A short course in Epidemiology with R

NoveNordisk, Epidemiology
August 2015
http://BendixCarstensen.com/Epi/Courses/NNepi
Version 2.1

Compiled Monday 10th August, 2015, 13:03
from: `/home/bendix/teach/Epi/NNepi/pracs/pracs`

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

Contents

Programme of the course

This 2-day course is centered around practical calculations in R, illustrating the concepts in analysis of real datasets and a fairly broad range of useful R tools useful in daily practise. All sessions will be alternating between lectures and practicals, most followed by a walk-through of the computing issues.

The last afternoon will be a demonstration of a real analysis (on a restricted dataset, though), which students are intended to do concurrently on their own computer in order to get a good grip of the practicalities of the analysis.

Please note the details of the computing requirements on the course web-site, `http://bendixcarstensen.com/Epi/Courses/NNepi`, including download of datasets and programs for the practicals.

Tuesday 11 August 2015

09:00 – 10:00	**Lecture:** Introduction to R.
	Reading data, data structures.
	Lanuage and basic graphics.
10:00 – 10:30	**Coffee break**
10:30 – 12:00	**Practical:** Basic R, Reading data
	Simple simulation (optional), Tabulation, Basic graphics
12:00 – 13:00	**Lunch**
13:00 – 14:00	**Lecture:** Introduction to rates and survival.
	Computing rates, RR and RD.
	Fitting a smooth curve and showing it
14:00 – 14:15	**Afternoon Tea**
14:15 – 16:00	**Practical:** Calulation of rates, RR and RD
	Fitting a smooth curve and showing it
16:00 – 16:30	Summary of the day.

Friday 14 August 2015

09:00 – 09:30	**Recap:** Rates, smooth curves, simple graphs.
09:30 – 10:15	**Lecture:** Representation of follow-up data: Lexis objects.
	Cox model and Poisson model.
10:15 – 10:30	**Coffee**
10:30 – 12:00	**Practical:** Cox and Poisson modelling using Lexis representation.
12:00 – 13:00	**Lunch**
13:00 – 16:00	**Lecture/demo:**
	Diabetes in Denmark: Mortality of Danish Diabetes patients.
	Timescales and SMR analysis.
16:00 – 16:30	Summary of the day.

Chapter 1

Basic concepts in survival and demography

The following is a summary of relations between various quantities used in analysis of follow-up studies. They are ubiquitous in the analysis and reporting of results. Hence it is important to be familiar with all of them and the relation between them.

1.1 Probability

Survival function:

$$\begin{aligned} S(t) &= \mathrm{P}\left\{\text{survival at least till } t\right\} \\ &= \mathrm{P}\left\{T > t\right\} = 1 - \mathrm{P}\left\{T \leq t\right\} = 1 - F(t) \end{aligned}$$

Conditional survival function:

$$\begin{aligned} S(t|t_{\text{entry}}) &= \mathrm{P}\left\{\text{survival at least till } t|\text{ alive at } t_{\text{entry}}\right\} \\ &= S(t)/S(t_{\text{entry}}) \end{aligned}$$

Cumulative distribution function of death times (cumulative risk):

$$\begin{aligned} F(t) &= \mathrm{P}\left\{\text{death before } t\right\} \\ &= \mathrm{P}\left\{T \leq t\right\} = 1 - S(t) \end{aligned}$$

Density function of death times:

$$f(t) = \lim_{h\to 0} \mathrm{P}\left\{\text{death in } (t, t+h)\right\}/h = \lim_{h\to 0} \frac{F(t+h) - F(t)}{h} = F'(t)$$

Intensity:

$$\begin{aligned} \lambda(t) &= \lim_{h\to 0} \mathrm{P}\left\{\text{event in } (t, t+h] \mid \text{alive at } t\right\}/h \\ &= \lim_{h\to 0} \frac{F(t+h) - F(t)}{S(t)h} = \frac{f(t)}{S(t)} \\ &= \lim_{h\to 0} -\frac{S(t+h) - S(t)}{S(t)h} = -\frac{\mathrm{d}\log S(t)}{\mathrm{d}t} \end{aligned}$$

The intensity is also known as the hazard function, hazard rate, rate, mortality/morbidity rate.

Note that f and λ are *scaled* quantities, they have dimension time^{-1}.

Relationships between terms:

$$\begin{aligned} -\frac{\mathrm{d}\log S(t)}{\mathrm{d}t} &= \lambda(t) \\ &\Updownarrow \\ S(t) &= \exp\left(-\int_0^t \lambda(u)\,\mathrm{d}u\right) = \exp\bigl(-\Lambda(t)\bigr) \end{aligned}$$

The quantity $\Lambda(t) = \int_0^t \lambda(s)\,\mathrm{d}s$ is called the *integrated intensity* or the **cumulative rate**. It is *not* an intensity (rate), it is dimensionless.

$$\lambda(t) = -\frac{\mathrm{d}\log(S(t))}{\mathrm{d}t} = -\frac{S'(t)}{S(t)} = \frac{F'(t)}{1-F(t)} = \frac{f(t)}{S(t)}$$

The cumulative *risk* of an event (to time t) is:

$$F(t) = \mathrm{P}\{\text{Event before time } t\} = \int_0^t \lambda(u)S(u)\,\mathrm{d}u = 1 - S(t) = 1 - \mathrm{e}^{-\Lambda(t)}$$

For small $|x|$ (< 0.05), we have that $1 - \mathrm{e}^{-x} \approx x$, so for small values of the integrated intensity:

$$\text{Cumulative risk to time } t \approx \Lambda(t) = \text{Cumulative rate}$$

1.2 Statistics

Likelihood from one person:

The likelihood from a number of small pieces of follow-up from one individual is a product of conditional probabilities:

$$\begin{aligned} \mathrm{P}\{\text{event at } t_4|\text{entry at } t_0\} &= \mathrm{P}\{\text{survive } (t_0,t_1)|\text{ alive at } t_0\} \times \\ &\quad\ \mathrm{P}\{\text{survive } (t_1,t_2)|\text{ alive at } t_1\} \times \\ &\quad\ \mathrm{P}\{\text{survive } (t_2,t_3)|\text{ alive at } t_2\} \times \\ &\quad\ \mathrm{P}\{\text{event at } t_4|\text{ alive at } t_3\} \end{aligned}$$

Each term in this expression corresponds to one *empirical rate*[1] $(d, y) = (\#\text{deaths}, \#\text{risk time})$, i.e. the data obtained from the follow-up of one person in the interval of length y. Each person can contribute many empirical rates, most with $d = 0$; d can only be 1 for the *last* empirical rate for a person.

Log-likelihood for one empirical rate (d, y):

$$\ell(\lambda) = d\log(\lambda) - \lambda y$$

This is under the assumption that the underlying rate (λ) is constant over the interval that the empirical rate refers to.

[1]This is a concept coined by BxC, and so is not necessarily generally recognized.

Log-likelihood for several persons. Adding log-likelihoods from a group of persons (only contributions with identical rates) gives:

$$D\log(\lambda) - \lambda Y,$$

where Y is the total follow-up time, and D is the total number of failures.

Note: The Poisson log-likelihood for an observation D with mean λY is:

$$D\log(\lambda Y) - \lambda Y = D\log(\lambda) + D\log(Y) - \lambda Y$$

The term $D\log(Y)$ does not involve the parameter λ, so the likelihood for an observed rate can be maximized by pretending that the no. of cases D is Poisson with mean λY. But this does *not* imply that D follows a Poisson-distribution. It is entirely a likelihood based computational convenience. Anything that is not likelihood based is not justified.

A linear model for the log-rate, $\log(\lambda) = X\beta$ implies that

$$\lambda Y = \exp\bigl(\log(\lambda) + \log(Y)\bigr) = \exp\bigl(X\beta + \log(Y)\bigr)$$

Therefore, in order to get a linear model for $\log(\lambda)$ we must require that $\log(Y)$ appear as a variable in the model for $D \sim (\lambda Y)$ with the regression coefficient fixed to 1, a so-called *offset*-term in the linear predictor.

1.3 Competing risks

Competing risks: If there is more than one, say 3, causes of death, occurring with (cause-specific) rates λ_1, λ_2, λ_3, that is:

$$\lambda_c(a) = \lim_{h\rightarrow 0} \mathrm{P}\left\{\text{death from cause } c \text{ in } (a, a+h] \mid \text{alive at } a\right\}/h, \quad c = 1,2,3$$

The survival function is then:

$$S(a) = \exp\left(-\int_0^a \lambda_1(u) + \lambda_2(u) + \lambda_3(u)\,\mathrm{d}u\right)$$

because you have to escape all 3 causes of death. The probability of dying from cause 1 before age a (the cause-specific cumulative risk) is:

$$\mathrm{P}\left\{\text{dead from cause 1 at } a\right\} = \int_0^a \lambda_1(u)S(u)\,\mathrm{d}u \neq 1 - \exp\left(-\int_0^a \lambda_1(u)\,\mathrm{d}u\right)$$

The term $\exp(-\int_0^a \lambda_1(u)\,\mathrm{d}u)$ is sometimes referred to as the "cause-specific survival", but it does not have any probabilistic interpretation in the real world. It is the survival under the assumption that only cause 1 existed an that the mortality rate from this cause was the same as when the other causes were present too.

Together with the survival function, the cause-specific cumulative risks represent a classification of the population at any time in those alive and those dead from causes 1, 2 and 3 respectively:

$$1 = S(a) + \int_0^a \lambda_1(u)S(u)\,\mathrm{d}u + \int_0^a \lambda_2(u)S(u)\,\mathrm{d}u + \int_0^a \lambda_3(u)S(u)\,\mathrm{d}u, \quad \forall a$$

Subdistribution hazard Fine and Gray defined models for the so-called subdistribution hazard. Recall the relationship between between the hazard (λ) and the cumulative risk (F):

$$\lambda(a) = -\frac{\mathrm{d}\log\big(S(a)\big)}{\mathrm{d}a} = -\frac{\mathrm{d}\log\big(1-F(a)\big)}{\mathrm{d}a}$$

When more competing causes of death are present the Fine and Gray idea is to use this transformation to the cause-specific cumulative risk for cause 1, say:

$$\tilde{\lambda}_1(a) = -\frac{\mathrm{d}\log\big(1-F_1(a)\big)}{\mathrm{d}a}$$

This is what is called the subdistribution hazard, it depends on the survival function S, which depends on *all* the cause-specific hazards:

$$F_1(a) = \mathrm{P}\left\{\text{dead from cause 1 at } a\right\} = \int_0^a \lambda_1(u)S(u)\,\mathrm{d}u$$

The subdistribution hazard is merely a transformation of the cause-specific cumulative risks. Namely the same transformation which in the single-cause case transforms the cumulative risk to the hazard.

1.4 Demography

Expected residual lifetime: The expected lifetime (at birth) is simply the variable age (a) integrated with respect to the distribution of age at death:

$$\mathrm{EL} = \int_0^\infty af(a)\,\mathrm{d}a$$

where f is the density of the distribution of lifetimes.

The relation between the density f and the survival function S is $f(a) = -S'(a)$, and so integration by parts gives:

$$\mathrm{EL} = \int_0^\infty a\big(-S'(a)\big)\,\mathrm{d}a = -\Big[aS(a)\Big]_0^\infty + \int_0^\infty S(a)\,\mathrm{d}a$$

The first of the resulting terms is 0 because $S(a)$ is 0 at the upper limit and a by definition is 0 at the lower limit.

Hence the expected lifetime can be computed as the integral of the survival function.

The expected *residual* lifetime at age a is calculated as the integral of the *conditional* survival function for a person aged a:

$$\mathrm{EL}(a) = \int_a^\infty S(u)/S(a)\,\mathrm{d}u$$

Lifetime lost due to a disease is the difference between the expected residual lifetime for a diseased person and a non-diseased (well) person at the same age. So all that is needed is a(n estimate of the) survival function in each of the two groups.

$$\mathrm{LL}(a) = \int_a^\infty S_{\mathrm{Well}}(u)/S_{\mathrm{Well}}(a) - S_{\mathrm{Diseased}}(u)/S_{\mathrm{Diseased}}(a)\,\mathrm{d}u$$

Note that the definition of the survival function for a non-diseased person requires a decision as to whether one will consider non-diseased persons immune to the disease in question or not. That is whether we will include the possibility of a well person getting ill and subsequently die. This does not show up in the formulae, but is a decision required in order to devise an estimate of S_{Well}.

Lifetime lost by cause of death is using the fact that the difference between the survival probabilities is the same as the difference between the death probabilities. If several causes of death (3, say) are considered then:

$$\begin{aligned} S(a) = 1 &- \mathrm{P}\left\{\text{dead from cause 1 at } a\right\} \\ &- \mathrm{P}\left\{\text{dead from cause 2 at } a\right\} \\ &- \mathrm{P}\left\{\text{dead from cause 3 at } a\right\} \end{aligned}$$

and hence:

$$\begin{aligned} S_{\text{Well}}(a) - S_{\text{Diseased}}(a) &= \mathrm{P}\left\{\text{dead from cause 1 at } a|\text{Diseased}\right\} \\ &+ \mathrm{P}\left\{\text{dead from cause 2 at } a|\text{Diseased}\right\} \\ &+ \mathrm{P}\left\{\text{dead from cause 3 at } a|\text{Diseased}\right\} \\ &- \mathrm{P}\left\{\text{dead from cause 1 at } a|\text{Well}\right\} \\ &- \mathrm{P}\left\{\text{dead from cause 2 at } a|\text{Well}\right\} \\ &- \mathrm{P}\left\{\text{dead from cause 3 at } a|\text{Well}\right\} \end{aligned}$$

So we can conveniently define the lifetime lost due to cause 2, say, by:

$$\begin{aligned} \mathrm{LL}_2(a) = \int_a^\infty & \mathrm{P}\left\{\text{dead from cause 2 at } u|\text{Diseased \& alive at } a\right\} \\ & - \mathrm{P}\left\{\text{dead from cause 2 at } u|\text{Well \& alive at } a\right\} \,\mathrm{d}u \end{aligned}$$

These quantities have the property that their sum is the total years of life lost due to the disease:

$$\mathrm{LL}(a) = \mathrm{LL}_1(a) + \mathrm{LL}_2(a) + \mathrm{LL}_3(a)$$

The terms in the integral are computed as (see the section on competing risks):

$$\begin{aligned} \mathrm{P}\left\{\text{dead from cause 2 at } u|\text{Diseased \& alive at } a\right\} &= \int_a^u \lambda_{2,\text{Dis}}(x) S_{\text{Dis}}(x)/S_{\text{Dis}}(a)\,\mathrm{d}x \\ \mathrm{P}\left\{\text{dead from cause 2 at } u|\text{Well \& alive at } a\right\} &= \int_a^u \lambda_{2,\text{Well}}(x) S_{\text{Well}}(x)/S_{\text{Well}}(a)\,\mathrm{d}x \end{aligned}$$

Chapter 2

Exercises

2.1 Practice with basic R

The main purpose of this session is to give participants who have not had much (or any) experience with using R a chance to practice the basics and to ask questions.

R can be installed on all major platforms. We do not assume in this exercise that you are using any particular platform or graphical user interface (GUI). Some GUIs will allow you to perform certain operations using a menu rather than typing commands (e.g. stopping R or changing your working directory). If you are using a GUI, take the time to discover the facilities it offers.

2.1.1 The working directory

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

If you are using a GUI then you will typically need to change to the correct working directory after starting R. You can do this from the menu system (Look under the "File" menu). If you are working on the command line in a terminal, then you can change to the correct working directory and then launch R by typing "R". In either case you quit R by typing

```
> q()
```

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

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

2.1.2 Read-evaluate-print

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

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

2.1.3 Using R as a calculator

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

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

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

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

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

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

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

You can also use a right arrow, as in

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

Standard probability functions are readily available. For example, the probability below 1.96 in a standard normal (i.e. Gaussian) distribution is obtained with

```
> pnorm(1.96)
```

while

```
> pnorm(1.96, mean=0, sd=2)
```

will return the probability below 1.96 for a normal distribution with mean 0 and standard deviation 2. There is a number of different probability distributions implemented. For instance,

```
> pchisq(3.84, df=1)
```

will return the probability below 3.84 in a χ^2 distribution on 1 degree of freedom, and

```
> pchisq(3.84, df=1, lower.tail=FALSE)
```

will return the probability above 3.84. You can also find quantiles for given probabilities:

```
> qnorm(0.999)
> qnorm(0.999, mean=0, sd=2)
> qexp(0.999)
```

Exercise 1.

1. Find the probability above 4.3 in a chi-squared distribution on 1 degree of freedom.
2. Find the 75% quantile of the standard normal distribution

2.1.4 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 collections 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 or character). 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, 6, "name of company")
```

creates a list with 3 components. 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
> m
> m+1
```

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; thy

```
> str(v)
> str(m)
```

2.1.5 Sequences

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

```
> 1:10
```

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

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

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

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

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

Exercise 2.

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

2.1.6 Displaying and changing parts of a vector (indexing)

One reason for creating sequences of numbers is to use them to see select subsets of your data structures, for example, try to understand the following commands:

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

You will see that you can extract elements of a vector by supplying a vector of integer indices inside square brackets. You can extract a single element by giving a scalar index (e.g. `x[4]`) and extract a sub-vector by supplying any expression that creates an integer vector. Negative subscripts mean "drop this element". So `x[-1]` returns every element of `x` except the first. Also you note that the same index can appear more than once.

R also allows *logical subscripting*. Try the following

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

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

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

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

This replaces the first element of `x`. Logical subscripting is useful for replacing parts of a vector that satisfy a certain condition:

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

If you want to create an entirely new vector based on some logical condition then the `ifelse()` function

```
>  ifelse(round(x/2)==x/2,"even","odd")
```

Now try the following:

1. Display every third element in x
2. Display elements that are less than 10, but greater than 4
3. Modify the vector x, replacing by 10 all values that are greater than 10
4. Modify the vector x, multiplying by 2 all elements that are smaller than 5
5. Create a new vector y with elements 0,1,0,1, ... (12 elements) and a vector z that equals x when y=0 and 3x when y=1. (You can do it using `ifelse`, but there are other possibilities)

2.1.7 Data frames, indexing and subsets

Start with creating a simple data frame:

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

See what happens (and why):

```
>  mydata
>  names(mydata)
>  mydata[,"age"]
>  mydata[2,3]
>  mydata[,2]
>  mydata[1,]
```

Data frames are two-dimensional objects so you can use square brackets with two comma-separated arguments to take subsets. There are other ways to extract a whole column:

```
>  mydata[[2]]
>  mydata$age
```

Often it is useful to have row names defined as unique subject indicators (or names):

```
>  rownames(mydata)<-mydata$name
>  mydata["Tom",]
```

Note that only values that are unique for each individual can be row (or column) names.
Now let's create another data frame with more individuals than the in first one:

```
> weights<-data.frame(weight=c(67,81,56,90,72,79,69))
> rownames(weights)=c("Ann","Peter","Sue","Jack","Tom","Joe","Jane")
> weights[substr(rownames(weights),1,1)=="J",]
```

How to add weights to individuals in mydata, using the new data frame?

```
> mydata$weight<- weights[rownames(mydata),"weight"]
```

(There is also the function `merge()` to join datasets – see `help(merge)`).

> **Exercise 3.** Using the same idea, add the variable `height` to the dataset `weights`. See what happens with the individuals who are not in `mydata`.

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

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

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

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

Another advantage of subset is that it will drop observations with missing values. In the example above, if a `height` is missing then `subset` will drop that row. But [] will do something you might not expect. It will include the row with missing `height`, but will replace every element in that row with the missing value `NA`.

> **Exercise 4.** Set the height of the 2nd observation to missing with `mydata$height[2] <- NA` and compare the two subset expressions.

2.1.8 Working with "real" data frames

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

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

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

```
help(births)
```

```
> names(births)
> head(births)
```

Exercise 5.

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

   ```
   > data(diet)
   ```

   and print the contents of the data frame to the screen..
2. Check that you now have two objects, `births`, and `diet` in your work space.
3. Obtain a description of the object `diet`.
4. Remove the object `diet` with the command

   ```
   > rm(diet)
   ```

   Check that the object `diet` is not any more there.

2.1.9 Referencing parts of the data frame (indexing)

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. Alternatively

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

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

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

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

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

and to list all the data for this variable, try

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

To list the data for the first subject on all variable except the second try

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

Exercise 6.

1. Display the data on the variable `gestwks` for row 7 in the `births` data frame.
2. Display all the data in row 7.
3. Display the first 10 rows of the data on the variable `gestwks`.

2.1.10 Summaries

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

```
> summary(births)
```

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

```
> names(births)
```

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

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

Alternatively you can use

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

In most datasets there will be some missing values. The summary shows the number of missing (`NA`) values for each variable.

2.1.11 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. For R to recognize that the codes refer to categories it is necessary to convert the variables to be factors, and to label the levels.

To convert the variable `hyp` to be a factor, try

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

Alternatively you can use

```
> births <- transform(births, hyp=factor(hyp))
> str(births)
```

Note that either way `hyp` is now a factor with two levels, labelled `"0"` and `"1"` which are the original values taken by the variable. It is possible to change the labels to (say) `"normal"` and `"hyper"` with

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

or

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

Exercise 7.

1. Convert the variable `sex` into a factor
2. Label the levels of `sex` as `"M"` and `"F"`.

2.1.12 Frequency tables

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

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

or by specifying the data frame as in

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

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

Exercise 8.

1. Find the frequency distribution of `sex`.
2. Find the two-way frequency distribution of `sex` and `hyp`.

2.1.13 Grouping the values of a numeric variable

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

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

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

Observations which are not inside the range specified in the `breaks()` part of the command result in missing values for the new factor. You can specify that you want to cut a variable into a given number of intervals of equal length by specifying the number of intervals. For example

```
> births <- transform(births, agegrp = cut(matage, breaks = 5, right=FALSE))
> with(births, table(agegrp))
```

shows 5 intervals of width 4.

Exercise 9.

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.

2.1.14 Generating new variables

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

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

produces the variable `logbw` in your work space (Global environment), while

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

produces the variable `logbw` in the `births` data frame. Logs base 10 are obtained with `log10( )`.

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

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

creates a logical variable `vlow` (in `births` with values TRUE and FALSE, according to whether `bweight` is less than 2000 or not. One common use of logical variables is to restrict a command to a subset of the data. For example, to create a new dataframe restricted to women with babies of very low birth weight, try

```
> births.low<-subset(births, bweight<2000)
> summary(births.low)
```

Exercise 10.

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

2.1.15 Where am I?

You can save any R object to disc. For example, to save the data frame `births` try

```
> save(births, file="births2")
```

which will save the births data frame in the file `births2`. 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. To load the object from the file use

```
> load("births2")
```

The commands `save()` and `load()` can be used with any R objects, but they are particularly useful when dealing with large data frames.

2.1.16 The search path

R organizes objects in different positions on a search path. The command

```
> search()
```

shows these positions. The first is the work space, or global environment, the second is the Epi package, the third is a package of commands called methods, the fourth is a package called stats, and so on. To see what is in the work space try

```
> objects()
```

You should see just the objects `births` and `diet`. The command `ls()` does the same as `objects()`. To see what is in the Epi package, try

```
> objects(2)
```

When you type the name of an object R looks for it in the order of the search path and will return the first object with this name that it finds. This is why it is best to start your session with a clean workspace, otherwise you might have an object in your workspace that masks another one later in the search path.

2.1.17 Attaching a data frame

The function `objects()` shows that the data frame `births` is in your workspace. To refer to variables in `births` by name it is necessary to specify the name of the data frame, as in `births$hyp`. This is quite cumbersome, and provided you are working primarily with one data frame, it can help to put a copy of the variables from a data frame in their own position on the search path. This is done with the function

```
> attach(births)
```

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

```
> objects(2)
```

which shows the objects in this position are the variables from the `births` data frame. Note that the methods package has now been moved up to position 3, as shown by the `search()` function.

When you type the command:

```
> hyp
```

R will look in the first position where it fails to find `hyp`, then the second position where it finds `hyp`, which now gets printed.

Although convenient, attaching a data frame can give rise to confusion. For example, when you create a new object from the variables in an attached data frame, as in

```
> subgrp <- bweight[hyp==1]
```

the object `subgrp` will be in your workspace (position 1 on the search path) not in position 2. To demonstrate this, try

```
> objects(1)
> objects(2)
```

Similarly, if you modify the data frame in the workspace the changes will not carry through to the attached version of the data frame. The best advice is to regard any operation on an attached data frame as temporary, intended only to produce output such as summaries and tabulations.

Beware of attaching a data frame more than once - the second attached copy will be attached in position 2 of the search path, while the first copy will be moved up to position 3. You can see this with

```
> attach(births)
> search()
```

Having several copies of the same data set can lead to great confusion. To detach a data frame, use the command

```
> detach(births)
```

which will detach the copy in position 2 and move everything else down one position. To detach the second copy repeat the command `detach(births)`.

Exercise 11.

1. Use `search()` to make sure you have no data frames attached.
2. Use `objects()` to check that you have the data frame `births` in your work space.
3. Verify that typing `births$hyp` will print the data on the variable `hyp` but typing `hyp` will not.
4. Attach the `births` data frame in position 2 and check that the variables from this data frame are now in position 2.
5. Verify that typing `hyp` will now print the data on the the variable `hyp`.
6. Summarize the variable `bweight` for hypertensive women.

2.1.18 Interactive use vs. scripting

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

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

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

2.2 Reading data into R

2.2.1 Introduction

It is said that Mrs Beeton, the 19th century cook and writer, began her recipe for rabbit stew with the instruction "First catch your rabbit". Sadly, the story is untrue, but it does contain an important moral. R is a language and environment for data analysis. If you want to do something interesting with it, you need data.

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

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

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

```
> help(Titanic)
```

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

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

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

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

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

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

You will need the following files in the sub-directory `data` of your working directory: `fem.dat`, `fem-dot.dat`, `fem.csv`, `fem.dta` (Reminder: use `setwd()` to set your working directory).

2.2.2 Data sources

Sources of data can be classified into three groups:

1. Data in human readable form, which can be inspected with a text editor.
2. Data in binary format, which can only be read by a program that understands that format (SAS, SPSS, Stata, Excel, ...).
3. Online data from a database management system (DBMS)

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

2.2.3 Data in text files

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

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

1. How are columns in the table separated?
2. How are missing values represented?
3. Are variable names included in the file?

The file `fem.dat` contains data on 118 female psychiatric patients. It is available in different forms in the folder http://bendixcarstensen.com/Epi/Courses/NNepi/data/

The data set contains nine variables:

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

Inspect the file `fem.dat` with a text editor (such as Notepad) to answer the questions above.

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

Read in the table with (depends on which folder you used for the data files):

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

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

You can see the names of the variables with

```
> names(fem)
```

The structure of the data frame can be seen with

```
> str(fem)
```

You can also inspect the top few rows with

```
> head(fem)
```

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

You can recode the missing values with

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

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

2.2.4 Things that can go wrong

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

2.2.4.1 Forgetting the headers

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

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

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

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

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

```
> fem2$IQ
```

as the variable will not exist

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

2.2.4.2 Using the wrong separator

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

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

How many variables are there in the data set?

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

2.2.4.3 Mis-specifying the representation of missing values

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

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

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

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

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

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

2.2.5 Spreadsheet data

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

To understand why there are two functions, inspect the contents of the function `read.csv()` by typing its name

```
> read.csv
function (file, header = TRUE, sep = ",", quote = "\"", dec = ".",
    fill = TRUE, comment.char = "", ...)
read.table(file = file, header = header, sep = sep, quote = quote,
    dec = dec, fill = fill, comment.char = comment.char, ...)
<bytecode: 0x2e9b138>
<environment: namespace:utils>
```

The first two lines show the arguments to the `read.csv()` function and their default values (`header=TRUE`, etc) The next two lines show the *body* of the function, which shows that the default arguments are simply passed verbatim onto the `read.table()` function.

Hence `read.csv()` is a *wrapper* function that chooses the correct arguments for `read.table()` for you. You only need to supply the name of the CSV file and all the other details are taken care of.

Now inspect the `read.csv2` function to find the difference between this function and `read.csv`.

> **Explanation:** The CSV format is not a single standard. The file format depends on the *locale* of your computer – the settings that determine how numbers are represented. In some countries, the decimal separator is a point "." and the variable separator in a CSV file is a comma ",". In other countries, the decimal separator is a comma "," and the variable separator is a semi-colon ";". The `read.csv()` function is used for the first format and the `read.csv2()` function is used for the second format.

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

On Microsoft Windows, you can copy values directly from an open Excel spreadsheet using the clipboard. Highlight the cells you want to copy in the spread sheet and select copy from the pull-down edit menu. Then type `read.table(file="clipboard")` to read the data in. Beware, however, that the clipboard on Windows operates on the WYSIWYG principle (what-you-see-is-what-you-get). If you have a value 1.23456789 in your spreadsheet, but have formatted the cell so it is displayed to two decimal places, then the value read into R will be the truncated value 1.23.

2.2.6 Binary data

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

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

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

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

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

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

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

```
> attributes(fem5)
```

or just the attribute names with

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

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

2.2.7 Summary

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

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

2.3 Simple simulation

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

1. Whenever using a *random number generator* (RNG) for a simulation study, or for producing a randomization list to be used in a clinical trial, it is a good practice to set first the *seed*. It is a number that determines the initial state of the RNG, from which it starts creating the desired sequence of pseudo-random numbers. Explicit specification of the seed enables the reproducibility of the sequence. In serious applications (like in clinical trials) it is important that the seed (and the whole randomization list) is concealed from persons with certain important roles in the study (like the physicians who recruit or treat the patients in a clinical trial). – Instead of the number 5462319 below you may use your own seed of choice.

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

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

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

   Repeat the above lines and compare the results.

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

4. Generate a sample of size 1000 from a Uniform(0,1) distribution (`runif(1000)`) and look at the a) histogram of the original values, b) histogram of their natural logarithms, and c) histogram of the logit-transforms of the values.

```
> x <- runif(1000)
> hist(x)
> hist(log(x))
> hist(log(x/(1-x)))
```

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

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

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

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

7. Generate data obeying a simple linear regression model $y_i = 5 + 0.5x_i + \varepsilon_i$, $i = 1, \ldots, 100$, in which $\varepsilon_i \sim N(0, 10^2)$, and x_i values are integers from 1 to 100. Plot the (x_i, y_i)-values, and estimate the parameters of that model.

```
> x <- 1:100
> y <- 5 + 0.5*x + rnorm(100,0,10)
> plot(x,y)
> abline(lm(y~x))
> summary(lm(y~x))$coef
```

8. Now change the slope coefficient of x to (a) 0.05, and (b) 0.01, respectively, and perform a similar simulation run. Do you still discover association between x and y? (Look at the scatter plot and the error margin of the slope coefficient). Check also, what happens if you change the standard deviation of the error term to be a) 1, and b) 100.

2.4 Tabulation

2.4.1 Introduction

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

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

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

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

2.4.2 The births data

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

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

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

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

Alternatively it can be dome with the `transform` function, which avoids typing `births$` repeatedly:

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

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

2.4.3 One-way tables

The simplest table one-way table is created by

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.4.4 Improving the Presentation of Tables

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

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

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

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

2.4.5 Two-way Tables

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

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

Add the count to this table and repeat the function call using `margin = TRUE` to calculate the marginal tables.

Use `stat.table` with the ratio function to obtain a 2 × 2 table of percent low birth weight by `sex` and `hyp`.

You can have fine-grained control over which margins to calculate by giving a logical vector to the `margin` argument. Use `margin=c(FALSE, TRUE)` to calculate margins over `sex` but not `hyp`. This might not be what you expect, but the `margin` argument indicates which of the index variables are to be **marginalized out**, not which index variables are to remain.

2.4.6 Printing

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

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

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

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

2.5 Graphics in R

There are three kinds of plotting functions in R:

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

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

2.5.1 Simple plot on the screen

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

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

Now attach the dataframe and look at the birthweight distribution with

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

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

```
> help(hist)
```

and try some of the options, for example:

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

To look at the relationship between birthweight 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)
```

Exercise 12.

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)")
   ```

2.5.2 Colours

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

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

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

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

2.5.3 Adding to a plot

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

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

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

Then 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"))
```

which puts the legend in the top left hand corner.

Finally we can add a title to the plot with

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

2.5.3.1 Using indexing for plot elements

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

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

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

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

```
> c("blue","red")
> 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 over 40 say. We first generate the indexing variable:

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

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

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

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

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

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

Exercise 13.

1. Make a three level factor for maternal age with cutpoints at 30 and 40 years using the `cut` function. (Recall that the `breaks` argument must include lower and upper limits beyond the range of the data, or you will get some missing values).
2. Use this to make the plot of bweight versus gestational weeks with three different plotting symbols. (Hint: Indexing with a factor automatically gives indexes 1,2,3 etc.).

2.5.3.2 Generating colours

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

```
> rainbow(4)
```

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

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

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

2.5.4 Interacting with a plot

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

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

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

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

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

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

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

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

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

2. View all the variables corresponding to these records with

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

2.5.5 Saving your graphs for use in other documents

Once you have a graph on the screen you can click on $\boxed{\text{File}}\rightarrow\boxed{\text{Save as}}$, and choose the format you want your graph in. The `PDF` (Acrobat reader) format is normally the most economical, and Acrobat reader has good options for viewing in more detail on the screen. The `Metafile` format will give you an enhanced metafile `.emf`, which can be imported into a Word document by $\boxed{\text{Insert}}\rightarrow\boxed{\text{Picture}}\rightarrow\boxed{\text{From File}}$. Metafiles can be resized and edited inside Word (This graphics device is only available on Windows).

If you want exact control of the size of your plot-file you can start a non-interactive 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 portable document file `plot1.pdf` with a graph which is 3 inches tall and 4 inches wide.

2.5.6 The `par()` command

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

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

```
> help(par)
```

or better, use the the html-version through $\boxed{\text{Help}}\rightarrow\boxed{\text{Html help}}\rightarrow\boxed{\text{Packages}}\rightarrow\boxed{\text{graphics}}\rightarrow\boxed{\text{P}}\rightarrow\boxed{\text{par}}$.

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

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

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

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

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

To restore the layout to a single plot per page use

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

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

2.6 Calculation of rates, RR and RD

This exercise is `very` prescriptive, so you should make an effort to really understand everything you type into R.

Recall that the standard error of log-rate is $1/\sqrt{D}$, so that a 95% confidence interval for the log of a rate is:

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

If we take the exponential, we get the confidence interval for the rate:

$$\lambda \overset{\times}{\div} \underbrace{\exp(1.96/\sqrt{D})}_{\text{error factor,erf}}$$

1. Now, suppose you have 15 events during 5532 person-years. Now use R as a simple desk calculator to derive the rate and a confidence interval:

```
> library( Epi )
```

```
> D <- 15
> Y <- 5532
> rate <- D / Y
> erf <- exp( 1.96 / sqrt(D) )
> c( rate, rate/erf, rate*erf )
```

   You can explore the function `ci.mat()`, which lets you use matrix multiplication to produce confidence interval from an estimate and a standard error (or columns of such):

```
> ci.mat()
> exp( c( log(D/Y), 1/sqrt(D) ) %*% ci.mat() )
```

2. Now try to achieve this estimate and c.i. using a Poisson model. Use the number of events as the response and the log-person-years as offset:

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

   What is the interpretation of the parameter in this model?

3. You can extract a confidence interval directly from the model with the `ci.lin` or `ci.exp` functions from `Epi`:

```
> ci.lin( mm )
> ci.exp( mm )
```

4. There is an alternative way to fit a Poisson model, using the rates a the Poisson response, and the person-years as weights instead (albeit it will give you a warning about non-integer response in a Poisson model):

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

Verify that this gives the same results as above.

5. The advantage of this latter approach is that it will also make sense to use an identity link — the response is the same but the parameter estimated is now the rate, not the log-rate:

```
> ma <- glm( D/Y ~ 1, weight=Y, family=poisson(link=identity) )
```

   What is the meaning of the intercept in this model?

   Verify that you actually get the same rate estimate as before.

6. Now use `ci.lin` or `ci.exp` to produce the estimate and the confidence intervals from this model:

```
> ci.lin( ma )
> ci.exp( ma, Exp=FALSE )
> ci.exp( ma, Exp=F )
```

   Why are the confidence limits not the same as from the multiplicative model?

   (*Esoteric, but enlightning:*) Derive the formula for the standard error of this estimated rate.

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

```
> Dx <- c(3,7,5)
> Yx <- c(1412,2783,1337)
> Px <- 1:3
```

   Try to fit the same model as before to the data from the separate periods.

```
> m1  <- glm(     Dx   ~ 1, offset=log(    Yx ), family=poisson )
> m1s <- glm( sum(Dx)  ~ 1, offset=log(sum(Yx)), family=poisson )
```

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

```
> mp <- glm( Dx ~ factor(Px), offset=log(Yx), family=poisson )
```

   and compare the two models using `anova` with the argument `test="Chisq"`:

```
> anova( m1, mp, test="Chisq" )
```

   Compare the test statistic to the deviance of the model `mp`.

   What is the deviance good for?

9. If we have observations of two rates λ_1 and λ_0, based on (D_1, Y_1) and (D_0, Y_0) the variance of the difference of the log of the rates, that is the log(RR), is:

$$\begin{aligned}\text{var}(\log(\text{RR})) &= \text{var}(\log(\lambda_1/\lambda_0)) \\ &= \text{var}(\log(\lambda_1)) + \text{var}(\log(\lambda_0)) \\ &= 1/D_1 + 1/D_0\end{aligned}$$

   As before a 95% c.i. for the RR is then:

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

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

   Compute the the rate-ratio and c.i. by:

```
> D0 <- 15   ; D1 <- 28
> Y0 <- 5532 ; Y1 <- 4783
> RR <- (D1/Y1)/(D0/Y0)
> erf <- exp( 1.96 * sqrt(1/D0+1/D1) )
> c( RR, RR/erf, RR*erf )
> exp( c( log(RR), sqrt(1/D0+1/D1) ) %*% ci.mat() )
```

10. Now achieve this using a Poisson model:

```
> D <- c(D0,D1) ; Y <- c(Y0,Y1); xpos <- 0:1
> mm <- glm( D ~ factor(xpos), offset=log(Y), family=poisson )
```

   What does the parameters mean in this model?

   You can extract the exponentiated parameters by:

```
> ci.exp( mm )
```

11. If we instead want the rate-difference, we just subtract the rates, and the variance of the difference is (since the rates are based on independent samples) just the sum of the variances:

$$\begin{aligned}\text{var}(\text{RD}) &= \text{var}(\lambda_1) + \text{var}(\lambda_0) \\ &= D_1/Y_1^2 + D_0/Y_0^2\end{aligned}$$

   Use this formula to compute the rate difference and a 95% confidence interval for it:

```
> rd <- diff( D/Y )
> sd <- sqrt( sum( D/Y^2 ) )
> c( rd, sd ) %*% ci.mat()
```

12. Verify that this is the confidence interval you get when you fit an additive model with exposure as factor:

```
> ma <- glm( D/Y ~ factor(xpos), weight=Y,
+                  family=poisson(link=identity) )
> ci.exp( ma, Exp=FALSE )
```

13. Normally one would like to get both the rates and the ratio between them. This can be achieved in one go using the `ctr.mat` argument to `ci.exp`. Try:

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

14. Refit the model with `Y/1000` as the person time, so you get the estimated rates in units of cases per 1000.

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

```
> rownames( CM ) <- c("rate 0","rate 1","RD 1 vs. 0")
> ma <- glm( D/Y ~ factor(xpos), weight=Y,
+                  family=poisson(link=identity) )
> ci.exp( ma, ctr.mat=CM, Exp=FALSE )
```

2.7 Fitting a smooth curve

1. For illustration we fit a very crude model to the mortality follow-up of the men in the Danish Diabetes register:

```
> library( Epi )
> library( splines )
> data( DMlate )
> head( DMlate )
```

2. Now define outcome and age and date of diagnosis for convenience, and restrict data to only men:

```
> DMlate <- transform( DMlate, D = !is.na(dodth),
+                              Y = dox-dodm,
+                              A = dodm-dobth,
+                              P = dodm )
> DMlate <- subset( DMlate, Y>0 & sex=="M" )
> str( DMlate )
```

3. Now fit a model for mortality only depending on age and date at entry:

```
> m0 <- glm( D ~ A + P, family=poisson, offset=log(Y), data=DMlate )
```

— and use `ci.lin` and `ci.exp` to explore the parameters:

```
> ci.lin( m0 )
> round( ci.exp( m0 )[-1,], 3 )
```

How much is mortality changing by age and how much by calendar time?

4. How much older should a man be in order to double his mortality?

5. Now try to add a quadratic term in date of diagnosis to the model:

```
> mq <- glm( D ~ A + P + I(P^2), family=poisson, offset=log(Y), data=DMlate )
> round( ci.lin( mq ), 3 )
```

What is the interpretation of the coefficients now (if any)?

6. Now try to make a graph of the RR of death relative to 2005, say: For a given time P the log-rate is

$$\mu + \alpha \mathtt{A} + \beta_1 \mathtt{P} + beta_2 \mathtt{P}^2$$

and for 2005 the log-rate is:

$$\mu + \alpha \mathtt{A} + \beta_1 \mathsf{2005} + beta_2 \mathsf{2005}^2$$

so the log-RR between these is:

$$\beta_1 \mathtt{P} + beta_2 \mathtt{P}^2 - \beta_1 \mathsf{2005} - beta_2 \mathsf{2005}^2 = \beta_1(\mathtt{P} - \mathsf{2005}) + \beta_2(\mathtt{P}^2 - \mathsf{2005}^2)$$

Now devise some points between 1995 and 2010 and construct the two columns of numbers to be multiplied by the two parameters:

```
> P.pt <- 1995:2010
> p1 <- P.pt - 2005
> p2 <- P.pt^2 - 2005^2
> coef( mq )
> lRR <- coef(mq)[3] * p1 + coef(mq)[4] * p2
>  RR <- exp( lRR )
> plot( P.pt, RR, type="l", lwd=3, log="y" )
> abline( h=1,v=2005)
```

What is the substantial conclusion from the shape of the RR relative to 2005?

7. Draw the RR relative to year 2000 Is the conclusion substantially different?

8. This curve does not give the confidence limits; to this end we must use the estimated standard errors of β_1 and β_2 *and* the estimated covariance between them. That is hairy.

   There is a facility in the functions `ci.lin` and `ci.exp`, that will both select the relevant parameters (in this case those with names `P` and `I(P^2)`).

   Try (and look at the help page for `ci.lin`):

```
> ci.lin( mq, subset="P" )
> ci.exp( mq, subset="P" )
```

   There is a further argument to these functions, `ctr.mat` — contrast matrix which is the columns of numbers we defined above to multiply by each of the parameters, try:

```
> cbind( p1, p2 )
> ci.exp( mq, subset="P", ctr.mat=cbind(p1,p2) )
```

9. The result is the estimated curve with confidence intervals, so plot it (use the function `matplot`):

```
> matplot( P.pt, ci.exp( mq, subset="P", ctr.mat=cbind(p1,p2) ),
+          type="l", col="Black", lty=1, lwd=c(3,1,1), log="y" )
> abline( h=1,v=2000)
```

10. What really goes on here is that we as contrast matrix use the difference between the matrix of `P` and `P`^{2}, and the matrix that consists of the 2000-row all way through:

```
> ( MP <- cbind( P.pt, P.pt^2 ) )
> ( Mr <- cbind( rep(2000,length(P.pt)), 2000^2 ) )
> MP-Mr
> ci.exp( mq, subset="P", ctr.mat=MP-Mr )
> matplot( P.pt, ci.exp( mq, subset="P", ctr.mat=MP-Mr ),
+          type="l", col="Black", lty=1, lwd=c(3,1,1), log="y" )
> abline( h=1,v=2000)
```

11. Now suppose we want to model the period effect by a `cubic spline` instead. This is a function that is a cubic between a set of points called *knots*. In the `Epi` package is a function `Ns`, that will generate a set of columns corresponding to this — think of it as the counterpart of the columns `P` and `P`^{2}:

```
> p.kn <- c(1997,2000,2003,2006,2009)
> Ns( P.pt, knots=p.kn )
```

Now put this in a model:

```
> ms <- glm( D ~ A + Ns(P,knots=p.kn), family=poisson, offset=log(Y), data=DMlate )
> ci.exp( ms )
> ci.exp( ms, subset="P" )
```

The parameters do not have any meaning *per se*, just as in the case of the coefficients in the quadratic case.

12. Now extract the RR relative to 2005 from this new model as before.

    First construct the matrix for the points 1995—2010 and the reference for 2005:

```
> CP <- Ns( P.pt, knots=p.kn )
> Cr <- Ns( rep(2005,length(P.pt)), knots=p.kn )
> CP-Cr
```

    Then we can extract the RR ad plot it:

```
> RR <- ci.exp( ms, subset="P", ctr.mat=CP-Cr )
> matplot( P.pt, RR,
+          type="l", col="Black", lty=1, lwd=c(3,1,1), log="y" )
> abline( h=1,v=2000)
```

13. Now explore whether there is any non-linearity by age (on the log-mortality scale, that is):

```
> ma <- update( ms, . ~ . + I(A^2) )
> round( ci.lin( ma ), 3 )
```

14. In order to report the model `ms` in full, we will also need to show the estimated mortality rates as a function of age. For that purpose we of course must use the 2005 reference point.

    (a) The first possibility is to devise a prediction data frame:

```
> nd <- data.frame( A = 40:85, P=2005, Y=1000 )
```

    Note that you must provide values for *all* covariates, including the person-years, that enter in the model as offset, that is as a covariate with fixed regression coefficient. The function `ci.pred` is a simple convenience wrapper for `predict.glm`:

```
> rate <- ci.pred( ma, newdata=nd )
> matplot( nd$A, rate,
+          type="l", col="Black", lty=1, lwd=c(3,1,1), log="y" )
```

    Note that since you initially entered `Y` in units of 1 person-year, we get the rates in units of events per 1000 person-years by entering Y with the value of 1000 in the prediction frame.

(b) The other possibility is to use `ci.exp` directly to extract the predicted rates from the model, but they will be in he units of the `Y` entered into the model. Note that you in this case must be careful to get the order of the columns in `ctr.mat` right:

```
> ci.exp( ma )
> Rate <- ci.exp( ma, ctr.mat=cbind(1,40:85,Ns(rep(2005,46),knots=p.kn),(40:85)^2) )*1000
> matplot( nd$A, Rate,
+           type="l", col="Black", lty=1, lwd=c(3,1,1), log="y" )
```

15. Try to explore if there are further non-linearities in the age-effect by including a spline with several knots

```
> ( a.kn <- 3:9*10 )
> mA <- update( ms, . ~ . - A + Ns(A,knots=a.kn) )
> round( ci.lin( mA ), 3 )
```

How many parameters are in the age-effect?

```
> rate <- ci.pred( mA, newdata=nd )
> matplot( nd$A, rate,
+           type="l", col="Black", lty=1, lwd=c(3,1,1), log="y" )


> ci.exp( ms )
```

2.8 Cox and Poisson modelling

This practical is to show how results from a Cox-model can be reproduced exactly by a Poisson model, and in particular how more sensible and relevant results can be obtained from a Poisson model.

2.8.1 The lung cancer data

The data is the lung cancer data from the `survival` package which comes with `R` by default. We start by declaring a really large chunk of memory, because we need that to fit a silly model for illustration:

```
> # memory.size( 3000 )
> library( Epi )
> library( survival )
> sessionInfo()
```

Note that loading the `survival` package automatically also loads the `splines` package, which is also needed in the exercise.

1. First, load the `lung` data set and have a look at it:

   ```
   > data( lung )
   > str( lung )
   > lung[1:10,]
   ```

2. The deaths are indicated by `status` being equal to 2 — how may deaths are there?

3. How many distinct survival times are there?

2.8.2 Cox-models

4. Fit a traditional Cox-model for the the Mayo clinic lung cancer by `coxph`, where the response is a `Surv` object:

   ```
   > system.time(
   + m0.cox <- coxph( Surv( time, status==2 ) ~ age + factor( sex ),
   +                 method="breslow", eps=10^-8, iter.max=25, data=lung )
   +            )
   > summary( m0.cox )
   ```

5. Create a Lexis object from the dataset

   ```
   > Lung <- Lexis( exit = list( tfe=time ),
   +                exit.status = factor(status,labels=c("Alive","Dead")),
   +                data = lung )
   > summary( Lung )
   > head( Lung )
   ```

   What do you see from the `summary` command?

6. Now try to fit the same Cox-model to data using the formal structures of the `Lexis` object:

   ```
   > mL.cox <- coxph( Surv( tfe, tfe+lex.dur, lex.Xst=="Dead" ) ~
   +                  age + factor( sex ),
   +                  method="breslow", eps=10^-8, iter.max=25, data=Lung )
   > cbind( coef(m0.cox), coef(mL.cox) )
   ```

2.8.3 Poisson models

7. Now split the follow-up data split in small intervals, using all recorded survival times as breakpoints:

   ```
   > Lung.s <- splitLexis( Lung,
   +                       breaks=c(0,sort(unique(Lung$time))),
   +                       time.scale="tfe" )
   > summary( Lung.s )
   ```

   List all records from one person you choose — use a table of the variable `lex.id` to identify a person with not too many records.

8. Now fit the Cox model to the split dataset

   ```
   > system.time(
   + mLs.cox <- coxph( Surv( tfe, tfe+lex.dur, lex.Xst=="Dead" ) ~
   +                   age + factor( sex ),
   +                   method="breslow", eps=10^-8, iter.max=25, data=Lung.s )
   +             )
   ```

   Are the results the same?

9. Now fit a Poisson model with a factor accommodating the time-scale defined in the `Lexis` object. You should use the command `factor` to devise a categorical variable:

   ```
   > nlevels( factor( Lung.s$tfe ) )
   ```

   Note it involves fitting a model with many parameters, so will take some time. Note that the response variable `lex.Xst=="Dead"` is a logical, but by R converts it into a 0/1 numeric:

   ```
   > system.time(
   + mLs.pois.fc <- glm( lex.Xst=="Dead" ~ factor( tfe ) +
   +                             age + factor( sex ),
   +                             offset = log(lex.dur),
   +                       family=poisson, data=Lung.s, eps=10^-8, maxit=25 )
   +             )
   > length( coef(mLs.pois.fc) )
   ```

   How does the regression coefficients look compared to the Cox-model?

10. Now replace the factor-model for the time-scale by a smooth spline function. A (cubic) spline is a function made up of 3$^{\text{rd}}$ degree polynomials in different intervals defined by knots, in such a way that the polynomials fit nicely together at the knots.

    First defining the knots for the spline, for example:

    ```
    > t.kn <- c(0,25,100,500,1000)
    > dim( Ns(Lung.s$tfe,knots=t.kn) )
    ```

    and then fit the model using `Ns` (look it up!) from the `Epi` package:

    ```
    > system.time(
    + mLs.pois.sp <- glm( lex.Xst=="Dead" ~ Ns( tfe, knots=t.kn ) +
    +                             age + factor( sex ),
    +                       offset = log(lex.dur),
    +                       family=poisson, data=Lung.s, eps=10^-8, maxit=25 )
    +             )
    > ci.exp( mLs.pois.sp )
    > ci.exp( mLs.pois.sp, subset=c("age","sex") )
    > ci.exp( mLs.pois.fc, subset=c("age","sex") )
    ```

2.8.4 Comparing Cox and Poisson models

11. Compare the estimates of the regression parameters and their confidence intervals between the Cox-model, the factor-Poisson-model and the spline Poisson model.

    What do you conclude?

12. Now use the fitted model to derive the estimated mortality at 0, 10, 20, ..., 1000 days after diagnosis. You must set up a *contrast matrix* with columns corresponding to the parameters of the model, and rows corresponding to the points in time where you want the mortality:

    ```
    > CM <- cbind( 1, Ns( seq(0,1000,10), knots=t.kn ), 60, 1 )
    > CM[1:5,]
    ```

    The mortality rates at these time points, for a 60-year old man are then:

    ```
    > lambda <- ci.exp( mLs.pois.sp, ctr.mat=CM )
    ```

    What are the units in which `lambda` is measured?

    Also compute the *cumulative* mortality rates (including the s.e.of this), by using the function `ci.cum` (look it up!):

    ```
    > Lambda <- ci.cum( mLs.pois.sp, ctr.mat=CM, intl=10 )
    > Lambda <- rbind( 0, Lambda )
    ```

    Also get the estimate of the survival curve for a male aged 60 from the Cox-model; remember that sex must be specified as a factor with two levels in the data frame in the argument `newdata`:

    ```
    > sf <-  survfit( m0.cox,
    +                 newdata=data.frame( sex=factor(2,levels=1:2),
    +                                     age=c(60) ) )
    ```

13. Plot the mortality rates (`lambda`) as a function of time since diagnosis.

    Also plot the estimated survival function from the Cox model on top of the estimated survival function based on the cumulative hazard, using the relationship:

    $$S(t) = \exp(-\Lambda(t))$$

    How do the survival curves from the two approaches compare? Which one do you consider the more sensible summary of the survival of 60 year old men with lung cancer?

2.9 Diabetes in Denmark: Time-splitting, time-scales and SMR

This exercise is using data from the National Danish Diabetes register. There is a sample of 10,000 records from this in the `Epi` package. Actually there are two, we shall use the one with only cases of diabetes diagnosed after 1995. This is of interest because it is only for these where the data of diagnosis is certain, and hence for whom we can compute the duration of diabetes during follow-up. The exercise is about assessing how mortality depends age, calendar time and duration of diabetes. And how to understand and compute SMR, and assess how it depends on these factors as well.

1. First load the data and take a look at the data:

   ```
   library( Epi )
   data( DMlate )
   str( DMlate )
   ```

   You can get a more detailed explanation of the data by referring to the help page:

   ```
   ?DMlate
   ```

2. Set up the dataset as a `Lexis` object with age, calendar time and duration of diabetes as timescales, and date of death as event. Make sure that you know what each of the arguments to `Lexis` mean:

   ```
   LL <- Lexis( entry = list( A = dodm-dobth,
                              P = dodm,
                            dur = 0 ),
                 exit = list( P = dox ),
          exit.status = factor( !is.na(dodth),
                                labels=c("Alive","Dead") ),
                 data = DMlate )
   ```

   Take a look at the first few lines of the resulting dataset using `head()` and an overview using `summary()`

3. Get an overall overview of the mortality by sex by using `stat.table` to tabulate no. deaths, person-years and the crude mortality rate for both sexes.

4. If we want to assess how mortality depends on age, calendar time and duration, we should split the follow-up along all three time scales. In practice it is sufficient to split it along one of the time-scales and then just use the value of each of the time-scales at the left endpoint of the intervals. Use `splitLexis` to split the follow-up along the age-axis in 1-year age-intervals:

   ```
   SL <- splitLexis( LL, breaks=seq(0,125,1), time.scale="A" )
   summary( SL )
   ```

   How many records are now in the dataset? How many person-years? Compare to the original `Lexis`-dataset.

5. Now estimate a crude age-specific mortality curve for men and women separately, using natural splines. But first define a set of knots on the age-scale; this is just points where we assume that curve is a 3rd degree polynomial between each pair of points:

   ```
   library( splines )
   a.kn <- seq(5,95,10)
   r.m <- glm( (lex.Xst=="Dead") ~ Ns( A, knots=a.kn, intercept=TRUE ) - 1,
               offset = log( lex.dur ),
               family = poisson,
                 data = subset( SL, sex=="M" ) )
   r.f <- update( r.m, data = subset( SL, sex=="F" ) )
   ```

   Make sure you understand all the components on this modeling statement. With these objects we can get the estimated log-rates by using `ci.pred`, and supplying a data frame of prediction points, so first make a data frame of prediction points, it must have variables corresponding to the predictor variables in the model, including the off-set variable.

   ```
   nd <-  data.frame( A = seq(10,90,0.5),
                lex.dur = 1000)
   p.m <- ci.pred( r.m, newdata = nd )
   p.f <- ci.pred( r.f, newdata = nd )
   str( p.m )
   ```

   Plot the two sets of estimated rates (men and women).
   (Hint: use `matplot`)

6. We can compare the mortality rates of the diabetes patients with the mortality rates from the general population; they are available in the data frame `M.dk`

   ```
   data( M.dk )
   head( M.dk )
   ```

   Plot the mortality rates from a particular year on top of the estimated rates, for example:

   ```
   with( subset( M.dk, sex==1 & P==2005 ), lines( A, rate, col="blue", lty="12", lwd=3 ) )
   ```

   Now, guess how to plot the mortality rates for women. . .

7. It would more natural to model the population mortality rates in a similar fashion as the diabetes mortality rates, try:

   ```
   R.m <- glm( D ~ Ns( A, knots=a.kn ),
               offset = log( Y ),
               family = poisson,
                 data = subset( M.dk, sex==1 & P>1994 ) )
   ```

   Now obtain the same model for women, and construct the predicted rates as before — note that you will need a a new dataset for prediction, because in the population dataset the person-years are called `Y` whereas in the dataset with the patient follow-up person-years were in a variable called `lex.dur`. Add the curves with predicted rates to the plot of the patient mortality rates.

8. We now want to model the mortality rates among diabetes patients also including current date and duration of diabetes. However, we shall not just use the positioning of knots for the splines as provided by `ns`, because this is based on the allocating knots so that the number of observations (lines in the dataset), is the same between knots. The information in a follow-up study is in the number of events, so it would be better to allocate knots so that number of events were the same between knots. We will be using so-called *natural splines* that are linear beyond the boundary knots, and hence we take the 5th and 95th percentile of deaths as the boundary knots for age (`A`) and calendar time (`P`) but for duration where we actually have follow-up from time 0 on the timescale we use 0 as the first knot. Therefore, find points (knots) so that the number of events is the same between each pair. Try this:

```
kn.A <- with( subset( SL, lex.Xst=="Dead" ),
              quantile( A+lex.dur, probs=seq(5,95,10)/100 ) )
```

   Take a look at where these points are and make a similar construction for calendar time (`P`) and diabetes duration (`dur`).

9. With these we can now model mortality rates (separately for men and women), as functions of age, calendar time and duration. To this end you will need the `splines` package and the convenience wrapper, `Ns` from the `Epi` package that does the allocation of knots. You can now specify a model very simply (remember to check the names of the vectors where you put the positions of the knots):

```
mm <- glm( (lex.Xst=="Dead") ~ Ns( A, kn=kn.A ) +
                               Ns( P, kn=kn.P ) +
                               Ns( dur, kn=kn.dur ),
           offset = log( lex.dur ),
           family = poisson,
             data = subset( SL, sex=="M" ) )
summary( mm )
mf <- update( mm, data = subset( SL, sex=="F" ) )
```

   What is the interpretation of the parameters (if any)?

10. How do these models fit relative to the models with only age as a descriptor of the rates? (Hint: Use the `anova`-function with the argument `test="Chisq"`). What is the problem with this approach?

11. The models that you fitted separately for men and women has three terms: age (`A`), calendar time (`P`) and diabetes duration (`dur`). Since the outcome is a rate with dimension time^{-1} we must put the rate dimension on one of these terms and leave the two others as rate-ratios. In order to do this we must fix reference values for the two rate-ratio terms. The natural variable for the rate-dimension is age, so that we get estimated age-specific rate-ratios for a specific calendar time, 1.1.2008, say, and a specific duration of diabetes, 2 years, say. In order to extract these terms from the model we need contrast matrices, that is matrices where each row corresponds to a set of values for age or period or duration, and the columns correspond to the spline basis as used in the model. This is one reason for explicitly fixing the knots in the spline definitions; when we extract the effects we must use the same set of knots as in the model specification. We will need matrices for specified set of values for age,

calendar time and duration, but also matrices where all rows refer to the chosen reference values for calendar time and duration. We begin by specifying the prediction points for the time scales and the reference points. There is formally no reason to require that the matrices all have the same number of rows, but it makes the handling of the reference points much easier.

```
N <- 100
pr.A <- seq(10,90,,N)
pr.P <- seq(1995,2010,,N)
pr.d <- seq(0,15,,N)
rf.P <- 2009
rf.d <- 2
```

With these in place we generate the matrices we shall multiply to the parameter estimates:

```
AC <- Ns( pr.A, knots=kn.A )
PC <- Ns( pr.P, knots=kn.P )
dC <- Ns( pr.d, knots=kn.dur )
PR <- Ns( rep(rf.P,N), knots=kn.P )
dR <- Ns( rep(rf.d,N), knots=kn.dur )
```

What are the dimensions of these matrices? Note that the rows of `AC` refer to `N` points on the age-scale, `PC` to N points on the calendar time scale, etc. These matrices are the necessary input for extracting the effects; this is done by the function `ci.exp`, remember to take a look at the help page for this. Note that we make use of *all* parameters when extracting the age-effect — this is the effect where we have the dimension of the response (rate), and hence the intercept, and where we have fixed the values of date and duration at their reference values. The rate-ratios for calendar time and duration are estimated exclusively from the parameters for these terms, but note that we subtract the values at the reference point:

```
m.A <- ci.exp( mm, ctr.mat=cbind(1,AC,PR,dR) ) * 1000
m.P <- ci.exp( mm, subset="P"  , ctr.mat=PC-PR )
m.d <- ci.exp( mm, subset="dur", ctr.mat=dC-dR )
f.A <- ci.exp( mf, ctr.mat=cbind(1,AC,PR,dR) ) * 1000
f.P <- ci.exp( mf, subset="P"  , ctr.mat=PC-PR )
f.d <- ci.exp( mf, subset="dur", ctr.mat=dC-dR )
```

12. Plot the three effects in three panels beside each other. Plot the estimates for men and women together. (Hint: use the function `matplot` to plot both estimates and confidence limits for both sexes in one go. How are the estimated effects — is the chosen parametrization plausible? Would you want to reconsider the number of knots you used for any of the terms?

13. We have so far fitted models separately for men and women, we might fit a model for the entire dataset with common period and duration effects, but different age-effect for the two sexes. Try to fit this interaction model and take a look at the estimates:

```
m2 <- glm( (lex.Xst=="Dead") ~ sex +
                           sex:Ns( A, kn=kn.A ) +
                               Ns( P, kn=kn.P ) +
                               Ns( dur, kn=kn.dur ),
           offset = log( lex.dur ),
           family = poisson,
             data = SL )
ci.exp(m2)
```

What do these estimates refer to?

14. Now test this model against the separate models; the deviance and degrees of freedom from the separate models for men and women add up to that of a joint model with interaction between all terms and sex. Compare the total deviance for these with that of the fitted interaction model Hint: to find the deviance and degrees of freedom, look at the model object by saying: `names(m2)`. Is there any evidence of different period and duration effects between thew sexes?

15. Would you test whether there were proportional (or even identical) mortality rates between men and women? And how?

16. Extract the parameters from the model, showing the separate age-effects for men and women, and the common period and duration effects. To get the age-specific rates for men and women at the reference time and reference duration you may need the subset-argument to `ci.exp`, for example:

```
ci.exp( m2, subset=c("Int","sexF","P","dur") )
```

17. The model we fitted has three time-scales: current age, current date and current duration of diabetes, so the effects that we report are not immediately interpretable, as they are (as in all multiple regression) to be interpreted as "all else equal" which they are not, as the three time scales advance by the same pace. The reporting would therefore more naturally be *only* on the mortality scale, but showing the mortality for persons diagnosed in different ages, using separate displays for separate years of diagnosis. This is most easily done using the `ci.pred` function with the `newdata=` argument. So a person diagnosed in age 50 will have a mortality measure in cases per 1000 PY as:

```
pts <- seq(0,20,1)
nd <- data.frame( A=  50+pts,
                  P=1995+pts,
                dur=     pts,
            lex.dur=1000 )
ci.pred( mm, newdata=nd )
```

Take a look at the result from the `ci.pred` statement and construct prediction of mortality for men and women diagnosed in a range of ages, say 50, 60, 70, and plot these together. sapply(predict(mm, newdata=nd, se.fit=TRUE)[1:2],cbind)

18. The model we used for the mortality rates used three time-scales: age, calendar time and duration of diabetes. It would be of interest to see whether we would get the same (or better) description by adding age at diagnosis and date of diagnosis to the model. Now, age at diagnosis = current age − duration of diabetes, and date of diagnosis = current date − duration of diabetes, so the terms we might add only constitute the *non-linear* effects of these variables. Now add the effects one at at time and test whether age at diagnosis or current age is the better predictor. You want to use a set of knots which is aligned to the new variables we consider, for example:

```
kn.Ad <- with( subset( SL, lex.Xst=="Dead" ),
               quantile( A-dur, probs=seq(5,95,10)/100 ) )
kn.Pd <- with( subset( SL, lex.Xst=="Dead" ),
               quantile( P-dur, probs=seq(5,95,20)/100 ) )
```

If you just want to test whether adding a new term to the model it is convenient to use the `update` function, for example:

```
anova( mm,
       update( mm, . ~ . + Ns(A-dur,knots=kn.Ad) ),
       update( mm, . ~ . + Ns(A-dur,knots=kn.Ad) - Ns(A,knots=kn.A) ),
       test = "Chisq" )
```

Is there any indication of whether current age or age at diagnosis, or current date or date of diagnosis is the better choice?

19. Fit the models with age at diagnosis and date of diagnosis as explanatory variables instead. To extract the effects you also need new contrast matrices, because the deaths are distributed differently along these “entry”-variables.

20. Show the effects together with the effects from the model with three time scales (that is the model with current age and current date of follow-up.

21. Now compute the expected number of deaths as the person-time multiplied by the corresponding population rate. Use `stat.table` to make a table of observed, expected and the ratio (SMR) by age (suitably grouped) and sex.

22. Now model the SMR by including the log-expected numbers instead of the log-person-years as offset, using separate models for men and women. Remember to exclude those units where no deaths in the population occur (that is where the rate is 0). Plot the estimates a s you did before for the rates. What do the extracted effects represent now?

23. Is there any difference between SMR for males and females?

24. Fit the model with common SMR for the two sexes. Plot the estimated common effects for SMR.

25. Try to simplify the model to one with effect of date of diagnosis, and using only knots at 0,1,and 2 years for duration, giving an estimate of the change in SMR as duration increases beyond 2 years.

26. What are the estimated annual change in SMR by date of diagnosis and by duration after 2 years? Thus the change in SMR is a 1.5% annual decrease (95¸i.: (0.3-2.7)%).

Chapter 3

Solutions

3.6 Calculation of rates, RR and RD

Recall that the standard error of log-rate is $1/\sqrt{D}$, so that a 95% confidence interval for the log of a rate is:

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

If we take the exponential, we get the confidence interval for the rate:

$$\lambda \overset{\times}{\div} \underbrace{\exp(1.96/\sqrt{D})}_{\text{error factor,erf}}$$

1. Now, suppose you have 15 events during 5532 person-years. Now we use R as a simple desk calculator to derive the rate and a confidence interval (note that you can stick several R-commands on one line if you separate them by ";"):

   ```
   > library( Epi )
   > D <- 15 ; Y <- 5532 ; rate <- D / Y ; erf <- exp( 1.96 / sqrt(D) )
   > c( rate, rate/erf, rate*erf )
   ```

   ```
   [1] 0.002711497 0.001634654 0.004497720
   ```

   The function `ci.mat()` returns a 2 by 3 matrix, which lets you use matrix multiplication to produce confidence interval from an estimate and a standard error (or columns of such):

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

   ```
        Estimate      2.5%    97.5%
   [1,]        1  1.000000 1.000000
   [2,]        0 -1.959964 1.959964
   ```

   ```
   > exp( c( log(D/Y), 1/sqrt(D) ) %*% ci.mat() )
   ```

   ```
           Estimate        2.5%       97.5%
   [1,] 0.002711497 0.001634669 0.004497678
   ```

2. Now we use a Poisson model to estimate a rate and its confidence interval. We use the number of events as the response and the log-person-years as offset:

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

Call:
glm(formula = D ~ 1, family = poisson, offset = log(Y))

Deviance Residuals:
[1]  0

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -5.9103     0.2582  -22.89   <2e-16

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: -8.8818e-16  on 0  degrees of freedom
Residual deviance: -8.8818e-16  on 0  degrees of freedom
AIC: 6.557

Number of Fisher Scoring iterations: 3
```

The default link used is the log link, and we are using the log-person-years as offset, so the model is:

$$\log\bigl(\mathrm{E}(D)\bigr) = \mu + \log(Y) \quad \Leftrightarrow \quad \log\bigl(\mathrm{E}(D)/Y\bigr) = \mu$$

The parameter μ in this model is therefore the log of the rate.

3. A confidence interval can be extracted directly from the model with the `ci.lin()` or `ci.exp()`-function from `Epi`; note that the `Exp=TRUE` argument will transform the estimate and the confidence interval to the rate-scale — normally we would only want this, and so subset the output from `ci.lin`

```
> ci.lin( mm )

             Estimate    StdErr         z             P      2.5%     97.5%
(Intercept) -5.910254 0.2581989 -22.89032 5.801722e-116 -6.416315 -5.404194

> ci.exp( mm )

              exp(Est.)        2.5%       97.5%
(Intercept) 0.002711497 0.001634669 0.004497678

> round( ci.exp( mm ), 5 )

            exp(Est.)    2.5%  97.5%
(Intercept)   0.00271 0.00163 0.0045
```

4. The alternative way to fit a Poisson model, using the rates a the Poisson response, and the person-years as weights instead (albeit it will give you a warning about non-integer response in a Poisson model):

```
> mmx <- glm( D/Y ~ 1, weight=Y, family=poisson )
> round( ci.exp( mmx ), 5 )

            exp(Est.)    2.5%  97.5%
(Intercept)   0.00271 0.00163 0.0045
```

We see that this gives the same results as above.

5. The advantage of this approach is that it will also make sense to use an identity link — the response is the same but the parameter estimated is now the rate, not the log-rate:

```
> ma <- glm( D/Y ~ 1, weight=Y, family=poisson(link=identity) )
```

The intercept in this model is now the rate itself, because of the identity link.

We see that we get the same estimate as before:

```
> log( coef(ma) )
(Intercept)
  -5.910254
```

6. We can then use `ci.lin` (or `ci.exp` with argument `Exp=FALSE`) to produce the estimate and the confidence intervals from this model:

```
> ci.lin( ma )
               Estimate       StdErr        z            P        2.5%       97.5%
(Intercept) 0.002711497 0.0007001054 3.872983 0.0001075112 0.001339315 0.004083678
> ci.exp( ma, Exp=FALSE )
               Estimate        2.5%       97.5%
(Intercept) 0.002711497 0.001339315 0.004083678
> round( ci.exp( ma, Exp=FALSE ), 5 )
            Estimate    2.5%   97.5%
(Intercept)  0.00271 0.00134 0.00408
```

The confidence limits from this model are based on the 2nd derivative of the log-likelihood with respect to the *rate*, and not as before with respect to the *log rate*, and therefor they are different — they are symmetrical on the rate-scale and not on the log-rate scale:

$$\ell(\lambda) = D\ln(\lambda) - \lambda Y \qquad \ell'(\lambda) = D/\lambda - Y \qquad \ell''(\lambda) = -D/\lambda^2\big|_{\lambda=D/Y} = -Y^2/D$$

Thus the observed information is Y^2/D and hence the approximate standard deviation of the rate is square root of the inverse of this, $\sqrt{D}/Y$, which is exactly the standard deviation you got from the model:

```
> c( sqrt(D)/Y, ci.lin( ma )[,2] )
[1] 0.0007001054 0.0007001054
```

In the classical model which is linera in the log-rate, the log-likelihood is in terms of $\theta = \log(\lambda)$:

$$\ell(\theta) = D\theta - e^{\theta}Y \qquad \ell'(\theta) = D - e^{\theta}Y \qquad \ell''(\theta) = -e^{\theta}Y\big|_{\theta=\log(D/Y)} = -D$$

so in this case the observed infromation is $-D$, and hence the s.e.of the estimate for θ id $1/\sqrt(D)$:

```
> c( 1/sqrt(D), ci.lin( mm )[,2] )

[1] 0.2581989 0.2581989
```

7. Now assume that the events and person years are collected over three time periods, which we for convenience number 1 to 3:

```
> Dx <- c(3,7,5)
> Yx <- c(1412,2783,1337)
> Px <- 1:3
> cbind( Dx, Yx, Px )

     Dx   Yx Px
[1,]  3 1412  1
[2,]  7 2783  2
[3,]  5 1337  3
```

If we fit the same model as before to the data from the separate periods, we get the same estimates, because the Poisson log-likelihood for three independent observations with the same relationship between mean and person-years is identical to the likelihood for the sum of the observations with an exp-offset equal to the sum of the exp-offsets:

$$\sum_i \bigl(D_i \log(\lambda) - \lambda Y_i\bigr) = \bigl(\sum_i D_i\bigr)\log(\lambda) - \lambda\bigl(\sum_i Y_i\bigr)$$

— basically this is a consequence of the fact that the likelihood for follow-up data with constant rate is additive *both* in the no. events *and* the person-time. SO we see we get the same estiamte as before;

```
> m1 <- glm( Dx ~ 1, offset=log(Yx), family=poisson )
> summary( m1 )

Call:
glm(formula = Dx ~ 1, family = poisson, offset = log(Yx))

Deviance Residuals:
      1        2        3
-0.4403  -0.2013   0.6824

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -5.9103     0.2582  -22.89   <2e-16

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 0.70003  on 2  degrees of freedom
Residual deviance: 0.70003  on 2  degrees of freedom
AIC: 12.98

Number of Fisher Scoring iterations: 4

> rbind( ci.exp( m1 ),
+        ci.exp( mm ) )

               exp(Est.)        2.5%       97.5%
(Intercept) 0.002711497 0.001634669 0.004497678
(Intercept) 0.002711497 0.001634669 0.004497678
```

8. With separate observations from three periods we can test whether the rates are the same in the three periods; we just fit a model with the period as a factor:

   ```
   > mp <- glm( Dx ~ factor(Px), offset=log(Yx), family=poisson )
   ```

   and compare the two models via a log-likelihood ratio test using `anova` with the argument `test="Chisq"`:

   ```
   > anova( m1, mp, test="Chisq" )

   Analysis of Deviance Table

   Model 1: Dx ~ 1
   Model 2: Dx ~ factor(Px)
     Resid. Df Resid. Dev Df Deviance Pr(>Chi)
   1         2    0.70003
   2         0    0.00000  2  0.70003   0.7047
   ```

   We note that the test statistic is the same as deviance of the model `mp`. This is because the deviance of a model is the log-likelihood ratio test statistic of the model versus the saturated model; i.e. the model with one parameter per observation, in this case the model `mp`.

9. If we have observations of two rates λ_1 and λ_0, based on (D_1, Y_1) and (D_0, Y_0) the variance of the difference of the log of the rates, that is the log(RR), is:

   $$\begin{aligned}\text{var}\big(\log(\text{RR})\big) &= \text{var}\big(\log(\lambda_1/\lambda_0)\big)\\ &= \text{var}\big(\log(\lambda_1)\big) + \text{var}\big(\log(\lambda_0)\big)\\ &= 1/D_1 + 1/D_0\end{aligned}$$

   As before a 95% c.i. for the RR is then:

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

   If we have 15 events during 5532 person-years in an unexposed group and 28 events during 4783 person-years in an exposed group, we can then compute the the rate-ratio and c.i. by:

   ```
   > D0 <- 15   ; D1 <- 28
   > Y0 <- 5532 ; Y1 <- 4783
   > RR <- (D1/Y1)/(D0/Y0)
   > erf <- exp( 1.96 * sqrt(1/D0+1/D1) )
   > c( RR, RR/erf, RR*erf )

   [1] 2.158980 1.153146 4.042153

   > exp( c( log(RR), sqrt(1/D0+1/D1) ) %*% ci.mat() )

        Estimate    2.5%    97.5%
   [1,]  2.15898 1.15316 4.042106
   ```

10. But this can also be achieved using a Poisson model:

```
> D <- c(D0,D1) ; Y <- c(Y0,Y1); xpos <- 0:1
> mm <- glm( D ~ factor(xpos), offset=log(Y), family=poisson )
> summary( mm )

Call:
glm(formula = D ~ factor(xpos), family = poisson, offset = log(Y))

Deviance Residuals:
[1]  0  0

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)    -5.9103     0.2582 -22.890   <2e-16
factor(xpos)1   0.7696     0.3200   2.405   0.0162

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 6.1110e+00  on 1  degrees of freedom
Residual deviance: 1.7764e-15  on 0  degrees of freedom
AIC: 13.733

Number of Fisher Scoring iterations: 3
```

The parameters in this model are:

`(Intercept)`: the log rate in the reference group, scaled to the units of `Y`

`factor(xpos)1`: the log RR between group 1 and 0.

We can extract the exponentiated parameters, corresponding to the rate and the rate-ratio by:

```
> ci.exp( mm )

                exp(Est.)        2.5%       97.5%
(Intercept)   0.002711497 0.001634669 0.004497678
factor(xpos)1 2.158979720 1.153159560 4.042106222
```

11. If we instead wanted the rate-difference as a comparative measure, we just subtract the rates, and the variance of the difference is (since the rates are based on independent samples) just the sum of the variances:

$$\begin{aligned}\text{var}(\log(\text{RD})) &= \text{var}(\lambda_1) + \text{var}(\lambda_0)\\ &= D_1/Y_1^2 + D_0/Y_0^2\end{aligned}$$

When we use this formula to compute the rate difference and a 95% confidence interval for it we get:

```
> rd <- diff( D/Y )
> sd <- sqrt( sum( D/Y^2 ) )
> c( rd, sd ) %*% ci.mat()

       Estimate         2.5%       97.5%
[1,] 0.00314257 0.0005765288 0.005708611
```

12. This is also the confidence interval we get when you fit an additive model with exposure as factor. Note that since the model is a model where the rates and rate differences are parameters, we shall *not* us the `Exp=TRUE` argument to `ci.lin`:

```
> ma <- glm( D/Y ~ factor(xpos), weight=Y,
+                  family=poisson(link=identity) )
> ci.exp( ma, Exp=FALSE )

                  Estimate         2.5%       97.5%
(Intercept)    0.002711497 0.0013393153 0.004083678
factor(xpos)1 0.003142570 0.0005765288 0.005708611
```

13. Normally one would like to get both the rates and the ratio between them. This can be achieved in one go using the `ctr.mat` argument to `ci.lin`:

```
> CM <- rbind( c(1,0), c(1,1), c(0,1) )
> rownames( CM ) <- c("rate 0","rate 1","RR 1 vs. 0")
> CM

           [,1] [,2]
rate 0        1    0
rate 1        1    1
RR 1 vs. 0    0    1

> mm <- glm( D ~ factor(xpos),
+                offset=log(Y), family=poisson )
> ci.exp( mm )

                 exp(Est.)        2.5%       97.5%
(Intercept)    0.002711497 0.001634669 0.004497678
factor(xpos)1 2.158979720 1.153159560 4.042106222

> round( ci.exp( mm, ctr.mat=CM ), 3 )

           exp(Est.)  2.5% 97.5%
rate 0         0.003 0.002 0.004
rate 1         0.006 0.004 0.008
RR 1 vs. 0     2.159 1.153 4.042
```

14. If we want the rates in units of cases per 1000, we just use `Y/1000` as the person time:

```
> mm <- glm( D ~ factor(xpos),
+                offset=log(Y/1000), family=poisson )
> ci.exp( mm, ctr.mat=CM )

           exp(Est.)     2.5%    97.5%
rate 0      2.711497 1.634669 4.497678
rate 1      5.854066 4.041994 8.478512
RR 1 vs. 0  2.158980 1.153160 4.042106

> round( ci.exp( mm, ctr.mat=CM ), 3 )

           exp(Est.)  2.5% 97.5%
rate 0         2.711 1.635 4.498
rate 1         5.854 4.042 8.479
RR 1 vs. 0     2.159 1.153 4.042
```

15. The same machinery can be used to the additive model to get the rates and the rate-difference in one go. We want the rates per 1000, so we rescale; also note that the corrected annotation of the resulting estimates are via the row-names of the contrast matrix (which otherwise is the same):

```
> rownames(CM)[3] <- "RD 1 vs. 0"
> ma <- glm( D/(Y/1000) ~ factor(xpos), weight=Y/1000,
+                  family=poisson(link=identity) )
> round( ci.exp( ma, ctr.mat=CM, Exp=FALSE ), 3 )

           Estimate  2.5% 97.5%
rate 0        2.711 1.339 4.084
rate 1        5.854 3.686 8.022
RD 1 vs. 0    3.143 0.577 5.709
```

Note there is a significant difference between rates because the lover bound of the rate difference is larger thn 0; there is no restriction that the rate difference should be positive. If we reverse the comparison we get:

```
> CM[3,] <- -CM[3,]
> rownames(CM)[3] <- "RD 0 vs. 1"
> round( ci.exp( ma, ctr.mat=CM, Exp=FALSE ), 3 )

           Estimate   2.5%  97.5%
rate 0        2.711  1.339  4.084
rate 1        5.854  3.686  8.022
RD 0 vs. 1   -3.143 -5.709 -0.577
```

3.7 Fitting a smooth curve

1. For illustration we fit a very crude model to the mortality follow-up of the men in the Danish Diabetes register:

```
library( Epi )
library( splines )
data( DMlate )
head( DMlate )
```

```
       sex    dobth     dodm    dodth    dooad doins      dox
50185    F 1940.256 1998.917       NA       NA    NA 2009.997
307563   M 1939.218 2003.309       NA 2007.446    NA 2009.997
294104   F 1918.301 2004.552       NA       NA    NA 2009.997
336439   F 1965.225 2009.261       NA       NA    NA 2009.997
245651   M 1932.877 2008.653       NA       NA    NA 2009.997
216824   F 1927.870 2007.886 2009.923       NA    NA 2009.923
```

2. Now define outcome and age and date of diagnosis for convenience, and restrict data to only men:

```
DMlate <- transform( DMlate, D = !is.na(dodth),
                             Y = dox-dodm,
                             A = dodm-dobth,
                             P = dodm )
DMlate <- subset( DMlate, Y>0 & sex=="M" )
str( DMlate )
```

```
'data.frame':        5183 obs. of  11 variables:
 $ sex  : Factor w/ 2 levels "M","F": 1 1 1 1 1 1 1 1 1 1 ...
 $ dobth: num  1939 1933 1946 1940 1933 ...
 $ dodm : num  2003 2009 2007 2007 2010 ...
 $ dodth: num  NA NA NA NA NA ...
 $ dooad: num  2007 NA 2007 2007 NA ...
 $ doins: num  NA NA NA NA NA ...
 $ dox  : num  2010 2010 2010 2010 2010 ...
 $ D    : logi  FALSE FALSE FALSE FALSE FALSE FALSE ...
 $ Y    : num  6.689 1.344 2.782 3.324 0.214 ...
 $ A    : num  64.1 75.8 60.7 66.2 76.6 ...
 $ P    : num  2003 2009 2007 2007 2010 ...
```

3. Now fit a model for mortality only depending on age and date at entry:

```
m0 <- glm( D ~ A + P, family=poisson, offset=log(Y), data=DMlate )
```

— and use `ci.lin` and `ci.exp` to explore the parameters:

```
ci.lin( m0 )
```

```
               Estimate       StdErr         z             P        2.5%        97.5%
(Intercept) 86.82024444 14.940861088  5.810926  6.212807e-09 57.53669481 116.10379407
A            0.07666534  0.002413784 31.761476 2.204730e-221  0.07193441   0.08139627
P           -0.04736773  0.007475160 -6.336684  2.347625e-10 -0.06201878  -0.03271669
```

```
round( ci.exp( m0 )[-1,], 3 )
```

```
  exp(Est.)  2.5% 97.5%
A     1.080 1.075 1.085
P     0.954 0.940 0.968
```

How much is mortality changing by age and how much by time?

4. How much older should a man be in order to double his mortality? This is really a digression, but the RR per year is:

```
ci.exp( m0 )[2,1]
```
```
[1] 1.079681
```

and the log RR per year is:

```
ci.lin( m0 )[2,1]
```
```
[1] 0.07666534
```

If a man grows x years older, his mortality will increase RR^x, so we shold solve the equation:

$$\text{RR}^x = 2 \quad \Leftrightarrow \quad x\log(\text{RR}) = \log(2) \quad \Leftrightarrow \quad x = \log(2)/\log(\text{RR})$$

so the doubling time is

```
log(2) / ci.lin( m0 )[2,1]
```
```
[1] 9.041207
```

thus the doubling time of mortality for diabetic men is about 9 years. If we want a confidence interva we can just do:

```
round( log(2) / ci.lin( m0 )[2,c(1,6,5)], 2 )
```
```
Estimate    97.5%     2.5%
    9.04     8.52     9.64
```

5. Now try to add a quadratic term in date of diagnosis to the model:

```
mq <- glm( D ~ A + P + I(P^2), family=poisson, offset=log(Y), data=DMlate )
round( ci.lin( mq ), 3 )
```
```
             Estimate   StdErr      z     P      2.5%     97.5%
(Intercept) 16425.926 7831.092  2.098 0.036 1077.268 31774.585
A               0.077    0.002 31.733 0.000    0.072     0.081
P             -16.375    7.826 -2.093 0.036  -31.714    -1.037
I(P^2)          0.004    0.002  2.086 0.037    0.000     0.008
```

What is the interpretation of the coefficients now (if any)?

6. Now try to make a graph of the RR of death relative to 2005, say: For a given time P the log-rate is

$$\mu + \alpha\mathtt{A} + \beta_1\mathtt{P} + beta_2\mathtt{P}^2$$

and for 2005 the log-rate is:

$$\mu + \alpha\mathtt{A} + \beta_1\mathsf{2005} + beta_2\mathsf{2005}^2$$

so the log-RR between these is:

$$\beta_1\mathtt{P} + beta_2\mathtt{P}^2 - \beta_1\mathsf{2005} - beta_2\mathsf{2005}^2 = \beta_1(\mathtt{P} - \mathsf{2005}) + \beta_2(\mathtt{P}^2 - \mathsf{2005}^2)$$

Now devise some points between 1995 and 2010 and construct the two columns of numbers to be multiplied by the two parameters:

```
P.pt <- 1995:2010
p1 <- P.pt - 2005
p2 <- P.pt^2 - 2005^2
coef( mq )
```

```
 (Intercept)             A             P         I(P^2)
1.642593e+04  7.660107e-02 -1.637542e+01  4.079240e-03
```

```
lRR <- coef(mq)[3] * p1 + coef(mq)[4] * p2
 RR <- exp( lRR )
plot( P.pt, RR, type="l", lwd=3, log="y" )
abline( h=1,v=2005)
```

What is the substantial conclusion from the shape of the RR relative to 2005?

7. Then we draw the RR relative to year 2000

```
P.pt <- 1995:2010
p1 <- P.pt - 2000
p2 <- P.pt^2 - 2000^2
coef( mq )
```

```
 (Intercept)             A             P         I(P^2)
1.642593e+04  7.660107e-02 -1.637542e+01  4.079240e-03
```

```
lRR <- coef(mq)[3] * p1 + coef(mq)[4] * p2
 RR <- exp( lRR )
plot( P.pt, RR, type="l", lwd=3, log="y" )
abline( h=1,v=2000)
```

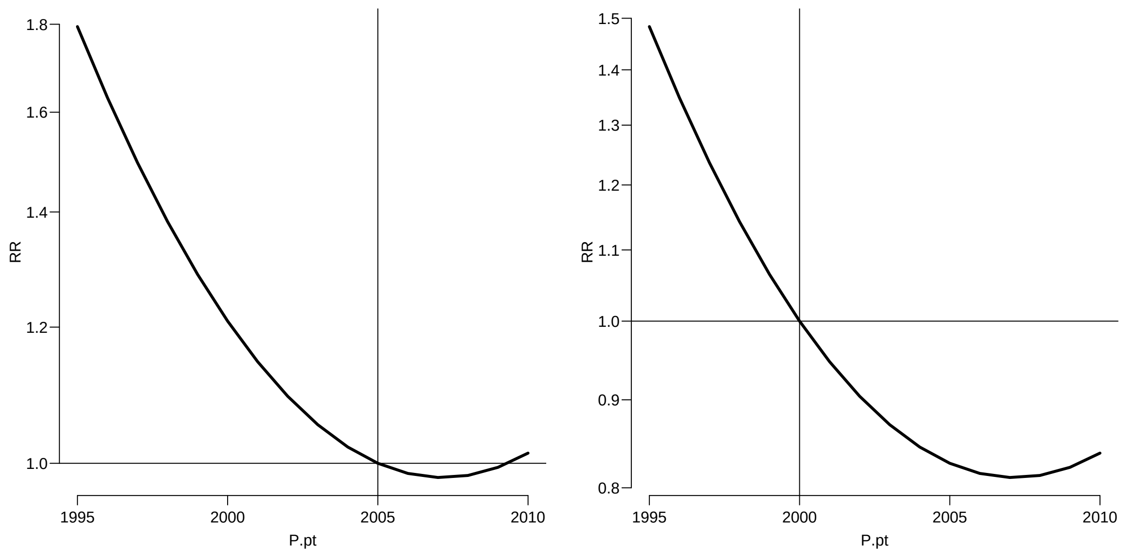

Figure 3.1: *RR relative to 2005 and 2000, respectively. The curves are identical except for the y-axis.*

8. These curves do not have confidence limits; but there is a facility in the functions `ci.lin` and `ci.exp`, that will both select the relevant parameters (in this case those with names `P` and `I(P^2)`):

```
ci.lin( mq, subset="P" )
```

```
              Estimate      StdErr         z          P           2.5%       97.5%
P       -16.37542295 7.825707708 -2.092517 0.03639233 -3.171353e+01 -1.03731769
I(P^2)    0.00407924 0.001955077  2.086486 0.03693467  2.473593e-04  0.00791112
```

```
ci.exp( mq, subset="P" )
```

```
          exp(Est.)        2.5%    97.5%
P      7.731151e-08 1.686513e-14 0.354404
I(P^2) 1.004088e+00 1.000247e+00 1.007942
```

There is a further argument to these functions, `ctr.mat` — a so called contrast matrix which is the columns of numbers we defined above to multiply by each of the parameters:

```
cbind( p1, p2 )
```

```
      p1     p2
 [1,] -5 -19975
 [2,] -4 -15984
 [3,] -3 -11991
 [4,] -2  -7996
 [5,] -1  -3999
 [6,]  0      0
 [7,]  1   4001
 [8,]  2   8004
 [9,]  3  12009
[10,]  4  16016
[11,]  5  20025
[12,]  6  24036
[13,]  7  28049
[14,]  8  32064
[15,]  9  36081
[16,] 10  40100
```

```
ci.exp( mq, subset="P", ctr.mat=cbind(p1,p2) )
```

```
      exp(Est.)      2.5%     97.5%
 [1,] 1.4833518 1.2578707 1.7492519
 [2,] 1.3486802 1.1982418 1.5180060
 [3,] 1.2362803 1.1425703 1.3376762
 [4,] 1.1425313 1.0908379 1.1966744
 [5,] 1.0645412 1.0431817 1.0863380
 [6,] 1.0000000 1.0000000 1.0000000
 [7,] 0.9470670 0.9322492 0.9621203
 [8,] 0.9042835 0.8783501 0.9309826
 [9,] 0.8705058 0.8339919 0.9086184
[10,] 0.8448545 0.7955757 0.8971858
[11,] 0.8266761 0.7606876 0.8983891
[12,] 0.8155151 0.7280788 0.9134517
[13,] 0.8110952 0.6972184 0.9435715
[14,] 0.8133076 0.6678925 0.9903827
[15,] 0.8222066 0.6400008 1.0562858
[16,] 0.8380122 0.6134772 1.1447279
```

9. The result is the estimated curve with confidence intervals, so we can now plot it (use the function `matplot`):

```
matplot( P.pt, ci.exp( mq, subset="P", ctr.mat=cbind(p1,p2) ),
         type="l", col="Black", lty=1, lwd=c(3,1,1), log="y" )
abline( h=1, v=2000 )
```

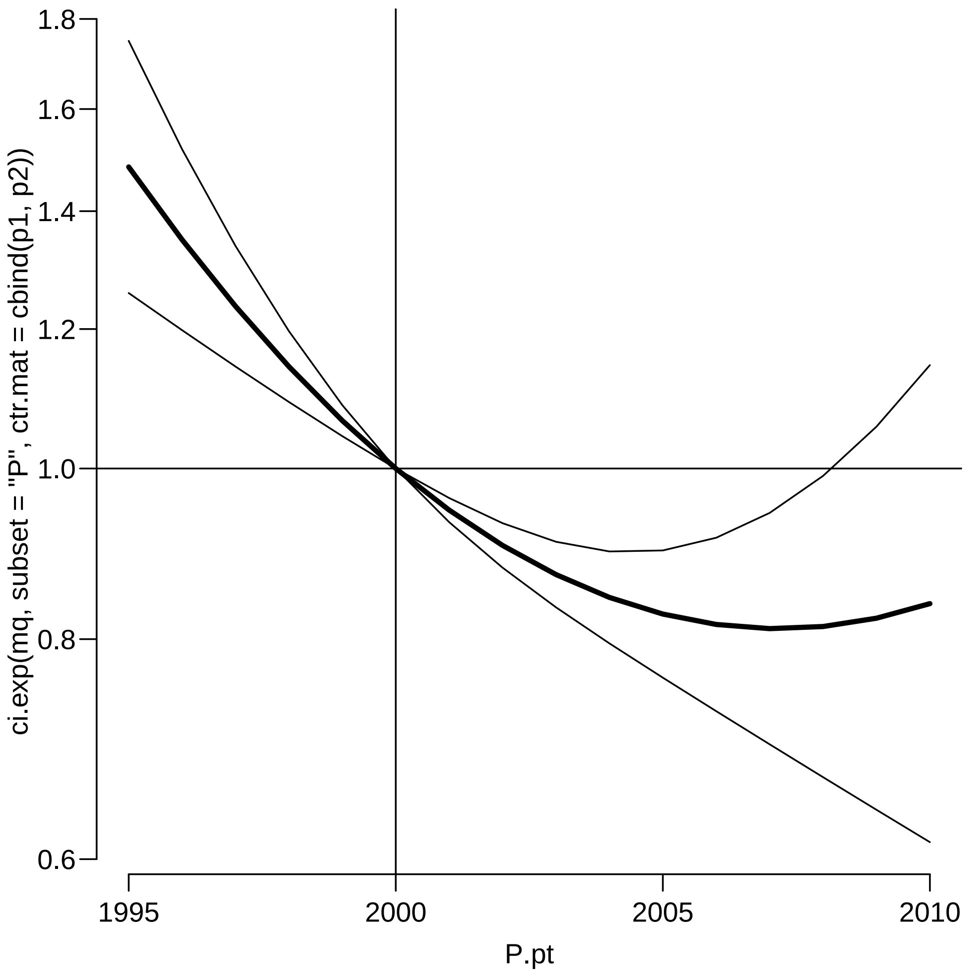

Figure 3.2: *RR by period with confidence intervals*

10. What really goes on here is that we as contrast matrix use the difference between the matrix of `P` and `P`^{2}, and the matrix that consists of the 2000-row all way through:

```
( MP <- cbind( P.pt, P.pt^2 ) )
```

```
      P.pt
 [1,] 1995 3980025
 [2,] 1996 3984016
 [3,] 1997 3988009
 [4,] 1998 3992004
 [5,] 1999 3996001
 [6,] 2000 4000000
 [7,] 2001 4004001
 [8,] 2002 4008004
 [9,] 2003 4012009
[10,] 2004 4016016
[11,] 2005 4020025
[12,] 2006 4024036
[13,] 2007 4028049
[14,] 2008 4032064
[15,] 2009 4036081
[16,] 2010 4040100
```

```
( Mr <- cbind( rep(2000,length(P.pt)), 2000^2 ) )
```

```
     [,1]  [,2]
 [1,] 2000 4e+06
 [2,] 2000 4e+06
 [3,] 2000 4e+06
 [4,] 2000 4e+06
 [5,] 2000 4e+06
 [6,] 2000 4e+06
 [7,] 2000 4e+06
 [8,] 2000 4e+06
 [9,] 2000 4e+06
[10,] 2000 4e+06
[11,] 2000 4e+06
[12,] 2000 4e+06
[13,] 2000 4e+06
[14,] 2000 4e+06
[15,] 2000 4e+06
[16,] 2000 4e+06
```

```
MP-Mr
```

```
     P.pt
 [1,]   -5 -19975
 [2,]   -4 -15984
 [3,]   -3 -11991
 [4,]   -2  -7996
 [5,]   -1  -3999
 [6,]    0      0
 [7,]    1   4001
 [8,]    2   8004
 [9,]    3  12009
[10,]    4  16016
[11,]    5  20025
[12,]    6  24036
[13,]    7  28049
[14,]    8  32064
[15,]    9  36081
[16,]   10  40100
```

```
ci.exp( mq, subset="P", ctr.mat=MP-Mr )
```

```
     exp(Est.)      2.5%     97.5%
 [1,] 1.4833518 1.2578707 1.7492519
 [2,] 1.3486802 1.1982418 1.5180060
 [3,] 1.2362803 1.1425703 1.3376762
 [4,] 1.1425313 1.0908379 1.1966744
 [5,] 1.0645412 1.0431817 1.0863380
 [6,] 1.0000000 1.0000000 1.0000000
 [7,] 0.9470670 0.9322492 0.9621203
 [8,] 0.9042835 0.8783501 0.9309826
 [9,] 0.8705058 0.8339919 0.9086184
[10,] 0.8448545 0.7955757 0.8971858
[11,] 0.8266761 0.7606876 0.8983891
[12,] 0.8155151 0.7280788 0.9134517
[13,] 0.8110952 0.6972184 0.9435715
[14,] 0.8133076 0.6678925 0.9903827
[15,] 0.8222066 0.6400008 1.0562858
[16,] 0.8380122 0.6134772 1.1447279
```

```
matplot( P.pt, ci.exp( mq, subset="P", ctr.mat=MP-Mr ),
         type="l", col="Black", lty=1, lwd=c(3,1,1), log="y" )
abline( h=1,v=2000)
```

11. If we want to model the period effect by a `cubic spline` instead. This is a function that is a cubic between a set of points called *knots*. In the `Epi` package is a function

Ns, that will generate a set of columns corresponding to this — think of it as the counterpart of the columns P and P^2:

```
p.kn <- c(1997,2000,2003,2006,2009)
Ns( P.pt, knots=p.kn )
```

```
                1           2          3           4
 [1,] 0.00000000  0.16903085 -0.5070926  0.33806170
 [2,] 0.00000000  0.08451543 -0.2535463  0.16903085
 [3,] 0.00000000  0.00000000  0.0000000  0.00000000
 [4,] 0.00617284 -0.08138522  0.2441557 -0.16277045
 [5,] 0.04938272 -0.14398924  0.4319677 -0.28797849
 [6,] 0.16666667 -0.16903085  0.5070926 -0.33806170
 [7,] 0.37037037 -0.13982242  0.4379858 -0.29199052
 [8,] 0.57407407 -0.04805065  0.2923001 -0.19486673
 [9,] 0.66666667  0.11250419  0.1624874 -0.10832495
[10,] 0.57407407  0.33051271  0.1195730 -0.07354249
[11,] 0.37037037  0.52444901  0.1488752 -0.04986741
[12,] 0.16666667  0.59523810  0.2142857  0.02380952
[13,] 0.04938272  0.47266314  0.2857143  0.19223986
[14,] 0.00617284  0.20194004  0.3571429  0.43474427
[15,] 0.00000000 -0.14285714  0.4285714  0.71428571
[16,] 0.00000000 -0.50000000  0.5000000  1.00000000
attr(,"degree")
[1] 3
attr(,"knots")
[1] 2000 2003 2006
attr(,"Boundary.knots")
[1] 1997 2009
attr(,"intercept")
[1] FALSE
attr(,"class")
[1] "ns"     "basis"  "matrix"
```

When we put this in a model, we get parameters that do not have any immediate meaning:

```
ms <- glm( D ~ A + Ns(P,knots=p.kn), family=poisson, offset=log(Y), data=DMlate )
ci.exp( ms )
```

```
                        exp(Est.)         2.5%        97.5%
(Intercept)           0.000430476 0.0003072398 0.0006031433
A                     1.079690213 1.0745850602 1.0848196186
Ns(P, knots = p.kn)1 0.752806894 0.5788409351 0.9790569131
Ns(P, knots = p.kn)2 0.629990707 0.4771921832 0.8317158257
Ns(P, knots = p.kn)3 0.530294086 0.4108359422 0.6844868924
Ns(P, knots = p.kn)4 0.830236504 0.5967627706 1.1550530398
```

```
ci.exp( ms, subset="P" )
```

```
                      exp(Est.)      2.5%     97.5%
Ns(P, knots = p.kn)1 0.7528069 0.5788409 0.9790569
Ns(P, knots = p.kn)2 0.6299907 0.4771922 0.8317158
Ns(P, knots = p.kn)3 0.5302941 0.4108359 0.6844869
Ns(P, knots = p.kn)4 0.8302365 0.5967628 1.1550530
```

12. But we extract the RR relative to 2005 from this new model as before, so we first construct the matrix for the points 1995—2010 and then for the reference for 2005:

```
CP <- Ns( P.pt, knots=p.kn )
Cr <- Ns( rep(2005,length(P.pt)), knots=p.kn )
CP-Cr
```

```
               1           2           3           4
 [1,] -0.3703704 -0.35541816 -0.65596775  0.38792912
 [2,] -0.3703704 -0.43993358 -0.40242147  0.21889827
 [3,] -0.3703704 -0.52444901 -0.14887520  0.04986741
 [4,] -0.3641975 -0.60583423  0.09528048 -0.11290303
 [5,] -0.3209877 -0.66843825  0.28309253 -0.23811107
 [6,] -0.2037037 -0.69347986  0.35821736 -0.28819429
 [7,]  0.0000000 -0.66427143  0.28911059 -0.24212311
 [8,]  0.2037037 -0.57249966  0.14342490 -0.14499932
 [9,]  0.2962963 -0.41194482  0.01361223 -0.05845753
[10,]  0.2037037 -0.19393630 -0.02930220 -0.02367508
[11,]  0.0000000  0.00000000  0.00000000  0.00000000
[12,] -0.2037037  0.07078909  0.06541052  0.07367694
[13,] -0.3209877 -0.05178587  0.13683909  0.24210727
[14,] -0.3641975 -0.32250897  0.20826766  0.48461168
[15,] -0.3703704 -0.66730615  0.27969623  0.76415313
[16,] -0.3703704 -1.02444901  0.35112480  1.04986741
attr(,"degree")
[1] 3
attr(,"knots")
[1] 2000 2003 2006
attr(,"Boundary.knots")
[1] 1997 2009
attr(,"intercept")
[1] FALSE
attr(,"class")
[1] "ns"     "basis"  "matrix"
```

Then we can extract the RR ad plot it:

```
RR <- ci.exp( ms, subset="P", ctr.mat=CP-Cr )
matplot( P.pt, RR,
         type="l", col="Black", lty=1, lwd=c(3,1,1), log="y" )
abline( h=1,v=2000)
```

13. Now we check if is any non-linearity by age (on the log-mortality scale, that is):

```
ma <- update( ms, . ~ . + I(A^2) )
round( ci.lin( ma ), 3 )
```

```
                     Estimate StdErr       z     P   2.5%  97.5%
(Intercept)            -7.504  0.650 -11.548 0.000 -8.778 -6.231
A                       0.069  0.019   3.574 0.000  0.031  0.107
Ns(P, knots = p.kn)1   -0.284  0.134  -2.117 0.034 -0.547 -0.021
Ns(P, knots = p.kn)2   -0.463  0.142  -3.265 0.001 -0.741 -0.185
Ns(P, knots = p.kn)3   -0.637  0.130  -4.884 0.000 -0.892 -0.381
Ns(P, knots = p.kn)4   -0.188  0.169  -1.115 0.265 -0.518  0.142
I(A^2)                  0.000  0.000   0.392 0.695  0.000  0.000
```

We see from the confidence interval there there is no evidence of non-linearity.

14. In order to report the model `ms` in full, we must show the estimated mortality rates as a function of age. For that purpose we of course must use the 2005 reference point.

    (a) The first possibility is to devise a prediction data frame:

    ```
    nd <- data.frame( A = 40:85, P=2005, Y=1000 )
    ```

    Note that you must provide values for *all* covariates, including the person-years, that enter in the model as offset, that is as a covariate with fixed regression coefficient. The function `ci.pred` is a simple convenience wrapper for `predict.glm`:

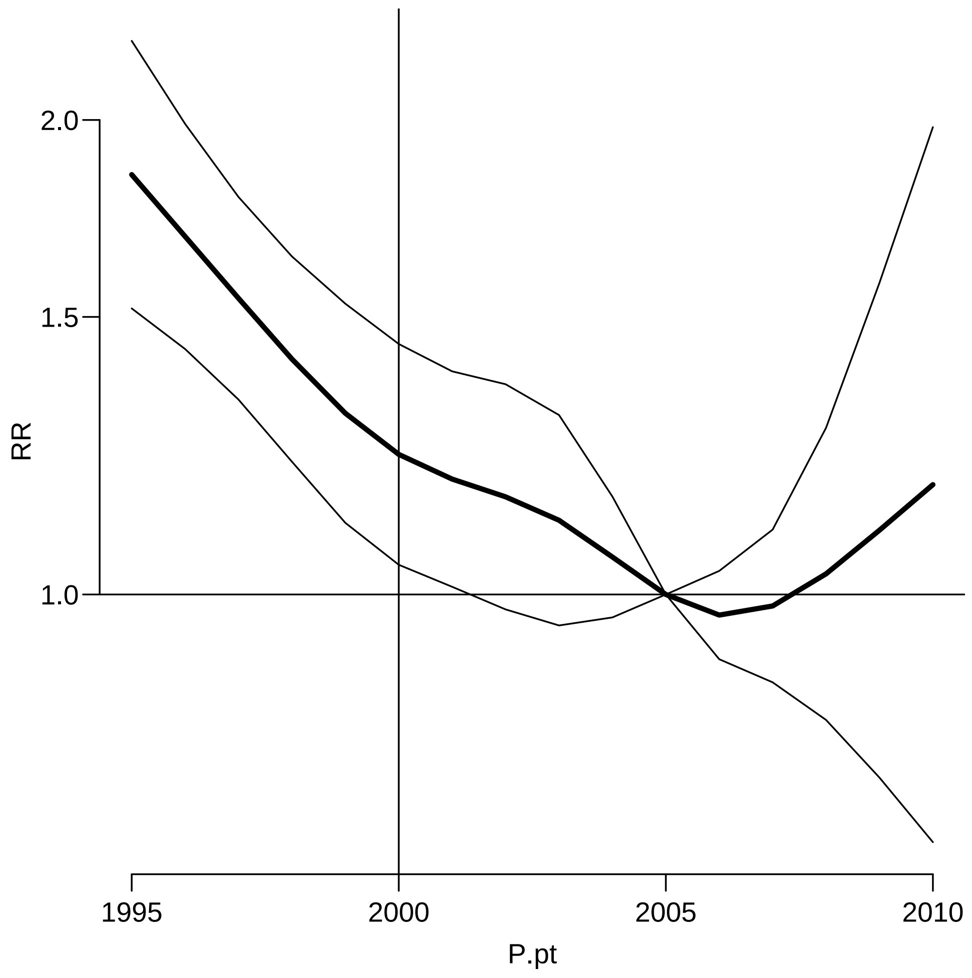

Figure 3.3: *RR by calendar time, modelled by a spline*

```
rate <- ci.pred( ma, newdata=nd )
matplot( nd$A, rate,
         type="l", col="Black", lty=1, lwd=c(3,1,1), log="y" )
```

Note that since you initially entered `Y` in units of 1 person-year, we get the rates in units of events per 1000 person-years by entering Y with the value of 1000 in the prediction frame.

(b) The other possibility is to use `ci.exp` directly to extract the predicted rates from the model, but they will be in he units of the `Y` entered into the model. Note that you in this case must be careful to get the order of the columns in `ctr.mat` right:

```
ci.exp( ma )
```

```
                          exp(Est.)         2.5%       97.5%
(Intercept)             0.000550656 0.0001540717 0.001968058
A                       1.071590052 1.0317148908 1.113006364
Ns(P, knots = p.kn)1 0.752908898 0.5789206188 0.979187458
Ns(P, knots = p.kn)2 0.629530651 0.4768295353 0.831133164
Ns(P, knots = p.kn)3 0.528963174 0.4096715676 0.682991112
Ns(P, knots = p.kn)4 0.828645308 0.5955280996 1.153015360
I(A^2)                  1.000055922 0.9997763008 1.000335621
```

```
Rate <- ci.exp( ma, ctr.mat=cbind(1,40:85,Ns(rep(2005,46),knots=p.kn),(40:85)^2) )*1000
matplot( nd$A, Rate,
         type="l", col="Black", lty=1, lwd=c(3,1,1), log="y" )
```

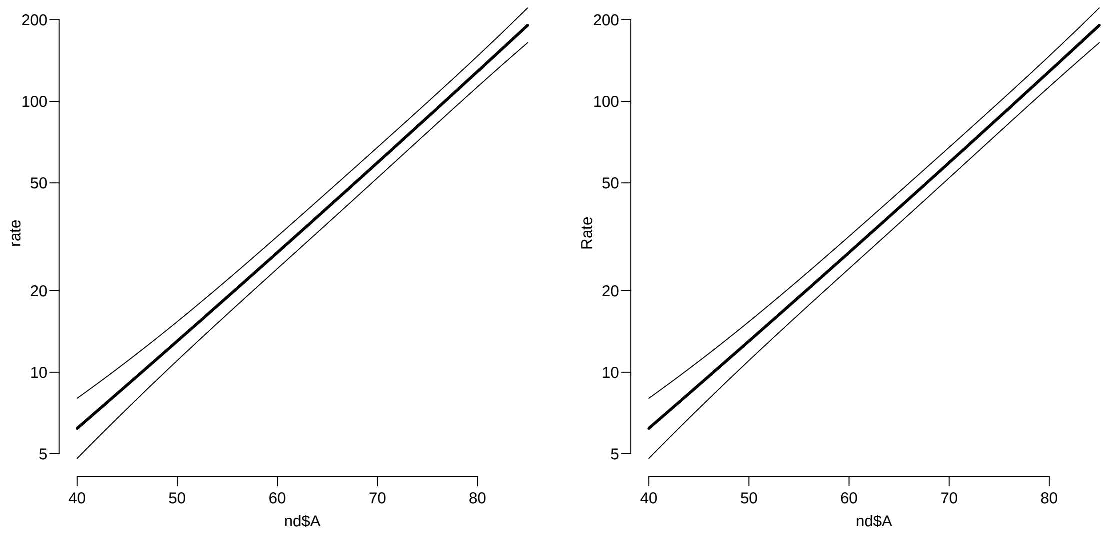

Figure 3.4: *Estimated age-specific mortality on 2005, by the two different approaches.*

15. Finally we explore if there are further non-linearities in the age-effect by including a spline with many knots

```
 ( a.kn <- 3:9*10 )

[1] 30 40 50 60 70 80 90

 mA <- update( ms, . ~ . - A + Ns(A,knots=a.kn) )
 round( ci.lin( mA ), 3 )

                        Estimate StdErr       z     P   2.5%  97.5%
(Intercept)               -5.439  0.305 -17.855 0.000 -6.036 -4.842
Ns(P, knots = p.kn)1      -0.275  0.134  -2.051 0.040 -0.538 -0.012
Ns(P, knots = p.kn)2      -0.460  0.142  -3.242 0.001 -0.738 -0.182
Ns(P, knots = p.kn)3      -0.624  0.131  -4.780 0.000 -0.880 -0.368
Ns(P, knots = p.kn)4      -0.178  0.169  -1.053 0.292 -0.508  0.153
Ns(A, knots = a.kn)1       1.758  0.385   4.564 0.000  1.003  2.513
Ns(A, knots = a.kn)2       2.092  0.325   6.432 0.000  1.454  2.729
Ns(A, knots = a.kn)3       3.103  0.328   9.452 0.000  2.460  3.747
Ns(A, knots = a.kn)4       3.667  0.267  13.749 0.000  3.144  4.190
Ns(A, knots = a.kn)5       4.879  0.573   8.517 0.000  3.756  6.002
Ns(A, knots = a.kn)6       4.171  0.265  15.748 0.000  3.652  4.690
```

We see that 7 knots produces a curve with 6 parameters.

```
 rate <- ci.pred( mA, newdata=nd )
 matplot( nd$A, rate,
          type="l", col="Black", lty=1, lwd=c(3,1,1), log="y" )
```

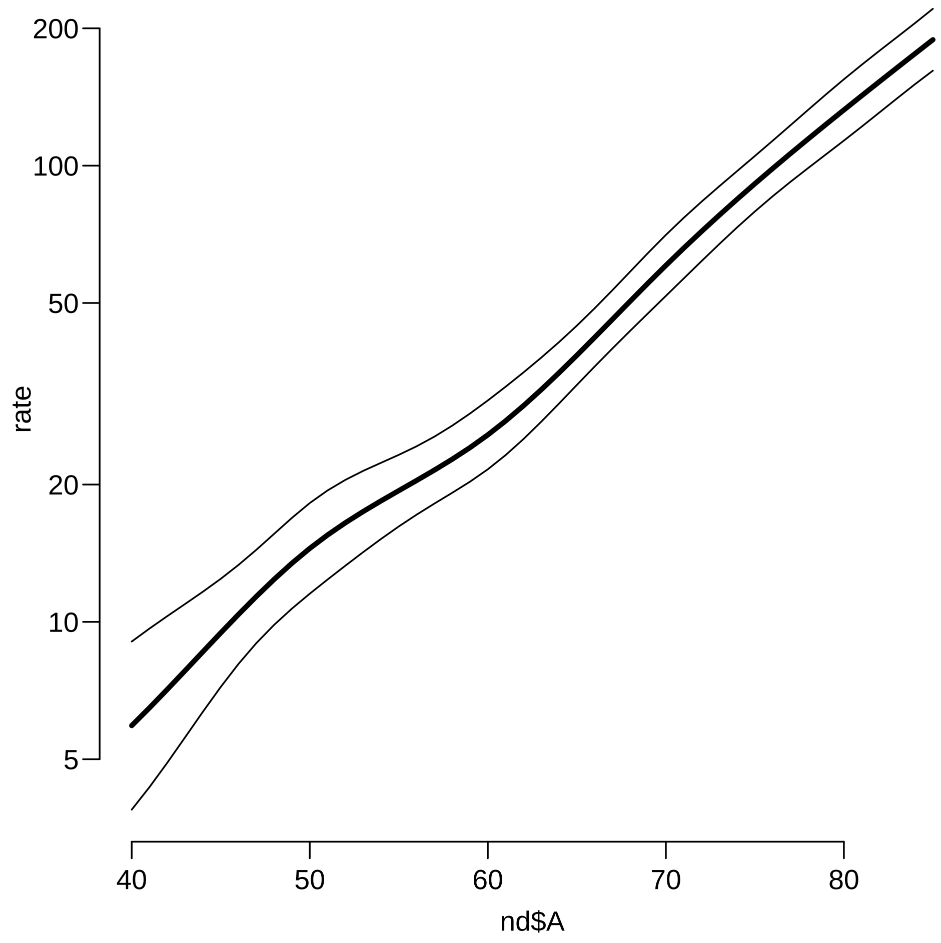

Figure 3.5: *Age-specific mortality among Danish DM men in 2005, using a 6-parameter natural spline.*

3.8 Cox and Poisson modelling

This practical is to show how results from a Cox-model can be reproduced exactly by a Poisson model, and in particular how more sensible and relevant results can be obtained from a Poisson model.

3.8.1 The lung cancer data

The data is the lung cancer data from the `survival` package which comes with R by default. We start by declaring a really large chunk of memory, because we need that to fit a silly model for illustration:

```
 memory.size( 3000 )
```

```
[1] Inf
```

```
 library( Epi )
 library( survival )
 sessionInfo()
```

```
R version 3.2.1 (2015-06-18)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 14.04.2 LTS

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C               LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=en_US.UTF-8     LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                  LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] utils     datasets  graphics  grDevices stats     methods   base

other attached packages:
[1] survival_2.38-3 Epi_1.1.69

loaded via a namespace (and not attached):
[1] cmprsk_2.2-7    MASS_7.3-42     parallel_3.2.1  etm_0.6-2       splines_3.2.1
[6] grid_3.2.1      lattice_0.20-31
```

Note that loading the `survival` package automatically also loads the `splines` package, which is also needed in the exercise.

1. First we load the `lung` data set and have a look at it:

   ```
   data( lung )
   str( lung )
   ```

   ```
   'data.frame':       228 obs. of  10 variables:
    $ inst     : num  3 3 3 5 1 12 7 11 1 7 ...
    $ time     : num  306 455 1010 210 883 ...
    $ status   : num  2 2 1 2 2 1 2 2 2 2 ...
    $ age      : num  74 68 56 57 60 74 68 71 53 61 ...
    $ sex      : num  1 1 1 1 1 1 2 2 1 1 ...
    $ ph.ecog  : num  1 0 0 1 0 1 2 2 1 2 ...
    $ ph.karno : num  90 90 90 90 100 50 70 60 70 70 ...
    $ pat.karno: num  100 90 90 60 90 80 60 80 80 70 ...
    $ meal.cal : num  1175 1225 NA 1150 NA ...
    $ wt.loss  : num  NA 15 15 11 0 0 10 1 16 34 ...
   ```

   ```
   lung[1:10,]
   ```

```
   inst time status age sex ph.ecog ph.karno pat.karno meal.cal wt.loss
1     3  306      2  74   1       1       90       100     1175      NA
2     3  455      2  68   1       0       90        90     1225      15
3     3 1010      1  56   1       0       90        90       NA      15
4     5  210      2  57   1       1       90        60     1150      11
5     1  883      2  60   1       0      100        90       NA       0
6    12 1022      1  74   1       1       50        80      513       0
7     7  310      2  68   2       2       70        60      384      10
8    11  361      2  71   2       2       60        80      538       1
9     1  218      2  53   1       1       70        80      825      16
10    7  166      2  61   1       2       70        70      271      34
```

2. The deaths are indicated by `status` being equal to 2, so we tabulate the number of records with different values of `status`:

   ```
   table( lung$status )
   ```

   ```
     1   2
    63 165
   ```

   — so we see there are 165 deaths.

3. Some of the recorded survival times are identical we see:

   ```
   addmargins( table( table( lung$time ) ) )
   ```

   ```
     1   2   3 Sum
   146  38   2 186
   ```

   In total there are 186 survival times.

3.8.2 Cox-models

4. Fitting a traditional Cox-model for the the Mayo clinic lung cancer data is done by `coxph`, where the response is a `Surv` object:

   ```
    system.time(
    m0.cox <- coxph( Surv( time, status==2 ) ~ age + factor( sex ),
                     method="breslow", eps=10^-8, iter.max=25, data=lung )
                )
   ```

   ```
      user  system elapsed
     0.008   0.000   0.008
   ```

   ```
    summary( m0.cox )
   ```

   ```
   Call:
   coxph(formula = Surv(time, status == 2) ~ age + factor(sex),
       data = lung, method = "breslow", eps = 10^-8, iter.max = 25)

     n= 228, number of events= 165

                      coef exp(coef)  se(coef)      z Pr(>|z|)
   age            0.017013  1.017158  0.009222  1.845  0.06506
   factor(sex)2 -0.512565  0.598957  0.167462 -3.061  0.00221

                exp(coef) exp(-coef) lower .95 upper .95
   age              1.017     0.9831    0.9989    1.0357
   factor(sex)2     0.599     1.6696    0.4314    0.8316
   ```

```
Concordance= 0.603  (se = 0.026 )
Rsquare= 0.06   (max possible= 0.999 )
Likelihood ratio test= 14.08  on 2 df,   p=0.0008741
Wald test            = 13.44  on 2 df,   p=0.001208
Score (logrank) test = 13.69  on 2 df,   p=0.001067
```

5. Now we create a Lexis object from the dataset

```
 Lung <- Lexis( exit = list( tfe=time ),
               exit.status = factor(status,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 tfe timescale.

 summary( Lung )

Transitions:
     To
From    Alive Dead  Records:  Events: Risk time:  Persons:
  Alive    63  165       228      165      69593       228
```

6. We can fit the same Cox-model to data using the formal structures of the `Lexis` object, and we see we get the same estimates:

```
 mL.cox <- coxph( Surv( tfe, tfe+lex.dur, lex.Xst=="Dead" ) ~
                 age + factor( sex ),
                 method="breslow", eps=10^-8, iter.max=25, data=Lung )
 cbind( coef(m0.cox), coef(mL.cox) )

                     [,1]        [,2]
age            0.01701289  0.01701289
factor(sex)2 -0.51256479 -0.51256479
```

3.8.3 Poisson models

7. Now we split data split in small intervals, in fact at all recorded survival times, which mean that all events occur at the end of an interval:

```
 Lung.s <- splitLexis( Lung,
                      breaks=c(0,sort(unique(Lung$time))),
                      time.scale="tfe" )
 summary( Lung.s )

Transitions:
     To
From    Alive Dead  Records:  Events: Risk time:  Persons:
  Alive 19857  165     20022      165      69593       228

 subset( Lung.s, lex.id==96 )

     lex.id tfe lex.dur lex.Cst lex.Xst inst time status age sex ph.ecog ph.karno
9235     96   0       5   Alive   Alive   12   30      2  72   1       2       80
9236     96   5       6   Alive   Alive   12   30      2  72   1       2       80
9237     96  11       1   Alive   Alive   12   30      2  72   1       2       80
9238     96  12       1   Alive   Alive   12   30      2  72   1       2       80
9239     96  13       2   Alive   Alive   12   30      2  72   1       2       80
9240     96  15      11   Alive   Alive   12   30      2  72   1       2       80
9241     96  26       4   Alive    Dead   12   30      2  72   1       2       80
```

```
     pat.karno meal.cal wt.loss
9235        60      288       7
9236        60      288       7
9237        60      288       7
9238        60      288       7
9239        60      288       7
9240        60      288       7
9241        60      288       7
```

8. We then fit the Cox model to the split dataset

```
system.time(
mLs.cox <- coxph( Surv( tfe, tfe+lex.dur, lex.Xst=="Dead" ) ~
                  age + factor( sex ),
                  method="breslow", eps=10^-8, iter.max=25, data=Lung.s )
            )

   user  system elapsed
  0.110   0.003   0.113
```

   ...and again we get exactly the same estimates

```
cbind( coef(m0.cox), coef(mL.cox), coef(mLs.cox) )

                     [,1]        [,2]        [,3]
age            0.01701289  0.01701289  0.01701289
factor(sex)2 -0.51256479 -0.51256479 -0.51256479
```

9. Then we fit a Poisson model with a factor accommodating the time-scale, in this case called `tfe`, which has exactly one level per recorded survival time:

```
nlevels( factor( Lung.s$tfe ) )

[1] 186
```

   But it involves fitting a model with 186+2 parameters, so it takes some time, and requires quite some memory, hence the memory allocation at start. Note that the response variable `lex.Xst=="Dead"` is a logical, but by R converted into a 0/1 numeric:

```
system.time(
mLs.pois.fc <- glm( lex.Xst=="Dead" ~ factor( tfe ) +
                            age + factor( sex ),
                            offset = log(lex.dur),
                    family=poisson, data=Lung.s, eps=10^-8, maxit=25 )
            )

   user  system elapsed
 14.421   0.024  14.439

length( coef(mLs.pois.fc) )

[1] 188
```

   So we have 188, parameters, but is only the last two that are of interest, and they are exactly the same as for the Cox-models:

```
 rbind( coef( m0.cox), coef( mLs.pois.fc )[188-1:0] )

           age factor(sex)2
[1,] 0.01701289   -0.5125648
[2,] 0.01701289   -0.5125648
```

10. Hence the Cox model in reality is a model for the rates that no one in their sane mind would fit, we would of course want to fit a model where the baseline hazard were modelled using the actual values of the time-scale, and devising it as a continuous function of time.

    So we define internal and boundary knots for the spline basis and fit the model with natural splines for the baseline. Using 5 knots gives us a restricted cubic spline (natural spline) basis with 4 parameters, not counting the intercept. Note that we are using the wrapper `Ns` from the Epi package to avoid the hassle of specifying the boundary and internal knots separately.

```
 t.kn <- c(0,25,100,500,1000)
 dim( Ns(Lung.s$tfe,knots=t.kn) )

[1] 20022     4
```

As opposed to the model with 188 parameters, this model only has 7, so it is very quickly fitted:

```
 system.time(
 mLs.pois.sp <- glm( lex.Xst=="Dead" ~ Ns( tfe, knots=t.kn ) +
                               age + factor( sex ),
                     offset = log(lex.dur),
                     family=poisson, data=Lung.s, eps=10^-8, maxit=25 )
             )

   user  system elapsed
  0.211   0.004   0.215

 ci.exp( mLs.pois.sp )

                           exp(Est.)         2.5%        97.5%
(Intercept)             0.0005600982 0.0001311645  0.002391729
Ns(tfe, knots = t.kn)1 2.5751960590 0.9916627245  6.687389350
Ns(tfe, knots = t.kn)2 2.6355488430 0.8560015677  8.114608625
Ns(tfe, knots = t.kn)3 3.2029000448 0.4769285447 21.509655507
Ns(tfe, knots = t.kn)4 3.1689618387 0.6843090390 14.675122733
age                     1.0161894486 0.9980328610  1.034676348
factor(sex)2            0.5998287489 0.4319932401  0.832870736

 ci.exp( mLs.pois.sp, subset=c("age","sex") )

             exp(Est.)      2.5%     97.5%
age          1.0161894 0.9980329 1.0346763
factor(sex)2 0.5998287 0.4319932 0.8328707
```

3.8.4 Comparing Cox and Poisson models

11. We can now compare the estimates of the regression parameters and their confidence intervals

```
ests <-
rbind( ci.exp(m0.cox),
       ci.exp(mLs.pois.fc,subset=c("age","sex")),
       ci.exp(mLs.pois.sp,subset=c("age","sex")) )
cmp <- cbind( ests[c(1,3,5)  ,],
              ests[c(1,3,5)+1,] )
rownames( cmp ) <- c("Cox","Poisson-factor","Poisson-spline")
colnames( cmp )[c(1,4)] <- c("age","sex")
round( cmp, 5 )
```

```
                   age    2.5%   97.5%     sex    2.5%   97.5%
Cox            1.01716 0.99894 1.03571 0.59896 0.43137 0.83165
Poisson-factor 1.01716 0.99894 1.03571 0.59896 0.43137 0.83165
Poisson-spline 1.01619 0.99803 1.03468 0.59983 0.43199 0.83287
```

We can also take a look at the estimated standard deviations of the log-RR:

```
round(
rbind( ci.lin(m0.cox)[,2],
       ci.lin(mLs.pois.fc,subset=c("age","sex"))[,2],
       ci.lin(mLs.pois.sp,subset=c("age","sex"))[,2] ), 6 )
```

```
          age factor(sex)2
[1,] 0.009222     0.167462
[2,] 0.009222     0.167462
[3,] 0.009199     0.167470
```

For all practical purposes they are the same too, so it is not so that the Cox-model or the factor-Poisson model inflates the s.e. of the regression estimates by estimating all the superfluous parameters.

12. We now use the parametrically estimated baseline intensity from the spline model to compute the estimated cumulative intensities over 100 10-day periods (0–1000 days after diagnosis) for men 60 year old at diagnosis, and then use these to compute the cumulative intensity since diagnosis and subsequently the survival function.

Now, in order to get the predictions from the spline model we need to devise the right contrast matrix because we need the covariance between the point estimates for log-incidence rates.

The model matrix, corresponding to times 0,10,20,...,1000:

```
CM <- cbind( 1, Ns( seq(0,1000,10), knots=t.kn ), 60, 1 )
```

The mortality rates at these time points, for a 60-year old man are then:

```
lambda <- ci.exp( mLs.pois.sp, ctr.mat=CM )
```

The cumulative mortality mortality rates (including the s.e.of this) are compute using `ci.cum`. Since this is a cumulative measure, we must explicitly supply the length of the intervals that each rate refer to, and for convenience we add

```
Lambda <- ci.cum( mLs.pois.sp, ctr.mat=CM, intl=10 )
Lambda <- rbind( 0, Lambda )
```

The Breslow-estimator of the survival curve for a male aged 60 Note that sex must be specified as a factor with two levels in the data frame in the argument `newdata`:

```
sf <-  survfit( m0.cox,
                newdata=data.frame( sex=factor(2,levels=1:2),
                                    age=c(60) ) )
```

13. We can then plot the mortality rates (`lambda`) and the survival function in two adjacent panels. Note that since we entered the risk time in days, the estimates of lambda we got out were rates *per day*, so we multiply them by 365.25 to the the mortality rates *per year* instead. Also note the we compute the survival function as $\exp(-\Lambda)$ on the fly, using the confidence intervals generated by `ci.cum` on the Λ-scale.

```
par( mfrow=c(1,2), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6, oma=c(0,0,0,0),
     las=1, bty="n" )
matplot( 0:100*10, lambda * 365.25,
         type="l", lwd=c(4,1,1), lty=1, col="black", log="y",
         xlim=c(0,900), xaxs="i", ylim=c(1/10,5),
         xlab="Days since diagnosis",
         ylab="Mortality rate per year")
# Then the survival curves by the two methods
# Here is the Breslow-estimator; note
plot( sf, lwd=c(4,1,1), col="red", conf.int=T, mark.time=F,
          xlab="Days since diagnosis",
          ylab="Survival", xlim=c(0,900), xaxs="i", lty=1)
matlines( 0:101*10, exp(-Lambda[,1:3]), lwd=c(4,1,1), col="black", lty=1 )
```

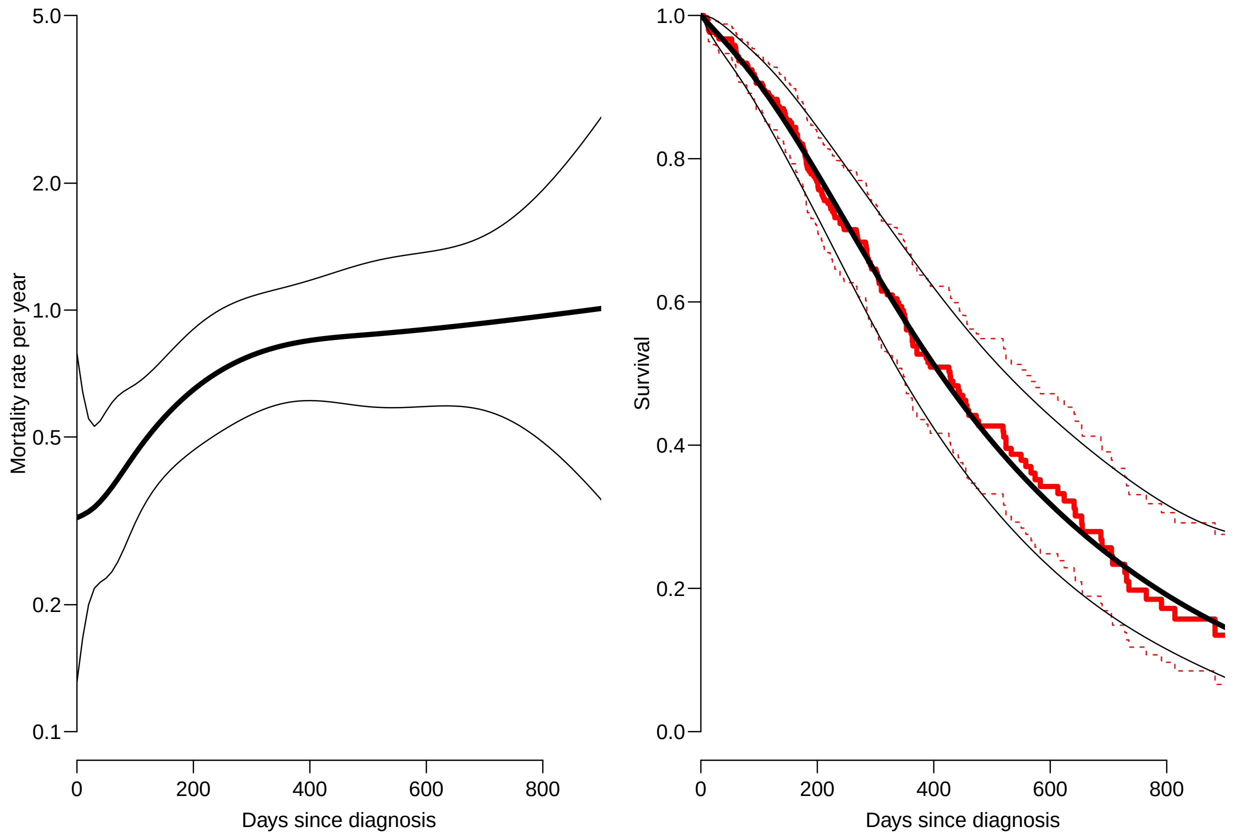

Figure 3.6: *Left: Hazard function for 60-year old men from spline model with 95% c.i. Right: Survival curve for 60 year old men; black from spline model, red from Cox-model.*

3.9 Diabetes in Denmark: Time-splitting, time-scales and SMR

This exercise is using data from the National Danish Diabetes register. There is a sample of 10,000 records from this in the `Epi` package. Actually there are two, we shall use the one with only cases of diabetes diagnosed after 1995. This is of interest because it is only for these where the data of diagnosis is certain, and hence for whom we can compute the duration of diabetes during follow-up.

The exercise is about assessing how mortality depends age, calendar time and duration of diabetes. And how to understand and compute SMR, and assess how it depends on these factors as well.

1. First, we load the `Epi` package and the dataset, and take a look at it:

```
> options( width=120 )
> library( Epi )
> clear()
> data( DMlate )
> str( DMlate )
> head( DMlate )
> summary( DMlate )
```

2. First we set up the dataset as a `Lexis` object with age, calendar time and duration of diabetes as timescales, and date of death as event.

   In the dataset we have a date of exit `dox` which is either the day of censoring or the date of death:

```
> with( DMlate, table( dead=!is.na(dodth),
+                      same=(dodth==dox), exclude=NULL ) )
```

   So we can set up the `Lexis` object by specifying the timescales `A`, `P` and `dur` and the exit status:

```
> LL <- Lexis( entry = list( A = dodm-dobth,
+                            P = dodm,
+                          dur = 0 ),
+               exit = list( P = dox ),
+        exit.status = factor( !is.na(dodth),
+                              labels=c("Alive","Dead") ),
+               data = DMlate )
```

   We can get an overview of the data by using `head` or the `summary` function on the object:

```
> head( LL )
> summary( LL )
```

3. A very crude picture of the mortality by sex can be obtained by the `stat.table` function:

```
> stat.table( sex,
+             list( D=sum( lex.Xst=="Dead" ),
+                   Y=sum( lex.dur ),
+                rate=ratio( lex.Xst=="Dead", lex.dur, 1000 ) ),
+             data=LL )
```

So not surprisingly, we see that men have a higher mortality than women.

4. We now want to assess how mortality depends on age, calendar time and duration. In principle we could split the follow-up along all three time scales, but in practice it would be sufficient to split it along one of the time-scales and then just use the value of each of the time-scales at the left endpoint of the intervals. This of course requires modeling of these as continuous variables or grouping of them prior to analysis.

   We note that the total follow-up time was some 54,000 person-years, so if we split the follow-up in 12-month intervals we get a bit more than 54,000 records:

```
> SL <- splitLexis( LL, breaks=seq(0,125,1), time.scale="A" )
> summary( SL )
```

5. With this in place we can start by making a crude age-specific mortality curve for men and women separately, using natural splines. But using these it requires that we define the knots — points on the age scale between which we have the curve as a 3rd degree polynomial. In this case we just take the knots reasonably evenly spread over the age-range:

```
> library( splines )
> a.kn <- seq(5,95,10)
> r.m <- glm( (lex.Xst=="Dead") ~ Ns( A, knots=a.kn, intercept=TRUE ) - 1,
+             offset = log( lex.dur ),
+             family = poisson,
+               data = subset( SL, sex=="M" ) )
> r.f <- update( r.m, data = subset( SL, sex=="F" ) )
```

   With these objects we can get the estimated log-rates by using `predict`, and supplying a data frame of prediction points, and finally use the wrapper `ci.pred` to get the rates with CIs:

```
> nd <-  data.frame( A = seq(10,90,0.5),
+               lex.dur = 1000)
> p.m <- ci.pred( r.m, newdata = nd )
> p.f <- ci.pred( r.f, newdata = nd )
```

   and then we can plot the two sets of estimated rates:

```
> matplot( seq(10,90,0.5), cbind(p.m,p.f),
+          type="l", lty=1, lwd=c(3,1,1), las=1,
+          col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(0.1,200),
+          xlab="Age", ylab="Mortality rates per 1000 PY" )
```

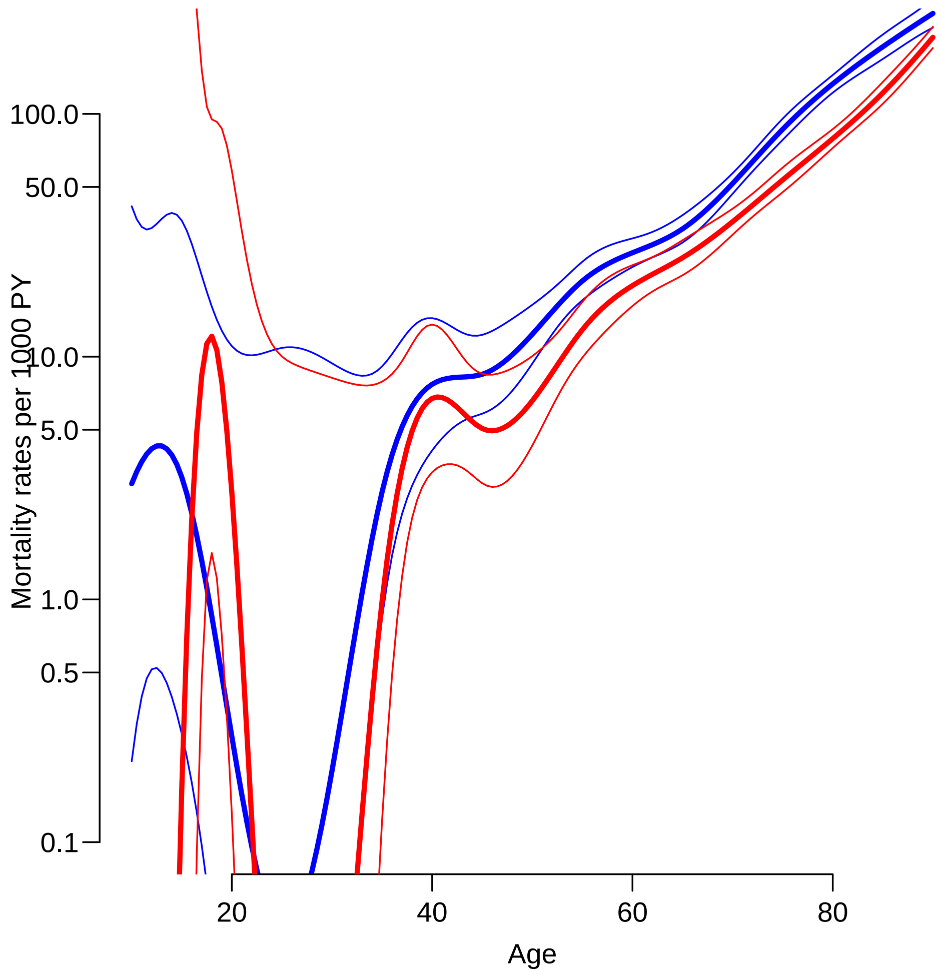

Figure 3.7: *Age-specific mortality rates for Danish diabetes patients as estimated from a model with only age. Blue: men, red: women.*

Graphical comparison with the population rates

6. We can compare with the mortality rates from the general population; they are available in the data frame `M.dk`

```
> data( M.dk )
> head( M.dk )
```

So we just plot the empirical mortality rates from 2005 on top of this:

```
> with( subset( M.dk, sex==1 & P==2005 ), lines( A, rate, col="blue", lty="12", lwd=3 ) )
> with( subset( M.dk, sex==2 & P==2005 ), lines( A, rate, col="red" , lty="12", lwd=3 ) )
```

7. It would however be more prudent to model these rates in a similar fashion as the diabetes mortality:

```
> R.m <- glm( D ~ ns( A, knots=a.kn ),
+             offset = log( Y ),
+             family = poisson,
```

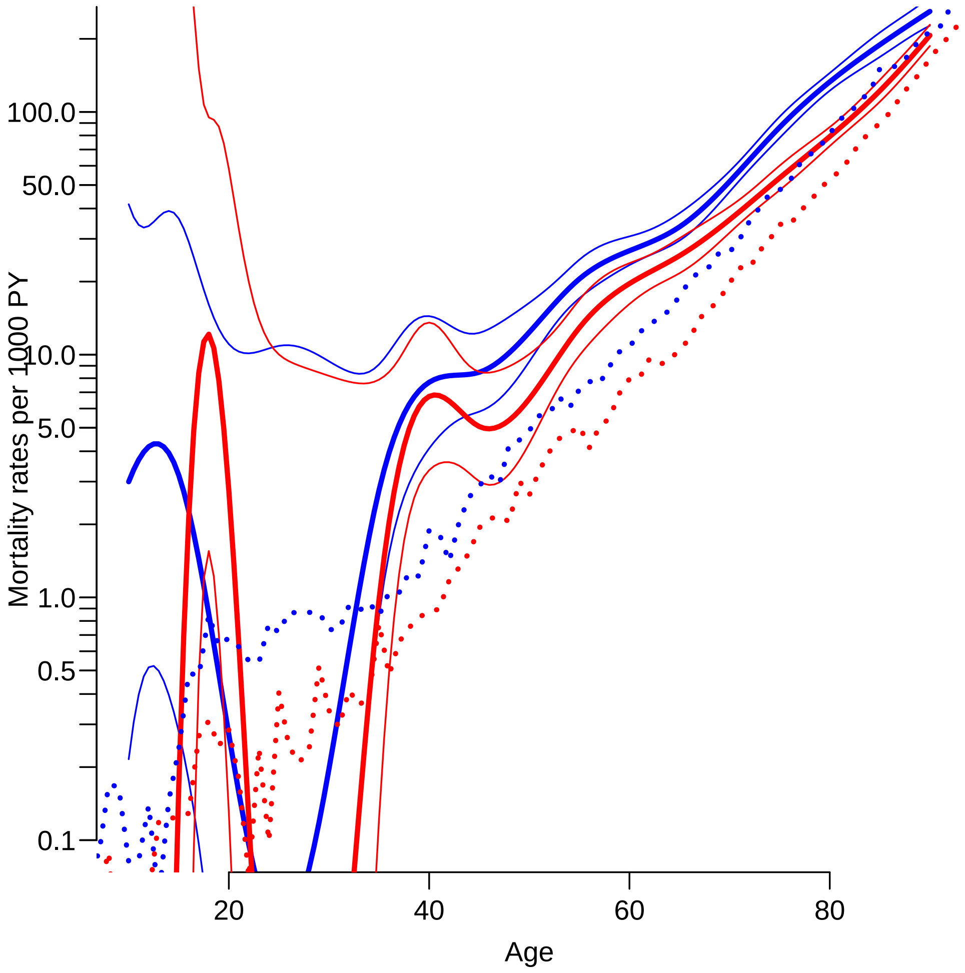

Figure 3.8: *Age-specific mortality rates for Danish diabetes patients as estimated from a model with only age. Broken lines are empirical rates from 2005. Blue: men, red: women.*

```
+                 data = subset( M.dk, sex==1 & P>1994 ) )
> R.f <- update( R.m, data = subset( M.dk, sex==2 & P>1994 ) )
> nd <-  data.frame( A = seq(10,90,0.5),
+                    Y = 1000)
> P.m <- ci.pred( R.m, newdata = nd )
> P.f <- ci.pred( R.f, newdata = nd )
```

Once we have the predicted rates from a smoothing model we can redo the plot with these overlaid:

```
> matplot( seq(10,90,0.5), cbind(p.m,p.f),
+          type="l", lty=1, lwd=c(3,1,1),
+          col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(0.1,200),
+          xlab="Age", ylab="Mortality rates per 1000 PY" )
> matlines( seq(10,90,0.5), cbind(P.m,P.f), lty="12",
+           col=c("blue","red"), lwd=c(3,0,0) )
```

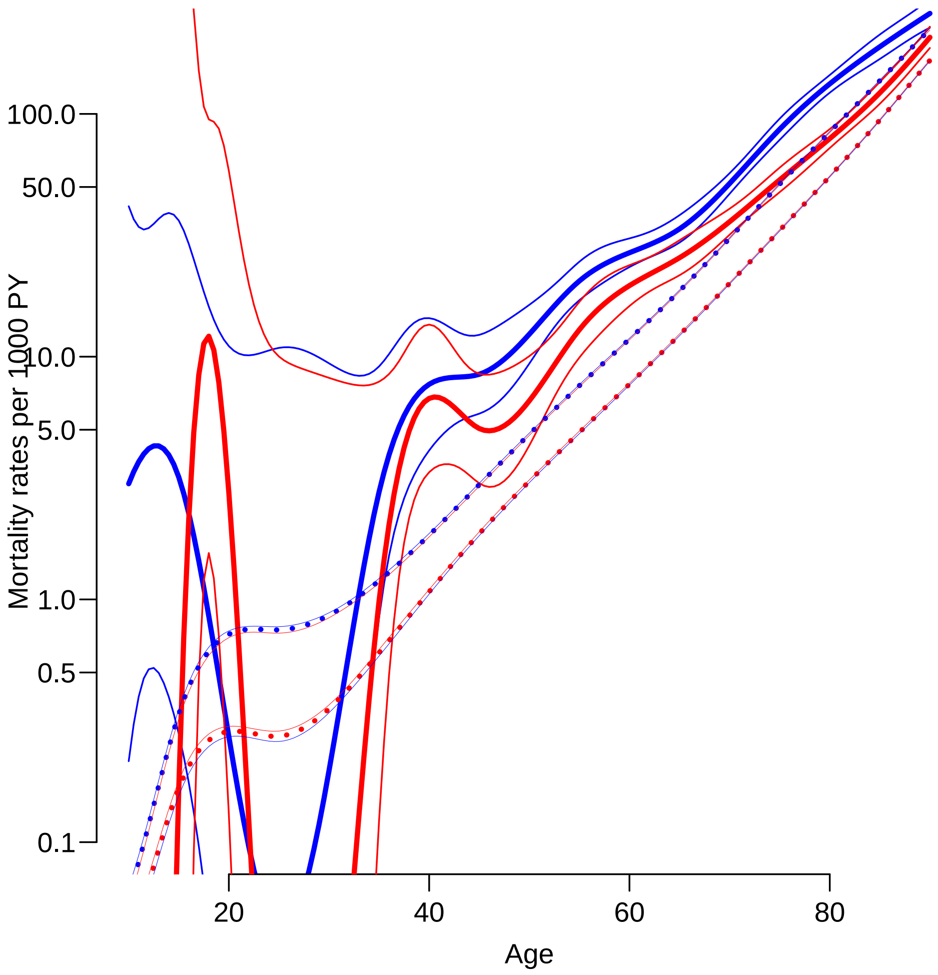

Figure 3.9: *Age-specific mortality rates for Danish diabetes patients as estimated from a model with only age. Broken lines are modeled population rates 1995–2010. Blue: men, red: women.*

Period and duration effects

8. We now want to model the mortality rates among diabetes patients also including current date and duration of diabetes. However, we shall not just use the positioning of knots for the splines as provided by `ns`, because this is based on the allocating knots so that the number of observations (lines in the dataset), is the same between knots. However the information in a follow-up study is in the number of events, so it would be better to allocate knots so that number of events were the same between knots.

   We will be using so-called *natural splines* that are linear beyond the boundary knots, and hence we take the 5th and 95th percentile of deaths as the boundary knots for age (`A`) and calendar time (`P`) but for duration where we actually have follow-up from tine 0 on the timescale we use 0 as the first knot.

   So we start out by placing knots so that the number of events is the same between each pair of knots (strictly speaking we should do this separately for men and women, but we pass on that one here):

```
> ( kn.A <- with( subset( SL, lex.Xst=="Dead" ),
+                 quantile( A+lex.dur, probs=seq(5,95,10)/100 ) ) )
> ( kn.P <- with( subset( SL, lex.Xst=="Dead" ),
+                 quantile( P+lex.dur, probs=seq(5,95,30)/100 ) ) )
> ( kn.dur <- c(0,with( subset( SL, lex.Xst=="Dead" ),
+                 quantile( dur+lex.dur, probs=seq(5,95,10)/100 ) ) ) )
```

9. With these we can now model mortality rates (separately for men and women), as functions of age, calendar time and duration:

```
> mm <- glm( (lex.Xst=="Dead") ~ Ns( A, kn=kn.A ) +
+                               Ns( P, kn=kn.P ) +
+                               Ns( dur, kn=kn.dur ),
+            offset = log( lex.dur ),
+            family = poisson,
+              data = subset( SL, sex=="M" ) )
> summary( mm )
> mf <- update( mm, data = subset( SL, sex=="F" ) )
```

10. These models fit substantially better than the model with only age as we can see from this comparison:

```
> anova( mm, r.m, test="Chisq" )
> anova( mf, r.f, test="Chisq" )
```

The models are not formally nested since the location of the knots are different, so from a formal point of view these test are not valid, but is is clear that the more extensive modeling provides a much better description of the rates.

11. The model fitted separately for men and women has three terms: age (`A`), calendar time (`P`) and diabetes duration (`dur`). Since the outcome is a rate with dimension time^{-1} we must put the rate dimension on one of these terms and leave the two others as rate-ratios. In order to do this we must fix reference values for the two rate-ratio terms. The natural variable for the rate-dimension is age, so that we get estimated age-specific rate-ratios for a specific calendar time, 1.1.2008, say, and a specific duration of diabetes, 2 years, say.

In order to extract these terms from the model we need contrast matrices, that is matrices where each row corresponds to a set of values for age or period or duration, and the columns correspond to the columns in the spline basis as used in the model *i.e.* the parameters.

This is one reason for explicitly fixing the knots in the spline definitions; when we extract the effects we *must* use the same set of knots as in the model specification in order to get the right predictions.

We will need matrices for specified set of values for age, calendar time and duration, but also matrices where all rows refer to the chosen reference values for calendar time and duration.

We begin by specifying the prediction points for the time scales and the reference points. There is formally no reason to require that the matrices all have the same number of rows, but it makes the handling of the reference points much easier.

```
> N <- 100
> pr.A <- seq(10,90,,N)
> pr.P <- seq(1995,2010,,N)
> pr.d <- seq(0,15,,N)
> rf.P <- 2009
> rf.d <- 2
```

With these in place we generate the matrices we shall multiply to the parameter estimates:

```
> AC <- Ns( pr.A, knots=kn.A )
> PC <- Ns( pr.P, knots=kn.P )
> dC <- Ns( pr.d, knots=kn.dur )
> PR <- Ns( rep(rf.P,N), knots=kn.P )
> dR <- Ns( rep(rf.d,N), knots=kn.dur )
```

Note that the rows of `AC` refer to `N` points on the age-scale, `PC` to N points on the calendar time scale, etc.

These matrices are the necessary input for extracting the effects; this is done by the function `ci.exp` — remember to take a look at the help page for this.

Note that we make use of *all* parameters when extracting the age-effect — this is the effect where we have the dimension of the response (rate), and hence the intercept, and where we have fixed the values of date and duration at their reference values.

The rate-ratios for calendar time and duration are estimated exclusively from the parameters for these terms, but note that we subtract the values at the reference point:

```
> m.A <- ci.exp( mm, ctr.mat=cbind(1,AC,PR,dR) ) * 1000
> f.A <- ci.exp( mf, ctr.mat=cbind(1,AC,PR,dR) ) * 1000
> m.P <- ci.exp( mm, subset="P"  , ctr.mat=PC-PR )
> f.P <- ci.exp( mf, subset="P"  , ctr.mat=PC-PR )
> m.d <- ci.exp( mm, subset="dur", ctr.mat=dC-dR )
> f.d <- ci.exp( mf, subset="dur", ctr.mat=dC-dR )
```

12. We now plot the three effects in three panels beside each other:

```
> par( mfrow=c(1,3), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> matplot( pr.A, cbind(m.A,f.A),
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", xlab="Age", ylab="Mortality rate per 1000 PY" )
> matplot( pr.P,  cbind(m.P,f.P),
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", xlab="Date of follow-up", ylab="Mortality rate ratio" )
> matplot( pr.d,  cbind(m.d,f.d),
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", xlab="Diabetes duration", ylab="Mortality rate ratio" )
```

Figure **??** clearly shows that the duration effect is grossly over-modeled, and that the rate-ratios have a much smaller variability than the mortality rates.

Moreover the y-axis for mortality rates should be from about 0.1 to 200, and the y-axes for the rate-ratios should be on approximately the same scale. To make the RR-axes symmetric, from 1/30 to 30, that is a factor $30^2 = 900$, and the the rate-axis from 0.2 to 180.

So we redefine the duration knots, refit the models, re-extract parameters and plot using pre-specified axis ranges:

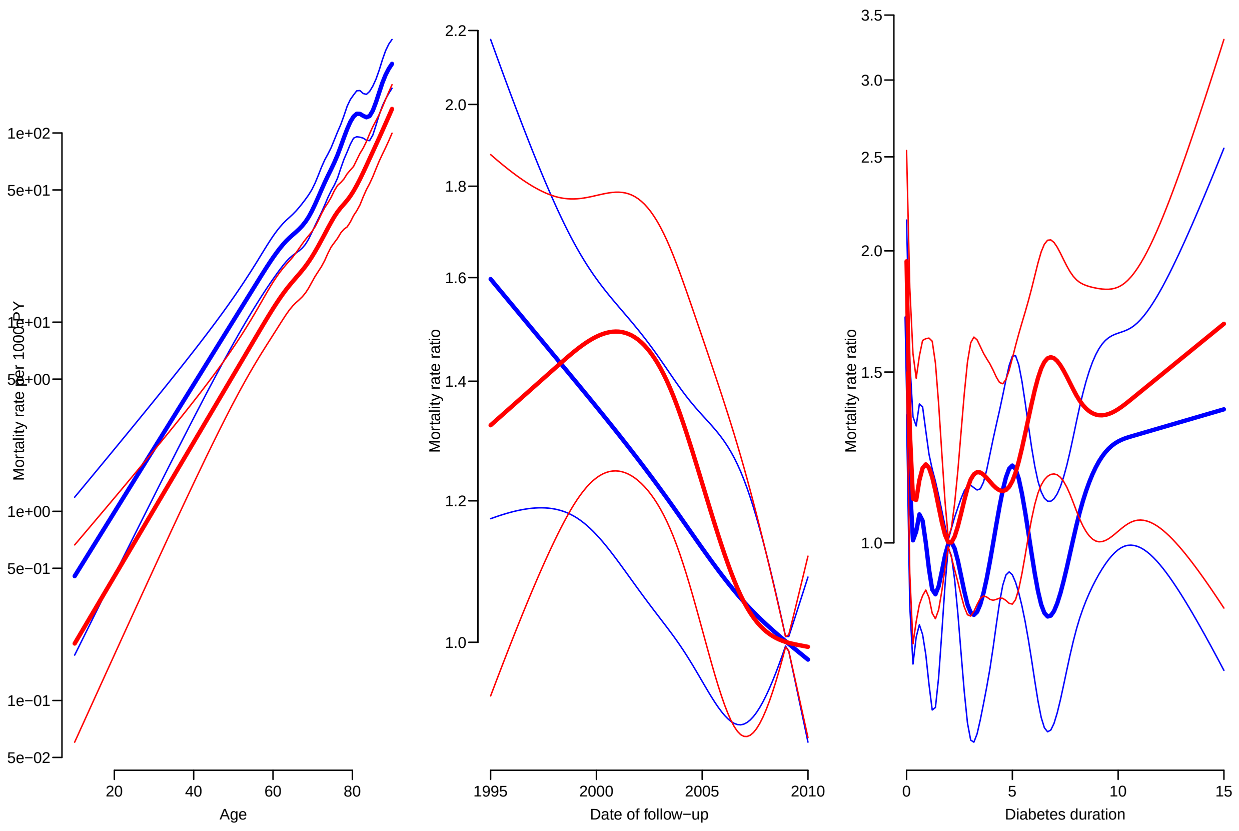

Figure 3.10: *Estimates from model for mortality of Danish diabetes patients. The duration is modeled with 10 parameters, which is clearly way too much.*

```
> kn.dur <- c(0,with( subset( SL, lex.Xst=="Dead" ),
+                quantile( dur+lex.dur, probs=seq(5,95,30)/100 ) ))
> dC <- Ns( pr.d, knots=kn.dur )
> dR <- Ns( rep(rf.d,N), knots=kn.dur )
> mm <- glm( (lex.Xst=="Dead") ~ Ns( A, kn=kn.A ) +
+                                Ns( P, kn=kn.P ) +
+                                Ns( dur, kn=kn.dur ),
+             offset = log( lex.dur ),
+             family = poisson,
+               data = subset( SL, sex=="M" ) )
> mf <- update( mm, data = subset( SL, sex=="F" ) )
> m.A <- ci.exp( mm, ctr.mat=cbind(1,AC,PR,dR) ) * 1000
> f.A <- ci.exp( mf, ctr.mat=cbind(1,AC,PR,dR) ) * 1000
> m.P <- ci.exp( mm, subset="P"   , ctr.mat=PC-PR )
> f.P <- ci.exp( mf, subset="P"   , ctr.mat=PC-PR )
> m.d <- ci.exp( mm, subset="dur", ctr.mat=dC-dR )
> f.d <- ci.exp( mf, subset="dur", ctr.mat=dC-dR )
> par( mfrow=c(1,3), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> matplot( pr.A, cbind(m.A,f.A),
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(0.2,180),
+          xlab="Age", ylab="Mortality rate per 1000 PY" )
> matplot( pr.P,  cbind(m.P,f.P),
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(1/30,30),
+          xlab="Date of follow-up", ylab="Mortality rate ratio" )
> abline( h=1 )
> matplot( pr.d,  cbind(m.d,f.d),
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(1/30,30),
+          xlab="Diabetes duration", ylab="Mortality rate ratio" )
> abline( h=1 )
```

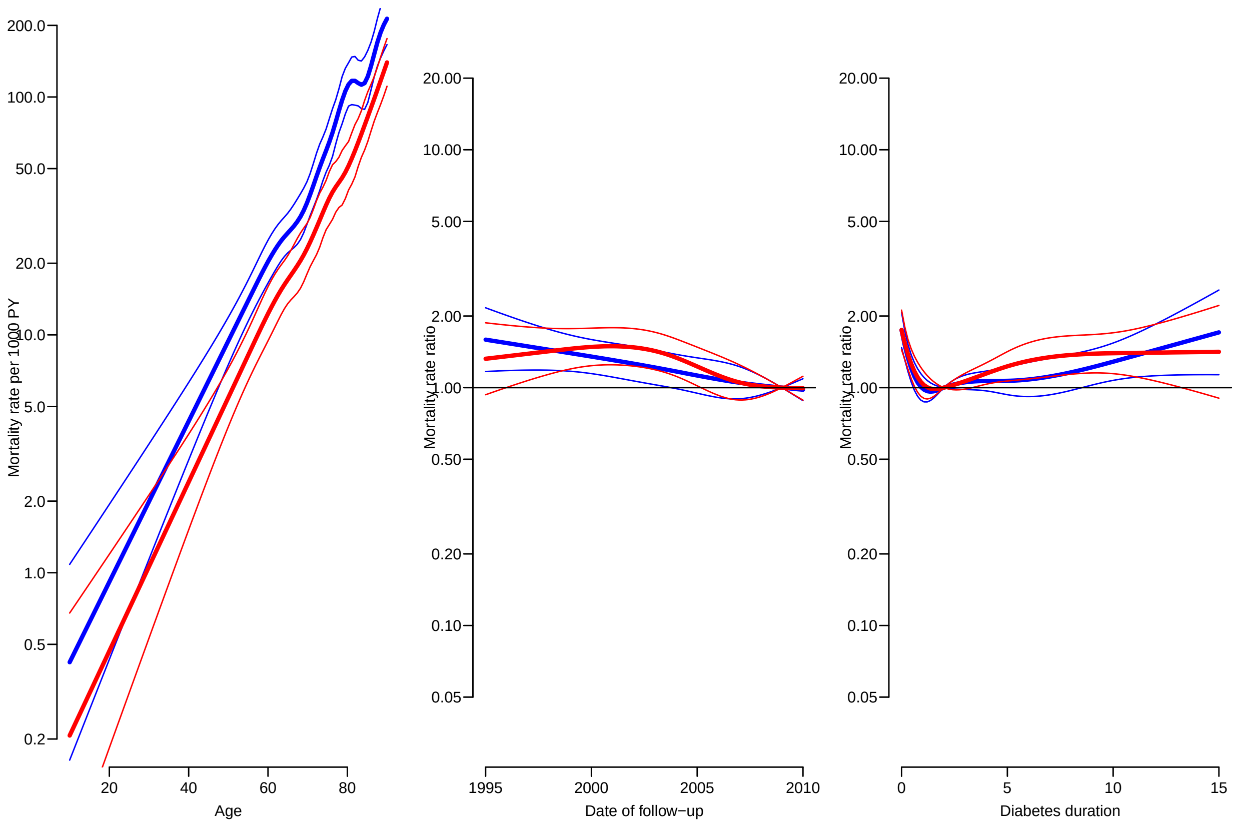

Figure 3.11: *Estimates from the model for mortality of Danish diabetes patients with only 5 knots (corresponding to 4 parameters) for duration.*

We might argue that we do not need the same scale for the y-axes for rates and RRs:

```
> par( mfrow=c(1,3), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> matplot( pr.A, cbind(m.A,f.A),
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(0.2,180),
+          xlab="Age", ylab="Mortality rate per 1000 PY" )
> matplot( pr.P,  cbind(m.P,f.P),
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(1/3,3),
+          xlab="Date of follow-up", ylab="Mortality rate ratio" )
> abline( h=1 )
> matplot( pr.d,  cbind(m.d,f.d),
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(1/3,3),
+          xlab="Diabetes duration", ylab="Mortality rate ratio" )
> abline( h=1 )
```

13. We have so far fitted models separately for men and women, but judging from the display of the parameters in figure ??, the period and duration effects are the same, so we might fit a model for the entire dataset with common period and duration effects, but different age-effect for the two sexes:

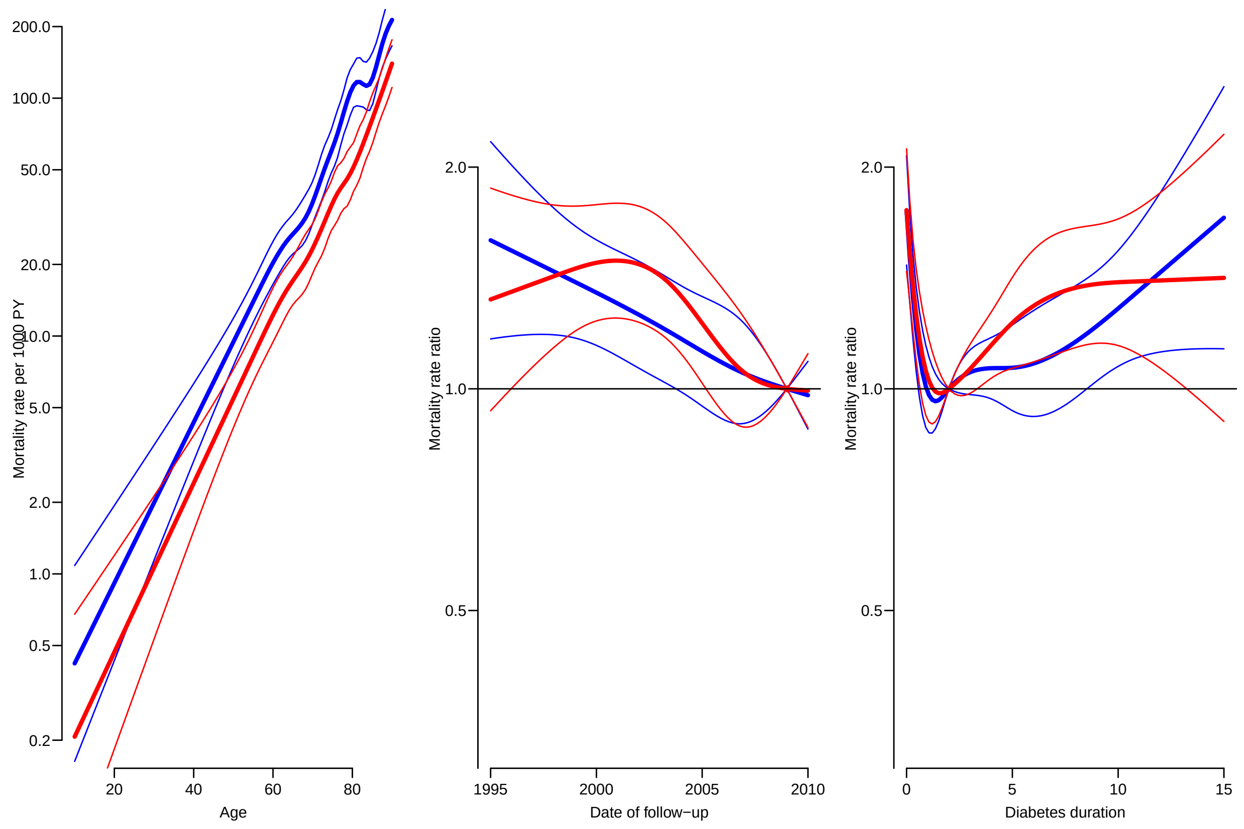

Figure 3.12: *Estimates from model for mortality of Danish diabetes patients.*

```
> m2 <- glm( (lex.Xst=="Dead") ~ sex +
+                              sex:Ns( A, kn=kn.A ) +
+                                  Ns( P, kn=kn.P ) +
+                                  Ns( dur, kn=kn.dur ),
+             offset = log( lex.dur ),
+             family = poisson,
+               data = SL )
> ci.exp(m2)
```

14. We can formally test this model against the separate models; the deviance and degrees of freedom from the separate models for men and women add up to that of a joint model with interaction between all terms and sex. Note that we add 1 to the degrees of freedom for the joint model; this is because the degrees of freedom is equal to the number of parameters *minus 1*, so the sum of the degrees of freedom from the two models is 1 too small — loosely speaking the intercepts from the two separate models correspond to the overall intercept and the main effect of sex in a joint model, and the sex parameter should be counted too.

```
> j.dev <- mm$dev + mf$dev
> j.df  <- mm$df.r + mf$df.r + 1
> 1 - pchisq( m2$dev - j.dev, m2$df.r - j.df )
```

So there is indeed no evidence of different period and duration effects.

15. We might from a purely technical point of view contemplate a model where the difference in age-specific mortality between men and women were either constant or

exponentially increasing or decreasing by age. And we might even accept a model of that sort by a statistical test, but given the different biology of men and women over their life span, it would make little sense. And therefore we have not done it here.

16. We can now extract the parameters from the model. Note that the sequence (and hence meaning) of the parameters depend on how the model is specified. The age-specific rates for men and women at the reference time and reference duration will need parameters extracted by the following subset-argument to `ci.exp`:

```
> ci.exp( m2, subset=c("Int","sexM","P","dur") )
> ci.exp( m2, subset=c("Int","sexF","P","dur") )
```

Note that the two subsets of parameters have different length; the parameters for the women (sex="F") has one more column:

```
> mi.A <- ci.exp( m2, subset=c("Int","sexM","P","dur"), ctr.mat=cbind(1  ,AC,PR,dR) ) * 1000
> fi.A <- ci.exp( m2, subset=c("Int","sexF","P","dur"), ctr.mat=cbind(1,1,AC,PR,dR) ) * 1000
> b.P  <- ci.exp( m2, subset="P"   , ctr.mat=PC-PR )
> b.d  <- ci.exp( m2, subset="dur", ctr.mat=dC-dR )
```

```
> par( mfrow=c(1,3), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> matplot( pr.A, cbind(m.A,f.A,mi.A,fi.A),
+          type="l", lty=rep(c(3,1),each=6), lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(0.2,180),
+          xlab="Age", ylab="Mortality rate per 1000 PY" )
> matplot( pr.P,  cbind(m.P,f.P,b.P),
+          type="l", lty=rep(c(3,1),c(6,3)), lwd=c(3,1,1), col=rep(c("blue","red","black"),eac
+          log="y", ylim=c(1/3,3),
+          xlab="Date of follow-up", ylab="Mortality rate ratio" )
> abline( h=1 )
> matplot( pr.d,  cbind(m.d,f.d,b.d),
+          type="l", lty=rep(c(3,1),c(6,3)), lwd=c(3,1,1), col=rep(c("blue","red","black"),eac
+          log="y", ylim=c(1/3,3),
+          xlab="Diabetes duration", ylab="Mortality rate ratio" )
> abline( h=1 )
```

We shall return to the set-up with separate effects for men and women.

17. The model we fitted has three time-scales: current age, current date and current duration of diabetes, so the effects that we report are not immediately interpretable, as they are (as in all multiple regression) to be interpreted as "all else equal" which they are not, as the three time scales advance by the same pace.

The reporting would therefore more naturally be *only* on the mortality scale, but showing the mortality for persons diagnosed in different ages, using separate displays for separate years of diagnosis.

Incidentally, this is most easily done using the `ci.pred` function with the `newdata=` argument. So a person diagnosed in age 50 will have a (log-)mortality measure in cases per 1000 PY as:

```
> pts <- seq(0,20,1)
> nd <- data.frame( A=  50+pts,
+                   P=1995+pts,
+                 dur=     pts,
+             lex.dur=1000 )
> ci.pred( mm, newdata=nd )
```

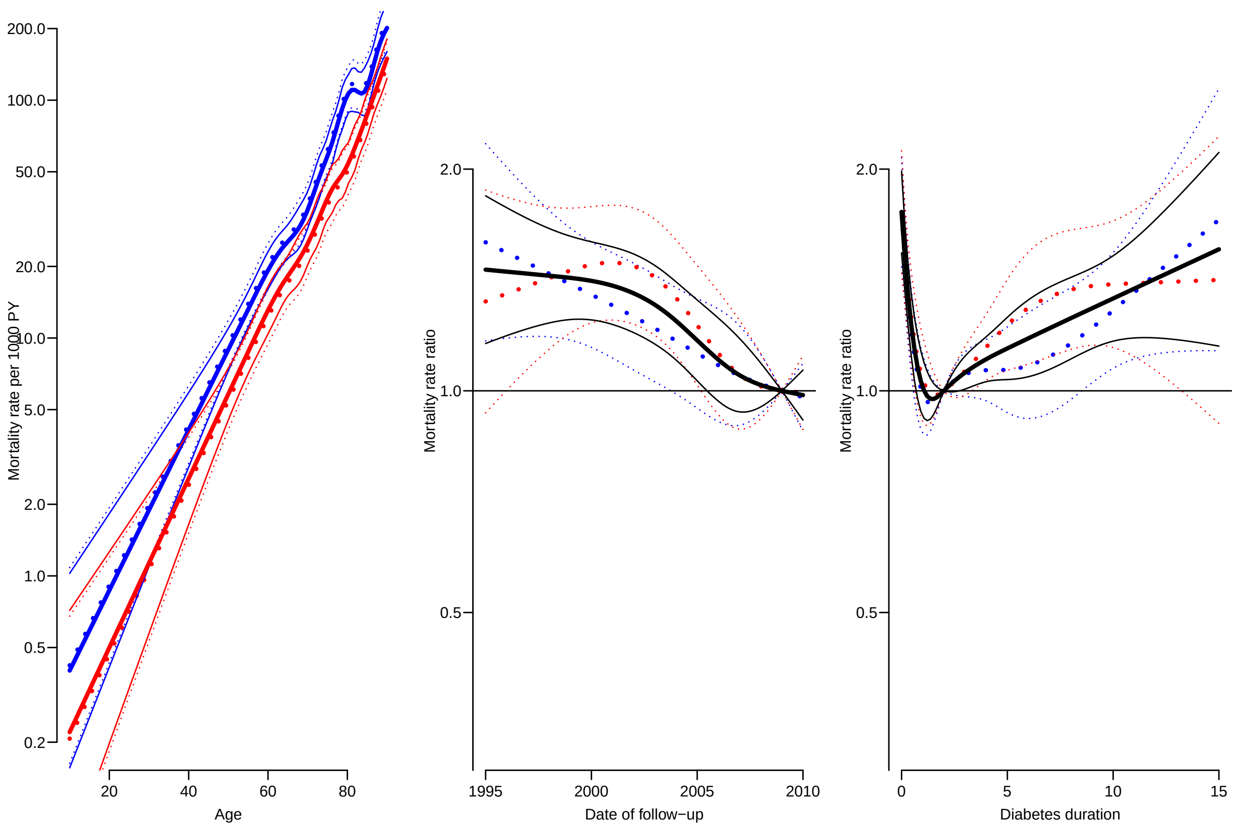

Figure 3.13: *Estimates from models for mortality of Danish diabetes patients. The broken lines are from the full interaction model, full lines with common effects of date and duration. Men:blue, women:red, both (i.e. common): black.*

We can wrap this so that we get the predicted rates with confidence intervals:

```
> sapply(predict( mm, newdata=nd, se.fit=TRUE )[1:2],cbind) %*% ci.mat()
```

This can be nicely wrapped in a function that takes age and date of diagnosis as input and returns the estimated mortality rates for a male and a female diagnosed this age and date:

```
> DMm <-
+ function( A, P, pts=seq(0,25,0.1) )
+ {
+ nd <- data.frame( A=A+pts,
+                   P=P+pts,
+                 dur=  pts,
+             lex.dur=1000 )
+ cbind( nd$A, ci.pred( mm, newdata=nd ),
+              ci.pred( mf, newdata=nd ) )
+ }
> DMm( 50, 1996, pts=0:10 )
```

With this in place we can now plot the mortality rates for persons diagnosed at different ages and different dates:

```
> DMm.1996 <-
+ rbind(
+ DMm( 30, 1996 ), NA,
+ DMm( 40, 1996 ), NA,
+ DMm( 50, 1996 ), NA,
+ DMm( 60, 1996 ), NA,
+ DMm( 70, 1996 ), NA,
+ DMm( 80, 1996 ), NA,
+ DMm( 90, 1996 ) )
> DMm.2005 <-
+ rbind(
+ DMm( 30, 2005 ), NA,
+ DMm( 40, 2005 ), NA,
+ DMm( 50, 2005 ), NA,
+ DMm( 60, 2005 ), NA,
+ DMm( 70, 2005 ), NA,
+ DMm( 80, 2005 ), NA,
+ DMm( 90, 2005 ) )
> par( mfrow=c(1,2), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> matplot( DMm.1996[,1], DMm.1996[,-1],
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(1,1000), xlim=c(30,95), las=1,
+          xlab="Age", ylab="Mortality rate per 1000 PY" )
> text( 30, 1000, "DM diagnosed 1996", adj=c(0,1) )
> matplot( DMm.2005[,1], DMm.2005[,-1],
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(1,1000), xlim=c(30,95), las=1,
+          xlab="Age", ylab="Mortality rate per 1000 PY" )
> text( 30, 1000, "DM diagnosed 2005", adj=c(0,1) )
```

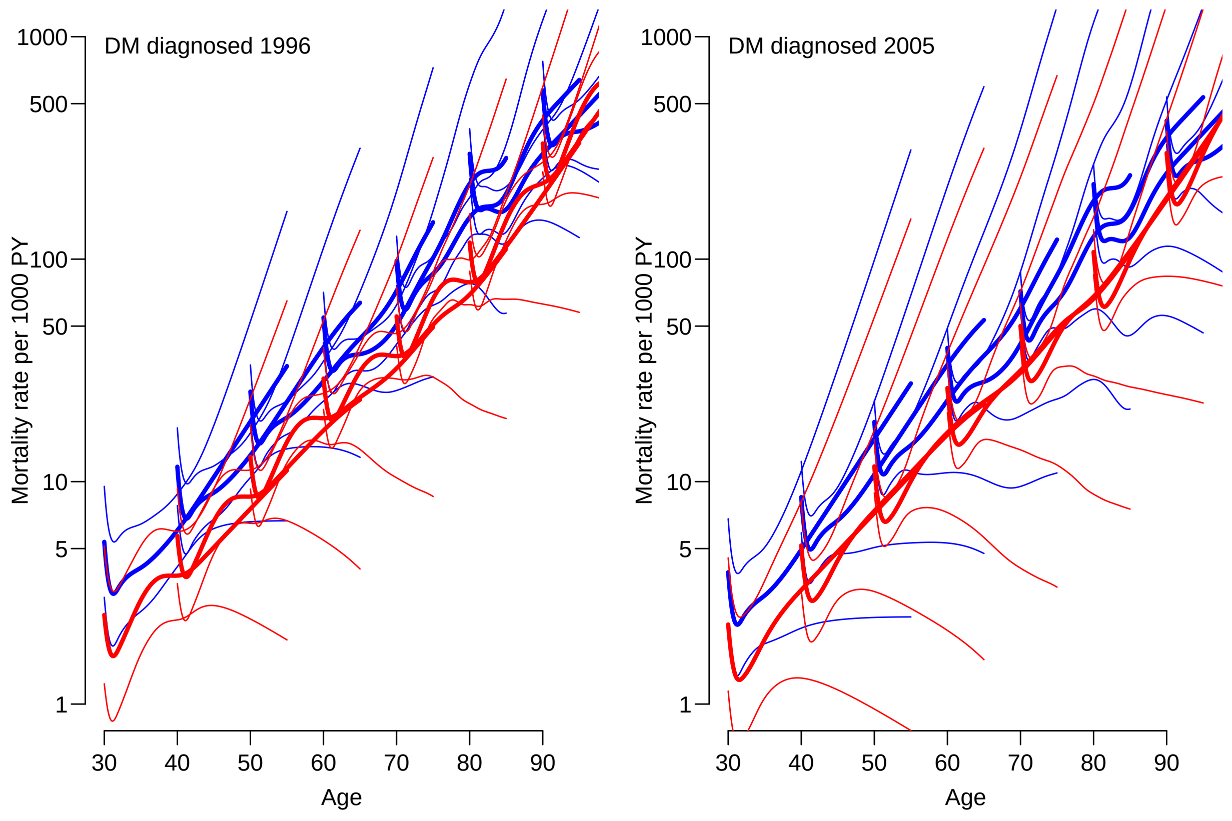

Figure 3.14: *Estimates of mortality of Danish diabetes patients for patients diagnosed in ages 30, 40, . . . , 90.*

Note from figure 3.14 that it seems that mortality among men is higher the younger age at diagnosis, but not for women. But also note that we predicted from 0 to 25 years of diabetes duration, which is a bit bold, given that we only have 15 years of observation, and thus no one with diabetes duration longer than that. Also the rightmost boundary knot for the duration effect is at 10 years, so we are effectively assuming that the duration effect is (log-)linear beyond this — for 15 years, out which we have data for the first 5!

18. The model we used for the mortality rates used three time-scales: age, calendar time and duration of diabetes.

    It would be of interest to see whether we would get the same (or better) description by adding age at diagnosis and date of diagnosis to the model.

    Now, age at diagnosis = current age − duration of diabetes, and date of diagnosis = current date − duration of diabetes, so the terms we might add only constitute the *non-linear* effects of these variables.

    We add the effects one at at time and test whether age at diagnosis or current age is the better predictor, but we want to use a set of knots which is aligned to the new variables we consider:

```
> kn.Ad <- with( subset( SL, lex.Xst=="Dead" ),
+                quantile( A-dur, probs=seq(5,95,10)/100 ) )
> kn.Pd <- with( subset( SL, lex.Xst=="Dead" ),
+                quantile( P-dur, probs=seq(5,95,20)/100 ) )
```

    We can now make on-the-fly tests of the non-linear effects of these fixed effects using `anova`:

```
> anova( mm,
+        update( mm, . ~ . + Ns(A-dur,knots=kn.Ad) ),
+        update( mm, . ~ . + Ns(A-dur,knots=kn.Ad) - Ns(A,knots=kn.A) ),
+        test = "Chisq" )
> anova( mm,
+        update( mm, . ~ . + Ns(P-dur,knots=kn.Pd) ),
+        update( mm, . ~ . + Ns(P-dur,knots=kn.Pd) - Ns(P,knots=kn.P) ),
+        test = "Chisq" )
> anova( mf,
+        update( mf, . ~ . + Ns(A-dur,knots=kn.Ad) ),
+        update( mf, . ~ . + Ns(A-dur,knots=kn.Ad) - Ns(A,knots=kn.A) ),
+        test = "Chisq" )
> anova( mf,
+        update( mf, . ~ . + Ns(P-dur,knots=kn.Pd) ),
+        update( mf, . ~ . + Ns(P-dur,knots=kn.Pd) - Ns(P,knots=kn.P) ),
+        test = "Chisq" )
```

    From this it is pretty clear that there is not much difference between using current age or age at diagnosis, and likewise for date of diagnosis, except possibly for period for women, where it seems more appropriate to use current age (since the p-value for removing this from the model is 0.033). But since the tests concerning the age-effects are insignificant, we could argue that an equally good description of data could be obtained using age at diagnosis and duration of diabetes.

    In conclusion, there does not seem to be much need to change the model we fitted.

19. But we try to fit the models with age at diagnosis and date of diagnosis as explanatory variables instead. To this end we also need new contrast matrices, because the deaths are distributed differently along these “entry”-variables, and we therefor placed the knots differently.

```
> AC <- Ns( pr.A, knots=kn.Ad )
> PC <- Ns( pr.P, knots=kn.Pd )
> PR <- Ns( rep(rf.P,N), knots=kn.Pd )
> Mm <- glm( (lex.Xst=="Dead") ~ Ns( A-dur, kn=kn.Ad ) +
+                                Ns( P-dur, kn=kn.Pd ) +
+                                Ns(   dur, kn=kn.dur ),
+            offset = log( lex.dur ),
+            family = poisson,
+              data = subset( SL, sex=="M" ) )
> Mf <- update( Mm, data = subset( SL, sex=="F" ) )
> M.A <- ci.exp( Mm, ctr.mat=cbind(1,AC,PR,dR) ) * 1000
> F.A <- ci.exp( Mf, ctr.mat=cbind(1,AC,PR,dR) ) * 1000
> M.P <- ci.exp( Mm, subset="P"  , ctr.mat=PC-PR )
> F.P <- ci.exp( Mf, subset="P"  , ctr.mat=PC-PR )
> M.d <- ci.exp( Mm, subset="kn.dur", ctr.mat=dC-dR )
> F.d <- ci.exp( Mf, subset="kn.dur", ctr.mat=dC-dR )
```

Once the models are fitted, we can plot the estimated effects, as seen in figure **??**

```
> par( mfrow=c(1,3), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> matplot( pr.A, cbind(M.A,F.A),
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(0.2,180),
+          xlab="Age at diagnosis", ylab="Mortality rate at 2 years duration per 1000 PY" )
> matplot( pr.P,  cbind(M.P,F.P),
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(1/3,3),
+          xlab="Date of diagnosis", ylab="Mortality rate ratio" )
> abline( h=1 )
> matplot( pr.d,  cbind(M.d,F.d),
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(1/3,3),
+          xlab="Diabetes duration", ylab="Mortality rate ratio" )
> abline( h=1 )
```

The effects shown in figure are shown in a slightly counter-intuitive way; the age-effect is the effect of age *at diagnosis*, the period effect is the effect of date *at diagnosis*, and the duration effect is the only time-scale in the model, the effect of time *since* diagnosis.

20. In order to see how the effects from the two approaches using age/date at diagnosis/follow-up relate to each other we can plot them on top of each other:

```
> par( mfrow=c(1,3), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> matplot( pr.A, cbind(M.A,F.A,m.A,f.A),
+          type="l", lty=rep(c(1,3),each=6), lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(0.2,180),
+          xlab="Age at diagnosis/follow-up", ylab="Mortality rate at 2 years duration per 100
> matplot( pr.P,  cbind(M.P,F.P,m.P,f.P),
+          type="l", lty=rep(c(1,3),each=6), lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(1/3,3),
+          xlab="Date of diagnosis/follow-up", ylab="Mortality rate ratio" )
> abline( h=1 )
> matplot( pr.d,  cbind(M.d,F.d,m.d,f.d),
+          type="l", lty=rep(c(1,3),each=6), lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(1/3,3),
+          xlab="Diabetes duration", ylab="Mortality rate ratio" )
> abline( h=1 )
```

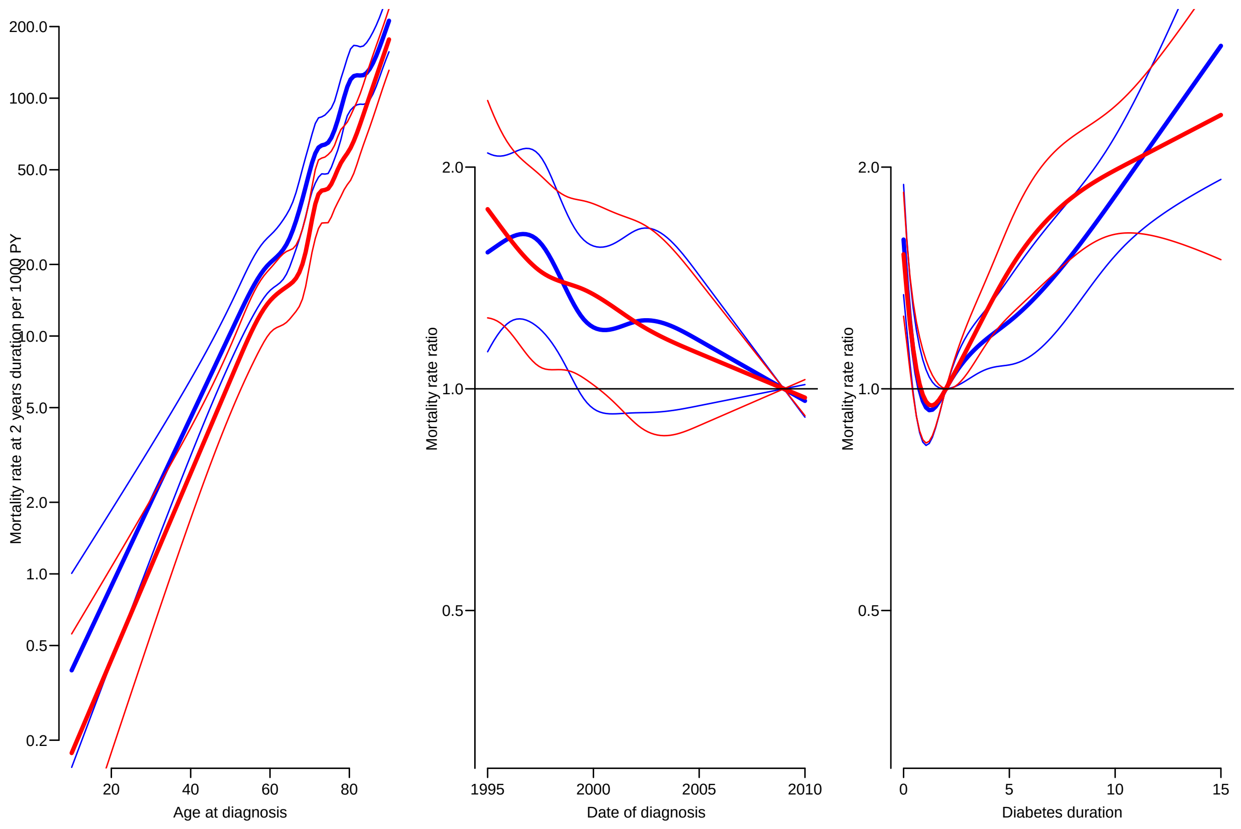

Figure 3.15: *Model for diabetes patient mortality using age and date at diagnosis.*

From figure 3.16 we see that the age and duration curves from the model with two time scales have smaller slopes than those from the model with the age and calendar time as fixed effects. This is because in the latter all the time effect (that is the effect of the clock advancing) is in the duration effect. The *sum* of the average slopes are however the same.

3.9.1 SMR

The SMR is the standardized mortality ratio, which is mortality rate-ratio between the diabetes patients and the general population. In real studies we would subtract the deaths and the person-years among the diabetes patients from those of the general population, but since we do not have access to these (recall that we only have a random sample of 10,000 diabetes patients), we make the comparison to the general population at large, *i.e.* also including the diabetes patients.

There are two ways to make the comparison to the population mortality; one is toe amend the diabetes patient dataset with the population mortality dataset, the other (classical) one is to include the population mortality rates as a fixed variable in the calculations.

The latter requires that each analytical unit in the diabetes patient dataset is amended with a variable with the population mortality for the corresponding sex, age and calendar time.

This can be achieved in two ways: Either we just use the current split of follow-up time

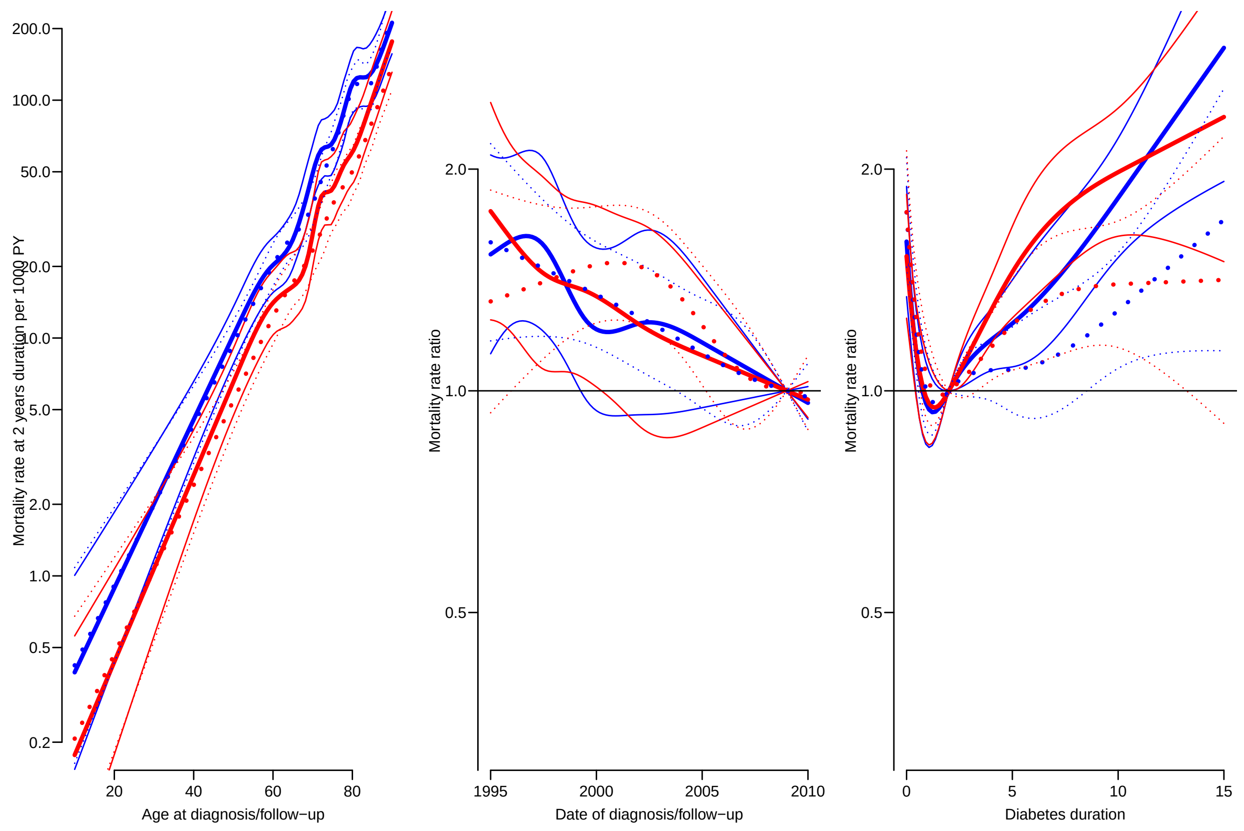

Figure 3.16: *Comparison of estimates from two different models; the full lines give the estimates from the model where age and date are included as fixed variables with the value at diabetes diagnosis, whereas the dotted lines are estimates from the model where age and calendar time are included as time scales.*

and allocate the population mortality rates for some suitably chosen (mid-)point of the follow-up in each, or we make a second split by date, so that follow-up in the diabetes patients is in the same classification of age and data as the population mortality table.

21. We will use the second approach, that is include as an extra variable the population mortality as available from the data set `M.dk`.

    First we create the variables in the diabetes dataset that we need for matching with the population mortality data, that is age, date and sex at the midpoint of each of the intervals (or rater at a point 3 months after the left end point of the interval — recall we split the follow-up in 6 month intervals).

    We need to have variables with the same names in both data sets, moreover, they should be of the same type, so we must transform the sex variable in `M.dk` to a factor:

```
> str( SL )
> SL$Am <- floor( SL$A+0.5 )
> SL$Pm <- floor( SL$P+0.5 )
> data( M.dk )
> str( M.dk )
> M.dk <- transform( M.dk, Am = A,
+                          Pm = P,
+                          sex = factor( sex, labels=c("M","F") ) )
> str( M.dk )
```

Then we can match up the rates from `M.dk`:

```
> SLr <- merge( SL, M.dk[,c("Am","Pm","sex","rate")] )
> dim( SL )
> dim( SLr )
```

This merge only takes rows that have information from both data sets, hence the slightly fewer rows in `SLr` than in SL. There is no point in including observations where there is no risk time among the diabetes patients; the computed expected numbers will be 0, and hence crash the analysis.

22. We can now compute the SMR as the observed divided by the expected numbers by say age and sex:

```
> stat.table( list( Age=floor(A/10)*10,
+                   Sex=sex ),
+             list( D=sum(lex.Xst=="Dead"),
+                   E=sum(lex.dur*rate/1000),
+                 SMR=ratio(lex.Xst=="Dead",lex.dur*rate/1000) ),
+             margins = TRUE,
+                data = SLr )
```

We see that the overall SMR is 1.6, but strongly varying with age and to some extent by sex. Moreover, it may seem that the variation with age is not the same for the two sexes.

23. We can now model the SMR by including the log-expected numbers instead of the log-person-years as offset, using separate models for men and women. Also note that we exclude those units where no deaths in the population occur. Also we compute the expected numbers, `E`:

```
> SLr <- subset( SLr, rate>0)
> SLr$E <- SLr$lex.dur * SLr$rate / 1000
> Sm <- glm( (lex.Xst=="Dead") ~ Ns( A-dur, kn=kn.Ad ) +
+                                Ns( P-dur, kn=kn.Pd ) +
+                                Ns( dur, kn=kn.dur ),
+            offset = log( E ),
+            family = poisson,
+              data = subset( SLr, sex=="M" ) )
> Sf <- update( Sm, data = subset( SLr, sex=="F" ) )
```

The estimates are extracted exactly as for the mortality model; but the results are not mortality rates but rather SMRs (rate-ratios):

```
> sM.A <- ci.exp( Sm, ctr.mat=cbind(1,AC,PR,dR) )
> sF.A <- ci.exp( Sf, ctr.mat=cbind(1,AC,PR,dR) )
> sM.P <- ci.exp( Sm, subset="P"     , ctr.mat=PC-PR )
> sF.P <- ci.exp( Sf, subset="P"     , ctr.mat=PC-PR )
> sM.d <- ci.exp( Sm, subset="kn.dur", ctr.mat=dC-dR )
> sF.d <- ci.exp( Sf, subset="kn.dur", ctr.mat=dC-dR )
```

— plotted using the same code (with obvious adjustments of the axes:

```
> par( mfrow=c(1,3), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> matplot( pr.A, cbind(sM.A,sF.A),
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(1/3,3),
+          xlab="Age at follow-up", ylab="SMR" )
> abline( h=1 )
> matplot( pr.P,  cbind(sM.P,sF.P),
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(1/3,3),
+          xlab="Date of follow-up", ylab="SMR ratio" )
> abline( h=1 )
> matplot( pr.d,  cbind(sM.d,sF.d),
+          type="l", lty=1, lwd=c(3,1,1), col=rep(c("blue","red"),each=3),
+          log="y", ylim=c(1/3,3),
+          xlab="Diabetes duration", ylab="SMR ratio" )
> abline( h=1 )
```

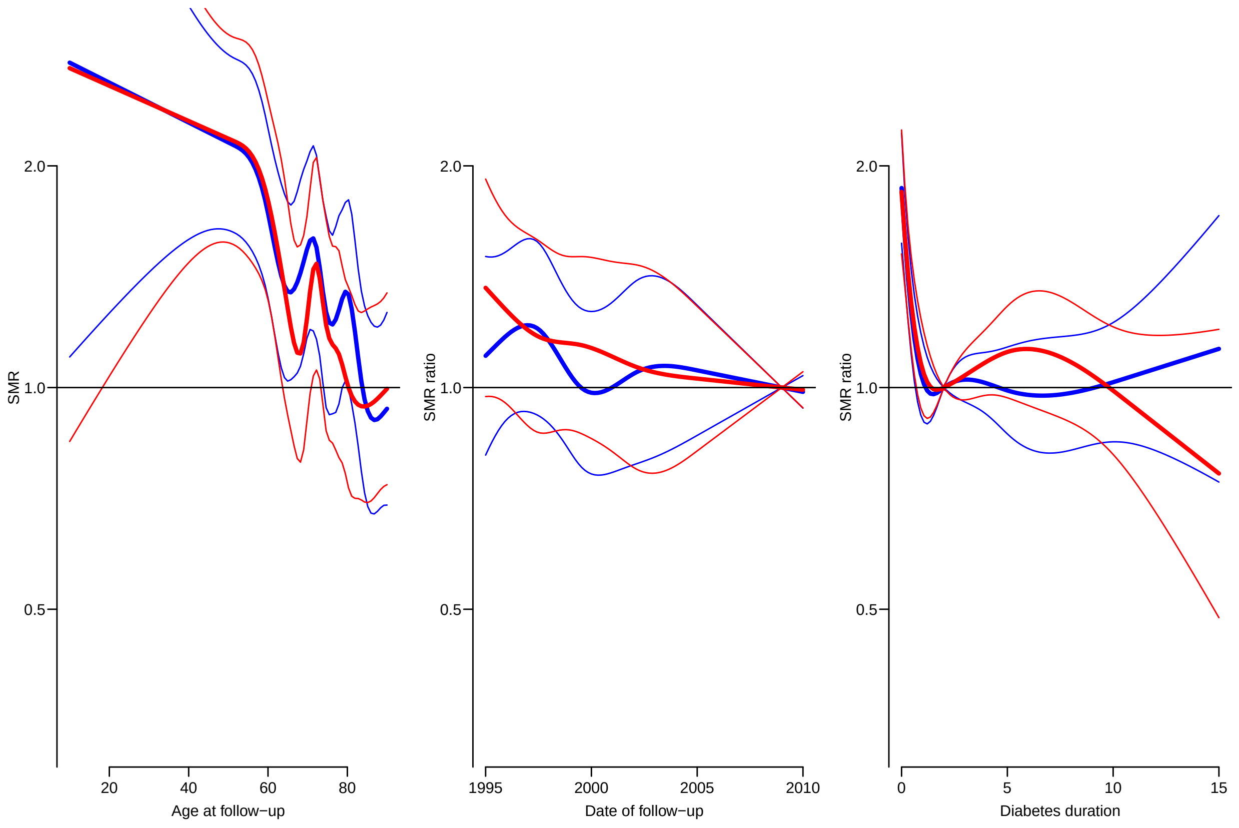

Figure 3.17: *SMR in the diabetic population relative to the (entire) Danish population. Clearly the effect of age is over-modeled.*

24. It seems reasonably from figure **??** clear that there is very little difference between SMR for males and females once we controlled for age, date and duration of diabetes. This can be formally tested by fitting models with and without sex-interaction and also a model with no overall effect of sex:

```
> Sb   <- update( Sm, data = SLr )
> Sb.s <- update( Sb, . ~. + sex )
> Sb.i <- update( Sb, . ~. + sex:( Ns( A-dur, kn=kn.Ad  ) +
+                                  Ns( P-dur, kn=kn.Pd  ) +
+                                  Ns(   dur, kn=kn.dur ) ) )
> anova( Sb, Sb.s, Sb.i, test="Chisq" )
```

So we see there is absolutely no difference between the SMR between the sexes.

25. We therefore extract the parameters from the model with common SMR for the two sexes.

```
> Sb.A <- ci.exp( Sb, ctr.mat=cbind(1,AC,PR,dR) )
> Sb.P <- ci.exp( Sb, subset="P"     , ctr.mat=PC-PR )
> Sb.d <- ci.exp( Sb, subset="kn.dur", ctr.mat=dC-dR )
> par( mfrow=c(1,3), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> matplot( pr.A, Sb.A,
+          type="l", lty=1, lwd=c(3,1,1), col="black",
+          log="y", ylim=c(1/3,3),
+          xlab="Age at diagnosis", ylab="SMR" )
> abline( h=1 )
> matplot( pr.P, Sb.P,
+          type="l", lty=1, lwd=c(3,1,1), col="black",
+          log="y", ylim=c(1/3,3),
+          xlab="Date of diagnosis", ylab="SMR ratio" )
> abline( h=1 )
> matplot( pr.d, Sb.d,
+          type="l", lty=1, lwd=c(3,1,1), col="black",
+          log="y", ylim=c(1/3,3),
+          xlab="Diabetes duration", ylab="SMR ratio" )
> abline( h=1 )
```

26. We can simplify the model to one the is easier to convey to users by using a linear effect of date of diagnosis, and using only knots at 0,1,and 2 years for duration, giving an estimate of the change in SMR as duration increases beyond 2 years. At the same time we also limit the number of knots for the age-effect:

```
> kn.Ad <- with( subset( SL, lex.Xst=="Dead" ),
+                quantile( A-dur, probs=seq(5,95,20)/100 ) )
> kn.dur <- 0:2
> AC <- Ns( pr.A, knots=kn.Ad )
> dC <- Ns( pr.d, knots=kn.dur )
> dR <- Ns( rep(rf.d,N), knots=kn.dur )
> Sx <- glm( (lex.Xst=="Dead") ~ Ns( A-dur, kn=kn.Ad ) +
+                                 I( P-dur ) +
+                                Ns(   dur, kn=kn.dur ),
+            offset = log( E ),
+            family = poisson,
+              data = SLr )
```

Having fitted the model, we can then plot the estimates from it:

```
> Sx.A <- ci.exp( Sx, ctr.mat=cbind(1,AC,rf.P,dR) )
> Sx.P <- ci.exp( Sx, subset="P"     , ctr.mat=cbind(pr.P-rf.P) )
> Sx.d <- ci.exp( Sx, subset="kn.dur", ctr.mat=dC-dR )
> par( mfrow=c(1,3), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> matplot( pr.A, Sx.A,
+          type="l", lty=1, lwd=c(3,1,1), col="black",
+          log="y", ylim=c(1/2,4),
+          xlab="Age at diagnosis", ylab="SMR" )
> abline( h=1 )
> abline( v=4:8*10, col="gray" )
> matplot( pr.P, Sx.P,
+          type="l", lty=1, lwd=c(3,1,1), col="black",
+          log="y", ylim=c(1/2,4),
+          xlab="Date of diagnosis", ylab="SMR ratio" )
> abline( h=1 )
> matplot( pr.d, Sx.d,
```

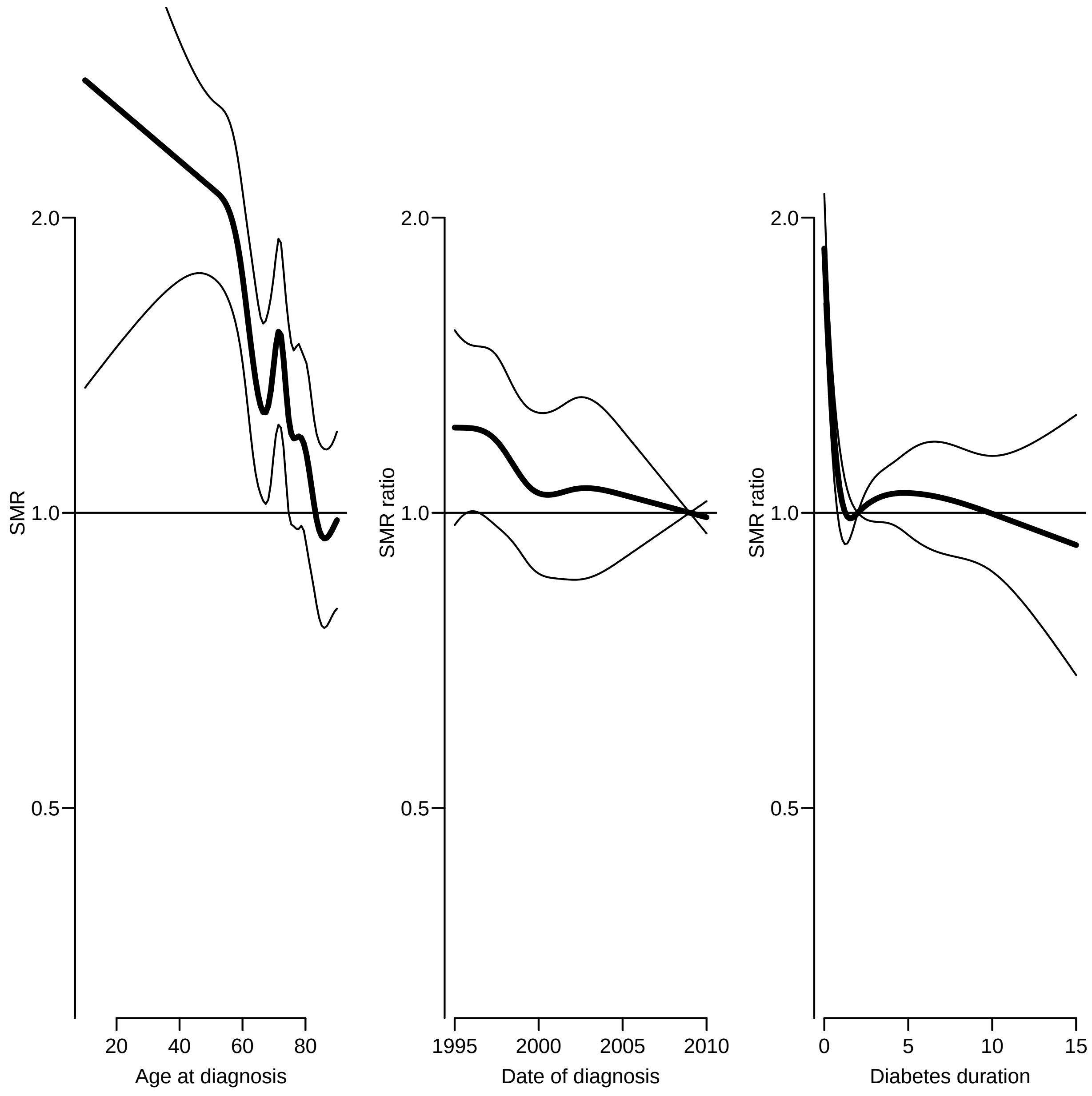

Figure 3.18: *SMR in the diabetic population for both sexes, relative to the (entire) Danish population.*

```
+           type="l", lty=1, lwd=c(3,1,1), col="black",
+           log="y", ylim=c(1/2,4),
+           xlab="Diabetes duration", ylab="SMR ratio" )
> abline( h=1,v=2 )
```

27. We can quantify the period and duration effects by looking at the estimated parameters:

```
> 100*( 1 - ci.exp( Sx, subset="P" ) )
```

Thus the change in SMR is a 1.5% annual decrease (95% c.i.: (0.3-2.7)%).

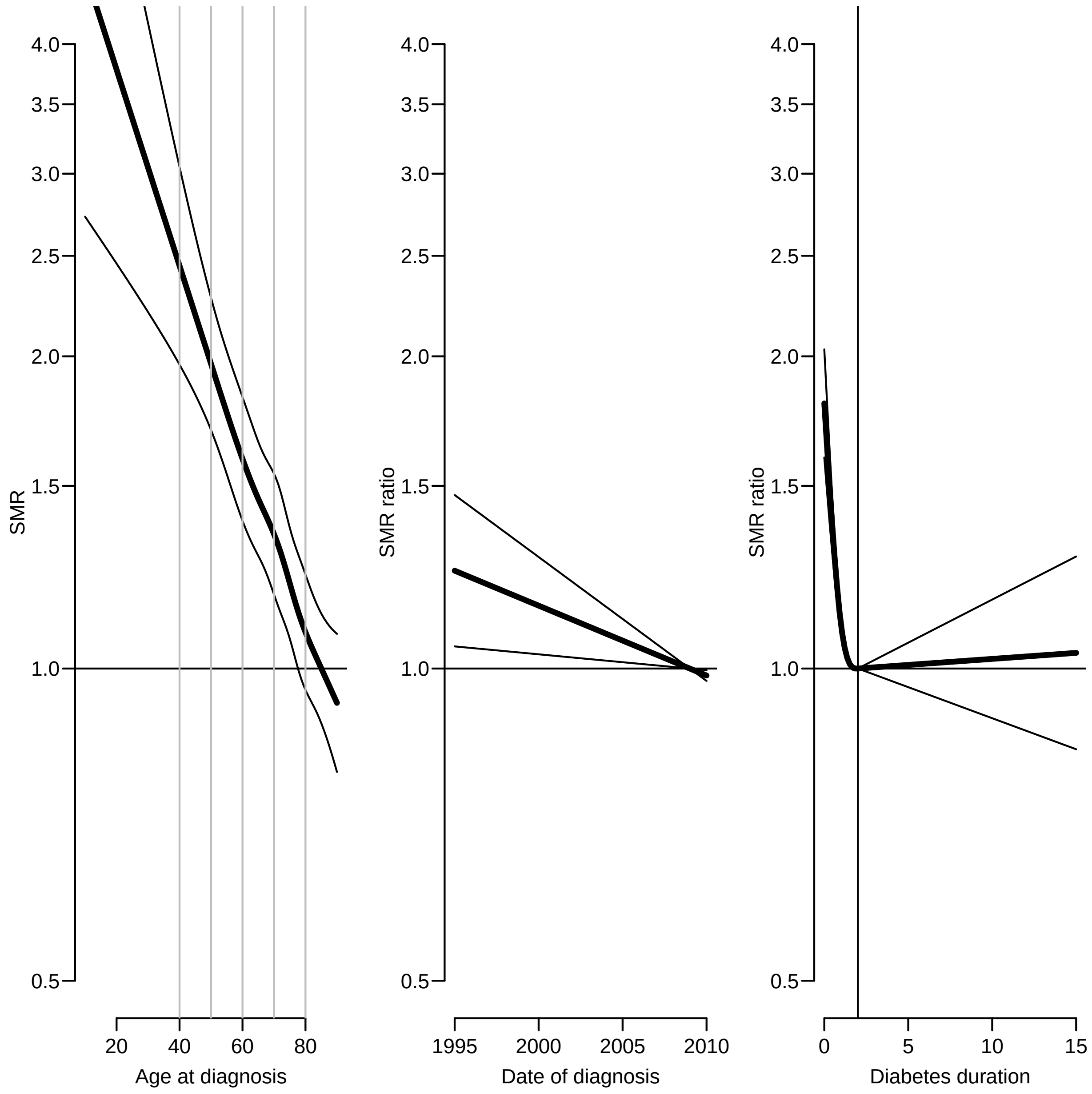

Figure 3.19: *SMR in the diabetic population for both sexes, relative to the (entire) Danish population — simplified model.*

If we want to assess the annual change in SMR by duration of diabetes we can calculate the duration effects at say 5 and 6 years and subtract them:

```
> d6 <- Ns( 6, knots=kn.dur )
> d5 <- Ns( 5, knots=kn.dur )
> 100*( ci.exp( Sx, subset="kn.dur", ctr.mat=d6-d5 ) - 1 )
```

Thus the estimate is an annual increase in SMR of 0.3% (-1.3–1.9)%, thus no evidence of any increasing SMR after 2 years of diabetes duration.

The conclusion is that SMR for diabetes patients diagnosed at age 50 is about 2 after two years of duration and does not change, whereas it for patients aged 70 is about

1.4 after 2 years of diabetes and does not change. The SMR is initially (just after diagnosis) about twice as high, and does not change.

3.9.2 Interaction models

28. We may explore whether there is an interaction between age and duration by including a product of the duration effects and age at diagnosis:

```
> Six <- update( Sx, . ~. + I(A-dur):Ns(dur,knots=kn.dur) )
> anova( Six, Sx, test="Chisq" )
> ci.exp( Six )
```

Even if the effect is not statistically significant, we would still want to explore the shape of it:

```
> Six.A <- ci.exp( Six, ctr.mat=cbind(1,AC,rf.P,dR,dR*pr.A) )
> Six.P <- ci.exp( Six, subset="P"     , ctr.mat=cbind(pr.P-rf.P) )
> Six.d <- ci.exp( Six, subset="kn.dur", ctr.mat=cbind(dC-dR,(dC-dR)*50) )
> for( a in seq(55,90,5) ) Six.d <- cbind( Six.d,
+               ci.exp( Six, subset="kn.dur", ctr.mat=cbind(dC-dR,(dC-dR)*a) ) )
> dim( Six.d )
> par( mfrow=c(1,3), mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> matplot( pr.A, Six.A,
+          type="l", lty=1, lwd=c(3,1,1), col="black",
+          log="y", ylim=c(1/2,4),
+          xlab="Age at diagnosis", ylab="SMR" )
> abline( h=1 )
> abline( v=4:8*10, col="gray" )
> matplot( pr.P, Six.P,
+          type="l", lty=1, lwd=c(3,1,1), col="black",
+          log="y", ylim=c(1/2,4),
+          xlab="Date of diagnosis", ylab="SMR ratio" )
> abline( h=1 )
> matplot( pr.d, Six.d,
+          type="l", lty=1, lwd=c(3,1,1), col=rep(gray(9:1/10),each=3),
+          log="y", ylim=c(1/2,4),
+          xlab="Diabetes duration", ylab="SMR ratio" )
> abline( h=1 )
```

29. This approach is however a bit artificial, because we have fixed the duration effects to be 1 at duration 2 years. It would be appropriate to combine the effects of age at diagnosis and duration to show how the SMR looks as a function of current age.

```
> pts <- c(seq(0,15,0.1),NA)
> np <- length( pts )
> nd <- data.frame( A=rep(seq(50,90,5),each=np)+pts,
+                   P=rf.P+pts,
+                 dur=     pts,
+                   E=1 )
> A.si <- ci.pred( Six, newdata=nd )
> A.sm <- ci.pred( Sx , newdata=nd )
```

```
> matplot( NA, NA,
+          log="y", ylim=c(1/2,5), xlim=c(50,100),
+          xlab="Age at follow-up", ylab="SMR" )
> abline( h=c(5:19/10,seq(2,5,0.5)), v=seq(50,100,5), col=gray(0.8) )
> matlines( nd$A, A.si, type="l", lty=1   , lwd=c(3,1,1), col="forestgreen" )
> matlines( nd$A, A.sm, type="l", lty=c("21","42","42"),
+                                    lwd=c(2,1,1), col="limegreen" )
> abline( h=1 )
```

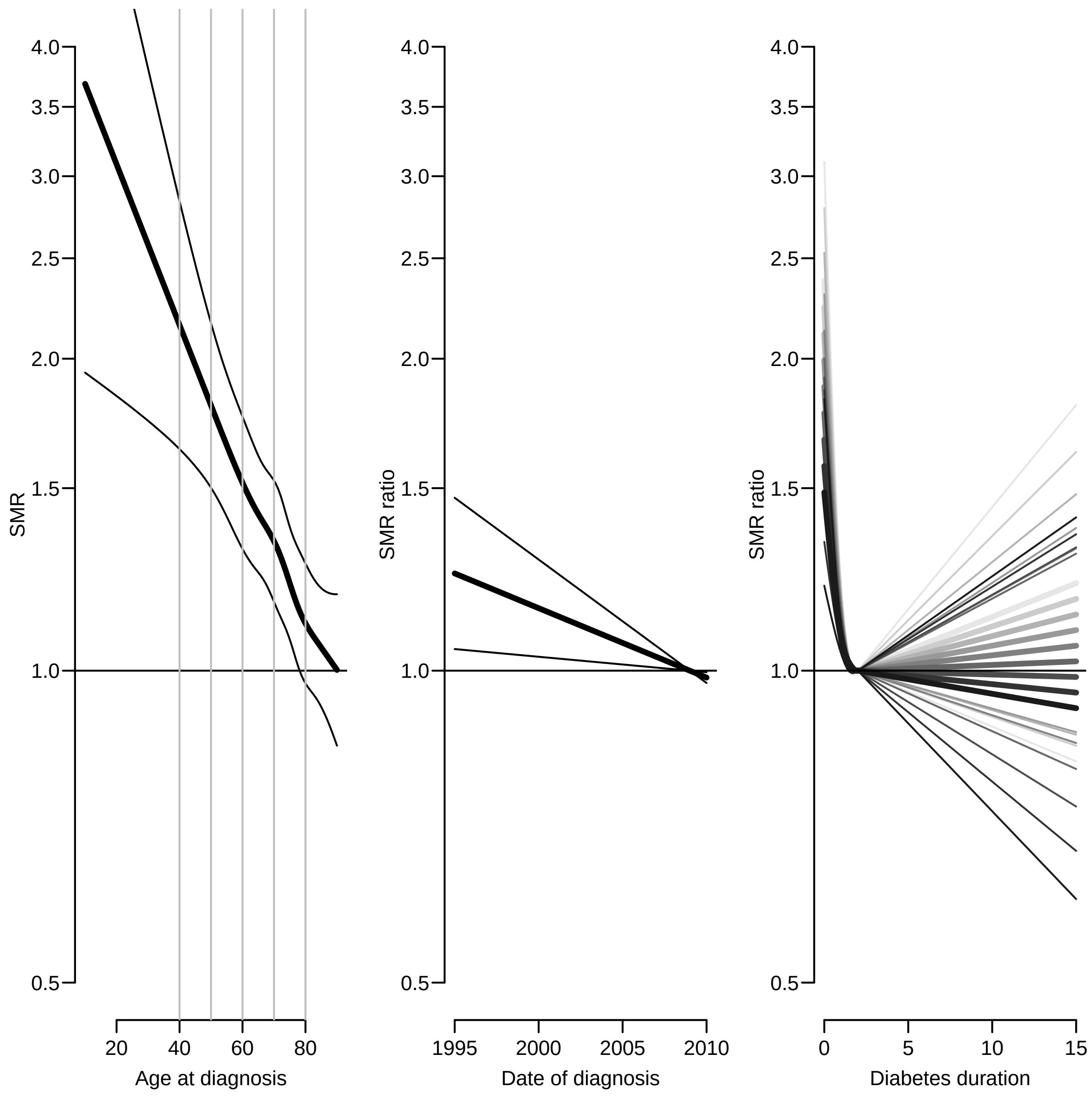

Figure 3.20: *SMR in the diabetic population for both sexes, relative to the (entire) Danish population — interaction model with age-specific duration effects - darkness increases by age.*

From figure **??** it is clear that the interaction means that the patients diagnosed at young age (50–60, that is) do not experience a declining SMR, on the contrary, they have a relative mortality that is close to what it is a year or so after diagnosis, which is about 2 for 50-year old , 1.4 for 70 year old and 1.1 for 80 year old

30. This interaction machinery with linear age easily generalizes to more complex age-effects, it is just a question of choosing another age-effect:

```
> SiX <- update( Sx, . ~. + Ns(A-dur,knots=kn.Ad):Ns(dur,knots=kn.dur) )
> anova( SiX, Six, Sx, test="Chisq" )
```

And we can use the exact same code to show the interaction and plot it along the

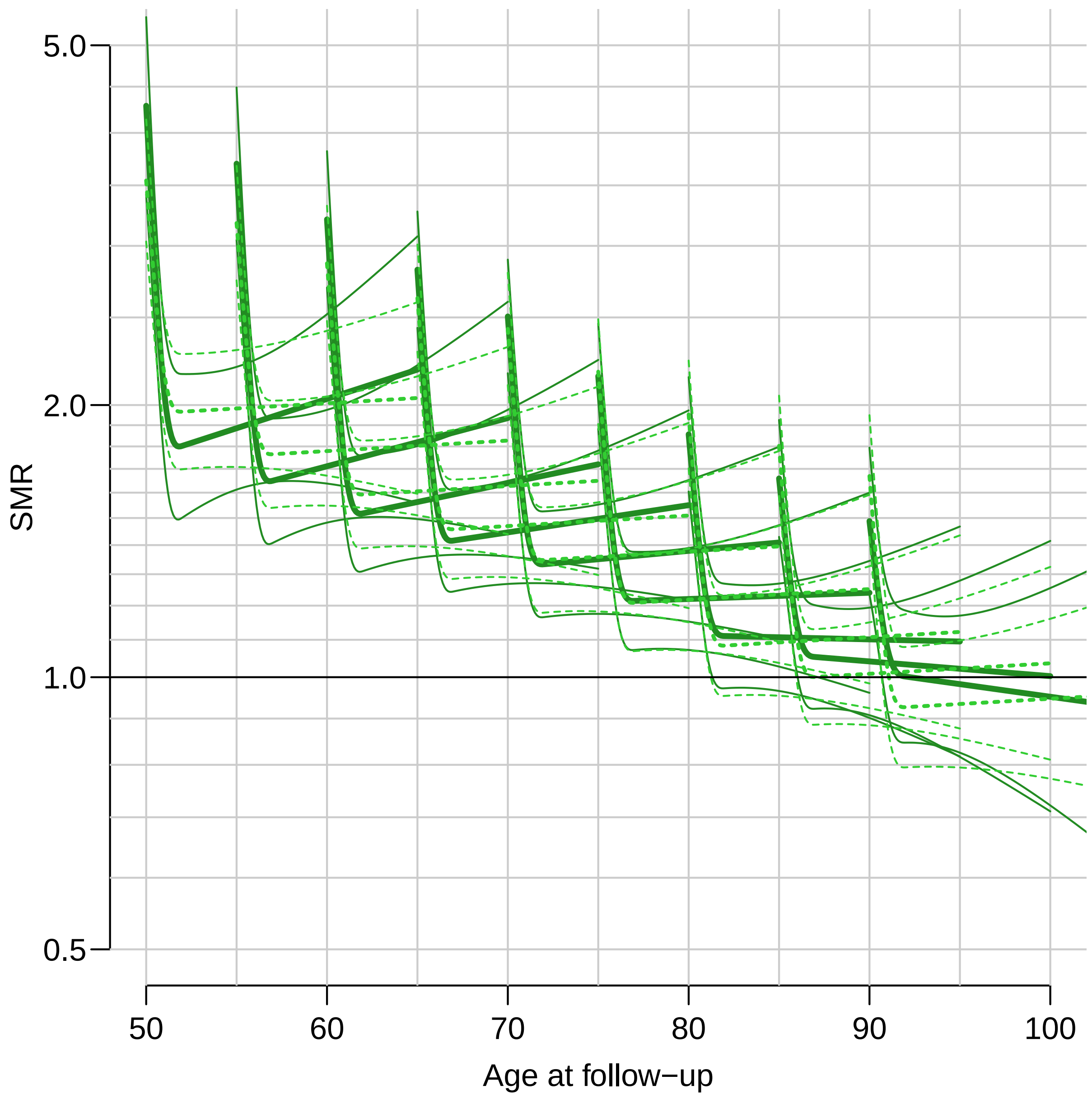

Figure 3.21: *SMR in the diabetic population for both sexes, relative to the (entire) Danish population — interaction model with age-specific duration effects, shown for patients diagnosed at ages 50 to 90. The dotted lines are predictions from the model with no age-duration interaction.*

others in a similar plot:

```
> A.sX <- exp(sapply(predict( SiX, newdata=nd, se.fit=TRUE )[1:2],cbind) %*% ci.mat())
> matplot( NA, NA,
+          log="y", ylim=c(1/2,5), xlim=c(50,100),
+          xlab="Age at follow-up", ylab="SMR" )
> abline( h=c(5:19/10,seq(2,5,0.5)), v=seq(50,100,5), col=gray(0.8) )
> matlines( nd$A, cbind(A.sX,A.si,A.sm),
+           type="l", lty=rep(c(1,3),c(6,3)), lwd=c(3,1,1),
+           col=rep(c("magenta","forestgreen"),c(3,6)) )
> abline( h=1 )
```

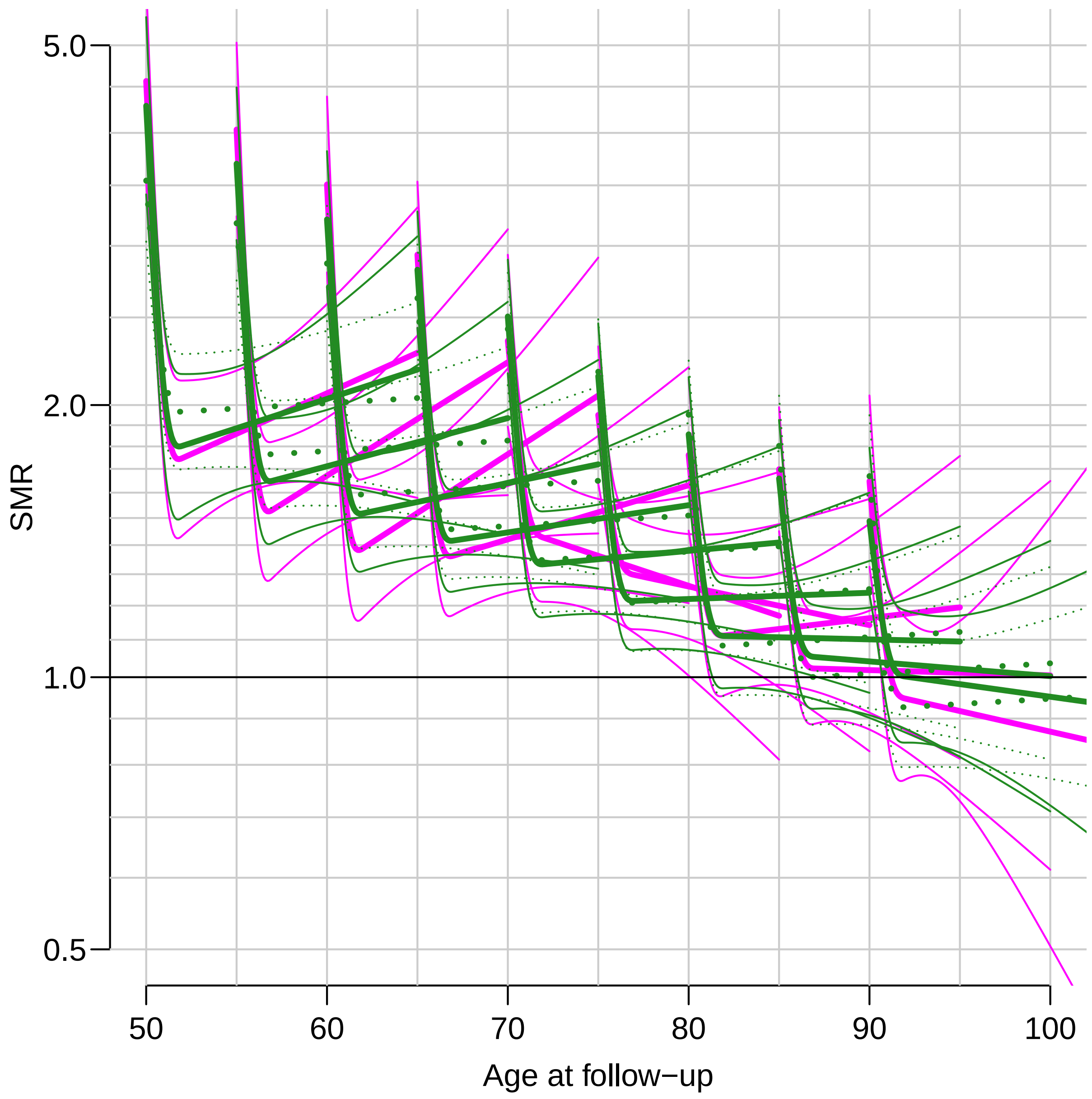

Figure 3.22: *SMR in the diabetic population for both sexes, relative to the (entire) Danish population — interaction model with age-specific duration effects, shown for patients diagnosed at ages 50, 60, 70, 80 and 90. The magenta coloured curves are from the more complex interaction model.*

From figure **??** it is seen that the interaction chosen was way too complex; the long-term variations in the SMR as estimated here do not seem believable. Although the general pattern is pretty much the same; it is the age at diagnosis that determines the SMR.