A short introduction to R for Epidemiology

Friday 5th August, 2011
Version 3

Compiled Friday 5th August, 2011, 14:11
from: `C:/Bendix/undervis/SPE/Intro/R-intro.tex`

Michael Hills
Retired
Highgate, London

Martyn Plummer
International Agency for Research on Cancer, Lyon
plummer@iarc.fr

Bendix Carstensen
Steno Diabetes Center, Gentofte, Denmark
& Department of Biostatistics, University of Copenhagen
bxc@steno.dk
www.pubhealth.ku.dk/~bxc

Edition 2011 by Bendix Carstensen

Contents

1 Getting R running on your computer 1

1.1 What is R? . . . 1

1.2 Getting R . . . 1

1.2.1 Starting R . . . 1

1.2.2 Quitting R . . . 2

1.3 Working with the script editor . . . 2

1.3.1 Try! . . . 2

1.4 Changing the looks of R . . . 2

1.5 Further reading . . . 3

2 Some basic commands in R 4

2.1 Preliminaries . . . 4

2.2 Using R as a calculator . . . 4

2.3 Objects and functions . . . 5

2.4 Sequences . . . 5

2.5 The births data . . . 6

2.6 Referencing parts of the data frame . . . 7

2.7 Summaries . . . 7

2.8 Turning a variable into a factor . . . 8

2.9 Frequency tables . . . 8

2.10 Grouping the values of a metric variable . . . 9

2.11 Tables of means and other things . . . 9

2.12 Generating new variables . . . 10

2.13 Logical variables . . . 10

3 Working with R 12

3.1 Saving the work space . . . 12

3.2 Saving output in a file . . . 12

3.3 Saving R objects in a file . . . 12

3.4 Using a text editor with R . . . 13

3.5 The search path . . . 14

3.6 Attaching a data frame . . . 14

4 Graphs in R 16

4.1 Simple plot on the screen . . . 16

4.2 Colours . . . 17

4.3 Adding to a plot . . . 17

4.3.1 Using indexing for plot elements . . . 17

4.3.2 Generating colours . . . 18

4.4 Interacting with a plot . . . 18

4.5 Saving your graphs for use in other documents . . . 19
4.6 The `par()` command . . . 19

5 The `effx` function for effects estimation 20
5.1 The function effx . . . 20
5.2 Factors on more than two levels . . . 21
5.3 Stratified effects . . . 21
5.4 Controlling the effect of hyp for sex . . . 22
5.5 Numeric exposures . . . 22
5.6 Checking on linearity . . . 22
5.7 Frequency data . . . 23

6 Dates in R 24

7 Follow-up data in the `Epi` package 26
7.1 Timescales . . . 26
7.2 Splitting the follow-up time along a timescale . . . 27
7.3 Cutting time at a specific date . . . 30
7.4 Competing risks — multiple types of events . . . 33
7.5 Multiple events of the same type (recurrent events) . . . 34
References . . . 37

8 R command sheet 38
Getting help . . . 38
Input and output . . . 38
Data creation . . . 39
Slicing and extracting data . . . 39
Variable conversion . . . 39
Variable information . . . 39
Data selection and manipulation . . . 40
Math . . . 40
Matrices . . . 40
Advanced data processing . . . 41
Strings . . . 41
Dates and Times . . . 41
Plotting . . . 41
Low-level plotting commands . . . 42
Graphical parameters . . . 43
Lattice (Trellis) graphics . . . 43
Optimization and model fitting . . . 44
Statistics . . . 44
Distributions . . . 44
Programming . . . 44
The `Epi` package . . . 44

9 The `Epi` package 46
apc.fit . . . 46
apc.frame . . . 49
apc.lines . . . 51
apc.plot . . . 53
bdendo . . . 54

bdendo11 54
births 55
blcaIT 55
brv 56
cal.yr 56
ccwc 58
ci.cum 59
ci.lin 60
ci.pd 63
clogistic 64
contr.cum 65
cutLexis 66
detrend 69
diet 69
DMconv 70
DMlate 71
effx 72
effx.match 73
ewrates 75
expand.data 75
fit.add 76
fit.baseline 77
fit.mult 78
float 79
ftrend 80
gmortDK 81
hivDK 82
Icens 83
lep 84
Lexis 85
Lexis.diagram 87
Lexis.lines 89
Life.lines 90
lls 91
lungDK 92
merge.data.frame 93
merge.Lexis 94
mh 95
mortDK 96
msdata.Lexis 97
ncut 98
nice 99
nickel 100
occup 100
pctab 101
plot.Lexis 102
plotEst 104
plotevent 106
projection.ip 107
rateplot 107

Relevel . . . 111
ROC . . . 111
S.typh . . . 113
splitLexis . . . 114
stack.Lexis . . . 115
start.Lexis . . . 116
stat.table . . . 117
stattable.funs . . . 119
subset.Lexis . . . 120
summary.Lexis . . . 120
tbox . . . 121
thoro . . . 125
timeBand . . . 126
timeScales . . . 127
transform.Lexis . . . 128
twoby2 . . . 128

Chapter 1

Getting R running on your computer

1.1 What is R?

R is free program for data analysis and graphics. It contains all state of the art statistical methods, and has become the preferred analysis tool for most professional statisticians in the world. It can be used as simple calculator and as a very specialized statistical analysis and reporting machinery.

The special thing about R is that you enter commands from the keyboard into a console window, where you also see the results. This is an advantage because you end up with a script that you can use to *reproduce* your analyses—a requirement in any scientific endeavour.

The disadvantage is that you somehow have to find out what to type. The practicals will contain some hints, and you will mostly be using R as a calculator, as you just saw — type an expression, hit the return key and you get the result.

1.2 Getting R

You can obtain R, which is free, from CRAN (the **C**omprehensive **R** **A**rchive **N**etwork), at `http://cran.r-project.org/`. Under "`Download and Install R`" click on "Windows" and under "`R for Windows`" click on "base". Then on "Download R 2.13.1 for Windows", which is a self-extracting installer. This means that if you save it to your computer somewhere and click on it, it will install R for you.

Apart from what you have downloaded there are several thousand add-on packages to R dealing with all sorts of problems from ecology to fiance and incidentally, epidemiology. You must download these manually. In this course we shall only need the `Epi` package.

1.2.1 Starting R

You start R by clicking on the icon that the installer has put on your desktop. You should edit the properties of this, so that R starts in the folder that you have created on your computer for this course.

Once you have installed R, start it, and in the menu bar click on Packages → Install package(s)..., chose a mirror (this is just a server where you can get the stuff), and the `Epi` package.

Once R (hopefully) has told you that it has been installed, you can type:

```
> library( Epi )
```

to get access to the `Epi` package. You can get an overview of the functions and datasets in the package by typing:

```
> library( help=Epi )
```

It should be apparent that you have version 1.1.24 of the `Epi` package.

1.2.2 Quitting R

Type `q()` in the console, and answer “No” when asked whether you want to save workspace image.

1.3 Working with the script editor

If you click on File → New script, R will open a window for you which is a text-editor very much like Notepad.

If you write a command in it you can transfer it to the R console and have it executed by pressing CTRL-r. If nothing is highlighted, the line where the cursor is will be transmitted to the console and the cursor will move to the next line. If a part of the screen is highlighted the highlighted part will be transmitted to the console. Highlighting can also be used to transmit only a part of a line of code.

1.3.1 Try!

Now open a script by File → New script, and type (omit the “>” in the beginning of the line):

```
> 5+7
> pi
> 1:10
> N <- c(27,33,81)
> N
```

Run the lines one at a time by pressing CTRL-r, and see what happens.

You can also type the commands in the console directly. But then you will not have a record of what you have done. Well, you can press File → Save History and save all you typed in the console (including the 73.6% commands with errors).

1.4 Changing the looks of R

If you want R to start up with a different font, different colors etc., the go to the folder where R is installed — most likely `Program Files\R\R-2.13.1`, then to the folder `etc`, and open the file `Rconsole` with Notepad. In the file are specifications on how R will look when you start it, pretty self-explanatory, except perhaps for MDI.

MDI means “**M**ultiple **D**isplay **I**nterface”, which means you get a single R-window, and within that sub-windows with the console, the script editor, graphs etc. If this is set to “no”, you get SDI which means “**S**ingle **D**isplay **I**nterface”, which means that R will open the console, script editor etc. in separate windows of their own.

A withe background can be trying to look at so on my (BxC) computer I use a bold font and the following colors:

```
> background = gray5
> normaltext = yellow2
> usertext = green
> pagerbg = gray5
> pagertext = yellow2
> highlight = red
> dataeditbg = gray5
> dataedittext = red
> dataedituser = yellow2
> editorbg = gray5
> editortext = lightblue
```

(If you want to know which colors are available in R, just give the command `colors()`).

1.5 Further reading

On the CRAN web-site the last menu-entry on the left is "Contributed" and will take you to a very long list of various introductions to R, including manuals in esoteric languages such as Danish, Finnish and Hungarian.

Chapter 2

Some basic commands in R

2.1 Preliminaries

The purpose of these notes is to describe a small subset of the Rlanguage, sufficient to allow someone new to R to get started. The exercises are important because they reinforce basic aspects of R. For further details about R we refer the reader to **An Introduction to R** by W.N.Venables, D.M.Smith, and the R development team. This can be downloaded from the R website at http://www.r-project.org.

To start R click on the R icon. To change your working directory click on File → Change dir... and select the directory you want to work in. Alternatively you can write:

```
> setwd("c:/where/alll/my/files/are")
```

To get out of R click on the File menu and select Exit, or simpler just type “q()”. You will be offered the chance to save the work space, but at this stage just exit without saving, then start R again, and change the working directory, as before.

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. When using R it makes sense to avoid as much typing as possible by recalling previous commands using the vertical arrow key and editing them.

2.2 Using R as a calculator

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

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

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

which can be used in further calculations. The expression <-, pronounced ”gets”, is called the assignment operator, and is obtained by typing < and then -. The assignment operator can also be used in the opposite direction, as in

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

The contents of a can be printed by typing 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

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

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

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

will return the probability above 3.84.

Exercise 2.1.

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

2.3 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
```

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.

2.4 Sequences

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

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

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

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

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 2.2.

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

2.5 The births data

Table 2.1: *Variables in the births dataset*

Variable	Units or Coding	Type	Name
Subject number	–	categorical	`id`
Birth weight	grams	metric	`bweight`
Birth weight < 2500 g	1=yes, 0=no	categorical	`lowbw`
Gestational age	weeks	metric	`gestwks`
Gestational age < 37 weeks	1=yes, 0=no	categorical	`preterm`
Maternal age	years	metric	`matage`
Maternal hypertension	1=hypertensive, 0=normal	categorical	`hyp`
Sex of baby	1=male, 2=female	categorical	`sex`

The most important example of a vector in epidemiology is the data on a variable recorded for a group of subjects. To introduce R we use the births data which concern 500 mothers who had singleton births in a large London hospital. These data are available as an R object called `births` in the Epi package. You can get them into your workspace by:

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

Try

```
> objects()
```

to make sure that you have an object called births in your working directory. A more detailed overview of the objects in your workspace is obtained by:

```
> lls()
```

The function

```
> str(births)
```

shows that the object `births` is a data frame with 500 observations of 8 variables. The names and types of the variables are also shown together with the first 10 values of each variable.

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

Exercise 2.3.

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)
   ```
5. Check that you only have the object births left.

2.6 Referencing parts of the data frame

Typing births will list the entire data frame - not usually very helpful. Now try

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

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

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

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

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

and to list all the data for this variable, try

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

Exercise 2.4.

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

2.7 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)
```

In most datasets there will be some missing values. These are usually coded using tab delimited blanks to mark the values which are missing. R then codes the missing values using the NA (not available) symbol. The summary shows the number of missing values for each variable.

2.8 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

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

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

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

which shows that hyp, in the births data frame, is now a factor with two levels, labeled "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 <- transform( births, hyp=factor(hyp,labels=c("normal","hyper")) )
> str(births)
```

Exercise 2.5.

1. Convert the variable sex into a factor
2. Label the levels of sex as "male" and "female".

2.9 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 tables is stat.table. This is currently part of the Epi package, so you will need to load this package first with

```
> library(Epi)
```

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

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

Their cross-tabulation is obtained by typing

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

Cross-tabulations are useful when checking for consistency, but because no distinction is drawn between the response variable and any explanatory variables, they are not useful as a way of presenting data.

2.10 Grouping the values of a metric 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))
> stat.table(agegrp,data=births)
```

By default the factor levels are labeled [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.

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

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

Only observations from 20 up to, but not including 35, are included. For the rest, `agegrp` is coded missing. 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))
> stat.table(agegrp,data=births)
```

shows 5 intervals of width 4.

Exercise 2.6.

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.

2.11 Tables of means and other things

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

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

The headings of the table can be improved with

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

To make a two-way table of mean birth weight by sex and hypertension, try

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

and to tabulate the count as well as the mean, try

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

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

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

The scale factor 100 makes the odds per 100. Margins can be added to the tables, as required. For example,

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

for a one-way table, and

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

for a two-way table.

Exercise 2.7.

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

2.12 Generating new variables

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

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

The sign `^` means "to the power of", `log` means "natural logarithm", and `sqrt` means "square root".

The `transform()` function allows you to transform or generate variables in a data frame. For example, try

```
> births <- transform(births,
+           num1=1,
+           num2=2,
+           logbw=log(bweight))
```

The variable `logbw` is the natural logarithm of birth weight. Logs base 10 are obtained with `log10( )`.

2.13 Logical variables

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 <- transform(births, low=bweight<2000)
> str(births)
```

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

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

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

```
> births$bweight[births$hyp=="hyper"]
```

If you want the entire dataframe restricted to hypertensive women try:

```
> births[births$hyp=="hyper",]
```

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

```
> subset(births, hyp=="hyper")
```

Exercise 2.8.

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

Chapter 3

Working with R

3.1 Saving the work space

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

3.2 Saving output in a file

To save the output from an R command in a file, for future use, the `sink()` command is used. For example,

```
> sink("output.txt")
> summary(births)
```

first instructs R to re-direct output away from the R terminal to the file `"output.txt"` and then summarizes the births data frame, the output from which goes to the sink. While a sink is open all output will go to it, replacing what is already in the file. To append output to a file, use the `append=TRUE` option with `sink()`. To close a sink, use

```
> sink()
```

Exercise 3.9.

1. Sink output to a file called `"output1.txt"`.
2. Make frequency tables of `hyp` and `sex`
3. Make a table of mean birth weight by sex
4. Close the sink
5. From windows, have a look inside the file `output1.txt` and check that the output you expected is in the file.

3.3 Saving R objects in a file

The command `read.table()` is relatively slow because it carries out quite a lot of processing as it reads the data. To avoid doing this more than once you can save the data frame, which includes the R information, and read from this saved file in future. For example,

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

will save the births data frame in the file `births.Rdata`. 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("births.Rdata")
```

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

Exercise 3.10.

1. Use `read.table()` to read the data in the file `diet.txt` into a data frame called `diet`.
2. Save this data frame in the file `"diet.Rdata"`
3. Remove the data frame
4. Load the data frame from the file `"diet.Rdata"`.

3.4 Using a text editor with R

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. This means you can use the cut and paste facilities of the editor to cut down on typing. For Windows we recommend using the text editor Tinn-R, but you can use your favorite text editor instead if you prefer, and copy-paste commands from it into the R-console.

Alternatively you can use the built-in script-editor: Click on `File`→`New script`, or `File`→`Open script`, according to whether you are using an old script. You can move the current line from the script-editor to the console by `CTRL-R`. If you have highlighted a section of the script the highlighted part will be moved to the console.

Now start up the editor and enter the following lines:

```
> births <- transform( births,
+                      lowbw = factor(lowbw, labels=c("normal","low")),
+                        hyp = factor(hyp, labels=c("normal","hyper")),
+                        sex = factor(sex, labels=c("male","female")) )
```

Now save the script as `mygetbirths.R` and run it. One major advantage of running all your R commands from a script is that you end up with a record of exactly what you did which can be repeated at any time.

This will also help you redo the analysis in the (highly likely) event that your data changes before you have finished all analyses.

Exercise 3.11.

1. Create a script called mytab.R which includes the lines
   ```
   > stat.table(hyp,data=births)
   > stat.table(sex,data=births)
   ```
   and run just these two lines.
2. Edit the script to include the lines
   ```
   > stat.table(sex,mean(bweight),data=births)
   > stat.table(hyp,mean(bweight),data=births)
   ```
   and run these two lines.
3. Edit the script to create a factor cutting `matage` at 20, 30, 35, 40, 45 years, and run just this part of the script.

4. Edit the script to create a factor cutting gestwks at 20, 35, 37, 39, 45 weeks, and run just this part of the script.
5. Save and run the entire script.

3.5 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 objects(1) does the same as objects(). A shorther name for the same function is ls(). In the Epi package is a function that gives a more detailed picture, lls(); try:

```
> lls()
```

To see what is in the Epi package, try

```
> ls(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.

3.6 Attaching a data frame

The function objects(1) shows that the only objects in the workspace are births and diet. To refer to variables in the births data frame 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 3.12.

1. Use `search()` to make sure you have no data frames attached.
2. Use `objects(1)` 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.

```
> setwd(sweave.wd)
```

Chapter 4

Graphs 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.

4.1 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

```
> ?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 4.13.

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)")
   ```

4.2 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 center of the cluster of points and it is no longer possible to see individual points. You can recover this information by overwriting the points with black circles using the `points()` function.

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

4.3 Adding to a plot

The `points()` function 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")
```

4.3.1 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.

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 4.14.

1. Make a three level factor for maternal age with cutpoints at 30 and 40 years.
2. Use this to make the plot of gestational weeks with three different plotting symbols. (Hint: Indexing with a factor automatically gives indexes 1,2,3 etc.).

4.3.2 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.

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 )
```

4.4 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.

Exercise 4.15.

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), ]
```

4.5 Saving your graphs for use in other documents

Once you have a graph on the screen you can click on File → 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 Insert → Picture → From File. Metafiles can be resized and edited inside Word.

If you want exact control of the size of your plot you can start a graphics device *before* doing the plot. Instead of appearing on the screen, the plot will be written directly to a file. After the plot has been completed you will need to close the device again in order to be able to access the file. Try:

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

This will give you a enhanced metafile `plot1.emf` with a graph which is 3 inches tall and 4 inches wide.

4.6 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

```
> ?par
```

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

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

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

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

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

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

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

Finally for more complex graphical lay-outs you can use the functions `layout()`, take a look:

```
> ?layout
```

Chapter 5

The `effx` function for effects estimation

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

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

The response variable must be numeric.

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

- Primary role: exposure
- Secondary role: confounder

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

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

What other measures of effects might be used?

5.1 The function effx

The function `effx` is intended to introduce the estimation of effects in epidemiology, together with the related ideas of stratification and controlling, without the need for familiarity with statistical modelling.

We shall use the births data in the Epi package, which can be loaded and inspected with

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

The variables we shall be interested in are `bweight` (birth weight) and `hyp` (hypertension). An alternative way of characterizing birth weight is shown in `lowbw` which is coded 1 for babies with low birth weight, and 0 otherwise. Other variables of interest are `sex` (of the baby) and `gestwks`, the gestation time.

All variables are numeric, so first we need first to do a little housekeeping:

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

Now try

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

The effect of sex on birth weight, measured as a difference in means, is -197. The command

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

verifies this ($3032.8 - 3229.9 = -197.1$). The p-value refers to the test that there is no effect of `sex` on birth weight. Use `effx` to find the effect of `hyp` on `bweight`.

For another example, consider the effect of `sex` on the binary response `lowbw`.

```
> effx(response=lowbw,typ="binary",exposure=sex,data=births)
```

The effect of `sex` on `lowbw`, measured as an odds ratio, is 1.43. The command

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

can be used to verify this ($16.26/11.39 = 1.427$). Use `effx` to find the effect of `hyp` on `lowbw`.

5.2 Factors on more than two levels

The variable `gest4` is the result of cutting `gestwks` into 4 groups with boundaries [20,35) [35,37) [37,39) [39,45). We shall find the effects of `gest4` on the metric response `bweight`.

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

There are now 3 effects

```
[35,37) vs [20,35)  856.6
[37,39) vs [20,35) 1360.0
[39,45) vs [20,35) 1668.0
```

The command

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

verifies that the effect of `agegrp` (level 2 vs level 1) is $2590 - 1733 = 857$, etc. Find the effects of `gest4` on `lowbw`. Use the option `base=4` to change the baseline for `gest4` from 1 to 4.

5.3 Stratified effects

As an example we shall stratify the effects of `hyp` on `bweight` by `sex` with

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

The effects of `hyp` in the different strata defined by `sex` are -496 and -380.

Use `effx` to stratify the effect of `hyp` on `lowbw` first by `sex` and then by `gest4`.

5.4 Controlling the effect of hyp for sex

The effect of `hyp` is controlled for `sex` by first looking at the effects of `hyp` in the two strata defined by `sex`, and then combining these effects if they are similar. In this case the effcts were -496 and -380 which look similar (the test for effect modification is a test of whether they differ significantly) so we can combine them, and control for `sex`.

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

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

The effect of `hyp` on `bweight` controlled for `sex` is -448. Note that it is the name of the control variable which is passed, not the variable itself. There can be more than one control variable, `control=list(sex,agegrp)`.

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

5.5 Numeric exposures

If we wished to study the effect of gestation time on the baby's birth weight then `gestwks` is a numeric exposure. Assuming that the relationship of the response with `gestwks` is roughly linear (for a metric response) or log-linear (for a binary response) we can find the linear effect of `gestwks`.

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

The linear effect of `gestwks` is 197 g per extra week of gestation. The linear effect of `gestwks` on `lowbw` can be found similarly

```
> effx(response=lowbw, type="binary", exposure=gestwks,data=births)
```

The linear effect of `gestwks` on `lowbw` is a reduction by a factor of 0.408 per extra week of gestation, i.e. the odds of a baby having a low birth weight is reduced by a factor of 0.408 per one week increase in gestation.

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

```
> effx(lowbw, type="binary",exposure=gestwks,strata=agegrp,data=births)
```

You can control for a numeric variable by putting it in `control=`.

5.6 Checking on linearity

At this stage it will be best to make a visual check using `plot`. For example, to check whether `bweight` goes up linearly with `gestwks` try

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

Is the relationship roughly linear? It is not possible to check graphically whether log odds of a baby being low birth weight goes down linearly with gestation because the individual odds are either 0 or ∞. Instead we use the grouped variable `gest4`:

```
> tab<-stat.table(gest4,ratio(lowbw,1-lowbw,100),data=births)
> str(tab)
> #Extract the odds from tab, and plot the logodds against 1:4
> odds<-tab[1,1:4]
> plot(1:4,log(odds),type="b")
```

The relationship is remarkably linear, but remember this is quite crude because it takes no account of unequal gestation intervals. More about checking for linearity later.

5.7 Frequency data

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

```
> UCBAdmissions
```

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

The command

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

coerces the contingency table to a data frame, and shows the first 10 lines. The relationship between the contingency table and the data frame should be clear. The command

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

turns `Admit` into a numeric variable coded 1 for rejection, 0 for admission, so

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

shows the odds of rejection for female applicants to be 1.84 times the odds for males (note the use of `weights` to take account of the frequencies). A crude analysis therefore suggests there is a strong bias against admitting females. Continue the analysis by stratifying the crude analysis by department - does this still support a bias against females? What is the effect of gender controlled for department?

Chapter 6

Dates in R

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

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

```
> str(diet)
```

shows that these three variables are Date variables.

You will also see that the values are just numbers, but if you try

```
> head( diet )
```

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

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

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

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

The format parts, "%d" etc., identify elements of the dates, whereas the "/"s are just the separator characters that are in the character string. There are other possibilities for formats, see ?strftime or the section on dates and times in the R command sheet at the end of this document.

Reading dates from an external file is done by reading the fields as character variables and then transforming them to date variables by the function as.Date

If you want to enter a fixed date, for example if you want to terminate follow-up at 1st April 1975 you could say:

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

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

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

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

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

Exercise 6.16.

1. Convert `doe` and `dox` to date variables.
2. Generate a new variable `y` which is the elapsed time in years between the date of entry and the date of exit.
3. The file `getdiet.R` reads the diet data, converts all three date variables to standard form using the `transform` function, and generates the variable `y`. Run this script and check the results are what you want.
4. Enter your own birtday as a date. Print it using `format.Date()` with the format `"%A %d %B %Y"`. Did you learn anything new?
5. Enter the birthday of your husband/wife/... as a date too. When will you be (were you) 100 years old together? (Hint: `mean()` works on vectors of dates as well.)

In the Epi package is also a function `cal.yr` which converts dates to fractional years:

```
> as.Date( "1952-7-14" )
> cal.yr( as.Date("1952-7-14") )
> cal.yr( "1952-7-14" )
```

The function will also find all date-variabels in a dataframe and convert them; try:

```
> data( diet )
> str( diet )
> str( cal.yr(diet) )
```

Chapter 7

Follow-up data in the Epi package

In the Epi-package, follow-up data is represented by adding some extra variables to a dataframe. Such a dataframe is called a Lexis object. The tools for handling follow-up data then use the structure of this for special plots, tabulations etc.

Follow-up data basically consists of a time of entry, a time of exit and an indication of the status at exit (normally either "alive" or "dead"). Implicitly is also assumed a status *during* the follow-up (usually "alive").

7.1 Timescales

A timescale is a variable that varies deterministically *within* each person during follow-up, *e.g.*:

- Age
- Calendar time
- Time since treatment
- Time since relapse

All timescales advance at the same pace, so the time followed is the same on all timescales. Therefore, it suffices to use only the entry point on each of the time scale, for example:

- Age at entry.
- Date of entry.
- Time since treatment (*at* treatment this is 0).
- Time since relapse (*at* relapse this is 0)..

In the Epi package, follow-up in a cohort is represented in a Lexis object. A Lexis object is a dataframe with a bit of extra structure representing the follow-up. For the nickel data we would construct a Lexis object by:

```
> data( nickel )
> nicL <- Lexis( entry = list( per=agein+dob,
+                              age=agein,
+                              tfh=agein-age1st ),
+                 exit = list( age=ageout ),
+          exit.status = ( icd %in% c(162,163) )*1,
+                 data = nickel )
```

The `entry` argument is a *named* list with the entry points on each of the timescales we want to use. It defines the names of the timescales and the entry points. The `exit` argument gives the exit time on *one* of the timescales, so the name of the element in this list must match one of the names of the `entry` list. This is sufficient, because the follow-up time on all time scales is the same, in this case `ageout - agein`. Now take a look at the result:

```
> str( nickel )
> str( nicL )
> head( nicL )
> summary( nicL )
```

The `Lexis` object `nicL` has a variable for each timescale which is the entry point on this timescale. The follow-up time is in the variable `lex.dur` (duration).

We defined the exit status to be death from lung cancer (ICD7 162,163), i.e. this variable is 1 if follow-up ended with a death from this cause. If follow-up ended alive or by death from another cause, the exit status is coded 0, i.e. as a censoring.

Note that the exit status is in the variable `lex.Xst` (eXit status. The variable `lex.Cst` is the state where the follow-up takes place (Current status), in this case 0 (alive).

It is possible to get a visualization of the follow-up along the timescales chosen by using the `plot` method for `Lexis` objects. `nicL` is an object of *class* `Lexis`, so using the function `plot()` on it means that R will look for the function `plot.Lexis` and use this function.

```
> plot( nicL )
```

The function allows a lot of control over the output, and a `points.Lexis` function allows plotting of the endpoints of follow-up.

```
> par( mar=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
> plot( nicL, 1:2, lwd=1, col=c("blue","red")[(nicL$exp>0)+1],
+       grid=TRUE, lty.grid=1, col.grid=gray(0.7),
+       xlim=1900+c(0,90), xaxs="i",
+       ylim=  10+c(0,90), yaxs="i", las=1 )
> points( nicL, 1:2, pch=c(NA,3)[nicL$lex.Xst+1],
+         col="lightgray", lwd=3, cex=1.2 )
> points( nicL, 1:2, pch=c(NA,3)[nicL$lex.Xst+1],
+         col=c("blue","red")[(nicL$exp>0)+1], lwd=1, cex=1.2 )
```

If you want to learn a bit more about drawing Lexis diagrams, you can take a look at the example shown on the help page for the dataset `occup`. One way to run the code is to say:

```
> example(occup)
```

7.2 Splitting the follow-up time along a timescale

The follow-up time in a cohort can be subdivided by for example current age. This is achieved by the `splitLexis` (note that it is *not* called `split.Lexis`). This requires that the timescale and the breakpoints on this timescale are supplied. Try:

```
> nicS1 <- splitLexis( nicL, "age", breaks=seq(0,100,10) )
> str( nicL )
```

```
Classes 'Lexis' and 'data.frame':       679 obs. of  14 variables:
 $ per     : num  1934 1934 1934 1934 1934 ...
 $ age     : num  45.2 48.3 53 47.9 54.7 ...
 $ tfh     : num  27.7 25.1 27.7 23.2 24.8 ...
 $ lex.dur : num  47.75 15 1.17 21.77 22.1 ...
 $ lex.Cst : num  0 0 0 0 0 0 0 0 0 0 ...
 $ lex.Xst : num  0 1 1 0 0 1 0 0 0 0 ...
 $ lex.id  : int  1 2 3 4 5 6 7 8 9 10 ...
 $ id      : num  3 4 6 8 9 10 15 16 17 18 ...
 $ icd     : num  0 162 163 527 150 163 334 160 420 12 ...
 $ exposure: num  5 5 10 9 0 2 0 0.5 0 0 ...
 $ dob     : num  1889 1886 1881 1886 1880 ...
 $ age1st  : num  17.5 23.2 25.2 24.7 30 ...
 $ agein   : num  45.2 48.3 53 47.9 54.7 ...
 $ ageout  : num  93 63.3 54.2 69.7 76.8 ...
 - attr(*, "time.scales")= chr  "per" "age" "tfh"
 - attr(*, "breaks")=List of 3
  ..$ per: NULL
  ..$ age: NULL
  ..$ tfh: NULL
```

```
> str( nicS1 )
```

```
Classes 'Lexis' and 'data.frame':       2210 obs. of  14 variables:
 $ lex.id  : int  1 1 1 1 1 1 2 2 2 3 ...
 $ per     : num  1934 1939 1949 1959 1969 ...
 $ age     : num  45.2 50 60 70 80 ...
 $ tfh     : num  27.7 32.5 42.5 52.5 62.5 ...
 $ lex.dur : num  4.77 10 10 10 10 ...
 $ lex.Cst : num  0 0 0 0 0 0 0 0 0 0 ...
 $ lex.Xst : num  0 0 0 0 0 0 0 0 1 1 ...
```

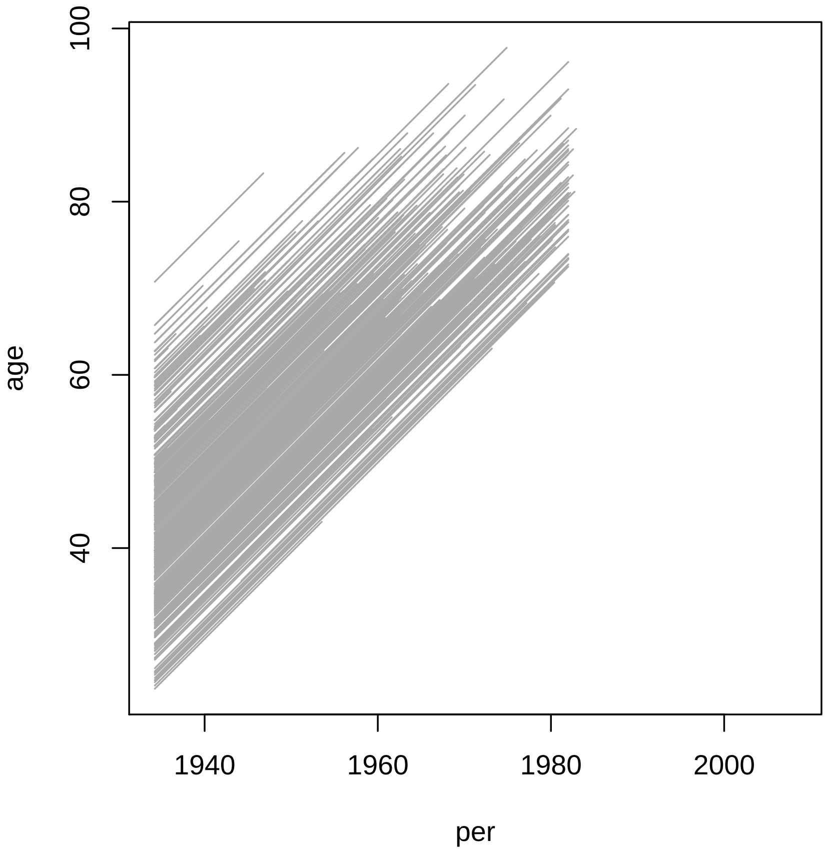


Figure 7.1: *Lexis diagram of the `nickel` dataset.*

```
 $ id      : num  3 3 3 3 3 3 4 4 4 6 ...
 $ icd     : num  0 0 0 0 0 0 162 162 162 163 ...
 $ exposure: num  5 5 5 5 5 5 5 5 5 10 ...
 $ dob     : num  1889 1889 1889 1889 1889 ...
 $ age1st  : num  17.5 17.5 17.5 17.5 17.5 ...
 $ agein   : num  45.2 45.2 45.2 45.2 45.2 ...
 $ ageout  : num  93 93 93 93 93 ...
 - attr(*, "breaks")=List of 3
  ..$ per: NULL
  ..$ age: num  0 10 20 30 40 50 60 70 80 90 ...
  ..$ tfh: NULL
 - attr(*, "time.scales")= chr  "per" "age" "tfh"
```

```
> round( subset( nicS1, id %in% 8:10 ), 2 )
```

```
   lex.id     per   age   tfh lex.dur lex.Cst lex.Xst id icd exposure     dob age1st agein
11      4 1934.25 47.91 23.19    2.09       0       0  8 527        9 1886.34  24.72 47.91
12      4 1936.34 50.00 25.28   10.00       0       0  8 527        9 1886.34  24.72 47.91
13      4 1946.34 60.00 35.28    9.68       0       0  8 527        9 1886.34  24.72 47.91
14      5 1934.25 54.75 24.79    5.25       0       0  9 150        0 1879.50  29.96 54.75
15      5 1939.50 60.00 30.04   10.00       0       0  9 150        0 1879.50  29.96 54.75
16      5 1949.50 70.00 40.04    6.84       0       0  9 150        0 1879.50  29.96 54.75
```

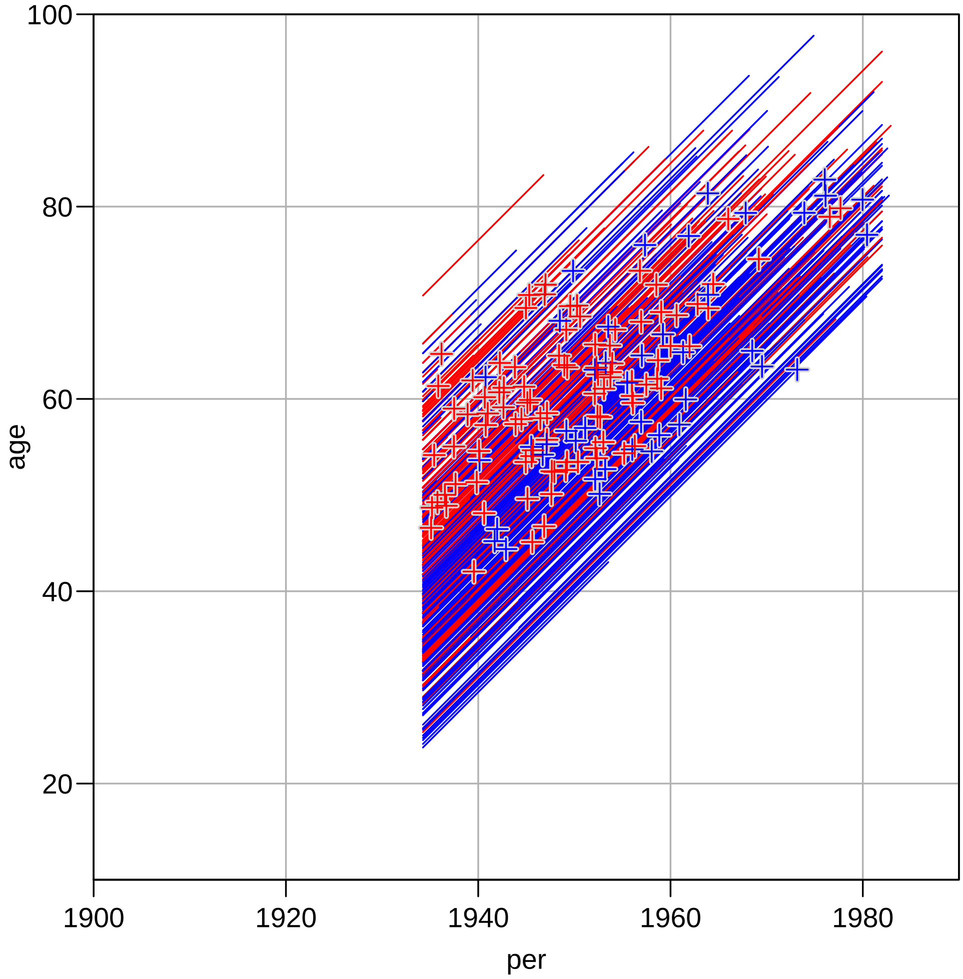


Figure 7.2: *Lexis diagram of the* **nickel** *dataset, with bells and whistles. The red lines are for persons with exposure*> 0*, so it is pretty evident that the oldest ones are the exposed part of the cohort.*

```
17      6 1934.25 44.33 23.04    5.67       0       0 10 163        2 1889.91  21.29 44.33
18      6 1939.91 50.00 28.71   10.00       0       0 10 163        2 1889.91  21.29 44.33
19      6 1949.91 60.00 38.71    2.54       0       1 10 163        2 1889.91  21.29 44.33
   ageout
11  69.68
12  69.68
13  69.68
14  76.84
15  76.84
16  76.84
17  62.54
18  62.54
19  62.54
```

The resulting object is again a `Lexis` object, and so follow-up may be split further along another timescale. Try this and list the result for individuals 4 and 6:

```
> nicS2 <- splitLexis( nicS1, "tfh", breaks=c(0,1,5,10,20,30,100) )
> round( subset( nicS2, id %in% 8:10 ), 2 )
```

```
   lex.id     per   age   tfh lex.dur lex.Cst lex.Xst id icd exposure     dob age1st agein
13      4 1934.25 47.91 23.19    2.09       0       0  8 527        9 1886.34  24.72 47.91
14      4 1936.34 50.00 25.28    4.72       0       0  8 527        9 1886.34  24.72 47.91
15      4 1941.06 54.72 30.00    5.28       0       0  8 527        9 1886.34  24.72 47.91
16      4 1946.34 60.00 35.28    9.68       0       0  8 527        9 1886.34  24.72 47.91
17      5 1934.25 54.75 24.79    5.21       0       0  9 150        0 1879.50  29.96 54.75
18      5 1939.46 59.96 30.00    0.04       0       0  9 150        0 1879.50  29.96 54.75
19      5 1939.50 60.00 30.04   10.00       0       0  9 150        0 1879.50  29.96 54.75
20      5 1949.50 70.00 40.04    6.84       0       0  9 150        0 1879.50  29.96 54.75
21      6 1934.25 44.33 23.04    5.67       0       0 10 163        2 1889.91  21.29 44.33
22      6 1939.91 50.00 28.71    1.29       0       0 10 163        2 1889.91  21.29 44.33
23      6 1941.20 51.29 30.00    8.71       0       0 10 163        2 1889.91  21.29 44.33
24      6 1949.91 60.00 38.71    2.54       0       1 10 163        2 1889.91  21.29 44.33
   ageout
13  69.68
14  69.68
15  69.68
16  69.68
17  76.84
18  76.84
19  76.84
20  76.84
21  62.54
22  62.54
23  62.54
24  62.54
```

If we want to model the effect of these timescales we will for each interval use either the value of the left endpoint in each interval or the middle. There is a function `timeBand` which returns these. Try:

```
> timeBand( nicS2, "age", "middle" )[1:10]
```

Note that these are the midpoints of the intervals defined by `breaks=`, *not* the midpoints of the actual follow-up intervals. This is because the variable to be used in modeling must be independent of the censoring and mortality pattern — it should only depend on the chosen grouping of the timescale.

7.3 Cutting time at a specific date

If we have a recording of the date of a specific event as for example recovery or relapse, we may classify follow-up time as being before or after this intermediate event. This is achieved with the function `cutLexis`, which takes three arguments: the time point, the timescale, and the name of the (new) state following the date.

Now we define the age for the nickel workers where the cumulative exposure exceeds 50 exposure years:

```
> subset( nicL, id %in% 8:10 )

       per     age     tfh lex.dur lex.Cst lex.Xst lex.id id icd exposure      dob  age1st
4 1934.246 47.9067 23.1861 21.7727       0       0      4  8 527        9 1886.340 24.7206
5 1934.246 54.7465 24.7890 22.0977       0       0      5  9 150        0 1879.500 29.9575
6 1934.246 44.3314 23.0437 18.2099       0       1      6 10 163        2 1889.915 21.2877
    agein  ageout
4 47.9067 69.6794
5 54.7465 76.8442
6 44.3314 62.5413
```

```
> agehi <- nicL$age1st + 50/nicL$exposure
> nicC <- cutLexis( data=nicL, cut=agehi, timescale="age",
+                   new.state=2, precursor.states=0 )
> subset( nicC[order(nicC$id,nicC$age),], id %in% 8:10 )

          per     age     tfh lex.dur lex.Cst lex.Xst lex.id id icd exposure      dob  age1st
4100 1934.246 47.9067 23.1861 21.7727       2       2      4  8 527        9 1886.340 24.7206
5    1934.246 54.7465 24.7890 22.0977       0       0      5  9 150        0 1879.500 29.9575
6    1934.246 44.3314 23.0437  1.9563       0       2      6 10 163        2 1889.915 21.2877
680  1936.203 46.2877 25.0000 16.2536       2       1      6 10 163        2 1889.915 21.2877
       agein  ageout
4100 47.9067 69.6794
5    54.7465 76.8442
6    44.3314 62.5413
680  44.3314 62.5413
```

(The `precursor.states=` argument is explained below). Note that individual 6 has had his follow-up split at age 25 where 50 exposure-years were attained. This could also have been achieved in the split dataset `nicS2` instead of `nicL`, try:

```
> subset( nicS2, id %in% 8:10 )

   lex.id      per     age     tfh lex.dur lex.Cst lex.Xst id icd exposure      dob  age1st
13      4 1934.246 47.9067 23.1861  2.0933       0       0  8 527        9 1886.340 24.7206
14      4 1936.340 50.0000 25.2794  4.7206       0       0  8 527        9 1886.340 24.7206
15      4 1941.060 54.7206 30.0000  5.2794       0       0  8 527        9 1886.340 24.7206
16      4 1946.340 60.0000 35.2794  9.6794       0       0  8 527        9 1886.340 24.7206
17      5 1934.246 54.7465 24.7890  5.2110       0       0  9 150        0 1879.500 29.9575
18      5 1939.457 59.9575 30.0000  0.0425       0       0  9 150        0 1879.500 29.9575
19      5 1939.500 60.0000 30.0425 10.0000       0       0  9 150        0 1879.500 29.9575
20      5 1949.500 70.0000 40.0425  6.8442       0       0  9 150        0 1879.500 29.9575
21      6 1934.246 44.3314 23.0437  5.6686       0       0 10 163        2 1889.915 21.2877
22      6 1939.915 50.0000 28.7123  1.2877       0       0 10 163        2 1889.915 21.2877
23      6 1941.203 51.2877 30.0000  8.7123       0       0 10 163        2 1889.915 21.2877
24      6 1949.915 60.0000 38.7123  2.5413       0       1 10 163        2 1889.915 21.2877
     agein  ageout
13 47.9067 69.6794
14 47.9067 69.6794
15 47.9067 69.6794
16 47.9067 69.6794
17 54.7465 76.8442
18 54.7465 76.8442
19 54.7465 76.8442
20 54.7465 76.8442
21 44.3314 62.5413
22 44.3314 62.5413
23 44.3314 62.5413
24 44.3314 62.5413
```

```
> agehi <- nicS2$age1st + 50/nicS2$exposure
> nicS2C <- cutLexis( data=nicS2, cut=agehi, timescale="age",
+                     new.state=2, precursor.states=0 )
> subset( nicS2C[order(nicS2C$id,nicS2C$age),], id %in% 8:10 )
```

```
     lex.id      per     age     tfh lex.dur lex.Cst lex.Xst id icd exposure      dob  age1st
3142      4 1934.246 47.9067 23.1861  2.0933       2       2  8 527        9 1886.340 24.7206
3143      4 1936.340 50.0000 25.2794  4.7206       2       2  8 527        9 1886.340 24.7206
3144      4 1941.060 54.7206 30.0000  5.2794       2       2  8 527        9 1886.340 24.7206
3145      4 1946.340 60.0000 35.2794  9.6794       2       2  8 527        9 1886.340 24.7206
17        5 1934.246 54.7465 24.7890  5.2110       0       0  9 150        0 1879.500 29.9575
18        5 1939.457 59.9575 30.0000  0.0425       0       0  9 150        0 1879.500 29.9575
19        5 1939.500 60.0000 30.0425 10.0000       0       0  9 150        0 1879.500 29.9575
20        5 1949.500 70.0000 40.0425  6.8442       0       0  9 150        0 1879.500 29.9575
21        6 1934.246 44.3314 23.0437  1.9563       0       2 10 163        2 1889.915 21.2877
3150      6 1936.203 46.2877 25.0000  3.7123       2       2 10 163        2 1889.915 21.2877
3151      6 1939.915 50.0000 28.7123  1.2877       2       2 10 163        2 1889.915 21.2877
3152      6 1941.203 51.2877 30.0000  8.7123       2       2 10 163        2 1889.915 21.2877
3153      6 1949.915 60.0000 38.7123  2.5413       2       1 10 163        2 1889.915 21.2877
       agein  ageout
3142 47.9067 69.6794
3143 47.9067 69.6794
3144 47.9067 69.6794
3145 47.9067 69.6794
17   54.7465 76.8442
18   54.7465 76.8442
19   54.7465 76.8442
20   54.7465 76.8442
21   44.3314 62.5413
3150 44.3314 62.5413
3151 44.3314 62.5413
3152 44.3314 62.5413
3153 44.3314 62.5413
```

```
> summary( nicS2C )
```

```
Transitions:
     To
From     0   1    2  Records:  Events: Risk time:  Persons:
  0   2043  65   74      2182      139   10772.53       466
  2      0  72  949      1021       72    4575.52       296
  Sum 2043 137 1023      3203      211   15348.06       679

Rates:
     To
From 0    1    2 Total
   0 0 0.01 0.01  0.01
   2 0 0.02 0.00  0.02
```

Note that follow-up subsequent to the event is classified as being in state 2, but that the final transition to state 1 (death from lung cancer) is preserved. This is the point of the `precursor.states=` argument. It names the states (in this case 0, “Alive”) that will be over-written by `new.state` (in this case 2, “High exposure”). Clearly, state 1 (“Dead”) should not be updated even if it is after the time where the persons moves to state 2. On other words, only state 0 is a precursor to state 2, state 1 is always subsequent to state 2.

Note if the intermediate event is to be used as a time-dependent variable in a Cox-model, then `lex.Cst` should be used as the time-dependent variable, and `lex.Xst==1` as the event.

It is possible to illustrate the transitions between the different states by the command `boxes.Lexis` — if you omit `boxpos=TRUE`, you will be asked to click on the screen to locate the boxes.

```
> boxes( nicS2C, boxpos=TRUE )
```

7.4 Competing risks — multiple types of events

If we want to consider death from lung cancer and death from other causes as separate events we can code these as for example 1 and 2.

```
> data( nickel )
> nicL <- Lexis( entry = list( per=agein+dob,
+                              age=agein,
+                              tfh=agein-age1st ),
+                 exit = list( age=ageout ),
+          exit.status = ( icd > 0 ) + ( icd %in% c(162,163) ),
+                 data = nickel )
> str( nicL )
> head( nicL )
> subset( nicL, id %in% 8:10 )
```

If we want to label the states, we can enter the names of these in the `states` parameter, try for example:

```
> nicL <- Lexis( entry = list( per=agein+dob,
+                              age=agein,
+                              tfh=agein-age1st ),
+                 exit = list( age=ageout ),
+          exit.status = ( icd > 0 ) + ( icd %in% c(162,163) ),
+                 data = nickel,
+               states = c("Alive","D.oth","D.lung") )
> str( nicL )
```

You can get an overview of the number of records by state and transitions between states as well as the person-years in each state by using `summary.Lexis()`, and computing rates:

```
> summary( nicL, scale=1000 )
```

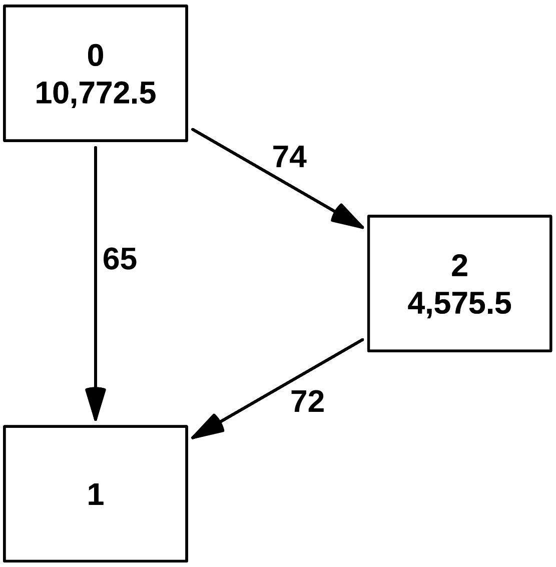


Figure 7.3: *The persons years (in the boxes) and number of transitions between the states.*

When we cut at a date as in this case, the date where cumulative exposure exceeds 50 exposure-years, we get the follow-up *after* the date classified as being in the new state if the exit (`lex.Xst`) was to a state we defined as one of the `precursor.states`:

```
> nicL$agehi <- nicL$age1st + 50/nicL$exposure
> nicC <- cutLexis( data=nicL, cut=nicL$agehi, "age",
+                   new.state="HiExp", precursor.states="Alive" )
> subset( nicC, id %in% 8:10 )
> summary( nicC, scale=1000 )
```

Note that the persons-years is the same, but that the number of events has changed. This is because events are now defined as any transition from alive, including the transitions to `HiExp`.

As before we can illustrate the different states with little boxes:

```
> boxes( nicC, boxpos=TRUE )
```

7.5 Multiple events of the same type (recurrent events)

Sometimes more events of the same type are recorded for each person and one would then like to count these and put follow-up time in states accordingly. So states must be *numbered.* Essentially, each set of cutpoints represents progressions from one state to the next. Therefore the states should be numbered, and the numbering of states subsequently occupied be increased accordingly.

This is a behaviour different from the one outlined above, and it is achieved by the argument `count=TRUE` to `cutLexis`. When `count` is set to `TRUE`, the value of the arguments `new.state` and

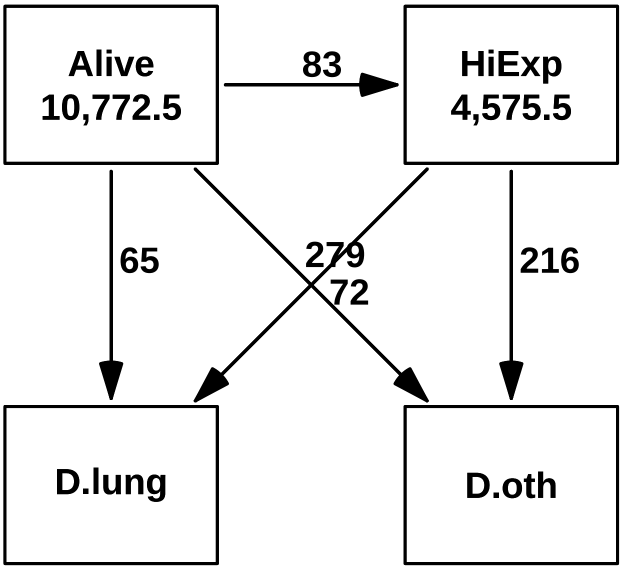


Figure 7.4: *The persons years (in the boxes) and number of transitions between states in the competing risks model.*

precursor.states are ignored. Actually, when using the argument count=TRUE, the function countLexis is called, so an alternative is to use this directly.

If we record when persons pass thresholds of exposure we have this situation. But if we at the same time want to keep track of when people die, we must code death by a sufficiently large number, because all states will be increased by one for each event:

```
> nicL <- Lexis( entry = list( per=agein+dob,
+                              age=agein,
+                              tfh=agein-age1st ),
+                 exit = list( age=ageout ),
+          exit.status = ( icd > 0 )*100,
+                 data = nickel )
> summary( nicL )

Transitions:
     To
From  0 100  Records:  Events: Risk time:  Persons:
   0 47 632       679      632   15348.06       679

Rates:
     To
From 0  100 Total
   0 0 0.04  0.04
```

We now cut the follow-up at successive exposure thresholds — note that we go through the levsle (*i.e.* the times at which they are crossed) by going throught them in random order (sample.int(x) returns a random permutation of the numbers $1, \ldots, x$).

```
> nicC <- nicL
> exlev <- seq(20,140,40)
> for( level in exlev[sample.int(length(exlev))] )
+    {
+    agehi <- nicC$age1st + level/nicC$exposure
+    nicC <- cutLexis( data=nicC, cut=agehi, "age", count=TRUE )
+    }
> summary( nicC )
```

We can now plot these:

```
> nc <- length( table( nicC$lex.Cst ) )
> boxes( nicC, boxpos=list( x=rep( seq(5,95,,nc), 2 ),
+                           y=rep( c(80,20), each=nc) ) )
```

We can put a few extra bells and whistles on the graph, by redefining the names of the names of the states by first making them factors (using factorize), then by pasting the relevant pieces of text to it. Moreover we also ask that rates instead of no. transitions be shown.

```
> nicF <- factorize( nicC )
> xlev <- paste( c("<",rep("",nc-1)),
+                c(exlev[1],exlev),
+                c("",rep("-",nc-1)), sep="" )
> levels( nicF$lex.Cst ) <-
+ levels( nicF$lex.Xst ) <-
+ c( paste( "Cum.ex.\n", xlev, "\n" ),
+    paste( "Dead\n", xlev ) )
> levels( nicF$lex.Cst )

 [1] "Cum.ex.\n <20 \n"  "Cum.ex.\n 20- \n"  "Cum.ex.\n 60- \n"  "Cum.ex.\n 100- \n"
 [5] "Cum.ex.\n 140- \n" "Dead\n <20"        "Dead\n 20-"        "Dead\n 60-"
 [9] "Dead\n 100-"       "Dead\n 140-"
```

```
> boxes( nicF, boxpos=list( y=rep( c(80,20), each=nc),
+                           x=rep( seq(5,95,,nc), 2 ) ),
+              eq.ht=FALSE, hmult=1.5, scale.D=1000, pos=0.3 )
```

The resulting graphs are shown in figure 7.5. A more thorough explanation of the `Lexis` machinery and its practical use in modeling is given in the papers [1, 2].

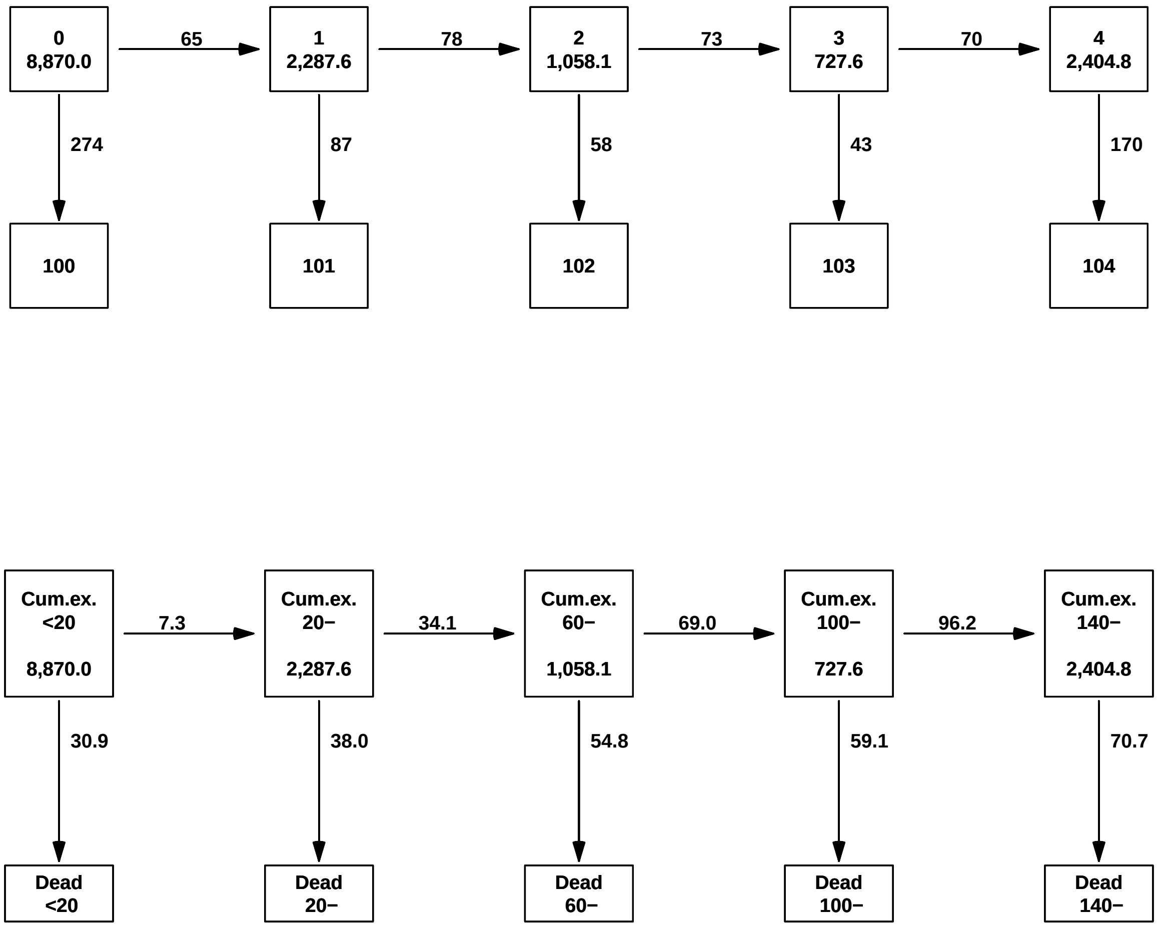


Figure 7.5: *The person years (in the boxes) and number of transitions between states in the counting model. The bottom display is enhanced by labeling of exposure levels, and showing the transition* rates *rather than the no. of transitions.*

Bibliography

[1] Martyn Plummer and Bendix Carstensen. Lexis: An R class for epidemiological studies with long-term follow-up. *Journal of Statistical Software*, 38(5):1–12, 1 2011.

[2] Bendix Carstensen and Martyn Plummer. Using Lexis objects for multi-state models in R. *Journal of Statistical Software*, 38(6):1–18, 1 2011.

Chapter 8

R command sheet

This R Reference Card is written by Tom Short, EPRI PEAC, `tshort@epri-peac.com`, 2004-10-21 and granted to the public domain. See `www.Rpad.org` for the source and latest version. Includes material from *R for Beginners* by Emmanuel Paradis (with permission).

It is also available separately as a 4-page landscape document from the R-hompage `www.r-project.org`, Manuals → contributed documentation.

Getting help

Most R functions have online documentation.

`help(topic)` documentation on `topic`

`?topic` — the same.

`help.search("topic")` search the help system

`apropos("topic")` the names of all objects in the search list matching the regular expression ”topic”

`help.start()` start the HTML version of help

`str(a)` display the internal *str*ucture of an R object

`summary(a)` gives a “summary” of `a`, usually a statistical summary but it is *generic* meaning it has different operations for different classes of `a`

`ls()` show objects in the search path; specify `pat="pat"` to search on a pattern

`ls.str()` str() for each variable in the search path

`dir()` show files in the current directory

`methods(a)` shows S3 methods of `a`

`methods(class=class(a))` lists all the methods to handle objects of class `a`.

Input and output

`load()` load the datasets written with `save`

`data(x)` loads specified data sets

`library(x)` load add-on packages

`read.table(file)` reads a file in table format and creates a data frame from it; the default separator `sep=""` is any whitespace; use `header=TRUE` to read the first line as a header of column names; use `as.is=TRUE` to prevent character vectors from being converted to factors; use `comment.char=""` to prevent `"#"` from being interpreted as a comment; use `skip=n` to skip `n` lines before reading data; see the help for options on row naming, NA treatment, and others

`read.csv("filename",header=TRUE)` id. but with defaults set for reading comma-delimited files

`read.delim("filename",header=TRUE)` id. but with defaults set for reading tab-delimited files

`read.fwf(file,widths,header=FALSE,sep="",as.is=FALSE)` read a table of *f*ixed *w*idth *f*ormatted data into a ’data.frame’; `widths` is an integer vector, giving the widths of the fixed-width fields

`save(file,...)` saves the specified objects (...) in the XDR platform-independent binary format

`save.image(file)` saves all objects

`cat(..., file="", sep=" ")` prints the arguments after coercing to character; `sep` is the character separator between arguments

`print(a, ...)` prints its arguments; generic, meaning it can have different methods for different objects

`format(x,...)` format an R object for pretty printing

`write.table(x,file="",row.names=TRUE,col.names=TRUE, sep=" ")` prints `x` after converting to a data frame; if `quote` is `TRUE`, character or factor columns are surrounded by quotes (`"`); `sep` is the field separator; `eol` is the end-of-line separator; `na` is the string for missing values; use `col.names=NA` to add a blank column header to get the column headers aligned correctly for spreadsheet input

`sink(file)` output to `file`, until `sink()`

Most of the I/O functions have a `file` argument. This can often be a character string naming a file or a connection. `file=""` means the standard input or output. Connections can include files, pipes, zipped files, and R variables.

On windows, the file connection can also be used with `description = "clipboard"`. To read a table copied from Excel, use

```
x <- read.delim("clipboard")
```

To write a table to the clipboard for Excel, use

```
write.table(x,"clipboard",sep="\t",col.names=NA)
```

For database interaction, see packages `RODBC`, `DBI`, `RMySQL`, `RPgSQL`, and `ROracle`. See packages `XML`, `hdf5`, `netCDF` for reading other file formats.

Data creation

`c(...)` generic function to combine arguments with the default forming a vector; with `recursive=TRUE` descends through lists combining all elements into one vector

`from:to` generates a sequence; ":" has operator priority; 1:4 + 1 is "2,3,4,5"

`seq(from,to)` generates a sequence `by=` specifies increment; `length=` specifies desired length

`seq(along=x)` generates `1, 2, ..., length(along)`; useful for `for` loops

`rep(x,times)` replicate `x times`; use `each=` to repeat "each" element of `x each` times; `rep(c(1,2,3),2)` is 1 2 3 1 2 3; `rep(c(1,2,3),each=2)` is 1 1 2 2 3 3

`data.frame(...)` create a data frame of the named or unnamed arguments; `data.frame(v=1:4,ch=c("a","B","c","d"),n=10)`; shorter vectors are recycled to the length of the longest

`list(...)` create a list of the named or unnamed arguments; `list(a=c(1,2),b="hi",c=3i)`;

`array(x,dim=)` array with data `x`; specify dimensions like `dim=c(3,4,2)`; elements of `x` recycle if `x` is not long enough

`matrix(x,nrow=,ncol=)` matrix; elements of `x` recycle

`factor(x,levels=)` encodes a vector `x` as a factor

`gl(n,k,length=n*k,labels=1:n)` generate levels (factors) by specifying the pattern of their levels; `k` is the number of levels, and `n` is the number of replications

`expand.grid()` a data frame from all combinations of the supplied vectors or factors

`rbind(...)` combine arguments by rows for matrices, data frames, and others

`cbind(...)` id. by columns

Slicing and extracting data

Indexing vectors

`x[n]`	`n`th element
`x[-n]`	all *but* the `n`th element
`x[1:n]`	first `n` elements
`x[-(1:n)]`	elements from `n+1` to the end
`x[c(1,4,2)]`	specific elements
`x["name"]`	element named `"name"`
`x[x > 3]`	all elements greater than 3
`x[x > 3 & x < 5]`	all elements between 3 and 5
`x[x %in% c("a","and","the")]`	elements in the given set

Indexing lists

`x[n]`	list with elements `n`
`x[[n]]`	`n`th element of the list
`x[["name"]]`	element of the list named `"name"`
`x$name`	id.

Indexing matrices

`x[i,j]`	element at row `i`, column `j`
`x[i,]`	row `i`
`x[,j]`	column `j`
`x[,c(1,3)]`	columns 1 and 3
`x["name",]`	row named `"name"`

Indexing data frames (matrix indexing plus the following)

`x[["name"]]`	column named `"name"`
`x$name`	id.

Variable conversion

`as.array(x), as.data.frame(x), as.numeric(x), as.logical(x), as.complex(x), as.character(x), ...` convert type; for a complete list, use `methods(as)`

Variable information

`is.na(x), is.null(x), is.array(x), is.data.frame(x), is.numeric(x), is.complex(x), is.character(x), ...` test for type; for a complete list, use `methods(is)`

`length(x)` number of elements in `x`

`dim(x)` Retrieve or set the dimension of an object; `dim(x) <- c(3,2)`

`dimnames(x)` Retrieve or set the dimension names of an object

`nrow(x)` number of rows; `NROW(x)` is the same but treats a vector as a one-row matrix

`ncol(x)` and `NCOL(x)` id. for columns

`class(x)` get or set the class of `x`; `class(x) <- "myclass"`

`unclass(x)` remove the class attribute of `x`

`attr(x,which)` get or set the attribute `which` of `x`

`attributes(obj)` get or set the list of attributes of `obj`

Data selection and manipulation

`which.max(x)` returns the index of the greatest element of `x`
`which.min(x)` returns the index of the smallest element of `x`
`rev(x)` reverses the elements of `x`
`sort(x)` sorts the elements of `x` in increasing order; to sort in decreasing order: `rev(sort(x))`
`cut(x,breaks)` divides `x` into intervals (factors); `breaks` is the number of cut intervals or a vector of cut points
`match(x, y)` returns a vector of the same length than `x` with the elements of `x` which are in `y` (`NA` otherwise)
`which(x == a)` returns a vector of the indices of `x` if the comparison operation is true (`TRUE`), in this example the values of `i` for which `x[i] == a` (the argument of this function must be a variable of mode logical)
`choose(n, k)` computes the combinations of k events among n repetitions = $n!/[(n-k)!k!]$
`na.omit(x)` suppresses the observations with missing data (`NA`) (suppresses the corresponding line if `x` is a matrix or a data frame)
`na.fail(x)` returns an error message if `x` contains at least one `NA`
`unique(x)` if `x` is a vector or a data frame, returns a similar object but with the duplicate elements suppressed
`table(x)` returns a table with the numbers of the differents values of `x` (typically for integers or factors)
`subset(x, ...)` returns a selection of `x` with respect to criteria (`...`, typically comparisons: `x$V1 < 10`); if `x` is a data frame, the option `select` gives the variables to be kept or dropped using a minus sign
`sample(x, size)` resample randomly and without replacement `size` elements in the vector `x`, the option `replace = TRUE` allows to resample with replacement
`prop.table(x,margin=)` table entries as fraction of marginal table

Math

`sin,cos,tan,asin,acos,atan,atan2,log,log10,exp`
`max(x)` maximum of the elements of `x`
`min(x)` minimum of the elements of `x`
`range(x)` id. then `c(min(x), max(x))`
`sum(x)` sum of the elements of `x`
`diff(x)` lagged and iterated differences of vector `x`
`prod(x)` product of the elements of `x`
`mean(x)` mean of the elements of `x`
`median(x)` median of the elements of `x`
`quantile(x,probs=)` sample quantiles corresponding to the given probabilities (defaults to 0,.25,.5,.75,1)
`weighted.mean(x, w)` mean of `x` with weights `w`
`rank(x)` ranks of the elements of `x`
`var(x)` or `cov(x)` variance of the elements of `x` (calculated on $n-1$); if `x` is a matrix or a data frame, the variance-covariance matrix is calculated
`sd(x)` standard deviation of `x`
`cor(x)` correlation matrix of `x` if it is a matrix or a data frame (1 if `x` is a vector)
`var(x, y)` or `cov(x, y)` covariance between `x` and `y`, or between the columns of `x` and those of `y` if they are matrices or data frames
`cor(x, y)` linear correlation between `x` and `y`, or correlation matrix if they are matrices or data frames
`round(x, n)` rounds the elements of `x` to `n` decimals
`log(x, base)` computes the logarithm of `x` with base `base`
`scale(x)` if `x` is a matrix, centers and reduces the data; to center only use the option `center=FALSE`, to reduce only `scale=FALSE` (by default `center=TRUE, scale=TRUE`)
`pmin(x,y,...)` a vector which ith element is the minimum of `x[i]`, `y[i]`, ...
`pmax(x,y,...)` id. for the maximum
`cumsum(x)` a vector which ith element is the sum from `x[1]` to `x[i]`
`cumprod(x)` id. for the product
`cummin(x)` id. for the minimum
`cummax(x)` id. for the maximum
`union(x,y)`, `intersect(x,y)`, `setdiff(x,y)`, `setequal(x,y)`, `is.element(el,set)` "set" functions
`Re(x)` real part of a complex number
`Im(x)` imaginary part
`Mod(x)` modulus; `abs(x)` is the same
`Arg(x)` angle in radians of the complex number
`Conj(x)` complex conjugate
`convolve(x,y)` compute the several kinds of convolutions of two sequences
`fft(x)` Fast Fourier Transform of an array
`mvfft(x)` FFT of each column of a matrix
`filter(x,filter)` applies linear filtering to a univariate time series or to each series separately of a multivariate time series

Many math functions have a logical parameter `na.rm=FALSE` to specify missing data (NA) removal.

Matrices

`t(x)` transpose
`diag(x)` diagonal
`%*%` matrix multiplication
`solve(a,b)` solves `a %*% x = b` for `x`
`solve(a)` matrix inverse of `a`

`rowsum(x)` sum of rows for a matrix-like object; `rowSums(x)` is a faster version
`colsum(x)`, `colSums(x)` id. for columns
`rowMeans(x)` fast version of row means
`colMeans(x)` id. for columns

Advanced data processing

`apply(X,INDEX,FUN=)` a vector or array or list of values obtained by applying a function `FUN` to margins (`INDEX`) of `X`
`lapply(X,FUN)` apply `FUN` to each element of the list `X`
`tapply(X,INDEX,FUN=)` apply `FUN` to each cell of a ragged array given by `X` with indexes `INDEX`
`by(data,INDEX,FUN)` apply `FUN` to data frame `data` subsetted by `INDEX`
`merge(a,b)` merge two data frames by common columns or row names
`xtabs(a b,data=x)` a contingency table from cross-classifying factors
`aggregate(x,by,FUN)` splits the data frame `x` into subsets, computes summary statistics for each, and returns the result in a convenient form; `by` is a list of grouping elements, each as long as the variables in `x`
`stack(x, ...)` transform data available as separate columns in a data frame or list into a single column
`unstack(x, ...)` inverse of `stack()`
`reshape(x, ...)` reshapes a data frame between 'wide' format with repeated measurements in separate columns of the same record and 'long' format with the repeated measurements in separate records; use (direction="wide") or (direction="long")

Strings

`paste(...)` concatenate vectors after converting to character; `sep=` is the string to separate terms (a single space is the default); `collapse=` is an optional string to separate "collapsed" results
`substr(x,start,stop)` substrings in a character vector; can also assign, as `substr(x, start, stop) <- value`
`strsplit(x,split)` split `x` according to the substring `split`
`grep(pattern,x)` searches for matches to `pattern` within `x`; see `?regex`
`gsub(pattern,replacement,x)` replacement of matches determined by regular expression matching `sub()` is the same but only replaces the first occurrence.
`tolower(x)` convert to lowercase
`toupper(x)` convert to uppercase
`match(x,table)` a vector of the positions of first matches for the elements of `x` among `table`
`x %in% table` id. but returns a logical vector
`pmatch(x,table)` partial matches for the elements of `x` among `table`
`nchar(x)` number of characters

Dates and Times

The class `Date` has dates without times. `POSIXct` has dates and times, including time zones. Comparisons (e.g. >), `seq()`, and `difftime()` are useful. `Date` also allows + and −. `?DateTimeClasses` gives more information. See also package `chron`. `as.Date(s)` and `as.POSIXct(s)` convert to the respective class. `format(dt)` converts to a string representation. The default string format is "2001-02-21". These accept a second argument to specify a format for conversion. Some common formats are:

`%a`, `%A` Abbreviated and full weekday name.
`%b`, `%B` Abbreviated and full month name.
`%d` Day of the month (01–31).
`%H` Hours (00–23).
`%I` Hours (01–12).
`%j` Day of year (001–366).
`%m` Month (01–12).
`%M` Minute (00–59).
`%p` AM/PM indicator.
`%S` Second as decimal number (00–61).
`%U` Week (00–53); the first Sunday as day 1 of week 1.
`%w` Weekday (0–6, Sunday is 0).
`%W` Week (00–53); the first Monday as day 1 of week 1.
`%y` Year without century (00–99). Don't use.
`%Y` Year with century.
`%z` (output only.) Offset from Greenwich; `-0800` is 8 hours west of.
`%Z` (output only.) Time zone as a character string (empty if not available).

Where leading zeros are shown they will be used on output but are optional on input. See `?strftime`.

Plotting

`plot(x)` plot of the values of `x` (on the y-axis) ordered on the x-axis
`plot(x, y)` bivariate plot of `x` (on the x-axis) and `y` (on the y-axis)
`hist(x)` histogram of the frequencies of `x`
`barplot(x)` histogram of the values of `x`; use `horiz=FALSE` for horizontal bars
`dotplot(x)` if `x` is a data frame, plots a Cleveland dot plot (stacked plots line-by-line and column-by-column)
`piechart(x)` circular pie-chart
`boxplot(x)` "box-and-whiskers" plot

`sunflowerplot(x, y)` id. than `plot()` but the points with similar coordinates are drawn as flowers which petal number represents the number of points

`stripplot(x)` plot of the values of `x` on a line (an alternative to `boxplot()` for small sample sizes)

`coplot(x~y | z)` bivariate plot of `x` and `y` for each value or interval of values of `z`

`interaction.plot (f1, f2, y)` if `f1` and `f2` are factors, plots the means of `y` (on the y-axis) with respect to the values of `f1` (on the x-axis) and of `f2` (different curves); the option `fun` allows to choose the summary statistic of `y` (by default `fun=mean`)

`matplot(x,y)` bivariate plot of the first column of `x` *vs.* the first one of `y`, the second one of `x` *vs.* the second one of `y`, etc.

`fourfoldplot(x)` visualizes, with quarters of circles, the association between two dichotomous variables for different populations (`x` must be an array with `dim=c(2, 2, k)`, or a matrix with `dim=c(2, 2)` if $k = 1$)

`assocplot(x)` Cohen–Friendly graph showing the deviations from independence of rows and columns in a two dimensional contingency table

`mosaicplot(x)` ‘mosaic’ graph of the residuals from a log-linear regression of a contingency table. Also useful for graphical display of contingency tables.

`pairs(x)` if `x` is a matrix or a data frame, draws all possible bivariate plots between the columns of `x`

`plot.ts(x)` if `x` is an object of class `"ts"`, plot of `x` with respect to time, `x` may be multivariate but the series must have the same frequency and dates

`ts.plot(x)` id. but if `x` is multivariate the series may have different dates and must have the same frequency

`qqnorm(x)` quantiles of `x` with respect to the values expected under a normal law

`qqplot(x, y)` quantiles of `y` with respect to the quantiles of `x`

`contour(x, y, z)` contour plot (data are interpolated to draw the curves), `x` and `y` must be vectors and `z` must be a matrix so that `dim(z)=c(length(x), length(y))` (`x` and `y` may be omitted)

`filled.contour(x, y, z)` id. but the areas between the contours are coloured, and a legend of the colours is drawn as well

`image(x, y, z)` id. but with colours (actual data are plotted)

`persp(x, y, z)` id. but in perspective (actual data are plotted)

`stars(x)` if `x` is a matrix or a data frame, draws a graph with segments or a star where each row of `x` is represented by a star and the columns are the lengths of the segments

`symbols(x, y, ...)` draws, at the coordinates given by `x` and `y`, symbols (circles, squares, rectangles, stars, thermometres or “boxplots”) which sizes, colours ... are specified by supplementary arguments

`termplot(mod.obj)` plot of the (partial) effects of a regression model (`mod.obj`)

The following parameters are common to many plotting functions:

`add=FALSE` if `TRUE` superposes the plot on the previous one (if it exists)

`axes=TRUE` if `FALSE` does not draw the axes and the box

`type="p"` specifies the type of plot, `"p"`: points, `"l"`: lines, `"b"`: points connected by lines, `"o"`: id. but the lines are over the points, `"h"`: vertical lines, `"s"`: steps, the data are represented by the top of the vertical lines, `"S"`: id. but the data are represented by the bottom of the vertical lines

`xlim=, ylim=` specifies the lower and upper limits of the axes, for example with `xlim=c(1, 10)` or `xlim=range(x)`

`xlab=, ylab=` annotates the axes, must be variables of mode character

`main=` main title, must be a variable of mode character

`sub=` sub-title (written in a smaller font)

Low-level plotting commands

`points(x, y)` adds points (the option `type=` can be used)

`lines(x, y)` id. but with lines

`text(x, y, labels, ...)` adds text given by `labels` at coordinates (x,y); a typical use is: `plot(x, y, type="n"); text(x, y, names)`

`mtext(text, side=3, line=0, ...)` adds text given by `text` in the margin specified by `side` (see `axis()` below); `line` specifies the line from the plotting area

`segments(x0, y0, x1, y1)` draws lines from points (x0,y0) to points (x1,y1)

`arrows(x0, y0, x1, y1, angle= 30, code=2)` id. with arrows at points (x0,y0) if `code=2`, at points (x1,y1) if `code=1`, or both if `code=3`; `angle` controls the angle from the shaft of the arrow to the edge of the arrow head

`abline(a,b)` draws a line of slope `b` and intercept `a`

`abline(h=y)` draws a horizontal line at ordinate `y`

`abline(v=x)` draws a vertical line at abcissa `x`

`abline(lm.obj)` draws the regression line given by `lm.obj`

`rect(x1, y1, x2, y2)` draws a rectangle which left, right, bottom, and top limits are `x1`, `x2`, `y1`, and `y2`, respectively

`polygon(x, y)` draws a polygon linking the points with coordinates given by `x` and `y` `legend(x, y, legend)` adds the legend at the point (x,y) with the symbols given by `legend`

`title()` adds a title and optionally a sub-title

`axis(side, vect)` adds an axis at the bottom (`side=1`), on the left (2), at the top (3), or on the right (4); `vect` (optional) gives the abcissa (or ordinates) where tick-marks are drawn

`rug(x)` draws the data `x` on the x-axis as small vertical lines

`locator(n, type="n", ...)` returns the coordinates (x, y) after the user has clicked `n` times on the plot with the mouse; also draws symbols (`type="p"`) or lines (`type="l"`) with respect to optional graphic parameters (`...`); by default nothing is drawn (`type="n"`)

Graphical parameters

These can be set globally with `par(...)`; many can be passed as parameters to plotting commands.

`adj` controls text justification (`0` left-justified, `0.5` centred, `1` right-justified)

`bg` specifies the colour of the background (ex. : `bg="red"`, `bg="blue"`, . . . the list of the 657 available colours is displayed with `colors()`)

`bty` controls the type of box drawn around the plot, allowed values are: `"o"`, `"l"`, `"7"`, `"c"`, `"u"` ou `"]"` (the box looks like the corresponding character); if `bty="n"` the box is not drawn

`cex` a value controlling the size of texts and symbols with respect to the default; the following parameters have the same control for numbers on the axes, `cex.axis`, the axis labels, `cex.lab`, the title, `cex.main`, and the sub-title, `cex.sub`

`col` controls the color of symbols and lines; use color names: `"red"`, `"blue"` see `colors()` or as `"#RRGGBB"`; see `rgb()`, `hsv()`, `gray()`, and `rainbow()`; as for `cex` there are: `col.axis`, `col.lab`, `col.main`, `col.sub`

`font` an integer which controls the style of text (`1`: normal, `2`: italics, `3`: bold, `4`: bold italics); as for `cex` there are: `font.axis`, `font.lab`, `font.main`, `font.sub`

`las` an integer which controls the orientation of the axis labels (`0`: parallel to the axes, `1`: horizontal, `2`: perpendicular to the axes, `3`: vertical)

`lty` controls the type of lines, can be an integer or string (`1`: `"solid"`, `2`: `"dashed"`, `3`: `"dotted"`, `4`: `"dotdash"`, `5`: `"longdash"`, `6`: `"twodash"`, or a string of up to eight characters (between `"0"` and `"9"`) which specifies alternatively the length, in points or pixels, of the drawn elements and the blanks, for example `lty="44"` will have the same effect than `lty=2`

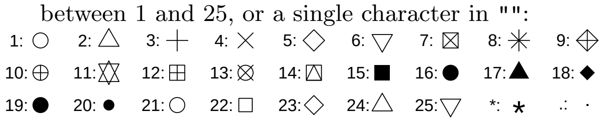
`lwd` a numeric which controls the width of lines, default `1`

`mar` a vector of 4 numeric values which control the space between the axes and the border of the graph of the form `c(bottom, left, top, right)`, the default values are `c(5.1, 4.1, 4.1, 2.1)`

`mfcol` a vector of the form `c(nr,nc)` which partitions the graphic window as a matrix of `nr` lines and `nc` columns, the plots are then drawn in columns

`mfrow` id. but the plots are drawn by row

`pch` controls the type of symbol, either an integer between 1 and 25, or a single character in `""`:



`ps` an integer which controls the size in points of texts and symbols

`pty` a character which specifies the type of the plotting region, `"s"`: square, `"m"`: maximal

`tck` a value which specifies the length of tick-marks on the axes as a fraction of the smallest of the width or height of the plot; if `tck=1` a grid is drawn

`tcl` a value which specifies the length of tick-marks on the axes as a fraction of the height of a line of text (by default `tcl=-0.5`)

`xaxt` if `xaxt="n"` the x-axis is set but not drawn (useful in conjonction with `axis(side=1, ...)`)

`yaxt` if `yaxt="n"` the y-axis is set but not drawn (useful in conjonction with `axis(side=2, ...)`)

Lattice (Trellis) graphics

`barchart(y~x)` histogram of the values of `y` with respect to those of `x`

`bwplot(y~x)` “box-and-whiskers” plot

`densityplot(~x)` density functions plot

`dotplot(y~x)` Cleveland dot plot (stacked plots line-by-line and column-by-column)

`histogram(~x)` histogram of the frequencies of `x`

`qqmath(~x)` quantiles of `x` with respect to the values expected under a theoretical distribution

`stripplot(y~x)` single dimension plot, `x` must be numeric, `y` may be a factor

`qq(y~x)` quantiles to compare two distributions, `x` must be numeric, `y` may be numeric, character, or factor but must have two ‘levels’

`xyplot(y~x)` bivariate plots (with many functionalities)

`levelplot(z~x*y)` coloured plot of the values of `z` at the coordinates given by `x` and `y` (`x`, `y` and `z` are all of the same length)

`splom(~x)` matrix of bivariate plots

`parallel(~x)` parallel coordinates plot

Optimization and model fitting

`optim(par, fn, method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN")` general-purpose optimization; `par` is initial values, `fn` is function to optimize (normally minimize)

`nlm(f,p)` minimize function `f` using a Newton-type algorithm with starting values `p`

`lm(formula)` fit linear models; `formula` is typically of the form `response  termA + termB + ...`; use `I(x*y) + I(x^2)` for terms made of nonlinear components

`glm(formula,family=)` fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution; `family` is a description of the error distribution and link function to be used in the model; see `?family`

`nls(formula)` nonlinear least-squares estimates of the nonlinear model parameters

`approx(x,y=)` linearly interpolate given data points; `x` can be an xy plotting structure

`spline(x,y=)` cubic spline interpolation

`loess(formula)` fit a polynomial surface using local fitting

Many of the formula-based modeling functions have several common arguments: `data=` the data frame for the formala variables, `subset=` a subset of variables used in the fit, `na.action=` action for missing values: `"na.fail"`, `"na.omit"`, or a function. The following generics often apply to model fitting functions:

`predict(fit,...)` predictions from `fit` based on input data

`df.residual(fit)` returns the number of residual degrees of freedom

`coef(fit)` returns the estimated coefficients (sometimes with their standard-errors)

`residuals(fit)` returns the residuals

`deviance(fit)` returns the deviance

`fitted(fit)` returns the fitted values

`logLik(fit)` computes the logarithm of the likelihood and the number of parameters

`AIC(fit)` computes the Akaike information criterion or AIC

Statistics

`aov(formula)` analysis of variance model

`anova(fit,...)` analysis of variance (or deviance) tables for one or more fitted model objects

`density(x)` kernel density estimates of `x`

`binom.test()`, `pairwise.t.test()`, `power.t.test()`, `prop.test()`, `t.test()`, ... use `help.search("test")`

Distributions

`rnorm(n, mean=0, sd=1)` Gaussian (normal)

`rexp(n, rate=1)` exponential

`rgamma(n, shape, scale=1)` gamma

`rpois(n, lambda)` Poisson

`rweibull(n, shape, scale=1)` Weibull

`rcauchy(n, location=0, scale=1)` Cauchy

`rbeta(n, shape1, shape2)` beta

`rt(n, df)` 'Student' (t)

`rf(n, df1, df2)` Fisher–Snedecor (F) (χ^2)

`rchisq(n, df)` Pearson

`rbinom(n, size, prob)` binomial

`rgeom(n, prob)` geometric

`rhyper(nn, m, n, k)` hypergeometric

`rlogis(n, location=0, scale=1)` logistic

`rlnorm(n, meanlog=0, sdlog=1)` lognormal

`rnbinom(n, size, prob)` negative binomial

`runif(n, min=0, max=1)` uniform

`rwilcox(nn, m, n)`, `rsignrank(nn, n)` Wilcoxon's statistics

All these functions can be used by replacing the letter `r` with `d`, `p` or `q` to get, respectively, the probability density (d*func*`(x, ...)`), the cumulative probability density (p*func*`(x, ...)`), and the value of quantile (q*func*`(p, ...)`, with $0 < p < 1$).

Programming

`function( arglist ) expr` function definition

`return(value)`

`if(cond) expr`

`if(cond) cons.expr else alt.expr`

`for(var in seq) expr`

`while(cond) expr`

`repeat expr`

`break`

`next`

Use braces {} around statements

`ifelse(test, yes, no)` a value with the same shape as `test` filled with elements from either `yes` or `no`

`do.call(funname, args)` executes a function call from the name of the function and a list of arguments to be passed to it.

The Epi package

The purpose of the `Epi` package is to provide tools for advanced epidemiological data manipulation and analysis. This section does *not* provide the full set of arguments for the functions, so please consult the help pages.

`Lexis(entry,exit,duration,enty.status, exit.status,id,data,merge,states)` Define a `Lexis` object with follow-up on several timescales (and possibly several types of events).

`plot.Lexis(),lines.Lexis(),points.Lexis()` Plot a Lexis diagram from a `Lexis` object, and add lines and points.

`splitLexis(lex,breaks,time.scale)` Split the follow-up time in a `Lexis` object along one time scale.

`cutLexis(data,cut,timescale)` Cut the follow-up at one specific point on a timescale.

`summary.Lexis(x)` Tabulate events and risk time from a `Lexis` object.

`boxes.Lexis(x)` Illustrate a multistate model, and show person.yeras and transitions.

`timeScales(), timeBand(), breaks()` Utilites to acces parts of a `Lexis` object.

`cal.yr(x,format)` Convert `x` to fractional calendar year.

`stat.table(index,contents,...)` Make tables, classified by `index`, of sums, ratios etc. given in `contents`.

`effx(response, type, exposure, ...)` Epidemiological estimates of effects.

`ci.lin(obj,ctr.mat,subset,diffs,Exp)` Extract parameters and linear functions of them from a model object.

`ci.cum(obj,ctr.mat,subset,intl,Exp)` Extract parameters and a model object and compute the cumulative sum.

`plotEst(ests,...)` Make a plot of parameter estimates.

`twoby2(exposure,outcome,...)` Analysis of a 2×2 table. Input can be either two binary variables or a matrix of counts.

More esoteric topics in the `Epi` package (look at the help pages for links):

`Icens()` Fit a model to interval censored follow-up data.

`apc.fit()` Fit age-period-cohort models to tabulated data.

Chapter 9

The `Epi` package

The following is a printout of the manual pages for the commands available in the `Epi` package.

Version 1.1.24

Date 2011-07-19

Title A package for statistical analysis in epidemiology.

Author Bendix Carstensen, Martyn Plummer, Esa Laara, Michael Hills et. al.

Maintainer Bendix Carstensen `<bxc@steno.dk>`

Depends utils

Suggests splines, nlme, survival, mstate, MASS

Description Functions for demographic and epidemiological analysis in the Lexis diagram, i.e. register and cohort follow-up data, including interval censored data and representation of multistate data. Also some useful functions for tabulation and plotting. Contains some epidemiological datasets.

License GPL-2

URL `http://www.pubhealth.ku.dk/~bxc/Epi/`

`apc.fit` *Fit an Age-Period-Cohort model to tabular data.*

Description

Fits the classical five models to tabulated rate data (cases, person-years) classified by two of age, period, cohort: Age, Age-drift, Age-Period, Age-Cohort and Age-period. There are no assumptions about the age, period or cohort classes being of the same length, or that tabulation should be only by two of the variables. Only requires that mean age and period for each tabulation unit is given.

Usage

```
apc.fit( data,
            A,
            P,
            D,
            Y,
        ref.c,
        ref.p,
          dist = c("poisson","binomial"),
          model = c("ns","bs","ls","factor"),
        dr.extr = c("weighted","Holford"),
           parm = c("ACP","APC","AdCP","AdPC","Ad-P-C","Ad-C-P","AC-P","AP-C"),
           npar = c( A=5, P=5, C=5 ),
          scale = 1,
          alpha = 0.05,
      print.AOV = TRUE )
```

Arguments

`data` Data frame with (at least) variables, `A` (age), `P` (period), `D` (cases, deaths) and `Y` (person-years). Cohort (date of birth) is computed as `P-A`. If thsi argument is given the arguments `A`, `P`, `D` and `Y` are ignored.

`A` Age; numerical vector with mean age at diagnosis for each unit.

`P` Period; numerical vector with mean date of diagnosis for each unit.

`D` Cases, deaths; numerical vector.

`Y` Person-years; numerical vector. Also used as denominator for binomial data, see the `dist` argument.

`ref.c` Reference cohort, numerical. Defaults to median date of birth among cases. If used with `parm="AdCP"` or `parm="AdPC"`, the resdiual cohort effects will be 1 at `ref.c`

`ref.p` Reference period, numerical. Defaults to median date of diagnosis among cases.

`dist` Distribution (or more precisely: Likelihood) used for modelling. if a binomial model us ised, `Y` is assuemd to be the denominator; `"binomial"` gives a binomial model with logit link.

`model` Type of model fitted:

- `ns` fits a model with natural splines for each of the terms, with `npar` parameters for the terms.
- `bs` fits a model with B-splines for each of the terms, with `npar` parameters for the terms.
- `ls` fits a model with linear splines.
- `factor` fits a factor model with one parameter per value of `A`, `P` and `C`. `npar` is ignored in this case.

`dr.extr` Character. How the drift parameter should be extracted from the age-period-cohort model. `"weighted"` (default) lets the weighted average (by marginal no. cases, `D`) of the estimated period and cohort effects have 0 slope. `"Holford"` uses the naive average over all values for the estimated effects, disregarding the no. cases.

`parm` Character. Indicates the parametrization of the effects. The first four refer to the ML-fit of the Age-Period-Cohort model, the last four give Age-effects from a smaller model and residuals relative to this. If one of the latter is chosen, the argument `dr.extr` is ignored. Possible values for `parm` are:

- `"ACP"`: ML-estimates. Age-effects as rates for the reference cohort. Cohort effects as RR relative to the reference cohort. Period effects constrained to be 0 on average with 0 slope.
- `"APC"`: ML-estimates. Age-effects as rates for the reference period. Period effects as RR relative to the reference period. Cohort effects constrained to be 0 on average with 0 slope.
- `"AdCP"`: ML-estimates. Age-effects as rates for the reference cohort. Cohort and period effects constrained to be 0 on average with 0 slope. These effects do not multiply to the fitted rates, the drift is missing and needs to be included to produce the fitted values.

- "AdPC": ML-estimates. Age-effects as rates for the reference period. Cohort and period effects constrained to be 0 on average with 0 slope. These effects do not multiply to the fitted rates, the drift is missing and needs to be included to produce the fitted values.
- "Ad-C-P": Age effects are rates for the reference cohort in the Age-drift model (cohort drift). Cohort effects are from the model with cohort alone, using log(fitted values) from the Age-drift model as offset. Period effects are from the model with period alone using log(fitted values) from the cohort model as offset.
- "Ad-P-C": Age effects are rates for the reference period in the Age-drift model (period drift). Period effects are from the model with period alone, using log(fitted values) from the Age-drift model as offset. Cohort effects are from the model with cohort alone using log(fitted values) from the period model as offset.
- "AC-P": Age effects are rates for the reference cohort in the Age-Cohort model, cohort effects are RR relative to the reference cohort. Period effects are from the model with period alone, using log(fitted values) from the Age-Cohort model as offset.
- "AP-C": Age effects are rates for the reference period in the Age-Period model, period effects are RR relative to the reference period. Cohort effects are from the model with cohort alone, using log(fitted values) from the Age-Period model as offset.

`npar` The number of parameters to use for each of the terms in the model. It can be a list of three numerical vectors, in which case these taken as the knots for the age, period and cohort effect, the first and last element in each vector are used as the boundary knots.

`alpha` The significance level. Estimates are given with (1-`alpha`) confidence limits.

`scale` numeric(1), factor multiplied to the rate estimates before output.

`print.AOV` Should the analysis of deviance table for the models be printed?

Value

An object of class "apc" (recognized by `apc.lines` and `apc.plot`) — a list with components:

`Age` Matrix with 4 colums: `A.pt` with the ages (equals `unique(A)`) and three columns giving the estimated rates with c.i.s.

`Per` Matrix with 4 colums: `P.pt` with the dates of diagnosis (equals `unique(P)`) and three columns giving the estimated RRs with c.i.s.

`Coh` Matrix with 4 colums: `C.pt` with the dates of birth (equals `unique(P-A)`) and three columns giving the estimated RRs with c.i.s.

`Drift` A 3 column matrix with drift-estimates and c.i.s: The first row is the ML-estimate of the drift (as defined by `drift`), the second row is the estimate from the Age-drift model. For the sequential parametrizations, only the latter is given.

`Ref` Numerical vector of length 2 with reference period and cohort. If ref.p or ref.c was not supplied the corresponding element is NA.

`AOV` Analysis of deviance table comparing the five classical models.

`Type` Character string explaining the model and the parametrization.

`Knots` If `model` is one of `"ns"` or `"bs"`, a list with three components: `Age`, `Per`, `Coh`, each one a vector of knots. The max and the min are the boundary knots.

Author(s)

Bendix Carstensen, `http://www.biostat.ku.dk/~bxc`

References

The considerations behind the parametrizations used in this function are given in details in a preprint from Department of Biostatistics in Copenhagen:
http://www.pubhealth.ku.dk/bs/publikationer/rr-06-1.pdf, later published as: B. Carstensen: Age-period-cohort models for the Lexis diagram. Statistics in Medicine, 10; 26(15):3018-45, 2007.

See Also

apc.frame, apc.lines, apc.plot.

Examples

```
library( Epi )
data(lungDK)

# Taylor a dataframe that meets the requirements
exd <- lungDK[,c("Ax","Px","D","Y")]
names(exd)[1:2] <- c("A","P")

# Two different ways of parametrizing the APC-model, ML
ex.H <- apc.fit( exd, npar=7, model="ns", dr.extr="Holford",  parm="ACP", scale=10^5 )
ex.W <- apc.fit( exd, npar=7, model="ns", dr.extr="weighted", parm="ACP", scale=10^5 )

# Sequential fit, first AC, then P given AC.
ex.S <- apc.fit( exd, npar=7, model="ns", parm="AC-P", scale=10^5 )

# Show the estimated drifts
ex.H[["Drift"]]
ex.W[["Drift"]]
ex.S[["Drift"]]

# Plot the effects
fp <- apc.plot( ex.H )
apc.lines( ex.W, frame.par=fp, col="red" )
apc.lines( ex.S, frame.par=fp, col="blue" )
```

---

apc.frame *Produce an empty frame for display of parameter-estimates from Age-Period-Cohort-models.*

Description

A plot is generated where both the age-scale and the cohort/period scale is on the x-axis. The left vertical axis will be a logarithmic rate scale referring to age-effects and the right a logarithmic rate-ratio scale of the same relative extent as the left referring to the cohort and period effects (rate ratios).

Only an empty plot frame is generated. Curves or points must be added with `points`, `lines` or the special utility function apc.lines.

Usage

```
apc.frame( a.lab,
          cp.lab,
           r.lab,
          rr.lab = r.lab / rr.ref,
          rr.ref = r.lab[length(r.lab)/2],
           a.tic = a.lab,
          cp.tic = cp.lab,
           r.tic = r.lab,
          rr.tic = r.tic / rr.ref,
         tic.fac = 1.3,
           a.txt = "Age",
          cp.txt = "Calendar time",
           r.txt = "Rate per 100,000 person-years",
          rr.txt = "Rate ratio",
        ref.line = TRUE,
             gap = diff(range(c(a.lab, a.tic)))/3,
        col.grid = gray(0.85),
           sides = c(1,2,4) )
```

Arguments

`a.lab` Numerical vector of labels for the age-axis.

`cp.lab` Numerical vector of labels for the cohort-period axis.

`r.lab` Numerical vector of labels for the rate-axis (left vertical)

`rr.lab` Numerical vector of labels for the RR-axis (right vertical)

`rr.ref` At what level of the rate scale is the RR=1 to be.

`a.tic` Location of additional tick marks on the age-scale

`cp.tic` Location of additional tick marks on the cohort-period-scale

`r.tic` Location of additional tick marks on the rate-scale

`rr.tic` Location of additional tick marks on the RR-axis.

`tic.fac` Factor with which to diminish intermediate tick marks

`a.txt` Text for the age-axis (left part of horizontal axis).

`cp.txt` Text for the cohort/period axis (right part of horizontal axis).

`r.txt` Text for the rate axis (left vertical axis).

`rr.txt` Text for the rate-ratio axis (right vertical axis)

`ref.line` Logical. Should a reference line at RR=1 be drawn at the calendar time part of the plot?

`gap` Gap between the age-scale and the cohort-period scale

`col.grid` Colour of the grid put in the plot.

`sides` Numerical vector indicating on which sides axes should be drawn and annotated. This option is aimed for multi-panel displays where axes only are put on the outer plots.

Details

The function produces an empty plot frame for display of results from an age-period-cohort model, with age-specific rates in the left side of the frame and cohort and period rate-ratio parameters in the right side of the frame. There is a gap of `gap` between the age-axis and the calendar time axis, vertical grid lines at `c(a.lab,a.tic,cp.lab,cp.tic)`, and horizontal grid lines at `c(r.lab,r.tic)`.

The function returns a numerical vector of length 2, with names `c("cp.offset","RR.fac")`. The y-axis for the plot will be a rate scale for the age-effects, and the x-axis will be the age-scale. The cohort and period effects are plotted by subtracting the first element (named `"cp.offset"`) of the returned result form the cohort/period, and multiplying the rate-ratios by the second element of the returned result (named `"RR.fac"`).

Value

A numerical vector of length two, with names `c("cp.offset","RR.fac")`. The first is the offset for the cohort period-axis, the second the multiplication factor for the rate-ratio scale.

Side-effect: A plot with axes and grid lines but no points or curves. Moreover, the option `apc.frame.par` is given the value `c("cp.offset","RR.fac")`, which is recognized by `apc.plot` and `apc.lines`.

Author(s)

Bendix Carstensen, Steno Diabetes Center, `http://www.pubhealth.ku.dk/~bxc/`

References

`http://www.pubhealth.ku.dk/~bxc/APC/notes.pdf`

See Also

`apc.lines`,`apc.fit`

Examples

```
par( mar=c(4,4,1,4) )
res <-
apc.frame( a.lab=seq(30,90,20), cp.lab=seq(1880,2000,30), r.lab=c(1,2,5,10,20,50),
           a.tic=seq(30,90,10), cp.tic=seq(1880,2000,10), r.tic=c(1:10,1:5*10),
           gap=27 )
res
# What are the axes actually?
par(c("usr","xlog","ylog"))
# How to plot in the age-part: a point at (50,10)
points( 50, 10, pch=16, cex=2, col="blue" )
# How to plot in the cohort-period-part: a point at (1960,0.3)
points( 1960-res[1], 0.3*res[2], pch=16, cex=2, col="red" )
```

`apc.lines` *Plot APC-estimates (and other things) in an APC-frame.*

Description

When an APC-frame has been produced by `apc.frame`, this function draws a set of estimates from an APC-fit in the frame. An optional drift parameter can be added to the period parameters and subtracted from the cohort and age parameters.

Usage

```
 apc.lines( A, P, C,
        scale = c("log","ln","rates","inc","RR"),
    frame.par = options()[["apc.frame.par"]],
        drift = 0,
           c0 = median( C[,1] ),
           a0 = median( A[,1] ),
           p0 = c0 + a0,
           ci = rep( FALSE, 3 ),
          lwd = c(3,1,1),
          lty = 1,
              col = "black",
             type = "l",
            knots = FALSE,
              ... )
  pc.points( x, y, ... )
  pc.lines( x, y, ... )
  pc.matpoints( x, y, ... )
  pc.matlines( x, y, ... )
```

Arguments

A	Age effects. A 4-column matrix with columns age, age-specific rates, lower and upper c.i. If A is of class `apc` (see `apc.fit`, `P`, `C`, `c0`, `a0` and `p0` are ignored, and the estimates from there plotted.
P	Period effects. Rate-ratios. Same form as for the age-effects.
C	Cohort effects. Rate-ratios. Same form as for the age-effects.
scale	Are effects given on a log-scale? Character variable, one of `"log"`, `"ln"`, `"rates"`, `"inc"`, `"RR"`. If `"log"` or `"ln"` it is assumed that effects are log(rates) and log(RRs) otherwise the actual effects are assumed given in `A`, `P` and `C`. If `A` is of class `apc`, it is assumed to be `"rates"`.
frame.par	2-element vector with the cohort-period offset and RR multiplicator. This will typically be the result from the call of `apc.frame`. See this for details.
drift	The drift parameter to be added to the period effect. If `scale="log"` this is assumed to be on the log-scale, otherwise it is assumed to be a multiplicative factor per unit of the first columns of `A`, `P` and `C`
c0	The cohort where the drift is assumed to be 0; the subtracted drift effect is `drift*(C[,1]-c0)`.
a0	The age where the drift is assumed to be 0.
p0	The period where the drift is assumed to be 0.
ci	Should confidence interval be drawn. Logical or character. If character, any occurrence of `"a"` or `"A"` produces confidence intervals for the age-effect. Similarly for period and cohort.
lwd	Line widths for estimates, lower and upper confidence limits.
lty	Linetypes for the three effects.
col	Colours for the three effects.
type	What type of lines / points should be used.
knots	Should knots from the model be shown?
...	Further parameters to be transmitted to `points` `lines`, `matpoints` or `matlines` used for plotting the three sets of curves.
x	vector of `x`-coordinates.
y	vector of `y`-coordinates.

Details

The drawing of three effects in an APC-frame is a rather trivial task, and the main purpose of the utility is to provide a function that easily adds the functionality of adding a drift so that several sets of lines can be easily produced in the same frame.

Since the Age-part of the frame is referred to by its real coordinates plotting in the calendar time part requires translation and scaling to put things correctly there, that is done by the functions `pc.points` etc.

Value

A list of three matrices with the effects plotted is returned invisibly.

Author(s)

Bendix Carstensen, Steno Diabetes Center, http://www.pubhealth.ku.dk/~bxc

See Also

apc.frame, apc.fit, apc.plot

---

apc.plot *Plot the estimates from a fitted Age-Period-Cohort model*

Description

This function plots the estimates created by apc.fit in a single graph. It just calls apc.frame after computing some sensible values of the parameters, and subsequently plots the estimates using apc.lines.

Usage

```
apc.plot(obj, r.txt = "Rate", ...)
```

Arguments

obj	An object of class apc.
r.txt	The text to put on the vertical rate axis.
...	Additional arguments passed on to apc.lines.

Value

A numerical vector of length two, with names c("cp.offset","RR.fac"). The first is the offset for the cohort period-axis, the second the multiplication factor for the rate-ratio scale. Therefore, if you want to plot at (x,y) in the right panel, use (x-res["cp.offset"],y/res["RR.fac"])=(x-res[1],y/res[2]). This vector should be supplied for the parameter frame.par to apc.lines if more sets of estimates is plotted in the same graph.

Author(s)

Bendix Carstensen, Steno Diabetes Center, http://www.pubhealth.ku.dk/~bxc

See Also

apc.lines ,apc.frame, apc.fit

Examples

```
data( lungDK )
attach( lungDK )
apc1 <- apc.fit( A=Ax, P=Px, D=D, Y=Y/10^5 )
fp <- apc.plot( apc1 )
apc.lines( apc1, frame.par=fp, drift=1.01, col="red" )
for( i in 1:11 )
  apc.lines( apc1, frame.par=fp, drift=1+(i-6)/100, col=rainbow(12)[i] )
```

bdendo *A case-control study of endometrial cancer*

Description

The bdendo data frame has 315 rows and 13 columns. These data concern a study in which each case of endometrial cancer was matched with 4 controls. Matching was by date of birth (within one year), marital status, and residence.

Format

This data frame contains the following columns:

set: Case-control set: a numeric vector
d: Case or control: a numeric vector (1=case, 0=control)
gall: Gall bladder disease: a factor with levels No Yes.
hyp: Hypertension: a factor with levels No Yes.
ob: Obesity: a factor with levels No Yes.
est: A factor with levels No Yes.
dur: Duration of conjugated oestrogen therapy: an ordered factor with levels 0 < 1 < 2 < 3 < 4.
non: Use of non oestrogen drugs: a factor with levels No Yes.
duration: Months of oestrogen therapy: a numeric vector.
age: A numeric vector.
cest: Conjugated oestrogen dose: an ordered factor with levels 0 < 1 < 2 < 3.
agegrp: A factor with levels 55-59 60-64 65-69 70-74 75-79 80-84
age3: a factor with levels <64 65-74 75+

Source

Breslow NE, and Day N, Statistical Methods in Cancer Research. Volume I: The Analysis of Case-Control Studies. IARC Scientific Publications, IARC:Lyon, 1980.

Examples

```
data(bdendo)
```

bdendo11 *A 1:1 subset of the endometrial cancer case-control study*

Description

The bdendo11 data frame has 126 rows and 13 columns. This is a subset of the dataset bdendo in which each case was matched with a single control.

Source

Breslow NE, and Day N, Statistical Methods in Cancer Research. Volume I: The Analysis of Case-Control Studies. IARC Scientific Publications, IARC:Lyon, 1980.

Examples

```
data(bdendo11)
```

`births` *Births in a London Hospital*

Description

Data from 500 singleton births in a London Hospital

Usage

```
data(births)
```

Format

A data frame with 500 observations on the following 8 variables.

`id`: Identity number for mother and baby.
`bweight`: Birth weight of baby.
`lowbw`: Indicator for birth weight less than 2500 g.
`gestwks`: Gestation period.
`preterm`: Indicator for gestation period less than 37 weeks.
`matage`: Maternal age.
`hyp`: Indicator for maternal hypertension.
`sex`: Sex of baby: 1:Male, 2:Female.

Source

Anonymous

References

Michael Hills and Bianca De Stavola (2002). A Short Introduction to Stata 8 for Biostatistics, Timberlake Consultants Ltd `http://www.timberlake.co.uk`

Examples

```
data(births)
```

`blcaIT` *Bladder cancer mortality in Italian males*

Description

Number of deaths from bladder cancer and person-years in the Italian male population 1955–1979, in ages 25–79.

Format

A data frame with 55 observations on the following 4 variables:

`age`: Age at death. Left endpoint of age class
`period`: Period of death. Left endpoint of period
`D`: Number of deaths
`Y`: Number of person-years.

Examples

```
data(blcaIT)
```

brv *Bereavement in an elderly cohort*

Description

The `brv` data frame has 399 rows and 11 columns. The data concern the possible effect of marital bereavement on subsequent mortality. They arose from a survey of the physical and mental health of a cohort of 75-year-olds in one large general practice. These data concern mortality up to 1 January, 1990 (although further follow-up has now taken place).

Subjects included all lived with a living spouse when they entered the study. There are three distinct groups of such subjects: (1) those in which both members of the couple were over 75 and therefore included in the cohort, (2) those whose spouse was below 75 (and was not, therefore, part of the main cohort study), and (3) those living in larger households (that is, not just with their spouse).

Format

This data frame contains the following columns:

`id`: subject identifier, a numeric vector
`couple`: couple identifier, a numeric vector
`dob`: date of birth, a date
`doe`: date of entry into follow-up study, a date
`dox`: date of exit from follow-up study, a date
`dosp`: date of death of spouse, a date (if the spouse was still alive at the end of follow-up, this was coded to Janu
`fail`: status at end of follow-up, a numeric vector (0=alive,1=dead)
`group`: see Description, a numeric vector
`disab`: disability score, a numeric vector
`health`: perceived health status score, a numeric vector
`sex`: a factor with levels `Male` `Female`

Source

Jagger C, and Sutton CJ, Death after Marital Bereavement. Statistics in Medicine, 10:395-404, 1991. (Data supplied by Carol Jagger).

Examples

```
data(brv)
```

cal.yr *Functions to convert character, factor and various date objects into a number, and vice versa.*

Description

Dates are converted to a numerical value, giving the calendar year as a fractional number. 1 January 1970 is converted to 1970.0, and other dates are converted by assuming that years are all 365.25 days long, so inaccuracies may arise, for example, 1 Jan 2000 is converted to 1999.999. Differences between converted values will be 1/365.25 of the difference between corresponding `Date` objects.

Usage

```
cal.yr( x, format="%Y-%m-%d", wh=NULL )
as.Date.cal.yr( x, ... )
```

Arguments

`x`	A factor or character vector, representing a date in format `format`, or an object of class `Date`, `POSIXlt`, `POSIXct`, `date`, `dates` or `chron` (the latter two requires the `chron` package). If `x` is a data frame, all variables in the data-frame which are of one the classes mentioned are converted to class `cal.yr`. See arguemt `wh`, though.
`format`	Format of the date values if `x` is factor or character. If this argument is supplied and `x` is a datafame, all character variables are converted to class `cal.yr`. Factors in the dataframe will be ignored.
`wh`	Indices of the variables to convert if `x` is a data frame. Can be either a numerical or character vector.
`...`	Arguments passed on from other methods.

Value

`cal.yr` returns a numerical vector of the same length as `x`, of class `c("cal.yr","numeric")`. If `x` is a data frame a dataframe with some of the columns converted to class `"cal.yr"` is returned.

`as.Date.cal.yr` returns a `Date` object.

Author(s)

Bendix Carstensen, Steno Diabetes Center \& Dept. of Biostatistics, University of Copenhagen, `<bxc@steno.dk>`, `http://www.pubhealth.ku.dk/~bxc`

See Also

`DateTimeClasses`, `Date`

Examples

```
# Character vector of dates:
birth <- c("14/07/1852","01/04/1954","10/06/1987","16/05/1990",
           "01/01/1996","01/01/1997","01/01/1998","01/01/1999")
# Proper conversion to class "Date":
birth.dat <- as.Date( birth, format="%d/%m/%Y" )
# Converson of character to class "cal.yr"
bt.yr <- cal.yr( birth, format="%d/%m/%Y" )
# Back to class "Date":
bt.dat <- as.Date( bt.yr )
# Numerical calculation of days since 1.1.1970:
days <- Days <- (bt.yr-1970)*365.25
# Blunt assignment of class:
class( Days ) <- "Date"
# Then data.frame() to get readable output of results:
data.frame( birth, birth.dat, bt.yr, bt.dat, days, Days, round(Days) )
```

ccwc *Generate a nested case-control study*

Description

Given the basic outcome variables for a cohort study: the time of entry to the cohort, the time of exit and the reason for exit (”failure” or ”censoring”), this function computes risk sets and generates a matched case-control study in which each case is compared with a set of controls randomly sampled from the appropriate risk set. Other variables may be matched when selecting controls.

Usage

```
ccwc(entry=0, exit, fail, origin=0, controls=1, match=list(), include=list(), data=NULL, silent=
```

Arguments

`entry`	Time of entry to follow-up
`exit`	Time of exit from follow-up
`fail`	Status on exit (1=Fail, 0=Censored)
`origin`	Origin of analysis time scale
`controls`	The number of controls to be selected for each case
`match`	List of categorical variables on which to match cases and controls
`include`	List of other variables to be carried across into the case-control study
`data`	Data frame in which to look for input variables
`silent`	If False, echos a . to the screen for each case-control set created; otherwise produces no output.

Value

The case-control study, as a dataframe containing:

`Set`	case-control set number
`Map`	row number of record in input dataframe
`Time`	failure time of the case in this set
`Fail`	failure status (1=case, 0=control)

These are followed by the matching variables, and finally by the variables in the `include` list

Author(s)

David Clayton

References

Clayton and Hills, Statistical Models in Epidemiology, Oxford University Press, Oxford:1993.

See Also

`Lexis`

Examples

```
#
# For the diet and heart dataset, create a nested case-control study
# using the age scale and matching on job
#
data(diet)
dietcc <- ccwc(doe, dox, chd, origin=dob, controls=2, data=diet,
               include=energy, match=job)
```

`ci.cum` *Compute cumulative sum of estimates.*

Description

Computes the cumulative sum of parameter functions and the standard error of it. Optionally the exponential is applied to the parameter functions before it is cumulated.

Usage

```
ci.cum( obj,
    ctr.mat = NULL,
     subset = NULL,
       intl = 1,
      alpha = 0.05,
        Exp = TRUE,
     sample = FALSE )
```

Arguments

`obj`	A model object (of class `lm`, `glm`, `coxph`, `survreg`, `lme`,`mer`,`nls`,`gnlm`, `MIresult` or `polr`).
`ctr.mat`	Contrast matrix defining the parameter functions from the parameters of the model.
`subset`	Subset of the parameters of the model to which `ctr.mat` should be applied.
`intl`	Interval length for the cumulation. Either a constant or a numerical vector of length `nrow(ctr.mat)`.
`alpha`	Significance level used when computing confidence limits.
`Exp`	Should the parameter function be exponentiated before it is cumulated?
`sample`	Should a sample of the original parameters be used to compute a cumulative rate?

Details

The purpose of this function is to compute cumulative rate based on a model for the rates. If the model is a multiplicative model for the rates, the purpose of `ctr.mat` is to return a vector of rates or log-rates when applied to the coefficients of the model. If log-rates are returned from the model, the they should be exponentiated before cumulated, and the variances computed accordingly. Since log-linear models are the most common the `Exp` parameter defaults to TRUE.

Value

A matrix with 4 columns: Estimate, lower and upper c.i. and standard error. If `sample` is TRUE, a sampled vector is reurned, if `sample` is numeric a matrix with `sample` columns is returned, each column a cumulative rate based on a random sample from the distribution of the parameter estimates.

Author(s)

Bendix Carstensen, http://www.pubhealth.ku.dk/~bxc

See Also

See also ci.lin

Examples

```
# Packages required for this example
library( splines )
library( survival )
data( lung )
par( mfrow=c(1,2) )

# Plot the Kaplan-meier-estimator
plot( survfit( Surv( time, status==2 ) ~ 1, data=lung ) )

# Declare data as Lexis
lungL <- Lexis( exit=list("tfd"=time),
                exit.status=(status==2)*1, data=lung )
summary( lungL )

# Cut the follow-up every 10 days
sL <- splitLexis( lungL, "tfd", breaks=seq(0,1100,10) )
str( sL )
summary( sL )

# Fit a Poisson model with a natural spline for the effect of time.
# Extract the variables needed
D <- status(sL, "exit")
Y <- dur(sL)
tB <- timeBand( sL, "tfd", "left" )
MM <- ns( tB, knots=c(50,100,200,400,700), intercept=TRUE )
mp <- glm( D ~ MM - 1 + offset(log(Y)),
           family=poisson, eps=10^-8, maxit=25 )

# Contrast matrix to extract effects, i.e. matrix to multiply with the
# coefficients to produce the log-rates: unique rows of MM, in time order.
T.pt <- sort( unique( tB ) )
T.wh <- match( T.pt, tB )
Lambda <- ci.cum( mp, ctr.mat=MM[T.wh,], intl=diff(c(0,T.pt)) )

# Put the estimated survival function on top of the KM-estimator
matlines( c(0,T.pt[-1]), exp(-Lambda[,1:3]), lwd=c(3,1,1), lty=1, col="Red" )

# Extract and plot the fitted intensity function
lambda <- ci.lin( mp, ctr.mat=MM[T.wh,], Exp=TRUE )
matplot( T.pt, lambda[,5:7]*10^3, type="l", lwd=c(3,1,1), col="black", lty=1,
         log="y", ylim=c(0.2,20) )
```

ci.lin *Compute linear functions of parameters with s.e.*

Description

For a given model object the function computes a linear function of the parameters and the corresponding standard errors, p-values and confidence intervals.

Usage

```
ci.lin( obj,
    ctr.mat = NULL,
     subset = NULL,
     subint = NULL,
      diffs = FALSE,
       fnam = !diffs,
       vcov = FALSE,
      alpha = 0.05,
         df = Inf,
        Exp = FALSE,
     sample = FALSE )
Wald( obj, H0=0, ... )
ci.mat( alpha = 0.05, df=Inf )
```

Arguments

`obj` A model object (of class `lm`, `glm`, `coxph`, `survreg`, `lme`,`mer`,`nls`,`gnlm`, `MIresult` or `polr`).

`ctr.mat` Contrast matrix to be multiplied to the parameter vector, i.e. the desired linear function of the parameters.

`subset` The subset of the parameters to be used. If given as a character vector, the elements are in turn matched against the parameter names (using `grep`) to find the subset. Repeat parameters may result from using a character vector. This is considered a facility.

`subint` SUBset selection like for `subset`, except that elements of a character vector given as argument will be used to select subsets of parameters and only the INTersection of these is returned.

`diffs` If TRUE, all differences between parameters in the subset are computed. `ctr.mat` is ignored. If `obj` inherits from `lm`, and `subset` is given as a string `subset` is used to search among the factors in the model and differences of all factor levels for the first match are shown. If `subset` does not match any of the factors in the model, all pairwise differences between parameters matching are returned.

`fnam` Should the common part of the parameter names be included with the annotation of contrasts? Ignored if `diffs==T`. If a sting is supplied this will be prefixed to the labels.

`vcov` Should the covariance matrix of the set of parameters be returned? If this is set, `Exp` is ignored. See details.

`alpha` Significance level for the confidence intervals.

`df` Integer. Number of degrees of freedom in the t-distribution used to compute the quantiles used to construct the confidence intervals.

`Exp` If `TRUE` columns 5:6 are replaced with exp(columns 1,5,6).

`sample` Logical or numerical. If `TRUE` or numerical a sample of the linear parameter function as defined by `subset` and `ctr.mat` is returned.

`H0` The null values for the selected/transformed parameters to be tested by a Wald test. Must have the same length as the selected parameter vector.

`...` Parameters passed on to `ci.lin`.

Value

ci.lin returns a matrix with number of rows and rownames as ctr.mat. The columns are Estimate, Std.Err, z, P, 2.5% and 97.5%. If vcov=TRUE a list with components est, the desired functional of the parameters and vcov, the variance covariance matrix of this, is returned but not printed. If Exp==TRUE the confidence intervals for the parameters are replaced with three columns: exp(estimate,c.i.).

Wald computes a Wald test for a subset of (possibly linearly transformed) parameters. The selection of the subset of parameters is the same as for ci.lin. Using the ctr.mat argument makes it possible to do a Wald test for equality of parameters. Wald returns a named numerical vector of lenght 3, with names Chisq, d.f. and P.

ci.mat returns a 2 by 3 matrix with rows c(1,0,0) and c(0,-1,1)*1.96, devised to post-multiply to a p by 2 matrix with columns of estimates and standard errors, so as to produce a p by 3 matrix of estimates and confidnece limits. Used internally in ci.lin and ci.cum. The 1.96 is replaced by the appropriate quantile from the normal or t-distribution when arguments alpha and/or df are given.

Author(s)

Bendix Carstensen, http://www.pubhealth.ku.dk/~bxc & Michaal Hills
http://www.mhills.pwp.blueyonder.co.uk/

See Also

See also ci.cum

Examples

```
# Bogus data:
f <- factor( sample( letters[1:5], 200, replace=TRUE ) )
g <- factor( sample( letters[1:3], 200, replace=TRUE ) )
x <- rnorm( 200 )
y <- 7 + as.integer( f ) * 3 + 2 * x + 1.7 * rnorm( 200 )

# Fit a simple model:
mm <- lm( y ~ x + f + g )
ci.lin( mm )
ci.lin( mm, subset=3:6, diff=TRUE, fnam=FALSE )
ci.lin( mm, subset=3:6, diff=TRUE, fnam=TRUE )
ci.lin( mm, subset="f", diff=TRUE, fnam="f levels:" )
print( ci.lin( mm, subset="g", diff=TRUE, fnam="gee!:", vcov=TRUE ) )

# Use character defined subset to get ALL contrasts:
ci.lin( mm, subset="f", diff=TRUE )

# A Wald test of wheter the g-parameters are 0
Wald( mm, subset="g" )
# Wald test of whether the three first f-parameters are equal:
( CM <- rbind( c(1,-1,0,0), c(1,0,-1,0)) )
Wald( mm, subset="f", ctr.mat=CM )
# or alternatively
( CM <- rbind( c(1,-1,0,0), c(0,1,-1,0)) )
Wald( mm, subset="f", ctr.mat=CM )
```

ci.pd *Compute confidence limits for a difference of two independent proportions.*

Description

The usual formula for the c.i. of at difference of proportions is inaccurate. Newcombe has compared 11 methods and method 10 in his paper looks like a winner. It is implemented here.

Usage

```
ci.pd(aa, bb=NULL, cc=NULL, dd=NULL,
     method = "Nc",
      alpha = 0.05, conf.level=0.95,
     digits = 3,
      print = TRUE,
detail.labs = FALSE )
```

Arguments

`aa`	Numeric vector of successes in sample 1. Can also be a matrix or array (see details).
`bb`	Successes in sample 2.
`cc`	Failures in sample 1.
`dd`	Failures in sample 2.
`method`	Method to use for calculation of confidence interval, see ”Details”.
`alpha`	Significance level
`conf.level`	Confidence level
`print`	Should an account of the two by two table be printed.
`digits`	How many digits should the result be rounded to if printed.
`detail.labs`	Should the computing of probability differences be reported in the labels.

Details

Implements method 10 from Newcombe(1998) (method=”Nc”) or from Agresti & Caffo(2000) (method=”AC”).

`aa`, `bb`, `cc` and `dd` can be vectors. If `aa` is a matrix, the elements `[1:2,1:2]` are used, with successes `aa[,1:2]`. If `aa` is a three-way table or array, the elements `aa[1:2,1:2,]` are used.

Value

A matrix with three columns: probability difference, lower and upper limit. The number of rows equals the length of the vectors `aa`, `bb`, `cc` and `dd` or, if `aa` is a 3-way matrix, `dim(aa)[3]`.

Author(s)

Bendix Carstensen, Esa Laara. http://www.biostat.ku.dk/~bxc

References

RG Newcombe: Interval estimation for the difference between independent proportions. Comparison of eleven methods. Statistics in Medicine, 17, pp. 873-890, 1998.

A Agresti & B Caffo: Simple and effective confidence intervals for proportions and differences of proportions result from adding two successes and two failures. The American Statistician, 54(4), pp. 280-288, 2000.

See Also

twoby2, binom.test

Examples

```
( a <- matrix( sample( 10:40, 4 ), 2, 2 ) )
ci.pd( a )
twoby2( t(a) )
prop.test( t(a) )
( A <- array( sample( 10:40, 20 ), dim=c(2,2,5) ) )
ci.pd( A )
ci.pd( A, detail.labs=TRUE, digits=3 )
```

clogistic *Conditional logistic regression*

Description

Estimates a logistic regression model by maximizing the conditional likelihood. The conditional likelihood calculations are exact, and scale efficiently to strata with large numbers of cases.

Usage

```
clogistic(formula, strata, data, subset, na.action, init,
model = TRUE, x = FALSE, y = TRUE, contrasts = NULL,
iter.max=20, eps=1e-6, toler.chol = sqrt(.Machine$double.eps))
```

Arguments

`formula`	Model formula
`strata`	Factor describing membership of strata for conditioning
`data`	data frame containing the variables in the formula and strata arguments
`subset`	subset of records to use
`na.action`	missing value handling
`init`	initial values
`model`	a logical value indicating whether *model frame* should be included as a component of the returned value
`x,y`	logical values indicating whether the response vector and model matrix used in the fitting process should be returned as components of the returned value.
`contrasts`	an optional list. See the `contrasts.arg` of `model.matrix.default`
`iter.max`	maximum number of iterations
`eps`	Convergence tolerence. Iteration continues until the relative change in the conditional log likelihood is less than `eps`. Must be positive.
`toler.chol`	Tolerance used for detection of a singularity during a Cholesky decomposition of the variance martrix. This is used to detect redundant predictor variables. Must be less than `eps`.

Value

An object of class `"clogistic"`. This is a list containing the following components:

`coefficients` the estimates of the log-odds ratio parameters. If the model is over-determined there will be missing values in the vector corresponding to the redundant columns in the model matrix.

`var` the variance matrix of the coefficients. Rows and columns corresponding to any missing coefficients are set to zero.

`loglik` a vector of length 2 containing the log-likelihood with the initial values and with the final values of the coefficients.

`iter` number of iterations used.

`n` number of observations used. Observations may be dropped either because they are missing, or because they belong to a homogenous stratum. For more details on which observations were used, see `informative` below.

`informative` if `model=TRUE`, a logical vector of length equal to the number of rows in the model frame. This indicates whether an observation is informative, in the sense that it makes a non-zero contribution to the log-likelihood. If `model=FALSE`, this is NULL.

The output will also contain the following, for documentation see the `glm` object: `terms`, `formula`, `call`, `contrasts`, `xlevels`, and, optionally, `x`, `y`, and/or `frame`.

Author(s)

Martyn Plummer

See Also

`glm`

Examples

```
data(bdendo)
clogistic(d ~ cest + dur, strata=set, data=bdendo)
```

`contr.cum` *Contrast matrices*

Description

Return a matrix of contrasts for factor coding.

Usage

```
contr.cum(n)
contr.diff(n)
contr.2nd(n)
contr.orth(n)
```

Arguments

`n` A vector of levels for a factor, or the number of levels.

Details

These functions are used for creating contrast matrices for use in fitting regression models. The columns of the resulting matrices contain contrasts which can be used for coding a factor with `n` levels.

`contr.cum` gives a coding corresponding to successive differences between factor levels.

`contr.diff` gives a coding that correspond to the cumulative sum of the value for each level. This is not meaningful in a model where the intercept is included, therefore `n` columns ia always returned.

`contr.2nd` gives contrasts corresponding to 2nd order differences between factor levels. Returns a matrix with `n-2` columns.

`contr.orth` gives a matrix with `n-2` columns, which are mutually orthogonal and orthogonal to the matrix `cbind(1,1:n)`

Value

A matrix with `n` rows and `k` columns, with `k`=`n` for `contr.diff` `k`=`n-1` for `contr.cum` `k`=`n-2` for `contr.2nd` and `contr.orth`.

Author(s)

Bendix Carstensen

See Also

`contr.treatment`

Examples

```
contr.cum(6)
contr.2nd(6)
contr.diff(6)
contr.orth(6)
```

`cutLexis` *Cut follow-up at a specified date for each person.*

Description

Follow-up intervals in a Lexis object are divided into two sub-intervals: one before and one after an intermediate event. The intermediate event may denote a change of state, in which case the entry and exit status variables in the split Lexis object are modified.

Usage

```
cutLexis( data, cut, timescale = 1,
                     new.state = nlevels(data$lex.Cst)+1,
                     new.scale = FALSE,
                     split.states = FALSE,
                     progressive = FALSE,
                     precursor.states = NULL,
                     count = FALSE)
countLexis( data, cut, timescale = 1 )
```

Arguments

`data` A `Lexis` object.

`cut` A numeric vector with the times of the intermediate event. If a time is missing (`NA`) then the event is assumed to occur at time `Inf`. `cut` can also be a dataframe, see details.

`timescale` The timescale that `cut` refers to. Numeric or character.

`new.state` The state to which a transition occur at time `cut`. It may be a single value, which is then applied to all rows of `data`, or a vector with a separate value for each row

`new.scale` Name of the timescale defined as ”time since entry to new.state”. If `TRUE` a name for the new scale is constructed. See details.

`split.states` Should states that are not precursor states be split according to whether the intermediate event has occurred.

`progressive` a logical flag that determines the changes to exit status. See details.

`precursor.states` an optional vector of states to be considered as ”less severe” than `new.state`. See Details below

`count` logical indicating whether the `countLexis` options should be used. Specifying `count=TRUE` amounts to calling `countLexis`, in which case the arguments `new.state`, `progressive` and `precursor.states` will be ignored.

Details

The `cutLexis` function allows a number of different ways of specifying the cutpoints and of modifying the status variable.

If the `cut` argument is a dataframe it must have columns `lex.id`, `cut` and `new.state`. The values of `lex.id` must be unique. In this case it is assumed that each row represents a cutpoint (on the timescale indicated in the argument `timescale`). This cutpoint will be applied to all records in `data` with the corresponding `lex.id`. This makes it possible to apply `cutLexis` to a split `Lexis` object.

If a `new.state` argument is supplied, the status variable is only modified at the time of the cut point. However, it is often useful to modify the status variable after the cutpoint when an important event occurs. There are three distinct ways of doing this.

If the `progressive=TRUE` argument is given, then a ”progressive” model is assumed, in which the status can either remain the same or increase during follow-up, but never decrease. This assumes that the state variables `lex.Cst` and `lex.Xst` are either numeric or ordered factors. In this case, if `new.state=X`, then any exit status with a value less than `X` is replaced with `X`. The Lexis object must already be progressive, so that there are no rows for which the exit status is less than the entry status. If `lex.Cst` and `lex.Xst` are factors they must be ordered factors if `progressive=TRUE` is given.

As an alternative to the `progressive` argument, an explicit vector of precursor states, that are considered less severe than the new state, may be given. If `new.state=X` and `precursor.states=c(Y,Z)` then any exit status of `Y` or `Z` in the second interval is replaced with `X` and all other values for the exit status are retained.

The `countLexis` function is a variant of `cutLexis` when the cutpoint marks a recurrent event, and the status variable is used to count the number of events that have occurred. Times given in `cut` represent times of new events. Splitting with `countLexis` increases the status variable by 1. If the current status is `X` and the exit status is `Y` before cutting, then after cutting the entry status is `X`, `X+1` for the first and second intervals, respectively, and the exit status is `X+1`, `Y+1` respectively. Moreover the values of the status is increased by 1 for all intervals for all intervals after the cut for the person in question. Hence, a call to `countLexis` is needed for as many times as the person with most events. But also it is immaterial in what order the cutpoints are entered.

Value

A Lexis object, for which each follow-up interval containing the cutpoint is split in two: one before and one after the cutpoint. An extra time-scale is added; the time since the event at cut. This is NA for any follow-up prior to the intermediate event.

Note

The cutLexis function superficially resembles the splitLexis function. However, the splitLexis function splits on a vector of common cut-points for all rows of the Lexis object, whereas the cutLexis function splits on a single time point, which may be distinct for each row, modifies the status variables, and adds a new timescale.

Author(s)

Bendix Carstensen, Steno Diabetes Center, <bxc@steno.dk>, Martyn Plummer, IARC, <plummer@iarc.fr>.

See Also

splitLexis, Lexis, summary.Lexis

Examples

```
# A small artificial example
xx <- Lexis( entry=list(age=c(17,24,33,29),per=c(1920,1933,1930,1929)),
             duration=c(23,57,12,15), exit.status=c(1,2,1,2) )
xx
cut <- c(33,47,29,50)
cutLexis(xx, cut, new.state=3, precursor=1)
cutLexis(xx, cut, new.state=3, precursor=2)
cutLexis(xx, cut, new.state=3, precursor=1:2)
# The same as the last example
cutLexis(xx, cut, new.state=3)

# The same example with a factor status variable
yy <- Lexis(entry = list(age=c(17,24,33,29),per=c(1920,1933,1930,1929)),
            duration = c(23,57,12,15),
            entry.status = factor(rep("alpha",4),
            levels=c("alpha","beta","gamma")),
            exit.status = factor(c("alpha","beta","alpha","beta"),
            levels=c("alpha","beta","gamma")))

cutLexis(yy,c(33,47,29,50),precursor="alpha",new.state="gamma")
cutLexis(yy,c(33,47,29,50),precursor=c("alpha","beta"),new.state="aleph")

## Using a dataframe as cut argument
rl <- data.frame( lex.id=1:3, cut=c(19,53,26), timescale="age", new.state=3 )
rl
cutLexis( xx, rl )
cutLexis( xx, rl, precursor=1 )
cutLexis( xx, rl, precursor=0:2 )

## It is immaterial in what order splitting and cutting is done
xs <- splitLexis( xx, breaks=seq(0,100,10), time.scale="age" )
xs
xsC <- cutLexis(xs, rl, precursor=0 )
```

```
xC <- cutLexis( xx, rl, pre=0 )
xC
xCs <- splitLexis( xC, breaks=seq(0,100,10), time.scale="age" )
xCs
```

detrend *Projection of a model matrix on to the orthogonal complement of a trend.*

Description

The columns of the model matrix `M` is projected on the orthogonal complement to the matrix `(1,t)`. Orthogonality is defined w.r.t. an inner product defined by the weights `weight`.

Usage

```
detrend( M, t, weight = rep(1, nrow(M)) )
```

Arguments

`M`	A model matrix.
`t`	The trend defining a subspace. A numerical vector of length `nrow(M)`
`weight`	Weights defining the inner product of vectors `x` and `y` as `sum(x*w*y)`. A numerical vector of length `nrow(M)`, defaults to a vector of `1`s.

Details

The functions is intended to be used in parametrization of age-period-cohort models.

Value

A full-rank matrix with columns orthogonal to `(1,t)`.

Author(s)

Bendix Carstensen, Steno Diabetes Center, `http://www.pubhealth.ku.dk/~bxc`, with help from Peter Dalgaard.

See Also

`projection.ip`

diet *Diet and heart data*

Description

The `diet` data frame has 337 rows and 14 columns. The data concern a subsample of subjects drawn from larger cohort studies of the incidence of coronary heart disease (CHD). These subjects had all completed a 7-day weighed dietary survey while taking part in validation studies of dietary questionnaire methods. Upon the closure of the MRC Social Medicine Unit, from where these studies were directed, it was found that 46 CHD events had occurred in this group, thus allowing a serendipitous study of the relationship between diet and the incidence of CHD.

Format

This data frame contains the following columns:

`id`: subject identifier, a numeric vector.
`doe`: date of entry into follow-up study, a `Date` variable.
`dox`: date of exit from the follow-up study, a `Date` variable.
`dob`: date of birth, a `Date` variable.
`y`: - number of years at risk, a numeric vector.
`fail`: status on exit, a numeric vector (codes 1, 3, 11, and 13 represent CHD events)
`job`: occupation, a factor with levels `Driver` `Conductor` `Bank worker`
`month`: month of dietary survey, a numeric vector
`energy`: total energy intake (KCal per day/100), a numeric vector
`height`: (cm), a numeric vector
`weight`: (kg), a numeric vector
`fat`: fat intake (g/day), a numeric vector
`fibre`: dietary fibre intake (g/day), a numeric vector
`energy.grp`: high daily energy intake, a factor with levels `<=2750 KCal` `>2750 KCal`
`chd`: CHD event, a numeric vector (1=CHD event, 0=no event)

Source

The data are described and used extensively by Clayton and Hills, Statistical Models in Epidemiology, Oxford University Press, Oxford:1993. They were rescued from destruction by David Clayton and reentered from paper printouts.

Examples

```
data(diet)
# Illustrate the follow-up in a Lexis diagram
Lexis.diagram( age=c(30,75), date=c(1965,1990),
               entry.date=cal.yr(doe), exit.date=cal.yr(dox), birth.date=cal.yr(dob),
               fail=(fail>0), pch.fail=c(NA,16), col.fail=c(NA,"red"), cex.fail=1.0,
               data=diet )
```

`DMconv` *Conversion to diabetes*

Description

Data from a randomized intervention study ("Addition") where persons with prediabetic conditions are followed up for conversion to diabetes (DM). Conversion dates are interval censored. Original data are not published yet, so id-numbers have been changed and all dates have been randomly perturbed.

Usage

```
data(DMconv)
```

Format

A data frame with 1519 observations on the following 6 variables.

`id` Person identifier

`doe` Date of entry, i.e. first visit.

`dlw` Date last seen well, i.e. last visit without DM.

`dfi` Date first seen ill, i.e. first visit with DM.

`gtol` Glucose tolerance. Factor with levels: 1="IFG" (impaired fasting glucose), 2="IGT" (impaired glucose tolerance).

`grp` Randomization. Factor with levels: 1="Intervention", 2="Control".

Source

Signe Saetre Rasmussen, Steno Diabetes Center. The Addition Study.

Examples

```
data(DMconv)
str(DMconv)
head(DMconv)
```

DMlate *The Danish National Diabetes Register.*

Description

These two datasets each contain a random sample of 10,000 persons from the Danish National Diabetes Register. `DMrand` is a random sample from the register, whereas `DMlate` is a random sample among those with date of diagnosis after 1.1.1995.

Usage

```
data(DMrand)
       data(DMlate)
```

Format

A data frame with 10000 observations on the following 6 variables.

`sex` Sex, a factor with levels `M` `F`

`dobth` Date of birth

`dodm` Date of inclusion in the register

`dodth` Date of death

`doins` Date of first insulin prescription

`dox` Date of exit from follow-up.

Details

All dates are given in fractions of years, so 1997.00 corresponds to 1 January 1997 and 1997.997 to 31 December 1997.

Source

Danish National Board of Health.

References

B Carstensen, JK Kristensen, P Ottosen and K Borch-Johnsen: The Danish National Diabetes Register: Trends in incidence, prevalence and mortality, Diabetologia, 51, pp 2187–2196, 2008.

In partucular see the appendix at the end of the paper.

Examples

```
data(DMlate)
str(DMlate)
dml <- Lexis( entry=list(Per=dodm, Age=dodm-dobth, DMdur=0 ),
               exit=list(Per=dox),
        exit.status=factor(!is.na(dodth),labels=c("DM","Dead")),
               data=DMlate )
# Split follow-up at Insulin
dmi <- cutLexis( dml, cut=dml$doins, new.state="Ins", pre="DM" )
summary( dmi )
# Introduce a new timescale
dmi <- cutLexis( dml, cut=dml$doins, new.state="Ins", pre="DM", new.scale=TRUE )
head( dmi )
# Split the states following insulin and explictily name the new timescale
dmi <- cutLexis( dml, cut=dml$doins, new.state="Ins",
                      pre="DM", new.scale="Instime", split.states=TRUE )
summary( dmi )
```

`effx` *Function to calculate effects*

Description

The function calculates the effects of an exposure on a response, possibly stratified by a stratifying variable, and/or controlled for one or more confounding variables.

Usage

```
effx( response, type = "metric",
         fup = NULL,
    exposure,
      strata = NULL,
     control = NULL,
             weights = NULL,
       alpha = 0.05,
        base = 1,
      digits = 3,
        data = NULL )
```

Arguments

`response` The `response` variable - must be numeric

`type` The type of response`type` - must be one of ”metric”, ”binary”, ”failure”, or ”count”

`fup` The `fup` variable contains the follow-up time for a failure response. This must be numeric.

`exposure` The `exposure` variable can be numeric or a factor

`strata` The `strata` stratifying variable - must be a factor

`control` The `control` variable(s) - these are passed as a list if there are more than one.

`weights` Frequency weights for binary response only

`base` Baseline for the effects of a categorical exposure, default 1

`digits` Number of significant digits for the effects, default 3

`alpha` 1 - confidence level

`data` `data` refers to the data used to evaluate the function

Details

The function is a wrapper for glm. Effects are calculated as differences in means for a metric response, odds ratios for a binary response, and rate ratios for a failure or count response.

The k-1 effects for a categorical exposure with k levels are relative to a baseline which, by default, is the first level. The effect of a metric (quantitative) exposure is calculated per unit of exposure.

The exposure variable can be numeric or a factor, but if it is an ordered factor the order will be ignored.

Value

comp1	Effects of exposure
comp2	Tests of significance

Author(s)

Michael Hills

References

www.mhills.pwp.blueyonder.co.uk

Examples

```
library(Epi)
data(births)
births$hyp <- factor(births$hyp,labels=c("normal","hyper"))
births$sex <- factor(births$sex,labels=c("M","F"))

# bweight is the birth weight of the baby in gms, and is a metric
# response (the default)

# effect of hypertension on birth weight
effx(bweight,exposure=hyp,data=births)
# effect of hypertension on birth weight stratified by sex
effx(bweight,exposure=hyp,strata=sex,data=births)
# effect of hypertension on birth weight controlled for sex
effx(bweight,exposure=hyp,control=sex,data=births)
# effect of gestation time on birth weight
effx(bweight,exposure=gestwks,data=births)
# effect of gestation time on birth weight stratified by sex
effx(bweight,exposure=gestwks,strata=sex,data=births)
# effect of gestation time on birth weight controlled for sex
effx(bweight,exposure=gestwks,control=sex,data=births)

# lowbw is a binary response coded 1 for low birth weight and 0 otherwise
# effect of hypertension on low birth weight
effx(lowbw,type="binary",exposure=hyp,data=births)
# etc.
```

effx.match *Function to calculate effects for individually matched case-control studies*

Description

The function calculates the effects of an exposure on a response, possibly stratified by a stratifying variable, and/or controlled for one or more confounding variables.

Usage

```
effx.match(response,
exposure,
match,
strata=NULL,
control=NULL,
base=1,
digits=3,
alpha=0.05,
data=NULL)
```

Arguments

`response`	The `response` variable - must be numeric
`exposure`	The `exposure` variable can be numeric or a factor
`match`	The variable which identifies the matched sets
`strata`	The `strata` stratifying variable - must be a factor
`control`	The `control` variable(s). These are passed as a list if there are more than one of them.
`base`	Baseline for the effects of a categorical exposure, default 1
`digits`	Number of significant digits for the effects, default 3
`alpha`	1 - confidence level
`data`	`data` refers to the data used to evaluate the function

Details

Effects are calculated odds ratios. The function is a wrapper for clogit, from the survival package. The k-1 effects for a categorical exposure with k levels are relative to a baseline which, by default, is the first level. The effect of a metric (quantitative) exposure is calculated per unit of exposure. The exposure variable can be numeric or a factor, but if it is an ordered factor the order will be ignored.

Value

`comp1`	Effects of exposure
`comp2`	Tests of significance

Author(s)

Michael Hills

References

www.mhills.pwp.blueyonder.co.uk

Examples

```
library(Epi)
library(survival)
data(bdendo)

# d is the case-control variable, set is the matching variable.
# The variable est is a factor and refers to estrogen use (no,yes)
# The variable hyp is a factor with 2 levels and refers to hypertension (no, yes)
# effect of est on the odds of being a case
```

```
effx.match(d,exposure=est,match=set,data=bdendo)
# effect of est on the odds of being a case, stratified by hyp
effx.match(d,exposure=est,match=set,strata=hyp,data=bdendo)
# effect of est on the odds of being a case, controlled for hyp
effx.match(d,exposure=est,match=set,control=hyp,data=bdendo)
```

ewrates *Rates of lung and nasal cancer mortality, and total mortality.*

Description

England and Wales mortality rates from lung cancer, nasal cancer, and all causes 1936 - 1980. The 1936 rates are repeated as 1931 rates in order to accomodate follow up for the nickel study.

Usage

```
data(ewrates)
```

Format

A data frame with 150 observations on the following 5 variables:

id:	Subject identifier (numeric)
year	Calendar period, 1931: 1931–35, 1936: 1936–40, ...
age	Age class: 10: 10–14, 15:15–19, ...
lung	Lung cancer mortality rate per 1,000,000 py.
nasal	Nasal cancer mortality rate per 1,000,000 py.
other	All cause mortality rate per 1,000,000 py.

Source

From Breslow and Day, Vol II, Appendix IX.

Examples

```
data(ewrates)
str(ewrates)
```

expand.data *Function to expand data for regression analysis of interval censored data.*

Description

This is a utility function.

The original records with `first.well`, `last.well` and `first.ill` are expanded to multiple records; several for each interval where the person is known to be well and one where the person is known to fail. At the same time columns for the covariates needed to estimate rates and the response variable are generated.

Usage

```
expand.data(fu, formula, breaks, data)
```

Arguments

`fu` A 3-column matrix with `first.well`, `last.well` and `first.ill` in each row.

`formula` Model fromula, used to derive the model matrix.

`breaks` Defines the intervals in which the baseline rate is assumed constant. All follow-up before the first and after the last break is discarded.

`data` Datafrem in which `fu` and `formula` is interpreted.

Value

Returns a list with three components

`rates.frame` Dataframe of covariates for estimation of the baseline rates — one per interval defined by `breaks`.

`cov.frame` Dataframe for estimation of the covariate effects. A data-framed version of the designmatrix from `formula`.

`y` Response vector.

Author(s)

Martyn Plummer, `<plummer@iarc.fr>`

References

B Carstensen: Regression models for interval censored survival data: application to HIV infection in Danish homosexual men. Statistics in Medicine, 15(20):2177-2189, 1996.

See Also

`Icens` `fit.mult` `fit.add`

`fit.add` *Fit an addive excess risk model to interval censored data.*

Description

Utility function.

The model fitted assumes a piecewise constant intensity for the baseline, and that the covariates act additively on the rate scale.

Usage

```
fit.add( y, rates.frame, cov.frame, start )
```

Arguments

`y` Binary vector of outcomes

`rates.frame` Dataframe expanded from the original data by `expand.data`, cooresponding to covariates for the rate parameters.

`cov.frame` do., but covariates corresponding to the `formula` argument of `Icens`

`start` Starting values for the rate parameters. If not supplied, then starting values are generated.

Value

A list with one component:

`rates` A glm object from a binomial model with log-link function.

Author(s)

Martyn Plummer, `<plummer@iarc.fr>`

References

B Carstensen: Regression models for interval censored survival data: application to HIV infection in Danish homosexual men. Statistics in Medicine, 15(20):2177-2189, 1996.

CP Farrington: Interval censored survival data: a generalized linear modelling approach. Statistics in Medicine, 15(3):283-292, 1996.

See Also

`Icens` `fit.mult`

Examples

```
  data( HIV.dk )
```

`fit.baseline` *Fit a piecewise contsnt intesity model for interval censored data.*

Description

Utility function

Fits a binomial model with logaritmic link, with `y` as outcome and covariates in `rates.frame` to estimate rates in the inttervals between `breaks`.

Usage

```
fit.baseline( y, rates.frame, start )
```

Arguments

`y` Binary vector of outcomes

`rates.frame` Dataframe expanded from the original data by `expand.data`

`start` Starting values for the rate parameters. If not supplied, then starting values are generated.

Value

A `glm` object, with binomial error and logaritmic link.

Author(s)

Martyn Plummer, `<plummer@iarc.fr>`

See Also

`fit.add` `fit.mult`

fit.mult *Fits a multiplicative relative risk model to interval censored data.*

Description

Utility function.

The model fitted assumes a piecewise constant baseline rate in intervals specified by the argument `breaks`, and a multiplicative relative risk function.

Usage

```
fit.mult( y, rates.frame, cov.frame, start )
```

Arguments

`y`	Binary vector of outcomes
`rates.frame`	Dataframe expanded from the original data by `expand.data`, cooresponding to covariates for the rate parameters.
`cov.frame`	do., but covariates corresponding to the `formula` argument of `Icens`
`start`	Starting values for the rate parameters. If not supplied, then starting values are generated.

Details

The model is fitted by alternating between two generalized linear models where one estimates the underlying rates in the intervals, and the other estimates the log-relative risks.

Value

A list with three components:

`rates`	A glm object from a binomial model with log-link, estimating the baseline rates.
`cov`	A glm object from a binomial model with complementary log-log link, estimating the log-rate-ratios
`niter`	Nuber of iterations, a scalar

Author(s)

Martyn Plummer, `<plummer@iarc.fr>`, Bendix Carstensen, `<bxc@steno.dk>`

References

B Carstensen: Regression models for interval censored survival data: application to HIV infection in Danish homosexual men. Statistics in Medicine, 15(20):2177-2189, 1996.

CP Farrington: Interval censored survival data: a generalized linear modelling approach. Statistics in Medicine, 15(3):283-292, 1996.

See Also

`Icens` `fit.add`

Examples

```
data( HIV.dk )
```

float *Calculate floated variances*

Description

Given a fitted model object, the `float()` function calculates floating variances (a.k.a. quasi-variances) for a given factor in the model.

Usage

```
float(object, factor, iter.max=50)
```

Arguments

`object`	a fitted model object
`factor`	character string giving the name of the factor of interest. If this is not given, the first factor in the model is used.
`iter.max`	Maximum number of iterations for EM algorithm

Details

The `float()` function implements the ”floating absolute risk” proposal of Easton, Peto and Babiker(1992). This is an alternative way of presenting parameter estimates for factors in regression models, which avoids some of the difficulties of treatment contrasts. It was originally designed for epidemiological studies of relative risk, but the idea is widely applicable.

Treatment contrasts are not orthogonal. Consequently, the variances of treatment contrast estimates may be inflated by a poor choice of reference level, and the correlations between them may also be high. The `float()` function associates each level of the factor with a ”floating” variance (or quasi-variance), including the reference level. Floating variances are not real variances, but they can be used to calculate the variance error of contrast by treating each level as independent.

Plummer (2003) showed that floating variances can be derived from a covariance structure model applied to the variance-covariance matrix of the contrast estimates. This model can be fitted by minimizing the Kullback-Leibler information divergence between the true distribution of the parameter estimates and the simplified distribution given by the covariance structure model. Fitting is done using the EM algorithm.

In order to check the goodness-of-fit of the floating variance model, the `float()` function compares the standard errors predicted by the model with the standard errors derived from the true variance-covariance matrix of the parameter contrasts. The maximum and minimum ratios between true and model-based standard errors are calculated over all possible contrasts. These should be within 5 percent, or the use of the floating variances may lead to invalid confidence intervals.

Value

An object of class `floated`. This is a list with the following components

`coef`	A vector of coefficients. These are the same as the treatment contrasts but the reference level is present with coefficient 0.
`var`	A vector of floating (or quasi-) variances
`limits`	The bounds on the accuracy of standard errors over all possible contrasts

Note

Menezes(1999) and Firth and Menezes (2004) take a slightly different approach to this problem, using a pseudo-likelihood approach to fit the quasi-variance model. Their work is implemented in the package qvcalc.

Author(s)

Martyn Plummer

References

Easton DF, Peto J and Babiker GAG (1991) Floating absolute risk: An alternative to relative risk in survival and case control analysis avoiding an arbitrary reference group. *Statistics in Medicine*, 10, 1025-1035.

Firth D and Mezezes RX (2004) Quasi-variances. *Biometrika* 91, 65-80.

Menezes RX(1999) More useful standard errors for group and factor effects in generalized linear models. *D.Phil. Thesis*, Department of Statistics, University of Oxford.

Plummer M (2003) Improved estimates of floating absolute risk, *Statistics in Medicine*, 23, 93-104.

See Also

`ftrend`, `qvcalc`

`ftrend` *Fit a floating trend to a factor in generalized linear model*

Description

Fits a ”floating trend” model to the given factor in a glm in a generalized linear model by centering covariates.

Usage

```
ftrend(object, ...)
```

Arguments

`object`	fitted `lm` or `glm` object. The model must not have an intercept term
`...`	arguments to the `nlm` function

Details

`ftrend()` calculates ”floating trend” estimates for factors in generalized linear models. This is an alternative to treatment contrasts suggested by Greenland et al. (1999). If a regression model is fitted with no intercept term, then contrasts are not used for the first factor in the model. Instead, there is one parameter for each level of this factor. However, the interpretation of these parameters, and their variance-covariance matrix, depends on the numerical coding used for the covariates. If an arbitrary constant is added to the covariate values, then the variance matrix is changed.

The `ftrend()` function takes the fitted model and works out an optimal constant to add to the covariate values so that the covariance matrix is approximately diagonal. The parameter estimates can then be treated as approximately independent, thus simplifying their presentation. This is particularly useful for graphical display of dose-response relationships (hence the name).

Greenland et al. (1999) originally suggested centring the covariates so that their weighted mean, using the fitted weights from the model, is zero. This heuristic criterion is improved upon by `ftrend()` which uses the same minimum information divergence criterion as used by Plummer (2003) for floating variance calculations. `ftrend()` calls `nlm()` to do the minimization and will pass optional arguments to control it.

Value

A list with the following components

`coef` coefficients for model with adjusted covariates.

`vcov` Variance-covariance matrix of adjusted coefficients.

Note

The ”floating trend” method is an alternative to the ”floating absolute risk” method, which is implemented in the function `float()`.

Author(s)

Martyn Plummer

References

Greenland S, Michels KB, Robins JM, Poole C and Willet WC (1999) Presenting statistical uncertainty in trends and dose-response relations, *American Journal of Epidemiology*, 149, 1077-1086.

See Also

`float`

gmortDK *Population mortality rates for Denmark in 5-years age groups.*

Description

The `gmortDK` data frame has 418 rows and 21 columns.

Format

This data frame contains the following columns:

`agr`: Age group, 0:0–4, 5:5–9,..., 90:90+.
`per`: Calendar period, 38: 1938–42, 43: 1943–47, ..., 88:1988-92.
`sex`: Sex, 1: male, 2: female.
`risk`: Number of person-years in the Danish population.
`dt`: Number of deaths.
`rt`: Overall mortality rate in cases per 1000 person-years, i.e. `rt=1000*dt/risk`
Cause-specific mortality rates in cases per 1000 person-years:
`r1`: Infections
`r2`: Cancer.
`r3`: Tumors, benign, unspecific nature.
`r4`: Endocrine, metabolic.
`r5`: Blood.
`r6`: Nervous system, psychiatric.
`r7`: Cerebrovascular.
`r8`: Cardiac.
`r9`: Respiratory diseases, excl. cancer.
`r10`: Liver, excl. cancer.
`r11`: Digestive, other.
`r12`: Genitourinary.
`r13`: Ill-defined symptoms.
`r14`: All other, natural.
`r15`: Violent.

Source

Statistics Denmark, National board of health provided original data. Michael Andersson grouped the causes of death.

See Also

thoro, mortDK

Examples

```
data(gmortDK)
```

---

hivDK *hivDK: seroconversion in a cohort of Danish men*

Description

Data from a survey of HIV-positivity of a cohort of Danish men followed by regular tests from 1983 to 1989.

Usage

```
data(hivDK)
```

Format

A data frame with 297 observations on the following 7 variables.

`id` ID of the person

`entry` Date of entry to the study. Date variable.

`well` Date last seen seronegative. Date variable.

`ill` Date first seen seroconverted. Date variable.

`bth` Year of birth minus 1950.

`pyr` Annual number of sexual partners.

`us` Indicator of wheter the person has visited the USA.

Source

Mads Melbye, Statens Seruminstitut.

References

Becker N.G. and Melbye M.: Use of a log-linear model to compute the empirical survival curve from interval-censored data, with application to data on tests for HIV-positivity, Australian Journal of Statistics, 33, 125–133, 1990.

Melbye M., Biggar R.J., Ebbesen P., Sarngadharan M.G., Weiss S.H., Gallo R.C. and Blattner W.A.: Seroepidemiology of HTLV-III antibody in Danish homosexual men: prevalence, transmission and disease outcome. British Medical Journal, 289, 573–575, 1984.

Examples

```
data(hivDK)
str(hivDK)
```

Icens *Fits a regression model to interval censored data.*

Description

The models fitted assumes a piecewise constant baseline rate in intervals specified by the argument `breaks`, and for the covariates either a multiplicative relative risk function (default) or an additive excess risk function.

Usage

```
Icens( first.well, last.well, first.ill,
       formula, model.type=c("MRR","AER"), breaks,
       boot=FALSE, alpha=0.05, keep.sample=FALSE,
       data )
```

Arguments

`first.well` Time of entry to the study, i.e. the time first seen without event. Numerical vector.

`last.well` Time last seen without event. Numerical vector.

`first.ill` Time first seen with event. Numerical vector.

`formula` Model formula for the log relative risk.

`model.type` Which model should be fitted.

`breaks` Breakpoints between intervals in which the underlying timescale is assumed constant. Any observation outside the range of `breaks` is discarded.

`boot` Should bootstrap be performed to produce confidence intervals for parameters. If a number is given this will be the number of bootsrap samples. The default is 1000.

`alpha` 1 minus the confidence level.

`keep.sample` Should the bootstrap sample of the parameter values be returned?

`data` Data frame in which the times and formula are interpreted.

Details

The model is fitted by calling either `fit.mult` or `fit.add`.

Value

An object of class `"Icens"`: a list with three components:

`rates` A glm object from a binomial model with log-link, estimating the baseline rates, and the excess risk if `"AER"` is specfied.

`cov` A glm object from a binomial model with complementary log-log link, estimating the log-rate-ratios. Only if `"MRR"` is specfied.

`niter` Nuber of iterations, a scalar

`boot.ci` If `boot=TRUE`, a 3-column matrix with estimates and 1-`alpha` confidence intervals for the parameters in the model.

`sample` A matrix of the parameterestimates from the bootstrapping. Rows refer to parameters, columns to bootstrap samples.

Author(s)

Martyn Plummer, `<plummer@iarc.fr>`, Bendix Carstensen, `<bxc@steno.dk>`

References

B Carstensen: Regression models for interval censored survival data: application to HIV infection in Danish homosexual men. Statistics in Medicine, 15(20):2177-2189, 1996.

CP Farrington: Interval censored survival data: a generalized linear modelling approach. Statistics in Medicine, 15(3):283-292, 1996.

See Also

fit.add fit.mult

Examples

```
data( hivDK )
# Convert the dates to fractional years so that rates are
# expressed in cases per year
for( i in 2:4 ) hivDK[,i] <- cal.yr( hivDK[,i] )

m.RR <- Icens( entry, well, ill,
               model="MRR", formula=~pyr+us, breaks=seq(1980,1990,5),
               data=hivDK)
# Currently the MRR model returns a list with 2 glm objects.
round( ci.lin( m.RR$rates ), 4 )
round( ci.lin( m.RR$cov, Exp=TRUE ), 4 )
# There is actually a print method:
print( m.RR )

m.ER <- Icens( entry, well, ill,
               model="AER", formula=~pyr+us, breaks=seq(1980,1990,5),
               data=hivDK)
# There is actually a print method:
print( m.ER )
```

lep

An unmatched case-control study of leprosy incidence

Description

The `lep` data frame has 1370 rows and 7 columns. This was an unmatched case-control study in which incident cases of leprosy in a region of N. Malawi were compared with population controls.

Format

This data frame contains the following columns:

`id`: subject identifier: a numeric vector
`d`: case/control status: a numeric vector (1=case, 0=control)
`age`: a factor with levels `5-9` `10-14` `15-19` `20-24` `25-29` `30-44` `45+`
`sex`: a factor with levels `male`, `female`
`bcg`: presence of vaccine scar, a factor with levels `no` `yes`
`school`: schooling, a factor with levels `none` `1-5yrs` `6-8yrs` `sec/tert`
`house`: housing, a factor with levels `brick` `sunbrick` `wattle` `temp`

Source

The study is described in more detail in Clayton and Hills, Statistical Models in Epidemiology, Oxford University Press, Oxford:1993.

Examples

```
data(lep)
```

`Lexis` *Create a Lexis object*

Description

Create an object of class `Lexis` to represent follow-up on multiple time scales.

Usage

```
Lexis(entry, exit, duration, entry.status = 0, exit.status = 0, id, data,
      merge=TRUE, states )
```

Arguments

`entry`	a named list of entry times. Each element of the list is a numeric variable representing the entry time on the named time scale. All time scales must have the same units (e.g. years). The names of the timescales must be different from any column name in `date`.
`exit`	a named list of exit times.
`duration`	a numeric vector giving the duration of follow-up.
`entry.status`	a vector or a factor giving the status at entry
`exit.status`	a vector or factor giving status at exit. Any change in status during follow-up is assumed to take place exactly at the exit time.
`id`	a vector giving a unique identity value for each row of the Lexis object.
`data`	an optional data frame, list, or environment containing the variables. If not found in `data`, the variables are taken from the environment from which `Lexis` was called.
`merge`	a logical flag. If `TRUE` then the `data` argument will be coerced to a data frame and then merged with the resulting `Lexis` object.
`states`	A vector of labels for the states. If given, the state variables `lex.Cst` and `lex.Xst` are returned as factors with identical levels attributes.

Details

The analysis of long-term population-based follow-up studies typically requires multiple time scales to be taken into account, such as age, calender time, or time since an event. A `Lexis` object is a data frame with additional attributes that allows these multiple time dimensions of follow-up to be managed.

Separate variables for current end exit state allows representation of multistate data.

Lexis objects are named after the German demographer Wilhelm Lexis (1837-1914), who is credited with the invention of the ”Lexis diagram” for representing population dynamics simultaneously by several timescales.

The `Lexis` function creates a minimal `Lexis` object with only those variables required to define the follow-up history in each row. Additional variables can be merged into the `Lexis` object using the `merge` method for `Lexis` objects. This is the default.

There are also `merge`, `subset` and `transform` methods for `Lexis` objects. They work as the corresponding methods for data-frames but ensures that the result is a `Lexis` object.

Value

An object of class Lexis. This is represented as a data frame with a column for each time scale, and additional columns with the following names:

lex.id	Identification of the inidvdiual
lex.dur	Duration of follow-up
lex.Cst	Entry status (Current state), i.e. the state in which the follow up takes place.
lex.Xst	Exit status (eXit state), i.e. that state taken up after dur in lex.Cst.

If merge=TRUE then the Lexis object will also contain all variables from the data argument.

Note

Only two of the three arguments entry, exit and duration need to be given. If the third parameter is missing, it is imputed. If duration is given, it must be the same on all time scales.

entry, exit must be numeric, using Date variables will cause some of the utilites to crash. Transformation by cal.yr is recommended.

If only either exit or duration are supplied it is assumed that entry is 0. This is only meaningful (and therefore checked) if there is only one timescale.

If any of entry.status or exit.status are of mode character, they will both be converted to factors.

If entry.status is not given, then its class is automatically set to that of exit.status. If exit.status is factor, the value of entry.status is set to the first level. This may be highly undesirable, and therefore noted. For example, if exit.status is character the first level will be the first in the alphabetical ordering; slightly unfortunate if values are c("Well","Diseased"). If exit.status is logical, the value of entry.status set to FALSE.

If entry.status or exit.status are factors or character, the corresponding state variables in the returned Lexis object, lex.Cst and lex.Xst will be (unordered) factors with identical levels, namely the union of the levels of entry.status and exit.status.

Author(s)

Martyn Plummer

See Also

plot.Lexis, splitLexis, cutLexis, merge.Lexis, subset.Lexis, transform.Lexis, summary.Lexis, timeScales, timeBand, entry, exit, dur

Examples

```
# A small bogus cohort
xcoh <- structure( list( id = c("A", "B", "C"),
                      birth = c("14/07/1952", "01/04/1954", "10/06/1987"),
                      entry = c("04/08/1965", "08/09/1972", "23/12/1991"),
                       exit = c("27/06/1997", "23/05/1995", "24/07/1998"),
                       fail = c(1, 0, 1) ),
                     .Names = c("id", "birth", "entry", "exit", "fail"),
                  row.names = c("1", "2", "3"),
                      class = "data.frame" )

# Convert the character dates into numerical variables (fractional years)
xcoh <- cal.yr( xcoh, format="%d/%m/%Y", wh=2:4 )
# See how it looks
xcoh
```

```
# Define as Lexis object with timescales calendar time and age
Lcoh <- Lexis( entry = list( per=entry ),
                exit = list( per=exit, age=exit-birth ),
         exit.status = fail,
                data = xcoh )
Lcoh

# Using character states may have undesired effects:
xcoh$Fail <- c("Dead","Well","Dead")
Lexis( entry = list( per=entry ),
        exit = list( per=exit, age=exit-birth ),
 exit.status = Fail,
        data = xcoh )

# unless you order the levels correctly
( xcoh$Fail <- factor( xcoh$Fail, levels=c("Well","Dead") ) )
Lexis( entry = list( per=entry ),
        exit = list( per=exit, age=exit-birth ),
 exit.status = Fail,
        data = xcoh )
```

Lexis.diagram *Plot a Lexis diagram*

Description

Draws a Lexis diagram, optionally with life lines from a cohort, and with lifelines of a cohort if supplied. Intended for presentation purposes.

Usage

```
Lexis.diagram( age = c( 0, 60),
              alab = "Age",
              date = c( 1940, 2000 ),
              dlab = "Calendar time",
               int = 5,
           lab.int = 2*int,
          col.life = "black",
          lwd.life = 2,
          age.grid = TRUE,
         date.grid = TRUE,
          coh.grid = FALSE,
          col.grid = gray(0.7),
          lwd.grid = 1,
               las = 1,
        entry.date = NA,
         entry.age = NA,
         exit.date = NA,
          exit.age = NA,
         risk.time = NA,
        birth.date = NA,
              fail = NA,
          cex.fail = 1.1,
          pch.fail = c(NA,16),
          col.fail = rep( col.life, 2 ),
              data = NULL, ... )
```

Arguments

`age` Numerical vector of length 2, giving the age-range for the diagram

`alab` Label on the age-axis.

`date` Numerical vector of length 2, giving the calendar time-range for the diagram

`dlab` label on the calendar time axis.

`int` The interval between grid lines in the diagram. If a vector of length two is given, the first value will be used for spacing of age-grid and the second for spacing of the date grid.

`lab.int` The interval between labelling of the grids.

`col.life` Colour of the life lines.

`lwd.life` Width of the life lines.

`age.grid` Should grid lines be drawn for age?

`date.grid` Should grid lines be drawn for date?

`coh.grid` Should grid lines be drawn for birth cohorts (diagonals)?

`col.grid` Colour of the grid lines.

`lwd.grid` Width of the grid lines.

`las` How are the axis labels plotted?

`entry.date, entry.age, exit.date, exit.age, risk.time, birth.date`
Numerical vectors defining lifelines to be plotted in the diagram. At least three must be given to produce lines. Not all subsets of three will suffice, the given subset has to define life lines. If insufficient data is given, no life lines are produced.

`fail` Logical of event status at exit for the persons whose life lines are plotted.

`pch.fail` Symbols at the end of the life lines for censorings (`fail==0`) and failures (`fail != 0`).

`cex.fail` Expansion of the status marks at the end of life lines.

`col.fail` Character vector of length 2 giving the colour of the failure marks for censorings and failures respectively.

`data` Dataframe in which to interpret the arguments.

`...` Arguments to be passed on to the initial call to plot.

Details

The default unit for supplied variables are (calendar) years. If any of the variables `entry.date`, `exit.date` or `birth.date` are of class ”`Date`” or if any of the variables `entry.age`, `exit.age` or `risk.time` are of class ”`difftime`”, they will be converted to calendar years, and plotted correctly in the diagram. The returned dataframe will then have colums of classes ”`Date`” and ”`difftime`”.

Value

If sufficient information on lifelines is given, a data frame with one row per person and columns with entry ages and dates, birth date, risk time and status filled in.

Side effect: a plot of a Lexis diagram is produced with the life lines in it is produced. This will be the main reason for using the function. If the primary aim is to illustrate follow-up of a cohort, then it is better to represent the follow-up in a `Lexis` object, and use the generic `plot.Lexis` function.

Author(s)

Bendix Carstensen, `http://www.biostat.ku.dk/~bxc`

See Also

`Life.lines`, `Lexis.lines`

Examples

```
Lexis.diagram( entry.age = c(3,30,45),
               risk.time = c(25,5,14),
              birth.date = c(1970,1931,1925.7),
                    fail = c(TRUE,TRUE,FALSE) )
LL <- Lexis.diagram( entry.age = sample( 0:50, 17, replace=TRUE ),
                     risk.time = sample( 5:40, 17, r=TRUE),
                    birth.date = sample( 1910:1980, 17, r=TRUE ),
             fail = sample( 0:1, 17, r=TRUE ),
              cex.fail = 1.1,
              lwd.life = 2 )
# Identify the persons' entry and exits
text( LL$exit.date, LL$exit.age, paste(1:nrow(LL)), col="red", font=2, adj=c(0,1) )
text( LL$entry.date, LL$entry.age, paste(1:nrow(LL)), col="blue", font=2, adj=c(1,0) )
data( nickel )
attach( nickel )
LL <- Lexis.diagram( age=c(10,100), date=c(1900,1990),
             entry.age=age1st, exit.age=ageout, birth.date=dob,
     fail=(icd %in% c(162,163)), lwd.life=1,
     cex.fail=0.8, col.fail=c("green","red") )
abline( v=1934, col="blue" )
nickel[1:10,]
LL[1:10,]
```

Lexis.lines *Draw life lines in a Lexis diagram.*

Description

Add life lines to a Lexis diagram.

Usage

```
Lexis.lines( entry.date = NA,
              exit.date = NA,
             birth.date = NA,
              entry.age = NA,
               exit.age = NA,
              risk.time = NA,
               col.life = "black",
               lwd.life = 2,
                   fail = NA,
               cex.fail = 1,
               pch.fail = c(NA, 16),
               col.fail = col.life,
                   data = NULL )
```

Arguments

`entry.date, entry.age, exit.date, exit.age, risk.time, birth.date`
: Numerical vectors defining lifelines to be plotted in the diagram. At least three must be given to produce lines. Not all subsets of three will suffice, the given subset has to define life lines. If insufficient data is given, no life lines are produced.

`col.life`
: Colour of the life lines.

`lwd.life`	Width of the life lines.
`fail`	Logical of event status at exit for the persons whose life lines are plotted.
`cex.fail`	The size of the status marks at the end of life lines.
`pch.fail`	The status marks at the end of the life lines.
`col.fail`	Colour of the marks for censorings and failures respectively.
`data`	Data frame in which to interpret values.

Value

If sufficient information on lifelines is given, a data frame with one row per person and columns with entry ages and dates, birth date, risk time and status filled in.

Side effect: Life lines are added to an existing Lexis diagram. Lexis.lines adds life lines to an existing plot.

Author(s)

Bendix Carstensen, Steno Diabetes Center, http://www.biostat.ku.dk/~bxc

See Also

`Lexis.diagram`, `Life.lines`

Examples

```
Lexis.diagram( entry.age = c(3,30,45),
               risk.time = c(25,5,14),
              birth.date = c(1970,1931,1925.7),
                    fail = c(TRUE,TRUE,FALSE) )
Lexis.lines( entry.age = sample( 0:50, 100, replace=TRUE ),
             risk.time = sample( 5:40, 100, r=TRUE),
            birth.date = sample( 1910:1980, 100, r=TRUE ),
                  fail = sample(0:1,100,r=TRUE),
              cex.fail = 0.5,
              lwd.life = 1 )
```

`Life.lines` *Compute dates/ages for life lines in a Lexis diagram*

Description

Fills out the missing information for follow up of persons in a Lexis diagram if sufficient information is given.

Usage

```
Life.lines( entry.date = NA,
             exit.date = NA,
            birth.date = NA,
             entry.age = NA,
              exit.age = NA,
             risk.time = NA )
```

Arguments

`entry.date, exit.date,birth.date, entry.age, exit.age, risk.time`
: Vectors defining lifelines to be plotted in the diagram. At least three must be given to produce a result. Not all subsets of three will suffice, the given subset has to define life lines. If insufficient data is given, nothing is returned and a warning is given.

Value

Data frame with variables `entry.date`, `entry.age`, `exit.date`, `exit.age`, `risk.time`, `birth.date`, with all entries computed for each person. If any of `entry.date`, `exit.date` or `birth.date` are of class `Date` or if any of `entry.age`, `exit.age` or `risk.time` are of class `difftime` the date variables will be of class `Date` and the other three of class `difftime`.

See Also

`Lexis.diagram`, `Lexis.lines`

Examples

```
( Life.lines( entry.age = c(3,30,45),
              risk.time = c(25,5,14),
             birth.date = c(1970,1931,1925.7) ) )

# Draw a Lexis diagram
Lexis.diagram()

# Compute entry and exit age and date.
( LL <-  Life.lines( entry.age = c(3,30,45),
                     risk.time = c(25,5,14),
                    birth.date = c(1970,1931,1925.7) ) )
segments( LL[,1], LL[,2], LL[,3], LL[,4] ) # Plot the life lines.

# Compute entry and exit age and date, supplying a date variable
bd <- ( c(1970,1931,1925.7) - 1970 ) * 365.25
class( bd ) <- "Date"
( Life.lines( entry.age = c(3,30,45),
              risk.time = c(25,5,14),
             birth.date = bd ) )
```

---

lls *Functions to manage and explore the workspace*

Description

These functions help you to find out what has gone wrong and to start afresh if needed.

Usage

```
lls(pos = 1, pat = "", all=FALSE, print=TRUE )
clear()
```

Arguments

`pos`	Numeric. What position in the search path do you want listed.
`pat`	Character. List only objects that have this string in their name.

`all` Logical. Should invisible objects be printed too - see `ls` to which this argument is passed.

`print` Logical. Should the result be printed?

Details

`lls` is designed to give a quick overview of the name, mode, class and dimension of the object in your workspace. They may not always be what you think they are.

`clear` clears all your objects from workspace, and all attached objects too — it only leaves the loaded packages in the search path; thus allowing a fresh start without closing and restarting R.

Value

`lls` returns a data frame with four character variables: codename, codemode, codeclass and codesize and one row per object in the workspace (if `pos=1`). `size` is either the length or the dimension of the object. The data frame is by default printed with left-justified columns.

Author(s)

`lls`: Unknown. Modified by Bendix Carstensen from a long forgotten snatch.

`clear`: Michael Hills / David Clayton.

Examples

```
x <- 1:10
y <- rbinom(10, 1, 0.5)
m1 <- glm( y ~ x, family=binomial )
M <- matrix( 1:20, 4, 5 )
.M <- M
lls()
clear()
lls()
```

`lungDK` *Male lung cancer incidence in Denmark*

Description

Male lung cancer cases and population riks time in Denmark, for the period 1943–1992 in ages 40–89.

Usage

```
data(lungDK)
```

Format

A data frame with 220 observations on the following 9 variables.

`A5`: Left end point of the age interval, a numeric vector.
`P5`: Left enpoint of the period interval, a numeric vector.
`C5`: Left enpoint of the birth cohort interval, a numeric vector.
`up`: Indicator of upper trianges of each age by period rectangle in the Lexis diagram. (`up=(P5-A5-C5)/5`).
`Ax`: The mean age of diagnois (at risk) in the triangle.
`Px`: The mean date of diagnosis (at risk) in the triangle.
`Cx`: The mean date of birth in the triangle, a numeric vector.

`D`: Number of diagnosed cases of male lung cancer.
`Y`: Risk time in the male population, person-years.

Details

Cases and person-years are tabulated by age and date of diagnosis (period) as well as date of birth (cohort) in 5-year classes. Each observation in the dataframe correponds to a triangle in a Lexis diagram. Triangles are classified by age and date of diagnosis, period of diagnosis and date of birth, all in 5-year groupings.

Source

The Danish Cancer Registry and Statistics Denmark.

References

For a more thorough exposition of statistical inference in the Lexis diagram, see:
`http://staff.pubhealth.ku.dk/~bxc/APC/notes.pdf`

Examples

```
data( lungDK )
# Draw a Lexis diagram and show the number of cases in it.
attach( lungDK )
Lexis.diagram( age=c(40,90), date=c(1943,1993), coh.grid=TRUE )
text( Px, Ax, paste( D ), cex=0.7 )
```

`merge.data.frame` *Merge data frame with a Lexis object*

Description

Merge two data frames, or a data frame with a `Lexis` object.

Usage

```
## S3 method for class 'data.frame'
merge(x, y, ...)
```

Arguments

`x, y`	data frames, or objects to be coerced into one
`...`	optional arguments for the merge method

Details

This version of `merge.default` masks the one in the `base`. It ensures that, if either `x` or `y` is a `Lexis` object, then `merge.Lexis` is called.

Value

A merged `Lexis` object or data frame.

Author(s)

Martyn Plummer

See Also

Lexis

merge.Lexis *Merge a Lexis object with a data frame*

Description

Merge additional variables from a data frame into a Lexis object.

Usage

```
## S3 method for class 'Lexis'
merge(x, y, id, by, ...)
```

Arguments

x	an object of class Lexis
y	a data frame
id	the name of the variable in y to use for matching against the variable lex.id in x.
by	if matching is not done by id, a vector of variable names common to both x and y
...	optional arguments to be passed to merge.data.frame

Details

A Lexis object can be considered as an augmented data frame in which some variables are time-dependent variables representing follow-up. The Lexis function produces a minimal object containing only these time-dependent variables. Additional variables may be added to a Lexis object using the merge method.

Value

A Lexis object with additional columns taken from the merged data frame.

Note

The variable given as the by.y argument must not contain any duplicate values in the data frame y.

Author(s)

Martyn Plummer

See Also

merge.data.frame, subset.Lexis

mh — *Mantel-Haenszel analyses of cohort and case-control studies*

Description

This function carries out Mantel-Haenszel comparisons in tabulated data derived from both cohort and case-control studies.

Usage

```
mh(cases, denom, compare=1, levels=c(1, 2), by=NULL,
     cohort=!is.integer(denom), confidence=0.9)
```

Arguments

`cases`	the table of case frequencies (a multiway array).
`denom`	the denominator table. For cohort studies this should be a table of person-years observation, while for case-control studies it should be a table of control frequencies.
`compare`	the dimension of the table which defines the comparison groups (can be referred to either by number or by name). The default is the first dimension of the table.
`levels`	a vector identifying (either by number or by name) the two groups to be compared. The default is the first two levels of the selected dimension.
`by`	the dimensions not to be collapsed in the Mantel-Haenszel computations. Thus, this argument defines the structure of the resulting tables of estimates and tests.
`cohort`	an indicator whether the data derive from a cohort or a case-control study. If the denominator table is stored as an integer, a case-control study is assumed.
`confidence`	the approximate coverage probability for the confidence intervals to be computed.

Details

Multiway tables of data are accepted and any two levels of any dimension can be chosen as defining the comparison groups. The rate (odds) ratio estimates and the associated significance tests may be collapsed over all the remaining dimensions of the table, or over selected dimensions only, so that tables of estimates and tests are computed.

Value

A list giving tables of rate (odds) ratio estimates, their standard errors (on a log scale), lower and upper confidence limits, chi-squared tests (1 degree of freedom) and the corresponding p-values. The result list also includes numerator and denominator of the Mantel-Haenszel estimates (q, r), and score test statistics and score variance (u, v).

Side Effects

None

References

Clayton, D. and Hills, M. : Statistical Models in Epidemiology, Oxford University Press (1993).

See Also

`Lexis`

Examples

```
# If d and y are 3-way tables of cases and person-years
# observation formed by tabulation by two confounders
# (named "C1" and "C2") an exposure of interest ("E"),
# the following command will calculate an overall
# Mantel-Haenszel comparison of the first two exposure
# groups.
#
# Generate some bogus data
dnam <- list( E=c("low","medium","high"), C1=letters[1:2], C2=LETTERS[1:4] )
d <- array( sample( 2:80, 24),
            dimnames=dnam, dim=sapply( dnam, length ) )
y <- array( abs( rnorm( 24, 227, 50 ) ),
            dimnames=dnam, dim=sapply( dnam, length ) )
mh(d, y, compare="E")
#
# Or, if exposure levels named "low" and "high" are to be
# compared and these are not the first two levels of E :
#
mh(d, y, compare="E", levels=c("low", "high"))
#
# If we wish to carry out an analysis which controls for C1,
# but examines the results at each level of C2:
#
mh(d, y, compare="E", by="C2")
#
# It is also possible to look at rate ratios for every
# combination of C1 and C2 :
#
mh(d, y, compare="E", by=c("C1", "C2"))
#
# If dimensions and levels of the table are unnamed, they must
# be referred to by number.
#
```

`mortDK` *Population mortality rates for Denmark in 1-year age-classes.*

Description

The `mortDK` data frame has 1820 rows and 21 columns.

Format

This data frame contains the following columns:

`age`: Age class, 0–89, 90:90+.
`per`: Calendar period, 38: 1938–42, 43: 1943–47, ..., 88:1988-92.
`sex`: Sex, 1: male, 2: female.
`risk`: Number of person-years in the Danish population.
`dt`: Number of deaths.
`rt`: Overall mortality rate in cases per 1000 person-years, i.e. `rt=1000*dt/risk`
Cause-specific mortality rates in cases per 1000 person-years:
`r1`: Infections
`r2`: Cancer.

r3:	Tumors, benign, unspecific nature.
r4:	Endocrine, metabolic.
r5:	Blood.
r6:	Nervous system, psychiatric.
r7:	Cerebrovascular.
r8:	Cardiac.
r9:	Respiratory diseases, excl. cancer.
r10:	Liver, excl. cancer.
r11:	Digestive, other.
r12:	Genitourinary.
r13:	Ill-defined symptoms.
r14:	All other, natural.
r15:	Violent.

Source

Statistics Denmark, National board of health provided original data. Michael Andersson grouped the causes of death.

See Also

thoro, gmortDK

Examples

```
data(mortDK)
```

msdata.Lexis *Create a dataframe suitable for use with the mstate package.*

Description

The mstate package requires input in the form of a stacked dataset with specific variable names. This is provided by this function. The resulting dataframe contains the same information as the result of a call to stack.Lexis.

Usage

```
msdata(obj, ...)
## S3 method for class 'Lexis'
msdata(obj, time.scale = timeScales(obj)[1], ...)
```

Arguments

obj	A Lexis object.
time.scale	Name or number of timescale in the Lexis object.
...	Not used.

Value

A dataframe with the Lexis specific variables stripped, and with the following added: id, Tstart, Tstop, from, to, trans, status, which are used in the mstate package.

Author(s)

Bendix Carstensen, <bxc@steno.dk>, www.biostat.ku.dk/~bxc

See Also

stack.Lexis

Examples

```
data(DMlate)
str(DMlate)
dml <- Lexis( entry=list(Per=dodm,Age=dodm-dobth,DMdur=0),
               exit=list(Per=dox),
        exit.status=factor(!is.na(dodth),labels=c("DM","Dead")),
               data=DMlate )
dmi <- cutLexis( dml, cut=dml$doins, new.state="Ins", pre="DM" )
summary( dmi )
ms.dmi <- msdata.Lexis( dmi )
summary( dmi )
# Check that all the transitions and person-years got across.
with( ms.dmi, rbind( table(status,trans),
                     tapply(Tstop-Tstart,trans,sum) ) )
```

---

ncut *Function to group a variable in intervals.*

Description

Cuts a continuous variable in intervals. As opposed to `cut` which returns a factor, `ncut` returns a numeric variable.

Usage

```
ncut(x, breaks, type="left" )
```

Arguments

`x`	A numerical vector.
`breaks`	Vector of breakpoints. `NA` will results for values below `min(x)` if `type="left"`, for values above `max(x)` if `type="right"` and for values outside `range(x)` if `type="mid"`
`type`	Character: one of `c("left","right","mid")`, indicating whether the left, right or midpoint of the intervals defined in breaks is returned.

Details

The function uses the base function `findInterval`.

Value

A numerical vector of the same length as `x`.

Author(s)

Bendix Carstensen, Steno Diabetes Center, <bxc@steno.dk>, http://www.biostat.ku.dk/~bxc/, with essential input from Martyn Plummer, IARC.

See Also

cut, findInterval

Examples

```
br <- c(-2,0,1,2.5)
x <- c( rnorm( 10 ), br, -3, 3 )
cbind( x, l=ncut( x, breaks=br, type="l" ),
          m=ncut( x, breaks=br, type="m" ),
          r=ncut( x, breaks=br, type="r" ) )[order(x),]
x <- rnorm( 200 )
plot( x, ncut( x, breaks=br, type="l" ), pch=16, col="blue", ylim=range(x) )
abline( 0, 1 )
abline( v=br )
points( x, ncut( x, breaks=br, type="r" ), pch=16, col="red" )
points( x, ncut( x, breaks=br, type="m" ), pch=16, col="green" )
```

nice *Nice breakpoints*

Description

The function calls pretty for linear scale. For a log-scale nice are computed using a set of specified number in a decade.

Usage

```
nice(x, log = F, lpos = c(1, 2, 5), ...)
```

Arguments

x	Numerical vector to
log	Logical. Is the scale logartimic?
lpos	Numeric. Numbers between 1 and 10 giving the desired breakpoints in this interval.
...	Arguments passed on to pretty if log=FALSE

Value

A vector of breakpoints.

Author(s)

Bendix Carstensen, <bxc@steno.dk>, http://www.biostat.ku.dk/~bxc

See Also

pretty

Examples

```
nice( exp( rnorm( 100 ) ), log=TRUE )
```

nickel *A Cohort of Nickel Smelters in South Wales*

Description

The nickel data frame has 679 rows and 7 columns. The data concern a cohort of nickel smelting workers in South Wales and are taken from Breslow and Day, Volume 2. For comparison purposes, England and Wales mortality rates (per 1,000,000 per annum) from lung cancer (ICDs 162 and 163), nasal cancer (ICD 160), and all causes, by age group and calendar period, are supplied in the dataset ewrates.

Format

This data frame contains the following columns:

id:	Subject identifier (numeric)
icd:	ICD cause of death if dead, 0 otherwise (numeric)
exposure:	Exposure index for workplace (numeric)
dob:	Date of birth (numeric)
age1st:	Age at first exposure (numeric)
agein:	Age at start of follow-up (numeric)
ageout:	Age at end of follow-up (numeric)

Source

Breslow NE, and Day N, Statistical Methods in Cancer Research. Volume II: The Design and Analysis of Cohort Studies. IARC Scientific Publications, IARC:Lyon, 1987.

Examples

```
data(nickel)
str(nickel)
```

occup *A small occupational cohort*

Description

This is the data that is behind the illustrative Lexis diagram in Breslow & Day's book on case-control studies.

Usage

```
data(occup)
```

Format

A data frame with 13 observations on the following 4 variables.

AoE a numeric vector, Age at Entry

DoE a numeric vector, Date of entry

DoX a numeric vector, Date of eXit

Xst eXit status D-event, W-withdrawal, X-censoring

References

Breslow & Day: Statistical Methods in Cancer Research, vol 1: The analysis of case-control studies, figure 2.2, p. 48.

Examples

```
data(occup)
lx <- Lexis( entry = list( per=DoE, age=AoE ),
              exit = list( per=DoX ),
      entry.status = "W",
       exit.status = Xst,
              data = occup )
plot( lx )
# Split follow-up in 5-year classes
sx <- splitLexis( lx, seq(1940,1960,5), "per" )
sx <- splitLexis( sx, seq(  40,  60,5), "age" )
plot( sx )

# Plot with a bit more paraphernalia and a device to get
# the years on the same physical scale on both axes
ypi <- 2.5 # Years per inch
x11( height=15/ypi+1, width=20/ypi+1 )    # add an inch in each direction for
par( mai=c(3,3,1,1)/4, mgp=c(3,1,0)/1.6 ) # the margins set in inches by mai=
plot(sx,las=1,col="black",lty.grid=1,lwd=2,type="l",
     xlim=c(1940,1960),ylim=c(40,55),xaxs="i",yaxs="i",yaxt="n",
     xlab="Calendar year", ylab="Age (years)")
axis( side=2, at=seq(40,55,5), las=1 )
points(sx,pch=c(NA,16)[(sx$lex.Xst=="D")+1] )
box()
# Annotation with the person-years
PY.ann.Lexis( sx, cex=0.8 )
```

---

pctab *Create percentages in a table*

Description

Computes percentages and a margin of totals along a given margin of a table.

Usage

```
pctab(TT, margin = length(dim(TT)), dec=1)
```

Arguments

TT	A table or array object
margin	Which margin should be the the total?
dec	How many decimals should be printed? If 0 or FALSE nothing is printed

Value

A table, where all dimensions except the one specified margin has two extra levels named ”All” (where all entries are 100) and ”N”. The function prints the table with dec decimals.

Author(s)

Bendix Carstensen, Steno Diabetes Center, http://www.biostat.ku.dk/~bxc.

See Also

addmargins

Examples

```
Aye <- sample( c("Yes","Si","Oui"), 177, replace=TRUE )
Bee <- sample( c("Hum","Buzz"), 177, replace=TRUE )
Sea <- sample( c("White","Black","Red","Dead"), 177, replace=TRUE )
A <- table( Aye, Bee, Sea )
A
ftable( pctab( A ) )
ftable( pctab( addmargins( A, 1 ), 3 ) )
round( ftable( pctab( addmargins( A, 1 ), 3 ), row.vars=3 ), 1)
```

---

plot.Lexis *Lexis diagrams*

Description

The follow-up histories represented by a Lexis object can be plotted using one or two dimensions. The two dimensional plot is a Lexis diagram showing follow-up time simultaneously on two time scales.

Usage

```
## S3 method for class 'Lexis'
plot(x=Lexis( entry=list(Date=1900,Age=0), exit=list(Age=0) ),
                     time.scale = NULL, type="l", breaks="lightgray", ...)
## S3 method for class 'Lexis'
points(x, time.scale = options()[["Lexis.time.scale"]] , ...)
## S3 method for class 'Lexis'
lines(x, time.scale = options()[["Lexis.time.scale"]], ...)
## S3 method for class 'Lexis'
PY.ann(x, time.scale = options()[["Lexis.time.scale"]], digits=1, ...)
```

Arguments

`x`	An object of class `Lexis`. The default is a bogus `Lexis` object, so that `plot.Lexis` can be called without the first argument and still produce a(n empty) Lexis diagram. Unless arguments `xlim` and `ylim` are given in this case the diagram is looking pretty daft.
`time.scale`	A vector of length 1 or 2 giving the time scales to be plotted either by name or numerical order
`type`	Character indication what to draw: "n" nothing (just set up the diagram), "l" - liefelines, "p" - endpoints of follow-up, "b" - both lifelines and endpoints.
`breaks`	a string giving the colour of grid lines to be drawn when plotting a split Lexis object. Grid lines can be suppressed by supplying the value `NULL` to the `breaks` argument
`digits`	Numerical. How many digits after the demimal points should be when plotting the person-years.
`...`	Further graphical parameters to be passed to the plotting methods. Grids can be drawn (behind the life lines) using the following parameters in `plot`:

- `grid` If logical, a background grid is set up using the axis ticks. If a list, the first component is used as positions for the vertical lines and the last as positions for the horizontal. If a nunerical vector, grids on both axes are set up using the distance between the numbers.
- `col.grid="lightgray"` Color of the background grid.
- `lty.grid=2` Line type for the grid.
- `coh.grid=FALSE` Should a 45 degree grid be plotted?

Details

The plot method for `Lexis` objects traces "life lines" from the start to the end of follow-up. The `points` method plots points at the end of the life lines.

If `time.scale` is of length 1, the life lines are drawn horizontally, with the time scale on the X axis and the id value on the Y axis. If `time.scale` is of length 2, a Lexis diagram is produced, with diagonal life lines plotted against both time scales simultaneously.

If `lex` has been split along one of the time axes by a call to `splitLexis`, then vertical or horizontal grid lines are plotted (on top of the life lines) at the break points.

`PY.ann` writes the length of each (segment of) life line at the middle of the line. Not advisable to use with large cohorts. Another example is in the example file for `occup`.

Author(s)

Martyn Plummer

See Also

`Lexis`, `splitLexis`

Examples

```
# A small bogus cohort
xcoh <- structure( list( id = c("A", "B", "C"),
                      birth = c("14/07/1952", "01/04/1957", "10/06/1987"),
                      entry = c("04/08/1965", "08/09/1972", "23/12/1991"),
                       exit = c("27/06/1997", "23/05/1995", "24/07/1998"),
                       fail = c(1, 0, 1) ),
                     .Names = c("id", "birth", "entry", "exit", "fail"),
                  row.names = c("1", "2", "3"),
                      class = "data.frame" )

# Convert the character dates into numerical variables (fractional years)
xcoh$bt <- cal.yr( xcoh$birth, format="%d/%m/%Y" )
xcoh$en <- cal.yr( xcoh$entry, format="%d/%m/%Y" )
xcoh$ex <- cal.yr( xcoh$exit , format="%d/%m/%Y" )

# See how it looks
xcoh

# Define as Lexis object with timescales calendar time and age
Lcoh <- Lexis( entry = list( per=en ),
                exit = list( per=ex, age=ex-bt ),
         exit.status = fail,
                data = xcoh )

# Default plot of follow-up
plot( Lcoh )
```

```
# Show follow-up time
PY.ann( Lcoh )

# Show exit status
plot( Lcoh, type="b" )
# Same but failures only
plot( Lcoh, type="b", pch=c(NA,16)[Lcoh$fail+1] )

# With a grid and deaths as endpoints
plot( Lcoh, grid=0:10*10, col="black" )
points( Lcoh, pch=c(NA,16)[Lcoh$lex.Xst+1] )
# With a lot of bells and whistles:
plot( Lcoh, grid=0:20*5, col="black", xaxs="i", yaxs="i",
      xlim=c(1960,2010), ylim=c(0,50), lwd=3, las=1 )
points( Lcoh, pch=c(NA,16)[Lcoh$lex.Xst+1], col="red", cex=1.5 )
```

plotEst *Plot estimates with confidence limits*

Description

Plots parameter estimates with confidence intervals, annotated with parameter names. A dot is plotted at the estimate and a horizontal line extending from the lower to the upper limit is superimposed.

Usage

```
plotEst( ests,
            y = dim(ests)[1]:1,
          txt = rownames(ests),
       txtpos = y,
         ylim = range(y)-c(0.5,0),
         xlab = "",
         xtic = nice(ests[!is.na(ests)], log = xlog),
         xlim = range( xtic ),
         xlog = FALSE,
          pch = 16,
          cex = 1,
          lwd = 2,
          col = "black",
    col.lines = col,
   col.points = col,
         vref = NULL,
         grid = FALSE,
     col.grid = gray(0.9),
  restore.par = TRUE )

linesEst( ests, y = dim(ests)[1]:1, pch = 16, cex = 1, lwd = 2,
          col="black", col.lines=col, col.points=col )

pointsEst( ests, y = dim(ests)[1]:1, pch = 16, cex = 1, lwd = 2,
          col="black", col.lines=col, col.points=col )
```

Arguments

ests	Matrix with three columns: Estimate, lower limit, upper limit. If a model object is supplied, ci.lin is invoked for this object first.

`y`	Vertical position of the lines.
`txt`	Annotation of the estimates.
`txtpos`	Vertical position of the text. Defaults to `y`.
`ylim`	Extent of the vertical axis.
`xlab`	Annotation of the horizontal axis.
`xtic`	Location of tickmarks on the x-axis.
`xlim`	Extent of the x-axis.
`xlog`	Should the x-axis be logarithmic?
`pch`	What symbol should be used?
`cex`	Expansion of the symbol.
`col`	Colour of the points and lines.
`col.lines`	Colour of the lines.
`col.points`	Colour of the symbol.
`lwd`	Thickness of the lines.
`vref`	Where should vertical reference line(s) be drawn?
`grid`	If TRUE, vertical gridlines are drawn at the tickmarks. If a numerical vector is given vertical lines are drawn at `grid`.
`col.grid`	Colour of the vertical gridlines
`restore.par`	Should the graphics parameters be restored? If set to `FALSE` the coordinate system will still be available for additional plotting, and `par("mai")` will still have the very large value set in order to make room for the labelling of the estimates.

Details

`plotEst` make a news plot, whereas `linesEst` and `pointsEst` (identical functions) adds to an existing plot.

Value

NULL

Author(s)

Bendix Carstensen, `<bxc@steno.dk>`, `http://www.pubhealth.ku.dk/~bxc`

See Also

ci.lin

Examples

```
# Bogus data and a linear model
f <- factor( sample( letters[1:5], 100, replace=TRUE ) )
x <- rnorm( 100 )
y <- 5 + 2 * as.integer( f ) + 0.8 * x + rnorm(100) * 2
m1 <- lm( y ~ f )

# Produce some confidence intervals for contrast to first level
( cf <- summary( m1 )$coef[2:5,1:2] %*% rbind( c(1,1,1), 1.96*(c(0,-1,1) ) ) )

# Plots with increasing amount of bells and whistles
par( mfcol=c(3,2), mar=c(3,3,2,1) )
```

```
plotEst( cf )
plotEst( cf, grid=TRUE )
plotEst( cf, grid=TRUE, cex=2, lwd=3 )
plotEst( cf, grid=TRUE, cex=2, col.points="red", col.lines="green" )
plotEst( cf, grid=TRUE, cex=2, col.points="red", col.lines="green",
           xlog=TRUE, xtic=c(1:8), xlim=c(0.8,6) )
rownames( cf )[1] <- "Contrast to fa:\n\n fb"
plotEst( cf, grid=TRUE, cex=2, col.points=rainbow(4), col.lines=rainbow(4), vref=1 )
```

plotevent *Plot Equivalence Classes*

Description

For interval censored data, segments of times between last.well and first.ill are plotted for each conversion in the data. It also plots the equivalence classes.

Usage

```
plotevent(last.well, first.ill, data)
```

Arguments

`last.well`	Time at which the individuals are last seen negative for the event
`first.ill`	Time at which the individuals are first seen positive for the event
`data`	Data with a transversal shape

Details

last.well and first.ill should be written as character in the function.

Value

Graph

Author(s)

Delphine Maucort-Boulch, Bendix Carstensen, Martyn Plummer

References

Carstensen B. Regression models for interval censored survival data: application to HIV infection in Danish homosexual men.Stat Med. 1996 Oct 30;15(20):2177-89.

Lindsey JC, Ryan LM. Tutorial in biostatistics methods for interval-censored data.Stat Med. 1998 Jan 30;17(2):219-38.

See Also

Icens

projection.ip *Projection of columns of a matrix.*

Description

Projects the columns of the matrix M on the space spanned by the columns of the matrix X, with respect to the inner product defined by weight: <x|y>=sum(x*w*y).

Usage

```
projection.ip(X, M, orth = FALSE, weight = rep(1, nrow(X)))
```

Arguments

X	Matrix defining the space to project onto.
M	Matrix of columns to be projected. Must have the same number of rows as X.
orth	Should the projection be on the orthogonal complement to span(X)?
weight	Weights defining the inner product. Numerical vector of length nrow(X).

Value

A matrix of full rank with columns in span(X).

Author(s)

Bendix Carstensen, Steno Diabetes Center, http://www.pubhealth.ku.dk/~bxc, with help from Peter Dalgaard.

See Also

detrend

rateplot *Functions to plot rates from a table classified by age and calendar time (period)*

Description

Produces plots of rates versus age, connected within period or cohort (Aplot), rates versus period connected within age-groups (Pplot) and rates and rates versus date of birth cohort (Cplot). rateplot is a wrapper for these, allowing to produce the four classical displays with a single call.

Usage

```
rateplot( rates,
          which = c("ap","ac","pa","ca"),
            age = as.numeric( dimnames( rates )[[1]] ),
            per = as.numeric( dimnames( rates )[[2]] ),
           grid = FALSE,
         a.grid = grid,
         p.grid = grid,
         c.grid = grid,
          ygrid = grid,
       col.grid = gray( 0.9 ),
          a.lim = range( age, na.rm=TRUE ) + c(0,diff( range( age ) )/30),
          p.lim = range( per, na.rm=TRUE ) + c(0,diff( range( age ) )/30),
          c.lim = NULL,
           ylim = range( rates[rates>0], na.rm=TRUE ),
             at = NULL,
         labels = paste( at ),
          a.lab = "Age at diagnosis",
          p.lab = "Date of diagnosis",
          c.lab = "Date of birth",
           ylab = "Rates",
           type = "l",
            lwd = 2,
            lty = 1,
         log.ax = "y",
            las = 1,
            ann = FALSE,
          a.ann = ann,
          p.ann = ann,
          c.ann = ann,
          xannx = 1/20,
        cex.ann = 0.8,
         a.thin = seq( 1, length( age ), 2 ),
         p.thin = seq( 1, length( per ), 2 ),
         c.thin = seq( 2, length( age ) + length( per ) - 1, 2 ),
            col = par( "fg" ),
          a.col = col,
          p.col = col,
          c.col = col,
            ... )

Aplot( rates, age = as.numeric( dimnames( rates )[[1]] ),
       per = as.numeric( dimnames( rates )[[2]] ), grid = FALSE,
       a.grid = grid, ygrid = grid, col.grid = gray( 0.9 ),
       a.lim = range( age, na.rm=TRUE ), ylim = range( rates[rates>0], na.rm=TRUE ),
       at = NULL, labels = paste( at ), a.lab = names( dimnames( rates ) )[1],
       ylab = deparse( substitute( rates ) ), type = "l", lwd = 2, lty = 1,
       col = par( "fg" ), log.ax = "y", las = 1, c.col = col, p.col = col,
       c.ann = FALSE, p.ann = FALSE, xannx = 1/20, cex.ann = 0.8,
       c.thin = seq( 2, length( age ) + length( per ) - 1, 2 ),
       p.thin = seq( 1, length( per ), 2 ), p.lines = TRUE,
       c.lines = !p.lines, ... )

Pplot( rates, age = as.numeric( dimnames( rates )[[1]] ),
       per = as.numeric( dimnames( rates )[[2]] ), grid = FALSE,
       p.grid = grid, ygrid = grid, col.grid = gray( 0.9 ),
       p.lim = range( per, na.rm=TRUE ) + c(0,diff(range(per))/30),
       ylim = range( rates[rates>0], na.rm=TRUE ), p.lab = names( dimnames( rates ) )[2],
       ylab = deparse( substitute( rates ) ), at = NULL, labels = paste( at ),
       type = "l", lwd = 2, lty = 1, col = par( "fg" ), log.ax = "y",
       las = 1, ann = FALSE, cex.ann = 0.8, xannx = 1/20,
       a.thin = seq( 1, length( age ), 2 ), ... )

Cplot( rates, age = as.numeric( rownames( rates ) ),
       per = as.numeric( colnames( rates ) ), grid = FALSE,
       c.grid = grid, ygrid = grid, col.grid = gray( 0.9 ),
       c.lim = NULL, ylim = range( rates[rates>0], na.rm=TRUE ),
       at = NULL, labels = paste( at ), c.lab = names( dimnames( rates ) )[2],
       ylab = deparse( substitute( rates ) ), type = "l", lwd = 2, lty = 1,
       col = par( "fg" ), log.ax = "y", las = 1, xannx = 1/20, ann = FALSE,
       cex.ann = 0.8, a.thin = seq( 1, length( age ), 2 ), ...  )
```

Arguments

`rates` A two-dimensional table (or array) with rates to be plotted. It is assumed that the first dimension is age and the second is period.

`which` A character vector with elements from `c("ap","ac","apc","pa","ca")`, indication which plots should be produced. One plot per element is produced. The first letter indicates the x-axis of the plot, the remaining which groups should be connected, i.e. `"pa"` will plot rates versus period and connect age-classes, and `"apc"` will plot rates versus age, and connect both periods and cohorts.

`age` Numerical vector giving the means of the age-classes. Defaults to the rownames of `rates` as numeric.

`per` Numerical vector giving the means of the periods. Defaults to the columnnames of `rates` as numeric.

`grid` Logical indicating whether a background grid should be drawn.

`a.grid` Logical indicating whether a background grid on the age-axis should be drawn. If numerical it indicates the age-coordinates of the grid.

`p.grid` do. for the period.

`c.grid` do. for the cohort.

`ygrid` do. for the rate-dimension.

`col.grid` The colour of the grid.

`a.lim` Range for the age-axis.

`p.lim` Range for the period-axis.

`c.lim` Range for the cohort-axis.

`ylim` Range for the y-axis (rates).

`at` Position of labels on the y-axis (rates).

`labels` Labels to put on the y-axis (rates).

`a.lab` Text on the age-axis. Defaults to ”Age”.

`p.lab` Text on the period-axis. Defaults to ”Date of diagnosis”.

`c.lab` Text on the cohort-axis. Defaults to ”Date of birth”.

`ylab` Text on the rate-axis. Defaults to the name of the rate-table.

`type` How should the curves be plotted. Defaults to `"l"`.

`lwd` Width of the lines. Defaults to 2.

`lty` Which type of lines should be used. Defaults to 1, a solid line.

`log.ax` Character with letters from `"apcyr"`, indicating which axes should be logarithmic. `"y"` and `"r"` both refer to the rate scale. Defaults to `"y"`.

`las` see `par`.

`ann` Should the curves be annotated?

`a.ann` Logical indicating whether age-curves should be annotated.

`p.ann` do. for period-curves.

`c.ann` do. for cohort-curves.

`xannx` The fraction that the x-axis is expanded when curves are annotated.

cex.ann Expansion factor for characters annotating curves.

a.thin Vector of integers indicating which of the age-classes should be labelled.

p.thin do. for the periods.

c.thin do. for the cohorts.

col Colours for the curves.

a.col Colours for the age-curves.

p.col do. for the period-curves.

c.col do. for the cohort-curves.

p.lines Should rates from the same period be connected?

c.lines Should rates from the same cohort be connected?

... Additional arguments pssed on to matlines when plotting the curves.

Details

Zero values of the rates are ignored. They are neiter in the plot nor in the calculation of the axis ranges.

Value

NULL. The function is used for its side-effect, the plot.

Author(s)

Bendix Carstensen, Steno Diabetes Center, http://www.pubhealth.ku.dk/~bxc/

See Also

apc.frame

Examples

```
data( blcaIT )
attach(blcaIT)

# Table of rates:
bl.rate <- tapply( D, list(age,period), sum ) /
           tapply( Y, list(age,period), sum )
bl.rate

# The four classical plots:
par( mfrow=c(2,2) )
rateplot( bl.rate*10^6 )

# The labels on the vertical axis could be nicer:
rateplot( bl.rate*10^6, at=10^(-1:3), labels=c(0.1,1,10,100,1000) )

# More bells an whistles
par( mfrow=c(1,3), mar=c(3,3,1,1), oma=c(0,3,0,0), mgp=c(3,1,0)/1.6 )
rateplot( bl.rate*10^6, ylab="", ann=TRUE, which=c("AC","PA","CA"),
                      at=10^(-1:3), labels=c(0.1,1,10,100,1000),
                      col=topo.colors(11), cex.ann=1.2 )
```

Relevel *Reorder and combine levels of a factor*

Description

The levels of a factor are re-ordered so that the levels specified by `ref` is first and the others are moved down. This is useful for `contr.treatment` contrasts which take the first level as the reference. Levels may also be combined.

Usage

```
Relevel(f, ref, first = TRUE, collapse="+" )
```

Arguments

`f`	An unordered factor
`ref`	The names or numbers of levels to be the first. If `ref` is a list, factor levels mentioned in each list element are combined. If the list is named the names are used as new factor levels.
`first`	Should the levels mentioned in ref come before those not?
`collapse`	String used when collapsing factor levels.

Value

An unordered factor.

Examples

```
ff <- factor( sample( letters[1:5], 100, replace=TRUE ) )
table( ff, Relevel( ff, list( AB=1:2, "Dee"=4, c(3,5) ) ) )
table( ff, rr=Relevel( ff, list( 5:4, Z=c("c","a") ), coll="-und-", first=FALSE ) )
```

ROC *Function to compute and draw ROC-curves.*

Description

Computes sensitivity, specificity and positive and negative predictive values for a test based on dichotomizing along the variable `test`, for prediction of `stat`. Alternatively a model formula may given, in which case the the linear predictor is the test variable and the response is taken as the true status variable. Plots curves of these and a ROC-curve.

Usage

```
ROC( test = NULL,
     stat = NULL,
     form = NULL,
     plot = c("sp", "ROC"),
       PS = is.null(test),
       PV = TRUE,
       MX = TRUE,
       MI = TRUE,
      AUC = TRUE,
     grid = seq(0,100,10),
 col.grid = gray( 0.9 ),
     cuts = NULL,
      lwd = 2,
     data = parent.frame(),
      ... )
```

Arguments

`test` Numerical variable used for prediction.

`stat` Logical variable of true status.

`form` Formula used in a logistic regression. If this is given, `test` and `stat` are ignored. If not given then both `test` and `stat` must be supplied.

`plot` Character variable. If ”sp”, the a plot of sensitivity, specificity and predictive values against test is produced, if ”ROC” a ROC-curve is plotted. Both may be given.

`PS` logical, if TRUE the x-axis in the plot ”ps”-plot is the the predicted probability for `stat`==TRUE, otherwise it is the scale of `test` if this is given otherwise the scale of the linear predictor from the logistic regression.

`PV` Should sensitivity, specificity and predictive values at the optimal cutpoint be given on the ROC plot?

`MX` Should the “optimal cutpoint” (i.e. where sens+spec is maximal) be indicated on the ROC curve?

`MI` Should model summary from the logistic regression model be printed in the plot?

`AUC` Should the area under the curve (AUC) be printed in the ROC plot?

`grid` Numeric or logical. If FALSE no background grid is drawn. Otherwise a grid is drawn on both axes at `grid` percent.

`col.grid` Colour of the grid lines drawn.

`cuts` Points on the test-scale to be annotated on the ROC-curve.

`lwd` Thickness of the curves

`data` Data frame in which to interpret the variables.

`...` Additional arguments for the plotting of the ROC-curve. Passed on to `plot`

Value

A list with two components:

`res` dataframe with variables sn, sp, pvp, pvn and fv. The latter is the unique values of test (for PS==FALSE) or linear predictor from the logistic regression

`lr` glm object with the logistic regression result used for construction of the ROC curve

0, 1 or 2 plots are produced according to the setting of `plot`.

Author(s)

Bendix Carstensen, Steno Diabetes Center \& University of Copenhagen,
http://www.biostat.ku.dk/~bxc

Examples

```
x <- rnorm( 100 )
z <- rnorm( 100 )
w <- rnorm( 100 )
tigol <- function( x ) 1 - ( 1 + exp( x ) )^(-1)
y <- rbinom( 100, 1, tigol( 0.3 + 3*x + 5*z + 7*w ) )
ROC( form = y ~ x + z, plot="ROC" )
```

`S.typh` *Salmonella Typhimurium outbreak 1996 in Denmark.*

Description

Matched case-control study of food poisoning.

Format

A data frame with 136 observations on the following 15 variables:

`id`:	Person identification
`set`:	Matched set indicator
`case`:	Case-control status (1:case, 0:control
`age`:	Age of individual
`sex`:	Sex of individual (1:male, 2:female)
`abroad`:	Within the last two weeks visited abroad (1:yes, 0:no)
`beef`:	Within the last two weeks eaten beef
`pork`:	Within the last two weeks eaten pork
`veal`:	Within the last two weeks eaten veal
`poultry`:	Within the last two weeks eaten poultry
`liverp`:	Within the last two weeks eaten liverpaste
`veg`:	Within the last two weeks eaten vegetables
`fruit`:	Within the last two weeks eaten fruit
`egg`:	Within the last two weeks eaten eggs
`plant7`:	Within the last two weeks eaten meat from plant no. 7

Details

In the fall of 1996 an unusually large number of Salmonella Typhimurium cases were recorded in Fyn county in Denmark. The Danish Zoonosis Centre set up a matched case-control study to find the sources. Cases and two age-, sex- and residency-matched controls were telephone interviewed about their food intake during the last two weeks.

The participants were asked at which retailer(s) they had purchased meat. Retailers were independently of this linked to meat processing plants, and thus participants were linked to meat processing plants. This way persons could be linked to (amongst other) plant no 7.

Source

Tine Hald.

References

Molbak K and Hald T: Salmonella Typhimurium outbreak in late summer 1996. A Case-control study. (In Danish: Salmonella typhimurium udbrud paa Fyn sensommeren 1996. En case-kontrol undersogelse.) Ugeskrift for Laeger., 159(36):5372-7, 1997.

Examples

```
data(S.typh)
```

splitLexis *Split follow-up time in a Lexis object*

Description

The splitLexis function divides each row of a Lexis object into disjoint follow-up intervals according to the supplied break points.

Usage

```
splitLexis(lex, breaks, time.scale, tol=.Machine$double.eps^0.5)
```

Arguments

lex	an object of class Lexis
breaks	a vector of break points
time.scale	the name or number of the time scale to be split
tol	numeric value >= 0. Intervals shorter than this value are dropped

Value

An object of class Lexis with multiple rows for each row of the argument lex. Each row of the new Lexis object contains the part of the follow-up interval that falls inside one of the time bands defined by the break points.

The variables representing the various time scales, are appropriately updated in the new Lexis object. The entry and exit status variables are also updated according to the rule that the entry status is retained until the end of follow-up. All other variables are considered to represent variables that are constant in time, and so are replicated across all rows having the same id value.

Note

The splitLexis() function divides follow-up time into intervals using breakpoints that are common to all rows of the Lexis object. To split a Lexis object by break points that are unique to each row, use the cut.Lexis function.

Author(s)

Martyn Plummer

See Also

timeBand, cutLexis, summary.Lexis

Examples

```
# A small bogus cohort
xcoh <- structure( list( id = c("A", "B", "C"),
                      birth = c("14/07/1952", "01/04/1954", "10/06/1987"),
                      entry = c("04/08/1965", "08/09/1972", "23/12/1991"),
                       exit = c("27/06/1997", "23/05/1995", "24/07/1998"),
                       fail = c(1, 0, 1) ),
                     .Names = c("id", "birth", "entry", "exit", "fail"),
                  row.names = c("1", "2", "3"),
                      class = "data.frame" )
```

```
# Convert the character dates into numerical variables (fractional years)
xcoh$bt <- cal.yr( xcoh$birth, format="%d/%m/%Y" )
xcoh$en <- cal.yr( xcoh$entry, format="%d/%m/%Y" )
xcoh$ex <- cal.yr( xcoh$exit , format="%d/%m/%Y" )

# See how it looks
xcoh

# Define as Lexis object with timescales calendar time and age
Lcoh <- Lexis( entry = list( per=en ),
                exit = list( per=ex, age=ex-bt ),
         exit.status = fail,
                data = xcoh )

# Default plot of follow-up
plot( Lcoh )
# With a grid and deaths as endpoints
plot( Lcoh, grid=0:10*10, col="black" )
points( Lcoh, pch=c(NA,16)[Lcoh$lex.Xst+1] )
# With a lot of bells and whistles:
plot( Lcoh, grid=0:20*5, col="black", xaxs="i", yaxs="i",
      xlim=c(1960,2010), ylim=c(0,50), lwd=3, las=1 )
points( Lcoh, pch=c(NA,16)[Lcoh$lex.Xst+1], col="red", cex=1.5 )

# Split time along two time-axes
( x2 <- splitLexis( Lcoh, breaks = seq(1900,2000,5), time.scale="per") )
( x2 <- splitLexis( x2, breaks = seq(0,80,5), time.scale="age" ) )
str( x2 )

# Tabulate the cases and the person-years
summary( x2 )
tapply( status(x2,"exit")==1, list( timeBand(x2,"age","left"),
                                     timeBand(x2,"per","left") ), sum )
tapply( dur(x2),  list( timeBand(x2,"age","left"),
                        timeBand(x2,"per","left") ), sum )
```

stack.Lexis *Functions to facilitate analysis of multistate models.*

Description

stack.Lexis produces a stacked object suited for analysis of several transitions simultaneously.

Usage

```
## S3 method for class 'Lexis'
stack(x, ...)
tmat( x, ... )
## S3 method for class 'Lexis'
tmat(x, ...)
```

Arguments

x	A Lexis object.
...	Not used.

Value

tmat.Lexis returns a square transition matrix, classified by the levels of lex.Cst and lex.Xst, it has a 1 for every transition occurring and NA in all oter entries.

stack.Lexis returns a dataframe to be used for analysis of multistate data when all transitions are modelled together, for example if some parameters are required to be the same for different transitions.

The dataframe has same variables as the original Lexis object, but with each record duplicated as many times as there are possible exits from the current state, lex.Cst. Two variables are added: lex.Fail, an indicator of wheter an event for the transition names in lex.Tr has occurred or not. lex.Tr is a factor with levels made up of combinations of the levels of lex.Cst and lex.Xst that do occur together in x, joined by a ”->”.

Author(s)

Bendix Carstensen, <bxc@steno.dk>, www.biostat.ku.dk/~bxc

See Also

splitLexis cutLexis Lexis

Examples

```
data(DMlate)
str(DMlate)
dml <- Lexis( entry=list(Per=dodm, Age=dodm-dobth, DMdur=0 ),
               exit=list(Per=dox),
        exit.status=factor(!is.na(dodth),labels=c("DM","Dead")),
               data=DMlate )
dmi <- cutLexis( dml, cut=dml$doins, new.state="Ins", pre="DM" )
summary( dmi )
ls.dmi <- stack( dmi )
str( ls.dmi )
# Check that all the transitions and person-years got across.
with( ls.dmi, rbind( table(lex.Fail,lex.Tr),
                     tapply(lex.dur,lex.Tr,sum) ) )
```

---

start.Lexis *Time series methods for Lexis objects*

Description

Extract the entry time, exit time, status, or duration of follow-up from a Lexis object.

Usage

```
entry(x, time.scale = NULL)
exit(x, time.scale = NULL)
status(x, at="exit")
dur(x)
```

Arguments

x	an object of class Lexis.
time.scale	a string or integer indicating the time scale. If omitted, all times scales are used.
at	string indicating the time point(s) at which status is to be measured.

Value

The `entry` and `exit` functions return a vector of entry times and exit times, respectively, on the requested time scale. If multiple time scales are requested, then a matrix is returned.

The `status` function returns a vector giving the status at entry or exit and `dur` returns a vector with the lengths of the follow-up intervals.

Author(s)

Martyn Plummer

See Also

`Lexis`

`stat.table` *Tables of summary statistics*

Description

`stat.table` creates tabular summaries of the data, using a limited set of functions. A list of index variables is used to cross-classify summary statistics. It does NOT work inside `with()`!

Usage

```
stat.table(index, contents = count(), data, margins = FALSE)
## S3 method for class 'stat.table'
print(x, width=7, digits,...)
```

Arguments

`index` A factor, or list of factors, used for cross-classification. If the list is named, then the names will be used when printing the table. This feature can be used to give informative labels to the variables.

`contents` A function call, or list of function calls. Only a limited set of functions may be called (See Details below). If the list is named, then the names will be used when printing the table.

`data` an optional data frame containing the variables to be tabulated. If this is omitted, the variables will be searched for in the calling environment.

`margins` a logical scalar or vector indicating which marginal tables are to be calculated. If a vector, it should be the same length as the `index` argument: values corresponding to `TRUE` will be retained in marginal tables.

`x` an object of class `stat.table`.

`width` a scalar giving the minimum column width when printing.

`digits` a scalar, or named vector, giving the number of digits to print after the decimal point. If a named vector is used, the names should correspond to one of the permitted functions (See Details below) and all results obtained with that function will be printed with the same precision.

`...` further arguments passed to other print methods.

Details

This function is similar to tapply, with some enhancements: multiple summaries of multiple variables may be mixed in the same table; marginal tables may be calculated; columns and rows may be given informative labels; pretty printing may be controlled by the associated print method.

This function is not a replacement for tapply as it also has some limitations. The only functions that may be used in the contents argument are: count, mean, weighted.mean, sum, quantile, median, IQR, max, min, ratio, and percent.

The count() function, which is the default, simply creates a contingency table of counts. The other functions are applied to each cell created by combinations of the index variables.

Value

An object of class stat.table, which is a multi-dimensional array. A print method is available to create formatted one-way and two-way tables.

Note

The permitted functions in the contents list are defined inside stat.table. They have the same interface as the functions callable from the command line, except for two differences. If there is an argument na.rm then its default value is always TRUE. A second difference is that the quantile function can only produce a single quantile in each call.

Author(s)

Martyn Plummer

See Also

table, tapply, mean, weighted.mean, sum, quantile, median, IQR, max, min, ratio, percent, count

Examples

```
data(warpbreaks)
# A one-way table
stat.table(tension,list(count(),mean(breaks)),data=warpbreaks)
# The same table with informative labels
stat.table(index=list("Tension level"=tension),list(N=count(),
           "mean number of breaks"=mean(breaks)),data=warpbreaks)

# A two-way table
stat.table(index=list(tension,wool),mean(breaks),data=warpbreaks)
# The same table with margins over tension, but not wool
stat.table(index=list(tension,wool),mean(breaks),data=warpbreaks,
           margins=c(TRUE, FALSE))

# A table of column percentages
stat.table(list(tension,wool), percent(tension), data=warpbreaks)
# Cell percentages, with margins
stat.table(list(tension,wool),percent(tension,wool), margin=TRUE,
           data=warpbreaks)

# A table with multiple statistics
# Note how each statistic has its own default precision
a <- stat.table(index=list(wool,tension),
                contents=list(count(),mean(breaks),percent (wool)),
                data=warpbreaks)
```

```
print(a)
# Print the percentages rounded to the nearest integer
print(a, digits=c(percent=0))
```

`stattable.funs` *Special functions for use in stat.table*

Description

These functions may be used as `contents` arguments to the function `stat.table`. They are defined internally in `stat.table` and have no independent existence.

Usage

```
count(id)
ratio(d,y,scale=1, na.rm=TRUE)
percent(...)
```

Arguments

`id`	numeric vector in which identical values identify the same individual.
`d, y`	numeric vectors of equal length (`d` for Deaths, `y` for person-Years)
`scale`	a scalar giving a value by which the ratio should be multiplied
`na.rm`	a logical value indicating whether `NA` values should be stripped before computation proceeds.
`...`	a list of variables taken from the `index` argument to `stat.table`

Value

When used as a `contents` argument to `stat.table`, these functions create the following tables:

`count`	If given without argument (`count()`) it returns a contingency table of counts. If given an `id` argument it returns a table of the number of different values of `id` in each cell, i.e. how many persons contribute in each cell.
`ratio`	returns a table of values `scale * sum(d)/sum(y)`
`percent`	returns a table of percentages of the classifying variables. Variables that are in the `index` argument to `stat.table` but not in the call to `percent` are used to define strata, within which the percentages add up to 100.

Author(s)

Martyn Plummer

See Also

`stat.table`

subset.Lexis *Subsetting Lexis objects*

Description

Return subsets of Lexis objects which meet conditions

Usage

```
## S3 method for class 'Lexis'
subset(x, ...)
```

Arguments

x	an object of class Lexis
...	additional arguments to be passed to subset.data.frame

Details

The subset method for Lexis objects works exactly as the method for data frames.

Value

A Lexis object with selected rows and columns.

Author(s)

Martyn Plummer

See Also

Lexis, merge.Lexis

summary.Lexis *Summarize transitions and risk time from a Lexis object*

Description

A two-way table of records and transitions classified by states (lex.Cst and lex.Xst), as well the risk time in each state.

Usage

```
  ## S3 method for class 'Lexis'
summary( object, simplify=TRUE, scale=1, ... )
  ## S3 method for class 'summary.Lexis'
print( x, ..., digits=2 )
```

Arguments

`object` A Lexis object.

`x` A `summary.Lexis` object.

`simplify` Should rows with 0 follow-up time be dropped?

`scale` Scaling factor for the rates. The calculated rates are multiplied by this number.

`digits` How many digits should be used for printing?

`...` Other parameters - ignored

Value

An object of class `summary.Lexis`, a list with two components, `Transitions` and `Rates`, each one a matrix with rows classified by states where persons spend time, and columns classified by stated to which persons transit. The `Transitions` contains number of transitions and has two extra columns of total number events and total risk time attached. The `Rates` contians the transitions rates.

Author(s)

Bendix Carstensen, `<bxc@steno.dk>`

Examples

```
data( nickel )
# Lung cancer deaths and other deaths are coded 1 and 2
nic <- Lexis( data=nickel,
             entry=list(age=agein),
              exit=list(age=ageout,cal=ageout+dob,tfh=ageout-age1st),
       exit.status=factor( (icd > 0) + (icd %in% c(162,163)),
                          labels=c("Alive","Other","Lung") ) )
str( nic )
head( nic )
summary( nic )
```

`tbox` *Draw boxes and arrows for illustration of multistate models.*

Description

Boxes can be drawn with text (`tbox`) or a cross (`dbox`), and arrows pointing between the boxes (`boxarr`) can be drawn automatically not overlapping the boxes. `Lexis` objects can be used to generate displays with person-years and events.

Usage

```
tbox( txt, x, y, wd, ht,
      font=2, lwd=2,
      col.txt="black",
      col.border="black",
      col.bg="transparent" )
dbox( x, y, wd, ht=wd,
      font=2, lwd=2, cwd=5,
      col.cross="black",
      col.border="black",
      col.bg="transparent"  )
boxarr( b1, b2, offset=FALSE, pos=0.45, ... )
```

```
   boxes( obj, ... )
      ## S3 method for class 'Lexis'
boxes( obj, file,
             detailed = FALSE,
               boxpos = FALSE,
                wmult = 1.5,
                hmult = 1.5*wmult,
                  cex = 1.5,
               show   = inherits( obj, "Lexis" ),
               show.Y = show,
              scale.Y = 1,
             digits.Y = 1,
               show.D = show,
              scale.D = FALSE,
             digits.D = as.numeric(as.logical(scale.D)),
                eq.wd = TRUE,
                eq.ht = TRUE,
                   wd,
                   ht,
               subset = NULL,
              exclude = NULL,
                 font = 2,
                  lwd = 2,
              col.txt = "black",
           col.border = col.txt,
               col.bg = "transparent",
              col.arr = "black",
              lwd.arr = 2,
             font.arr = 2,
              txt.arr = NULL,
          col.txt.arr = col.arr,
           offset.arr = 2, ... )
   fillarr( x1, y1, x2, y2, gap=2, fr=0.8,
            angle=17, lwd=2, length=par("pin")[1]/30, ... )
```

Arguments

`txt` Text to be placed inside the box.

`x` x-coordinate of center of box.

`y` y-coordinate of center of box.

`wd` width of boxes in percentage of the plot width.

`ht` height of boxes in percentage of the plot height.

`font` Font for the text. Defaults to 2 (=bold).

`lwd` Line width of the boxborders.

`col.txt` Color for the text in boxes.

`col.border` Color of the box border.

`col.bg` Background color for the interior of the box.

`...` Arguments to be passed on to the call of other functions.

`cwd` Width of the lines in the cross.

`col.cross` Color of the cross.

`b1` Coordinates of the ”from” box. A vector with 4 components, `x`, `y`, `w`, `h`.

`b2` Coordinates of the ”to” box; like `b1`.

offset Logical. Should the arrow be offset a bit to the left.

pos Numerical between 0 and 1, determines the position of the point on the arrow which is returned.

obj A Lexis object, or a transition matrix; that is a matrix

file Name of the file with the code reproducing the plot.

detailed Should the output of R-code be detailed, showing all parameters?

boxpos If TRUE the boxes are positioned equidistantly on a circle, if FALSE (the default) you are queried to click on the screen for the positions. This argument can also be a named list with elements x and y, both numerical vectors, giving the centers of the boxes.

wmult Multiplier for the width of the box relative to the width of the text in the box.

hmult Multiplier for the height of the box relative to the height of the text in the box.

cex Character expansion for text in the box.

show Should person-years and transitions be put in the plot. Ignored if obj is not a Lexis object.

show.Y Should person-years be put in the boxes. Ignored if obj is not a Lexis object.

scale.Y What scale should be used for annotation of person-years.

digits.Y How many digits after the decimal point should be used for the person-years.

show.D Should transitions be put alongside the arrows. Ignored if obj is not a Lexis object.

scale.D If this a scalar, rates instead of no. transitions are printed at the arrows, scaled by scale.D.

digits.D How many digits after the decimal point should be used for the rates.

eq.wd Should boxes all have the same width?

eq.ht Should boxes all have the same height?

subset Draw only boxes and arrows for a subset of the states. Can be given either as a numerical vector or character vector state names.

exclude Exclude states from the plot. The complementary of subset. Ignored if subset is given.

col.arr Color of the arrows between boxes. A vector of character strings, the arrows are referred to as the row-wise sequence of non-NA elements of the transition matrix. Thus the first ones refer to the transitions out of state 1, in order of states.

lwd.arr Line withs of the arrows.

font.arr Font of the text annotation the arrows.

txt.arr Text put on the arrows.

col.txt.arr Colors for text on the arrows.

offset.arr The amount offset between arrows that go between the same pair of boxes (two-way transitions).

x1 x-coordinate of the starting point.

y1 y-coordinate of the starting point.

x2 x-coordinate of the end point.

y2 y-coordinate of the end point.

gap Length of the gap between the box and the ends of the arrows.

fr Length of the arrow as the fraction of the distance between the boxes. Ignored unless given explicitly, in which case any value given for gap is ignored.

angle What angle should the arrow-head have?

length Length of the arrow head in inches. Defaults to 1/30 of the physical width of the plot.

Details

These functions are designed to facilitate the drawing of multistate models, mainly by automatic calculation of the arrows between boxes.

`tbox` draws a box with centered text, and returns a vector of location, height and width of the box. This is used when drawing arrows between boxes. `dbox` draws a box with a cross, symbolizing a death state. `boxarr` draws an arrow between two boxes, making sure it does not intersect the boxes. Only straight lines are drawn.

`boxes.Lexis` takes as input a Lexis object sets up an empty plot area (with axes 0 to 100 in both directions) and if `boxpos=FALSE` (the default) prompts you to click on the locations for the state boxes, and then draws arrows implied by the actual transitions in the `Lexis` object.

A transition matrix can also be supplied, in which case the row/column names are used as state names.

Optionally returns the R-code reproducing the plot in a file, which can be useful if you want to produce exactly the same plot with differing arrow colors etc.

`boxarr` draws an arrow between two boxes, on the line connecting the two box centers. The `offset` argument is used to offset the arrow a bit to the left (as seen in the direction of the arrow) on order to accommodate arrows both ways between boxes. `boxarr` returns a named list with elements `x`, `y` and `d`, where the two former give the location of a point on the arrow used for printing (see argument `pos`) and the latter is a unit vector in the direction of the arrow, which is used by `boxes.Lexis` to position the annotation of arrows with the number of transitions. `fill.arr` is just a utility drawing nicer arrows than the default `arrows` command, basically by using filled arrow-heads; called by `boxarr`.

Value

The functions `tbox` and `dbox` return the location and dimension of the boxes, `c(x,y,w,h)`, which are designed to be used as input to the `boxarr` function.

The `boxarr` function returns the coordinates (as a named list with names `x` and `y`) of a point on the arrow, designated to be used for annotation of the arrow.

Author(s)

Bendix Carstensen

Examples

```
par( mar=c(0,0,0,0), cex=1.5 )
plot( NA,
      bty="n",
      xlim=0:1*100, ylim=0:1*100, xaxt="n", yaxt="n", xlab="", ylab="" )
bw  <- tbox( "Well"    , 10, 60, 22, 10, col.txt="blue" )
bo  <- tbox( "other Ca", 45, 80, 22, 10, col.txt="gray" )
bc  <- tbox( "Ca"      , 45, 60, 22, 10, col.txt="red" )
bd  <- tbox( "DM"      , 45, 40, 22, 10, col.txt="blue" )
bcd <- tbox( "Ca + DM" , 80, 60, 22, 10, col.txt="gray" )
bdc <- tbox( "DM + Ca" , 80, 40, 22, 10, col.txt="red" )
      boxarr( bw, bo , col=gray(0.7), lwd=3 )
# Note the argument adj= can takes values outside (0,1)
text( boxarr( bw, bc , col="blue", lwd=3 ),
      expression( lambda[Well] ), col="blue", adj=c(1,-0.2), cex=0.8 )
      boxarr( bw, bd , col=gray(0.7) , lwd=3 )
      boxarr( bc, bcd, col=gray(0.7) , lwd=3 )
text( boxarr( bd, bdc, col="blue", lwd=3 ),
      expression( lambda[DM] ), col="blue", adj=c(1.1,-0.2), cex=0.8 )

# Set up a transition matrix allowing recovery
tm <- rbind( c(NA,1,1), c(1,NA,1), c(NA,NA,NA) )
```

```
rownames(tm) <- colnames(tm) <- c("Cancer","Recurrence","Dead")
boxes.Lexis( tm, file="", boxpos=TRUE )
boxes.Lexis( tm, file="", detailed=TRUE, boxpos=TRUE )
# Illustrate texting of arrows
boxes.Lexis( tm, boxpos=TRUE, txt.arr=c("en","to","tre","fire") )
boxes.Lexis( tm, boxpos=TRUE, txt.arr=c(expression(lambda[C]),
                                        expression(mu[C]),
                                        "recovery",
                                        expression(mu[R]) ) )

# Set up a Lexis object
data(DMlate)
str(DMlate)
dml <- Lexis( entry=list(Per=dodm, Age=dodm-dobth, DMdur=0 ),
               exit=list(Per=dox),
        exit.status=factor(!is.na(dodth),labels=c("DM","Dead")),
               data=DMlate )
# Split follow-up at Insulin
dmi <- cutLexis( dml, cut=dml$doins, new.state="Ins", pre="DM" )
summary( dmi )
boxes( dmi, boxpos=TRUE, file="" )
# Set up a bogus recovery date
dmi$dorec <- dmi$doins + runif(nrow(dmi),0.5,10)
dmi$dorec[dmi$dorec>dmi$dox] <- NA
dmR <- cutLexis( dmi, cut=dmi$dorec, new.state="DM", pre="Ins" )
summary( dmR )
boxes( dmR, boxpos=TRUE, file="" )
```

thoro *Thorotrast Study*

Description

The `thoro` data frame has 2470 rows and 14 columns. Each row represents one patient that have had cerebral angiography (X-ray of the brain) with an injected contrast medium, either Thorotrast or another one (the controls).

Format

This data frame contains the following columns:

`id`: Identification of person.

`sex`: Sex, 1: male / 2: female.

`birthdat`: Date of birth, `Date` variable.

`contrast`: Group, 1: Thorotrast / 2: Control.

`injecdat`: Date of contrast injection, `Date` variable.

`volume`: Injected volume of Thorotrast in ml. Control patients have a 0 in this variable.

`exitdat`: Date of exit from the study, `Date` variable.

`exitstat`: Status at exit, 1: dead / 2: alive, censored at closing of study, 20 February 1992 / 3: censored alive at so

`cause`: Cause of death. See causes in the helpfile for gmortDK

`liverdat`: Date of liver cancer diagnosis, `Date` variable.

`liver`: Indicator of liver cancer diagnosis. Not all livercancers are histologically verified, hence `liver >= hepcc`

`hepcc`: Hepatocellular carcinoma at `liverdat`.

`chola`: Cholangiocellular carcinoma at `liverdat`.

`hmang`: Haemangisarcoma carcinoma at `liverdat`.

Source

M Andersson, M Vyberg, J Visfeldt, B Carstensen & HH Storm: Primary liver tumours among Danish patients exposed to Thorotrast. Radiation Research, 137, pp. 262–273, 1994.

M Andersson, B Carstensen HH Storm: Mortality and cancer incidence after cerebral angiography. Radiation Research, 142, pp. 305–320, 1995.

See Also

`mortDK`, `gmortDK`

Examples

```
data(thoro)
str(thoro)
```

`timeBand` *Extract time band data from a split Lexis object*

Description

The break points of a `Lexis` object (created by a call to `splitLexis`) divide the follow-up intervals into time bands along a given time scale. The `breaks` function returns the break points, for a given time scale, and the `timeBand` classifies each row (=follow-up interval) into one of the time bands.

Usage

```
timeBand(lex, time.scale, type="integer")
breaks(lex, time.scale)
```

Arguments

`lex`	an object of class `Lexis`
`time.scale`	a character or integer vector of length 1 identifying the time scale of interest
`type`	a string that determines how the time bands are labelled. See Details below

Details

Time bands may be labelled in various ways according to the `type` argument. The permitted values of the `type` argument, and the corresponding return values are:

"integer" a numeric vector with integer codes starting from 0.

"factor" a factor (unordered) with labels "(left,right]"

"left" the left-hand limit of the time band

"middle" the midpoint of the time band

"right" the right-hand limit of the time band

Value

The `breaks` function returns a vector of break points for the `Lexis` object, or NULL if no break points have been defined by a call to `splitLexis`. The `timeBand` function returns a numeric vector or factor, depending on the value of the `type` argument.

Note

A newly created Lexis object has no break points defined. In this case, breaks will return NULL, and timeBand will a vector of zeros.

Author(s)

Martyn Plummer

See Also

Lexis

Examples

```
data(diet)
diet <- cal.yr(diet)
diet.lex <- Lexis(entry=list(period=doe),
                   exit=list(period=dox, age=dox-dob),
            exit.status=chd,
                   data=diet)
diet.split <- splitLexis(diet.lex, breaks=seq(40,70,5), "age" )
age.left <- timeBand(diet.split, "age", "left")
table(age.left)
age.fact <- timeBand(diet.split, "age", "factor")
table(age.fact)
age.mid <- timeBand(diet.split, "age", "mid")
table(age.mid)
```

---

timeScales *The time scales of a Lexis object*

Description

Function to get the names of the time scales of a Lexis object.

Usage

```
timeScales(x)
```

Arguments

x	an object of class Lexis

Value

A character vector containing the names of the variables in x that represent the time scales

Author(s)

Martyn Plummer

See Also

Lexis, splitLexis

transform.Lexis *Transform a Lexis objects*

Description

Transform a Lexis object

Usage

```
## S3 method for class 'Lexis'
transform(`_data`, ...)
## S3 method for class 'Lexis'
factorize(obj, ...)
```

Arguments

`_data`	an object of class `Lexis`.
`obj`	an object of class `Lexis`.
`...`	additional arguments to be passed to `transform.data.frame`.

Details

The transform method for `Lexis` objects works exactly as the method for data frames.

`factorize` simply transforms the variables `lex.Cst` and `lex.Xst` to factors with the same set of levels.

Value

A transformed `Lexis` object.

Author(s)

Martyn Plummer, Bendix Carstensen

See Also

`Lexis`, `merge.Lexis`, `subset.Lexis`

twoby2 *Analysis of a two by two table*

Description

Computes the usual measures of association in a 2 by 2 table with confidence intervals. Also produces asymtotic and exact tests. Assumes that comparison of probability of the first column level between levels of the row variable is of interest. Output requires that the input matrix has meaningful row and column labels.

Usage

```
twoby2(exposure, outcome,
       alpha = 0.05, print = TRUE, dec = 4,
       conf.level = 1-alpha, F.lim = 10000)
```

Arguments

`exposure`	If a table the analysis is based on the first two rows and first two columns of this. If a variable, this variable is tabulated against
`outcome`	as the second variable
`alpha`	Significance level
`print`	Should the results be printed?
`dec`	Number of decimals in the printout.
`conf.level`	1-`alpha`
`F.lim`	If the table total exceeds `F.lim`, Fisher's exact test is not computed

Value

A list with elements:

`table`	The analysed 2 x 2 table augmented with probabilities and confidence intervals. The confidence intervals for the probabilities are computed using the normal approximation to the log-odds. Confidence intervals for the difference of proportions are computed using method 10 from Newcombe, Stat.Med. 1998, 17, pp.873 ff.
`measures`	A table of Odds-ratios and relative risk with confidence intervals.
`p.value`	Exact p-value for the null hypothesis of OR=1

Author(s)

Mark Myatt. Modified by Bendix Carstensen.

Examples

```
Treat <- sample(c("A","B"), 50, rep=TRUE )
Resp <- c("Yes","No")[1+rbinom(50,1,0.3+0.2*(Treat=="A"))]
twoby2( Treat, Resp )
twoby2( table( Treat, Resp )[,2:1] ) # Comparison the other way round
```

Index

∗Topic **aplot**
plot.Lexis, 102
∗Topic **array**
detrend, 69
merge.Lexis, 94
pctab, 101
projection.ip, 107
∗Topic **attributes**
lls, 91
∗Topic **attribute**
timeBand, 126
timeScales, 127
∗Topic **category**
stat.table, 117
stattable.funs, 119
∗Topic **chron**
cal.yr, 56
∗Topic **datagen**
ccwc, 58
∗Topic **datasets**
bdendo, 54
bdendo11, 54
births, 55
blcaIT, 55
brv, 56
diet, 69
DMconv, 70
DMlate, 71
ewrates, 75
gmortDK, 81
hivDK, 82
lep, 84
lungDK, 92
mortDK, 96
nickel, 100
occup, 100
S.typh, 113
thoro, 125
∗Topic **design**
contr.cum, 65
∗Topic **distribution**
ci.pd, 63
∗Topic **dplot**
Lexis.diagram, 87
Lexis.lines, 89
Life.lines, 90
∗Topic **hplot**
apc.frame, 49
apc.lines, 51
apc.plot, 53
Lexis.diagram, 87
Lexis.lines, 89
plot.Lexis, 102
plotEst, 104
rateplot, 107
tbox, 121
∗Topic **htest**
ci.pd, 63
mh, 95
ROC, 111
twoby2, 128
∗Topic **iplot**
tbox, 121
∗Topic **iteration**
stat.table, 117
stattable.funs, 119
∗Topic **manip**
cal.yr, 56
Life.lines, 90
merge.Lexis, 94
ncut, 98
nice, 99
pctab, 101
Relevel, 111
ROC, 111
splitLexis, 114
subset.Lexis, 120
transform.Lexis, 128
∗Topic **methods**
pctab, 101
∗Topic **models**
apc.fit, 46
ci.cum, 59
ci.lin, 60
clogistic, 64
contr.cum, 65
effx, 72
effx.match, 73
expand.data, 75
fit.add, 76
fit.baseline, 77
fit.mult, 78
Icens, 83
plotEst, 104
plotevent, 106
∗Topic **regression**

apc.fit, 46
ci.cum, 59
ci.lin, 60
effx, 72
effx.match, 73
expand.data, 75
fit.add, 76
fit.baseline, 77
fit.mult, 78
float, 79
ftrend, 80
Icens, 83
plotevent, 106

∗Topic **survival**
cutLexis, 66
expand.data, 75
fit.add, 76
fit.baseline, 77
fit.mult, 78
Icens, 83
Lexis, 85
msdata.Lexis, 97
plotevent, 106
stack.Lexis, 115
start.Lexis, 116
summary.Lexis, 120
tbox, 121

∗Topic **ts**
merge.data.frame, 93
start.Lexis, 116

∗Topic **univar**
twoby2, 128

addmargins, *102*
apc.fit, 46, *51–53*
apc.frame, *49*, 49, *51–53*, *110*
apc.lines, *48*, *49*, *51*, 51, *53*
apc.plot, *48*, *49*, *51*, *53*, 53
Aplot (*rateplot*), 107
arrows, *124*
as.Date.cal.yr (*cal.yr*), 56

bdendo, *54*, 54
bdendo11, 54
binom.test, *64*
births, 55
blcaIT, 55
boxarr (*tbox*), 121
boxes (*tbox*), 121
breaks (*timeBand*), 126
brv, 56

cal.yr, 56, *86*
ccwc, 58
ci.cum, 59, *62*
ci.lin, *60*, 60, *104*
ci.mat (*ci.lin*), 60
ci.pd, 63
clear (*lls*), 91
clogistic, 64
contr.2nd (*contr.cum*), 65
contr.cum, 65
contr.diff (*contr.cum*), 65
contr.orth (*contr.cum*), 65
contr.treatment, *66*
count, *118*
count (*stattable.funs*), 119
countLexis (*cutLexis*), 66
Cplot (*rateplot*), 107
cut, *99*
cutLexis, 66, *86*, *114*, *116*

Date, *56*, *57*, *70*, *86*
date, *57*
DateTimeClasses, *57*
dbox (*tbox*), 121
detrend, 69, *107*
diet, 69
DMconv, 70
DMlate, 71
DMrand (*DMlate*), 71
dur, *86*
dur (*start.Lexis*), 116

effx, 72
effx.match, 73
entry, *86*
entry (*start.Lexis*), 116
ewrates, 75, *100*
exit, *86*
exit (*start.Lexis*), 116
expand.data, 75, *76–78*

factorize (*transform.Lexis*), 128
fillarr (*tbox*), 121
findInterval, *99*
fit.add, *76*, 76, *77*, *78*, *83*, *84*
fit.baseline, 77
fit.mult, *76*, *77*, 78, *83*, *84*
float, 79, *81*
ftrend, *80*, 80

glm, *65*, *77*
gmortDK, 81, *97*, *125*, *126*

hivDK (*hivDK*), 82
hivDK, 82

Icens, *76–78*, 83, *106*
IQR, *118*

lep, 84
Lexis, *58*, *68*, 85, *88*, *94*, 95, *97*, *103*, *115–117*, *120*, *121*, *123*, *127*, *128*
Lexis.diagram, 87, *90*, *91*

Lexis.lines, *88*, 89, *91*
Life.lines, *88*, *90*, 90
lines.Lexis *(plot.Lexis)*, 102
linesEst *(plotEst)*, 104
lls, 91
ls, *92*
lungDK, 92

mathematical functions, 10
max, *118*
mean, *118*
median, *118*
merge.data.frame, 93, *94*
merge.Lexis, *86*, 94, *120*, *128*
mh, 95
min, *118*
mortDK, *82*, 96, *126*
msdata *(msdata.Lexis)*, 97
msdata.Lexis, 97

ncut, 98
nice, 99
nickel, *75*, 100

occup, 100, *103*

pc.lines *(apc.lines)*, 51
pc.matlines *(apc.lines)*, 51
pc.matpoints *(apc.lines)*, 51
pc.points *(apc.lines)*, 51
pctab, 101
percent, *118*
percent *(stattable.funs)*, 119
plot.Lexis, *86*, *88*, 102
plotEst, 104
plotevent, 106
points.Lexis *(plot.Lexis)*, 102
pointsEst *(plotEst)*, 104
POSIXct, *57*
POSIXlt, *57*
Pplot *(rateplot)*, 107
pretty, *99*
print.floated *(float)*, 79
print.Icens *(Icens)*, 83
print.stat.table *(stat.table)*, 117
print.summary.Lexis *(summary.Lexis)*, 120
projection.ip, *69*, 107
PY.ann *(plot.Lexis)*, 102

quantile, *118*

rateplot, 107
ratio, *118*
ratio *(stattable.funs)*, 119
Relevel, 111
ROC, 111

S.typh, 113
splitLexis, *68*, *86*, *103*, 114, *116*, *127*
stack.Lexis, *97*, *98*, 115
start.Lexis, 116
stat.table, 117, *119*
stattable.funs, 119
status *(start.Lexis)*, 116
subset.Lexis, *86*, *94*, 120, *128*
sum, *118*
summary.Lexis, *68*, *86*, *114*, 120

table, *118*
tapply, *118*
tbox, 121
thoro, *82*, *97*, 125
timeBand, *86*, *114*, 126
timeScales, *86*, 127
tmat *(stack.Lexis)*, 115
transform.Lexis, *86*, 128
twoby2, *64*, 128

Wald *(ci.lin)*, 60
weighted.mean, *118*