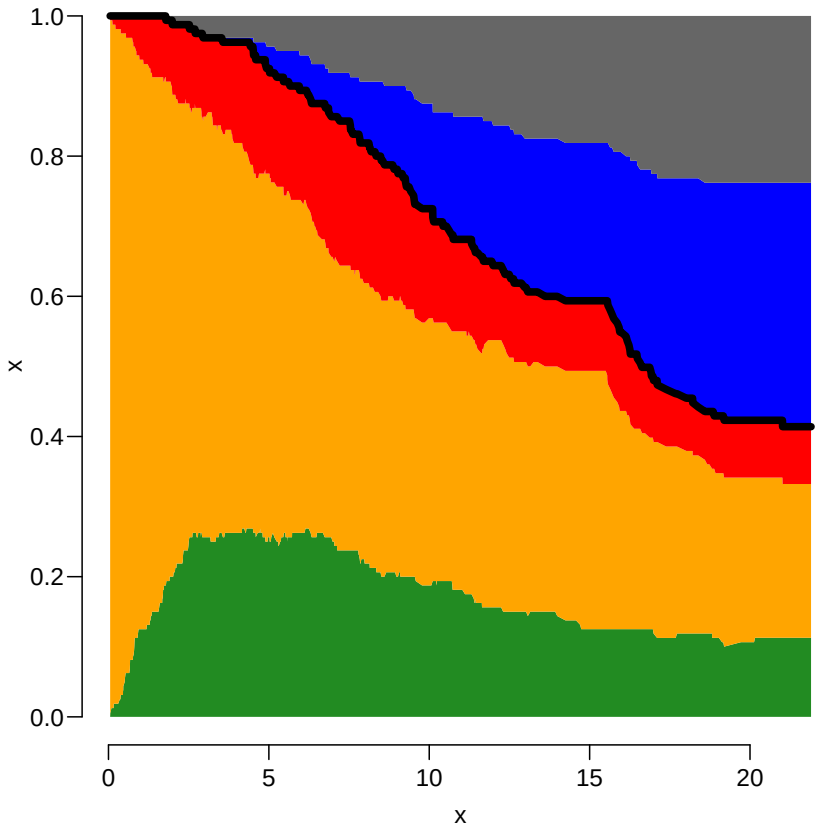


Figure 4.7: *Overall state probabilities from the Aalen-Johansen model*

../graph/ms-AaJ

```
+              data = R4)
> names(AaJ)
 [1] "n"           "time"        "n.risk"      "n.event"     "n.censor"    "pstate"
 [7] "p0"          "strata"      "std.err"     "sp0"         "logse"       "cumhaz"
[13] "transitions" "lower"       "upper"       "conf.type"   "conf.int"    "states"
[19] "type"        "call"
```

The result in the `AaJ` object is in a long vector of `time` and `pstate`, the two parts corresponding to `Int` and `Conv` put after one another, with the length of each part in `strata`.

```
> AaJ$states
[1] "Mic"    "Norm"   "Mac"    "D(CVD)" "D(oth)"
> AaJ$strata
 allo=Int allo=Conv
      375       337
> wh <- rep(substr(names(AaJ$strata), 6, 9), AaJ$strata)
> table(wh)
wh
Conv  Int
 337  375
```

So we just make the plots for the two subsets and place them next to each other as before:

```
> par(mfrow = c(1,2), mar=c(3,3,2,2))
> mat2pol(AaJ$pstate[wh=="Int",],
+         perm = c(2,1,3:5),
+         x = AaJ$time[wh=="Int"],
+         col = clr, xlim = c(0,21), xaxs = "i", yaxs = "i")
> lines(AaJ$time[wh=="Int"],
+       apply(AaJ$pstate[,1:3], 1, sum)[wh=="Int"], lwd = 4)
> mat2pol(AaJ$pstate[wh=="Conv",],
+         perm = c(2,1,3:5),
+         x = AaJ$time[wh=="Conv"],
+         col = clr, xlim = c(21,0), xaxs = "i", yaxs = "i")
> lines(AaJ$time[wh=="Conv"],
+       apply(AaJ$pstate[,1:3], 1, sum)[wh=="Conv"], lwd = 4)
> mtext(c("Int","Conv"), side = 3, at = c(1,3)/4, outer = TRUE, line = -2)
```

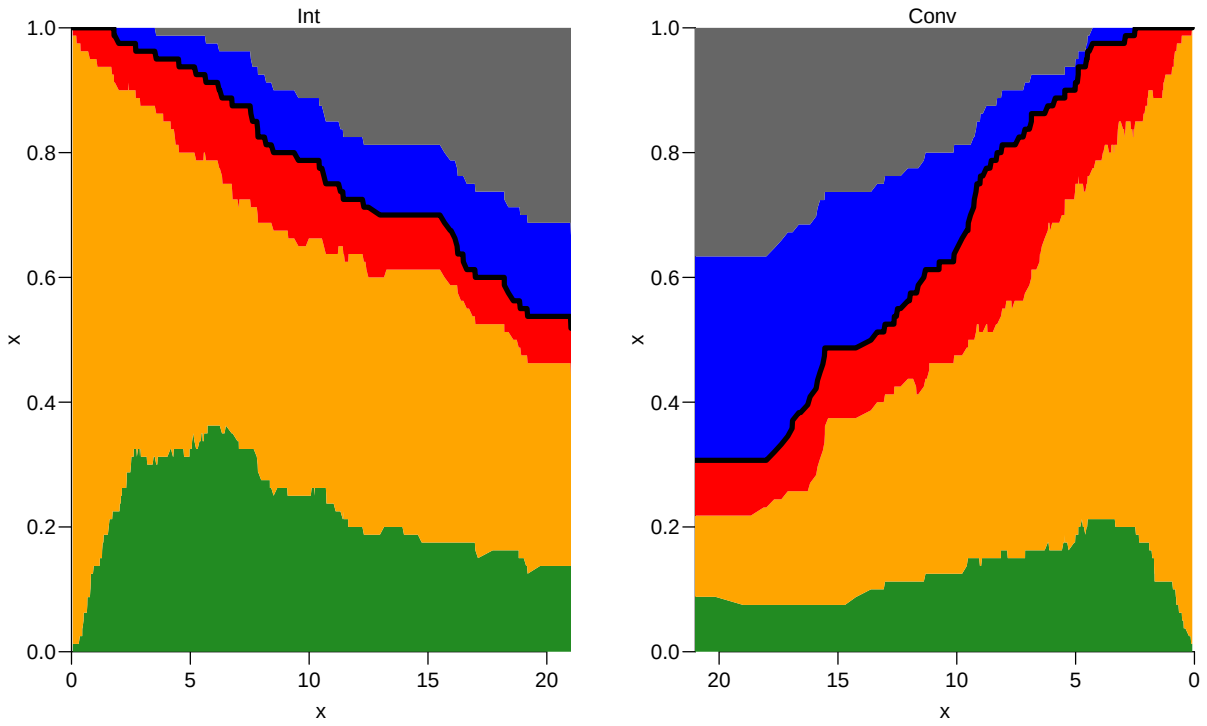


Figure 4.8: *Aalen-Johansen estimator of the state probabilities for the two intervention groups, for the original* total *Steno2 population, subdivided by intervention allocation.*

`../graph/ms-AaJstates`

38. This can be considered the empirical counterpart of figure 4.4; the state probabilities for a population as the one in the study. However not quite so; the models underlying figure 4.4 are proportional hazards in the sense that the effects of age and time since enrollment are proportional between state by allocation (6 groups for mortality, 4 groups for albuminuria state), whereas the figures in 4.8 are based on separate models for each transition and allocation.
39. We have confidence intervals for each of the state probabilities in the slots `lower` and `upper`, but not for the sums of these. And it is the sums of state probabilities we have

shown in the graph.
Moreover we would also want confidence intervals for areas under the curves. Neither are available from the Aalen-Johansen nor from the simulation approach. The simulation approach does not even give confidence intervals

4.5 Time spent in albuminuria states

Besides the state probabilities at different times after entry for groups of patients, we may also want to assess the time spent in each state, during, say, the first 15 or 20 years after entry.

40. We may want to compare groups by the expected time spent in the normoalbuminuric state during the first, say, 20 years. The expected time in a state is simply the time-integral of the probabilities, so we can easily compute it from `pdef`; each probability represents an interval of length 0.1, so we just take the midpoint of the probabilities at the ends of each interval.
Be careful when inspecting the results, it is not entirely obvious what apply does, keep track of the dimensions of each new table:

```
> mid <- function(x) x[-1] - diff(x) / 2
> pmid <- apply(pdef, c(1,2,4), mid)
> str(pmid)
 num [1:200, 1:5, 1:2, 1:5] 0.015 0.055 0.095 0.12 0.135 0.16 0.19 0.21 0.23 0.25 ...
 - attr(*, "dimnames")=List of 4
  ..$      : chr [1:200] "0.1" "0.2" "0.3" "0.4" ...
  ..$ ain  : chr [1:5] "45" "50" "55" "60" ...
  ..$ allo : chr [1:2] "Int" "Conv"
  ..$ State: chr [1:5] "Norm" "Mic" "Mac" "D(CVD)" ...
> pyr <- apply(pmid, 2:4, sum) * 0.1
> str(pyr)
 num [1:5, 1:2, 1:5] 7.03 5.43 4.53 3.87 1.61 ...
 - attr(*, "dimnames")=List of 3
  ..$ ain  : chr [1:5] "45" "50" "55" "60" ...
  ..$ allo : chr [1:2] "Int" "Conv"
  ..$ State: chr [1:5] "Norm" "Mic" "Mac" "D(CVD)" ...
> round(ftable(pyr, col.vars = 3:2), 1)
    State Norm       Mic       Mac      D(CVD)      D(oth)
    allo   Int Conv  Int Conv  Int Conv    Int Conv    Int Conv
ain
45         7.0  4.7  9.1  9.5  1.5  2.4    0.2  0.3    2.1  3.0
50         5.4  3.3  9.7  9.0  2.0  3.5    0.8  2.5    2.0  1.7
55         4.5  3.2 10.8  9.1  2.4  2.2    1.4  2.9    0.9  2.7
60         3.9  1.8  8.9  7.8  1.7  2.6    2.7  4.5    2.9  3.4
65         1.6  1.1  6.9  6.8  2.4  2.4    3.0  5.4    6.1  4.4
```

These numbers are the expected time (in years) spent in each state during the first 20 years after enrollment; we see that the intervention group spend far more time in `Norm` than do the conventional group, regardless of the age at entry.
The time spent in the two dead states are not really interpretable, it would be something like the number of years (during the first 20 years after enrollment) lost to each of the causes. We see that the most dramatic differences are for the CVD deaths. Look at the differences:

```
> round(pyr[,"Int",] - pyr[,"Conv",], 1)
    State
ain  Norm  Mic  Mac D(CVD) D(oth)
  45  2.3 -0.4 -0.9   -0.1   -0.9
  50  2.1  0.8 -1.5   -1.7    0.2
  55  1.3  1.7  0.2   -1.4   -1.7
  60  2.0  1.1 -0.9   -1.8   -0.5
  65  0.5  0.1 -0.1   -2.3    1.7
```

These are estimated times spent (sojourn times they are called) in each state. It is a bit strange to say that 55 year old enrollees in the intervention group spent 2.0 years less being dead from CVD than persons from the conventional group.

4.6 Clinical variables

So far we have only considered covariates that we know the value of at any time point, including future time points, that is the allocation status and timescales such as age and time since inclusion.

41. In the dataset st2clin are clinical measurements taken at different dates, up to six different occasions per person:

```
> data(st2clin)
>  str(st2clin)
'data.frame':        750 obs. of  5 variables:
 $ id  : num  1 2 3 4 5 6 7 8 9 10 ...
 $ doV : Date, format: "1993-05-07" "1993-05-05" ...
 $ a1c : num  87.3 66.5 73 61.2 102.7 ...
 $ chol: num  3.9 6.6 5.6 5.2 6 4.8 8.6 5.1 4.2 5.4 ...
 $ crea: num  83 83 68 97 149 55 56 78 123 79 ...
> st2clin <- rename(cal.yr(st2clin),
+                   lex.id = id,
+                      per = doV)
> summary(st2clin)
     lex.id            per            a1c              chol             crea
 Min.   :  1.00   Min.   :1993   Min.   : 32.80   Min.   : 2.200   Min.   :  28.00
 1st Qu.: 39.00   1st Qu.:1995   1st Qu.: 54.80   1st Qu.: 4.000   1st Qu.:  67.00
 Median : 84.50   Median :1997   Median : 66.35   Median : 4.800   Median :  88.00
 Mean   : 85.81   Mean   :2000   Mean   : 68.22   Mean   : 4.941   Mean   :  99.16
 3rd Qu.:131.00   3rd Qu.:2002   3rd Qu.: 79.38   3rd Qu.: 5.700   3rd Qu.: 115.25
 Max.   :176.00   Max.   :2015   Max.   :147.60   Max.   :14.000   Max.   :1067.00
                                 NA's   :4        NA's   :3        NA's   :2
> addmargins(table(table(st2clin$lex.id)))
  1   2   3   4   5   6 Sum
  2   6  23  38  31  60 160
```

Explain the contents of the table.

42. We can use `addCov.Lexis` to amend the follow-up data with the clinical measurements:

```
> S5 <- addCov.Lexis(S4, st2clin, "per")
> tt <- table(st2clin$lex.id)
> (who <- names(tt[tt == 3])[1])
[1] "5"
> subset(st2clin, lex.id == who)
    lex.id      per   a1c chol crea
5        5 1993.151 102.7  6.0  149
```

```
165      5 1995.511  54.7  8.8  140
321      5 1997.496  41.9  5.8  141
> nround(subset(S5,
+               lex.id == who,
+               select = c(lex.id,per,tfi,tfc,exnam,a1c,chol,crea)))
    lex.id     per  tfi  tfc exnam   a1c chol crea
159      5 1993.22 0.00 0.07   ex1 102.7  6.0  149
160      5 1993.72 0.50 0.57   ex1 102.7  6.0  149
161      5 1993.77 0.55 0.62   ex1 102.7  6.0  149
162      5 1994.22 1.00 1.07   ex1 102.7  6.0  149
163      5 1994.72 1.50 1.57   ex1 102.7  6.0  149
164      5 1995.22 2.00 2.07   ex1 102.7  6.0  149
165      5 1995.51 2.29 0.00   ex2  54.7  8.8  140
166      5 1995.72 2.50 0.21   ex2  54.7  8.8  140
167      5 1996.22 3.00 0.71   ex2  54.7  8.8  140
168      5 1996.72 3.50 1.21   ex2  54.7  8.8  140
169      5 1997.07 3.85 1.56   ex2  54.7  8.8  140
170      5 1997.22 4.00 1.71   ex2  54.7  8.8  140
171      5 1997.50 4.27 0.00   ex3  41.9  5.8  141
172      5 1997.72 4.50 0.23   ex3  41.9  5.8  141
> timeScales(S5)
[1] "per" "age" "tfi" "tfc"
> timeSince(S5)
per age tfi tfc
 ""  ""  ""  ""
```

We see that `tfc` is included as a time scale, but it is a not a proper time scale; it is reset to 0 at every clinical visit, and it also has some missing values, as do the clinical variables. The missing values are where there is follow-up before the earliest clinical measurement for a person.

But it needs to be a time scale in the `Lexis` object in order to be properly handled when subsequently cutting and splitting a `Lexis` object.

43. The values of the clinical measurements in `st2clin` are added to the follow-up data: extra cut points at the measurement dates are added, and the values of the clinical variables are propagated as LOCF (Last Observation Carried Forward), so it is possible to model the effect of these clinical variables on transition rates—creatinine is traditionally modeled on a log-scale, here we use the base 2 logarithm.

```
> detc <- glm.Lexis(S5, ~ Ns(tfi, knots = seq( 0, 20, 5)) +
+                         Ns(age, knots = seq(50, 80, 10)) +
+                         lex.Cst / allo +
+                         a1c + chol + log2(crea),
+                   from = c("Norm","Mic"),
+                     to = c("Mic","Mac"))
stats::glm Poisson analysis of Lexis object S5 with log link:
Rates for transitions: Norm->Mic, Mic->Mac
> impc <- glm.Lexis(S5, ~ Ns(tfi, knots = seq( 0, 20, 5)) +
+                         Ns(age, knots = seq(50, 80, 10)) +
+                         lex.Cst / allo +
+                         a1c + chol + log2(crea),
+                     to = c("Norm","Mic"),
+                   from = c("Mic","Mac"))
stats::glm Poisson analysis of Lexis object S5 with log link:
Rates for transitions: Mic->Norm, Mac->Mic
> round(ci.exp(detc), 3)
```

```
                                  exp(Est.)  2.5%  97.5%
(Intercept)                           0.033 0.002  0.553
Ns(tfi, knots = seq(0, 20, 5))1       0.680 0.259  1.788
Ns(tfi, knots = seq(0, 20, 5))2       0.280 0.078  1.008
Ns(tfi, knots = seq(0, 20, 5))3       0.244 0.040  1.478
Ns(tfi, knots = seq(0, 20, 5))4       0.228 0.063  0.830
Ns(age, knots = seq(50, 80, 10))1     2.096 0.880  4.993
Ns(age, knots = seq(50, 80, 10))2     4.283 1.096 16.735
Ns(age, knots = seq(50, 80, 10))3     3.358 0.906 12.447
lex.CstNorm                           2.587 1.459  4.589
a1c                                   1.005 0.993  1.018
chol                                  1.090 0.910  1.307
log2(crea)                            0.866 0.583  1.285
lex.CstMic:alloConv                   1.702 0.977  2.964
lex.CstNorm:alloConv                  0.433 0.193  0.973
> round(ci.exp(impc), 3)
                                  exp(Est.)  2.5%  97.5%
(Intercept)                           1.085 0.061 19.162
Ns(tfi, knots = seq(0, 20, 5))1       0.247 0.076  0.804
Ns(tfi, knots = seq(0, 20, 5))2       0.059 0.009  0.386
Ns(tfi, knots = seq(0, 20, 5))3       0.041 0.007  0.248
Ns(tfi, knots = seq(0, 20, 5))4       0.190 0.036  1.001
Ns(age, knots = seq(50, 80, 10))1     0.838 0.288  2.442
Ns(age, knots = seq(50, 80, 10))2     0.363 0.071  1.848
Ns(age, knots = seq(50, 80, 10))3     0.592 0.071  4.927
lex.CstMac                            1.059 0.468  2.396
a1c                                   0.991 0.978  1.003
chol                                  0.963 0.803  1.155
log2(crea)                            0.872 0.580  1.313
lex.CstMic:alloConv                   0.598 0.359  0.996
lex.CstMac:alloConv                   1.523 0.610  3.799
> moc <- glm.Lexis(S5, ~ Ns(tfi, knots = seq( 0, 20, 5)) +
+                        Ns(age, knots = seq(50, 80, 10)) +
+                        lex.Cst / allo +
+                        a1c + chol + log2(crea),
+                    to = "D(oth)")
stats::glm Poisson analysis of Lexis object S5 with log link:
Rates for transitions: Mic->D(oth), Norm->D(oth), Mac->D(oth)
> mCc <- glm.Lexis(S5, ~ Ns(tfi, knots = seq( 0, 20, 5)) +
+                        Ns(age, knots = seq(50, 80, 10)) +
+                        lex.Cst / allo +
+                        a1c + chol + log2(crea),
+                    to = "D(CVD)")
stats::glm Poisson analysis of Lexis object S5 with log link:
Rates for transitions: Mic->D(CVD), Norm->D(CVD), Mac->D(CVD)
> round(ci.exp(moc), 3)
                                  exp(Est.)  2.5%        97.5%
(Intercept)                           0.000 0.000 1.000000e-03
Ns(tfi, knots = seq(0, 20, 5))1     145.699 3.077 6.898293e+03
Ns(tfi, knots = seq(0, 20, 5))2      25.400 1.041 6.198740e+02
Ns(tfi, knots = seq(0, 20, 5))3   36604.424 5.048 2.654029e+08
Ns(tfi, knots = seq(0, 20, 5))4       1.751 0.286 1.074000e+01
Ns(age, knots = seq(50, 80, 10))1     2.115 0.676 6.610000e+00
Ns(age, knots = seq(50, 80, 10))2     1.119 0.108 1.161500e+01
Ns(age, knots = seq(50, 80, 10))3     8.945 2.938 2.723000e+01
lex.CstNorm                           1.033 0.384 2.778000e+00
```

```
lex.CstMac                                1.341 0.504 3.567000e+00
a1c                                       1.005 0.987 1.024000e+00
chol                                      0.845 0.635 1.124000e+00
log2(crea)                                1.849 1.140 3.002000e+00
lex.CstMic:alloConv                       1.936 0.875 4.288000e+00
lex.CstNorm:alloConv                      1.887 0.602 5.920000e+00
lex.CstMac:alloConv                       0.771 0.233 2.553000e+00
> round(ci.exp(mCc), 3)
                                      exp(Est.)  2.5%      97.5%
(Intercept)                               0.000 0.000      0.011
Ns(tfi, knots = seq(0, 20, 5))1           0.873 0.131      5.824
Ns(tfi, knots = seq(0, 20, 5))2           1.883 0.275     12.886
Ns(tfi, knots = seq(0, 20, 5))3           0.802 0.011     58.470
Ns(tfi, knots = seq(0, 20, 5))4           0.108 0.012      1.000
Ns(age, knots = seq(50, 80, 10))1         6.213 0.975     39.600
Ns(age, knots = seq(50, 80, 10))2       525.886 1.826 151495.697
Ns(age, knots = seq(50, 80, 10))3        19.071 4.167     87.283
lex.CstNorm                               1.248 0.307      5.069
lex.CstMac                                1.416 0.343      5.853
a1c                                       0.999 0.980      1.019
chol                                      1.007 0.738      1.374
log2(crea)                                1.346 0.755      2.399
lex.CstMic:alloConv                       1.674 0.550      5.091
lex.CstNorm:alloConv                      1.384 0.269      7.115
lex.CstMac:alloConv                       5.068 1.386     18.529
```

Only `crea` has any effect; a doubling of creatinine is associated with a 1.85 times higher mortality rate from other (non-CVD) causes. Confidence interval is (1.14,3.00), so not terribly precisely determined.

There are limitations in using clinical measurements as time-dependent variables without a model for the clinical variables. In order to simulate events based on models for transition rates we must know all covariates at all times, so models with non-deterministicly varying are not usable. Timescales are time-varying covariate, but they vary `deterministicly`, so their value for each person will be known at any time of follow-up.

So the models with effects of clinical variables as presented here cannot be used for prediction of state probabilities—that would requires some kind of model for the clinical variables over time as well.

4.7 Several transitions from one state: `stack`

So far, we have only jointly modeled transitions that originated in *different* states, for example

`Mic` → `Mac` and `Norm` → `Mic`;

`Norm` → `D(CVD)`, `Mic` → `D(CVD)` and `Mac` → `D(CVD)`.

As long as the different rates modeled are originating in *different* states, the likelihood will have at most one contribution from each record in the `Lexis` follow-up data set.

But if we want to create a joint model for more than one rate originating in a given state we must repeat some of risk time in different contributions to the likelihood. This means

that the modeling cannot be based on (subsets of) a `Lexis` object, we must repeat some records. This is detailed in section on Competing Risks in the PMM (Practical Multistate Modeling, http://bendixcarstensen.com/MSbook.pdf, very preliminary).

This behaviour can be achieved by the `stack.Lexis` function:

```
> St4 <- stack(S4)
NOTE: lex.Cst and lex.Xst now have levels:
 Mic Norm Mac D(oth) D(CVD)
> c(nrow(S4), nrow(St4))
[1]  5495 19773
> table(S4$lex.Cst)
   Mic   Norm    Mac D(oth) D(CVD)
  3288   1308    899      0      0
> table(St4$lex.Tr, St4$lex.Cst)
               Mic Norm  Mac D(oth) D(CVD)
  Mac->D(CVD)    0    0  899      0      0
  Mac->D(oth)    0    0  899      0      0
  Mac->Mic       0    0  899      0      0
  Mic->D(CVD)  3288    0    0      0      0
  Mic->D(oth)  3288    0    0      0      0
  Mic->Mac     3288    0    0      0      0
  Mic->Norm    3288    0    0      0      0
  Norm->D(CVD)    0 1308    0      0      0
  Norm->D(oth)    0 1308    0      0      0
  Norm->Mic       0 1308    0      0      0
> ftable(St4$lex.Tr, St4$lex.Xst, St4$lex.Fail, col.vars = 2:3)
               Mic        Norm        Mac      D(oth)      D(CVD)
             FALSE TRUE FALSE TRUE FALSE TRUE  FALSE TRUE  FALSE TRUE

Mac->D(CVD)     22    0     0    0   847    0     12    0      0   18
Mac->D(oth)     22    0     0    0   847    0      0   12     18    0
Mac->Mic         0   22     0    0   847    0     12    0     18    0
Mic->D(CVD)   3107    0    72    0    65    0     30    0      0   14
Mic->D(oth)   3107    0    72    0    65    0      0   30     14    0
Mic->Mac      3107    0    72    0     0   65     30    0     14    0
Mic->Norm     3107    0     0   72    65    0     30    0     14    0
Norm->D(CVD)    35    0  1254    0     0    0     13    0      0    6
Norm->D(oth)    35    0  1254    0     0    0      0   13      6    0
Norm->Mic        0   35  1254    0     0    0     13    0      6    0
```

We see that the `lex.Fail` is only `TRUE` where `lex.Xst` is equal to the second part if the `lex.Tr`.

The two ways of representing the data for person 102 are quite different:

```
> nround(subset(S4 , lex.id == 102)[,1:8], 1)
     lex.id    per  age tfi lex.dur lex.Cst lex.Xst  id
3348    102 1993.5 58.3 0.0     0.5     Mic     Mic 102
3349    102 1994.0 58.8 0.5     0.5     Mic     Mic 102
3350    102 1994.5 59.3 1.0     0.5     Mic     Mic 102
3351    102 1995.0 59.8 1.5     0.3     Mic  D(CVD) 102
> nround(subset(St4, lex.id == 102)[,1:9], 1)
      lex.id    per  age tfi lex.dur lex.Cst lex.Xst      lex.Tr lex.Fail
3348     102 1993.5 58.3 0.0     0.5     Mic     Mic   Mic->Norm    FALSE
3349     102 1994.0 58.8 0.5     0.5     Mic     Mic   Mic->Norm    FALSE
3350     102 1994.5 59.3 1.0     0.5     Mic     Mic   Mic->Norm    FALSE
3351     102 1995.0 59.8 1.5     0.3     Mic  D(CVD)   Mic->Norm    FALSE
33481    102 1993.5 58.3 0.0     0.5     Mic     Mic    Mic->Mac    FALSE
33491    102 1994.0 58.8 0.5     0.5     Mic     Mic    Mic->Mac    FALSE
```

```
33501    102 1994.5 59.3 1.0     0.5     Mic     Mic    Mic->Mac    FALSE
33511    102 1995.0 59.8 1.5     0.3     Mic  D(CVD)    Mic->Mac    FALSE
33482    102 1993.5 58.3 0.0     0.5     Mic     Mic Mic->D(oth)    FALSE
33492    102 1994.0 58.8 0.5     0.5     Mic     Mic Mic->D(oth)    FALSE
33502    102 1994.5 59.3 1.0     0.5     Mic     Mic Mic->D(oth)    FALSE
33512    102 1995.0 59.8 1.5     0.3     Mic  D(CVD) Mic->D(oth)    FALSE
33483    102 1993.5 58.3 0.0     0.5     Mic     Mic Mic->D(CVD)    FALSE
33493    102 1994.0 58.8 0.5     0.5     Mic     Mic Mic->D(CVD)    FALSE
33503    102 1994.5 59.3 1.0     0.5     Mic     Mic Mic->D(CVD)    FALSE
33513    102 1995.0 59.8 1.5     0.3     Mic  D(CVD) Mic->D(CVD)     TRUE
```

Suppose we wanted to fit a model for the two types of mortality assuming that, say, the effect of sex was the same.

Since some of the transitions we put in the same model originate from the same state we need the stacked data representation where each record corresponds to a likelihood term.

```
> cbind(with(subset(St4, grepl("D", lex.Tr)), table(lex.Tr)))
             [,1]
Mac->D(CVD)   899
Mac->D(oth)   899
Mac->Mic        0
Mic->D(CVD)  3288
Mic->D(oth)  3288
Mic->Mac        0
Mic->Norm       0
Norm->D(CVD) 1308
Norm->D(oth) 1308
Norm->Mic       0
```

We can then fit a model with common effect of

```
> stD <- glm(cbind(lex.Fail, lex.dur)
+          ~  Ns(tfi, knots = seq( 0, 20,  5)) * lex.Tr +
+             Ns(age, knots = seq(50, 80, 10)) * lex.Tr +
+             lex.Tr / allo + sex,
+         family = poisreg,
+         offset = log(lex.dur),
+           data = subset(St4, grepl("D", lex.Tr)))
> round(ci.exp(stD)[,1,drop=F],3)
                                                        exp(Est.)
(Intercept)                                          0.000000e+00
Ns(tfi, knots = seq(0, 20, 5))1                      9.296000e+00
Ns(tfi, knots = seq(0, 20, 5))2                      1.359700e+01
Ns(tfi, knots = seq(0, 20, 5))3                      7.635000e+00
Ns(tfi, knots = seq(0, 20, 5))4                      3.810000e-01
lex.TrMac->D(oth)                                    0.000000e+00
lex.TrMic->D(CVD)                                    3.551000e+00
lex.TrMic->D(oth)                                    1.600000e-02
lex.TrNorm->D(CVD)                                   0.000000e+00
lex.TrNorm->D(oth)                                   6.259000e+00
Ns(age, knots = seq(50, 80, 10))1                    7.196000e+00
Ns(age, knots = seq(50, 80, 10))2                    2.550790e+02
Ns(age, knots = seq(50, 80, 10))3                    7.351400e+01
sexM                                                 1.457000e+00
Ns(tfi, knots = seq(0, 20, 5))1:lex.TrMac->D(oth)    6.009169e+67
Ns(tfi, knots = seq(0, 20, 5))2:lex.TrMac->D(oth)    1.973003e+48
Ns(tfi, knots = seq(0, 20, 5))3:lex.TrMac->D(oth)   9.857399e+132
Ns(tfi, knots = seq(0, 20, 5))4:lex.TrMac->D(oth)    1.761155e+28
```

```
Ns(tfi, knots = seq(0, 20, 5))1:lex.TrMic->D(CVD)     9.000000e-03
Ns(tfi, knots = seq(0, 20, 5))2:lex.TrMic->D(CVD)     1.370000e-01
Ns(tfi, knots = seq(0, 20, 5))3:lex.TrMic->D(CVD)     1.930000e-01
Ns(tfi, knots = seq(0, 20, 5))4:lex.TrMic->D(CVD)     2.460000e-01
Ns(tfi, knots = seq(0, 20, 5))1:lex.TrMic->D(oth)     8.986370e+02
Ns(tfi, knots = seq(0, 20, 5))2:lex.TrMic->D(oth)     5.157600e+01
Ns(tfi, knots = seq(0, 20, 5))3:lex.TrMic->D(oth)     1.227135e+07
Ns(tfi, knots = seq(0, 20, 5))4:lex.TrMic->D(oth)     2.857700e+01
Ns(tfi, knots = seq(0, 20, 5))1:lex.TrNorm->D(CVD)    1.889027e+04
Ns(tfi, knots = seq(0, 20, 5))2:lex.TrNorm->D(CVD)    9.037953e+04
Ns(tfi, knots = seq(0, 20, 5))3:lex.TrNorm->D(CVD)    2.612224e+10
Ns(tfi, knots = seq(0, 20, 5))4:lex.TrNorm->D(CVD)    0.000000e+00
Ns(tfi, knots = seq(0, 20, 5))1:lex.TrNorm->D(oth)    4.070000e-01
Ns(tfi, knots = seq(0, 20, 5))2:lex.TrNorm->D(oth)    3.390000e-01
Ns(tfi, knots = seq(0, 20, 5))3:lex.TrNorm->D(oth)    6.371300e+01
Ns(tfi, knots = seq(0, 20, 5))4:lex.TrNorm->D(oth)    3.720000e-01
lex.TrMac->D(oth):Ns(age, knots = seq(50, 80, 10))1   2.070000e-01
lex.TrMic->D(CVD):Ns(age, knots = seq(50, 80, 10))1   1.894000e+00
lex.TrMic->D(oth):Ns(age, knots = seq(50, 80, 10))1   2.670000e-01
lex.TrNorm->D(CVD):Ns(age, knots = seq(50, 80, 10))1  2.220000e+00
lex.TrNorm->D(oth):Ns(age, knots = seq(50, 80, 10))1  1.094000e+00
lex.TrMac->D(oth):Ns(age, knots = seq(50, 80, 10))2   8.000000e-03
lex.TrMic->D(CVD):Ns(age, knots = seq(50, 80, 10))2   5.857000e+00
lex.TrMic->D(oth):Ns(age, knots = seq(50, 80, 10))2   5.000000e-03
lex.TrNorm->D(CVD):Ns(age, knots = seq(50, 80, 10))2  0.000000e+00
lex.TrNorm->D(oth):Ns(age, knots = seq(50, 80, 10))2  1.400000e-02
lex.TrMac->D(oth):Ns(age, knots = seq(50, 80, 10))3   1.340000e-01
lex.TrMic->D(CVD):Ns(age, knots = seq(50, 80, 10))3   1.910000e-01
lex.TrMic->D(oth):Ns(age, knots = seq(50, 80, 10))3   1.600000e-01
lex.TrNorm->D(CVD):Ns(age, knots = seq(50, 80, 10))3  0.000000e+00
lex.TrNorm->D(oth):Ns(age, knots = seq(50, 80, 10))3  2.680000e-01
lex.TrMac->D(CVD):alloConv                            9.182000e+00
lex.TrMac->D(oth):alloConv                            6.290000e-01
lex.TrMic->D(CVD):alloConv                            1.699000e+00
lex.TrMic->D(oth):alloConv                            2.125000e+00
lex.TrNorm->D(CVD):alloConv                           2.063000e+00
lex.TrNorm->D(oth):alloConv                           1.788000e+00
```

So under the assumption that the sex-effects are the same for all 6 mortality rates in figure 4.2 the M/W rate ratio is 1.46.

But it is only rarely that we want to model different rates out of the same state, so the actual use of `stack(.Lexis)` is seldom needed.

You should be aware that when using the `mstate` package, follow-up is stored as stacked objects, and so that

Chapter 5

Statistics Greenland

Statbank Greenland runs on a software package primarily developed by Statistics Sweden over the past 30 years in cooperation with about 30 national statistical institutes. Statistics Greenland has been participating in this work almost for the full period.

Today there are many ways to get data from the Statbank to local software via api(application programming interface) or more directly by a feature in the software, known as 'saved queries'.

5.0.1 api light - saved queries

'saved queries' are stored on the Statbank host-server and referred to with an identifyer string. When a table has been selected and further manipulated (like pivot, aggregations or other) one can save and distribute the 'saved query' combined with specified update-options for the time dimension.

When the Statbank is 'called'
https://bank.stat.gl:443/sq/< query-id >
a file is returned reflecting the table selection and subsequent workflow.
The query id can have options attached for specifying fileformat, action and/or language
<query-id.fileformat?action1&action2>
Valid fileformats are: .px - as PX-file
.xlsx - as Excel-file
.xlsx_doublecolumn - Excel-file with double column
.csv - default csv-file
.csv_tab - tabseparated csv-file without heading
.csv_tabhead - tabseparated csv-file with heading
.csv_comma - commaseparated csv-file without heading
.csv_commahead - commaseparated csv-file with heading
.csv_space - spaceseparated csv-file without heading
.csv_spacehead - spaceseparated csv-file with heading
.csv_semicolon - semicolonseparated csv-file without heading
.csv_semicolonhead - semicolonseparated csv-file with heading
.json_stat - json-stat-file

.json_stat2 - json-stat 2-file
.html5_table – HTML5 table
.relational_table – relational table (txt)
<query-id?action>
Valid actions are: select, lang
if select is specified instead of returning a file, the Statbank selection screen is shown, with the selected values high-lighted.
Valid languages are specific to each Statbank, in Greenland we use:
en (English),
kl (Greenlandic) or
da (Danish)
If more than one action is required the are separated by &
Example:
https://bank.stat.gl:443/sq/< query-id >?select&lang=kl
So data can be specified with any pxweb-statbank. To read data to R, the table can be imported to your r-script with:
as a 2-dimentional Dataframe
sq_data_csv <- read_csv(
"https://bank.stat.gl:443/sq/8fb0941c-3579-4848-a488-6a9afe4266ff.csv"
locale = locale(encoding = "latin1"))
as Dataframe with variables: sq_data_rel <- read_delim(
"https://bank.stat.gl:443/sq/< query-id >.relational_table",
locale = locale(encoding = "latin1"), delim = """)
Out-of-the-box the Pxweb software offers information on selected variables/values in a saved query by adding '?select'. But no information on added editing. Also if the metadata in the table, the saved query is based upon, has changed, Pxweb often reports error, with no help offered.
Query-id example:
https://bank.stat.gl:443/sq/8fb0941c-3579-4848-a488-6a9afe4266ff
With error:
https://bank.stat.gl:443/sq/8fb0941c-3579-4848-a488-6a9afe42lars
StatBank Greenland ONLY
StatGreenland has added a simple 'sqget'-asp function to bank.stat.gl software, that allows one to get information on any existing saved query:
https://bank.stat.gl/sqget.asp?8fb0941c-3579-4848-a488-6a9afe42lars

5.0.2 for more control

For more control and deeper integration, Pxweb-based statbanks offers a standard api to be consumed by many machine-languages. For this paper we focus on integration with R:
By March 2022 there are two free R resources on Cran to read pxweb-based Statbanks via api. ('pxR' reads local pcaxis-files only. Denmark and Ireland does not use pxweb-out-of-the-box)
https://cran.r-project.org (package repository):

pxweb
PxWebApiData (SSB.no)
csodata (only CSO.ie)
Github:
statgl (stat.gl)
dkstat (only DST.dk)

```
#=====================================================================
# Packages used below
# might need to be installed
#=====================================================================

library(tidyverse)

## Install or update packages from cran:
# install.packages("pxweb")
# install.packages("PxWebApiData")
# install.packages("csodata")

## Install or update packages from GitHub:
# if(!require("devtools")) install.packages("devtools")
# library("devtools")
#
# install_github("rOpenGov/dkstat")
# devtools::install_github("StatisticsGreenland/statgl")

#== end ==============================================================
```

5.0.3 Example 1: pxweb (cran)

Magnusson Måns, Kainu M, Huovari J, Lahti L (2019).

"pxweb: R tools for PX-WEB API." General interface to all pxweb based Statbanks.
Last updated 2021-10-09
Highlight:
Use pxweb_interactive to find relevant table(s) from one of 28 Statbanks and have ready to run r-script generated
In example 1 the pxweb package is used to get data from bank.stat.gl by
data_df_pxweb <- pxweb_get_data(url, query, variable.value.type = "code")
variable.value.type can be code or text
code are the same for all languages, text is dependent on the language code found in the url
" * " is short for all values in a variable. So if a variable has:
c("T","N","S") values, instead ' * ' return all

```
#=====================================================================
# Example 1: pxweb (cran)
# Magnusson M, Kainu M, Huovari J, Lahti L (2019).
# ■pxweb: R tools for PX-WEB API.■
#=====================================================================

library(pxweb)

#pxweb_interactive()

px_data <- pxweb_get_data(url =
"https://bank.stat.gl:443/api/v1/en/Greenland/BE/BE80/BEXCALC.PX",
           query =   list("year of birth" = "*",
                          gender = c("M", "K"),
                          "triangles(lexis)" = "*",
                          event = "*",
                          time = "*"),
           variable.value.type = "code")
```

Example 2: PxWebApiData (cran)

Statistics Norway, Øyvind Langsrud <oyl at ssb.no>

General interface to all pxweb based Statbanks.
Last updated 2021-10-11
In example 2 the PxWebApiData package is used to get data from statbank.hagstova.fo

```
#=====================================================================
# Example 2: PxWebApiData (cran)
# Statistics Norway, Øyvind Langsrud <oyl at ssb.no>
#=====================================================================

library(PxWebApiData)

meta <- ApiData(
  "https://statbank.hagstova.fo:443/api/v1/en/H2/DEV/COH/Lexis.px",
        returnMetaFrames = TRUE)

names(meta)
```

```
## [1] "year of birth"    "event"            "sex"              "Triangles(Lexis)"
## [5] "year"
```

```
meta[[2]]$values
```

```
## [1] "P" "B" "I" "O" "D" "C" "U"
```

```
data <- PxWebApiData::ApiData(
  "https://statbank.hagstova.fo:443/api/v1/en/H2/DEV/COH/Lexis.px",
        "year of birth" = TRUE,
        sex = c("M", "F"),
        "Triangles(Lexis)" = c("0", "1"),
        event = TRUE,
        year = TRUE # top3 : 3i instead of TRUE
)

# Extract the first list element, which contains full variable names.
data_df_PxWebApiData <- data[[1]]

head(data_df_PxWebApiData,5)
```

```
##   year of birth                       event   sex Triangles(Lexis) year value
## 1          1885 Population (start of year) Males            Upper 1985     1
## 2          1885 Population (start of year) Males            Upper 1986     1
## 3          1885 Population (start of year) Males            Upper 1987     0
## 4          1885 Population (start of year) Males            Upper 1988     0
## 5          1885 Population (start of year) Males            Upper 1989     0
```

Example 3: statgl (GitHub)

Statistics Greenland - https://github.com/StatisticsGreenland/statgl

General interface to all pxweb based Statbanks.

Last updated 2021-01-04

the statgl-package bundles pxweb-based statbank functionality with presentation features, used by Statistics Greenland on Sustainable Development Goals

In example 3 the statgl package is used to get data from statbank.hagstova.fo and also 2 Greenlandic examples to show additional features

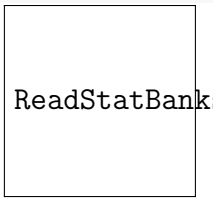
```
#=====================================================================
# Example 3: statgl (GitHub)
# Statistics Greenland - https://github.com/StatisticsGreenland/statgl
#=====================================================================

library(statgl)

#statgl_search("Population")

#statgl_search("Education", lang = "en", api_url = "https://statbank.hagstova.fo:44


data_df_statgl <- statgl_fetch(
```

area	gender	2016	2017	2018	2019	2020	2021	2022
c. Nuuk City	Total	17.316	17.600	17.796	17.984	18.326	18.800	19.261
c. Nuuk City	Female	8.183	8.334	8.437	8.533	8.703	8.903	9.131
c. Nuuk City	Male	9.133	9.266	9.359	9.451	9.623	9.897	10.130

```
  "https://statbank.hagstova.fo:443/api/v1/en/H2/DEV/COH/Lexis.px",
         "year of birth" = px_all(),
         sex = c("M", "F"),
         "Triangles(Lexis)" = c("0", "1"),
         event = px_all(),
         year = px_all(), # px_top(3)
         .val_code=TRUE)

CONST_statbank <- "https://bank.stat.gl/api/v1/en/Greenland"

statgl_url("BEXCALCR", api_url = CONST_statbank) %>%
  statgl_fetch(area = c("NUK"),
    event            = c("P"),
    gender           = px_all(),
    time             = px_top(7),
    .eliminate_rest    = TRUE ,
    .col_code          = TRUE,
    .val_code          = FALSE
  ) %>%
   select(-event) %>%
   pivot_wider(names_from = time,values_from = value) %>%
   statgl_table()
```

```
# https://stat.gl/en/sdg

  library(lubridate)

# Import
INXIU101_raw <-
  statgl_url("INXIU101", lang = "en") %>%
  statgl_fetch(
    indicator = 2:4,
    time      = px_all(),
    .col_code = TRUE
    ) %>%
  as_tibble()

# Transform
```

```
INXIU101 <-
  INXIU101_raw %>%
  mutate(
    time = time %>% make_date(),
    indicator = indicator %>% as.factor() %>% fct_rev()
    )

# Plot
INXIU101 %>%
  ggplot(aes(
    x    = time,
    y    = value,
    fill = indicator
    )) +
  geom_area(position = "identity") +
  scale_y_continuous(labels  = scales::percent_format(
    scale        = 1,
    accuracy     = 1.1,
    big.mark     = ".",
    decimal.mark = ","
    )) +
  theme_statgl() +
  scale_fill_statgl(reverse = TRUE, guide = guide_legend(reverse = TRUE)) +
  labs(
    title   = "At-risk-of-povery rate",
    x       = " ",
    y       = " "
    )
```



ReadStatBanks_files/figure-latex/Example3-1.pdf

Example 4: dkstat

https://github.com/rOpenGov/dkstat

Statbank Denmark specific

```
#=============================================================================
# Example 4: dkstat
# https://github.com/rOpenGov/dkstat
#=============================================================================
```

```
library(dkstat)

dkstat::dst_search("Grønland", lang="da") %>% head(5)
```

```
##       id                                                  text  unit            u
## 14 BEF5G Personer født i Grønland og bosat i Danmark 1. januar Antal 2022-02-11T08
##    firstPeriod latestPeriod active                              variables
## 14        2008         2022   TRUE køn, alder, forældrenes fødested, tid
```

```
bef5g_meta <- dst_meta("bef5g", lang = "da")

bef5g_meta[[1]]
```

```
## $id
## [1] "BEF5G"
##
## $text
## [1] "Personer født i Grønland og bosat i Danmark 1. januar"
##
## $description
## [1] "Personer født i Grønland og bosat i Danmark 1. januar efter køn, alder, foræl
##
## $unit
## [1] "Antal"
##
## $updated
## [1] "2022-02-11T08:00:00"
##
## $footnote
## NULL
```

```
bef5g_meta[[2]]
```

```
##      id                 text elimination
## 1   KØN                  køn        TRUE
## 2 ALDER                alder        TRUE
## 3    FF forældrenes fødested        TRUE
## 4   Tid                  tid       FALSE
```

```
bef5g_meta[[3]]$FF
```

```
##        id                                                        text
## 1     BDK                              Begge forældre født i Danmark
## 2    BGRL                             Begge forældre født i Grønland
## 3    BUDL                             Begge forældre født i udlandet
## 4    BUOP                                    Begge forældre uoplyst
## 5   DKGRL  En forælder født i Danmark og en forælder født i Grønland
## 6   DKUDL  En forælder født i Danmark og en forælder født i udlandet
```

```
## 7   DKUOP          En forælder født i Danmark og en forælder uoplyst
## 8  GRLUDL En forælder født i Grønland og en forælder født i udlandet
## 9  GRLUOP         En forælder født i Grønland og en forælder uoplyst
## 10 UDLUOP         En forælder født i udlandet og en forælder uoplyst
```

```
data_dkstat <- dst_get_data(table = "bef5g",
                  KØN = "*",
                  ALDER  = "*",
                  FF = "*",
                  Tid = "*",
                  lang = "en",
                  meta_data = bef5g_meta,
                  value_presentation="value") %>%
  as_tibble()
```

Example 5: csodata (cran)

Conor Crowley <conor.crowley at cso.ie>

Statbank Ireland specific

```
#=============================================================================
# Example 5: csodata (cran)
# Conor Crowley <conor.crowley at cso.ie>
#=============================================================================
# library(csodata)
#
# toc <- cso_get_toc()
# head(toc)
#
# population <- cso_search_toc("Population")
#
# tbl1 <- cso_get_data("PEB07")
#
# meta1 <- cso_get_meta("PEA19") %>% as_tibble()
# cso_disp_meta("PEA19")

# data_df_cso <- statgl_fetch(url =
# "https://ws.cso.ie/public/api.restful/PxStat.Data.Cube_API.PxAPIv1/en/17/PME/PEA2
#           Year = px_all(),
#           sex = px_all(),
#           Nationality = px_all(),
#           .val_code=TRUE)
#
# data_df_cso <- pxweb_get_data(url =
```

```
# "https://ws.cso.ie/public/api.restful/PxStat.Data.Cube_API.PxAPIv1/en/17/PME/PEA2
#           query =   list(Year = "*",
#                     sex = "*",
#                     Nationality = "*"))
#
```

5.1 Example 6: pxR (cran)

5.2 Carlos J. Gil Bellosta <cgb at datanalytics.com>

Read PX-files to R

Last updated 2020-06-07

```
#======================================================================
# Example 6: pxR (cran)
# Carlos J. Gil Bellosta <cgb at datanalytics.com>
#======================================================================

library(pxR)

# Read px-files

# library(pxR)
#
# Reading PC-Axis files into R
# Function read.px reads a PC-Axis file from a given location and returns
# an object of class px containing all the data and metadata in the
# original PC-Axis file.
#
# The single most important piece of information within a pxobject is the
# data matrix, which can be extracted into a R data.frame using function
# as.data.frame. For instance,
#
# my.px.object <- read.px("/path/to/pc-axis/file")
# my.px.data   <-  as.data.frame(my.px.object)
# will create the data.frame my.px.data with the data in the corresponding
# PC-Axis file.
#
```

```
# copy and run next line to console to convert to Latex
# rmarkdown::render("ReadStatBanks.Rmd", output_format = "latex_document")
```