Package 'Epi'

August 4, 2026

Version 2.66

Date 2026-08-01

Title Statistical Analysis in Epidemiology

Depends R (>= 3.5.0), utils

Imports cmprsk, etm, splines, MASS, survival, plyr, dplyr, Matrix, numDeriv, data.table, zoo, mgcv, magrittr

Suggests mstate, nlme, lme4, demography, popEpi, tidyr

Description Functions for demographic and epidemiological analysis in the Lexis diagram, i.e. register and cohort follow-up data. In particular representation, manipulation, rate estimation and simulation for multistate data - the Lexis suite of functions, which includes interfaces to 'mstate', 'etm' and 'cmprsk' packages. Contains functions for Age-Period-Cohort and Lee-Carter modeling and a function for interval censored data. Has functions for extracting and manipulating parameter estimates and predicted values (ci.lin and its cousins), as well as a number of epidemiological data sets.

License GPL-2

URL http://bendixcarstensen.com/Epi/

NeedsCompilation yes

Author Bendix Carstensen [aut, cre],
Martyn Plummer [aut],
Esa Laara [ctb],
Michael Hills [ctb]

Maintainer Bendix Carstensen <b@bxc.dk>

Repository CRAN

Date/Publication 2026-08-04 03:10:02 UTC

Contents

AaJ 4
addCov 5

addDrug 8
adtte2Lexis {Epi} 11
apc.fit 13
apc.frame 18
apc.LCa 20
apc.lines 22
B.dk 24
bdendo 25
births 26
blcaIT 27
bootLexis 27
boxes.MS 29
BrCa 35
brv 36
cal.yr 37
cbind.Lexis 39
ccwc 40
ci.Crisk 42
ci.cum 44
ci.eta 47
ci.lin 48
ci.pd 54
clogistic 55
contr.cum 57
crr.Lexis 58
cutLexis 60
detrend 63
diet 64
DMconv 65
DMepi 66
DMlate 67
effx 69
effx.match 71
entry.Lexis 72
Epi 73
erl 74
ewrates 77
expand.data 78
fit.add 79
fit.baseline 80
fit.mult 81
float 82
foreign.Lexis 84
ftrend 85
gen.exp 87
gmortDK 91
harm 92
hivDK 93

Icens . . . 94
in.span . . . 96
LCa.fit . . . 98
legendbox . . . 103
lep . . . 104
Lexis . . . 105
Lexis.diagram . . . 108
Lexis.lines . . . 111
Lexis2msm . . . 112
LexisScarf . . . 113
lgrep . . . 114
Life.lines . . . 115
lls . . . 117
lungDK . . . 118
M.dk . . . 119
mat2pol . . . 120
matshade . . . 121
mcutLexis . . . 123
merge.Lexis . . . 125
mh . . . 126
mod.Lexis . . . 128
mortDK . . . 132
N.dk . . . 133
N2Y . . . 134
NArray . . . 136
ncut . . . 137
nice . . . 138
nickel . . . 139
Ns . . . 139
occup . . . 142
overlap.Lexis . . . 143
paths.Lexis . . . 144
pc.lines . . . 145
pctab . . . 146
plot.apc . . . 147
plot.Lexis . . . 148
plotCIF . . . 150
plotEst . . . 153
plotevent . . . 155
poisreg . . . 156
pr . . . 158
projection.ip . . . 158
rateplot . . . 159
rcutLexis . . . 163
Relevel . . . 164
rm.tr . . . 166
ROC . . . 167
S.typh . . . 169

simLexis 170
splitLexis 175
stack.Lexis 177
stat.table 178
stattable.funs 180
steno2 181
subset.Lexis 183
summary.Lexis 184
Termplot 186
testisDK 187
thoro 188
timeBand 189
timeScales 190
transform.Lexis 191
twoby2 193
unLexis 194
xcutLexis 195
Y.dk 197

Index **199**

`AaJ` *The Aalen-Johansen estimator of state probabilities from a multistate* `Lexis` *object.*

Description

The Aalen-Johansen estimator is computed on the basis of a `Lexis` multistate object along a given time scale. The function is a wrapper for the `survfit`.

Usage

```
## S3 method for class 'Lexis'
AaJ(Lx,
                formula = ~ 1,
              timeScale = 1, ...)
```

Arguments

`Lx`	A `Lexis` object. The starting state must be the first among `levels(Lx)`.
`formula`	A one-sided formula passed on to `survfit`.
`timeScale`	Character or integer, selecting one of the timescales of the `Lexis` object.
`...`	Arguments passed on. Ignored.

Value

An object of class `survfitms` — see `survfit`.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

See Also

survfit ci.Crisk

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 )

# Cut the follow-up at insulin start
dmi <- cutLexis(dml,
                cut = dml$doins,
          new.state = "Ins",
        split.state = TRUE)
summary( dmi )
ms <- AaJ.Lexis(dmi, timeScale = "DMdur")
class(ms)
ms$states
head(ms$pstate)
```

---

addCov *Add covariates (typically clinical measurements) taken at known times to a Lexis object.*

Description

When follow-up in a multistate model is represented in a Lexis object we may want to add information on covariates, for example clinical measurements, obtained at different times. This function cuts the follow-up time (see cutLexis) at the times of measurement and carries the measurements forward in time to the next measurement occasion.

Usage

```
## S3 method for class 'Lexis'
addCov(Lx,
                     clin,
                timescale = 1,
                    exnam,
                      tfc = "tfc", ...)
```

Arguments

Lx	A Lexis object with follow-up of a cohort.
clin	A data frame with covariates to add (typically clinical measurements). Must contain a variable lex.id identifying the persons represented in Lx, as well as a variable with the same name as one of the timeScales in Lx, identifying the time at which covariates are measured. The times must be unique within each person; if not records with duplicate times are discarded, and a warning issued. This is done using duplicated, so not very well-defined, it is advisable that you do this yourself.
timescale	Numerical or character. Number or name of a timescale in Lx. The clin data frame must have a variable of this name indicating the time at which the covariate measurements were taken.
exnam	Character. Name of the variable in clin with the examination names (such as wave1, wave2 etc.). Values may not be repeated within person and cannot be equal to any of levels(Lx). Will be carried over to the resulting Lexis object. If there is no such variable in clin it will be constructed; if the argument is omitted, a variable called exnam with values ex1, ex2 etc. will be constructed.
tfc	Character (time from last clinical visit). Name of the variable in the result which will contain the time since the most recent covariate date. It will be included among the timescales of the resulting Lexis object. If the argument is omitted a variable called tfc will be constructed.
...	Arguments passed on. Ignored.

Value

A Lexis object representing the same follow-up as Lx, with cuts added at the times of examination, and covariate measurements added for all records representing follow-up after the most recent time of measurement.

Also tfc is added as a time scale, it is however not a proper timescale since it is reset at every clinical examination. Therefor the value of the timeSince attribute is set to "X" in order to distinguish it from other proper time scales that either have an empty string or the name of a state.

Author(s)

Bendix Carstensen, <b@bxc.dk>, http://bendixcarstensen.com

See Also

cutLexis, mcutLexis, splitLexis, Lexis

Examples

```
# A small bogus cohort
xcoh <- structure( list( id = c("A", "B", "C"),
                      birth = c("1952-07-14", "1954-04-01", "1987-06-10"),
                      entry = c("1965-08-04", "1972-09-08", "1991-12-23"),
                       exit = c("1997-06-27", "1995-05-23", "1998-07-24"),
```

```
                       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 )
xcoh$en <- cal.yr( xcoh$entry )
xcoh$ex <- cal.yr( xcoh$exit  )

# 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 = factor( fail, 0:1, c("Alive","Dead") ),
               data = xcoh )
str( Lcoh )
Lx <- Lcoh[,1:7]

# Data frame with clinical examination data, date of examination in per
clin <- data.frame(lex.id = c(1,1,3,2),
                     per = cal.yr(c("1977-4-7",
                                    "1971-7-1",
                                    "1996-2-15",
                                    "1990-7-3")),
                      bp = c(120,140,160,157),
                    chol = c(5,7,8,9),
                    xnam = c("X2","X1","X1","X2") )
Lx
clin
str(Lx)
str(clin)

# Different behavours when using exnam explicitly
addCov.Lexis( Lx, clin[,-5] )
addCov.Lexis( Lx, clin, exnam="xnam" )

# Works with time split BEFORE
Lb <- addCov.Lexis(splitLexis(Lx,
                              time.scale="age",
                              breaks=seq(0,80,5) ),
                   clin,
                   exnam="clX" )
Lb
# and also AFTER
La <- splitLexis(addCov.Lexis( Lx,
                               clin,
                              exnam = "xnam" ),
                 breaks=seq(0,80,5),
                 time.scale="age" )
La
La$tfc == Lb$tfc
La$age == Lb$age
str(La)
```

```
str(Lb)
```

---

addDrug *Expand a Lexis object with information of drug exposure based on purchase dates and -amounts*

Description

A Lexis object will contain information on follow-up for a cohort of persons through time, each record containing information of one time interval, including the time at the beginning of each interval. If information on drug purchase is known for the persons via lex.id in a list of data frames, addDrug.Lexis will expand the Lexis object by cutting at all drug purchase dates, and compute the exposure status for any number of drugs, and add these as variables.

In some circumstances the result is a Lexis object with a very large number of very small follow-up intervals. The function coarse.Lexis combines consecutive follow-up intervals using the covariates from the first of the intervals.

Usage

```
## S3 method for class 'Lexis'
addDrug(Lx,  # Lexis object
            pdat,  # list of data frames with drug purchase information
             amt = "amt", # name of the variable with purchased amount
             dpt = "dpt", # name of the variable with amount consumed per time
             apt = NULL,  # old name for dpt
          method = "ext", # method use to compute exposure
            maxt = NULL,  # max duration for a purchase when using "fix"
           grace = 0,     # grace  period to be added
            tnam = setdiff(names(pdat[[1]]), c("lex.id", amt))[1],
                           # name of the time variable from Lx
          prefix = TRUE,  # should drug names prefix variable names
         sepfix = ".",   # what should the separator be when forming prefix/suffix
          verbose = TRUE,
            ...)
coarse.Lexis(Lx, lim, keep = FALSE)
```

Arguments

Lx	A Lexis object.
pdat	Named list of data frames with drug purchase data.
amt	Name of the variable in the data frames in pdat with the purchased amount.
dpt	Name of the variable in the data frames in pdat with the consumed dose per time. Must be given in units of units of amt per units of lex.dur in Lx.
apt	Name previously used for dpt. Will disappear in next version.
method	Character. One of "ext" (default), "dos" or "fix", for a description, see details.

maxt	Numerical. Maximal duration for a purchase when using method="fix", same units as lex.dur.
grace	Numeric. Grace period to be added after last time of computed drug coverage to define end of exposure, same units as lex.dur.
tnam	Character. Name of the timescale used in the data frames in pdat.
prefix	Logical. Should the names of pdat be used as prefix for the 4 generated exposure variables for each drug. If false the names of pdat will be used as suffix.
sepfix	Character, used to separate the prefix and the name of the generated type of variable.
verbose	Logical. Should the function tell you about the choices you made?
lim	Numeric vector of length 2. Consecutive follow-up intervals are combined if the first has lex.dur < lim[1], and the sum of lex.dur in the two intervals is smaller than lim[2]. If a scalar i given, c(lim,3*lim) is used.
keep	Logical of length 1 or nrow(Lx) that points to records that cannot be combined with preceding records.
...	Arguments passed on. Ignored.

Details

This function internally uses addCov.Lexis to attach exposure status for several drugs (dispensed medicine) to follow-up in a Lexis object. Once that is done, the exposure measures are calculated at each time.

There is one input data frame per type of drug, each with variables lex.id, amt, a timescale variable and possibly a variable dpt.

Three different methods for computing drug exposures from dates and amounts of purchases are supported via the argument method.

- "ext": Extrapolation: the first drug purchase is assumed consumed over the interval to the second purchase. Exposure for subsequent purchases are assumed to last as long as it would have if consumed at a speed corresponding to the previous purchase being consumed over the time span between the previous and current purchase, plus a period of length grace.
- "dos": Dosage: assumes that each purchase lasts amt/dpt plus grace.
- "fix": Fixed time: assumes that each purchase lasts maxt.

So for each purchase we have defined an end of coverage (expiry date). If next purchase is before this, we assume that the amount purchased is consumed over the period between the two purchases, otherwise over the period to the end of coverage. So the only difference between the methods is the determination of the coverage for each purchase.

Based on this, for each date in the resulting Lexis four exposure variables are computed, see next section.

Value

A Lexis object with the same risk time, states and events as Lx. The follow-up for each person has been cut at the purchase times of each of the drugs, as well as at the expiry times for each drug coverage. Further, for each drug (i.e. the data frame in the pdat list) the name of the pdat component determines the prefix for the 4 variables that will be added. Supposing this is AA for a given drug, then 4 new variables will be:

- AA.ex: logical; is the person exposed in this interval
- AA.tf: numeric: time since first purchase, same units as lex.dur
- AA.ct: numeric: cumulative time on the drug, same units as lex.dur
- AA.cd: numeric: cumulative dose of the drug, same units as amt

So if pdat is a list of length 3 with names c("a","b","c") the function will add variables a.ex, a.tf, a.ct, a.cd, b.ex, b.tf, b.ct, b.cd, c.ex, c.tf, c.ct, c.cd

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

See Also

gen.exp, addCov.Lexis, cutLexis, rcutLexis, mcutLexis

Examples

```
# Follow-up of 2 persons
clear()
fu <- data.frame(doe = c(2006, 2008),
                 dox = c(2015, 2018),
                 dob = c(1950, 1951),
                 xst = factor(c("A","D")))
Lx <- Lexis(entry = list(per = doe,
                         age = doe- dob),
             exit = list(per = dox),
      exit.status = xst,
             data = fu)
Lx <- subset(Lx, select = -c(doe, dob, dox, xst))

# split FU in 1 year intervals
Sx <- splitLexis(Lx, "per", breaks = seq(1990, 2020, 1.0))

# drug purchases, one data frame for each drug
ra <- data.frame(per = c(2007 + runif(12,0,10)),
                 amt = sample(2:4, 12, r = TRUE),
              lex.id = sample(1:2, 12, r = TRUE))
ra <- ra[order(ra$lex.id, ra$per),]

rb <- data.frame(per = c(2009 + runif(10, 0, 10)),
                 amt = sample(round(2:4/3,1), 10, r = TRUE),
              lex.id = sample(1:2, 10, r = TRUE))
rb <- rb[order(rb$lex.id, rb$per),]
```

```
# put in a named list
pdat <- list(A = ra, B = rb)
pdat

ex1 <- addDrug.Lexis(Sx, pdat, method = "ext") # default
summary(ex1)
# collapsing some of the smaller intervals with the next
summary(coarse.Lexis(ex1, c(0.2,0.5)))

ex2 <- addDrug.Lexis(Sx, pdat, method = "ext", grace = 0.2)
dos <- addDrug.Lexis(Sx, pdat, method = "dos", dpt = 6)
fix <- addDrug.Lexis(Sx, pdat, method = "fix", maxt = 1)
```

`adtte2Lexis {Epi}` *Create a Lexis Object Given an ADAM ADTTE Dataset.*

Description

Takes (a subset of) an ADTTE ADaM-format dataset along with choices about recurrent events and outputs a corresponding Lexis-object. Time since first entry to each state is included as a time scale in the Lexis-object.

Usage

```
adtte2Lexis(dat,
            start.state,
            Tr.ev,
            Term.st)
```

Arguments

`dat`	A data.frame with data on a format complying with CDISC ADaM BDS standards, see https://www.cdisc.org/standards/foundational/adam/adam-basic-data-structure See details for necessary columns, and Examples for an example.
`start.state`	A character string, used as the name of the common starting state of all individuals.
`Tr.ev`	A vector containing a subset of the values taken by the EVNTDSC variable in ADTTE indicating possible states.
`Term.st`	A user-specified text string containing the name of the unique censoring state.

Details

The ADTTE data.frame, `dat`, should contain the columns `USUBJID`, `STARTDT`, `ADT`, `AGE`, `EVNTDESC` and `CNSR`. All other columns, including baseline values and FL-variables, will be discarded. If they are necessary in later analyses, they will need to be added back to the resulting Lexis-object manually.

Value

A Lexis-object with possible `lex.Cst` and `lex.Xst` states specified in `Tr.ev` and `Tr.st`, and with time-scales for calender period, `per`, age, `age`, time since entry, `t.sE` and additional time-scales for time since first entry into the states given in `Tr.ev`. Baseline information is not retained.

Author(s)

Lars Diaz, <larsjorgeful@gmail.com>

References

https://www.cdisc.org/standards/foundational/adam/adam-basic-data-structure-bds-time-event-tte-ana

Examples

```
## Creating columns in ADTTE dataset:
STUDYID <- c("ABC123", "ABC123", "ABC123", "ABC123",
             "ABC123", "ABC123", "ABC123", "ABC123", "ABC123", "ABC123",
             "ABC123")

USUBJID <- c("SUBJ001", "SUBJ001", "SUBJ001", "SUBJ002",
             "SUBJ002", "SUBJ002", "SUBJ002", "SUBJ002", "SUBJ002",
             "SUBJ002", "SUBJ002")

AGE <- c(65L, 65L, 65L, 67L, 67L, 67L, 67L, 67L, 67L, 67L,
         67L)

SEX <- c("M", "M", "M", "F", "F", "F", "F", "F", "F", "F", "F")

ARM <- c("Treatment A", "Treatment A", "Treatment A",
         "Treatment A", "Treatment A", "Treatment A", "Treatment A",
         "Treatment A", "Treatment A", "Treatment A", "Treatment A")

PARAMCD <- c("HOSP", "RELAPSE", "RELAPSE", "HOSP", "HOSP",
             "HOSP", "HOSP", "RELAPSE", "RELAPSE", "RELAPSE", "RELAPSE")

PARAM <- c("Time to Hospitl (days)",
           "Time to Relapse (days)",
           "Time to Relapse (days)",
           "Time to Hospitl (days)",
           "Time to Hospitl (days)",
           "Time to Hospitl (days)",
           "Time to Hospitl (days)",
           "Time to Relapse (days)",
           "Time to Relapse (days)",
           "Time to Relapse (days)",
           "Time to Relapse (days)")

EVNTSEQ <- c(1, 1, 2, 1, 2, 3, 4, 1, 2, 3, 4)

STARTDT <- c("2023-01-01", "2023-01-01", "2023-01-31",
             "2023-01-03", "2023-02-13", "2023-05-10", "2023-06-29",
```

```
              "2023-01-03", "2023-03-18", "2023-04-26", "2023-06-18")

ADT <- c("2023-07-20", "2023-01-31", "2023-07-20",
         "2023-02-13", "2023-05-10", "2023-06-29", "2023-07-22",
         "2023-03-18", "2023-04-26", "2023-06-18", "2023-07-22")

ADTF <- c(NA_character_, NA_character_, NA_character_,
          NA_character_, NA_character_, NA_character_, NA_character_,
          NA_character_, NA_character_, NA_character_, NA_character_)

AVAL <- c(200, 30, 170, 41, 86, 50, 23, 74, 39, 53, 34)

CNSR <- c(1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1)

EVNTDESC <- c("Censored", "Relapse", "Censored",
              "Hospitl", "Hospitl", "Hospitl",
              "Censored", "Relapse", "Relapse", "Relapse", "Censored")

ANL01FL <- c("Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y", "Y",
             "Y", "Y")

##collecting into example adtte dataset:
adtte <- data.frame(STUDYID, USUBJID, AGE, SEX, ARM, PARAMCD, PARAM, EVNTSEQ,
                    STARTDT, ADT, ADTF, AVAL, CNSR, EVNTDESC, ANL01FL)
head(adtte)

## transform to Lexis format
adtte2Lexis(adtte,
            start.state = "Rand",
            Tr.ev = c("Relapse", "Hospitl"),
            Term.st = "Censored") -> Lx
Lx
summary(Lx, timeScales = TRUE)
```

`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-Cohort. 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 = "Y",
          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 this 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 residual 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 modeling. if a binomial model us used, Y is assumed to be the denominator; "binomial" gives a binomial model with logit link. The Age-effects returned are converted to the probability scale, Period and Cohort effects are still odds-ratios.
model	Type of model (covariate effects) 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 P-A. npar is ignored in this case.
dr.extr	Character or numeric. How the drift parameter should be extracted from the age-period-cohort model. Specifies the inner product used for definition of orthogonality of the period / cohort effects to the linear effects — in terms of a diagonal matrix. "Y" (default) uses the no. person-time, Y, corresponding to the observed information about the square root of the rate.

"R" or "L" uses Y*Y/D corresponding to the observed information about the rate (usually termed "lambda", hence the "L").

"D" or "T" uses the no. events as the weight in the inner product, corresponding to the information about the log-rate (usually termed "theta", hence the "T").

If given "n" (naive) (well, in fact any other character value) will induce the use of the standard inner product putting equal weight on all units in the dataset.

If dr.extr is a numeric vector this is used as the diagonal of the matrix inducing the inner product.

If dr.extr is a numeric scalar, D + dr.extr*Y is used as the diagonal of the matrix inducing the inner product. This family of inner products are the only ones that meet the split-observation invariance criterion.

The setting of this parameter has no effect on the fit of the model, it only influences the parametrization returned in the Age, Per and Coh elements of the resulting list.

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. In this case returned 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. In this case returned 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/knots to use for each of the terms in the model. If it is vector of length 3, the numbers are taken as the no. of knots for Age, Period and Cohort, respectively. Unless it has a names attribute with values "A", "P" and "C" in which case these will be used. The knots chosen are the quantiles `(1:nk-0.5)/nk` of the events (i.e. of `rep(A,D)` and similarly for `P` and `C`).

`npar` may also be a named list of three numerical vectors with names "A", "P" and "C", in which case these taken as the knots for the age, period and cohort effect, the smallest and largest 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?

Details

Each record in the input data frame represents a subset of a Lexis diagram. The subsets need not be of equal length on the age and period axes, in fact there are no restrictions on the shape of these; they could be Lexis triangles for example. The requirement is that `A` and `P` are coded with the mean age and calendar time of observation in the subset. This is essential since `A` and `P` are used as quantitative variables in the models.

This approach is different from to the vast majority of the uses of APC-models in the literature where a factor model is used for age, period and cohort effects. The latter can be obtained by using `model="factor"`. Note however that the cohort factor is defined from `A` and `P`, so that it is not possible in this framework to replicate the Boyle-Robertson fallacy.

Value

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

Type — Text describing the model and parametrization returned.

Model — The model object(s) on which the parametrization is based.

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

Per — Matrix with 4 columns: `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 columns: `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. The first row name indicates which type of inner product were used for projections. 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.
Anova	Analysis of deviance table comparing the five classical models.
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 of the vectors are the boundary knots.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

References

The considerations behind the parametrizations used in this function are given in detail in: B. Carstensen: Age-Period-Cohort models for the Lexis diagram. Statistics in Medicine, 10; 26(15):3018-45, 2007.

Various links to course material etc. is available through http://bendixcarstensen.com/APC/

See Also

apc.frame, apc.lines, apc.plot, LCa.fit, apc.LCa.

Examples

```
library( Epi )
data(lungDK)

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

# Three different ways of parametrizing the APC-model, ML
ex.1 <- apc.fit( exd, npar=7, model="ns", dr.extr="1", parm="ACP", scale=10^5 )
ex.D <- apc.fit( exd, npar=7, model="ns", dr.extr="D", parm="ACP", scale=10^5 )
ex.Y <- apc.fit( exd, npar=7, model="ns", dr.extr="Y", 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.1[["Drift"]]
ex.D[["Drift"]]
ex.Y[["Drift"]]
ex.S[["Drift"]]

# Plot the effects
lt <- c("solid","22")[c(1,1,2)]
apc.plot( ex.1, lty=c(1,1,3) )
apc.lines( ex.D, col="red", lty=c(1,1,3) )
apc.lines( ex.Y, col="limegreen", lty=c(1,1,3) )
apc.lines( ex.S, col="blue", lty=c(1,1,3) )
```

`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)))/10,
         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://bendixcarstensen.com

References

B. Carstensen: Age-Period-Cohort models for the Lexis diagram. Statistics in Medicine, 26: 3018-3045, 2007.

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" )
# or referring to the period-cohort part of the plot
pc.points( 1960, 0.3, pch=16, cex=1, col="green" )
```

---

apc.LCa *Fit Age-Period-Cohort models and Lee-Carter models with effects modeled by natural splines.*

Description

apc.LCa fits an Age-Period-Cohort model and sub-models (using apc.fit) as well as Lee-Carter models (using LCa.fit). show.apc.LCa plots the models in little boxes with their residual deviance with arrows showing their relationships.

Usage

```
apc.LCa( data,
  keep.models = FALSE,
           ... )
show.apc.LCa( x,
       dev.scale = TRUE,
             top = "Ad", ... )
```

Arguments

data	A data frame that must have columns A, P, D and Y, see e.g. apc.fit
keep.models	Logical. Should the apc object and the 5 LCa objects be returned too?
...	Further parameters passed on to LCa.fit or boxes.matrix.
x	The result from a call to apc.LCa.
dev.scale	Should the vertical position of the boxes with the models be scales relative to the deviance between the Age-drift model and the extended Lee-Carter model?
top	The model presented at the top of the plot of boxes (together with any other model with larger deviance) when vertical position is scaled by deviances. Only "Ad", "AP", "AC", "APa" or "ACa" will make sense.

Details

The function apc.LCa fits all 9 models (well, 10) available as extension and sub-models of the APC-model and compares them by returning deviance and residual df.

Value

A 9 by 2 matrix classified by model and deviance/df; optionally (if models=TRUE) a list with the matrix as dev, apc, an apc object (from apc.fit), and LCa, a list with 5 LCa objects (from LCa.fit).

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

See Also

apc.fit, LCa.fit

Examples

```
library( Epi )
clear()
# Danish lung cancer incidence in 5x5x5 Lexis triangles
data( lungDK )
lc <- subset( lungDK, Ax>40 )[,c("Ax","Px","D","Y")]
names( lc )[1:2] <- c("A","P")
head( lc )

al <- apc.LCa( lc, npar=c(9,6,6,6,10), keep.models=TRUE, maxit=500, eps=10e-3 )
show.apc.LCa( al, dev=TRUE )

# Danish mortality data
## Not run:
data( M.dk )
mdk <- subset( M.dk, sex==1 )[,c("A","P","D","Y")]
head( mdk )

al <- apc.LCa( mdk, npar=c(15,15,20,6,6), maxit=50, eps=10e-3,
               quiet=FALSE, VC=FALSE )
show.apc.LCa( al, dev=FALSE )
show.apc.LCa( al, dev=TRUE )
show.apc.LCa( al, top="AP" )

# Fit a reasonable model to Danish mortality data and plot results
mAPa <- LCa.fit( mdk, model="APa", npar=c(15,15,20,6,6), c.ref=1930,
                 a.ref=70, quiet=FALSE, maxit=250 )
par( mfrow=c(1,3) )
plot( mAPa )
## End(Not run)
```

apc.lines *Plot APC-estimates 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

```
## S3 method for class 'apc'
lines( x, 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,
        shade = FALSE,
          ... )
 apc.lines( x, 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,
        shade = FALSE,
          ... )
```

Arguments

x	If an apc-object, (see apc.fit), then the arguments P, C, c0, a0 and p0 are ignored, and the estimates from x plotted.

	Can also be a 4-column matrix with columns age, age-specific rates, lower and upper c.i., in which case period and cohort effects are taken from the arguments `P` and `C`.
`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?
`shade`	Should confidence intervals be plotted as shaded areas? If true, the setting of `ci` is ignored.
`...`	Further parameters to be transmitted to `points` `lines`, `matpoints` or `matlines` used for plotting the three sets of curves.

Details

There is no difference between the functions `apc.lines` and `lines.apc`, except the the latter is the `lines` method for `apc` objects.

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.

Value

`apc.lines` returns (invisibly) a list of three matrices of the effects plotted.

Author(s)

Bendix Carstensen, Steno Diabetes Center, http://bendixcarstensen.com

See Also

apc.frame, pc.lines, apc.fit, apc.plot

B.dk *Births in Denmark by year and month of birth and sex*

Description

The number of live births as entered from printed publications from Statistics Denmark.

Usage

```
data(B.dk)
```

Format

A data frame with 1248 observations on the following 4 variables.

`year` Year of birth

`month` Month of birth

`m` Number of male live births

`f` Number of female live births

Details

Division of births by month and sex is only available for the years 1957–69 and 2002ff. For the remaining period, the total no. births in each month is divided between the sexes so that the fraction of boys is equal to the overall fraction for the years where the sex information is available.

There is a break in the series at 1920, when Sonderjylland was joined to Denmark.

Source

Statistiske Undersogelser nr. 19: Befolkningsudvikling og sundhedsforhold 1901-60, Copenhagen 1966. Befolkningens bevaegelser 1957 Befolkningens bevaegelser 1958 ... Befolkningens bevaegelser 2003 Befolkningens bevaegelser 2004 Vital Statistics 2005 Vital Statistics 2006

Examples

```
data(B.dk)
str(B.dk)
attach(B.dk)
# Plot the no of births and the M/F-ratio
par(las = 1, mar = c(4,4,2,4))
matplot(year + (month - 0.5) / 12, cbind(m, f),
        bty = "n", col = c("blue", "red"), lty = 1, lwd = 1, type = "l",
        ylim = c(0, 5000), xlab = "Date of birth", ylab = "" )
usr <- par()$usr
mtext("Monthly no. births in Denmark", side = 3,
      at = usr[1],  adj = 0.25, line = 1/1.6)
text(usr[1:2] %*% cbind(c(19,1), c(19,1)) / 20,
     usr[3:4] %*% cbind(c(1,19), c(2,18)) / 20,
     c("Boys","Girls"), col = c("blue","red"), adj = 0 )
lines(year + (month - 0.5) / 12, (m / (m + f) - 0.5) * 30000, lwd = 1)
axis(side = 4, at = (seq(0.505, 0.525, 0.005)-0.5) * 30000, labels = NA, tcl = -0.3 )
axis(side = 4, at = (50:53 / 100 - 0.5) * 30000, labels = 50:53, tcl = -0.5 )
axis(side = 4, at = (0.54 - 0.5) * 30000, labels = "% boys",
     tick = FALSE, mgp = c(3,0.1,0))
abline(v = 1920, col = gray(0.8))
```

---

bdendo *A case-control study of endometrial cancer*

Description

The bdendo data frame has 315 rows and 13 columns, bdendo11 126 rows. These data concern a study in which each case of endometrial cancer was matched with 4 controls. bdendo11 is a 1:1 mathed subset of bdendo. Matching was by date of birth (within one year), marital status, and residence.

Format

These data frames have 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: a 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: a 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)
str(bdendo)
```

---

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

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)
```

bootLexis *Create a bootstrap sample of persons (as identified by* lex.id*) from a Lexis object*

Description

lex.id is the person identifier in a Lexis object. This is used to sample persons from a Lexis object. If a person is sampled, all records from this persons is transported to the bootstrap sample.

Usage

```
nid( Lx, ... )
## S3 method for class 'Lexis'
nid( Lx, by=NULL, ... )
bootLexis( Lx, size = NULL, by = NULL, replace=TRUE )
```

Arguments

Lx	A Lexis object.
...	Parameters passed on to other methods.
size	Numeric. How many persons should be sampled from the Lexis object. Defaults to the number of persons in the Lx, or, if by is given, to the number of persons in each level of by. If by is given, size can have length length(unique(by)), to indicate how many are sampled from each level of by.

by	Character. Name of a variable (converted to factor) in the Lexis object. Bootstrap sampling is done within each level of by. Calculation of the number of persons (lex.id) is done within each level of by, and a vector returned.
replace	Should persons be sampled by replacement? Default is TRUE. Setting replace to FALSE enables selecting a random subset of persons from the Lexis object.

Value

bootLexis returns a Lexis object of the same structure as the input, with *persons* bootstrapped. The variable lex.id in the resulting Lexis object has values 1,2,... The original values of lex.id from Lx are stored in the variable old.id.

nid counts the number of persons in a Lexis object, possibly by by. If by is given, a named vector is returned.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com.

See Also

Relevel.Lexis,subset.Lexis

Examples

```
# A small bogus cohort
xcoh <- data.frame( id = c("A", "B", "C"),
                 birth = c("1952-07-14", "1954-04-01", "1987-06-10"),
                 entry = c("1965-08-04", "1972-09-08", "1991-12-23"),
                  exit = c("1997-06-27", "1995-05-23", "1998-07-24"),
                  fail = c(1, 0, 1),
                   sex = c("M","F","M") )

# Convert to calendar years
for( i in 2:4 ) xcoh[,i] <- cal.yr(xcoh[,i])
xcoh <- xcoh[sample(1:3, 10, replace = TRUE),]
xcoh$entry <- xcoh$entry + runif(10, 0, 10)
xcoh$exit  <- xcoh$entry + runif(10, 0, 10)

Lcoh <- Lexis(entry = list(per = entry),
              exit = list(per = exit,
                          age = exit - birth),
        exit.status = fail,
              data = xcoh)
Lcoh

Lx <- splitLexis(Lcoh, breaks = 0:10 * 10, "age")
Lx
nid(Lx)
nid(Lx, by="sex")
Lb <- bootLexis(Lx)
```

```
head(Lb)
nid(bootLexis(Lx, size = 7))
Li <- bootLexis(Lx, by = "id") # superfluous
summary(Lx)
summary(Li)
L2 <- bootLexis(Lx, by = "sex", size = c(2, 5))
nid(L2, by = "sex")
summary(L2, by = "sex")
```

---

boxes.MS *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. The boxes method for Lexis objects generates displays of states with person-years and transitions with events or rates.

Usage

```
   tbox( txt, x, y, wd, ht,
         font=2, lwd=2,
         col.txt=par("fg"),
         col.border=par("fg"),
         col.bg="transparent" )
   dbox( x, y, wd, ht=wd,
         font=2, lwd=2, cwd=5,
         col.cross=par("fg"),
         col.border=par("fg"),
         col.bg="transparent"  )
   boxarr( b1, b2, offset=FALSE, pos=0.45, ... )
## S3 method for class 'Lexis'
boxes( obj,
                    boxpos = FALSE,
                     wmult = 1.20,
                     hmult = 1.20 + 0.85*(!show.Y),
                       cex = 1.40,
                    show   = inherits( obj, "Lexis" ),
                    show.Y = show,
                   scale.Y = 1,
                  digits.Y = 1,
                   show.BE = FALSE,
                    BE.sep = c("","","          ",""),
                    show.D = show,
                   scale.D = FALSE,
                  digits.D = as.numeric(as.logical(scale.D)),
                    show.R = show & is.numeric(scale.R),
                   scale.R = 1,
```

```
                    digits.R = as.numeric(as.logical(scale.R)),
                      DR.sep = if( show.D ) c("\n(",")") else c("",""),
                       eq.wd = TRUE,
                       eq.ht = TRUE,
                          wd,
                          ht,
                      subset = NULL,
                     exclude = NULL,
                        font = 1,
                         lwd = 2,
                     col.txt = par("fg"),
                  col.border = col.txt,
                      col.bg = "transparent",
                     col.arr = par("fg"),
                     lwd.arr = lwd,
                    font.arr = font,
                     pos.arr = 0.45,
                     txt.arr = NULL,
                 col.txt.arr = col.arr,
                  offset.arr = 2,
                               ... )
## S3 method for class 'matrix'
boxes( obj, ... )
## S3 method for class 'MS'
boxes( obj, sub.st, sub.tr, cex=1.5, ... )
   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 box borders.
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 square matrix indexed by state in both dimensions, and the (i, j)th entry different from NA if a transition i to j can occur. If show.D=TRUE, the arrows between states are annotated by these numbers. If show.Y=TRUE, the boxes representing states are annotated by the numbers in the diagonal of obj. For boxes.matrix obj is a matrix and for boxes.MS, obj is an MS.boxes object (see below).
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. These must be numbers between 0 and 100 indicating percentages of the display in the two directions.
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	If logical: Should person-years be put in the boxes. If numeric: Numbers to put in boxes.
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.BE	Logical. Should number of persons beginning resp. ending follow up in each state be shown? If given as character "nz" or "noz" the numbers will be shown, but zeros omitted.
BE.sep	Character vector of length 4, used for annotation of the number of persons beginning and ending in each state: 1st element precedes no. beginning, 2nd trails it, 3rd precedes the no. ending (defaults to 8 spaces), and the 4th trails the no. ending.
show.D	Should no. transitions be put alongside the arrows. Ignored if obj is not a Lexis object.
scale.D	Synonymous with scale.R, retained for compatibility.
digits.D	Synonymous with digits.R, retained for compatibility.
show.R	Should the transition rates be shown on the arrows?
scale.R	If this a scalar, rates instead of no. transitions are printed at the arrows, scaled by scale.R.
digits.R	How many digits after the decimal point should be used for the rates.
DR.sep	Character vector of length 2. If rates are shown, the first element is inserted before and the second after the rate.

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 widths of the arrows.
font.arr	Font of the text annotation the arrows.
pos.arr	Numerical between 0 and 1, determines the position on the arrows where the text is written.
txt.arr	Text put on the arrows.
col.txt.arr	Colors for text on the arrows.
offset.arr	The amount offset between arrows representing two-way transitions, that is where there are arrows both ways between two boxes.
sub.st	Subset of the states to be drawn.
sub.tr	Subset of the transitions to be drawn.
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. The default is to annotate the transitions with the number of transitions.

A transition matrix can also be supplied, in which case the row/column names are used as state names, diagonal elements taken as person-years, and off-diagonal elements as number of transitions. This also works for `boxes.matrix`.

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.

`boxes.MS` re-draws what `boxes.Lexis` has done based on the object of class `MS` produced by `boxes.Lexis`. The point being that the `MS` object is easily modifiable, and thus it is a machinery to make variations of the plot with different color annotations etc.

`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.

The function `boxes.Lexis` returns an `MS` object, a list with five elements: 1) `Boxes` - a data frame with one row per box and columns `xx`, `yy`, `wd`, `ht`, `font`, `lwd`, `col.txt`, `col.border` and `col.bg`, 2) an object `State.names` with names of states (possibly an expression, hence not possible to include as a column in `Boxes`), 3) a matrix `Tmat`, the transition matrix, 4) a data frame, `Arrows` with one row per transition and columns: `lwd.arr`, `col.arr`, `pos.arr`, `col.txt.arr`, `font.arr` and `offset.arr` and 5) an object `Arrowtext` with names of states (possibly an expression, hence not possible to include as a column in `Arrows`)

An `MS` object is used as input to `boxes.MS`, the primary use is to modify selected entries in the `MS` object first, e.g. colors, or supply sub-setting arguments in order to produce displays that have the same structure, but with different colors etc.

Author(s)

Bendix Carstensen

See Also

`tmat.Lexis`, `legendbox`

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")
tm
boxes.matrix( tm, boxpos=TRUE )

# Illustrate texting of arrows
boxes.Lexis( tm, boxpos=TRUE, txt.arr=c("en","to","tre","fire") )
zz <- boxes( tm, boxpos=TRUE, txt.arr=c(expression(lambda[C]),
                                        expression(mu[C]),
                                        "recovery",
                                        expression(mu[R]) ) )

# Change color of a box
zz$Boxes[3,c("col.bg","col.border")] <- "green"
boxes( zz )

# 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[1:1000,] )

# Cut follow-up at Insulin
dmi <- cutLexis( dml, cut=dml$doins, new.state="Ins", pre="DM" )
summary( dmi )
boxes( dmi, boxpos=TRUE )
boxes( dmi, boxpos=TRUE, show.BE=TRUE )
boxes( dmi, boxpos=TRUE, show.BE="nz" )
boxes( dmi, boxpos=TRUE, show.BE="nz", BE.sep=c("In:","      Out:","") )

# Set up a bogus recovery date just to illustrate two-way transitions
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 )
boxes( dmR, boxpos=TRUE, show.D=FALSE )
boxes( dmR, boxpos=TRUE, show.D=FALSE, show.Y=FALSE )
boxes( dmR, boxpos=TRUE, scale.R=1000 )
MSobj <- boxes( dmR, boxpos=TRUE, scale.R=1000, show.D=FALSE )
MSobj <- boxes( dmR, boxpos=TRUE, scale.R=1000, DR.sep=c(" (",")") )
class( MSobj )
boxes( MSobj )
MSobj$Boxes[1,c("col.txt","col.border")] <- "red"
MSobj$Arrows[1:2,"col.arr"] <- "red"
boxes( MSobj )
```

BrCa *Clinical status, relapse, metastasis and death in 2982 women with breast cancer.*

Description

This dataset is a transformation of the example dataset used by Crowther and Lambert in their multistate paper.

Usage

```
data(BrCa)
```

Format

A data frame with 2982 observations on the following 17 variables:

`pid` Person-id; numeric

`year` Calendar year of diagnosis

`age` Age at diagnosis

`meno` Menopausal status; a factor with levels `pre` `post`

`size` Tumour size; a factor with levels `<=20 mm` `>20-50 mm` `>50 mm`

`grade` Tumour grade; a factor with levels `2` `3`

`nodes` Number of positive lymph nodes, a numeric vector

`pr` Progesteron receptor level

`pr.tr` Transformed progesteron level

`er` Estrogen receptor level

`hormon` Hormon therapy at diagnosis; a factor with levels `no` `yes`

`chemo` Chemotherapy treatment; a factor with levels `no` `yes`

`tor` Time of relapse, years since diagnosis

`tom` Time of metastasis, years since diagnosis

`tod` Time of death, years since diagnosis

`tox` Time of exit from study, years since diagnosis

`xst` Vital status at exit; a factor with levels `Alive` `Dead`

Details

The dataset has been modified to contain the times (since diagnosis) of the events of interest, to comply with the usual structure of data.

Source

The original data were extracted from: http://fmwww.bc.edu/repec/bocode/m/multistate_example.dta, this is modified representation of the same amount of information.

References

The data were used as example in the paper by Crowther and Lambert: Parametric multistate survival models: Flexible modelling allowing transition-specific distributions with application to estimating clinically useful measures of effect differences; Stat Med 36 (29), pp 4719-4742, 2017. (No, it is not the paper, just the title.)

A parallel analysis using the `Lexis` machinery is available as: http://bendixcarstensen.com/AdvCoh/papers/bcMS.pdf

Examples

```
data(BrCa)
```

---

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 January 1, 2000)

`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` and `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 )
  ## S3 method for class 'cal.yr'
as.Date( 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 Copenhagen, <b@bxc.dk>, http://bendixcarstensen.com

See Also

DateTimeClasses, Date

Examples

```
# Character vector of dates:
birth <- c("14/07/1852","01/04/1954","10/06/1987","16/05/1990",
           "12/11/1980","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) )
```

cbind.Lexis *Combining a Lexis objects with data frames or other Lexis objects*

Description

A Lexis object may be combined side-by-side with data frames. Or several Lexis objects may stacked, possibly increasing the number of states and time scales.

Usage

```
## S3 method for class 'Lexis'
cbind(...)
## S3 method for class 'Lexis'
rbind(...)
```

Arguments

...	For cbind a sequence of data frames or vectors of which exactly one has class Lexis. For rbind a sequence of Lexis objects, supposedly representing follow-up in the same population.

Details

Arguments to rbind.Lexis must all be Lexis objects; except for possible NULL objects. The timescales in the resulting object will be the union of all timescales present in all arguments. Values of timescales not present in a contributing Lexis object will be set to NA. The breaks for a given time scale will be NULL if the breaks of the same time scale from two contributing Lexis objects are different.

The arguments to cbind.Lexis must consist of at most one Lexis object, so the method is intended for amending a Lexis object with extra columns without losing the Lexis-specific attributes.

Value

A Lexis object. rbind renders a Lexis object with timescales equal to the union of timescales in the arguments supplied. Values of a given timescale are set to NA for rows corresponding to supplied objects. cbind basically just adds columns to an existing Lexis object.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

See Also

subset.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 <- cal.yr( xcoh, format="%d/%m/%Y", wh=2:4 )
# See how it looks
xcoh
str( 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
cbind( Lcoh, zz=3:5 )

# Lexis object wit time since entry time scale
Dcoh <- Lexis( entry = list( per=entry, tfe=0 ),
               exit = list( per=exit ),
         exit.status = fail,
               data = xcoh )
# A bit meningless to combie these two, really...
rbind( Dcoh, Lcoh )

# Split different places
sL <- splitLexis( Lcoh, time.scale="age", breaks=0:20*5 )
sD <- splitLexis( Dcoh, time.scale="tfe", breaks=0:50*2 )
sDL <- rbind( sD, sL )
```

---

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=FALSE )
```

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.Crisk` *Compute cumulative risks and expected sojourn times from models for cause-specific rates.*

Description

Consider a list of parametric models for rates of competing events, such as different causes of death, A, B, C, say. From estimates of the cause-specific rates we can compute 1) the cumulative risk of being in each state ('Surv' (=no event) and A, B and C) at different times, 2) the stacked cumulative rates such as A, A+C, A+C+Surv and 3) the expected (truncated) sojourn times in each state up to each time point.

This can be done by simple numerical integration using estimates from models for the cause specific rates. But the standard errors of the results are analytically intractable.

The function `ci.Crisk` computes estimates with confidence intervals using simulated samples from the parameter vectors of supplied model objects. Some call this a parametric bootstrap.

The times and other covariates determining the cause-specific rates must be supplied in a data frame which will be used for predicting rates for all transitions.

Usage

```
ci.Crisk(mods,
           nd,
         tnam = names(nd)[1],
           nB = 1000,
         perm = length(mods):0 + 1,
        alpha = 0.05,
      sim.res = 'none')
```

Arguments

`mods`	A named list of `glm`/`gam` model objects representing the cause-specific rates. If the list is not named the function will crash. The names will be used as names for the states (competing risks), while the state without any event will be called `"Surv"`.
`nd`	A data frame of prediction points and covariates to be used on all models supplied in `mods`.
`tnam`	Name of the column in `nd` which is the time scale.It must represent endpoints of equidistant intervals.
`nB`	Scalar. The number of simulations from the (posterior) distribution of the model parameters to be used in computing confidence limits.
`perm`	Numerical vector of length `length(mods)+1` indicating the order in which states are to be stacked. The `'Surv'` state is taken to be the first, the remaining in the reverse order supplied in the `mods` argument. The default is therefore to stack with the survival as the first, which may not be what you normally want.

alpha — numeric. 1 minus the confidence level used in calculating the c.i.s

sim.res — Character. What simulation samples should be returned. If 'none' (the default) the function returns a list of 3 arrays (see under 'value'). If 'rates' it returns an array of dimension nrow(nd) x length(mod) x nB of bootstrap samples of the rates. If 'crisk' it returns an array of dimension nrow(nd) x length(mod)+1 x nB of bootstrap samples of the cumulative rates. Only the first letter matters, regardless of whether it is in upper lower case.

Value

If sim.res='none' a named list with 4 components, the first 3 are 3-way arrays classified by time, state and estimate/confidence interval:

- Crisk Cumulative risks for the length(mods) events *and* the survival
- Srisk Stacked versions of the cumulative risks
- Stime Sojourn times in each states
- time Endpoints of intervals. It is just the numerical version of the names of the first dimension of the three arrays

All three arrays have (almost) the same dimensions:

- time, named as tnam; endpoints of intervals. Length nrow(nd).
- cause. The arrays Crisk and Stime have values "Surv" plus the names of the list mods (first argument). Srisk has length length(mod), with each level representing a cumulative sum of cumulative risks, in order indicated by the perm argument.
- Unnamed, ci.50%, ci.2.5%, ci.97.5% representing quantiles of the quantities derived from the bootstrap samples. If alpha is different from 0.05, names are of course different.

If sim.res='rates' the function returns bootstrap samples of rates for each cause as an array classified by time, cause and bootstrap sample.

If sim.res='crisk' the function returns bootstrap samples of cumulative risks for each cause (including survival) as an array classified by time, state (= causes + surv) and bootstrap sample.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

See Also

mat2pol simLexis plotCIF ci.surv

Examples

```
library(Epi)
data(DMlate)

# A Lexis object for survival
Ldm <- Lexis(entry = list( per = dodm,
                           age = dodm-dobth,
```

```
                            tfd = 0 ),
               exit = list( per = dox ),
        exit.status = factor( !is.na(dodth), labels = c("DM","Dead") ),
               data = DMlate[sample(1:nrow(DMlate),1000),] )
summary(Ldm, timeScales = TRUE)

# Cut at OAD and Ins times
Mdm <- mcutLexis(Ldm,
                  wh = c('dooad','doins'),
           new.states = c('OAD','Ins'),
           seq.states = FALSE,
                 ties = TRUE)
summary(Mdm$lex.dur)

# restrict to DM state and split
Sdm <- splitLexis(factorize(subset(Mdm, lex.Cst == "DM")),
                  time.scale = "tfd",
                  breaks = seq(0,20,1/12))
summary(Sdm)
summary(Relevel(Sdm, c(1, 4, 2, 3)))

boxes(Relevel(Sdm, c(1, 4, 2, 3)),
      boxpos = list(x = c(15, 85, 80, 15),
                    y = c(85, 85, 20, 15)),
      scale.R = 100)

# glm models for the cause-specific rates
system.time(
mD <- glm.Lexis(Sdm, ~ Ns(tfd, knots=0:6*2), to = 'Dead') )
system.time(
mO <- glm.Lexis(Sdm, ~ Ns(tfd, knots=0:6*2), to = 'OAD' ) )
system.time(
mI <- glm.Lexis(Sdm, ~ Ns(tfd, knots=0:6*2), to = 'Ins' ) )

# intervals for calculation of predicted rates
int <- 1 / 100
nd <- data.frame(tfd = seq(0, 10, int)) # not the same as the split,
                                        # and totally unrelated to it

# cumulaive risks with confidence intervals
# (too few timepoints, too few simluations)
system.time(
res <- ci.Crisk(list(OAD = mO,
                     Ins = mI,
                    Dead = mD),
                        nd = data.frame(tfd = 0:100 / 10),
                        nB = 100,
                      perm = 4:1))
str(res)
```

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

Description

Computes the cumulative sum of parameter functions and the standard error of it. Used for computing the cumulative rate, or the survival function based on a `glm` with parametric baseline.

Usage

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

Arguments

`obj`	A model object (of class `lm`, `glm`.
`ctr.mat`	Matrix or data frame. If `ctr.mat` is a matrix, it should be a contrast matrix to be multiplied to the parameter vector, i.e. the desired linear function of the parameters. If it is a data frame it should have columns corresponding to a prediction data frame for `obj`, see details for `ci.lin`.
`subset`	Subset of the parameters of the model to which a matrix `ctr.mat` should be applied.
`intl`	Interval length for the cumulation. Either a constant or a numerical vector of length `nrow(ctr.mat)`. If omitted taken as the difference between the two first elments if the first column in `ctr.mat`, assuming that that holds the time scale. A note is issued in this case.
`alpha`	Significance level used when computing confidence limits.
`Exp`	Should the parameter function be exponentiated before it is cumulated?
`ci.Exp`	Should the confidence limits for the cumulative rate be computed on the log-scale, thus ensuring that exp(-cum.rate) is always in [0,1]?
`sample`	Should a sample of the original parameters be used to compute a cumulative rate?

Details

The purpose of `ci.cum` is to the compute cumulative rate (integrated intensity) at a set of points based on a model for the rates. `ctr.mat` is a matrix which, when premultiplied to the parameters

of the model returns the (log)rates at a set of equidistant time points. If log-rates are returned from the prediction method for the model, the they should be exponentiated before cumulated, and the variances computed accordingly. Since the primary use is for log-linear Poisson models the `Exp` parameter defaults to TRUE.

Each row in the object supplied via `ctr.mat` is assumed to represent a midpoint of an interval. `ci.cum` will then return the cumulative rates at the *end* of these intervals. `ci.surv` will return the survival probability at the *start* of each of these intervals, assuming the the first interval starts at 0 - the first row of the result is `c(1,1,1)`.

The `ci.Exp` argument ensures that the confidence intervals for the cumulative rates are always positive, so that exp(-cum.rate) is always in [0,1].

Value

A matrix with 3 columns: Estimate, lower and upper c.i. If `sample` is TRUE, a single sampled vector is returned, 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.

`ci.surv` returns a 3 column matrix with estimate, lower and upper confidence bounds for the survival function.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

See Also

See also `ci.lin`, `ci.pred`

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)

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

# Fit a Poisson model with a natural spline for the effect of time (left
# end points of intervals are used as covariate)
```

```
mp <- glm(cbind(lex.Xst == 1, lex.dur)
          ~ Ns(tfd,knots = c(0, 50, 100, 200, 400, 700)),
          family = poisreg,
            data = sL)

# mp is now a model for the rates along the time scale tfd
# prediction data frame for select time points on this time scale
nd <- data.frame(tfd = seq(5,995,10)) # *midpoints* of intervals
Lambda <- ci.cum ( mp, nd, intl=10 )
surv   <- ci.surv( mp, nd, intl=10 )

# Put the estimated survival function on top of the KM-estimator
# recall the ci.surv return the survival at *start* of intervals
matshade(nd$tfd - 5, surv, col = "Red", alpha = 0.15)

# Extract and plot the fitted intensity function
lambda <- ci.pred(mp, nd) * 365.25 # mortality
matshade(nd$tfd, lambda, log = "y", ylim = c(0.2, 5), plot = TRUE,
          xlab = "Time since diagnosis",
          ylab = "Mortality per year")

# same thing works with gam from mgcv
library(mgcv)
mg <- gam(cbind(lex.Xst == 1, lex.dur) ~ s(tfd), family = poisreg, data=sL )
matshade(nd$tfd - 5, ci.surv(mg, nd, intl=10), plot=TRUE,
         xlab = "Days since diagnosis", ylab="P(survival)")
matshade(nd$tfd  , ci.pred(mg, nd) * 365.25, plot=TRUE, log="y",
         xlab = "Days since diagnosis", ylab="Mortality per 1 py")
```

`ci.eta` *Linear predictor (*`eta`*) from a formula, coefficients, vcov and a prediction frame.*

Description

Computes the linear predictor with its confidence limits from the model formula and the estimated parameters with the vcov.

Usage

```
ci.eta(form, cf, vcv, newdata,
       name.check = TRUE,
       alpha = 0.05, df = Inf, raw = FALSE)
```

Arguments

`form`	A model formula. A one-sided formula will suffice; left side will be ignored if two-sided.
`cf`	Coefficients from a model using `formula`.

vcv	variance-covariance matrix from a model using formula.
newdata	Prediction data frame with variables used in formula. Can also be a list of 2 or 4 prediction frames, for details see ci.lin.
name.check	Logical. Check if the column names of the genereated model matrix are identical to the names of the supplied coef vector.
alpha	Significance level for calculation of c.i.
df	Integer. Number of degrees of freedom in the t-distribution used to compute the quantiles used to construct the confidence intervals.
raw	Logical. Should predictions and their vcov be returned instead of predictions and confidence limits?

Details

Does pretty much the same as ci.lin, but requires only a formula and coefficients with vcov and not a full model object. Designed to avoid saving entire (homongously large) model objects and still be able to compute predictions. But only the linear predictor is returned, if there is a link in your model function it is your own responsibility to back-transform. If the model formula contains reference to vectors of spline knots or similar these must be in the global environment.

There is no guarantee that this function works for models that do not inherit from lm. But there is a guarantee that it will not work for gam objects with s() terms.

Value

The linear predictor for the newdata with a confidence interval as a nrow(newdata) by 3 matrix. If raw=TRUE, a list the linear predictor (eta) and its variance-covariance matrix (var).

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

See Also

ci.lin

---

ci.lin *Compute linear functions of parameters with standard errors and confidence limits, optionally transforming to a different scale.*

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,
      xvars = NULL,
      diffs = FALSE,
       fnam = !diffs,
       vcov = FALSE,
      alpha = 0.05,
         df = Inf,
        Exp = FALSE,
     sample = FALSE )
ci.exp( ..., Exp = TRUE, pval = FALSE )
Wald( obj, H0=0, ... )
ci.mat( alpha = 0.05, df = Inf )
ci.pred( obj,
     newdata,
         Exp = NULL,
       alpha = 0.05 )
ci.ratio( r1, r2,
          se1 = NULL,
          se2 = NULL,
      log.tr = !is.null(se1) & !is.null(se2),
       alpha = 0.05,
        pval = FALSE )
```

Arguments

obj	A model object (in general of class glm, but for ci.lin and ci.exp it may also be of class lm, coxph, survreg, clogistic, cch, lme, mer, lmerMod, glmerMod, gls, nls, gnlm, MIresult, mipo, polr, or rq).
ctr.mat	Matrix, data frame or list (of two or four data frames). If ctr.mat is a matrix, it should be a contrast matrix to be multiplied to the parameter vector, i.e. the desired linear function of the parameters. If it is a data frame it should have columns corresponding to a prediction frame, see details. If it is a list, it must contain two or four data frames that are (possibly partial) prediction frames for obj, see argument xvars and details.
xvars	Character vector. If quantitative variables in the model are omitted from data frames supplied in a list to ctr.mat, they should be listed here. Omitted factors need not be mentioned here.
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	Character. subset selection, but where each element of the character vector is used to select a subset of parameters and only the intersection of these is returned.
diffs	If TRUE, all differences between parameters in the subset are computed, and the subset argument is required. The argument 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 string 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	For ci.lin, if TRUE columns 5:6 are replaced with exp(columns 1,5,6). For ci.exp, if FALSE, the untransformed parameters are returned. For ci.pred it indicates whether the predictions should be exponentiated - the default (Exp=NULL) is to make a prediction with a Wald CI on the scale of the linear predictor and back-transform it by the inverse link function; if FALSE, the prediction on the link scale is returned.
sample	Logical or numerical. If TRUE or numerical a sample of size as.numeric(sample) is drawn from the multivariate normal with mean equal to the (subset defined) coefficients and variance equal to the estimated variance-covariance of these. These are then transformed by ctr.mat and returned.
pval	Logical. Should a column of P-values be included with the estimates and confidence intervals output by ci.exp.
H0	Numeric. 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.
newdata	Data frame of covariates where prediction is made.
r1, r2	Estimates of rates in two independent groups, with confidence limits. Can be either 3-column matrices or data frames with estimates and confidence intervals or 2 two column structures with confidence limits only. Only the confidence limits are used.
se1, se2	Standard errors of log-rates in the two groups. If given, it is assumed that r1 and r2 represent log-rates.
log.tr	Logical, if true, it is assumed that r1 and r2 represent log-rates with confidence intervals.

Value

ci.lin returns a matrix with number of rows and row names as ctr.mat. The columns are Estimate, Std.Err, z, P, 2.5% and 97.5% (or according to the value of alpha). If vcov=TRUE, instead a list of length 2 with components coef (a vector), the desired functional of the parameters and vcov (a square matrix), 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.).

ci.exp returns only the exponentiated parameter estimates with confidence intervals. It is merely a wrapper for ci.lin, fishing out the last 3 columns from ci.lin(...,Exp=TRUE). If you just want the estimates and confidence limits, but not exponentiated, use ci.exp(...,Exp=FALSE).

If ctr.mat is a data frame, the model matrix corresponding to this is constructed and supplied. This is only supported for objects of class lm, glm, gam and coxph.

So the default behaviour will be to produce the same as ci.pred, apparently superfluous. The purpose of this is to allow the use of the arguments vcov that produces the variance-covariance matrix of the predictions, and sample that produces a sample of predictions using sampling from the multivariate normal with mean equal to parameters and variance equal to the hessian.

If ctr.mat is a list of two data frames, the difference of the predictions from using the first versus the last as newdata arguments to predict is computed. Columns that would be identical in the two data frames can be omitted (see below), but names of numerical variables omitted must be supplied in a character vector xvars. Factors omitted need not be named.

If the second data frame has only one row, this is replicated to match the number of rows in the first. This facility is primarily aimed at teasing out RRs that are non-linear functions of a quantitative variable without setting up contrast matrices using the same code as in the model. Note however if splines are used with computed knots stored in a vector such as Ns(x,knots=kk) then the kk must be available in the (global) environment; it will not be found inside the model object. In practical terms it means that if you save model objects for later prediction you should save the knots used in the spline setups too.

If ctr.mat is a list of four data frames, the difference of the difference of predictions from using the first and second versus difference of predictions from using the third and fourth is computed. Simply (pr1-pr2) - (pr3-pr4) with obvious notation. Useful to derive esoteric interaction effects.

Finally, only arguments Exp, vcov, alpha and sample from ci.lin are honored when ctr.mat is a data frame or a list of two data frames.

You can leave out variables (columns) from the two data frames that would be identical, basically variables not relevant for the calculation of the contrast. In many cases ci.lin (really Epi:::ci.dfr) can figure out the names of the omitted columns, but occasionally you will have to supply the names of the omitted variables in the xvars argument. Factors omitted need not be listed in xvars, although no harm is done doing so.

Wald computes a Wald test for a subset of (possibly linear combinations of) parameters being equal to the vector of null values as given by H0. 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 length 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 confidence 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.

`ci.pred` returns a 3-column matrix with estimates and upper and lower confidence intervals as columns. This is just a convenience wrapper for `predict.glm(obj,se.fit=TRUE)` which returns a rather unhandy structure. The prediction with c.i. is made in the `link` scale, and by default transformed by the inverse link, since the most common use for this is for multiplicative Poisson or binomial models with either log or logit link.

`ci.ratio` returns the rate-ratio of two independent sets of rates given with confidence intervals or s.e.s. If `se1` and `se2` are given and `log.tr=FALSE` it is assumed that `r1` and `r2` are rates and `se1` and `se2` are standard errors of the log-rates.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com & Michael Hills

See Also

See `ci.eta` for a simple version only needing coefficients and variance-covariance matrix. See also `ci.cum` for a function computing cumulative sums of (functions of) parameter estimates, and `ci.surv` for a function computing confidence intervals for survival functions based on smoothed rates. The example code for `matshade` has an application of predicting a rate-ratio using a list of two prediction frames in the `ctr.mat` argument.

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 )

# Suppose the x-effect differs across levels of g:
mi <- update( mm, . ~ . + g:x )
ci.lin( mi )
# RR a vs. b by x:
nda <- data.frame( x = -3:3, g = "a", f = "b" )
ndb <- data.frame( x = -3:3, g = "b", f = "b" )
#
ci.lin( mi, list(nda,ndb) )
# Same result if f column is omitted because "f" columns are identical
ci.lin( mi, list(nda[,-3],ndb[,-3]) )
# However, crashes if knots in spline is supplied, and non-factor omitted
```

```
xk <- -1:1
xi <- c(-0.5,0.5)
ww <- rnorm(200)
mi <- update( mm, . ~ . -x + ww + Ns(x,knots=xk) + g:Ns(x,knots=xi) )
# Will crash
try( cbind( nda$x, ci.lin( mi, list(nda,ndb) ) ) )
# Must specify num vars (not factors) omitted from nda, ndb
cbind( nda$x, ci.lin( mi, list(nda,ndb), xvars="ww" ) )

# A Wald test of whether 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 )

# Confidence intervals for ratio of rates
# Rates and ci supplied, but only the range (lower and upper ci) is used
ci.ratio( cbind(10,8,12.5), cbind(5,4,6.25) )
ci.ratio( cbind(8,12.5), cbind(4,6.25) )

# Beware of the offset when making predictions with ci.pred and ci.exp
## Not run:
library( mgcv )
data( mortDK )
m.arg  <- glm( dt ~ age , offset=log(risk) , family=poisson, data=mortDK )
m.form <- glm( dt ~ age + offset(log(risk)), family=poisson, data=mortDK )
a.arg  <- gam( dt ~ age , offset=log(risk) , family=poisson, data=mortDK )
a.form <- gam( dt ~ age + offset(log(risk)), family=poisson, data=mortDK )

nd <- data.frame( age=60:65, risk=100 )
round( ci.pred( m.arg , nd ), 4 )
round( ci.pred( m.form, nd ), 4 )
round( ci.exp ( m.arg , nd ), 4 )
round( ci.exp ( m.form, nd ), 4 )
round( ci.pred( a.arg , nd ), 4 )
round( ci.pred( a.form, nd ), 4 )
round( ci.exp ( a.arg , nd ), 4 )
round( ci.exp ( a.form, nd ), 4 )

nd <- data.frame( age=60:65 )
try( ci.pred( m.arg , nd ) )
try( ci.pred( m.form, nd ) )
try( ci.exp ( m.arg , nd ) )
try( ci.exp ( m.form, nd ) )
try( ci.pred( a.arg , nd ) )
try( ci.pred( a.form, nd ) )
try( ci.exp ( a.arg , nd ) )
try( ci.exp ( a.form, nd ) )

## End(Not run)
```

`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://bendixcarstensen.com

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 tolerance. 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 matrix. 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 homogeneous 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. Strata that are homogeneous with respect to either the outcome variable or the predictor variables are uninformative, in the sense that they can be removed without modifying the estimates or standard errors. 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)
```

---

crr.Lexis *Fit a competing risks regression model (Fine-Gray model) using a Lexis object)*

Description

Fits a competing risks regression model using a Lexis object assuming that every person enters at time 0 and exits at time lex.dur. Thus is only meaningful for Lexis objects with one record per person, (so far).

Usage

```
crr.Lexis( obj, mod, quiet=FALSE, ...)
```

Arguments

obj	A Lexis object; variables in mod are taken from this.
mod	Formula, with the l.h.s. a character constant equal to a level of obj$lex.Xst, and the r.h.s. a model formula interpreted in obj.
quiet	Logical indicating whether a brief summary should be printed.
...	Further arguments passed on to crr.

Details

This function is a simple wrapper for crr, allowing a formula-specification of the model (which allows specifications of covariates on the fly), and utilizing the structure of Lexis objects to simplify specification of the outcome. Prints a summary of the levels used as event, competing events and censoring.

By the structure of the Lexis object it is not necessary to indicate what the censoring code or competing events are, that is automatically derived from the Lexis object.

Currently only one state is allowed as l.h.s. (response) in mod.

Value

A crr object (which is a list), with two extra elements in the list, model.Lexis - the model formula supplied, and transitions - a table of transitions and censorings showing which transition was analysed and which were taken as competing events.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

See Also

crr, Lexis

Examples

```
# Thorotrats patients, different histological types of liver cancer
# Load thorotrast data, and restrict to exposed
data(thoro)
tht <- thoro[thoro$contrast==1,]
# Define exitdate as the date of livercancer
tht$dox <- pmin( tht$liverdat, tht$exitdat, na.rm=TRUE )
tht <- subset( tht, dox > injecdat )
# Convert to calendar years in dates
tht <- cal.yr( tht )

# Set up a Lexis object with three subtypes of liver cancer and death
tht.L <- Lexis( entry = list( per = injecdat,
                              tfi = 0 ),
                exit = list( per = dox ),
          exit.status = factor( 1*hepcc+2*chola+3*hmang+
                                4*(hepcc+chola+hmang==0 & exitstat==1),
                                labels=c("No cancer","hepcc","chola","hmang","Dead") ),
                data = tht )
summary( tht.L )

# Show the transitions
boxes( tht.L, boxpos=list(x=c(20,rep(80,3),30),
                          y=c(60,90,60,30,10) ),
              show.BE="nz", scale.R=1000 )

# Fit a model for the Hepatocellular Carcinoma as outcome
# - note that you can create a variable on the fly:
library( cmprsk )
hepcc <- crr.Lexis( tht.L, "hepcc" ~ volume + I(injecdat-1940) )
hepcc$model.Lexis
hepcc$transitions

# Models for the three other outcomes:
chola <- crr.Lexis( tht.L, "chola" ~ volume + I(injecdat-1940) )
hmang <- crr.Lexis( tht.L, "hmang" ~ volume + I(injecdat-1940) )
dead  <- crr.Lexis( tht.L, "Dead"  ~ volume + I(injecdat-1940) )
```

```
# Compare the effects
# NOTE: This is not necessarily a joint model for all transitions.
zz <- rbind( ci.exp(hepcc),
             ci.exp(chola),
             ci.exp(hmang),
             ci.exp(dead) )
zz <- cbind( zz[c(1,3,5,7)  ,],
             zz[c(1,3,5,7)+1,] )
rownames( zz ) <- c("hepcc","chola","hmang","dead")
colnames( zz )[c(1,4)] <- rownames( ci.exp(chola) )
round( zz, 3 )
```

`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 = transient(data),
                          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. Any record representing follow up after the cutpoint has its value of lex.Cst updated to the new state. An extra time-scale is added; the time since the event at cut. This time scale will be NA for any follow-up prior to the intermediate event.

The function tsNA20 will replace all missing values in timescales with 0. This is commonly meeded when timescales defined as time since entry into an intermediate state are used in modeling. But you do not want to do that permanently in the cut data frame.

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, adds a new timescale and updates the attribute "time.since". This attribute is a character vector of the same length as the "time.scales" attribute, whose value is '""' if the corresponding timescale is defined for any piece of follow-up, and if the corresponding time scale is defined by say cutLexis(obj,new.state="A",new.scale=TRUE), it has the value "A".

Author(s)

Bendix Carstensen, Steno Diabetes Center, <b@bxc.dk>, Martyn Plummer, <martyn.plummer@r-project.org>

See Also

mcutLexis, rcutLexis, xcutLexis, addCov.Lexis, addDrug.Lexis, splitLexis, Lexis, summary.Lexis, timeSince, boxes.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
str(xCs)
```

`detrend` *Projection of a model matrix on the orthogonal complement of a trend or curvature.*

Description

The columns of a model matrix `M` is projected on the orthogonal complement to the matrix `(1,t)`, resp. `(1,t,t^2)`.

Orthogonality is w.r.t. an inner product defined by the positive definite matrix matrix `diag(weight)`. Non-diagonal matrices defining the inner product is not supported.

Usage

```
detrend( M, t, weight = rep(1, nrow(M)) )
decurve( 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. Must be all non-negative.

Details

The functions are intended to be used in construction of particular parametrizations of age-period-cohort models.

Value

`detrend` returns full-rank matrix with columns orthogonal to `(1,t)`; `decurve` returns full-rank matrix with columns orthogonal to `(1,t,t^2)`.

Author(s)

Bendix Carstensen, Steno Diabetes Center Copenhagen, http://bendixcarstensen.com, with essential 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 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 (10 g/day), a numeric vector
fibre:	dietary fibre intake (10 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)
```

DMepi *Epidemiological rates for diabetes in Denmark 1996–2015*

Description

Register based counts and person-years for incidence of diabetes and mortality with and without diabetes.

Usage

```
data("DMepi")
```

Format

A data frame with 4200 observations on the following 8 variables.

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

`A` Age class, 0–99

`P` Calendar year, 1996–2016

`X` Number of new diagnoses of diabetes among persons without diabetes

`D.nD` Number of deaths among persons without diabetes

`Y.nD` Person-years among persons without diabetes

`D.DM` Number of deaths among persons with diabetes

`Y.DM` Person-years among persons with diabetes

Details

Based on registers of the Danish population. Only included for illustrative purposes. Cannot be used as scientifically validated data, since small numbers are randomly permuted between units.

Examples

```
data(DMepi)
# Total deaths and person-years in the Danish population
ftable( addmargins( xtabs( cbind( Deaths=D.nD+D.DM,
                                  PYrs=Y.nD+Y.DM ) ~ P + sex,
                           data=DMepi ),
                    2 ),
        row.vars = 1 )
# Deaths and person-years in the population of diabetes patients
round(
ftable( addmargins( xtabs( cbind( Deaths=D.DM,
                                  PYrs=Y.DM ) ~ P + sex,
                           data=DMepi ),
                    2 ),
        row.vars = 1 ) )
```

```
# Model for age-specific incidence rates;
inc <- glm( X ~ sex + Ns( A, knots=seq(30,80,10) ) + P,
                offset = log(Y.nD),
                family = poisson,
                  data = DMepi )

# Predict for men and women separately in 2010:
ndm <- data.frame( sex="M", A=20:90, P=2010, Y.nD=1000 )
ndf <- data.frame( sex="F", A=20:90, P=2010, Y.nD=1000 )
prM <- ci.pred( inc, ndm )
prF <- ci.pred( inc, ndf )
matplot( ndm$A, cbind(prM,prF),
         type="l", lty=1, lwd=c(3,1,1),
         col=rep(c("blue","red"),each=3),
         log="y", xlab="Age", ylab="DM incidence per 1000 PY" )

# This is a proportional hazards model - add sex-age interaction
int <- update( inc, . ~ . + sex:Ns( A, knots=seq(30,80,10) ) )
prM <- ci.pred( int, ndm )
prF <- ci.pred( int, ndf )
matplot( ndm$A, cbind(prM,prF),
         type="l", lty=1, lwd=c(3,1,1),
         col=rep(c("blue","red"),each=3),
         log="y", xlab="Age", ylab="DM incidence per 1000 PY" )

# The rate-ratio is teased out using the ci.exp:
RRp <- ci.exp( inc, list(ndm,ndf) )
RRi <- ci.exp( int, list(ndm,ndf) )

# and added to the plot
matlines( ndm$A, cbind(RRi,RRp),
          type="l", lty=1, lwd=c(3,1,1), col=gray(rep(c(0.3,0.7),each=3)) )
abline(h=1)
axis(side=4)
mtext( "Male/Female IRR", side=4, line=2 )
```

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. All dates are radomly jittered by adding a U(-7,7) (days).

Usage

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

Format

A data frame with 10000 observations on the following 7 variables.

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

`dobth` Date of birth

`dodm` Date of inclusion in the register

`dodth` Date of death

`dooad` Date of 2nd prescription of OAD

`doins` Date of 2nd insulin prescription

`dox` Date of exit from follow-up.

Details

All dates are given in fractions of years, so 1998.000 corresponds to 1 January 1998 and 1998.997 to 31 December 1998.

All dates are randomly perturbed by a small amount, so no real persons have any of the combinations of the 6 dates in the dataset. But results derived from the data will be quite close to those that would be obtained if the entire 'real' diabetes register were used.

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 )

# Cut the follow-up at insulin start, and introduce a new timescale,
# and split non-precursor states
system.time(
dmi <- cutLexis( dml, cut = dml$doins,
                      pre = "DM",
                new.state = "Ins",
                new.scale = "t.Ins",
             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,
                 eff = NULL,
       alpha = 0.05,
        base = 1,
      digits = 3,
        data = NULL )
```

Arguments

response	The response variable - must be numeric or logical. If logical, TRUE is considered the outcome.
type	The type of responsetype - 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) (confounders) - these are passed as a list if there are more than one.
weights	Frequency weights for binary response only
eff	How should effects be measured. If response is binomial, the default is "OR" (odds-ratio) with "RR" (relative risk) as an option. If response is failure, the default is "RR" (rate-ratio) with "RD" (rate difference) as an option.
base	Baseline for the effects of a categorical exposure, either a number or a name of the level. Defaults to 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/relative risks for a binary response, and rate ratios/rate differences 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 (*1934-Jun-07, +2021-Jan-07)

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)

print( options('na.action') )
# 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)
effx(lowbw,type="binary",exposure=hyp,eff="RR",data=births)
```

`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)
```

`entry.Lexis` *Time series and other methods for Lexis objects*

Description

Extract the entry time, exit time, status or duration of follow-up from a `Lexis` object. Classify states.

Usage

```
     entry(x, time.scale = NULL, by.id=FALSE)
      exit(x, time.scale = NULL, by.id=FALSE)
    status(x, at="exit"        , by.id=FALSE)
       dur(x,                    by.id=FALSE)
 transient(x)
 absorbing(x)
 preceding(x, states)
    before(x, states)
succeeding(x, states)
     after(x, states)
```

Arguments

x	an object of class Lexis.
time.scale	a string or integer indicating the time scale. If omitted, all times scales are used.
by.id	Logical, if TRUE, only one record per unique value of lex.id is returned; either the first, the last, or for dur, the sum of lex.dur. If TRUE, the returned object have the lex.id as (row)names attribute.
at	string indicating the time point(s) at which status is to be measured. Possible values are "exit" or "entry".
states	Character vector of states.

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, a matrix is returned.

The status function returns a vector giving the status at "at" (either 'entry' or 'exit') and dur returns a vector with the lengths of the follow-up intervals.

entry, exit, status and dur return vectors of length nrow(x) if by.id=FALSE; if by.id=TRUE a vector of length length(unique(lex.id)).

The functions transient and absorbing return character vectors of the transient, resp. absorbing states in x. These are necessarily disjoint but the union may be a proper subset of levels(x), since the latter may have levels that are never assumed by either lex.Cst or lex.Xst.

preceding returns a character vector with names of the states of the Lexis object x from which one of the states in states can be reached directly - the preceding states. before is just a synonym for preceding.

succeeding returns a character vector with names of the states of the Lexis object x that can be reached directly from one of the states in states. after is just a synonym for succeeding.

Author(s)

Martyn Plummer & Bendix Carstensen

See Also

Lexis

Epi — *Epi: Functions for manipulation and statistical analysis of epidemiological data*

Description

Epi has grown out of the course 'Statistical Practise in Epidemiology with R' http://bendixcarstensen.com/SPE/.

The major contributions from this course have been the stat.table function for advanced tabulation and summary, and the functions for representation and the Lexis function(s) for manipulation of multistate data with multiple time scales.

Details

Click on the Index link below the line to access vignettes (tutorial documents) and an alphabetic list of the functions in Epi.

---

erl *Compute survival functions from rates and expected residual lifetime in an illness-death model as well as years of life lost to disease.*

Description

These functions compute survival functions from a set of mortality and disease incidence rates in an illness-death model. Expected residual life time can be computed under various scenarios by the erl function, and areas between survival functions can be computed under various scenarios by the yll function. Rates are assumed supplied for equidistant intervals of length int.

Usage

```
surv1( int, mu ,                age.in = 0, A = NULL )
 erl1( int, mu ,                age.in = 0 )
surv2( int, muW, muD, lam,      age.in = 0, A = NULL )
  erl( int, muW, muD, lam=NULL, age.in = 0, A = NULL,
       immune = is.null(lam), yll=TRUE, note=TRUE )
  yll( int, muW, muD, lam=NULL, age.in = 0, A = NULL,
       immune = is.null(lam), note=TRUE )
```

Arguments

int	Scalar. Length of intervals that rates refer to.
mu	Numeric vector of mortality rates at midpoints of intervals of length int
muW	Numeric vector of mortality rates among persons in the "Well" state at midpoints of intervals of length int. Left endpoint of first interval is age.in.
muD	Numeric vector of mortality rates among persons in the "Diseased" state at midpoints of intervals of length int. Left endpoint of first interval is age.in.
lam	Numeric vector of disease incidence rates among persons in the "Well" state at midpoints of intervals of length int. Left endpoint of first interval is age.in.
age.in	Scalar indicating the age at the left endpoint of the first interval.
A	Numeric vector of conditioning ages for calculation of survival functions.
immune	Logical. Should the years of life lost to the disease be computed using assumptions that non-diseased individuals are immune to the disease (lam=0) and that their mortality is yet still muW.
note	Logical. Should a warning of silly assumptions be printed?
yll	Logical. Should years of life lost be included in the result?

Details

The mortality rates given are supposed to refer to the ages `age.in+(i-1/2)*int`, `i=1,2,3,....`

The units in which `int` is given must correspond to the units in which the rates `mu`, `muW`, `muD` and `lam` are given. Thus if `int` is given in years, the rates must be given in the unit of events per year.

The ages in which the survival curves are computed are from `age.in` and then at the end of `length(muW)` (`length(mu)`) intervals each of length `int`.

The `age.in` argument is merely a device to account for rates only available from a given age. It has two effects, one is that labeling of the interval endpoint is offset by this quantity, thus starting at `age.in`, and the other that the conditioning ages given in the argument `A` will refer to the ages defined by this.

The `immune` argument is `FALSE` whenever the disease incidence rates are supplied. If set to `TRUE`, the years of life lost is computed under the assumption that individuals without the disease at a given age are immune to the disease in the sense that the disease incidence rate is 0, so transitions to the diseased state (with presumably higher mortality rates) are assumed not to occur. This is a slightly peculiar assumption (but presumably the most used in the epidemiological literature) and the resulting object is therefore given an attribute, `NOTE`, that point this out.

If however `muW` is the total mortality in the population (including the diseased) the result is a good approximation to the correct YLL.

The default of the `surv2` function is to take the possibility of disease into account.

Value

`surv1` and `surv2` return a matrix whose first column is the ages at the ends of the intervals, thus with `length(mu)+1` rows. The following columns are the survival functions (since `age.in`), and conditional on survival till ages as indicated in `A`, thus a matrix with `length(A)+2` columns. Columns are labeled with the actual conditioning ages; if `A` contains values that are not among the endpoints of the intervals used, the nearest smaller interval border is used as conditioning age, and columns are named accordingly.

`surv1` returns the survival function for a simple model with one type of death, occurring at intensity `mu`.

`surv2` returns the survival function for a person in the "Well" state of an illness-death model, taking into account that the person may move to the "Diseased" state, thus requiring all three transition rates to be specified. The conditional survival functions are conditional on being in the "Well" state at ages given in `A`.

`erl1` returns a three column matrix with columns `age`, `surv` (survival function) and `erl` (expected residual life time) with `length(mu)+1` rows.

`erl` returns a two column matrix, columns labeled "Well" and "Dis", and with row-labels `A`. The entries are the expected residual life times given survival to `A`. If `yll=TRUE` the difference between the columns is added as a third column, labeled "YLL".

Author(s)

Bendix Carstensen, <b@bxc.dk>

See Also

ci.cum

Examples

```
library( Epi )
data( DMlate )
# Naive Lexis object
Lx <- Lexis( entry = list( age = dodm-dobth ),
             exit = list( age = dox -dobth ),
       exit.status = factor( !is.na(dodth), labels=c("DM","Dead") ),
             data = DMlate )
# Cut follow-up at insulin inception
Lc <- cutLexis( Lx, cut = Lx$doins-Lx$dob,
             new.state = "DM/ins",
       precursor.states = "DM" )
summary( Lc )
# Split in small age intervals
Sc <- splitLexis( Lc, breaks=seq(0,120,2) )
summary( Sc )

# Overview of object
boxes( Sc, boxpos=TRUE, show.BE=TRUE, scale.R=100 )

# Knots for splines
a.kn <- 2:9*10

# Mortality among DM
mW <- glm( lex.Xst=="Dead" ~ Ns( age, knots=a.kn ),
           offset = log(lex.dur),
           family = poisson,
             data = subset(Sc,lex.Cst=="DM") )

# Mortality among insulin treated
mI <- update( mW, data = subset(Sc,lex.Cst=="DM/ins") )

# Total motality
mT <- update( mW, data = Sc )

# Incidence of insulin inception
lI <- update( mW, lex.Xst=="DM/ins" ~ . )

# From these we can now derive the fitted rates in intervals of 1 year's
# length. In real applications you would use much smaller interval like
# 1 month:
# int <- 1/12
int <- 1

# Prediction frame to return rates in units of cases per 1 year
# - we start at age 40 since rates of insulin inception are largely
# indeterminate before age 40
nd <- data.frame( age = seq( 40+int, 110, int ) - int/2,
```

```
              lex.dur = 1 )
muW <- predict( mW, newdata = nd, type = "response" )
muD <- predict( mI, newdata = nd, type = "response" )
lam <- predict( lI, newdata = nd, type = "response" )

# Compute the survival function, and the conditional from ages 50 resp. 70
s1 <- surv1( int, muD, age.in=40, A=c(50,70) )
round( s1, 3 )

s2 <- surv2( int, muW, muD, lam, age.in=40, A=c(50,70) )
round( s2, 3 )

# How much is YLL overrated by ignoring insulin incidence?
round( YLL <- cbind(
yll( int, muW, muD, lam, A = 41:90, age.in = 40 ),
yll( int, muW, muD, lam, A = 41:90, age.in = 40, immune=TRUE ) ), 2 )[seq(1,51,10),]

par( mar=c(3,3,1,1), mgp=c(3,1,0)/1.6, bty="n", las=1 )
matplot( 40:90, YLL,
         type="l", lty=1, lwd=3,
         ylim=c(0,10), yaxs="i", xlab="Age" )
```

`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 accommodate 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, <martyn.plummer@r-project.org>

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 additive 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, <martyn.plummer@r-project.org>

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, <martyn.plummer@r-project.org>

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, <martyn.plummer@r-project.org>, Bendix Carstensen, <b@bxc.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.
`...`	Optional arguments passed to the `vcov()` method for the fitted model object.

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`

`foreign.Lexis` *Create a data structures suitable for use with packages* `mstate` *or* `etm`.

Description

The `mstate` package requires input in the form of a stacked dataset with specific variable names. This is provided by `msdata.Lexis`. The resulting dataframe contains the same information as the result of a call to `stack.Lexis`.

The `etm` package requires input (almost) in the form of a `Lexis` object, but with specific column names etc. This is provided by `etm.Lexis`.

Usage

```
msdata(obj, ...)
## S3 method for class 'Lexis'
msdata(obj,
                time.scale = timeScales(obj)[1],
                       ... )
## S3 method for class 'Lexis'
etm( data,
               time.scale = timeScales(data)[1],
                cens.name = "cens",
                        s = 0,
                        t = "last",
               covariance = TRUE,
                 delta.na = TRUE,
                      ... )
```

Arguments

`obj`	A `Lexis` object.
`data`	A `Lexis` object.
`time.scale`	Name or number of timescale in the `Lexis` object.
`cens.name`	Name of the code for censoring used by `etm`. It is only necessary to change this if one of the states in the `Lexis` object has name `"cens"`.
`s`	Passed on to `etm`.
`t`	Passed on to `etm`.
`covariance`	Passed on to `etm`.
`delta.na`	Passed on to `etm`.
`...`	Further arguments.

Value

msdata.Lexis returns 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.

etm.Lexis transforms the Lexis object into a dataframe suitable for analysis by the function etm from the etm package, and actually calls this function, so returns an object of class etm.

Author(s)

Bendix Carstensen, <b@bxc.dk>, http://bendixcarstensen.com

See Also

stack.Lexis, msprep, etm

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[1:1000,] )
dmi <- cutLexis( dml, cut=dml$doins, new.state="Ins", pre="DM" )
summary( dmi )

# Use the interface to the mstate package
if( require(mstate) )
{
ms.dmi <- msdata.Lexis( dmi )
# Check that all the transitions and person-years got across.
with( ms.dmi, rbind( table(status,trans),
                     tapply(Tstop-Tstart,trans,sum) ) )
}

# Use the etm package directly with a Lexis object
if( require(etm) )
{
dmi <- subset(dmi,lex.id<1000)
etm.D <- etm.Lexis( dmi, time.scale=3 )
str( etm.D )
plot( etm.D, col=rainbow(5), lwd=2, lty=1, xlab="DM duration" )
}
```

---

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

---

gen.exp *Generate covariates for drug-exposure follow-up from drug purchase records.*

Description

From records of drug purchase and possibly known treatment intensity, the time since first drug use and cumulative dose at prespecified times is computed. Optionally, lagged exposures are computed too, i.e. cumulative exposure a prespecified time ago.

Usage

```
gen.exp( purchase, id="id", dop="dop", amt="amt", dpt="dpt",
               fu, doe="doe", dox="dox",
           breaks,
          use.dpt = ( dpt %in% names(purchase) ),
         push.max = Inf,
          rm.dose = FALSE,
             lags = NULL,
          lag.dec = 1,
          lag.pre = "lag.",
         pred.win = Inf )
```

Arguments

purchase	Data frame with columns id-person id, dop - date of purchase, amt - amount purchased, and optionally dpt - (dose per time) ("defined daily dose", DDD, that is), how much is assumed to be ingested per unit time. The units used for dpt is assumed to be units of amt per units of dop.
id	Character. Name of the id variable in the data frame.
dop	Character. Name of the date of purchase variable in the data frame.
amt	Character. Name of the amount purchased variable in the data frame.
dpt	Character. Name of the dose-per-time variable in the data frame.
fu	Data frame with follow-up period for each person, the person id variable must have the same name as in the purchase data frame.
doe	Character. Name of the date of entry variable.
dox	Character. Name of the date of exit variable.
breaks	Numerical vector of dates at which the time since first exposure, cumulative dose etc. are computed.
use.dpt	Logical: should we use information on dose per time.

push.max	Numerical. How much can purchases maximally be pushed forward in time. See details.
rm.dose	Logical. Should the dose from omitted period of exposure (due to the setting of push.max) be ignored. If FALSE, the cumulative dose will be the cumulation of the actually purchased amounts, regardless of how far the inception dates have been pushed.
lags	Numerical vector of lag-times used in computing lagged cumulative doses.
lag.dec	How many decimals to use in the construction of names for the lagged exposure variables
lag.pre	Character string used for prefixing names of lagged exposure variables. Aimed to facilitate the use of gen.exp for different drugs with the aim of merging information.
pred.win	The length of the window used for constructing the average dose per time used to compute the duration of the last purchase. Only used when use.dpt=FALSE. The default value Inf corresponds to using the time between first and last purchase of drug as the interval for computing average consumption per time, and thus the termination of use.

Details

The intention of this function is to generate covariates for a particular drug for the entire follow-up of each person. The reason that the follow-up prior to first drug purchase and post-exposure is included is that the covariates must be defined for all follow-up for each person in order to be useful for analysis of disease outcomes.

The functionality is described in terms of calendar time as underlying time scale, because this will normally be the time scale for drug purchases and for entry and exit for persons. In principle the variables termed as dates might equally well refer to say the age scale, but this would then have to be true *both* for the purchase data, the follow-up data and the breaks argument.

Drug purchase records (in purchase) are used to construct measures of drug exposure at prespecified timepoints (in breaks) in follow-up intervals (in fu). Each person may have more than one follow-up interval. They should be disjoint, but this is not checked.

If use.dpt is TRUE then the dose per time information is used to compute the exposure interval associated with each purchase. Exposure intervals are stacked, that is each interval is put after any previous. This means that the start of exposure to a given purchase can be pushed into the future. The parameter push.max indicates the maximally tolerated push. If this is reached by a person, the assumption is that some of the purchased drug may not be counted in the exposure calculations — see rm.dose.

The dpt can either be a constant, basically translating each purchased amount into exposure time the same way for all persons, or it can be a vector with different treatment intensities for each purchase. In any case the cumulative dose is computed taking dpt into account, unless rm.dose is FALSE in which case the actual purchased amount is cumulated. The latter is slightly counter-intuitive because we are using the dpt to push the intervals, and then disregard it when computing the cumulative dose. The counter argument is that if the limit push.max is reached, the actual dosage may be larger than indicated the dpt, and is essentially what this allows for.

If use.dpt is FALSE then the exposure from one purchase is assumed to stretch over the time to the next purchase, so we are effectively allowing different dosing rates (dose per time) between

purchases. Formally this approach conditions on the future, because the rate of consumption (the accumulation of cumulative exposure) is computed based on knowledge of when next purchase is made. Moreover, with this approach, periods of non-exposure does not exist, except after the last purchase where the future consumption rate is taken to be the average over the period of use (or a period of length `pred.win`), and hence defines a date of cessation of drug.

Finally, if `use.dpt` is `FALSE`, at least two purchase records are required to compute the measures. Therefore persons with only one drug purchase record are ignored in calculations.

Value

A data frame with one record per person and follow-up date (`breaks`). Date of entry and date of exit are included too; but only follow-up in the intersetion of `range(breaks)` and `range(fu$doe,fu$dox)` is output.

`id` person id.

`dof` date of follow up, i.e. start of interval. Apart from possibly the first interval for each person, this will assume values in the set of the values in `breaks`. All other variables refer to status as of this date.

`dur` the length (duration) of interval.

`tfi` time from first initiation of drug.

`off` Logical, indicating whether the person is `off` drug. So it is `FALSE` if the person is exposed at `dof`.

`doff` date of latest transition to `off` drug. Note that tis defined also at dates after drug exposure has been resumed.

`tfc` time from latest cessation of drug.

`ctim` cumulative `time` on the drug.

`cdos` cumulative `dose`.

`ldos` suffixed with one value per element in `lags`, the latter giving the cumulative doses `lags` before `dof`.

Author(s)

Bendix Carstensen, <b@bxc.dk>. The development of this function was supported partly through a grant from the EFSD (European Foundation for the Study of Diabetes)

See Also

`Lexis`, `cutLexis`, `mcutLexis`, `addCov.Lexis`

Examples

```
# Example data for drug purchases in 3 persons --- dates (dop) are
# measured in years, amount purchased (amt) in no. pills and dose per
# time (dpt) consequently given in units of pills/year. Note we also
# include a person (id=4) with one purchase record only.
n <- c( 10, 18, 8, 1 )
hole <- rep(0,n[2])
```

```
hole[10] <- 2 # to create a hole of 2 years in purchase dates
# dates of drug purchase
dop <- c( 1995.278+cumsum(sample(1:4/10,n[1],replace=TRUE)),
          1992.351+cumsum(sample(1:4/10,n[2],replace=TRUE)+hole),
          1997.320+cumsum(sample(1:4/10,n[3],replace=TRUE)),
          1996.470 )
# purchased amounts mesured in no. pills
amt <- sample( 1:3*50 , sum(n), replace=TRUE )
# prescribed dosage therefore necessarily as pills per year
dpt <- sample( 4:1*365, sum(n), replace=TRUE )
# collect to purchase data frame
dfr <- data.frame( id = rep(1:4,n),
                   dop,
                   amt = amt,
                   dpt = dpt )
head( dfr, 3 )

# a simple dataframe for follow-up periods for these 4 persons
fu <- data.frame( id = 1:4,
                  doe = c(1995,1992,1996,1997)+1:4/4,
                  dox = c(2001,2003,2002,2010)+1:4/5 )
fu

# Note that the following use of gen.exp relies on the fact that the
# purchase dataframe dfr has variable names "id", "dop", "amt" and
# "dpt"" and the follow-up data frame fu has variable names "id",
# "doe" and "dox"

# 1: using the dosage information
dposx <- gen.exp( dfr,
                    fu = fu,
               use.dpt = TRUE,
                breaks = seq(1990,2015,0.5),
                  lags = 2:4/4,
               lag.pre = "l_" )
format( dposx, digits=5 )

# 2: ignoring the dosage information,
#    hence person 4 with only one purchase is omitted
xposx <- gen.exp( dfr,
                    fu = fu,
               use.dpt = FALSE,
                breaks = seq(1990,2015,0.5),
                  lags = 2:3/5 )
format( xposx, digits=5 )

# It is possible to have disjoint follow-up periods for the same person:
fu <- fu[c(1,2,2,3),]
fu$dox[2] <- 1996.2
fu$doe[3] <- 1998.3
fu

# Note that drug purchase information for the period not at risk *is* used
```

```
dposx <- gen.exp( dfr,
                  fu = fu,
              use.dpt = TRUE,
               breaks = seq(1990,2015,0.1),
                 lags = 2:4/4 )
format( dposx, digits=5 )
```

---

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)
```

---

harm *Create a basis of harmonic functions.*

Description

Returns a matrix of harmonic functions usable for modeling periodic effects

Usage

```
harm(x, ord=1, per=1, verbose=FALSE )
```

Arguments

x	A numeric variable.
ord	Integer, the order of the harmonic.
per	Numeric, the length of the period on the x scale.
verbose	Logical: shall I tell what I do with dates?

Details

Columns are constructed under the assumption that the periodic function has period per on the x scale. Thus, the first columns is defined as sin(2*pi*x/per), cos(2*pi*x/per), sin(4*pi*x/per) etc.

Since sin and cos are periodic functions there is no requirement that x be in any particular range.

Value

A matrix with nrow(x) rows and 2*deg columns and columnnames sin1, cos1, sin2, cos2 etc.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

Examples

```
x <- seq(-1,1,0.01)
head( harm(x,ord=2) )
matplot( x, harm(x,ord=2), type="l", lty=1, lwd=3 )
```

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)
## S3 method for class 'Icens'
summary(object, scale = 1, ...)
## S3 method for class 'Icens'
print(x, scale = 1, digits = 4, ...)
```

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 rate ratios / excess rates.
model.type	Which model should be fitted.
breaks	Breakpoints between intervals in which the underlying timescale effect 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 bootstrap 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.
object	an Icens object.
x	an Icens object.
scale	scaling factor for rates.
digits	how many digits is used for printing results.
...	Other parameters passed on.

Details

The model is fitted by calling either `fit.mult` or `fit.add`.

Value

An object of class `"Icens"`: a list with two or three components:

`rates`	A glm object from a binomial model with log-link, estimating the baseline rates, and the excess rates if `"AER"` is specified. The parameters in the linear predictor are rates, *not* log-rates. They are in units corresponding to (the inverse of) the units in which the arguments `first.well`, `last.well`, `first.ill` and `breaks` are given.
`cov`	A glm object from a binomial model with complementary log-log link, estimating the log-rate-ratios. Only if `"MRR"` is specified.
`niter`	Number 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 parameter estimates from the bootstrapping. Rows refer to parameters, columns to bootstrap samples.

Author(s)

Martyn Plummer, <martyn.plummer@r-project.org>, Bendix Carstensen, <b@bxc.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 and not per day
for(i in 2:4) hivDK[,i] <- cal.yr(hivDK[,i])

# multiplicative rate ratio model, baseline rates and RRs
m.RR <- Icens(entry, well, ill,
              model = "MRR",
            formula = ~ pyr + us,
             breaks = seq(1980, 1990, 5),
               data = hivDK)
# the MRR model returns a list with 2 glm objects.
```

```
round(ci.lin(m.RR$rates), 4)         # rates per 1 year
round(ci.lin(m.RR$cov, Exp=TRUE), 4) # RRs
# There is a print method that does the job:
print(m.RR)
# and scales only the rate parameters, here per 10 years
print(m.RR, scale = 10)

# additive excess rate model, baseline rates and excess rates
m.ER <- Icens(entry, well, ill,
              model = "AER",
            formula = ~ pyr + us,
             breaks = seq(1980, 1990, 5),
               data = hivDK)
# The print method returns rates per 1 year:
print(m.ER)
# or per 10 years:
print(m.ER, scale = 10)
```

`in.span` *Is* `x` *in the column span of matrix* `A` *and what columns are linearly dependent?*

Description

The function `in.span` checks if the vector `x` (or columns of the matrix `x`) is in the column span of the matrix `A`. If desired, it returns the coefficient matrix `B` so that `AB=x`. The function `thinCol` removes linearly dependent columns an returns a matrix of full rank.

Usage

```
in.span( A,
         x,
      coef = FALSE,
       tol = 1e-08 )
inSpan( A, x, coef=FALSE, tol=1e-08 )
id.span( A, B, tol=1e-08 )
idSpan( A, B, tol=1e-08 )
thinCol( A, tol = 1e-06, col.num = FALSE )
```

Arguments

`A`	A matrix.
`B`	A matrix.
`x`	A vector or matrix. `length(x)` (or `nrow(x)`) must be equal to `nrow(A)`.
`coef`	Logical. Should the coefficient matrix (`k`) be returned, so that `Ak=x`?
`tol`	Tolerance for identity of matrices in check (`in.span`) or QR decomposition (`thinCol`)

col.num Logical. Should the positions of dependent columns be returned instead of the full-rank matrix?

Details

thinCol is mainly a workhorse in detrend, but made available because of its general usefulness.

in.span and inSpan are just different names for the same to accommodate different naming schools.

in.span (inSpan) is handy in checking whether different parametrizations of a model are identical in the sense of spanning the same linear space. Equivalent to checking whether fitted values under different parametrizations are identical, but has the further use of checking if subspaces of models are equivalent. The function simply checks if the regression of (columns of) x on the columns of A produces residuals that are all 0.

id.span (equivalent to idSpan) checks whether two matrices have the same column span.

Value

in.span returns a logical: is x is in span(A)? If coef=TRUE it returns a matrix k so that Ak=x. k is not necessarily unique (A may not have full rank).

id.span returns a logical: is span(A) the same as span(B)?

thinCol returns a matrix of full rank, formed from A by deleting columns linearly dependent on other. If col.num=TRUE (one possible set of) positions of columns forming a full rank basis for the column space of A is returned.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com with essential help from Lars Jorge Diaz and Peter Dalgaard.

See Also

det

Examples

```
# Matrices and vectors, x in span(A), z (hopefully) not
A <- matrix(round(rnorm(15)*20),5,3)
B <- matrix(round(rnorm(15)*20),5,3)
B <- cbind( B, B%*%c(3,4,2) )
x <- A %*% c(3,4,2)
z <- 5:9

# how they look
data.frame( A=A, x=x, z=z, B=B )

# vectors in span(A)?
in.span(A,x)
in.span(x,A)
in.span(A,x,coef=TRUE)
```

```
in.span(A,z)
in.span(A,z,coef=TRUE)

# Do matrices span the same space ?
in.span( A, B )
in.span( B, A )

# B is not in span of a subspace of B columns, but vice versa
( M <- matrix( rnorm(8)*7, 4, 2 ) )
in.span( B%*%M, B )
in.span( B, B%*%M )
id.span( B, B%*%M )

# But not unique for singular matrices:
( xx <- in.span( B, B%*%M, coef=TRUE ) )
cbind( B%*%M, B%*%xx )
cbind( xx, M )

# Easier for full rank matrices:
( K <- matrix( rnorm(9)*7, 3, 3 ) )
in.span( A%*%K, A )
in.span( A, A%*%K )
id.span( A, A%*%K )
in.span( A, A%*%K, coef=TRUE )
```

`LCa.fit` *Fit Lee-Carter-type models for rates to arbitrarily shaped observations of rates in a Lexis diagram.*

Description

The Lee-Carter model is originally defined as a model for rates observed in A-sets (age by period) of a Lexis diagram, as log(rate(x,t)) = a(x) + b(x)k(t), using one parameter per age(x) and period(t). This function uses natural splines for a(), b() and k(), placing knots for each effect such that the number of events is the same between knots.

Usage

```
LCa.fit( data, A, P, D, Y,
         model = "APa",    # or one of "ACa", "APaC", "APCa" or "APaCa"
         a.ref,            # age reference for the interactions
        pi.ref = a.ref,    # age reference for the period interaction
        ci.ref = a.ref,    # age reference for the cohort interaction
         p.ref,            # period reference for the interaction
         c.ref,            # cohort reference for the interactions
          npar = c(a = 6,  # no. knots for main age-effect
                   p = 6,  # no. knots for period-effect
                   c = 6,  # no. knots for cohort-effect
                  pi = 6,  # no. knots for age in the period interaction
```

```
                  ci = 6), # no. knots for age in the cohort interaction
             VC = TRUE,     # numerical calculation of the Hessian?
          alpha = 0.05,     # 1 minus confidence level
            eps = 1e-6,     # convergence criterion
          maxit = 100,      # max. no iterations
          quiet = TRUE )    # cut the crap
## S3 method for class 'LCa'
print( x, ... )
## S3 method for class 'LCa'
summary( object, show.est=FALSE, ... )
## S3 method for class 'LCa'
plot( x, ... )
## S3 method for class 'LCa'
predict( object, newdata,
                        alpha = 0.05,
                        level = 1-alpha,
                          sim = ( "vcov" %in% names(object) ),
                          ... )
```

Arguments

data	A data frame. Must have columns A(age), P(period, that is calendar time), D(no. of events) and Y(person-time, exposure). Alternatively these four quantities can be given as separate vectors:
A	Vector of ages (midpoint of observation).
P	Vector of period (midpoint of observation).
D	Vector of no. of events.
Y	Vector of person-time. Demographers would say "exposure", bewildering epidemiologists.
a.ref	Reference age for the age-interaction term(s) pi(x) and/or pi(x), where pi(a.ref)=1 and ci(a.ref)=1.
pi.ref	Same, but specifically for the interaction with period.
ci.ref	Same, but specifically for the interaction with cohort.
p.ref	Reference period for the time-interaction term kp(t) where kp(p.ref)=0.
c.ref	Reference period for the time-interaction term kp(t) where kc(c.ref)=0.
model	Character, either "APa" which is the classical Lee-Carter model for log-rates, other possibilities are "ACa", "APCa", "APaC" or "APaCa", see details.
npar	A (possibly named) vector or list, with either the number of knots or the actual vectors of knots for each term. If unnamed, components are taken to be in the order (a,b,t), if the model is "APaCa" in the order (a,p,c,pi,ci). If a vector, the three integers indicate the number of knots for each term; these will be placed so that there is an equal number of events (D) between each, and half as many below the first and above the last knot. If npar is a list of scalars the behavior is the same. If npar is a list of vectors, these are taken as the knots for the natural splines. See details for naming convention.

VC	Logical. Should the variance-covariance matrix of the parameters be computed by numerical differentiation? See details.
alpha	1 minus the confidence level used when calculating confidence intervals for estimates in LCa.fit and for predictions by predict.LCa.
eps	Convergence criterion for the deviance, we use the the relative difference between deviance from the two models fitted.
maxit	Maximal number of iterations.
quiet	Shall I shut up or talk extensively to you about iteration progression etc.?
object	An LCa object, see under "Value".
show.est	Logical. Should the estimates be printed?
x	An LCa object, see under "Value".
newdata	Prediction data frame, must have columns A and P. Any Y column is ignored, predictions are given in units of the Y supplied for the call that generated the LCa object.
level	Confidence level.
sim	Logical or numeric. If TRUE, prediction c.i.s will be based on 1000 simulations from the posterior parameters. If numeric, it will be based on that number of simulations.
...	Additional parameters passed on to the method.

Details

The Lee-Carter model is non-linear in age and time so does not fit in the classical glm-Poisson framework. But for fixed b(x) it is a glm, and also for fixed a(x), k(t). The function alternately fits the two versions until the same fit is produced (same deviance).

The multiplicative age by period term could equally well have been a multiplicative age by cohort or even both. Thus the most extensive model has 5 continuous functions:

$$\log(\lambda(a,p)) = f(a) + b_p(a)k_p(p) + b_c(a)k_c(p-a)$$

Each of these is fitted by a natural spline, with knots placed at the quantiles of the events along the age (a), calendar time (p) respective cohort (p-a) scales. Alternatively the knots can be specified explicitly in the argument npar as a named list, where a refers to $f(a)$, p refers to $k_p(p)$, c refers to $k_c(p-a)$, pi (period interaction) refers to $b_p(a)$ and ci (cohort interaction) refers to $b_c(p-a)$.

The naming convention for the models is a capital P and/or C if the effect is in the model followed by a lower case a if there is an interaction with age. Thus there are 5 different models that can be fitted: APa, ACa, APaC APCa and APaCa.

The standard errors of the parameters from the two separate model fits in the iterations are however wrong; they are conditional on a subset of the parameters having a fixed value. However, analytic calculation of the Hessian is a bit of a nightmare, so this is done numerically using the hessian function from the numDeriv package if VC=TRUE.

The coefficients and the variance-covariance matrix of these are used in predict.LCa for a parametric bootstrap (that is, a simulation from a multivariate normal with mean equal to the parameter

estimates and variance as the estimated variance-covariance) to get confidence intervals for the predictions if sim is TRUE — which it is by default if they are part of the object.

The plot for LCa objects merely produces between 3 and 5 panels showing each of the terms in the model. These are mainly for preliminary inspection; real reporting of the effects should use proper relative scaling of the effects.

Value

LCa.fit returns an object of class LCa (smooth effects Lee-Carter model); it is a list with the following components:

model	Character, either APa, ACa, APaC, APCa or APaCa, indicating the variable(s) interacting with age.
ax	3-column matrix of age-effects, c.i. from the age-time model. Row names are the unique occurring ages in the dataset. Estimates are rates.
pi	3-column matrix of age-period interaction effects, c.i. from the age model. Row names are the actually occurring ages in the dataset. Estimates are multipliers of the log-RRs in kp, centered at 1 at pi.ref.
kp	3-column matrix of period-effects, with c.i.s from the age-time model. Row names are the actually occurring times in the dataset. Estimates are rate-ratios centered at 1 at p.ref.
ci	3-column matrix of age-cohort interaction effects, c.i. from the age model. Row names are the actually occurring ages in the dataset. Estimates are multipliers of the log-RRs in kc, centered at 1 at ci.ref.
kc	3-column matrix of cohort-effects, with c.i.s from the age-time model. Row names are the actually occurring times in the dataset. Estimates are rate-ratios centered at 1 at c.ref.
mod.at	glm object with the final age-time model — estimates the terms ax, kp, kc. Gives the same fit as the mod.b model after convergence.
mod.b	glm object with the final age model — estimates the terms pi, ci. Gives the same fit as the mod.at model after convergence.
coef	All coefficients from both models, in the order ax, kp, kc, pi, ci. Only present if LCa.fit were called with VC=TRUE (the default).
vcov	Variance-covariance matrix of coefficients from both models, in the same order as in the coef. Only present if LCa.fit were called with VC=TRUE.
knots	List of vectors of knots used in for the age, period and cohort effects.
refs	List of reference points used for the age, period and cohort terms in the interactions.
deviance	Deviance of the model
df.residual	Residual degrees of freedom
iter	Number of iterations used to reach convergence.

plot.LCa plots the estimated effects in separate panels, using a log-scale for the baseline rates (ax) and the time-RR (kt). For the APaCa model 5 panels are plotted.

summary.LCa returns (invisibly) a matrix with the parameters from the models and a column of the conditional se.s and additionally of the se.s derived from the numerically computed Hessian (if LCa.fit were called with VC=TRUE.)

predict.LCa returns a matrix with one row per row in newdata. If LCa.fit were called with VC=TRUE there will be 3 columns, namely prediction (1st column) and c.i.s based on a simulation of parameters from a multivariate normal with mean coef and variance vcov using the median and alpha/2 quantiles from the sim simulations. If LCa.fit were called with VC=FALSE there will be 6 columns, namely estimates and c.i.s from age-time model (mod.at), and from the age-interaction model (mod.b), both using conditional variances, and therefore likely with too narrow confidence limits.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

This function was conceived while teaching a course on APC models at the Max Planck Institute of Demographic Research (MPIDR, https://www.demogr.mpg.de/en/) in Rostock in May 2016 (http://bendixcarstensen.com/APC/MPIDR-2016/), and finished during a week long research stay there, kindly sponsored by the MPIDR.

See Also

apc.fit, apc.LCa, lca

Examples

```
library( Epi )
# Load the testis cancer data by Lexis triangles
data( testisDK )
tc <- subset( testisDK, A>14 & A<60 )
head( tc )

# We want to see rates per 100,000 PY
tc$Y <- tc$Y / 10^5

# Fit the Lee-Carter model with age-period interaction (default)
LCa.tc <- LCa.fit( tc, model="ACa", a.ref=30, p.ref=1980, quiet=FALSE, eps=10e-4, maxit=50 )

LCa.tc
summary( LCa.tc )

# Inspect what we got
names( LCa.tc )

# show the estimated effects
par( mfrow=c(1,3) )
plot( LCa.tc )

# Prediction data frame for ages 15 to 60 for two time points:
nd <- data.frame( A=15:60 )
# LCa predictions
p70 <- predict.LCa( LCa.tc, newdata=cbind(nd,P=1970), sim=1000 )
```

```
p90 <- predict.LCa( LCa.tc, newdata=cbind(nd,P=1990), sim=1000 )

# Inspect the curves from the parametric bootstrap (simulation):
par( mfrow=c(1,1) )
head( cbind(p70,p90) )
matplot( nd$A, cbind(p70,p90), type="l", lwd=c(6,3,3), lty=c(1,3,3),
         col=rep( 2:3, each=3 ), log="y",
         ylab="Testis cancer incidence per 100,000 PY in 1970 resp. 1990", xlab="Age" )
```

`legendbox` *Draw a box with text explaining the numbers in and between boxes from* `boxes.MS` *and* `boxes.Lexis`

Description

When drawing boxes describing a multistate model a legend explaining the numbers in the plot is required. `legendbox` does this.

Usage

```
legendbox(x, y,
          state = "State",
             py = "Person-time",
          begin = "no. begin",
            end = "no. end",
          trans = "Transitions",
          rates = "\n(Rate)",
           font = 1,
          right = !left,
           left = !right,
            ...)
```

Arguments

`x`	x-coordinate of the center of the box.
`y`	y-coordinate of the center of the box.
`state`	Text describing the state
`py`	Text describing the risk time
`begin`	Text describing the no. persons starting FU in state
`end`	Text describing the no. persons ending FU in state
`trans`	Text describing the no. of transitions
`rates`	Text describing the rates
`font`	Font to use for the text
`right`	Should a text describing arrow texts be on the r.h.s. of the box? Defaults to TRUE.
`left`	Should a text describing arrow texts be on the l.h.s. of the box?
`...`	Arguments passed on to `tbox`

Details

The function is called for its side effect of adding an explanatory box to the plot. If `right` is true, an explanation of events and rates are added to the right of the box. Similarly for `left`. It is admissible that `left == right`.

Value

None.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

See Also

`boxes.Lexis`

`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 of follow-up*

Description

Create an object of class `Lexis` to represent follow-up in multiple states on multiple time scales.

Usage

```
Lexis( entry,
        exit,
    duration,
entry.status = 0,
 exit.status = 0,
          id,
        data,
       merge = TRUE,
      states,
       notes = TRUE,
         tol = .Machine$double.eps^0.5,
keep.dropped = FALSE)
## S3 method for class 'Lexis'
print(x, ...,
                      td = 2,
                      nd = td,
                    rnam = FALSE,
                     org = FALSE)
```

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. The name of the elements of the list will appear as names of variables designated as timescales in the resulting object. All time scales must have the same units (e.g. years). The names of the timescales must be different from any column name in `data`.
`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 person represented in the Lexis object. Defaults to `1:nrow(data)`
`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 equal to states.
notes	Logical. Should notes on entry states and time be given.
tol	Numerical tolerance for follow-up time. Rows with duration less than this value are automatically dropped.
keep.dropped	Logical. Should dropped rows from data be saved as an attribute with the object for inspection?
x	A Lexis object.
td	Number of digits after the decimal separator used for timescales and lex.dur when printing
nd	Number of digits after the decimal separator used for other numerical variables in the Lexis object.
rnam	Logical, should row names be printed?
org	Logical, should columns be printed in the original order?
...	Other parameters passed on to print.data.frame.

Details

The analysis of long-term population-based follow-up studies typically requires multiple time scales to be taken into account, such as age, calendar 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 in the book "Einleitung in die Theorie der Bevolkerungsstatistik" from 1875.

The Lexis function can create 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. The latter is the default.

The print method prints the time-scale variables and other numerical variables rounded, possibly differently. Reorders columns so the Lexis-specific variables comes first. Returns (invisibly) a character vector with the (re)ordering of the columns in the object, even if org = TRUE is set.

There are also merge, subset, transform and many other 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 (with names equal to the union of the names of entry and exit), and additional columns with the following names:

lex.id	Identification of the persons.

`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` (the default) 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.

`entry`, `exit` must be numeric, using `Date` variables will cause some of the utilities 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 a character or 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 `exit.status` is numeric, the value of `entry.status` set to 0.

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 set of levels, namely the union of the levels of `entry.status` and `exit.status`.

Author(s)

Martyn Plummer with contributions from Bendix Carstensen

See Also

`plot.Lexis`, `splitLexis`, `cutLexis`, `mcutLexis`, `rcutLexis`, `addCov.Lexis`, `merge.Lexis`, `subset.Lexis`, `cbind.Lexis`, `rbind.Lexis`, `transform.Lexis`, `summary.Lexis`, `unLexis`, `timeScales`, `timeBand`, `entry`, `exit`, `transient`, `absorbing`, `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)
# xcoh <- cal.yr(xcoh, format="%d/%m/%Y", wh=2:4)

# See how it looks
xcoh
str( xcoh )

# Define a 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)

# Using character states may have undesired effects:
xcoh$Fail <- c("Dead","Well","Dead")
xcoh
L1 <- Lexis(entry = list(per = entry),
             exit = list(per = exit,
                         age = exit - birth),
      exit.status = Fail,
             data = xcoh)
L1
# people start being dead!

# ...unless you order the levels sensibly
xcoh$Fail <- factor(xcoh$Fail, levels = c("Well", "Dead"))
L2 <- Lexis(entry = list(per = entry),
             exit = list(per = exit,
                         age = exit - birth),
      exit.status = Fail,
             data = xcoh)
L2
# behaviour of print method:
L2[,1:6]
L2[,6:1]
print(L2[,6:1], org=TRUE)
(print(L2[,-3]))
```

`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://bendixcarstensen.com

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://bendixcarstensen.com

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 )
```

Lexis2msm *Convert a Lexis obejct to a data set suitable for input to the* msm:msm *function.*

Description

The number of records in the resulting dataset will have a number of records that is normally nrec(Lx) + nid(Lx), that is one extra record for each person. If there are 'holes' in persons' follow-up, each hole will also generate an extra record in the result.

Usage

```
Lexis2msm(Lx,
       state = "state",
     verbose = FALSE)
```

Arguments

Lx	A Lexis object.
state	Character; the name of the state variable in the result.
verbose	If true, you will be reminded what the function did.

Value

A data frame of class msmLexis with the timescales preserved and lex.id preserved but with other lex. variables removed.

Has more records than the original Lexis object

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

See Also

Lexis

Examples

```
example(mcutLexis)
# we now have the Lexis object L3:
summary(L3)

# data frame for use with msm
msm3 <- Lexis2msm(L3)

# see the difference
subset(  L3, lex.id %in% 1:3)
subset(msm3, lex.id %in% 1:3)
timeScales(msm3)
```

LexisScarf *Generate a knitting pattern for a tube-shaped scarf with Lexis lines*

Description

The function generates a dotted pattern that refers to stitches of a knitting pattern. Since there are several colors involved the back side of the knit becomes quite messy, hence the pattern is for a tube-shaped scarf, that will have all the messiness on the inside. So use a circular needle for the work. And send the author a picture of your finished scarf.

Usage

```
LexisScarf(nw = 4,   # fields around (width of scarf)
           nl = 20,  # fields along  (length)
           nm = 23,  # no. stitches per field
           np = 30,  # no. persons
        scale = 1.5, # maximal FU in units of nw
          clr = c("red", "blue", "limegreen"))
```

Arguments

nw	number of fields around the tube
nl	number of fields along the tube
nm	no. stitches per field
np	no. lifelines (persons)
scale	maximal length of life lines in units of nw
clr	vector of colors to be used for life lines

Details

The function produces a graph; ask for a plotting area with aspect ratio of nl to nw.

Value

NULL. The function is used for its side-effect of producing a graph of the knitting pattern.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com, <b@bxc.dk>

Examples

```
# pdf("scarf.pdf", height = 10, width = 50)
LexisScarf()
# dev.off()
```

---

lgrep *Convenience versions of grep*

Description

Often you want the elements of a vector (or its names or levels) that meet a certain pattern. But grep only gives you the position, so these functions are designed to give you that.

Usage

```
fgrep( pattern, x, ... )
ngrep( pattern, x, ... )
lgrep( pattern, x, ... )
```

Arguments

`pattern`	Pattern searched for.
`x`	Object where `pattern` is searched. Or in whose `names` or `levels` attributes `pattern` is sought.
`...`	Arguments passed on to `grep`.

Value

Elements of the input `x` (`fgrep`) or its names attribute (`ngrep`) or levels attribute (`lgrep`).

Author(s)

Bendix Carstensen, <b@bxc.dk>, http://bendixcarstensen.com

See Also

`grep`

Examples

```
ff <- factor( ll <- paste( sample( letters[1:3], 20, replace=TRUE ),
                           sample( letters[1:3], 20, replace=TRUE ), sep="" ) )
ff
fgrep( "a", ff )
fgrep( "a", ll )
ngrep( "a", ff )
lgrep( "a", ff )
lgrep( "a", ff, invert=TRUE )
```

`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: `name`, `mode`, `class` and `size` 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
dfr <- data.frame(x,y)
attach( dfr )
```

```
lls()
search()
clear()
search()
lls()
lls(all=TRUE)
```

---

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: B. Carstensen: Age-Period-Cohort models for the Lexis diagram. Statistics in Medicine, 26: 3018-3045, 2007.

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 )
```

M.dk *Mortality in Denmark 1974 ff.*

Description

Mortality in one-year classes of age (0-98,99+) and period (1974 ff.) in Denmark.

Usage

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

Format

A data frame with 6400 observations on the following 6 variables.

`A` Age-class, 0-98, 99:99+

`sex` Sex. 1:males, 2:females

`P` Period (year) of death

`D` Number of deaths

`Y` Number of person-years

`rate` Mortality rate per 1000 person-years

Details

Deaths in ages over 100 are in the class labelled 99. Risk time is computed by tabulation of the risk time in `Y.dk`, except for the class 99+ where the average of the population size in ages 99+ at the first and last date of the year is used.

Source

http://www.statistikbanken.dk/statbank5a/SelectTable/omrade0.asp?SubjectCode=02&PLanguage=1&ShowNews=OFF

Examples

```
data(M.dk)
str(M.dk)

zz <- xtabs( rate ~ sex+A+P, data=M.dk )
zz[zz==0] <- NA # 0s makes log-scale plots crash
par(mfrow=c(1,2), mar=c(0,0,0,0), oma=c(3,3,1,1), mgp=c(3,1,0)/1.6 )
for( i in 1:2 )
{
matplot( dimnames(zz)[[2]], zz[i,,],
         lty=1, lwd=1, col=rev(heat.colors(37)),
         log="y", type="l", ylim=range(zz,na.rm=TRUE),
         ylab="", xlab="", yaxt="n" )
text( 0, max(zz,na.rm=TRUE), c("M","F")[i], font=2, adj=0:1, cex=2, col="gray" )
if( i==1 ) axis( side=2, las=1 )
}
mtext( side=1, "Age", line=2, outer=TRUE )
mtext( side=2, "Mortality rate", line=2, outer=TRUE )
```

`mat2pol` *Plot columns of a matrix as stacked areas.*

Description

matrix to polygon: Plot columns of a matrix as stacked areas.

Usage

```
mat2pol( pm,
       perm = 1:ncol(pm),
          x = as.numeric(rownames(pm)),
        col = rainbow(ncol(pm)),
         yl = 0:1,
     append = FALSE,
        ... )
```

Arguments

pm	Numerical matrix.
perm	integer vector of length ncol(pm), used to permute the columns of pm.
x	Numeric. The x-axis of the plot.
col	Colors of the areas.
yl	y-axis limits.
append	Logical. Should the polygons be added to an exiating plot
...	Further parameters passed to plot.

Details

The function is originally intended to plot stacked probabilities, hence the default of `0:1` for the y-axis.

Value

A matrix of `ncol(pm)+1` columns with the first equal to 0, and the remaining the cumulative sum of the columns of `pm[perm]`.

The function is called for its side effect - the stacked polygons.

Author(s)

Bendix Carstensen

Examples

```
M <- cbind( sort(runif(10)), sort(runif(10)), sort(runif(10)) )
pm <- sweep( M, 1, apply(M,1,sum), "/" )
mat2pol( pm )
```

---

matshade *Plot confidence intervals as shaded areas around lines.*

Description

Uses an x-vector and a matrix of 3*N columns with estimates and ci.s to produce the lines of estimates and confidence intervals as shaded areas in transparent colours around the lines of the estimates.

Usage

```
matshade( x, y, lty = 1,
          col = 1:(ncol(y)/3), col.shade=col, alpha=0.15,
         plot = dev.cur()==1,
          ... )
```

Arguments

`x`	Numerical vector. Unlike `matplot` this can only be a vector.
`y`	A matrix with 3*N columns — representing estimates and confidence bounds for N curves. Order of columns are assumed to be (est,lo,hi,est,lo,hi...) (or (est,hi,lo...))
`lty`	Line types for the curves.
`col`	Color(s) of the estimated curves.
`col.shade`	Color(s) of the shaded areas. These are the colors that are made transparent by the `alpha` factor. Defaults to the same colors as the lines.

alpha	Number in [0,1] indicating the transparency of the colors for the confidence intervals. Larger values makes the shades darker. Can be a vector which then applies to the curves in turn.
plot	Logical. Should a new plot frame be started? If no device is active, the default is to start one, and plot all ys versus x in transparent color. On the rare occasion a device is open, but no plot have been called you will get an error telling that plot.new has not been called yet, in which case you should explicitly set plot to TRUE.
...	Arguments passed on to matplot (if plot=TRUE) and matlines for use when plotting the lines. Note that lwd=0 will cause lines to be omitted and only the shades be plotted.

Details

All shaded areas are plotted first, the curves added afterwards, so that lines are not 'overshadowed'.

If there are NAs in x or y there will be separate shaded areas for each non-NA sequence. Applies separately to each set of confidence bands in y.

Note that if you repeat the same command, you will get the curves and the shaded areas overplotted in the same frame, so the effect is to have the shades darker, because the transparent colors are plotted on top of those from the first command.

Value

NULL. Used for its side effects.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

See Also

pc.matshade

Examples

```
# Follow-up data of Danish DM patients
data( DMlate )
mL <- Lexis( entry=list(age=dodm-dobth,per=dodm),
             exit=list(per=dox),
       exit.status=factor(!is.na(dodth),labels=c("Alive","Dead")),
             data=DMlate )
# Split follow-up and model by splines
sL <- splitLexis( mL, breaks=0:100, time.scale="age")
## Not run:
# the same thing with popEpi
sL <- splitMulti( mL, age=0:100 )

## End(Not run)
# Mortality rates separately for M and F:
mort <- glm( (lex.Xst=="Dead") ~ sex*Ns(age,knots=c(15,3:8*10)),
```

```
             offset = log(lex.dur),
             family = poisson,
               data = sL )
## Not run:
# The counterpart with gam
library( mgcv )
mort <- gam( (lex.Xst=="Dead") ~ s(age,by=sex) + sex,
             offset = log(lex.dur),
             family = poisson,
               data = sL )

## End(Not run)
# predict rates (per 1000 PY) for men and women
ndM <- data.frame( age=10:90, sex="M", lex.dur=1 )
ndF <- data.frame( age=10:90, sex="F", lex.dur=1 )
# gam objects ignores the offset in prediction so
# lex.dur=1000 in prediction frame wll not work.
prM <- ci.pred( mort, ndM )*1000
prF <- ci.pred( mort, ndF )*1000
# predict rate-ratio
MFr <- ci.exp( mort, ctr.mat=list(ndM,ndF) )
# plot lines with shaded confidence limits
# for illustration we make a holes for the RRs:
MFr[40:45,2] <- NA
MFr[44:49,1] <- NA
matshade( ndM$age, cbind( MFr, prF, prM ), col=c(1,2,4), lwd=3,
          log="y", xlab="Age", ylab="Mortality per 1000 PY (and RR)" )
abline( h=1 )
```

`mcutLexis` *Cut follow-up at multiple event dates and keep track of order of events*

Description

A generalization of `cutLexis` to the case where different events may occur in any order (but at most once for each).

Usage

```
mcutLexis( L0, timescale = 1, wh,
              new.states = NULL,
        precursor.states = transient(L0),
              seq.states = TRUE,
              new.scales = NULL,
            ties.resolve = FALSE )
```

Arguments

`L0`	A Lexis object.

`timescale` Which time scale do the variables in `L0[,wh]` refer to. Can be character or integer.

`wh` Which variables contain the event dates. Character or integer vector

`new.states` Names of the events forming new states. If `NULL` equal to the variable names from `wh`.

`precursor.states` Which states are precursor states. See `cutLexis` for definition of precursor states.

`seq.states` Should the sequence of events be kept track of? That is, should A-B be considered different from B-A. If `FALSE`, the state with combined preceding events A and B will be called A+B (alphabetically sorted).

May also be supplied as character: `s` - sequence, keep track of sequence of states occupied (same as `TRUE`), `u` - unordered, keep track only of states visited (same as `FALSE`) or `l`, `c` - last or current state, only record the latest state visited. If given as character, only the first letter converted to lower case is used.

`new.scales` Should we construct new time scales indicating the time since each of the event occurrences.

`ties.resolve` Should tied event times be resolved by adding random noise to tied event dates. If `FALSE` the function will not accept that two events occur at the same time for a person (ties). If `TRUE` a random quantity in the range `c(-1,1)/100` will be added to all event times in all records with at least one tie. If `ties.resolve` is numeric a random quantity in the range `c(-1,1)*ties.resolve` will be added to all event times in all records with at least one tie.

Value

A `Lexis` object with extra states created by occurrence of a number of intermediate events.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

See Also

`cutLexis`, `rcutLexis`, `xcutLexis`, `addCov.Lexis`, `addDrug.Lexis`, `Lexis`, `splitLexis`

Examples

```
# A dataframe of times
set.seed(563248)
dd <- data.frame( id = 1:30,
                  doN = round(runif(30,-30, 0),1),
                  doE = round(runif(30,  0,20),1),
                  doX = round(runif(30, 50,60),1),
                  doD = round(runif(30, 50,60),1),
                  # these are the event times
                  doA = c(NA,21,NA,27,35,NA,52, 5,43,80,
                          NA,22,56,28,53,NA,51, 5,43,80,
                          NA,23,NA,33,51,NA,55, 5,43,80),
```

```
                doB = c(NA,20,NA,53,27,NA, 5,52,34,83,
                        NA,20,23,37,35,NA,52, 8,33,NA,
                        25,NA,37,40,NA,NA,15,23,36,61) )

# set up a Lexis object with time from entry to death/exit
Lx <- Lexis( entry = list(time=doE,
                          age=doE-doN),
             exit = list(time=pmin(doX,doD)),
       exit.status = factor(doD<doX,labels=c("OK","D")),
             data = dd )
summary( Lx )

# cut the follow-up at dates doA and doB
L2 <- mcutLexis( Lx, "time", wh=c("doA","doB"),
                 new.states = c("A","B"),
           precursor.states = "OK",
                 seq.states = TRUE,
                 new.scales = c("tfA","tfB") )
summary( L2 )
L2

# show the states
boxes( L2, boxpos=list(x=c(10,60,50,90,50,90),
                       y=c(50,50,90,90,10,10)),
           scale.R=100, show.BE=TRUE, DR.sep=c(" (",")"))


L3 <- mcutLexis( Lx, "time", wh=c("doA","doB"),
                 new.states = c("A","B"),
           precursor.states = "OK",
                 seq.states = FALSE,
                 new.scales = c("tfA","tfB") )
summary( L3 )
boxes( L3, boxpos=list(x=c(10,50,50,90,50),
                       y=c(50,50,90,50,10)),
           show.R=FALSE, show.BE=TRUE )
```

`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)
## S3 method for class 'mh'
print(x, ...)
```

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.
x	a mh object
...	arguments passed on to print

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 of class mh 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.
#
```

`mod.Lexis` *Fit intensity models to follow-up data in Lexis objects*

Description

Modeling intensities based on Lexis objects, exploiting the structure of the Lexis objects where the events and risk time have predefined representations. This allows a simpler syntax than the traditional explicit modeling using `glm`, `gam` and `coxph`. Requires that `lex.Cst` and `lex.Xst` are defined as factors.

But it is just a set of wrappers for `glm`, `gam` and `coxph`.

Usage

```
glmLexis(Lx, formula,
         from = preceding(Lx, to), to = absorbing(Lx),
         paired = FALSE, link = "log", scale = 1, verbose = TRUE, ... )
gamLexis(Lx, formula,
         from = preceding(Lx, to), to = absorbing(Lx),
         paired = FALSE, link = "log", scale = 1, verbose = TRUE, ... )
coxphLexis(Lx, formula,
           from = preceding(Lx, to), to = absorbing(Lx),
           paired = FALSE, verbose = TRUE, ... )
  glm.Lexis( Lx,         # Lexis object
        formula,         # ~ model
           from = preceding(Lx, to), # 'from' states
             to = absorbing(Lx)    , # 'to' states
         paired = FALSE, # only the pairwise
           link = "log", # link function
          scale = 1,     # scaling of PY
        verbose = TRUE,  # report what is done?
            ... )        # further arguments to glm
  gam.Lexis( Lx,         # Lexis object
        formula,         # ~ model
           from = preceding(Lx, to), # 'from' states
             to = absorbing(Lx)    , # 'to' states
         paired = FALSE, # only the pairwise
           link = "log", # link function
          scale = 1,     # scaling of PY
        verbose = TRUE,  # report what is done?
            ... )        # further arguments to gam
coxph.Lexis( Lx,         # Lexis object
        formula,         # timescale ~ model
           from = preceding(Lx, to), # 'from' states
             to = absorbing(Lx)    , # 'to' states
         paired = FALSE, # only the pairwise
        verbose = TRUE,  # report what is done?
            ... )        # further arguments to coxph
```

Arguments

Lx	A Lexis object representing cohort follow-up.
formula	Model formula describing the model for the intensity(-ies). For glm and gam, the formula should be one-sided; for coxph the formula should be two-sided and have the name of the time-scale used for baseline hazard as the l.h.s.
from	Character vector of states **from** which transitions are considered. May also be an integer vector in which case the reference will be to the position of levels of lex.Cst. Defaults to the collection of transient states immediately preceding the absorbing states.
to	Character vector of states **to** which a transition is considered an event. May also

	be an integer vector in which case the reference will be to the position of levels of `lex.Xst`. Defaults to the set of absorbing states.
`paired`	Logical. Should the states mentioned in `to`, rep. `from` be taken as pairs, indicating the only transitions modeled. If `FALSE` all transitions from any of the states in `from` to any states in `to` are modeled.
`link`	Character; name of the link function used, allowed values are `'log'` (the default), `'identity'` and `'sqrt'`, see the family `poisreg`.
`scale`	Scalar. `lex.dur` is divided by this number before analysis, so that you can get resulting rates on a scale of your wish.
`verbose`	Print information on the states modeled?
`...`	Further arguments passed on to `glm`, `glm` or `coxph`

Details

The functions with and without dots in the name are identical

The `glm` and `gam` models are fitted using the family `poisreg` which is a bit faster than the traditional `poisson` family. The response variable for this family is a two-column vector of events and person-time respectively, so the predictions, for example using `ci.pred` does not require `lex.dur` (and would ignore this) as variable in the `newdata`. `ci.pred` will return the estimated rates in units of the `lex.dur` in the `Lexis` object, scaled by `scale`, which has a default value of 1.

The default is to model all transitions into any absorbing state by the same model (how wise is that??). If only `from` is given, `to` is set to all states reachable from `from`, which may be a really goofy model and if so a warning is issued. If only `to` is given, `from` is set to the collection of states from which `to` can be reached directly — see `preceding` and its cousins. This convention means that if you have a `Lexis` object representing a simple survival analysis, with states, say, "alive" and "dead", you can dispense with the `from` and `to` arguments.

Occasionally you only want to model a subset of the possible transitions from states in `from` to states in `to`, in which case you specify `from` and `to` as character vectors of the same length and set `paired=TRUE`. Then only transitions `from[i]` to `to[i]`, `i`=1,2,... will be modeled.

There is no working `update` functions for these objects (yet).

Strictly speaking, it is a bit counter-intuitive to have the time-scale on the l.h.s. of the formula for the `coxph` since the time scale is also a predictor of the occurrence rate. On the other hand, calling `coxph` directly would also entail having the name of the time scale in the `Surv` object on the l.h.s. of the formula. So the inconsistency is merely carried over from `coxph`.

Value

`glmLexis` returns a `glm` object, which is also of class `glm.lex`, `gamLexis` returns a `gam` object, which is also of class `gam.lex`, and `coxphLexis` returns a `coxph` object, which is also of class `coxph.lex`. These extra class attributes are meant to facilitate the (still pending) implementation of an `update` function.

The returned objects all have an extra attribute, `Lexis` which is a list with entries `data`, the name of the `Lexis` object modeled (note that it is *not* the object, only the name of it, which may not be portable); `trans`, a character vector of transitions modeled; `formula`, the model formula; and `scale`, the scaling applied to `lex.dur` before modeling.

Only the glm and gam objects have the scale element in the list; a scalar indicating the scaling of lex.dur before modeling. Note that the formula component of the Lexis attribute of a coxph object is a two-sided formula with the baseline time scale as the l.h.s.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com.

See Also

Lexis, cutLexis, mcutLexis, addCov.Lexis, absorbing, transient

Examples

```
library( Epi )
library( survival )
data( DMlate )

# Lexis object of total follow-up
mL <- Lexis( entry = list(age=dodm-dobth,per=dodm),
             exit = list(per=dox),
       exit.status = factor(!is.na(dodth),labels=c("Alive","Dead")),
             data = DMlate )

# Cut follow-up at start of insulin use
cL <- cutLexis( mL, cut = mL$doins,
             timescale = "per",
             new.state = "Ins",
       precursor.states = "Alive" )

# Split follow-up on age-axis
system.time( sL <- splitLexis( cL, breaks=0:25*4, time.scale="age") )
# ( consider splitMulti from the popEpi package )
summary( sL )

# glm models for rates based on the time-split dataset by insulin and sex

# Proportional hazards model with insulin as time-dependent variable
# - uses the defaul of modeling all transitions from both transient
# states ("Alive" and "Ins") to the absorbing state ("Dead").
mt <- glmLexis( sL, ~ sex + lex.Cst + Ns(age,knots=c(15,3:8*10)) )

# prediction of mortality rates from "Alive" with and without PH assumption
nA <- data.frame( age=40:70, sex="M", lex.Cst="Alive" )
nI <- data.frame( age=40:70, sex="M", lex.Cst="Ins" )
matshade( nA$age, cbind( ci.pred(mt,nA),
                         ci.pred(mt,nI) )*1000, plot=TRUE,
          lwd=3, lty=1, log="y", col=c("black","blue","red"),
          xlab="Age", ylab="Mortality per 1000 PY" )

# gam models may take some time to run so we leave it out
## Not run:
```

```
mt.gam <- gamLexis( sL, ~ sex + lex.Cst + s(age), to="Dead",
                    scale=1000 )

## End(Not run)

# Fit a Cox model for mortality with age as baseline time scale and
# insulin (lex.Cst) as time-dependent covariate
mt.cox <- coxphLexis( sL, age ~ sex + lex.Cst, c("Alive","Ins"), "Dead" )

# Pretty much the same results for regression paramters as the glm:
  ci.exp( mt    , subset="ex" )
# ci.exp( mt.gam, subset="ex" )
  ci.exp( mt.cox, subset="ex" )
```

`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)
```

N.dk *Population size in Denmark*

Description

The population size at 1st January in ages 0-99.

Usage

```
data(N.dk)
```

Format

A data frame with 7200 observations on the following 4 variables.

`sex` Sex, 1:males, 2:females

`A` Age. 0:0, 1:1, ..., 98:98, 99:99+

`P` Year

`N` Number of persons alive at 1st January year `P`

Source

http://www.statistikbanken.dk/statbank5a/SelectTable/omrade0.asp?SubjectCode=02&PLanguage=1&ShowNews=OFF

Examples

```
data(N.dk)
str(N.dk)
with(N.dk,addmargins(tapply(N,list(P,sex),sum),2))
with(subset(N.dk,P==max(P)),addmargins(tapply(N,list(A,sex),sum)))
```

N2Y *Create risk time ("Person-Years") in Lexis triangles from population count data.*

Description

Data on population size at equidistant dates and age-classes are used to estimate person-time at risk in Lexis-triangles, i.e. classes classified by age, period AND cohort (date of birth). Only works for data where age-classes have the same width as the period-intervals.

Usage

```
N2Y( A, P, N,
     data = NULL,
     return.dfr = TRUE)
```

Arguments

A	Name of the age-variable, which should be numeric, corresponding to the left endpoints of the age intervals.
P	Name of the period-variable, which should be numeric, corresponding to the date of population count.
N	The population size at date P in age class A.
data	A data frame in which arguments are interpreted.
return.dfr	Logical. Should the results be returned as a data frame (default TRUE) or as a table.

Details

The calculation of the risk time from the population figures is done as described in: B. Carstensen: Age-Period-Cohort models for the Lexis diagram. Statistics in Medicine, 26: 3018-3045, 2007.

The number of periods in the result is one less than the number of dates (nP=length(table(P))) in the input, so the number of distinct values is 2*(nP-1), because the P in the output is coded differently for upper and lower Lexis triangles.

The number of age-classes is the same as in the input (nA=length(table(A))), so the number of distinct values is 2*nA, because the A in the output is coded differently for upper and lower Lexis triangles.

In the paper "Age-Period-Cohort models for the Lexis diagram" I suggest that the risk time in the lower triangles in the first age-class and in the upper triangles in the last age-class are computed so that the total risk time in the age-class corresponds to the average of the two population figures for the age-class at either end of a period multiplied with the period length. This is the method used.

Value

A data frame with variables A, P and Y, representing the mean age and period in the Lexis triangles and the person-time in them, respectively. The person-time is in units of the distance between population count dates.

If return.dfr=FALSE a three-way table classified by the left end point of the age-classes and the periods and a factor wh taking the values up and lo corresponding to upper (early cohort) and lower (late cohort) Lexis triangles.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

References

B. Carstensen: Age-Period-Cohort models for the Lexis diagram. Statistics in Medicine, 26: 3018-3045, 2007.

See Also

splitLexis, apc.fit

Examples

```
# Danish population at 1 Jan each year by sex and age
data( N.dk )
# An illustrative subset
( Nx <- subset( N.dk, sex==1 & A<5 & P<1975 ) )
# Show the data in tabular form
xtabs( N ~ A + P, data=Nx )
# Lexis triangles as data frame
Nt <- N2Y( data=Nx, return.dfr=TRUE )
xtabs( Y ~ round(A,2) + round(P,2), data=Nt )
# Lexis triangles as a 3-dim array
ftable( N2Y( data=Nx, return.dfr=FALSE ) )

# Calculation of PY for persons born 1970 in 1972
( N.1.1972 <- subset( Nx, A==1 & P==1972)$N )
( N.2.1973 <- subset( Nx, A==2 & P==1973)$N )
N.1.1972/3 + N.2.1973/6
N.1.1972/6 + N.2.1973/3
# These numbers can be found in the following plot:

# Blue numbers are population size at 1 January
# Red numbers are the computed person-years in Lexis triangles:
Lexis.diagram(age=c(0,5), date=c(1970,1975), int=1, coh.grid=TRUE )
with( Nx, text(P,A+0.5,paste(N),srt=90,col="blue") )
with( Nt, text(P,A,formatC(Y,format="f",digits=1),col="red") )
text( 1970.5, 2, "Population count 1 January", srt=90, col="blue")
text( 1974.5, 2, "Person-\nyears", col="red")
```

NArray *Set up an array of NAs, solely from the list of dimnames*

Description

Defines an array of NAs, solely from the list of dimnames

Usage

```
NArray( x, cells=NA )
ZArray( x, cells=0 )
```

Arguments

x	A (possibly named) list to be used as dimnames for the resulting array
cells	Value(s) to fill the array

Details

This is a simple useful way of defining arrays to be used for collection of results. The point is that everything is defined from the named list, so in the process of defining what you want to collect, there is only one place in the program to edit. It's just a wrapper for `array`. `ZArray` is just a wrapper for `NArray` with a different default.

Value

An array with `dimnames` attribute `x`, and all values equal to `cells`.

Author(s)

Bendix Carstensen

Examples

```
ftable(
NArray( list(Aye = c("Yes", "Si", "Oui"),
             Bee = c("Hum", "Buzz"),
             Sea = c("White", "Black", "Red", "Dead") ) ) )
```

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(breaks)` if `type="left"`, for values above `max(breaks)` if `type="right"` and for values outside `range(breaks)` 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, <b@bxc.dk>, http://bendixcarstensen.com, with essential input from Martyn Plummer, <martyn.plummer@r-project.org>

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 for axes on plots*

Description

The function calls `pretty` for linear scale. For a log-scale nice are computed using a set of specified number in each decade.

Usage

```
nice(x, log = FALSE, lpos = c(1, 2, 5), xmx = 4, ...)
```

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.
`xmx`	Numeric. The maximal (absolute) power of 10 to be used for a log-scale.
`...`	Arguments passed on to `pretty` if `log=FALSE`

Value

A vector of breakpoints.

Author(s)

Bendix Carstensen, <b@bxc.dk>, http://bendixcarstensen.com

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)
```

Ns *Natural splines - (cubic splines linear beyond outermost knots) with convenient specification of knots and possibility of centering, detrending and clamping.*

Description

This function is partly for convenient specification of natural splines in practical modeling. The convention used is to take the smallest and the largest of the supplied knots as boundary knots. It also has the option of centering the effects provided at a chosen reference point as well as projecting the columns on the orthogonal space to that spanned by the intercept and the linear effect of the variable, and finally fixing slopes beyond boundary knots (clamping).

Usage

```
Ns( x, ref = NULL, df = NULL,
                knots = NULL,
            intercept = FALSE,
       Boundary.knots = NULL,
                fixsl = c(FALSE,FALSE),
              detrend = FALSE )
```

Arguments

`x`	A variable.
`ref`	Scalar. Reference point on the `x`-scale, where the resulting effect will be 0.
`df`	degrees of freedom.
`knots`	knots to be used both as boundary and internal knots. If `Boundary.knots` are given, this will be taken as the set of internal knots.
`intercept`	Should the intercept be included in the resulting basis? Ignored if any of `ref` or `detrend` is given.
`Boundary.knots`	The boundary knots beyond which the spline is linear. Defaults to the minimum and maximum of `knots`.
`fixsl`	Specification of whether slopes beyond outer knots should be fixed to 0. `FALSE` correponds to no restriction; a curve with 0 slope beyond the upper knot is obtained using `c(FALSE,TRUE)`. Ignored if `!(detrend==FALSE)`.
`detrend`	If `TRUE`, the columns of the spline basis will be projected to the orthogonal of `cbind(1,x)`. Optionally `detrend` can be given as a vector of non-negative numbers og length `length(x)`, used to define an inner product as `diag(detrend)` for projection on the orthogonal to `cbind(1,x)`. The default is projection w.r.t. the inner product defined by the identity matrix.

Value

A matrix of dimension c(length(x),df) where either `df` was supplied or if `knots` were supplied, `df = length(knots) - 1 + intercept`. `Ns` returns a spline basis which is centered at `ref`. `Ns` with the argument `detrend=TRUE` returns a spline basis which is orthogonal to `cbind(1,x)` with respect to the inner product defined by the positive definite matrix `diag(detrend)` (an assumption which is checked). Note the latter is data dependent and therefore making predictions with a `newdata` argument will be senseless.

Note

The need for this function is primarily from analysis of rates in epidemiology and demography, where the dataset are time-split records of follow-up, and the range of data therefore rarely is of any interest (let alone meaningful).

In Poisson modeling of rates based on time-split records one should aim at having the same number of *events* between knots, rather than the same number of observations.

Author(s)

Bendix Carstensen <b@bxc.dk>, Lars Jorge D\'iaz, Steno Diabetes Center Copenhagen.

Examples

```
require(splines)
require(stats)
require(graphics)

ns( women$height, df = 3)
Ns( women$height, knots=c(63,59,71,67) )

# Gives the same results as ns:
summary( lm(weight ~ ns(height, df = 3), data = women) )
summary( lm(weight ~ Ns(height, df = 3), data = women) )

# Get the diabetes data and set up as Lexis object
data(DMlate)
DMlate <- DMlate[sample(1:nrow(DMlate),500),]
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 in 1-year age intervals
dms <- splitLexis( dml, time.scale="Age", breaks=0:100 )
summary( dms )

# Model  age-specific rates using Ns with 6 knots
# and period-specific RRs around 2000 with 4 knots
# with the same number of deaths between each pair of knots
n.kn <- 6
( a.kn <- with( subset(dms,lex.Xst=="Dead"),
                quantile( Age+lex.dur, probs=(1:n.kn-0.5)/n.kn ) ) )
n.kn <- 4
( p.kn <- with( subset( dms, lex.Xst=="Dead" ),
                quantile( Per+lex.dur, probs=(1:n.kn-0.5)/n.kn ) ) )
m1 <- glm( lex.Xst=="Dead" ~ Ns( Age, kn=a.kn ) +
                             Ns( Per, kn=p.kn, ref=2000 ),
           offset = log( lex.dur ),
           family = poisson,
             data = dms )

# Plot estimated age-mortality curve for the year 2005 and knots chosen:
nd <- data.frame( Age=seq(40,100,0.1), Per=2005, lex.dur=1000 )
par( mfrow=c(1,2) )
matplot( nd$Age, ci.pred( m1, newdata=nd ),
         type="l", lwd=c(3,1,1), lty=1, col="black", log="y",
         ylab="Mortality rates per 1000 PY", xlab="Age (years)", las=1, ylim=c(1,1000) )
rug( a.kn, lwd=2 )

# Clamped Age effect to the right of rightmost knot.
```

```
m1.c <- glm( lex.Xst=="Dead" ~ Ns( Age, kn=a.kn, fixsl=c(FALSE,TRUE) ) +
                               Ns( Per, kn=p.kn, ref=2000 ),
             offset = log( lex.dur ),
             family = poisson,
               data = dms )

# Plot estimated age-mortality curve for the year 2005 and knots chosen.
matplot( nd$Age, ci.pred( m1.c, newdata=nd ),
         type="l", lwd=c(3,1,1), lty=1, col="black", log="y",
         ylab="Mortality rates per 1000 PY", xlab="Age (years)", las=1, ylim=c(1,1000) )
rug( a.kn, lwd=2 )

par( mfrow=c(1,1) )

# Including a linear Age effect of 0.05 to the right of rightmost knot.
m1.l <- glm( lex.Xst=="Dead" ~ Ns( Age, kn=a.kn, fixsl=c(FALSE,TRUE) ) +
                               Ns( Per, kn=p.kn, ref=2000 ),
             offset = log( lex.dur ) + pmax( Age, max( a.kn ) ) * 0.05,
             family = poisson,
               data = dms )

# Plot estimated age-mortality curve for the year 2005 and knots chosen.
nd <- data.frame(Age=40:100,Per=2005,lex.dur=1000)
matplot( nd$Age, ci.pred( m1.l, newdata=nd ),
         type="l", lwd=c(3,1,1), lty=1, col="black", log="y",
         ylab="Mortality rates per 1000 PY", xlab="Age (years)", las=1, ylim=c(1,1000) )
rug( a.kn, lwd=2 )
```

---

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
dev.new( 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 )
```

`overlap.Lexis` *Check if records in a Lexis object represent overlapping epochs*

Description

When constructing a `Lexis` object from a data frame with multiple records there is no guarantee that the records from a given person will represent disjoint time intervals (epochs), `overlap` checks this.

Usage

```
## S3 method for class 'Lexis'
overlap(Lx, ...)
```

Arguments

`Lx`	A `Lexis` object
`...`	ignored

Value

If no overlapping epochs are found, a message is printed and NULL is returned. Otherwise a vector of unique ids (from Lx$lex.id) for the persons with overlapping records is returned invisibly

Author(s)

Bendix Carstensen, http://bendixcarstensen.com, <b@bxc.dk>

See Also

Lexis

Examples

```
# run the DMlate example to get the Lexis object dmi with
# some persons with multiple (2 that is) records:
example(DMlate)
overlap.Lexis(dmi)
#
# now find 4 persons with multiple records
tt <- table(dmi$lex.id)
has2 <- dmi$lex.id %in% names(tt[tt > 1])[1:4]
table(has2)
#
# add time (0.3 y) to lex.dur for these persons to produce overlapping intervals
dmi$lex.dur[has2] <- dmi$lex.dur[has2] + 0.3
(overlap.Lexis(dmi))
```

---

paths.Lexis *Generate paths traveled through a Lexis multistate data frame.*

Description

For each person the path traveled through states in a Lexis object is derived.

Usage

```
## S3 method for class 'Lexis'
paths(Lx, dfr = FALSE, coll = "->", ...)
```

Arguments

Lx	A Lexis object
dfr	Logical. Should results be returned as a data frame with columns lex.id and path?
coll	Character string used to separate state names in the paths
...	Arguments passed on. Ignored

Value

A factor with levels describing each person's path through states. It is of length `length(nid(Lx))`, named by the (character) values of `Lx$lex.id`. It will usually be wise to use `table` on the output.

If `dfr` is `TRUE` a two-column data frame is returned.

Author(s)

Bendix Carstensen, <b@bxc.dk>, http://bendixcarstensen.com

See Also

`cutLexis`, `mcutLexis`, `rcutLexis`, `xcutLexis`, `nid`, `Lexis`

Examples

```
# a simple example from the package
example(DMlate)
str(paths.Lexis(dmi, dfr = TRUE))
str(pathD <- paths.Lexis(dmi))
cbind(addmargins(table(pathD)))
#
# an example with recurring events
example(steno2)
summary(L4)
str(pathS <- paths.Lexis(L4))
cbind(addmargins(table(pathS)))
```

---

pc.lines *Plot period and cohort effects in an APC-frame.*

Description

When an APC-frame has been produced by `apc.frame`, this function draws curves or points in the period/cohort part of the frame.

Usage

```
pc.points( x, y, ... )
pc.lines( x, y, ... )
pc.matpoints( x, y, ... )
pc.matlines( x, y, ... )
pc.matshade( x, y, ... )
cp.points( x, y, ... )
cp.lines( x, y, ... )
cp.matpoints( x, y, ... )
cp.matlines( x, y, ... )
cp.matshade( x, y, ... )
```

Arguments

x	vector of x-coordinates.
y	vector or matrix of y-coordinates.
...	Further parameters to be transmitted to points, lines, matpoints, matlines or matshade used for plotting curves in the calendar time realm of a graph generated by apc.frame

Details

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.

The functions cp.points etc. are just synonyms for these, in recognition of the fact that you can never remember whether it is "pc" or "cp".

Value

The functions return nothing.

Author(s)

Bendix Carstensen, Steno Diabetes Center Copenhagen, http://bendixcarstensen.com

See Also

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

---

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 of percentages, 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://bendixcarstensen.com.

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.apc *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

```
## S3 method for class 'apc'
plot( x, r.txt="Rate", ...)
          apc.plot( x, r.txt="Rate", ...)
```

Arguments

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

Details

`plot.apc` is just a wrapper for `apc.plot`.

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, however see `cp.points`.

Author(s)

Bendix Carstensen, Steno Diabetes Center, http://bendixcarstensen.com

See Also

`apc.lines`, `lines.apc`, `apc.frame`, `apc.fit`

Examples

```
data( lungDK )
apc1 <- apc.fit( transform( lungDK,
                            A = Ax, P = Px, Y = Y/10^5 ),
                 ref.c = 1920 )
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] )
```

`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 )
```

---

plotCIF *Plotting Aalen-Johansen curves for competing events*

Description

Function plotCIF plots, for one or more groups, the cumulative incidence curves for a selected event out of two or more competing events. Function stackedCIF plots, for one group or popula-

tion, the cumulative incidence curves for two or more competing events such that the cumulative incidences are stacked upon each other. The CIFs are are estimated by the Aalen-Johansen method.

Usage

```
## S3 method for class 'survfit'
 plotCIF( x, event = 1,
              xlab = "Time",
              ylab = "Cumulative incidence",
              ylim = c(0, 1),
               lty = 1,
               col = "black", ... )

## S3 method for class 'survfit'
stackedCIF( x, group = 1,
                 col = "black",
                fill = "white",
                ylim = c(0,1),
                xlab = "Time",
                ylab = "Cumulative incidence", ... )
```

Arguments

x	An object of class survfit, the type of event in Surv() being "mstate"; the first level of the event factor represents censoring and the remaining ones the alternative competing events.
event	Determines the event for which the cumulative incidence curve is plotted by plotCIF.
group	An integer showing the selected level of a possible grouping factor appearing in the model formula in survfit when plotting by stackedCIF
col	A vector specifying the plotting color(s) of the curve(s) for the different groups in plotCIF– default: all "black".
fill	A vector indicating the colours to be used for shading the areas pertinent to the separate outcomes in stackedCIF - default: all "white".
xlab	Label for the x-axis.
ylab	Label for the y-axis.
ylim	Limits of the y-axis.
lty	A vector specifying the line type(s) of the curve(s) for the different groups - default: all 1 (=solid).
...	Further graphical parameters to be passed.

Details

The order in which the curves with stackedCIF are piled upon each other is the same as the ordering of the values or levels of the competing events in the pertinent event variable. The ordering can be changed by permuting the levels as desired using function Relevel, after which survfit is called with the relevelled event variable in Surv()

Value

No value is returned but a plot is produced as a side-effect.

Note

Aalen-Johansen curves for competing events in several groups can also be plotted by function plot.survfit of the survival library as well as by some functions in other packages covering analysis of time-to-event data.

Author(s)

Esa Laara, <esa.laara@oulu.fi>

References

Putter, H., Fiocco, M., Geskus, R.B. (2007). Tutorial in biostatistics: competing risks and multi-state models. Statistics in Medicine, 26: 2389–2430.

See Also

survfit, plot, plot.survfit.

Examples

```
library(survival)   #  requires version 2.39-4 or later
head(mgus1)
#  Aalen-Johansen estimates of CIF are plotted by sex for two
#  competing events: (1) progression (pcm), and (2) death, in
#  a cohort of patients with monoclonal gammopathy.

#  The data are actually covering transitions from pcm to death, too,
#  for those entering the state of pcm. Such patients have two rows
#  in the data frame, and in their 2nd row the 'start' time is
#  the time to pcm (in days).

#  In our analysis we shall only include those time intervals with value 0
#  for variable 'start'. Thus, the relevant follow-up time is represented
#  by variable 'stop' (days). For convenience, days are converted to years.

fitCI <- survfit(Surv(stop/365.25, event, type="mstate") ~ sex,
             data= subset(mgus1, start==0) )
par(mfrow=c(1,2))
plotCIF(fitCI, event = 1, col = c("red", "blue"),
  main = "Progression", xlab="Time (years)" )
text( 38, 0.15, "Men", pos = 2)
text( 38, 0.4, "Women", pos = 2)
plotCIF(fitCI, event = 2, col = c("red", "blue"),
  main = "Death", xlab="Time (years)" )
text( 38, 0.8, "Men", pos = 2)
text( 38, 0.5, "Women", pos = 2)
```

```
par(mfrow=c(1,2))
stackedCIF(fitCI, group = 1, fill = c("gray80", "gray90"),
  main = "Women", xlab="Time (years)" )
text( 36, 0.15, "PCM", pos = 2)
text( 36, 0.6, "Death", pos = 2)
stackedCIF(fitCI, group = 2, fill = c("gray80", "gray90"),
  main = "Men", xlab="Time (years)" )
text( 39, 0.10, "PCM", pos = 2)
text( 39, 0.6, "Death", pos = 2)
```

---

plotEst *Plot estimates with confidence limits (forest plot)*

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.txt = "black",
    font.txt = 1,
   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. Either a character vector or an expression vector.
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.txt	Colour of the text annotating the estimates.
font.txt	Font for the text annotating the estimates.
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.
...	Arguments passed on to ci.lin when a model object is supplied as ests.

Details

plotEst makes a news plot, whereas linesEst and pointsEst (identical functions) adds to an existing plot.

If a model object of class "glm", "coxph", "clogistic" or "gnlm" is supplied the argument xlog defaults to TRUE, and exponentiated estimates are extracted by default.

Value

NULL

Author(s)

Bendix Carstensen, <b@bxc.dk>, http://bendixcarstensen.com

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 <- ci.lin( m1, subset=-1 )[,-(2:4)] )

# Plots with increasing amounts of bells and whistles
par( mfcol=c(3,2), mar=c(3,3,2,1) )
plotEst( cf )
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 fb"
plotEst( cf, grid=TRUE, cex=2, col.points=rainbow(4),
                               col.lines=rainbow(4), vref=1 )
#
etxt <- expression("Plain text, qouted",
                   "combined with maths:"*sqrt(a)*phi[c],
                   f^d*"  Hb"*A[1][c],
                   eff^e*"  kg/"*m^2)
plotEst( cf, txt=etxt, 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

`poisreg` *Family Object for Poisson Regression*

Description

The `poisreg` family allows Poisson regression models to be fitted using the `glm` function.

In a Poisson regression model, we assume that the data arise from a Poisson process. We observe D disease events in follow up time Y and wish to estimate the incidence rate, which is assumed to be constant during the follow-up period for any individual. The incidence rate varies between individuals according to the predictor variables and the link function in the model specification.

When using the `poisreg` family in the `glm` function, the response should be specified as a two-column matrix with the first column giving the number of events (D) and the second column giving the observation time (Y). This is similar to the `binomial` family for which a two-column outcome can be used representing the number of successes and the number of failures.

Usage

```
poisreg(link = "log")
```

Arguments

`link`	a specification for the model link function. The `poisreg` family accepts the links `identity`, `log` and `inverse`.

Value

An object of class "family". See family for details.

The family name, represented by the element "family" in the returned object, is "poisson" and not "poisreg". This is necessary to prevent the summary.glm function from estimating an overdispersion parameter (which should be fixed at 1) and therefore giving incorrect standard errors for the estimates.

Note

When using the log link, Poisson regression can also be carried out using the poisson family by including the log follow-up time log(Y) as an offset. However this approach does not generalize to other link functions. The poisreg family allows more general link functions including additive risk models with poisreg(link = "identity").

See Also

glm, family.

Examples

```
## Estimate incidence rate of diabetes in Denmark (1996-2015) by
## age and sex
data(DMepi)
DMepi$agegrp <- cut(DMepi$A, seq(from=0, to=100, by=5))
inc.diab <- glm(cbind(X, Y.nD) ~ -1 + agegrp + sex, family=poisreg,
                data=DMepi)
## The coefficients for agegrp are log incidence rates for men in each
## age group. The coefficient for sex is the log of the female:male
## incidence rate ratio.
summary(inc.diab)

## Smooth function with non-constant M/F RR:
requireNamespace("mgcv")
library( mgcv )
gam.diab <- gam( cbind(X, Y.nD) ~ s(A,by=sex) + sex,
                 family=poisreg,
                 data=DMepi)

## There is no need/use for Y.nD in prediction data frames:
nM <- data.frame( A=20:90, sex="M" )
nF <- data.frame( A=20:90, sex="F" )

## Rates are returned in units of (1 year)^-1, so we must scale the
## rates by hand:
matshade( nM$A, cbind( ci.pred(gam.diab,     nM    )*1000,
                       ci.pred(gam.diab,        nF )*1000,
                       ci.exp( gam.diab,list(nM,nF)) ),
          plot=TRUE, col=c("blue","red","black"),
         log="y", xlab="Age", ylab="DM incidence rates per 1000     /     M vs. F RR" )
abline(h=1)
```

pr *Diabetes prevance as of 2010-01-01 in Denmark*

Description

Diabetes prevalence as of 2010-01-01 in Denmark in 1-year age classes by sex.

Usage

```
data("pr")
```

Format

A data frame with 200 observations on the following 4 variables.

`A` Numeric, age, 0-99

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

`X` Number of diabetes patients

`N` Population size

Examples

```
data(pr)
str(pr)
```

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://bendixcarstensen.com, 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://bendixcarstensen.com

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 )
```

`rcutLexis` *A function to cut follow-up at intermediate event times.*

Description

Cuts follow-up at intermediate event times, multiple events per person are allowed, as well as recurrences of the sme type of event. The resulting states only refer to the last assumed state, unlike the result from `mcutLexis`.

Usage

```
rcutLexis( Lx, cut,
    timescale = 1,
    precursor.states = transient(Lx))
```

Arguments

`Lx`	A `Lexis` object to be amended,.
`cut`	A data frame with columns `lex.id`, `cut` (event times) and `new.state` (event type, character)
`timescale`	What time scale do values in `cut$cut` refer to. Numeric or character.
`precursor.states`	an optional vector of states to be considered as "less severe" than `new.state`. See Details in the documentation of `cutLexis`

Value

A `Lexis` object with follow-up cut at the event times supplied in `cut`

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

See Also

`cutLexis`, `mcutLexis`, `addCov.Lexis`, `Lexis`, `splitLexis`

Examples

```
df <- data.frame(lex.id = rep(c(3, 7), c(3, 5)))
df$new.state <- sample(LETTERS[2:4], 8, r = TRUE)
df$cut <- round(runif(8) * 100) + 1
df

Lx <- Lexis( exit = list(time=c(89, 97)),
               id = c(3, 7),
      exit.status = factor(c("A", "X")) )
Lx
rcutLexis(Lx, df, pre = "A")
```

`Relevel` *Reorder and combine levels of a factor*

Description

The levels of a factor are re-ordered so that the levels specified by `ref` appear first and remaining levels are moved down. This is useful for `contr.treatment` contrasts which take the first level as the reference. Factor levels may also be combined; two possibilities for specifying this are supported: hard coding or table look-up.

Usage

```
## S3 method for class 'factor'
Relevel( x, ref, first = TRUE, collapse="+",
                 xlevels=TRUE, nogroup=TRUE, ... )
```

Arguments

`x`	A(n unordered) factor
`ref`	Vector, list or data frame, array, matrix or table. If `ref` is a vector (integer or character), it is assumed it contains the names or numbers of levels to be the first ones; non mentioned levels are kept. If `ref` is a list (but not a data frame), factor levels mentioned in each list element are combined. If the list is named the names are used as new factor levels, otherwise new level names are constructed from the old. If `ref` is a data frame or 2-dimensional array, matrix or table, the first column is assumed to have unique levels of `x` and the second to have groupings of this, respectively.
`first`	Should the levels mentioned in `ref` (if it is a list) come before those not?
`collapse`	String used when constructing names for combined factor levels.
`xlevels`	Logical. Should all levels in the 2nd column of `ref` be maintained as levels of the result, or (if `FALSE`) only the actually occurring.

nogroup	Logical. Should levels present in the input but not in the 1st column of ref be maintained as levels after the grouping? If FALSE, NAs will be returned for such elements.
...	Arguments passed on to other methods.

Details

The facility where ref is a two-column matrix mimics the SAS-facility of formats where a dataset can be used to construct a format — SAS format is the grouping tool for variable values.

If ref is a two-column object and ref[,2] is a factor Relevel will preserve the order of levels from ref[,2].

Value

An unordered factor, where levels of x have been reordered and/or collapsed.

Author(s)

Bendix Carstensen http://bendixcarstensen.com, Lars Jorge Diaz

See Also

Relevel.Lexis

Examples

```
# Grouping using a list (hard coding)
#
ff <- factor(sample(letters[1:5], 100, replace = TRUE))
table( ff, Relevel(ff, list( AB = 1:2, "Dee" = 4, c(3,5))))
table( ff, Relevel(ff,
                   list( 5:4, Z = c("c", "a") ),
                   coll = "-und-",
                   first = FALSE ) )

## Grouping using a two-column matrix as input:
## A factor with levels to be grouped together
ff <- factor(c("Bear","Bear","Crocodile","Snake","Crocodile","Bear"))
ff
## A grouping table
(gg <- data.frame(Animal = c("Bear","Whale","Crocodile","Snake","Eagle"),
                  Class = c("Mammal","Mammal","Reptile","Reptile","Bird")))
str(gg)
Relevel(ff, gg, xlevels = FALSE)
Relevel(ff, gg )
Relevel(ff, gg[c(1:5,5:1),])

## This crashes with an error
(GG <- rbind( gg, c("Bear","Reptile")))
try(Relevel(ff, GG))
```

```
ff <- factor(c(as.character(ff), "Jellyfish", "Spider"))
Relevel(ff, gg)
# excludes non-occupied levels
Relevel(ff, gg, xlevels = FALSE)
# If you do not want unknown animals classified, this returns NAs:
Relevel(ff, gg, nogroup = FALSE)
# Both
Relevel(ff, gg, nogroup = FALSE, xlevels = FALSE)
```

`rm.tr` *Remove transitions from a Lexis object.*

Description

Sometimes certain transitions are not of interest. This function removes these and assigns the risk time in the target state of the transitions to the originating state.

Usage

```
rm.tr(obj, from, to)
```

Arguments

`obj`	A `Lexis` object.
`from`	Character; name of the state from which the transition to be purged originates. Must be a valid state name for `obj`.
`to`	Character; name of the state to which the transition to be purged targets. Must be a valid state name for `obj`.

Details

The function removes all transitions from `from` to `to`, and assigns all risk time in the `to` state after the transition (`lex.dur`) to the `from` state. This is only done for risk time in `to` occurring directly after `from`. Risk time in `to` occurring after a transition from states different from `from` is not affected. Transitions from `to` to another state, `other`, say, will be changed to transitions from `from` to `other`.

Value

A `Lexis` object with the indicated transition removed.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com.

See Also

`Relevel`

Examples

```
data(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 )

# A small subset for illustration
dml <- subset( dml, lex.id %in% c(13,15,20,28,40) )

# Cut the follow-up at start of insulin therapy
dmi <- cutLexis( dml, cut = dml$doins,
                      pre = "DM",
               new.state = "Ins" )[,1:10]

# How does it look?
dmi

# Remove all transitions DM -> Ins
rm.tr( dmi, "DM", "Ins" )
```

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`. 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

Details

As an alternative to a test and a status variable, 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. The test used to derive sensitivity, specificity, PV+ and PV- as a function of x is test$\geq x$ as a predictor of stat=TRUE.

Value

A list with two components:

res	dataframe with variables sens, spec, pvp, pvn and name of the test variable. The latter is the unique values of test or linear predictor from the logistic regression in ascending order with -Inf prepended. Since the sensitivity is defined as $P(test > x)\|status = TRUE$, the first row has sens equal to 1 and spec equal to 0, corresponding to drawing the ROC curve from the upper right to the lower left corner.
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 Copenhagen, http://bendixcarstensen.com

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)
```

`simLexis` *Simulate a Lexis object representing follow-up in a multistate model.*

Description

Based on a (pre-)`Lexis` object representing persons at given states and times, and full specification of transition intensities between states in the form of models for the transition rates, this function simulates transition times and -types for persons and returns a `Lexis` object representing the simulated cohort. The simulation scheme accommodates multiple timescales, including time since entry into an intermediate state, and accepts fitted Poisson models, Cox-models or just a function as specification of rates.

Usage

```
simLexis( Tr, init,
           N = 1,
      lex.id,
     t.range = 20,
       n.int = 101,
    time.pts = seq(0,t.range,length.out=n.int) )
nState( obj, at, from, time.scale = 1 )
pState( nSt, perm = 1:ncol(nSt) )
## S3 method for class 'pState'
plot( x,
                     col = rainbow(ncol(x)),
                  border = "transparent",
                    xlab = "Time",
                    ylim = 0:1,
                    ylab = "Probability", ... )
## S3 method for class 'pState'
lines( x,
                      col = rainbow(ncol(x)),
                   border = "transparent", ... )
```

Arguments

Tr	A named list of named lists. The names of the list are names of the transient states in the model. Each list element is again a named list. The names of the elements of this inner list are the names of the states reachable from the state with name equal to the list. Elements of the intter lists represent transitions. See details.
init	A (pre-)Lexis object representing the initial state of the persons whose trajectories through the multiple states we want to simulate. Must have attributes "time.scales" and "time.since" — see details. Duplicate values of lex.id are not sensible and not accepted.
N	Numeric. How many persons should be simulated. N persons with covariate configuration of each row in init will be simulated. Either a scalar or a vector of length nrow(init).
lex.id	Vector of ids of the simulated persons. Useful when simulating in chunks.
t.range	Numerical scalar. The range of time over which to compute the cumulative rates when simulating. Simulted times beyond this will result in an obervation censored at t.range after entry.
n.int	Number of intervals to use when computing (cumulative) rates.
time.pts	Numerical vector of times since start. Cumulative rates for transitions are computed at these times after stater and entry state. Simulation is only done till time max(time.pts) after start, where persons are censored. Must start with 0.
obj	A Lexis object.
from	The point on the time scale time.scale from which we start counting.
time.scale	The timescale to which from refer.
at	Time points (after from) where the number of persons in each state is to be computed.
nSt	A table obtained by nState.
perm	A permutation of columns used before cumulating row-wise and taking percentages.
x	An object of class pState, e.g. created by pState.
col	Colors for filling the areas between curves.
border	Colors for outline of the areas between curves.
xlab	Label on x-axis
ylim	Limits on y-axis
ylab	Label on y-axis
...	Further arguments passed on to plot.

Details

The simulation command simLexis is not defined as a method for Lexis objects, because the input is not a Lexis object, the Lexis-like object is merely representing a prevalent population and a specification of which variables that are timescales. The variables lex.dur and lex.Xst are ignored (and overwritten) if present. The core input is the list Tr giving the transitions.

The components of Tr represents the transition intensities between states. The transition from state A to B, say, is assumed stored in TrAB. Thus names of the elements of Tr are names of transient states, and the names of the elements of each these are the names of states reachable from the corresponding transient state.

The transition intensities are assumed modelled by either a glm with Poisson family or a Cox-model. In both cases the timescale(s) in the model must be using the names fo the timescales in a Lexis object representng the follow-up in a cohort, and the risk time must be taken from the variable lex.dur — see the example.

Alternatively, an element in Tr could be a function that from a data frame produces transition rates, or specifically cumulative transition rates over intervals of length lex.dur.

The pre-Lexis object init must contain values of all variables used in any of the objects in Tr, as well as all timescales - even those not used in the models. Moreover, the attributes time.scales and time.since must be present. The attribute time.since is a character vector of the same length as time.scales and an element has value "A" if the corresponding time scale is defined as "time since entry into state A", otherwise the value is "". If not present it will be set to a vector of ""s, which is only OK if no time scales are defined as time since entry to a state.

Note that the variables pre-Lexis object init must have the same mode and class as in the dataset used for fitting the models — hence the indexing of rows by brackets in the assignment of values used in the example below - this way the variables have their attributes preserved; using init[,"var"] <- or init$var <- replaces the variable, whereas init[1:4,"var"] <- or init$var[1:4] <- replaces values only and prevents you from entering non-existing factor levels etc.

The function Lexis automatically generates an attribute time.since, and cutLexis updates it when new time scales are defined. Hence, the simplest way of defining a initial pre-Lexis object representing a current state of a (set of) persons to be followed through a multistate model is to take NULL rows of an existing Lexis object (normally the one used for estimation), and so ensuring that all relevant attributes and state levels are properly defined. See the example code.

The prevalence function nState computes the distribution of individuals in different states at pre-specified times. Only sensible for a simulated Lexis object. The function pState takes a matrix as output by nState and computes the row-wise cumulative probabilities across states, and leaves an object of class pState, suitable for plotting.

Value

simLexis returns a Lexis object representing the experience of a population starting as init followed through the states according to the transitions in Tr.

The function nState returns a table of persons classified by states at each of the times in at. Note that this function can easily produce meaningless results, for example if applied to a Lexis object not created by simulation. If you apply it to a Lexis object generated by simLexis, you must make sure that you start (from) the point where you started the simulation on the correct timescale, and you will get funny results if you try to tabulate beyond the censoring time for the simulation. The resulting object has class "table".

The result from using pState on the result from nState has class c("pState","matrix").

Author(s)

Bendix Carstensen, http://bendixcarstensen.com.

See Also

Lexis, cutLexis, splitLexis

Examples

```
data(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[runif(nrow(DMlate))<0.1,] )
# Split follow-up at insulin, introduce a new timescale,
# and split non-precursor states
dmi <- cutLexis( dml, cut = dml$doins,
                      pre = "DM",
                new.state = "Ins",
                new.scale = "t.Ins",
             split.states = TRUE )
# Split the follow in 1-year intervals for modelling
Si <- splitLexis( dmi, 0:30/2, "DMdur" )
# Define knots
nk <- 4
( ai.kn <- with( subset(Si,lex.Xst=="Ins"),
                 quantile( Age+lex.dur, probs=(1:nk-0.5)/nk ) ) )
( ad.kn <- with( subset(Si,lex.Xst=="Dead"),
                 quantile( Age+lex.dur, probs=(1:nk-0.5)/nk ) ) )
( di.kn <- with( subset(Si,lex.Xst=="Ins"),
                 quantile( DMdur+lex.dur, probs=(1:nk-0.5)/nk ) ) )
( dd.kn <- with( subset(Si,lex.Xst=="Dead"),
                 quantile( DMdur+lex.dur, probs=(1:nk-0.5)/nk ) ) )
( td.kn <- with( subset(Si,lex.Xst=="Dead(Ins)"),
                 quantile( t.Ins+lex.dur, probs=(1:nk-0.5)/nk ) ) )

# Fit Poisson models to transition rates
library( splines )
DM.Ins <- glm( (lex.Xst=="Ins") ~ Ns( Age  , knots=ai.kn ) +
                                  Ns( DMdur, knots=di.kn ) +
                                  I(Per-2000) + sex,
               family=poisson, offset=log(lex.dur),
               data = subset(Si,lex.Cst=="DM") )
DM.Dead <- glm( (lex.Xst=="Dead") ~ Ns( Age  , knots=ad.kn ) +
                                    Ns( DMdur, knots=dd.kn ) +
                                    I(Per-2000) + sex,
               family=poisson, offset=log(lex.dur),
               data = subset(Si,lex.Cst=="DM") )
Ins.Dead <- glm( (lex.Xst=="Dead(Ins)") ~ Ns( Age  , knots=ad.kn ) +
                                          Ns( DMdur, knots=dd.kn ) +
                                          Ns( t.Ins, knots=td.kn ) +
                                          I(Per-2000) + sex,
               family=poisson, offset=log(lex.dur),
               data = subset(Si,lex.Cst=="Ins") )

# Stuff the models into an object representing the transitions
```

```
Tr <- list( "DM" = list( "Ins"       = DM.Ins,
                         "Dead"      = DM.Dead  ),
           "Ins" = list( "Dead(Ins)" = Ins.Dead ) )
lapply( Tr, names )

# Define an initial object - note the subsetting that ensures that
# all attributes are carried over
ini <- Si[1,1:9][-1,]
ini[1:2,"lex.Cst"] <- "DM"
ini[1:2,"Per"] <- 1995
ini[1:2,"Age"] <- 60
ini[1:2,"DMdur"] <- 5
ini[1:2,"sex"] <- c("M","F")
str(ini)

# Simulate 200 of each sex using the estimated models in Tr
simL <- simLexis( Tr, ini, time.pts=seq(0,11,0.5), N=200 )
summary( simL )

# Find the number of persons in each state at a set of times.
# Note that the times are shirter than the time-span simulated.
nSt <- nState( subset(simL,sex=="M"),
               at=seq(0,10,0.1), from=1995, time.scale="Per" )
nSt

# Show the cumulative prevalences in a different order than that of the
# state-level ordering and plot them using all defaults
pp <- pState( nSt, perm=c(1,2,4,3) )
head( pp )
plot( pp )

# A more useful set-up of the graph
clr <- c("orange2","forestgreen")
par( las=1 )
plot( pp, col=clr[c(2,1,1,2)] )
lines( as.numeric(rownames(pp)), pp[,2], lwd=2 )
mtext( "60 year old male, diagnosed 1995", side=3, line=2.5, adj=0 )
mtext( "Survival curve", side=3, line=1.5, adj=0 )
mtext( "DM, no insulin   DM, Insulin", side=3, line=0.5, adj=0, col=clr[1] )
mtext( "DM, no insulin", side=3, line=0.5, adj=0, col=clr[2] )
axis( side=4 )

# Using a Cox-model for the mortality rates assuming the two mortality
# rates to be proportional:
# When we fit a Cox-model, lex.dur must be used in the Surv() function,
# and the I() constrction must be used when specifying intermediate
# states as covariates, since factors with levels not present in the
# data will create NAs in the parameter vector returned by coxph, which
# in return will crash the simulation machinery.
library( survival )
Cox.Dead <- coxph( Surv( DMdur, DMdur+lex.dur,
                         lex.Xst %in% c("Dead(Ins)","Dead")) ~
                   Ns( Age-DMdur, knots=ad.kn ) +
```

```
                  I(lex.Cst=="Ins") +
                  I(Per-2000) + sex,
               data = Si )
Cr <- list( "DM" = list( "Ins"       = DM.Ins,
                         "Dead"      = Cox.Dead  ),
          "Ins" = list( "Dead(Ins)" = Cox.Dead ) )
simL <- simLexis( Cr, ini, time.pts=seq(0,11,0.2), N=200 )
summary( simL )
nSt <- nState( subset(simL,sex=="M"),
               at=seq(0,10,0.2), from=1995, time.scale="Per" )
pp <- pState( nSt, perm=c(1,2,4,3) )
plot( pp )
```

`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, mcutLexis, 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 transition intensities simultaneously.

Usage

```
## S3 method for class 'Lexis'
stack(x, ...)
tmat( x, ... )
## S3 method for class 'Lexis'
tmat(x, Y=FALSE, mode = "numeric", ...)
```

Arguments

x	A Lexis object.
Y	Logical. Should the risk time be put in the diagonal? This is a facility which is used by boxes.Lexis.
mode	Should the matrix be returned as a numeric matrix with NAs at unused places or (mode="logical") as a logical matrix with FALSE on the diagonal.
...	Not used.

Value

tmat.Lexis returns a square transition matrix, classified by the levels of lex.Cst and lex.Xst, for every transition occurring the entry is the number of transitions occurring and NA in all oter entries. If Y=TRUE, the diagonal will contain the risk time in each of the states.

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 class stacked.Lexis, and inherits the attributes time.scales and breaks from the Lexis object, and so function timeBand applies to a stacked.Lexis object too.

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 named in the factor 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, <b@bxc.dk>, http://bendixcarstensen.com

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) ) )
```

---

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, percent, and sd.

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, sd.

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`

`steno2` *Clinical trial: Steno2 baseline and follow-up.*

Description

Steno-2 was a clinical trial conducted at Steno Diabetes Center 1993-2001. The intervention was intensified treatment versus conventional treatment of diabetes patients with micro-albuminuria. The datsets here concern the extended follow-up of the trial population till 2015. Three files are provided: `steno2` with one record per person, `st2clin` with one record per clinical visit and `st2alb` with one record per transition between states of albuminuria.

These dataset are entirely simulated, but designed to give approximately the same results as the original.

Usage

```
data("steno2")
       data("st2clin")
       data("st2alb")
```

Format

`steno2` is a data frame with 160 observations on the following 14 variables:

`id` person id, numeric

`allo` Original trial allocation, a factor with levels `Int` `Conv`

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

`baseCVD` 0/1 indicator of preexisting CVD at baseline

`deathCVD` 0/1 indicator whether cause of death was CVD

`doBth` Date of birth, a Date

`doDM` Date of diabetes diagnosis, a Date

`doBase` Date of entry to study, a Date

`doCVD1` Date of 1st CVD event, a Date

`doCVD2` Date of 2nd CVD event, a Date

`doCVD3` Date of 3rd CVD event, a Date

`doESRD` Date of end stage renal disease, a Date

`doEnd` Date of exit from follow-up, a Date

`doDth` Date of death, a Date

`st2clin` is data frame with 750 observations on clinical measurements at different clinical visits:

`id` person id, numeric

`doV` Date of clinical visit, a Date

`a1c` Glycosylated hemoglobin, mmol/mol

`chol` Total cholesterol, mg/mol

`crea` Creatinine, mg/mol

`st2alb` is data frame with 307 observations of changes in complication (albuminuria) state

`id` person id, numeric

`doTr` Date of transition, a Date

`state` State of albuminuria, factor with levels `Norm`, `Mic`, `Mac`. All persons begin in the state `Micro`-albuminuria.

Details

The data are not the original; all values of measurements and dates have been randomly perturbed, to prevent identifiability of individuals. Analysis of these data will give only (very) approximately the same results as in the published article, and only some of the aspects of data are included.

References

P. Gaede, J. Oellgaard, B. Carstensen, P. Rossing, H. Lund-Andersen, H. H. Parving & O. Pedersen: Years of life gained by multifactorial intervention in patients with type 2 diabetes mellitus and microalbuminuria: 21 years follow-up on the Steno-2 randomised trial. Diabetologia (2016), 59, pp 2298-2307

Examples

```
data(steno2)
data(st2alb)
L2 <- Lexis( entry = list(per = doBase,
                          age = doBase - doBth),
             exit = list(per = doEnd),
```

```
       exit.status = factor(deathCVD + !is.na(doDth),
                            labels=c("Mic","D(oth)","D(CVD)")),
                id = id,
              data = cal.yr(steno2) )
summary(L2)
#
# Cut at intermediate transitions
cut2 <- data.frame(lex.id = st2alb$id,
                       cut = cal.yr(st2alb$do),
                 new.state = st2alb$state)
L3 <- rcutLexis(L2, cut2)
summary(L3)
#
# no direct transitions Mic <-> Mac allowed, so put a cut in between:
dd <- subset(L3, (lex.Cst == "Mac" & lex.Xst =="Norm") |
                 (lex.Cst =="Norm" & lex.Xst == "Mac"))
# artificial visits to the middle state Mic:
cut3 <- data.frame( lex.id = dd$lex.id,
                        cut = dd$per + dd$lex.dur/2,
                  new.state = "Mic")
L4 <- rcutLexis(L3, cut3)
summary(L4)
#
# Show all transitions
boxes(L4, boxpos = list(x = c(15,15,15,85,85),
                        y = c(50,15,85,25,75)),
          show.BE = TRUE, scale.R = 1000,
          cex=0.8, pos.arr=0.7, font=1, font.arr=1)
```

`subset.Lexis` *Subsetting Lexis (and stacked.Lexis) objects*

Description

Return subsets of Lexis objects which meet conditions

Usage

```
## S3 method for class 'Lexis'
subset(x, ...)
## S3 method for class 'Lexis'
x[...]
## S3 method for class 'stacked.Lexis'
subset(x, ...)
```

Arguments

`x`	an object of class `Lexis`
`...`	additional arguments to be passed to `subset.data.frame`. This will normally be some logical expression selecting a subset of the rows. For details see `subset.data.frame`.

Details

The subset method for `Lexis` objects works exactly as the method for data frames. So does the "[" method. The special methods are needed in order to propagate the Lexis-specific attributes.

The method for `stacked.Lexis` objects also shrinks the set of levels for `lex.Cst` and `lex.Xst` to those actually occurring in the resulting data frame.

Value

A `Lexis` object with selected rows and columns.

Author(s)

Martyn Plummer

See Also

`Lexis`, `merge.Lexis`, `bootLexis`

`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,
                              by = NULL,
                           Rates = FALSE,
                      timeScales = FALSE, ...)
  ## S3 method for class 'summary.Lexis'
print(x, ..., digits = 2)
```

Arguments

`object`	A 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.

by	Character vector of name(s) of variable(s) in object. Used to give a separate summaries for subsets of object. If longer than than 1, the interaction between the variables is used to stratify the summary. It is also possible to supply a vector of length nrow(object), in which case the distinct values of this will be used to stratify the summary.
Rates	Should a component with transition rates be returned (and printed) too?
timeScales	Should the names of the timescales and the indication of since which entry also be returned?
x	A Lexis or summary.Lexis object.
digits	Number of digits after the decimal separator used when printing the summary.
...	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 spent time, and columns classified by states to which persons transit. The Transitions contains number of transitions and has 4 extra columns with number of records, total number of events, total risk time and number of person contributing attached. The Rates contains the transitions rates.

If the argument Rates is FALSE (the default), then only the first component of the list is returned.

If mat is FALSE a list is returned, possibly of length 1.

Author(s)

Bendix Carstensen, http://bendixcarstensen.com

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 )
# More detailed summary, by exposure level, and rates per 100 PY
summary(nic, by = nic$exposure > 5,
          Rates = TRUE,
          scale = 100)
```

`Termplot` *A wrapper for* `termplot` *that optionally (but by default) exponentiates terms, and plot them on a common log-scale. Also scales x-axes to the same physical scale.*

Description

The function uses `termplot` to extract terms from a model with, say, spline, terms, including the standard errors, computes confidence intervals and transform these to the rate / rate-ratio scale. Thus the default use is for models on the log-scale such as Poisson-regression models. The function produces a plot with panels side-by-side, one panel per term, and returns the

Usage

```
Termplot( obj,
          plot = TRUE,
          xlab = NULL,
          ylab = NULL,
           xeq = TRUE,
          yshr = 1,
         alpha = 0.05,
         terms = NULL,
        max.pt = NULL )
```

Arguments

`obj`	An object with a `terms`-method — for details the the documentation for `termplot`.
`plot`	Should a plot be produced?
`xlab`	Labels for the `x`-axes. Defaults to the names of the terms.
`ylab`	Labels for the `x`-axes. Defaults to blank.
`xeq`	Should the units all all plots have the same physical scale for the `x`-axes).
`yshr`	Shrinking of `y`-axis. By default, the `y`-axes have an extent that accommodates the entire range of confidence intervals. This is a shrinking parameter for the `y`-axes, setting it to less than 1 will lose a bit of the confidence limits on some of the panels.
`alpha`	1 minus the confidence level for computing confidence intervals
`terms`	Which terms should be reported. Passed on to `termplot` and eventually `predict`.
`max.pt`	The maximal number of points in which to report the terms. If `NULL` all unique points from the analysis dataset are reported for each term (this is a feature of `termplot`).

Value

A list with one component per term in the model object `obj`, each component is a 4-column matrix with x as the first column, and 3 columns with estimae and lower and upper confidence limit.

Author(s)

Bendix Cartensen

See Also

Ns, termplot

Examples

```
# Get the diabetes data and set up as Lexis object
data(DMlate)
DMlate <- DMlate[sample(1:nrow(DMlate),500),]
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 in 1-year age intervals
dms <- splitLexis( dml, time.scale="Age", breaks=0:100 )

# Model with 6 knots for both age and period
n.kn <- 6
# Model age-specific rates with period referenced to 2004
( a.kn <- with( subset(dms,lex.Xst=="Dead"),
                quantile( Age+lex.dur, probs=(1:n.kn-0.5)/n.kn ) ) )
( p.kn <- with( subset(dms,lex.Xst=="Dead"),
                quantile( Per+lex.dur, probs=(1:n.kn-0.5)/n.kn ) ) )
m2 <- glm( lex.Xst=="Dead" ~ -1 +
                             Ns( Age, kn=a.kn, intercept=TRUE ) +
                             Ns( Per, kn=p.kn, ref=2004 ),
           offset = log( lex.dur ), family=poisson, data=dms )

# Finally we can plot the two effects:
Termplot( m2, yshr=0.9 )
```

---

testisDK *Testis cancer incidence in Denmark, 1943–1996*

Description

Number of testiscancer cases and male person-years in the Danish population 1943–1996

Usage

```
data(testisDK)
```

Format

A data frame with 4860 observations on the following 4 variables.

`A` Age class, 0,1,2,...,89

`P` Year, 1943,...,1996

`D` Number of testis cancer cases

`Y` Person years

Source

The Danish Cancer Registry

Examples

```
data(testisDK)
head(testisDK)
```

`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 some earlier date.

`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 + chola + hmang`

`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

Functions to get the names and type of the time scales of a `Lexis` object.

Usage

```
timeScales(x)
timeSince(x)
tsNA20(x, all.scales = FALSE)
lexVars(x)
```

Arguments

x	an object of class Lexis.
all.scales	Should NAs in all timescales be replaced by 0? If FALSE (the default) only timescales defined as time since entry to a state get NAs replaced by 0s

Value

timeScales returns a character vector containing the names of the variables in x that represent the time scales. Extracted from the time.scales attribute of the object.

timeSince returns a named character vector, the names being the names of the timescales and the content being the names of the states to which the corresponding timescale is defined as time since entry. For those time scales that are not defined as such an empty string is used. Hence, if none of the timescales are defined as time since entry to a state timeSince will return a vector of empty strings.

lexVars returns a character vector containing the names if the Lexsi-specific variables of x; those starting with lex. and the names of the timescales.

Author(s)

Martyn Plummer, Bendix Carstensen

See Also

Lexis, splitLexis

---

transform.Lexis *Transform a Lexis (or stacked.Lexis) object*

Description

Modify a Lexis object.

Usage

```
## S3 method for class 'Lexis'
factorize(x, ..., verbose = FALSE)
## S3 method for class 'Lexis'
Relevel(x, ref, ...)
## S3 method for class 'Lexis'
levels(x)
## S3 method for class 'Lexis'
transform(`_data`, ...)
## S3 method for class 'stacked.Lexis'
transform(`_data`, ...)
 order.Lexis(x)
  orderLexis(x)
   sortLexis(x)
```

Arguments

_data	an object of class Lexis.
x	an object of class Lexis.
ref	New names (or order) of the factor levels (states) for lex.Cst and lex.Xst. Can be a list, in which case some levels are collapsed, see the documentation for Relevel. No sanity check for the latter type of operation is undertaken.
...	Additional arguments to be passed to transform.data.frame, Relevel.factor.
verbose	Logical. Should a list of new levels be printed?

Details

The transform method for Lexis objects works exactly as the method for data frames, but keeps the Lexis attributes.

factorize transforms the variables lex.Cst and lex.Xst to factors with identical sets of levels.

Relevel does the same as Relevel.factor, but for both the factors lex.Cst and lex.Xst in x. lex.Cst and lex.Xst must be factors with the same levels. They can be made so by factorize.

If ref is an integer or character vector, the levels of lex.Cst and lex.Xst are permuted to match the order of ref.

If ref is NULL, as when for example the argument is not passed to the function, the returned object have levels of lex.Cst, lex.Xst (and for stacked.Lexis objects lex.Tr) shaved down to the actually occurring values; that is, empty levels are discarded.

order.Lexis returns the order of the rows in a Lexis object to sort it by ()lex.id,ts), where ts is a timescale in the Lexis object with no NAs. orderLexis is just a synonym.

sortLexis returns the Lexis object sorted by (lex.id, ts) where ts is one of the timeScales with no NAs.

Value

A transformed Lexis object.

The function levels returns the names of the states (levels of the factors lex.Cst and lex.Xst.

Author(s)

Martyn Plummer, Bendix Carstensen

See Also

Lexis, merge.Lexis, subset.Lexis, subset.stacked.Lexis, Relevel, transient, absorbing

Examples

```
data( nickel )
nic <- Lexis( data = nickel,
                id = id,
             entry = list(age = agein),
              exit = list(age = ageout,
```

```
                          cal = ageout+dob,
                          tfh = ageout-age1st),
# Lung cancer deaths end as 2 and other deaths as 1
       exit.status = factor((icd > 0) + (icd %in% c(162,163)),
                            labels = c("Alive","Dead","Lung") ) )
str( nic )
levels( nic )
nit <- transform( nic, cumex = exposure * (agein - age1st) )
str( nit )

# It is still a Lexis object!
summary(nic)

# change order of levels
nix <- Relevel(nic, c("Alive", "Lung", "Dead"))
summary(nix)

# change names of levels
niw <- Relevel(nix, list("Alive" = 1, "Pulm" = "Lung", "Mort" = "Dead"))
summary(niw)
boxes(niw, boxpos = TRUE)

# combine levels
niz <- Relevel(niw, list("Alive", c("Pulm", "Mort")), coll=" \n& ")
summary(niz)
par( new = TRUE )
boxes(niz, boxpos = TRUE)

#stack Lexis object
siw <- stack(niw)
str(siw)
```

---

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
```

---

unLexis *Remove Lexis attributes from a* Lexis *object.*

Description

Removes the Lexis attributes, including the class Lexis from a Lexis object.

Usage

```
unLexis(Lx)
```

Arguments

Lx	A Lexis object

Value

The input object with "Lexis" removed from the class attribute.

Author(s)

Bendix Carstensen

See Also

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 <- cal.yr(xcoh, format="%d/%m/%Y", wh=2:4)
# xcoh <- cal.yr(xcoh, format="%d/%m/%Y", wh=2:4)

# Define a 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)
summary(Lcoh)
try(summary(unLexis(Lcoh)))
```

---

xcutLexis *Cut follow-up in a Lexis object by event date(s) while preserving the original states.*

Description

With a multistate Lexis object we might want a classification of the follow-up according to a set of events independent of the states in the Lexis object, while keeping the original states. This is essentially expanding the states with a cross-classification of the original states and a set of new ones, hence the "x". Limited to 1 event time per person.

Usage

```
xcutLexis(Lx, cut, timescale = 1, sep = ".")
```

Arguments

Lx	A Lexis object
cut	A data frame with at most one row per lex.id, with columns: • lex.id: person-id referring to lex.id in Lx • cut, a numeric; times of event om the time scale timescale • new.state, character or factor; name of the new state occurring at time cut
timescale	The timescale that cut refers to. Numeric or character.
sep	Character; string used in forming new state names by joining original state names in Lx and state names in cut$new.states

Details

The function was motivated by a follow-up through states of multimorbidity with state names 0morb, 1morb, 2morb, etc. where we wanted a subdivision of each state by the presence of T2 diabetes, resulting in states 0morb, 1morb, 2morb, 0morb.T2, 1morb.T2, 2morb.T2 etc.

At most 1 transition per person is allowed in cut.

Value

A Lexis object with states as in Lx plus states named by concatenating Lx state names with names in cut$new.state

Author(s)

Bendix Carstensen, Steno Diabetes Center Copenhagen, <b@bxc.dk>, http://bendixcarstensen.com

See Also

cutLexis, mcutLexis, rcutLexis, addCov.Lexis, addDrug.Lexis, splitLexis, Lexis, summary.Lexis, timeSince, boxes.Lexis

Examples

```
# construct a Lexis object and show states and transitions
example(DMlate)
levels(dmi)
#
# show transitions between states in dmi
boxes(dmi, boxpos = list(x = rep(50, 4),
                         y = c(65, 35, 95, 5)),
           scale.R = 1000,
           show.BE = TRUE,
           cex = 1)
```

```
#
# randomly generated intermediate events X and Y
set.seed(1952)
cutXY <- data.frame(lex.id = unique(dmi$lex.id),          # one row per id
                        cut = runif(nid(dmi), 1995, 2008), # event dates
                new.state = sample(c("X","Y"),           # event types
                                   nid(dmi),    # only 4 in 10 has an event
                                   repl = TRUE))[runif(nid(dmi)) < 0.4, ]
#
# cut at these event dates but also keep original states
Lxy <- xcutLexis(dmi, cutXY)
levels(Lxy)
#
# reorder the levels for easier specification of box places
Lxy <- Relevel(Lxy, as.vector(t(outer(levels(dmi),
                                      c("", ".X", ".Y"),
                                      paste0))))
cbind(levels(Lxy))
summary(Lxy)
#
# resulting transitions between boxes
boxes(Lxy, boxpos = list(x = rep(c(50, 15, 85), 4),
                         y = rep(c(65,35,95,5), each = 3)),
           scale.R = 1000,
           show.BE = TRUE,
           cex = 1)
```

`Y.dk` *Population risk time in Denmark*

Description

Risk time (person-years) in the Danish population, classified by sex, age, period and date of birth in 1-year classes. This corresponds to triangles in a Lexis diagram.

Usage

```
data(Y.dk)
```

Format

A data frame with 13860 observations on the following 6 variables.

`sex` Sex. 1:males, 2:females

`A` One-year age class

`P` Period

`C` Birth cohort

`Y` Person-years

`upper` Indicator of upper triangle in the Lexis diagram

Details

The risk time is computed from the population size figures in N.dk, using the formulae devised in: B. Carstensen: Age-period-cohort models for the Lexis diagram. Statistics in Medicine, 10; 26(15):3018-45, 2007.

Source

http://www.statistikbanken.dk/statbank5a/SelectTable/omrade0.asp?SubjectCode=02&PLanguage=1&ShowNews=OFF

Examples

```
data(Y.dk)
str(Y.dk)
# Compute mean age, period for the triangles
attach( Y.dk )
age <- A + (1+upper)/3
per <- P + (2-upper)/3
# Plot a Lexis diagram
library( Epi )
Lexis.diagram( age=c(0,10), date=c(1990,2000), coh.grid=TRUE, int=1 )
box()
# Print the person-years for males there
text( per[sex==1], age[sex==1],
      formatC( Y[sex==1]/1000, format="f", digits=1 ) )
```

Index

∗ **Data**
N2Y, 134
∗ **Lexis checks**
overlap.Lexis, 143
∗ **Lexis**
cutLexis, 60
paths.Lexis, 144
xcutLexis, 195
∗ **Multistate model**
paths.Lexis, 144
∗ **Multistate paths**
paths.Lexis, 144
∗ **aplot**
legendbox, 103
plot.Lexis, 148
∗ **array**
detrend, 63
merge.Lexis, 125
pctab, 146
projection.ip, 158
∗ **attributes**
lls, 117
∗ **attribute**
timeBand, 189
timeScales, 190
∗ **category**
stat.table, 178
stattable.funs, 180
∗ **chron**
cal.yr, 37
∗ **color**
matshade, 121
∗ **data manipulation**
gen.exp, 87
unLexis, 194
∗ **datagen**
ccwc, 40
∗ **datasets**
B.dk, 24
bdendo, 25
births, 26
blcaIT, 27
BrCa, 35
brv, 36
diet, 64
DMconv, 65
DMepi, 66
DMlate, 67
ewrates, 77
gmortDK, 91
hivDK, 93
lep, 104
lungDK, 118
M.dk, 119
mortDK, 132
N.dk, 133
nickel, 139
occup, 142
pr, 158
S.typh, 169
steno2, 181
testisDK, 187
thoro, 188
Y.dk, 197
∗ **design**
contr.cum, 57
∗ **distribution**
ci.pd, 54
∗ **dplot**
Lexis.diagram, 108
Lexis.lines, 111
Life.lines, 115
∗ **hplot**
apc.frame, 18
apc.lines, 22
boxes.MS, 29
Lexis.diagram, 108
Lexis.lines, 111

pc.lines, 145
plot.apc, 147
plot.Lexis, 148
plotEst, 153
rateplot, 159
Termplot, 186
∗ **htest**
ci.pd, 54
mh, 126
ROC, 167
twoby2, 193
∗ **iplot**
boxes.MS, 29
∗ **iteration**
stat.table, 178
stattable.funs, 180
∗ **linear predictor**
ci.eta, 47
∗ **manip**
addDrug, 8
adtte2Lexis {Epi}, 11
bootLexis, 27
cal.yr, 37
cbind.Lexis, 39
Lexis, 105
Lexis2msm, 112
LexisScarf, 113
lgrep, 114
Life.lines, 115
mat2pol, 120
merge.Lexis, 125
ncut, 137
nice, 138
overlap.Lexis, 143
pctab, 146
Relevel, 164
rm.tr, 166
ROC, 167
splitLexis, 175
subset.Lexis, 183
transform.Lexis, 191
∗ **math**
in.span, 96
∗ **methods**
pctab, 146
∗ **models**
AaJ, 4
apc.fit, 13
apc.LCa, 20
ci.cum, 45
ci.eta, 47
ci.lin, 48
clogistic, 55
contr.cum, 57
effx, 69
effx.match, 71
expand.data, 78
fit.add, 79
fit.baseline, 80
fit.mult, 81
harm, 92
Icens, 94
LCa.fit, 98
mod.Lexis, 128
plotEst, 153
plotevent, 155
poisreg, 156
∗ **multistate models**
cutLexis, 60
xcutLexis, 195
∗ **prediction frame**
ci.eta, 47
∗ **prediction**
ci.eta, 47
∗ **regression**
apc.fit, 13
apc.LCa, 20
ci.Crisk, 42
ci.cum, 45
ci.eta, 47
ci.lin, 48
effx, 69
effx.match, 71
expand.data, 78
fit.add, 79
fit.baseline, 80
fit.mult, 81
float, 82
ftrend, 85
harm, 92
Icens, 94
LCa.fit, 98
Ns, 139
plotevent, 155
∗ **state transition**
cutLexis, 60

xcutLexis, 195
∗ **survival**
addCov, 5
addDrug, 8
boxes.MS, 29
cbind.Lexis, 39
crr.Lexis, 58
cutLexis, 60
entry.Lexis, 72
erl, 74
expand.data, 78
fit.add, 79
fit.baseline, 80
fit.mult, 81
foreign.Lexis, 84
Icens, 94
Lexis, 105
Lexis2msm, 112
mcutLexis, 123
plotevent, 155
rcutLexis, 163
simLexis, 170
stack.Lexis, 177
summary.Lexis, 184
xcutLexis, 195
∗ **transition rates**
cutLexis, 60
xcutLexis, 195
∗ **ts**
entry.Lexis, 72
∗ **univar**
twoby2, 193
[.Lexis *(*subset.Lexis*)*, 183

AaJ, 4
absorbing, *107*, *131*, *192*
absorbing *(*entry.Lexis*)*, 72
addCov, 5
addCov.Lexis, *9, 10*, *62*, *89*, *107*, *124*, *131*, *163*, *196*
addDrug, 8
addDrug.Lexis, *62*, *124*, *196*
addmargins, *147*
adtte2Lexis *(*adtte2Lexis {Epi}*)*, 11
adtte2Lexis {Epi}, 11
after *(*entry.Lexis*)*, 72
apc.fit, 13, *19–22*, *24*, *102*, *135*, *146–148*
apc.frame, *17*, 18, *22–24*, *145–148*, *162*
apc.LCa, *17*, 20, *102*
apc.lines, *16–19*, 22, *147, 148*
apc.plot, *16, 17*, *19*, *24*
apc.plot *(*plot.apc*)*, 147
Aplot *(*rateplot*)*, 159
arrows, *33*
as.Date.cal.yr *(*cal.yr*)*, 37

B.dk, 24
bdendo, 25
bdendo11 *(*bdendo*)*, 25
before *(*entry.Lexis*)*, 72
binom.test, *55*
births, 26
blcaIT, 27
bootLexis, 27, *184*
boxarr *(*boxes.MS*)*, 29
boxes *(*boxes.MS*)*, 29
boxes.Lexis, *62*, *104*, *177*, *196*
boxes.matrix, *20*
boxes.MS, 29
BrCa, 35
breaks *(*timeBand*)*, 189
brv, 36

cal.yr, 37, *107*
cbind.Lexis, 39, *107*
ccwc, 40
ci.Crisk, *5*, 42
ci.cum, 44, *52*, *76*
ci.eta, 47, *52*
ci.exp *(*ci.lin*)*, 48
ci.lin, *45, 46*, *48*, 48, *154*
ci.mat *(*ci.lin*)*, 48
ci.pd, 54
ci.pred, *46*, *130*
ci.pred *(*ci.lin*)*, 48
ci.ratio *(*ci.lin*)*, 48
ci.surv, *43*, *52*
ci.surv *(*ci.cum*)*, 45
clear *(*lls*)*, 117
clogistic, 55
coarse.Lexis *(*addDrug*)*, 8
contr.2nd *(*contr.cum*)*, 57
contr.cum, 57
contr.diff *(*contr.cum*)*, 57
contr.orth *(*contr.cum*)*, 57
contr.treatment, *58*
count, *179*
count *(*stattable.funs*)*, 180

countLexis (cutLexis), 60
coxph, *128*, *130*
coxph.Lexis (mod.Lexis), 128
coxphLexis (mod.Lexis), 128
cp.lines (pc.lines), 145
cp.matlines (pc.lines), 145
cp.matpoints (pc.lines), 145
cp.matshade (pc.lines), 145
cp.points, *148*
cp.points (pc.lines), 145
Cplot (rateplot), 159
crr, *58*, *59*
crr.Lexis, 58
cut, *137*
cutLexis, *5*, *6*, *10*, 60, *89*, *107*, *123*, *124*, *131*, *145*, *163*, *172*, *173*, *176*, *178*, *196*

Date, *37*, *38*, *64*, *107*
date, *38*
DateTimeClasses, *38*
dbox (boxes.MS), 29
decurve (detrend), 63
det, *97*
detrend, 63, *97*, *159*
diet, 64
DMconv, 65
DMepi, 66
DMlate, 67
DMrand (DMlate), 67
dur, *107*
dur (entry.Lexis), 72

effx, 69
effx.match, 71
entry, *107*
entry (entry.Lexis), 72
entry.Lexis, 72
Epi, 73
Epi-package (Epi), 73
erl, 74
erl1 (erl), 74
etm, *85*
etm (foreign.Lexis), 84
ewrates, 77, *139*
exit, *107*
exit (entry.Lexis), 72
expand.data, 78, *79–81*

factorize (transform.Lexis), 191
family, *157*
fgrep (lgrep), 114
fillarr (boxes.MS), 29
findInterval, *137*
fit.add, *79*, 79, *80*, *82*, *95*
fit.baseline, 80
fit.mult, *79*, *80*, 81, *95*
float, 82, *87*
foreign.Lexis, 84
ftrend, *83*, 85

gam, *128*, *130*
gam.Lexis (mod.Lexis), 128
gamLexis (mod.Lexis), 128
gen.exp, *10*, 87
glm, *56*, *80*, *128*, *130*, *157*
glm.Lexis (mod.Lexis), 128
glmLexis (mod.Lexis), 128
gmortDK, 91, *133*, *188*, *189*
grep, *115*

harm, 92
hivDK, 93

Icens, *79*, *81*, *82*, 94, *156*
id.span (in.span), 96
idSpan (in.span), 96
in.span, 96
inSpan (in.span), 96
IQR, *179*

lca, *102*
LCa.fit, *17*, *20*, *21*, 98
legendbox, *33*, 103
lep, 104
levels.Lexis (transform.Lexis), 191
Lexis, *4–6*, *8–10*, *27*, *29*, *31*, *36*, *39*, *41*, *58*, *59*, *62*, *73*, *84*, *89*, 105, *110*, *113*, *124*, *127*, *129–131*, *143–145*, *149*, *163*, *166*, *171–173*, *177*, *178*, *184*, *190–192*, *194–196*
Lexis.diagram, 108, *112*, *116*
Lexis.lines, *110*, 111, *116*
Lexis2msm, 112
LexisScarf, 113
lexVars (timeScales), 190
lgrep, 114
Life.lines, *110*, *112*, 115
lines.apc, *146*, *148*

lines.apc *(apc.lines)*, 22
lines.Lexis *(plot.Lexis)*, 148
lines.pState *(simLexis)*, 170
linesEst *(plotEst)*, 153
lls, 117
ls, *117*
lungDK, 118

M.dk, 119
mat2pol, *43*, 120
matlines, *122*
matplot, *121*, *122*
matshade, *52*, 121
max, *179*
mcutLexis, *6*, *10*, *62*, *89*, *107*, 123, *131*, *145*, *163*, *176*, *196*
mean, *179*
median, *179*
merge.data.frame, *126*
merge.Lexis, *107*, 125, *184*, *192*
mh, 126
min, *179*
mod.Lexis, 128
mortDK, *92*, 132, *189*
msdata *(foreign.Lexis)*, 84
msprep, *85*

N.dk, 133, *198*
N2Y, 134
NArray, 136
ncut, 137
ngrep *(lgrep)*, 114
nice, 138
nickel, *77*, 139
nid, *112*, *145*
nid *(bootLexis)*, 27
Ns, 139, *187*
nState *(simLexis)*, 170

occup, 142, *149*
order.Lexis *(transform.Lexis)*, 191
orderLexis *(transform.Lexis)*, 191
overlap *(overlap.Lexis)*, 143
overlap.Lexis, 143

paths *(paths.Lexis)*, 144
paths.Lexis, 144
pc.lines, *24*, 145
pc.matlines *(pc.lines)*, 145
pc.matpoints *(pc.lines)*, 145
pc.matshade, *122*
pc.matshade *(pc.lines)*, 145
pc.points *(pc.lines)*, 145
pctab, 146
percent, *179*
percent *(stattable.funs)*, 180
plot, *152*
plot.apc, *146*, 147
plot.LCa *(LCa.fit)*, 98
plot.Lexis, *107*, *110*, 148
plot.pState *(simLexis)*, 170
plot.survfit, *152*
plotCIF, *43*, 150, *151*
plotEst, 153
plotevent, 155
points.Lexis *(plot.Lexis)*, 148
pointsEst *(plotEst)*, 153
poisreg, *130*, 156
POSIXct, *38*
POSIXlt, *38*
Pplot *(rateplot)*, 159
pr, 158
preceding, *130*
preceding *(entry.Lexis)*, 72
predict, *186*
predict.LCa *(LCa.fit)*, 98
pretty, *138*
print.floated *(float)*, 82
print.Icens *(Icens)*, 94
print.LCa *(LCa.fit)*, 98
print.Lexis *(Lexis)*, 105
print.mh *(mh)*, 126
print.stat.table *(stat.table)*, 178
print.summary.Lexis *(summary.Lexis)*, 184
projection.ip, *64*, 158
pState *(simLexis)*, 170
PY.ann *(plot.Lexis)*, 148

quantile, *179*

rateplot, 159
ratio, *179*
ratio *(stattable.funs)*, 180
rbind.Lexis, *107*
rbind.Lexis *(cbind.Lexis)*, 39
rcutLexis, *10*, *62*, *107*, *124*, *145*, 163, *196*
Relevel, 164, *166*, *192*
Relevel.factor, *192*

Relevel.Lexis, *28*, *165*
Relevel.Lexis *(*transform.Lexis*)*, 191
rm.tr, 166
ROC, 167

S.typh, 169
sd, *179*
show.apc.LCa *(*apc.LCa*)*, 20
simLexis, *43*, 170
sortLexis *(*transform.Lexis*)*, 191
splitLexis, *6*, *62*, *107*, *124*, *135*, *149*, *163*, *173*, 175, *178*, *191*, *196*
st2alb *(*steno2*)*, 181
st2clin *(*steno2*)*, 181
stack.Lexis, *84*, *85*, 177
stackedCIF, *151*
stackedCIF *(*plotCIF*)*, 150
stat.table, *73*, 178, *180*, *181*
stattable.funs, 180
status *(*entry.Lexis*)*, 72
steno2, 181
subset.data.frame, *183*
subset.Lexis, *28*, *39*, *107*, *126*, 183, *192*
subset.stacked.Lexis, *192*
subset.stacked.Lexis *(*subset.Lexis*)*, 183
succeeding *(*entry.Lexis*)*, 72
sum, *179*
summary.Icens *(*Icens*)*, 94
summary.LCa *(*LCa.fit*)*, 98
summary.Lexis, *62*, *107*, *176*, 184, *196*
surv1 *(*erl*)*, 74
surv2 *(*erl*)*, 74
survfit, *4*, *5*, *151*, *152*

table, *179*
tapply, *179*
tbox *(*boxes.MS*)*, 29
Termplot, 186
termplot, *186*
testisDK, 187
thinCol, *97*
thinCol *(*in.span*)*, 96
thoro, *92*, *133*, 188
timeBand, *107*, *176*, *177*, 189
timeScales, *6*, *107*, 190, *192*
timeSince, *62*, *196*
timeSince *(*timeScales*)*, 190
tmat *(*stack.Lexis*)*, 177
tmat.Lexis, *33*
transform.data.frame, *192*
transform.Lexis, *107*, 191
transform.stacked.Lexis *(*transform.Lexis*)*, 191
transient, *107*, *131*, *192*
transient *(*entry.Lexis*)*, 72
tsNA20 *(*timeScales*)*, 190
twoby2, *55*, 193

unLexis, *107*, 194

Wald *(*ci.lin*)*, 48
weighted.mean, *179*

xcutLexis, *62*, *124*, *145*, 195

Y.dk, *119*, 197
yll *(*erl*)*, 74

ZArray *(*NArray*)*, 136