Package: popEpi (via r-universe)

July 2, 2026

Title Functions for Epidemiological Analysis using Population Data

Version 0.5.0

Description Enables computation of epidemiological statistics, including those where counts or mortality rates of the reference population are used. Currently supported: excess hazard models (Dickman, Sloggett, Hills, and Hakulinen (2012) <doi:10.1002/sim.1597>), rates, mean survival times, relative/net survival (in particular the Ederer II (Ederer and Heise (1959)) and Pohar Perme (Pohar Perme, Stare, and Esteve (2012) <doi:10.1111/j.1541-0420.2011.01640.x>) estimators), and standardized incidence and mortality ratios, all of which can be easily adjusted for by covariates such as age. Fast splitting and aggregation of 'Lexis' objects (from package 'Epi') and other computations achieved using 'data.table'.

License MIT + file LICENSE

URL https://github.com/FinnishCancerRegistry/popEpi

BugReports https://github.com/FinnishCancerRegistry/popEpi/issues

Depends R (>= 3.5)

Imports data.table (>= 1.10.4), directadjusting (>= 0.6.0), Epi (>= 2.0), methods, splines, survival

Suggests knitr, mstate, relsurv, rmarkdown, roxygen2, testthat

VignetteBuilder knitr

ByteCompile true

Config/roxygen2/version 8.0.0

Encoding UTF-8

Language en-GB

LazyData true

NeedsCompilation no

Author Joonas Miettinen [cre, aut] (ORCID: <https://orcid.org/0000-0001-8624-6754>), Matti Rantanen [aut], Karri Seppa [ctb] (ORCID: <https://orcid.org/0000-0002-3847-8814>)

Maintainer Joonas Miettinen <joonas.miettinen@cancer.fi>

Repository https://cran.r-universe.dev

Date/Publication 2026-07-01 16:10:02 UTC

RemoteUrl https://github.com/cran/popEpi

RemoteRef HEAD

RemoteSha 1899a1b94217dd03a8b9ca9b56dea9be6bf8fa94

Contents

`adjust` *Adjust Estimates by Categorical Variables*

Description

This function is only intended to be used within a formula when supplied to e.g. `[survtab_ag]` and should not be used elsewhere.

Usage

```
adjust(...)
```

Arguments

`...`	variables to adjust by, e.g. `adjust(factor(v1), v2, v3)`

Value

Returns a list of promises of the variables supplied which can be evaluated.

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)

y ~ x + adjust(z)
```

---

aggre *Aggregation of split* Lexis *data*

Description

Aggregates a split Lexis object by given variables and / or expressions into a long-format table of person-years and transitions / end-points. Automatic aggregation over time scales by which data has been split if the respective time scales are mentioned in the aggregation argument to e.g. intervals of calendar time, follow-up time and/or age.

Usage

```
aggre(
  lex,
  by = NULL,
  type = c("unique", "full"),
  sum.values = NULL,
  subset = NULL,
  verbose = FALSE
)
```

Arguments

lex	a Lexis object split with e.g. [Epi::splitLexis] or [splitMulti]
by	variables to tabulate (aggregate) by. Flexible input, typically e.g. by = c("V1", "V2"). See Details and Examples.
type	determines output levels to which data is aggregated varying from returning only rows with pyrs > 0 ("unique") to returning all possible combinations of variables given in aggre even if those combinations are not represented in data ("full"); see Details
sum.values	optional: additional variables to sum by argument by. Flexible input, typically e.g. sum.values = c("V1", "V2")
subset	a logical condition to subset by before computations; e.g. subset = area %in% c("A", "B")
verbose	logical; if TRUE, the function returns timings and some information useful for debugging along the aggregation process

Details

Basics

aggre is intended for aggregation of split Lexis data only. See [Epi::Lexis] for forming Lexis objects by hand and e.g. [Epi::splitLexis], [splitLexisDT], and [splitMulti] for splitting the data. [lexpand] may be used for simple data sets to do both steps as well as aggregation in the same function call.

Here aggregation refers to computing person-years and the appropriate events (state transitions and end points in status) for the subjects in the data. Hence, it computes e.g. deaths (end-point and state transition) and censorings (end-point) as well as events in a multi-state setting (state transitions).

The result is a long-format data.frame or data.table (depending on options("popEpi.datatable"); see ?popEpi) with the columns pyrs and the appropriate transitions named as fromXtoY, e.g. from0to0 and from0to1 depending on the values of lex.Cst and lex.Xst.

The by argument

The by argument determines the length of the table, i.e. the combinations of variables to which data is aggregated. by is relatively flexible, as it can be supplied as

- a character string vector, e.g. c("sex", "area"), naming variables existing in lex
- an expression, e.g. factor(sex, 0:1, c("m", "f")) using any variable found in lex
- a list (fully or partially named) of expressions, e.g. list(gender = factor(sex, 0:1, c("m", "f"), area)

Note that expressions effectively allow a variable to be supplied simply as e.g. by = sex (as a symbol/name in R lingo).

The data is then aggregated to the levels of the given variables or expression(s). Variables defined to be time scales in the supplied Lexis are processed in a special way: If any are mentioned in the by argument, intervals of them are formed based on the breaks used to split the data: e.g. if age was split using the breaks c(0, 50, Inf), mentioning age in by leads to creating the age intervals [0, 50) and [50, Inf) and aggregating to them. The intervals are identified in the output as the lower bounds of the appropriate intervals.

The order of multiple time scales mentioned in by matters, as the last mentioned time scale is assumed to be a survival time scale for when computing event counts. E.g. when the data is split by the breaks list(FUT = 0:5, CAL = c(2008,2010)), time lines cut short at CAL = 2010 are considered to be censored, but time lines cut short at FUT = 5 are not. See Return.

Aggregation types (styles)

It is almost always enough to aggregate the data to variable levels that are actually represented in the data (default aggre = "unique"; alias "non-empty"). For certain uses it may be useful to have also "empty" levels represented (resulting in some rows in output with zero person-years and events); in these cases supplying aggre = "full" (alias "cartesian") causes aggre to determine the Cartesian product of all the levels of the supplied by variables or expressions and aggregate to them. As an example of a Cartesian product, try

merge(1:2, 1:5).

Value

A long data.frame or data.table of aggregated person-years (pyrs), numbers of subjects at risk (at.risk), and events formatted fromXtoY, where X and X are states transitioning from and to or

states at the end of each lex.id's follow-up (implying X = Y). Subjects at risk are computed in the beginning of an interval defined by any Lexis time scales and mentioned in by, but events occur at any point within an interval.

When the data has been split along multiple time scales, the last time scale mentioned in by is considered to be the survival time scale with regard to computing events. Time lines cut short by the extrema of non-survival-time-scales are considered to be censored ("transitions" from the current state to the current state).

Author(s)

Joonas Miettinen

See Also

[aggregate] for a similar base R solution, and [ltable] for a data.table based aggregator. Neither are directly applicable to split Lexis data.

Other aggregation functions: as.aggre(), lexpand(), setaggre(), summary.aggre()

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)

## form a Lexis object
library(Epi)
data(sibr)
x <- sibr[1:10,]
x[1:5,]$sex <- 0 ## pretend some are male
x <- Lexis(data = x,
           entry = list(AGE = dg_age, CAL = get.yrs(dg_date)),
           exit = list(CAL = get.yrs(ex_date)),
           entry.status=0, exit.status = status)
x <- splitMulti(x, breaks = list(CAL = seq(1993, 2013, 5),
                                 AGE = seq(0, 100, 50)))

## these produce the same results (with differing ways of determining aggre)
a1 <- aggre(x, by = list(gender = factor(sex, 0:1, c("m", "f")),
             agegroup = AGE, period = CAL))

a2 <- aggre(x, by = c("sex", "AGE", "CAL"))

a3 <- aggre(x, by = list(sex, agegroup = AGE, CAL))

## returning also empty levels
a4 <- aggre(x, by = c("sex", "AGE", "CAL"), type = "full")

## computing also expected numbers of cases
x <- lexpand(sibr[1:10,], birth = bi_date, entry = dg_date,
```

```
            exit = ex_date, status = status %in% 1:2,
            pophaz = popmort, fot = 0:5, age = c(0, 50, 100))
x$d.exp <- with(x, lex.dur*pop.haz)
## these produce the same result
a5 <- aggre(x, by = c("sex", "age", "fot"), sum.values = list(d.exp))
a5 <- aggre(x, by = c("sex", "age", "fot"), sum.values = "d.exp")
a5 <- aggre(x, by = c("sex", "age", "fot"), sum.values = d.exp)
## same result here with custom name
a5 <- aggre(x, by = c("sex", "age", "fot"),
            sum.values = list(expCases = d.exp))

## computing pohar-perme weighted figures
x$d.exp.pp <- with(x, lex.dur*pop.haz*pp)
a6 <- aggre(x, by = c("sex", "age", "fot"),
            sum.values = c("d.exp", "d.exp.pp"))
## or equivalently e.g. sum.values = list(expCases = d.exp, expCases.p = d.exp.pp).
```

`all_names_present` *Check if all names are present in given data*

Description

Given a character vector, checks if all names are present in `names(data)`. Throws error if `stops=TRUE`, else returns `FALSE` if some variable name is not present.

Usage

```
all_names_present(data, var.names, stops = TRUE, msg = NULL)
```

Arguments

`data`	dataset where the variable names should be found
`var.names`	a character vector of variable names, e.g. `c("var1", "var2")`
`stops`	logical, stop returns exception
`msg`	Custom message to return instead of default message. Special: include `%%VARS%%` in message string and the missing variable names will be inserted there (quoted, separated by comma, e.g. `'var1', 'var2'` — no leading or tracing white space).

Value

`TRUE` if all `var.names` are in `data`, else `FALSE`,

Author(s)

Joonas Miettinen

See Also

`[robust_values]`

`array_df_ratetable_utils` arrays, data.frames *and* ratetables

Description

Utilities to transform objects between `array`, `data.frame`, and survival::ratetable.

Usage

```
long_df_to_array(x, stratum.col.nms, value.col.nm)

long_df_to_ratetable(
  x,
  stratum.col.nms,
  value.col.nm,
  dim.types,
  cut.points = NULL
)

long_dt_to_array(x, stratum.col.nms, value.col.nm)

long_dt_to_ratetable(
  x,
  stratum.col.nms,
  value.col.nm,
  dim.types,
  cut.points = NULL
)

array_to_long_df(x)

array_to_long_dt(x)

array_to_ratetable(x, dim.types, cut.points = NULL)

ratetable_to_array(x)

ratetable_to_long_df(x)

ratetable_to_long_dt(x)
```

Arguments

x	[data.frame, data.table, array, ratetable] (mandatory, no default) • long_df_to_array: a data.frame • long_df_to_ratetable: a data.frame

- long_dt_to_array: a data.table
- long_dt_to_ratetable: a data.table
- array_to_long_df: an array
- array_to_long_dt: an array
- array_to_ratetable: an array
- ratetable_to_array: a survival::ratetable
- ratetable_to_long_df: a survival::ratetable
- ratetable_to_long_dt: a survival::ratetable

stratum.col.nms
: [character] (mandatory, no default)
a vector of column names in x by which values are stratified

value.col.nm
: [character] (mandatory, no default)
name of column in x containing values (these will be contents of the array)

dim.types
: [integer] (mandatory, no default)
see type under **Details** in survival::ratetable

cut.points
: [NULL, list] (optional, default NULL)
see cutpoints under **Details** in survival::ratetable

- NULL: automatically set using dimnames(x) and dim.types
- list: one element for each dimensions of x

Details

- long_df_to_array: converts a long-format data.frame to an array with one or more dimensions
- long_df_to_ratetable: calls long_df_to_array and then array_to_ratetable
- long_dt_to_array: simply asserts that x is a data.table and calls long_df_to_array
- long_dt_to_ratetable: calls long_dt_to_array and then array_to_ratetable
- array_to_long_df: converts an array with one or more dimensions into a long-format data.frame; any dimnames are used to name and fill the stratifying columns; for dimensions without a name, ".dX" is used for stratifying column number X; for each k, if there are no contents in dimnames(x)[[k]], the elements of seq(dim(x)[k]) are used to fill the corresponding stratifying column; the value column always has the name "value"
- array_to_long_dt: calls array_to_long_df and converts result to a data.table for convenience
- array_to_ratetable: converts an array to a survival::ratetable
- ratetable_to_array: converts a survival::ratetable to an array
- ratetable_to_long_df: calls ratetable_to_array and then array_to_long_df
- ratetable_to_long_dt: calls ratetable_to_array and then array_to_long_dt

Value

- long_df_to_array: an array
- long_df_to_ratetable: a survival::ratetable
- long_dt_to_array: an array
- long_dt_to_ratetable: a survival::ratetable
- array_to_long_df: an data.frame
- array_to_long_dt: an data.table
- array_to_ratetable: a survival::ratetable
- ratetable_to_array: an array
- ratetable_to_long_df: a data.frame
- ratetable_to_long_dt: a data.table

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)

long_dt <- popEpi::popmort
arr <- long_df_to_array(long_dt, c("agegroup", "year", "sex"), "haz")
rt <- array_to_ratetable(arr, dim.types = c(2L, 4L, 1L))

arr2 <- ratetable_to_array(rt)
long_df2 <- array_to_long_df(arr2)

identical(sort(long_dt[["haz"]]), sort(long_df2[["value"]]))
```

---

as.aggre *Coercion to Class* aggre

Description

Coerces an R object to an aggre object, identifying the object as one containing aggregated counts, person-years and other information.

Usage

```
as.aggre(x, values = NULL, by = NULL, breaks = NULL, ...)

## S3 method for class 'data.frame'
as.aggre(x, values = NULL, by = NULL, breaks = NULL, ...)

## S3 method for class 'data.table'
```

```
as.aggre(x, values = NULL, by = NULL, breaks = NULL, ...)

## Default S3 method:
as.aggre(x, ...)
```

Arguments

x	a data.frame or data.table
values	a character string vector; the names of value variables
by	a character string vector; the names of variables by which values have been tabulated
breaks	a list of breaks, where each element is a breaks vector as usually passed to e.g. [splitLexisDT]. The list must be fully named, with the names corresponding to time scales at the aggregate level in your data. Every unique value in a time scale variable in data must also exist in the corresponding vector in the breaks list.
...	arguments passed to or from methods

Value

Returns a copy of x with attributes set to those of an object of class "aggre".

Methods (by class)

- as.aggre(data.frame): Coerces a data.frame to an aggre object
- as.aggre(data.table): Coerces a data.table to an aggre object
- as.aggre(default): Default method for as.aggre (stops computations if no class-specific method found)

Author(s)

Joonas Miettinen

See Also

Other aggregation functions: aggre(), lexpand(), setaggre(), summary.aggre()

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
library("data.table")
df <- data.frame(sex = rep(c("male", "female"), each = 5),
                 obs = rpois(10, rep(7,5, each=5)),
                 pyrs = rpois(10, lambda = 10000))
dt <- as.data.table(df)
```

```
df <- as.aggre(df, values = c("pyrs", "obs"), by = "sex")
dt <- as.aggre(dt, values = c("pyrs", "obs"), by = "sex")

class(df)
class(dt)

BL <- list(fot = 0:5)
df <- data.frame(df)
df <- as.aggre(df, values = c("pyrs", "obs"), by = "sex", breaks = BL)
```

`as.Date.yrs` *Coerce Fractional Year Values to Date Values*

Description

Coerces an `yrs` object to a `Date` object. Some loss of information comes if `year.length = "approx"` was set when using `[get.yrs]`, so the transformation back to `Date` will not be perfect there. With `year.length = "actual"` the original values are perfectly retrieved.

Usage

```
## S3 method for class 'yrs'
as.Date(x, ...)
```

Arguments

`x`	an `yrs` object created by `get.yrs`
`...`	unused, included for compatibility with other `as.Date` methods

Value

A vector of `Date` values based on the input fractional years.

Author(s)

Joonas Miettinen

See Also

`[get.yrs]`

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
data("sire", package = "popEpi")

## approximate year lengths: here 20 % have an extra day added
sire$dg_yrs <- get.yrs(sire$dg_date)
summary(sire$dg_yrs)
dg_date2 <- as.Date(sire$dg_yrs)
summary(as.numeric(dg_date2 - as.Date(sire$dg_date)))

## using actual year lengths
sire$dg_yrs <- get.yrs(sire$dg_date, year.length = "actual")
summary(sire$dg_yrs)
dg_date2 <- as.Date(sire$dg_yrs)
summary(as.numeric(dg_date2 - as.Date(sire$dg_date)))
```

---

cast_simple *Cast* data.table/data.frame *from long format to wide format*

Description

Convenience function for using [data.table::dcast.data.table]; inputs are character strings (names of variables) instead of a formula.

Usage

```
cast_simple(data = NULL, columns = NULL, rows = NULL, values = NULL)
```

Arguments

data	a data.table or data.frame
columns	a character string vector; the (unique combinations of the) levels of these variable will be different rows
rows	a character string vector; the (unique combinations of the) levels of these variable will be different columns
values	a character string; the variable which will be represented on rows and columns as specified by columns and rows

Details

This function is just a small interface for dcast / dcast.data.table and less flexible than the originals.

Note that all data.table objects are also data.frame objects, but that each have their own dcast method. [data.table::dcast.data.table] is faster.

If any values in value.vars need to be aggregated, they are aggregated using sum. See ?dcast.

Value

A `data.table` just like `[data.table::dcast]`.

Author(s)

Matti Rantanen, Joonas Miettinen

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
library("data.table")
## e.g. silly counts from a long-format table to a wide format
test <- data.table::copy(popEpi::sire)
test$dg_y <- year(test$dg_date)
test$ex_y <- year(test$ex_date)
tab <- ltable(test, c("dg_y","ex_y"))
cast_simple(tab, columns='dg_y', rows="ex_y", values="obs")
```

---

cut_bound *Change output values from cut(..., labels = NULL) output*

Description

Selects lowest values of each factor after cut() based on the assumption that the value starts from index 2 and end in comma ",".

Usage

```
cut_bound(t, factor = TRUE)
```

Arguments

`t`	is a character vector of elements, e.g. "(20,60]"
`factor`	logical; TRUE returns informative character string, FALSE numeric (left value)

Details

type = 'factor': "[50,52)" -> "50-51" OR "[50,51)" -> "50"

type = 'numeric': lowest bound in numeric.

Value

If `factor = TRUE`, returns a character vector; else returns a numeric vector.

Author(s)

Matti Rantanen

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
cut_bound("[1900, 1910)") ## "1900-1909"
```

`direct_standardization`

Direct Adjusting in **popEpi** *Using Weights*

Description

Several functions in **popEpi** have support for direct standardization of estimates. This document explains the usage of weighting with those functions.

Details

Direct standardization is performed by computing estimates of `E` by the set of adjusting variables `A`, to which a set of weights `W` is applicable. The weighted average over `A` is then the direct-adjusted estimate of `E` (`E*`).

To enable both quick and easy as well as more rigorous usage of direct standardization with weights, the weights arguments in **popEpi** can be supplied in several ways. Ability to use the different ways depends on the number of adjusting variables.

The weights are always handled internally to sum to 1, so they do not need to be scaled in this manner when they are supplied. E.g. counts of subjects in strata may be passed.

Basic usage - one adjusting variable

In the simple case where we are adjusting by only one variable (e.g. by age group), one can simply supply a vector of weights:

`FUN(weights = c(0.1, 0.25, 0.25, 0.2, 0.2))`

which may be stored in advance:

`w <- c(0.1, 0.25, 0.25, 0.2, 0.2)`

`FUN(weights = w)`

The order of the weights matters. **popEpi** functions with direct adjusting enabled match the supplied weights to the adjusting variables as follows: If the adjusting variable is a `factor`, the order of the levels is used. Otherwise, the alphabetic order of the unique values is used (try `sort`

to see how it works). For clarity and certainty we recommend using `factor` or `numeric` variables when possible. `character` variables should be avoided: to see why, try `sort(15:9)` and `sort(as.character(15:9))`.

It is also possible to supply a `character` string corresponding to one of the age group standardization schemes integrated into **popEpi**:

- `'europe_1976_18of5'` - European std. population (1976), 18 age groups
- `'nordic_2000_18of5'` - Nordic std. population (2000), 18 age groups
- `'world_1966_18of5'` - world standard (1966), 18 age groups
- `'world_2000_18of5'` - world standard (2000), 18 age groups
- `'world_2000_20of5'` - world standard (2000), 20 age groups
- `'world_2000_101of1'` - world standard (2000), 101 age groups

Additionally, `[ICSS]` contains international weights used in cancer survival analysis, but they are not currently usable by passing a string to `weights` and must be supplied by hand.

You may also supply `weights = "internal"` to use internally computed weights, i.e. usually simply the counts of subjects / person-time experienced in each stratum. E.g.

```
FUN(weights = "world_2000_18of5")
```

will use the world standard population from 2000 as weights for 18 age groups, that your adjusting variable is assumed to contain. The adjusting variable must be coded in this case as a numeric variable containing `1:18` or as a `factor` with 18 levels (coded from the youngest to the oldest age group).

More than one adjusting variable

In the case that you employ more than one adjusting variable, separate weights should be passed to match to the levels of the different adjusting variables. When supplied correctly, "grand" weights are formed based on the variable-specific weights by multiplying over the variable-specific weights (e.g. if men have `w = 0.5` and the age group 0-4 has `w = 0.1`, the "grand" weight for men aged 0-4 is `0.5*0.1`). The "grand" weights are then used for adjusting after ensuring they sum to one.

When using multiple adjusting variables, you are allowed to pass either a named `list` of weights or a `data.frame` of weights. E.g.

```
WL <- list(agegroup = age_w, sex = sex_w)
FUN(weights = WL)
```

where `age_w` and `sex_w` are numeric vectors. Given the conditions explained in the previous section are satisfied, you may also do e.g.

```
WL <- list(agegroup = "world_2000_18of", sex = sex_w)
FUN(weights = WL)
```

and the world standard pop is used as weights for the age groups as outlined in the previous section.

Sometimes using a `data.frame` can be clearer (and it is fool-proof as well). To do this, form a `data.frame` that repeats the levels of your adjusting variables by each level of every other adjusting variable, and assign the weights as a column named `"weights"`. E.g.

```
wdf <- data.frame(sex = rep(0:1, each = 18), agegroup = rep(1:18, 2))
```

```
wdf$weights <- rbinom(36, size = 100, prob = 0.25)
FUN(weights = wdf)
```

If you want to use the counts of subjects in strata as the weights, one way to do this is by e.g.

```
wdf <- as.data.frame(x$V1, x$V2, x$V3) names(wdf) <- c("V1", "V2", "V3", "weights")
```

Author(s)

Joonas Miettinen

References

Source of the Nordic standard population in 5-year age groups (also contains European & 1966 world standards): https://www-dep.iarc.fr/NORDCAN/english/glossary.htm

Source of the 1976 European standard population:

Waterhouse, J.,Muir, C.S.,Correa, P.,Powell, J., eds (1976). Cancer Incidence in Five Continents, Vol. III. IARC Scientific Publications, No. 15, Lyon, IARC. ISBN: 9789283211150

Source of 2000 world standard population in 1-year age groups: https://seer.cancer.gov/stdpopulations/stdpop.singleages.html

See Also

Other weights: ICSS, stdpop101, stdpop18

Other popEpi argument evaluation docs: flexible_argument

---

fac2num *Convert factor variable to numeric*

Description

Convert factor variable with numbers as levels into a numeric variable

Usage

```
fac2num(x)
```

Arguments

x	a factor variable with numbers as levels

Details

For example, a factor with levels `c("5","7")` is converted into a numeric variable with values `c(5,7)`.

Value

A numeric vector based on the levels of `x`.

Source

Stackoverflow thread

See Also

[robust_values]

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
## this is often not intended
as.numeric(factor(c(5,7))) ## result: c(1,2)
## but this
fac2num(factor(c(5,7))) ## result: c(5,7)

## however
as.numeric(factor(c("5","7","a"))) ## 1:3

suppressWarnings(
  fac2num(factor(c("5","7","a"))) ## c(5,7,NA)
)
```

---

flexible_argument *Flexible Variable Usage in* **popEpi** *Functions*

Description

Certain arguments in **popEpi** can be passed in multiple ways. This document shows the usage and a pitfall in the usage of such flexible arguments.

Details

Flexible arguments in **popEpi** are used to pass variables existing in your data or in the environment where the function is used (for everyday users this is the global environment - in simple terms, where your data is / your work space). The flexible arguments are modelled after the by argument in `data.tables` - see `?data.table`. There are many ways to supply the same information to certain functions in **popEpi**, but the possible ways listed below may be limited in some of them to only allow for using only a part of them.

Everyday usage

Most commonly you may pass variable names as character strings, e.g.

```
FUN(arg = c("V1", "V2"), data = x)
```

which may be stored in advance:

```
vars <- c("V1", "V2")
FUN(arg = vars, data = x)
```

where x contains those variables. You may also supply variable names as symbols:

```
FUN(arg = V1, data = x)
```

Or as a list of symbols (similarly to as in [aggregate]):

```
FUN(arg = list(V1, V2), data = x)
```

Or as a list of expressions:

```
FUN(arg = list(V1 + 1, factor(V2)), data = x)
```

A formula without a left-hand-side specified is sometimes allowed as well:

```
FUN(arg = ~ I(V1 + 1) + factor(V2), data = x)
```

Using a symbol or a list of symbols/expressions typically causes the function to look for the variable(s) first in the supplied data (if any) and then where the function was called. For everyday users this means you might define e.g.

```
V3 <- factor(letters)
```

and do e.g.

```
FUN(arg = list(V1 + 1, factor(V2), V3), data = x)
```

provided V1 and V2 exist in x or in the function calling environment.

A pitfall

There is one way to use flexible arguments incorrectly: By supplying the name of a variable which exists both in the supplied data and the calling environment, and intending the latter to be used. E.g.

```
vars <- c("V2")
FUN(arg = V3, data = x)
```

where x has a column named vars. This causes the function to use x$vars and NOT x$V2.

Advanced

Function programmers are advised to pass character strings whenever possible. To fool-proof against conflicts as described in the section above, refer to the calling environment explicitly when passing the variable containing the character strings:

```
TF <- environment() ## current env to refer to
vars <- c("V1", "V2")
FUN(arg = TF$vars, data = x)
```

Even if x has columns named vars and TF, using TF$vars does not use those columns but only evaluates TF$vars in the calling environment. This is made possible by the fact that data is always passed as a data.frame, within which evaluation of expressions using the dollar operator is not

possible. Therefore it is safe to assume the data should not be used. However, lists of expressions will not be checked for dollar use and will fail in conflict situations:

`TF <- environment() ## current env to refer to`

`vars <- letters[1:5]`

`x <- data.frame(vars = 1:5, TF = 5:1, V1 = 10:6)`

`FUN(arg = list(TF$vars, V1), data = x)`

On the other hand you may typically also pass quoted ([quote]) or substituted [substitute] expressions etc., where the `env$object` trick will work as well:

`q <- quote(list(vars, V1))`

`FUN(arg = TF$q, data = x)`

This works even with

`a <- 1:5`

`V1 <- quote(TF$a)`

`FUN(arg = TF$V1, data = x)`

So no conflicts should occur.

Author(s)

Joonas Miettinen

See Also

Other popEpi argument evaluation docs: direct_standardization

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)

data(sire)
## prepare data for e.g. 5-year "period analysis" for 2008-2012
## note: sire is a simulated cohort integrated into popEpi.
BL <- list(fot=seq(0, 5, by = 1/12))
x <- lexpand(sire, birth = bi_date, entry = dg_date, exit = ex_date,
             status = status %in% 1:2,
             breaks = BL)

x <- aggre(x, by = fot)

## silly example of referring to pyrs data by fixed character string;
## its possible that the real name wont be fixed in a real-life application.
pyrs <- "actual_pyrs"
TF <- environment()
x$actual_pyrs <- as.numeric(x$pyrs)
```

```
x$pyrs <- 1

## this works (uses actual_pyrs eventually)
st <- survtab_ag(fot ~ 1, data = x, surv.type = "surv.obs",
                 pyrs = TF$pyrs, d = from0to1,
                 surv.method = "hazard")
## this would be wrong (sees expression 'pyrs' and uses that column,
## which is not what is intended here)
st <- survtab_ag(fot ~ 1, data = x, surv.type = "surv.obs",
                 pyrs = pyrs, d = from0to1,
                 surv.method = "hazard")
```

`get.yrs` *Fractional Years*

Description

Using Date objects, calculates given dates as fractional years.

Usage

```
get.yrs(x, year.length = "approx", ...)
```

Arguments

`x`	a `Date` object, or anything that `[as.Date]` accepts
`year.length`	character string, either `"actual"` or `"approx"`; can be abbreviated; see **Details**
`...`	additional arguments passed on to `[as.Date]`; typically `format` when `x` is a character string variable, and `origin` when `x` is numeric

Details

`x` should preferably be a `Date` or `IDate` object, although it can also be a character string variable which is coerced internally to `Date` format using `[as.Date.character]`.

When `year.length = 'actual'`, fractional years are calculated as `year + (day_in_year-1)/365` for non-leap-years and as `year + (day_in_year-1)/366` for leap years. If `year.length = 'approx'`, fractional years are always calculated as in `year + (day_in_year-1)/365.242199`.

There is a slight difference, then, between the two methods when calculating durations between fractional years. For meticulous accuracy one might instead want to calculate durations using dates (days) and convert the results to fractional years.

Note that dates are effectively converted to fractional years at `00:00:01` o'clock:

`get.yrs("2000-01-01") = 2000`, and `get.yrs("2000-01-02") = 2000 + 1/365.242199`.

Value

A numeric vector of fractional years.

Author(s)

Joonas Miettinen

See Also

[Epi::cal.yr], [as.Date.yrs], [as.Date]

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)

data("sire")
sire$dg_yrs <- get.yrs(sire$dg_date)
summary(sire$dg_yrs)

## see: ?as.Date.yrs
dg_date2 <- as.Date(sire$dg_yrs)
summary(as.numeric(dg_date2 - as.Date(sire$dg_date)))

## Epi's cal.yr versus get.yrs
d <- as.Date("2000-01-01")
Epi::cal.yr(d) ## 1999.999
get.yrs(d) ## 2000

## "..." passed on to as.Date, so character / numeric also accepted as input
## (and whatever else as.Date accepts)
get.yrs("2000-06-01")
get.yrs("20000601", format = "%Y%m%d")
get.yrs("1/6/00", format = "%d/%m/%y")

get.yrs(100, origin = "1970-01-01")
```

ICSS *Age standardisation weights from the ICSS scheme.*

Description

Contains three sets age-standardisation weights for age-standardized survival (net, relative or observed).

Format

data.table with columns

- age - lower bound of the age group
- ICSS1 - first set of weights, sums to 100 000
- ICSS2 - second set of weights, sums to 100 000
- ICSS3 - third set of weights, sums to 100 000

Source

ICSS weights (US National Cancer Institute website)

Corazziari, Isabella, Mike Quinn, and Riccardo Capocaccia. "Standard cancer patient population for age standardising survival ratios." European Journal of Cancer 40.15 (2004): 2307-2316.

See Also

Other popEpi data: `meanpop_fi`, `popmort`, `sibr`, `sire`, `stdpop101`, `stdpop18`

Other weights: `direct_standardization`, `stdpop101`, `stdpop18`

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
## aggregate weights to a subset of age groups
data(ICSS)
cut <- c(0, 30, 50, 70, Inf)
agegr <- cut(ICSS$age, cut, right = FALSE)
aggregate(ICSS1~agegr, data = ICSS, FUN = sum)
```

`is.Date` *Test if object is a* `Date` *object*

Description

Test whether `obj` inherits one of `Date` or `IDate`.

Usage

```
is.Date(obj)
```

Arguments

`obj`	object to test on

Value

TRUE if obj is of class "Date" or "IDate".

Author(s)

Joonas Miettinen

See Also

[get.yrs], [is_leap_year], [as.Date]

---

is_leap_year *Detect leap years*

Description

Given a vector or column of year values (numeric or integer), [is_leap_year] returns a vector of equal length of logical indicators, i.e. a vector where corresponding leap years have value TRUE, and FALSE otherwise.

Usage

```
is_leap_year(years)
```

Arguments

years	a vector or column of year values (numeric or integer)

Value

A logical vector where TRUE indicates a leap year.

Author(s)

Joonas Miettinen

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
## can be used to assign new columns easily, e.g. a dummy indicator column
df <- data.frame(yrs=c(1900,1904,2005,1995))
df$lyd <- as.integer(is_leap_year(df$yrs))

## mostly it is useful as a condition or to indicate which rows have leap years
which(is_leap_year(df$yrs)) # 2
```

```
df[is_leap_year(df$yrs),] # 2nd row
```

---

lexis_funs Lexis *Datasets*

Description

Make [Epi::Lexis] objects.

Usage

```
Lexis_fpa(
  data,
  birth = NULL,
  entry = NULL,
  exit = NULL,
  entry.status = NULL,
  exit.status = NULL,
  subset = NULL,
  ...
)

Lexis_dt(...)
```

Arguments

data	a data.frame; mandatory
birth	the time of birth; A character string naming the variable in data or an expression to evaluate - see Flexible input
entry	the time at entry to follow-up; supplied the same way as birth
exit	the time at exit from follow-up; supplied the same way as birth
entry.status	passed on to [Epi::Lexis] if not NULL; supplied the same way as birth
exit.status	passed on to [Epi::Lexis] if not NULL; supplied the same way as birth
subset	a logical condition to subset by before passing data and arguments to [Epi::Lexis]
...	arguments passed to [Epi::Lexis]

Value

popEpi::Lexis_fpa

Returns a Lexis object with the additional class data.table. It has the usual columns that Lexis objects have, and with time scale columns fot, per, and age. They are calculated as

fot = entry - entry (to ensure correct format, e.g. difftime)

per = entry

and

age = entry - birth.

popEpi::Lexis_dt

Returns a Lexis object that is also a data.table — therefore output has the classes c("Lexis", "data.table", "data.frame").

Functions

popEpi::Lexis_fpa

popEpi::Lexis_fpa collects data from its inputs to to call [Epi::Lexis]. This is a convenience function for making a Lexis object with the time scales fot, per, and age.

popEpi::Lexis_dt

popEpi::Lexis_dt performs the following steps:

- Calls [Epi::Lexis].
- Calls [data.table::setDT] to make output into a data.table, [data.table::setattr] to set its class to c("Lexis", "data.table", "data.frame"), and [data.table::setkeyv] to sort the output by lex.id and the time scales.
- Returns the Lexis / data.table object.

News for version 0.5.0

Lexis_fpa output is now also a data.table, so the complete class vector is c("Lexis", "data.table", "data.frame").

New function Lexis_dt, a wrapper of Epi::Lexis which also sets class to c("Lexis", "data.table", "data.frame").

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)

# popEpi::Lexis_fpa
data("sire", package = "popEpi")
lex <- Lexis_fpa(sire,
                 birth = "bi_date",
                 entry = dg_date,
                 exit = ex_date + 1L,
                 exit.status = "status")

## some special cases
myVar <- "bi_date"
l <- list(myVar = "bi_date")
sire$l <- sire$myVar <- 1
```

```
## conflict: myVar taken from data when "bi_date" was intended
lex <- Lexis_fpa(sire,
                 birth = myVar,
                 entry = dg_date,
                 exit = ex_date + 1L,
                 exit.status = "status")

## no conflict with names in data
lex <- Lexis_fpa(sire,
                 birth = l$myVar,
                 entry = dg_date,
                 exit = ex_date + 1L,
                 exit.status = "status")
stopifnot(
  identical(class(lex), c("Lexis", "data.table", "data.frame")),
  Epi::timeScales(lex) %in% c("fot", "per", "age")
)

# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)

# popEpi::Lexis_dt
lex_1 <- popEpi::Lexis_dt(
  data = popEpi::sire,
  entry = list(
    ts_fut = 0L,
    ts_age = as.integer(dg_date - bi_date),
    ts_cal = as.integer(dg_date)
  ),
  exit = list(ts_cal = as.integer(ex_date)),
  entry.status = 0L,
  exit.status = status
)
stopifnot(
  class(lex_1) == c("Lexis", "data.table", "data.frame"),
  Epi::timeScales(lex_1) %in% c("ts_fut", "ts_age", "ts_cal")
)

lex_2 <- popEpi::Lexis_dt(
  data = popEpi::sire,
  entry = list(
    ts_fut = 0L,
    ts_age = as.integer(dg_date - bi_date),
    ts_cal = as.integer(dg_date)
  ),
  duration = as.integer(ex_date - dg_date),
  entry.status = 0L,
  exit.status = status
)
stopifnot(
```

```
  class(lex_2) == c("Lexis", "data.table", "data.frame"),
  Epi::timeScales(lex_2) %in% c("ts_fut", "ts_age", "ts_cal")
)
```

`lexpand` *Split case-level observations*

Description

Given subject-level data, data is split by calendar time (`per`), `age`, and follow-up time (`fot`, from 0 to the end of follow-up) into subject-time-interval rows according to given `breaks` and additionally processed if requested.

Usage

```
lexpand(
  data,
  birth = NULL,
  entry = NULL,
  exit = NULL,
  event = NULL,
  status = status != 0,
  entry.status = NULL,
  breaks = list(fot = c(0, Inf)),
  id = NULL,
  overlapping = TRUE,
  aggre = NULL,
  aggre.type = c("unique", "cartesian"),
  drop = TRUE,
  pophaz = NULL,
  pp = TRUE,
  subset = NULL,
  merge = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

`data`	dataset of e.g. cancer cases as rows
`birth`	birth time in date format or fractional years; string, symbol or expression
`entry`	entry time in date format or fractional years; string, symbol or expression
`exit`	exit from follow-up time in date format or fractional years; string, symbol or expression
`event`	advanced: time of possible event differing from `exit`; typically only used in certain SIR/SMR calculations - see Details; string, symbol or expression

status	variable indicating type of event at exit or event; e.g. status = status != 0; expression or quoted variable name
entry.status	input in the same way as status; status at entry; see Details
breaks	a named list of vectors of time breaks; e.g. breaks = list(fot=0:5, age=c(0,45,65,Inf)); see Details
id	optional; an id variable; e.g. id = my_id; string, symbol or expression
overlapping	advanced, logical; if FALSE AND if data contains multiple rows per subject, ensures that the timelines of id-specific rows do not overlap; this ensures e.g. that person-years are only computed once per subject in a multi-state paradigm
aggre	e.g. aggre = list(sex, fot); a list of unquoted variables and/or expressions thereof, which are interpreted as factors; data events and person-years will be aggregated by the unique combinations of these; see Details
aggre.type	one of c("unique","cartesian"); can be abbreviated; see Details
drop	logical; if TRUE, drops all resulting rows after splitting that reside outside the time window as defined by the given breaks (all time scales)
pophaz	a dataset of population hazards to merge with split data; see Details
pp	logical; if TRUE, computes Pohar-Perme weights using pophaz; adds variable with reserved name pp; see Details for computing method
subset	a logical vector or any logical condition; data is subsetted before splitting accordingly
merge	logical; if TRUE, retains all original variables from the data
verbose	logical; if TRUE, the function is chatty and returns some messages along the way
...	e.g. fot = 0:5; instead of specifying a breaks list, correctly named breaks vectors can be given for fot, age, and per; these override any breaks in the breaks list; see Examples

Details

Basics

[lexpand] splits a given data set (with e.g. cancer diagnoses as rows) to subintervals of time over calendar time, age, and follow-up time with given time breaks using [splitMulti].

The dataset must contain appropriate Date / IDate format or other numeric variables that can be used as the time variables.

You may take a look at a simulated cohort [sire] as an example of the minimum required information for processing data with lexpand.

Many arguments can be supplied as a character string naming the appropriate variable (e.g. "sex"), as a symbol (e.g. sex) or as an expression (e.g. factor(sex, 0:1, c("m", "f"))) for flexibility.

Breaks

You should define all breaks as left inclusive and right exclusive time points (e.g.[a,b)) for 1-3 time dimensions so that the last member of a breaks vector is a meaningful "final upper limit", e.g. per = c(2002,2007,2012) to create a last subinterval of the form [2007,2012).

All breaks are explicit, i.e. if drop = TRUE, any data beyond the outermost breaks points are dropped. If one wants to have unspecified upper / lower limits on one time scale, use Inf: e.g. breaks =

list(fot = 0:5, age = c(0,45,Inf)). Breaks for per can also be given in Date / IDateformat, whereupon they are converted to fractional years before used in splitting.

The age time scale can additionally be automatically split into common age grouping schemes by naming the scheme with an appropriate character string:

- "18of5": age groups 0-4, 5-9, 10-14, ..., 75-79, 80-84, 85+
- "20of5": age groups 0-4, 5-9, 10-14, ..., 85-89, 90-94, 95+
- "101of1": age groups 0, 1, 2, ..., 98, 99, 100+

Time variables

If any of the given time variables (birth, entry, exit, event) is in any kind of date format, they are first coerced to fractional years before splitting using [get.yrs] (with year.length = "actual").

Sometimes in e.g. SIR/SMR calculation one may want the event time to differ from the time of exit from follow-up, if the subject is still considered to be at risk of the event. If event is specified, the transition to status is moved to event from exit using [Epi::cutLexis]. See Examples.

The status variable

The statuses in the expanded output (lex.Cst and lex.Xst) are determined by using either only status or both status and entry.status. If entry.status = NULL, the status at entry is guessed according to the type of variable supplied via status: For numeric variables it will be zero, for factors the first level (levels(status)[1]) and otherwise the first unique value in alphabetical order (sort(unique(status))[1]).

Using numeric or factor status variables is strongly recommended. Logical expressions are also allowed (e.g. status = my_status != 0L) and are converted to integer internally.

Merging population hazard information

To enable computing relative/net survivals with [survtab] and [relpois], lexpand merges an appropriate population hazard data (pophaz) to the expanded data before dropping rows outside the specified time window (if drop = TRUE). pophaz must, for this reason, contain at a minimum the variables named agegroup, year, and haz. pophaz may contain additional variables to specify different population hazard levels in different strata; e.g. popmort includes sex. All the strata-defining variables must be present in the supplied data. lexpand will automatically detect variables with common names in the two datasets and merge using them.

Currently year must be an integer variable specifying the appropriate year. agegroup must currently also specify one-year age groups, e.g. popmort specifies 101 age groups of length 1 year. In both year and agegroup variables the values are interpreted as the lower bounds of intervals (and passed on to a cut call). The mandatory variable haz must specify the appropriate average rate at the person-year level; e.g. haz = -log(survProb) where survProb is a one-year conditional survival probability will be the correct hazard specification.

The corresponding pophaz population hazard value is merged by using the mid points of the records after splitting as reference values. E.g. if age=89.9 at the start of a 1-year interval, then the reference age value is 90.4 for merging. This way we get a "typical" population hazard level for each record.

Computing Pohar-Perme weights

If pp = TRUE, Pohar-Perme weights (the inverse of cumulative population survival) are computed. This will create the new pp variable in the expanded data. pp is a reserved name and lexpand throws exception if a variable with that name exists in data.

When a survival interval contains one or several rows per subject (e.g. due to splitting by the per scale), pp is cumulated from the beginning of the first record in a survival interval for each subject to the mid-point of the remaining time within that survival interval, and that value is given for every other record that a given person has within the same survival interval.

E.g. with 5 rows of duration 1/5 within a survival interval [0,1)], pp is determined for all records by a cumulative population survival from 0 to 0.5. The existing accuracy is used, so that the weight is cumulated first up to the end of the second row and then over the remaining distance to the mid-point (first to 0.4, then to 0.5). This ensures that more accurately merged population hazards are fully used.

Event not at end of follow-up & overlapping time lines

event may be used if the event indicated by status should occur at a time differing from exit. If event is defined, cutLexis is used on the data set after coercing it to the Lexis format and before splitting. Note that some values of event are allowed to be NA as with cutLexis to accommodate observations without an event occurring.

Additionally, setting overlapping = FALSE ensures that (irrespective of using event) the each subject defined by id only has one continuous time line instead of possibly overlapping time lines if there are multiple rows in data by id.

Aggregating

Certain analyses such as SIR/SMR calculations require tables of events and person-years by the unique combinations (interactions) of several variables. For this, aggre can be specified as a list of such variables (preferably factor variables but not mandatory) and any arbitrary functions of the variables at one's disposal. E.g.

aggre = list(sex, agegr = cut(dg_age, 0:100))

would tabulate events and person-years by sex and an ad-hoc age group variable. Every ad-hoc-created variable should be named.

fot, per, and age are special reserved variables which, when present in the aggre list, are output as categories of the corresponding time scale variables by using e.g.

cut(fot, breaks$fot, right=FALSE).

This only works if the corresponding breaks are defined in breaks or via "...". E.g.

aggre = list(sex, fot.int = fot) with

breaks = list(fot=0:5).

The output variable fot.int in the above example will have the lower limits of the appropriate intervals as values.

aggre as a named list will output numbers of events and person-years with the given new names as categorizing variable names, e.g. aggre = list(follow_up = fot, gender = sex, agegroup = age).

The output table has person-years (pyrs) and event counts (e.g. from0to1) as columns. Event counts are the numbers of transitions (lex.Cst != lex.Xst) or the lex.Xst value at a subject's last record (subject possibly defined by id).

If aggre.type = "unique" (alias "non-empty"), the above results are computed for existing combinations of expressions given in aggre, but also for non-existing combinations if aggre.type = "cartesian" (alias "full"). E.g. if a factor variable has levels "a", "b", "c" but the data is limited to only have levels "a", "b" present (more than zero rows have these level values), the former

setting only computes results for "a", "b", and the latter also for "c" and any combination with other variables or expression given in aggre. In essence, "cartesian" forces also combinations of variables used in aggre that have no match in data to be shown in the result.

If aggre is not NULL and pophaz has been supplied, lexpand also aggregates the expected counts of events, which appears in the output data by the reserved name d.exp. Additionally, having pp = TRUE causes lexpand to also compute various Pohar-Perme weighted figures necessary for computing Pohar-Perme net survivals with [survtab_ag]. This can be slow, so consider what is really needed. The Pohar-Perme weighted figures have the suffix .pp.

[0,1)]: R:0,1) [survtab_ag]: R:survtab_ag

Value

If aggre = NULL, returns a data.table or data.frame (depending on options("popEpi.datatable"); see ?popEpi) object expanded to accommodate split observations with time scales as fractional years and pophaz merged in if given. Population hazard levels in new variable pop.haz, and Pohar-Perme weights as new variable pp if requested.

If aggre is defined, returns a long-format data.table/data.frame with the variable pyrs (person-years), and variables for the counts of transitions in state or state at end of follow-up formatted fromXtoY, where X and Y are the states transitioned from and to, respectively. The data may also have the columns d.exp for expected numbers of cases and various Pohar-Perme weighted figures as identified by the suffix .pp; see Details.

Author(s)

Joonas Miettinen

See Also

[Epi::Lexis], [popmort]

Other splitting functions: splitLexisDT(), splitMulti()

Other aggregation functions: aggre(), as.aggre(), setaggre(), summary.aggre()

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)

## prepare data for e.g. 5-year cohort survival calculation
x <- lexpand(sire, breaks=list(fot=seq(0, 5, by = 1/12)),
             birth = bi_date, entry = dg_date, exit = ex_date,
             status =  status != 0, pophaz=popmort)

## prepare data for e.g. 5-year "period analysis" for 2008-2012
BL <- list(fot = seq(0, 5, by = 1/12), per = c("2008-01-01", "2013-01-01"))
x <- lexpand(sire, breaks = BL,
             birth = bi_date, entry = dg_date, exit = ex_date,
```

```
             pophaz=popmort, status =  status != 0)

## aggregating
BL <- list(fot = 0:5, per = c("2003-01-01","2008-01-01", "2013-01-01"))
ag <- lexpand(sire, breaks = BL, status = status != 0,
             birth = bi_date, entry = dg_date, exit = ex_date,
              aggre=list(sex, period = per, surv.int = fot))

## aggregating even more
ag <- lexpand(sire, breaks = BL, status = status != 0,
              birth = bi_date, entry = dg_date, exit = ex_date,
              aggre=list(sex, period = per, surv.int = fot),
              pophaz = popmort, pp = TRUE)

## using "..."
x <- lexpand(sire, fot=0:5, status =  status != 0,
             birth = bi_date, entry = dg_date, exit = ex_date,
             pophaz=popmort)

x <- lexpand(sire, fot=0:5, status =  status != 0,
             birth = bi_date, entry = dg_date, exit = ex_date,
             aggre=list(sex, surv.int = fot))

## using the "event" argument: it just places the transition to given "status"
## at the "event" time instead of at the end, if possible using cutLexis
x <- lexpand(sire, status = status, event = dg_date,
             birth = bi_date, entry = dg_date, exit = ex_date,)

## aggregating with custom "event" time
## (the transition to status is moved to the "event" time)
x <- lexpand(sire, status = status, event = dg_date,
             birth = bi_date, entry = dg_date, exit = ex_date,
             per = 1970:2014, age = c(0:100,Inf),
             aggre = list(sex, year = per, agegroup = age))
```

`lines.sirspline` *lines method for sirspline-object*

Description

Plot SIR spline lines with R base graphics

Usage

```
## S3 method for class 'sirspline'
lines(x, conf.int = TRUE, print.levels = NA, select.spline, ...)
```

Arguments

x	an object returned by function sirspline
conf.int	logical; default TRUE draws also the 95 confidence intervals
print.levels	name(s) to be plotted. Default plots all levels.
select.spline	select which spline variable (a number or a name) is plotted.
...	arguments passed on to lines()

Details

In lines.sirspline most of graphical parameters is user adjustable. Desired spline variable can be selected with select.spline and only one can be plotted at a time. The spline variable can include several levels, e.g. gender (these are the levels of print from sirspline). All levels are printed by default, but a specific level can be selected using argument print.levels. Printing the levels separately enables e.g. to give different colours for each level.

Value

Always returns NULL invisibly. This function is called for its side effects.

Author(s)

Matti Rantanen

See Also

Other sir functions: plot.sirspline(), sir(), sir_exp(), sir_ratio(), sirspline()

---

lines.survmean *Graphically Inspect Curves Used in Mean Survival Computation*

Description

Plots the observed (with extrapolation) and expected survival curves for all strata in an object created by [survmean]

Usage

```
## S3 method for class 'survmean'
lines(x, ...)
```

Arguments

x	a survmean object
...	arguments passed (ultimately) to matlines; you may, therefore, supply e.g. lwd through this, though arguments such as lty and col will not work

Details

This function is intended to be a workhorse for [plot.survmean]. If you want finer control over the plotted curves, extract the curves from the survmean output using

attr(x, "curves")

where x is a survmean object.

Value

Always returns NULL invisibly. This function is called for its side effects.

Author(s)

Joonas Miettinen

See Also

Other survmean functions: Surv(), plot.survmean(), survmean()

---

lines.survtab *lines method for survtab objects*

Description

Plot lines from a survtab object

Usage

```
## S3 method for class 'survtab'
lines(x, y = NULL, subset = NULL, conf.int = TRUE, col = NULL, lty = NULL, ...)
```

Arguments

x	a survtab output object
y	a variable to plot; a quoted name of a variable in x; e.g. y = "surv.obs"; if NULL, picks last survival variable column in order in x
subset	a logical condition; obj is subset accordingly before plotting; use this for limiting to specific strata, e.g. subset = sex == "male"
conf.int	logical; if TRUE, also plots any confidence intervals present in obj for variables in y
col	line colour passed to matlines
lty	line type passed to matlines
...	additional arguments passed on to to a matlines call; e.g. lwd can be defined this way

Value

Always returns NULL invisibly. This function is called for its side effects.

Author(s)

Joonas Miettinen

See Also

Other survtab functions: Surv(), plot.survtab(), print.survtab(), summary.survtab(), survtab(), survtab_ag()

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
data(sire)
data(sibr)
si <- rbind(sire, sibr)
si$period <- cut(si$dg_date, as.Date(c("1993-01-01", "2004-01-01", "2013-01-01")), right = FALSE)
si$cancer <- c(rep("rectal", nrow(sire)), rep("breast", nrow(sibr)))
x <- lexpand(si, birth = bi_date, entry = dg_date, exit = ex_date,
             status = status %in% 1:2,
             fot = 0:5, aggre = list(cancer, period, fot))
st <- survtab_ag(fot ~ cancer + period, data = x,
                 surv.method = "lifetable", surv.type = "surv.obs")

plot(st, "surv.obs", subset = cancer == "breast", ylim = c(0.5, 1), col = "blue")
lines(st, "surv.obs", subset = cancer == "rectal", col = "red")

## or
plot(st, "surv.obs", col = c(2,2,4,4), lty = c(1, 2, 1, 2))
```

---

lower_bound *Return lower_bound value from char string (20,30]*

Description

selects lowest values of each factor after cut() based on that the value starts from index 2 and end in comma ",".

Usage

```
lower_bound(cut)
```

Arguments

cut	is a character vector of elements "(20,60]"

Value

A numeric vector.

Author(s)

Matti Rantanen

---

ltable *Tabulate Counts and Other Functions by Multiple Variables into a Long-Format Table*

Description

`ltable` makes use of `data.table` capabilities to tabulate frequencies or arbitrary functions of given variables into a long format `data.table`/`data.frame`. `expr.by.cj` is the equivalent for more advanced users.

Usage

```
ltable(
  data,
  by.vars = NULL,
  expr = list(obs = .N),
  subset = NULL,
  use.levels = TRUE,
  na.rm = FALSE,
  robust = TRUE
)

expr.by.cj(
  data,
  by.vars = NULL,
  expr = list(obs = .N),
  subset = NULL,
  use.levels = FALSE,
  na.rm = FALSE,
  robust = FALSE,
  .SDcols = NULL,
  enclos = parent.frame(1L),
  ...
)
```

Arguments

data	a data.table/data.frame
by.vars	names of variables that are used for categorization, as a character vector, e.g. c('sex','agegroup')
expr	object or a list of objects where each object is a function of a variable (see: details)
subset	a logical condition; data is limited accordingly before evaluating expr - but the result of expr is also returned as NA for levels not existing in the subset. See Examples.
use.levels	logical; if TRUE, uses factor levels of given variables if present; if you want e.g. counts for levels that actually have zero observations but are levels in a factor variable, use this
na.rm	logical; if TRUE, drops rows in table that have NA as values in any of by.vars columns
robust	logical; if TRUE, runs the output data's by.vars columns through robust_values before outputting
.SDcols	advanced; a character vector of column names passed to inside the data.table's brackets DT[, , ...]; see [data.table::data.table]; if NULL, uses all appropriate columns. See Examples for usage.
enclos	advanced; an environment; the enclosing environment of the data.
...	advanced; other arguments passed to inside the data.table's brackets DT[, , ...]; see [data.table::data.table]

Details

Returns expr for each unique combination of given by.vars.

By default makes use of any and all [levels] present for each variable in by.vars. This is useful, because even if a subset of the data does not contain observations for e.g. a specific age group, those age groups are nevertheless presented in the resulting table; e.g. with the default expr = list(obs = .N) all age group levels are represented by a row and can have obs = 0.

The function differs from the vanilla [table] by giving a long format table of values regardless of the number of by.vars given. Make use of e.g. [cast_simple] if data needs to be presented in a wide format (e.g. a two-way table).

The rows of the long-format table are effectively Cartesian products of the levels of each variable in by.vars, e.g. with by.vars = c("sex", "area") all levels of area are repeated for both levels of sex in the table.

The expr allows the user to apply any function(s) on all levels defined by by.vars. Here are some examples:

- .N or list(.N) is a function used inside a data.table to calculate counts in each group
- list(obs = .N), same as above but user assigned variable name
- list(sum(obs), sum(pyrs), mean(dg_age)), multiple objects in a list
- list(obs = sum(obs), pyrs = sum(pyrs)), same as above with user defined variable names

If use.levels = FALSE, no levels information will be used. This means that if e.g. the agegroup variable is a factor and has 18 levels defined, but only 15 levels are present in the data, no rows for the missing levels will be shown in the table.

na.rm simply drops any rows from the resulting table where any of the by.vars values was NA.

Value

A data.table of statistics (e.g. counts) stratified by the columns defined in by.vars.

Functions

- expr.by.cj(): Somewhat more streamlined ltable with defaults for speed. Explicit determination of enclosing environment of data.

Author(s)

Joonas Miettinen, Matti Rantanen

See Also

[table], [cast_simple], [data.table::melt]

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
data("sire", package = "popEpi")
sr <- sire
sr$agegroup <- cut(sr$dg_age, breaks=c(0,45,60,75,85,Inf))
## counts by default
ltable(sr, "agegroup")

## any expression can be given
ltable(sr, "agegroup", list(mage = mean(dg_age)))
ltable(sr, "agegroup", list(mage = mean(dg_age), vage = var(dg_age)))

## also returns levels where there are zero rows (expressions as NA)
ltable(sr, "agegroup", list(obs = .N,
                            minage = min(dg_age),
                            maxage = max(dg_age)),
       subset = dg_age < 85)

#### expr.by.cj
expr.by.cj(sr, "agegroup")

## any arbitrary expression can be given
expr.by.cj(sr, "agegroup", list(mage = mean(dg_age)))
expr.by.cj(sr, "agegroup", list(mage = mean(dg_age), vage = var(dg_age)))
```

```
## only uses levels of by.vars present in data
expr.by.cj(sr, "agegroup", list(mage = mean(dg_age), vage = var(dg_age)),
           subset = dg_age < 70)

## .SDcols trick
expr.by.cj(sr, "agegroup", lapply(.SD, mean),
           subset = dg_age < 70, .SDcols = c("dg_age", "status"))
```

`meanpop_fi` *Mean population counts in Finland year, sex, and age group.*

Description

Mean population counts in Finland year, sex, and age group.

Format

`data.table` with columns

- `sex` gender coded as male, female (0, 1)
- `year` calendar year 1981-2016
- `agegroup` - coded 0 to 100; one-year age groups
- `meanpop` the mean population count; that is, the mean of the annual population counts of two consecutive years; e.g. for 1990 `meanpop` is the mean of population counts for 1990 and 1991 (counted at 1990-01-01 and 1991-01-01, respectively)

Source

Statistics Finland

See Also

Other popEpi data: `ICSS`, `popmort`, `sibr`, `sire`, `stdpop101`, `stdpop18`

`na2zero` *Convert NA's to zero in data.table*

Description

Given a `data.table DT`, replaces any `NA` values in the variables given in `vars` in `DT`. Takes a copy of the original data and returns the modified copy.

Usage

```
na2zero(DT, vars = NULL)
```

Arguments

DT	data.table object
vars	a character string vector of variables names in DT; if NULL, uses all variable names in DT

Details

Given a data.table object, converts NA values to numeric (double) zeros for all variables named in vars or all variables if vars = NULL.

Value

A copy of DT where NA values have been replaced with zero.

Author(s)

Joonas Miettinen

---

plot.rate *plot method for rate object*

Description

Plot rate estimates with confidence intervals lines using R base graphics

Usage

```
## S3 method for class 'rate'
plot(x, conf.int = TRUE, eps = 0.2, left.margin, xlim, ...)
```

Arguments

x	a rate object (see [rate])
conf.int	logical; default TRUE draws the confidence intervals
eps	is the height of the ending of the error bars
left.margin	set a custom left margin for long variable names. Function tries to do it by default.
xlim	change the x-axis location
...	arguments passed on to graphical functions points and segment (e.g. col, lwd, pch and cex)

Details

This is limited explanatory tool but most graphical parameters are user adjustable.

Value

Always returns NULL invisibly. This function is called for its side effects.

Author(s)

Matti Rantanen

---

plot.sir *Plot method for sir-object*

Description

Plot SIR estimates with error bars

Usage

```
## S3 method for class 'sir'
plot(
  x,
  conf.int = TRUE,
  ylab,
  xlab,
  xlim,
  main,
  eps = 0.2,
  abline = TRUE,
  log = FALSE,
  left.margin,
  ...
)
```

Arguments

x	an object returned by function sir
conf.int	default TRUE draws confidence intervals
ylab	overwrites default y-axis label
xlab	overwrites default x-axis label
xlim	x-axis minimum and maximum values
main	optional plot title
eps	error bar vertical bar height (works only in 'model' or 'univariate')
abline	logical; draws a grey line in SIR = 1
log	logical; SIR is not in log scale by default
left.margin	adjust left marginal of the plot to fit long variable names
...	arguments passed on to plot(), segment and lines()

Details

Plot SIR estimates and confidence intervals

- univariate - plots SIR with univariate confidence intervals
- model - plots SIR with Poisson modelled confidence intervals

Customize Normal plot parameters can be passed to `plot`. These can be a vector when plotting error bars:

- `pch` - point type
- `lty` - line type
- `col` - line/point colour
- `lwd` - point/line size

Tips for plotting splines It's possible to use `plot` to first draw the confidence intervals using specific line type or colour and then plotting again the estimate using `lines(... , conf.int = FALSE)` with different settings. This works only when `plot.type` is 'splines'.

Value

Always returns `NULL` invisibly. This function is called for its side effects.

Author(s)

Matti Rantanen

See Also

`[sir]`, `[sirspline]`

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)

# Plot SIR estimates
# plot(sir.by.gender, col = c(4,2), log=FALSE, eps=0.2, lty=1, lwd=2, pch=19,
#      main = 'SIR by gender', abline=TRUE)
```

`plot.sirspline` `plot` *method for sirspline-object*

Description

Plot SIR splines using R base graphics.

Usage

```
## S3 method for class 'sirspline'
plot(x, conf.int = TRUE, abline = TRUE, log = FALSE, type, ylab, xlab, ...)
```

Arguments

`x`	an object returned by function sirspline
`conf.int`	logical; default TRUE draws also the 95 confidence intervals
`abline`	logical; draws a reference line where SIR = 1
`log`	logical; default FALSE. Should the y-axis be in log scale
`type`	select `type = 'n'` to plot only figure frames
`ylab`	overwrites default y-axis label; can be a vector if multiple splines fitted
`xlab`	overwrites default x-axis label; can be a vector if multiple splines fitted
`...`	arguments passed on to plot()

Details

In `plot.sirspline` almost every graphical parameter are user adjustable, such as `ylim`, `xlim`. `plot.sirsplines` calls `lines.splines` to add lines.

The plot axis without lines can be plotted using option `type = 'n'`. On top of the frame it's then possible to add a `grid`, `abline` or text before plotting the lines (see: `sirspline`).

Value

Always returns `NULL` invisibly. This function is called for its side effects.

Author(s)

Matti Rantanen

See Also

Other sir functions: `lines.sirspline()`, `sir()`, `sir_exp()`, `sir_ratio()`, `sirspline()`

plot.survmean *Graphically Inspect Curves Used in Mean Survival Computation*

Description

Plots the observed (with extrapolation) and expected survival curves for all strata in an object created by [survmean]

Usage

```
## S3 method for class 'survmean'
plot(x, ...)
```

Arguments

x	a survmean object
...	arguments passed (ultimately) to matlines; you may, therefore, supply e.g. xlab through this, though arguments such as lty and col will not work

Details

For examples see [survmean]. This function is intended only for graphically inspecting that the observed survival curves with extrapolation and the expected survival curves have been sensibly computed in survmean.

If you want finer control over the plotted curves, extract the curves from the survmean output using

```
attr(x, "curves")
```

where x is a survmean object.

Value

Always returns NULL invisibly. This function is called for its side effects.

Author(s)

Joonas Miettinen

See Also

Other survmean functions: Surv(), lines.survmean(), survmean()

plot.survtab plot *method for survtab objects*

Description

Plotting for survtab objects

Usage

```
## S3 method for class 'survtab'
plot(
  x,
  y = NULL,
  subset = NULL,
  conf.int = TRUE,
  col = NULL,
  lty = NULL,
  ylab = NULL,
  xlab = NULL,
  ...
)
```

Arguments

x	a survtab output object
y	survival a character vector of a variable names to plot; e.g. y = "r.e2"
subset	a logical condition; obj is subset accordingly before plotting; use this for limiting to specific strata, e.g. subset = sex == "male"
conf.int	logical; if TRUE, also plots any confidence intervals present in obj for variables in y
col	line colour; one value for each stratum; will be recycled
lty	line type; one value for each stratum; will be recycled
ylab	label for Y-axis
xlab	label for X-axis
...	additional arguments passed on to plot and lines.survtab; e.g. ylim can be defined this way

Value

Always returns NULL invisibly. This function is called for its side effects.

Author(s)

Joonas Miettinen

See Also

Other survtab functions: `Surv()`, `lines.survtab()`, `print.survtab()`, `summary.survtab()`, `survtab()`, `survtab_ag()`

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
data(sire)
data(sibr)
si <- rbind(sire, sibr)
si$period <- cut(si$dg_date, as.Date(c("1993-01-01", "2004-01-01", "2013-01-01")), right = FALSE)
si$cancer <- c(rep("rectal", nrow(sire)), rep("breast", nrow(sibr)))
x <- lexpand(si, birth = bi_date, entry = dg_date, exit = ex_date,
             status = status %in% 1:2,
             fot = 0:5, aggre = list(cancer, period, fot))
st <- survtab_ag(fot ~ cancer + period, data = x,
                 surv.method = "lifetable", surv.type = "surv.obs")

plot(st, "surv.obs", subset = cancer == "breast", ylim = c(0.5, 1), col = "blue")
lines(st, "surv.obs", subset = cancer == "rectal", col = "red")

## or
plot(st, "surv.obs", col = c(2,2,4,4), lty = c(1, 2, 1, 2))
```

`poisson.ci` *Get rate and exact Poisson confidence intervals*

Description

Computes confidence intervals for Poisson rates

Usage

```
poisson.ci(x, pt = 1, conf.level = 0.95)
```

Arguments

`x`	observed
`pt`	expected
`conf.level`	alpha level

Value

A `data.frame` with columns

- `x`: arg `x`
- `pt`: arg `pt`
- `rate`: result of `x / pt`
- `lower`: lower bound of CI
- `upper`: upper bound of CI
- `conf.level`: arg `conf.level`

Author(s)

epitools

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)

poisson.ci(x = 4, pt = 5, conf.level = 0.95)
```

---

pophaz *Expected / Population Hazard Data Sets Usage in* **popEpi**

Description

Several functions in **popEpi** make use of population or expected hazards in computing the intended estimates (e.g. `[survtab]`). This document explains using such data sets in this package.

Details

Population hazard data sets (pophaz for short) in **popEpi** should be `data.frames` in the "long" format where one of the columns must be named `haz` (for hazard), and other columns define the values or levels in variables relating to subjects in your data. For example, `[popmort]` contains Finnish population mortality hazards by sex, calendar year, and 1-year age group.

sex	year	agegroup	haz
0	1951	0	0.036363176
0	1951	1	0.003616547
0	1951	2	0.002172384
0	1951	3	0.001581249
0	1951	4	0.001180690
0	1951	5	0.001070595

The names of the columns should match to the names of the variables that you have in your subject-level data. Time variables in your pophaz may also correspond to `Lexis` time scales; see `[survtab]`.

Any time variables (as they usually have) should be coded consistently: When using fractional years in your data, the time variables in your pophaz must also be coded in fractional years. When using e.g. `Dates` in your data, ensure that the pophaz time variables are coded at the level of days (or `Dates` for calendar time).

The `haz` variable in your pophaz should also be coded consistently with the used time variables. E.g. `haz` values in life-tables reported as deaths per person-year should be multiplied by 365.25 when using day-level time variables. Typically you'll have calendar time and age expressed in years, which means `haz` should be expressed as the number of deaths per person-year.

If you have your population hazards in a `ratetable` object usable by functions in **survival** and **relsurv**, you may transform them to long-format `data.frames` using `[ratetable_to_long_dt]`. Ensure, however, that the created `haz` column is coded at the right level (events per days or years typically).

National statistical institutions, the WHO, and e.g. the Human Life-Table Database supply life-table data.

Author(s)

Joonas Miettinen

`popmort` — *Population mortality rates in Finland 1951 - 2013 in 101 age groups and by gender. This is an example of a population hazard table as used in* **popEpi***; for the general help page, see* `[pophaz]`.

Description

Population mortality rates in Finland 1951 - 2013 in 101 age groups and by gender. This is an example of a population hazard table as used in **popEpi**; for the general help page, see `[pophaz]`.

Format

`data.table` with columns

- `sex` gender coded as male, female (0, 1)
- `year` calendar year
- `agegroup` - coded 0 to 100; one-year age groups
- `haz` the average population mortality rate per person-year (d/(pyrs), where d is the number of deaths and pyrs is the person-years)

Source

Statistics Finland

See Also

[pophaz]

Other popEpi data: ICSS, meanpop_fi, sibr, sire, stdpop101, stdpop18

---

prepExpo *Prepare Exposure Data for Aggregation*

Description

prepExpo uses a Lexis object of periods of exposure to fill gaps between the periods and overall entry and exit times without accumulating exposure time in periods of no exposure, and splits the result if requested.

Usage

```
prepExpo(
  lex,
  freezeScales = "work",
  cutScale = "per",
  entry = min(get(cutScale)),
  exit = max(get(cutScale)),
  by = "lex.id",
  breaks = NULL,
  freezeDummy = NULL,
  subset = NULL,
  verbose = FALSE,
  ...
)
```

Arguments

lex	a [Epi::Lexis] object with ONLY periods of exposure as rows; one or multiple rows per subject allowed
freezeScales	a character vector naming Lexis time scales of exposure which should be frozen in periods where no exposure occurs (in the gap time periods)
cutScale	the Lexis time scale along which the subject-specific ultimate entry and exit times are specified
entry	an expression; the time of entry to follow-up which may be earlier, at, or after the first time of exposure in freezeScales; evaluated separately for each unique combination of by, so e.g. with entry = min(Var1) and by = "lex.id" it sets the lex.id-specific minima of Var1 to be the original times of entry for each lex.id
exit	the same as entry but for the ultimate exit time per unique combination of by

by	a character vector indicating variable names in lex, the unique combinations of which identify separate subjects for which to fill gaps in the records from entry to exit; for novices of [data.table::data.table], this is passed to a data.table's by argument.
breaks	a named list of breaks; e.g. list(work = 0:20,per = 1995:2015); passed on to [splitMulti] so see that function's help for more details
freezeDummy	a character string; specifies the name for a dummy variable that this function will create and add to output which identifies rows where the freezeScales are frozen and where not (0 implies not frozen, 1 implies frozen); if NULL, no dummy is created
subset	a logical condition to subset data by before computations; e.g. subset = sex == "male"
verbose	logical; if TRUE, the function is chatty and returns some messages and timings during its run.
...	additional arguments passed on to [splitMulti]

Details

prepExpo is a convenience function for the purpose of eventually aggregating person-time and events in categories of not only normally progressing [Epi::Lexis] time scales but also some time scales which should not progress sometimes. For example a person may work at a production facility only intermittently, meaning exposure time (to work-related substances for example) should not progress outside of periods of work. This allows for e.g. a correct aggregation of person-time and events by categories of cumulative time of exposure.

Given an [Epi::Lexis] object containing rows (time lines) where a subject is exposed to something (and NO periods without exposure), fills any gaps between exposure periods for each unique combination of by and the subject-specific "ultimate" entry and exit times, "freezes" the cumulative exposure times in periods of no exposure, and splits data using breaks passed to [splitMulti] if requested. Results in a (split) Lexis object where freezeScales do not progress in time periods where no exposure was recorded in lex.

This function assumes that entry and exit arguments are the same for each row within a unique combination of variables named in by. E.g. with by = "lex.id" only each lex.id has a unique value for entry and exit at most.

The supplied breaks split the data using splitMulti, with the exception that breaks supplied concerning any frozen time scales ONLY split the rows where the time scales are not frozen. E.g. with freezeScales = "work", breaks = list(work = 0:10, cal = 1995:2010) splits all rows over "cal" but only non-frozen rows over "work".

Only supports frozen time scales that advance and freeze contemporaneously: e.g. it would not currently be possible to take into account the cumulative time working at a facility and the cumulative time doing a single task at the facility, if the two are not exactly the same. On the other hand one might use the same time scale for different exposure types, supply them as separate rows, and identify the different exposures using a dummy variable.

Value

Returns a Lexis object that has been split if breaks is specified. The resulting time is also a data.table if options("popEpi.datatable") == TRUE (see: ?popEpi)

print.aggre *Print an* aggre *Object*

Description

Print method function for aggre objects; see [as.aggre] and [aggre].

Usage

```
## S3 method for class 'aggre'
print(x, subset = NULL, ...)
```

Arguments

x	an aggre object
subset	a logical condition to subset results table by before printing; use this to limit to a certain stratum. E.g. subset = sex == "male"
...	arguments passed to print.data.table; try e.g. top = 2 for numbers of rows in head and tail printed if the table is large, nrow = 100 for number of rows to print, etc.

Value

Always returns NULL invisibly. This function is called for its side effects.

Author(s)

Joonas Miettinen

print.rate *Print an rate object*

Description

Print method function for rate objects; see [rate].

Usage

```
## S3 method for class 'rate'
print(x, subset = NULL, ...)
```

Arguments

x	an rate object
subset	a logical condition to subset results table by before printing; use this to limit to a certain stratum. E.g. subset = sex == "female"
...	arguments for data.tables print method, e.g. row.names = FALSE suppresses row numbers.

Value

Always returns NULL invisibly. This function is called for its side effects.

Author(s)

Matti Rantanen

---

print.survtab *Print a survtab Object*

Description

Print method function for survtab objects; see [survtab_ag].

Usage

```
## S3 method for class 'survtab'
print(x, subset = NULL, ...)
```

Arguments

x	a survtab object
subset	a logical condition to subset results table by before printing; use this to limit to a certain stratum. E.g. subset = sex == "male"
...	arguments passed to print.data.table; try e.g. top = 2 for numbers of rows in head and tail printed if the table is large, nrow = 100 for number of rows to print, etc.

Value

Always returns NULL invisibly. This function is called for its side effects.

Author(s)

Joonas Miettinen

See Also

Other survtab functions: Surv(), lines.survtab(), plot.survtab(), summary.survtab(), survtab(), survtab_ag()

rate *Direct-Standardised Incidence/Mortality Rates*

Description

rate calculates adjusted rates using preloaded weights data or user specified weights.

Usage

```
rate(
  data,
  obs = NULL,
  pyrs = NULL,
  print = NULL,
  adjust = NULL,
  weights = NULL,
  subset = NULL
)
```

Arguments

data	aggregated data (see e.g. [lexpand], [aggre] if you have subject-level data)
obs	observations variable name in data. Flexible input, typically e.g. obs = obs.
pyrs	person-years variable name in data. Flexible input, typically e.g. pyrs = pyrs.
print	variable name to stratify the rates. Flexible input, typically e.g. print = sex or print = list(sex, area).
adjust	variable for adjusting the rates. Flexible input, typically e.g. adjust = agegroup.
weights	typically a list of weights or a character string specifying an age group standardization scheme; see the dedicated help page and examples.
subset	a logical expression to subset data.

Details

Input data needs to be in aggregated format with observations and person-years. For individual data use [lexpand], or [ltable] and merge person-years manually.

The confidence intervals are based on the normal approximation of the logarithm of the rate. The variance of the log rate that is used to derive the confidence intervals is derived using the delta method.

Value

Returns a data.table with observations, person-years, rates and adjusted rates, if available. Results are stratified by print. Adjusted rates are identified with suffix .adj and .lo and .hi are for confidence intervals lower and upper 95% bounds, respectively. The prefix SE. stands for standard error.

Author(s)

Matti Rantanen, Joonas Miettinen

See Also

[lexpand], [ltable]

Other main functions: Surv(), relpois(), relpois_ag(), sir(), sirspline(), survmean(), survtab(), survtab_ag()

Other rate functions: rate_ratio()

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
## Prepare data with lexpand and then reformat agegroup.
data(sibr)
x <- lexpand(sibr, birth = bi_date, entry = dg_date, exit = ex_date,
             breaks = list(per = c(1990,2000,2010,2020), age = c(0:17*5,Inf)),
             aggre = list(agegroup = age, year.cat = per),
             status =  status != 0)

x$agegroup <- cut(x$agegroup,  c(0:17*5,Inf), right = FALSE)

## calculate rates for selected periods with Nordic 2000 weights:
r1 <- rate( data = x, obs = from0to1, pyrs = pyrs, print = year.cat,
            adjust = agegroup, weights = 'nordic')
r1

## use total person-years by stratum as weights (some have zero)
w <- ltable(x, by.vars = "agegroup", expr = sum(pyrs))
w[is.na(w$V1),]$V1 <- 0

r2 <- rate( data = x, obs = from0to1, pyrs = pyrs, print = year.cat,
            adjust = agegroup,
            weights = w$V1)
r2

## use data.frame of weights:
names(w) <- c("agegroup", "weights")
r2 <- rate( data = x, obs = from0to1, pyrs = pyrs, print = year.cat,
            adjust = agegroup,
            weights = w)
r2

## internal weights (same result as above)
r3 <- rate( data = x, obs = from0to1, pyrs = pyrs, print = year.cat,
            adjust = agegroup,
            weights = "internal")
```

```
r3
```

`rate_ratio` *Confidence intervals for the rate ratios*

Description

Calculate rate ratio with confidence intervals for rate objects or observations and person-years.

Usage

```
rate_ratio(x, y, crude = FALSE, SE.method = TRUE)
```

Arguments

`x`	a rate-object, vector of two; rate and standard error or observed and person-years.
`y`	a rate-object, vector of two; rate and standard error or observed and person-years.
`crude`	set TRUE to use crude rates; default is FALSE.
`SE.method`	default TRUE; if `x` and `y` are vectors of observed and person-years, this must be changed to FALSE.

Details

Calculate rate ratio of two age standardized rate objects (see `[rate]`). Multiple rates for each objects is supported if there are an equal number of rates. Another option is to set `x` and `y` as a vector of two.

1. rate and its standard error, and set `SE.method = TRUE`.
2. observations and person-year, and set `SE.method = FALSE`.

See examples.

Value

A vector length of three: rate_ratio, and lower and upper confidence intervals.

Author(s)

Matti Rantanen

See Also

`[rate]`

Other rate functions: `rate()`

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)

# two rate ratios; silly example with female rectal / breast cancer
## mortality rates
data("sire", package = "popEpi")
data("sibr", package = "popEpi")

BL <- list(per = 2000:2005)

re <- lexpand(sire, birth = "bi_date", entry = "dg_date", exit = "ex_date",
              status = status == 1, breaks = BL, aggre = list(per))
br <- lexpand(sibr, birth = "bi_date", entry = "dg_date", exit = "ex_date",
              status = status == 1, breaks = BL, aggre = list(per))

r_re <- rate(re, obs = "from0to1", pyrs = "pyrs")
r_br <- rate(br, obs = "from0to1", pyrs = "pyrs")

rate_ratio(r_re, r_br, SE.method = TRUE)


# manually set rates (0.003 and 0.005) and SEs (0.001 and 0.002)
# so that x = y = c('rate', 'SE')
rate_ratio(x= c(0.003, 0.001), y= c(0.005, 0.002), SE.method = TRUE)

# observed numbers (10 and 20) and person-years (30000 and 40000):
rate_ratio(x = c(10, 30000), y = c(20, 40000), SE.method = FALSE)
```

`relpois` *Excess hazard Poisson model*

Description

Estimate a Poisson piecewise constant excess hazards model

Usage

```
relpois(data, formula, fot.breaks = NULL, subset = NULL, check = TRUE, ...)
```

Arguments

`data`	a dataset split with e.g. `[lexpand]`; must have expected hazard merged within
`formula`	a formula which is passed on to `glm`; see Details

fot.breaks	optional; a numeric vector of [a,b) breaks to specify survival intervals over the follow-up time; if NULL, the existing breaks along the mandatory fot time scale in data are used (e.g. the breaks for fot supplied to lexpand)
subset	a logical vector or condition; e.g. subset = sex == 1; limits the data before estimation
check	logical; if TRUE, tabulates excess cases by all factor variables in the formula to check for negative / NA excess cases before fitting the GLM
...	any argument passed on to glm

Details

Basics

relpois employs a custom link function of the Poisson variety to estimate piecewise constant parametric excess hazards. The pieces are determined by fot.breaks. A log(person-years) offset is passed automatically to the glm call.

Formula usage

The formula can be used like any ordinary glm formula. The user must define the outcome in some manner, which is usually lex.Xst after splitting with e.g. lexpand. The exception is the possibility of including the baseline excess hazard terms by including the reserved term FOT in the formula.

For example, lex.Xst != 0 ~ FOT + agegr estimates a model with constant excess hazards at the follow-up intervals as specified by the pertinent breaks used in splitting data, as well as for the different age groups. FOT is created ad hoc if it is used in the formula. If you leave out FOT, the hazard is effectively assumed to be constant across the whole follow-up time.

You can also simply use your own follow-up time interval variable that you have created before calling relpois. However, when using FOT, relpois automatically checks for e.g. negative excess cases in follow-up intervals, allowing for quickly finding splitting breaks where model estimation is possible. It also drops any data outside the follow-up time window.

Splitting and merging population hazard

The easiest way to both split and to include population hazard information is by using [lexpand]. You may also fairly easily do it by hand by splitting first and then merging in your population hazard information.

Data requirements

The population hazard information must be available for each record and named pop.haz. The follow-up time variable must be named "fot" e.g. as a result of using lexpand. The lex.dur variable must also be present, containing person-year information.

Value

A glm object created using a custom Poisson family construct. Some glm methods are applicable.

Author(s)

Joonas Miettinen, Karri Seppa

References

Paul W Dickman, Andy Sloggett, Michael Hills, and Timo Hakulinen. Regression models for relative survival. Stat Med. 2004 Jan 15;23(1):51-64. doi:10.1002/sim.1597

See Also

[lexpand], [stats::poisson], [stats::glm]

Other main functions: Surv(), rate(), relpois_ag(), sir(), sirspline(), survmean(), survtab(), survtab_ag()

Other relpois functions: RPL, relpois_ag(), rpcurve()

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
## use the simulated rectal cancer cohort
data("sire", package = "popEpi")
sire$agegr <- cut(sire$dg_age, c(0,45,60,Inf), right=FALSE)

## usable straight away after splitting
fb <- c(0,3/12,6/12,1,2,3,4,5)
x <- lexpand(sire, birth = bi_date, entry = dg_date,
             exit = ex_date, status=status,
             breaks = list(fot=fb), pophaz=popmort)
rpm <- relpois(x, formula = lex.Xst %in% 1:2 ~ FOT + agegr)

## some methods for glm work. e.g. test for interaction

rpm2 <- relpois(x, formula = lex.Xst %in% 1:2 ~ FOT*agegr)
anova(rpm, rpm2, test="LRT")
AIC(rpm, rpm2)
## update() won't work currently
```

---

relpois_ag *Excess hazard Poisson model*

Description

Estimate a Poisson Piecewise Constant Excess Hazards Model

Usage

```
relpois_ag(
  formula,
  data,
  d.exp,
  offset = NULL,
  breaks = NULL,
  subset = NULL,
  piecewise = TRUE,
  check = TRUE,
  ...
)
```

Arguments

formula	a formula with the counts of events as the response. Passed on to glm. May contain usage of the offset() function instead of supplying the offset for the Poisson model via the argument offset.
data	an aggre object (an aggregated data set; see [as.aggre] and [aggre])
d.exp	the counts of expected cases. Mandatory. E.g. d.exp = EXC_CASES, where EXC_CASES is a column in data.
offset	the offset for the Poisson model, supplied as e.g. offset = log(PTIME), where PTIME is a subject-time variable in data. Not mandatory, but almost always should be supplied.
breaks	optional; a numeric vector of [a,b) breaks to specify survival intervals over the follow-up time; if NULL, the existing breaks along the mandatory time scale mentioned in formula are used
subset	a logical vector or condition; e.g. subset = sex == 1; limits the data before estimation
piecewise	logical; if TRUE, and if any time scale from data is used (mentioned) in the formula, the time scale is transformed into a factor variable indicating intervals on the time scale. Otherwise the time scale left as it is, usually a numeric variable. E.g. if formula = counts ~ TS1*VAR1, TS1 is transformed into a factor before fitting model.
check	logical; if TRUE, performs check on the negativity excess cases by factor-like covariates in formula - negative excess cases will very likely lead to non-converging model
...	any other argument passed on to [stats::glm] such as control or weights

Value

A relpois object created using a custom Poisson family construct.

Author(s)

Joonas Miettinen, Karri Seppa

See Also

[lexpand], [poisson], [glm]

Other main functions: Surv(), rate(), relpois(), sir(), sirspline(), survmean(), survtab(), survtab_ag()

Other relpois functions: RPL, relpois(), rpcurve()

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
## use the simulated rectal cancer cohort
data(sire, package = "popEpi")
sire$agegr <- cut(sire$dg_age, c(0,45,60,Inf), right=FALSE)

## create aggregated example data
fb <- c(0,3/12,6/12,1,2,3,4,5)
x <- lexpand(sire, birth = bi_date, entry = dg_date,
             exit = ex_date, status=status %in% 1:2,
             breaks = list(fot=fb),
             pophaz=popmort, pp = FALSE,
             aggre = list(agegr, fot))

## fit model using aggregated data
rpm <- relpois_ag(formula = from0to1 ~ fot + agegr,  data = x,
                  d.exp = d.exp, offset = log(pyrs))
summary(rpm)

## the usual functions for handling glm models work
rpm2 <- update(rpm, . ~ fot*agegr)
anova(rpm, rpm2, test="LRT")
AIC(rpm, rpm2)

## other features such as residuals or predicting are not guaranteed
## to work as intended.
```

---

robust_values *Convert values to numeric robustly*

Description

Brute force solution for ensuring a variable is numeric by coercing a variable of any type first to factor and then to numeric

Usage

```
robust_values(num.values, force = FALSE, messages = TRUE)
```

Arguments

num.values	values to convert to numeric
force	logical; if TRUE, returns a vector of values where values that cannot be interpreted as numeric are set to NA; if FALSE, returns the original vector and gives a warning if any value cannot be interpreted as numeric.
messages	logical; if TRUE, returns a message of what was done with the num.values

Value

A numeric vector.

Note

Returns NULL if given num.values is NULL.

Author(s)

Joonas Miettinen

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
## this works
values <- c("1", "3", "5")
values <- robust_values(values)

## this works
values <- c("1", "3", "5", NA)
values <- robust_values(values)

## this returns originals and throws warnings
values <- c("1", "3", "5", "a")
suppressWarnings(
  values <- robust_values(values)
)


## this forces "a" to NA and works otherwise; throws warning about NAs
values <- c("1", "3", "5", "a")
suppressWarnings(
  values <- robust_values(values, force=TRUE)
)
```

rpcurve *Marginal piecewise parametric relative survival curve*

Description

Fit a marginal relative survival curve based on a `relpois` fit

Usage

```
rpcurve(object)
```

Arguments

`object`	a `relpois` object

Details

Estimates a marginal curve, i.e. the average of all possible individual curves.

Only supported when the reserved `FOT` variable was used in `relpois`. Computes a curve for each unique combination of covariates (e.g. 4 sets) and returns a weighted average curve based on the counts of subjects for each combination (e.g. 1000, 125, 50, 25 respectively). Fairly fast when only categorical variables have been used, otherwise go get a cup of coffee.

If delayed entry is present in data due to period analysis limiting, the marginal curve is constructed only for those whose follow-up started in the respective period.

Value

A `data.table` of relative survival curves.

Author(s)

Joonas Miettinen

See Also

Other relpois functions: `RPL`, `relpois()`, `relpois_ag()`

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)

## use the simulated rectal cancer cohort
data("sire", package = "popEpi")
ab <- c(0,45,55,65,70,Inf)
```

```
sire$agegr <- cut(sire$dg_age, breaks = ab, right = FALSE)

BL <- list(fot= seq(0,10,1/12))
pm <- data.frame(popEpi::popmort)
x <- lexpand(sire, breaks=BL, pophaz=pm,
             birth = bi_date,
             entry = dg_date, exit = ex_date,
             status  = status %in% 1:2)

rpm <- relpois(x, formula = lex.Xst %in% 1:2 ~ -1+ FOT + agegr,
               fot.breaks=c(0,0.25,0.5,1:8,10))
pmc <- rpcurve(rpm)

## compare with non-parametric estimates
names(pm) <- c("sex", "per", "age", "haz")
x$agegr <- cut(x$dg_age, c(0,45,55,65,75,Inf), right = FALSE)
st <- survtab(fot ~ adjust(agegr), data = x, weights = "internal",
              pophaz = pm)


plot(st, y = "r.e2.as")
lines(y = pmc$est, x = pmc$Tstop, col="red")
```

RPL *Relative Poisson family object*

Description

A family object for GLM fitting of relative Poisson models

Usage

```
RPL
```

Format

A list very similar to that created by `poisson()`.

Author(s)

Karri Seppa

See Also

Other relpois functions: `relpois()`, `relpois_ag()`, `rpcurve()`

setaggre *Set* aggre *attributes to an object by modifying in place*

Description

Coerces an R object to an aggre object, identifying the object as one containing aggregated counts, person-years and other information. setaggre modifies in place without taking any copies. Retains all other attributes.

Usage

```
setaggre(x, values = NULL, by = NULL, breaks = NULL)
```

Arguments

x	a data.frame or data.table
values	a character string vector; the names of value variables
by	a character string vector; the names of variables by which values have been tabulated
breaks	a list of breaks, where each element is a breaks vector as usually passed to e.g. [splitLexisDT]. The list must be fully named, with the names corresponding to time scales at the aggregate level in your data. Every unique value in a time scale variable in data must also exist in the corresponding vector in the breaks list.

Details

setaggre sets x to the aggre class in place without taking a copy as e.g. as.data.frame.XXX functions do; see e.g. [data.table::setDT].

Value

Returns x invisibly after setting attributes to it without taking a copy. This function is called for its side effects.

Author(s)

Joonas Miettinen

See Also

Other aggregation functions: aggre(), as.aggre(), lexpand(), summary.aggre()

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
df <- data.frame(sex = rep(c("male", "female"), each = 5),
                 obs = rpois(10, rep(7,5, each=5)),
                 pyrs = rpois(10, lambda = 10000))
## without any breaks
setaggre(df, values = c("obs", "pyrs"), by = "sex")
df <- data.frame(df)
df$FUT <- 0:4
## with breaks list
setaggre(df, values = c("obs", "pyrs"), by = "sex", breaks = list(FUT = 0:5))
```

---

setclass *Set the class of an object (convenience function for* `setattr(obj, "class", CLASS)`*; can add instead of replace*

Description

Sets the class of an object in place to `cl` by replacing or adding

Usage

```
setclass(obj, cl, add = FALSE, add.place = "first")
```

Arguments

`obj`	and object for which to set class
`cl`	class to set
`add`	if `TRUE`, adds `cl` to the classes of the `obj`; otherwise replaces the class information
`add.place`	`"first"` or `"last"`; adds `cl` to the front or to the back of the `obj`'s class vector

Author(s)

Joonas Miettinen

setcolsnull *Delete* data.table *columns if there*

Description

Deletes columns in a data.table conveniently. May only delete columns that are found silently. Sometimes useful in e.g. on.exit expressions.

Usage

```
setcolsnull(
  DT = NULL,
  delete = NULL,
  keep = NULL,
  colorder = FALSE,
  soft = TRUE
)
```

Arguments

DT	a data.table
delete	a character vector of column names to be deleted
keep	a character vector of column names to keep; the rest will be removed; keep overrides delete
colorder	logical; if TRUE, also does setcolorder using keep
soft	logical; if TRUE, does not cause an error if any variable name in keep or delete is missing; soft = FALSE useful for programming sometimes

Value

Always returns NULL invisibly. This function is called for its side effects.

Author(s)

Joonas Miettinen

`sibr` *sibr - a simulated cohort of Finnish female breast cancer patients*

Description

`sibr` is a simulated cohort pertaining female Finnish breast cancer patients diagnosed between 1993-2012. Instead of actual original dates, the dates are masked via modest randomization within several time windows. The dataset is additionally a random sample of 10 000 cases from the pertaining time window.

Format

data.table with columns

- sex - gender of the patient (1 = female)
- bi_date - date of birth
- dg_date - date of cancer diagnosis
- ex_date - date of exit from follow-up (death or censoring)
- status - status of the person at exit; 0 alive; 1 dead due to pertinent cancer; 2 dead due to other causes
- dg_age - age at diagnosis expressed as fractional years

Details

The closing date for the pertinent data was 2012-12-31, meaning status information was available only up to that point — hence the maximum possible `ex_date` is `2012-12-31`.

Author(s)

Karri Seppa

Source

The Finnish Cancer Registry

See Also

Other popEpi data: `ICSS`, `meanpop_fi`, `popmort`, `sire`, `stdpop101`, `stdpop18`

Other survival data: `sire`

`sir` *Calculate SIR or SMR*

Description

Poisson modelled standardised incidence or mortality ratios (SIRs / SMRs) i.e. indirect method for calculating standardised rates. SIR is a ratio of observed and expected cases. Expected cases are derived by multiplying the strata-specific population rate with the corresponding person-years of the cohort.

Usage

```
sir(
  coh.data,
  coh.obs,
  coh.pyrs,
  ref.data = NULL,
  ref.obs = NULL,
  ref.pyrs = NULL,
  ref.rate = NULL,
  subset = NULL,
  print = NULL,
  adjust = NULL,
  mstate = NULL,
  test.type = "homogeneity",
  conf.type = "profile",
  conf.level = 0.95,
  EAR = FALSE
)
```

Arguments

`coh.data`	aggregated cohort data, see e.g. `[lexpand]`
`coh.obs`	variable name for observed cases; quoted or unquoted. A vector when using `mstata`.
`coh.pyrs`	variable name for person years in cohort data; quoted (as a string `'myvar'`) or unquoted (AKA as a name; `myvar`)
`ref.data`	population data. Can be left NULL if `coh.data` is stratified in `print`. See `[pophaz]` for details.
`ref.obs`	variable name for observed cases; quoted or unquoted
`ref.pyrs`	variable name for person-years in population data; quoted or unquoted
`ref.rate`	population rate variable (cases/person-years). Overwrites arguments `ref.pyrs` and `ref.obs`. Quoted or unquoted
`subset`	logical condition to select data from `coh.data` before any computations

print	variable names to stratify results; quoted vector or unquoted named list with functions
adjust	variable names for adjusting without stratifying output; quoted vector or unquoted list
mstate	set column names for cause specific observations; quoted or unquoted. Relevant only when coh.obs length is two or more. See details.
test.type	Test for equal SIRs. Test available are 'homogeneity' and 'trend'.
conf.type	Confidence interval type: 'profile'(=default), 'wald' or 'univariate'.
conf.level	Level of type-I error in confidence intervals, default 0.05 is 95% CI.
EAR	logical; TRUE calculates Excess Absolute Risks for univariate SIRs. (see details)

Details

sir is a comprehensive tool for modelling SIRs/SMRs with flexible options to adjust and print SIRs, test homogeneity and utilize multi-state data. The cohort data and the variable names for observation counts and person-years are required. The reference data is optional, since the cohort data can be stratified (print) and compared to total.

Adjust and print

A SIR can be adjusted or standardised using the covariates found in both coh.data and ref.data. Variable to adjust are given in adjust. Variable names needs to match in both coh.data and ref.data. Typical variables to adjust by are gender, age group and calendar period.

print is used to stratify the SIR output. In other words, the variables assigned to print are the covariates of the Poisson model. Variable levels are treated as categorical. Variables can be assigned in both print and adjust. This means the output it adjusted and printed by these variables.

print can also be a list of expressions. This enables changing variable names or transforming variables with functions such as cut and round. For example, agegroup can be transformed on-the-go with

```
print = list(my_ag = cut(agegroup, my_ag_breaks))
```

ref.rate or ref.obs & ref.pyrs

The population rate variable can be given to the ref.rate parameter. That is, when using e.g. the popmort or a comparable data file, one may supply ref.rate instead of ref.obs and ref.pyrs, which will be ignored if ref.rate is supplied.

Note that if all the stratifying variables in ref.data are not listed in adjust, or when the categories are otherwise combined, the (unweighted) mean of rates is used for computing expected cases. This might incur a small bias in comparison to when exact numbers of observations and person-years are available.

mstate

E.g. using lexpand it's possible to compute counts for several outcomes so that the population at risk is same for each outcome such as a certain kind of cancer. The transition counts are in wide data format, and the relevant columns can be supplied to sir in a vector via the coh.obs argument. The name of the corresponding new column in ref.data is given in mstate. It's recommended to include the mstate variable in adjust, so the corresponding information should also be available in ref.data. More examples in sir-vignette.

This approach is analogous to where SIRs are calculated separately their own function calls.

Other parameters

`univariate` confidence intervals are calculated using exact Poisson intervals (`poisson.ci`). The options `profile` and `wald` are is based on a Poisson regression model: profile-likelihood confidence intervals or Wald's normal-approximation. P-value is Poisson model based `conf.type` or calculated using the method described by Breslow and Day. Function automatically switches to another `conf.type` if calculation is not possible with a message. Usually model fit fails if there is print stratum with zero expected values.

The LRT p-value tests the levels of `print`. The test can be either `"homogeneity"`, a likelihood ratio test where the model variables defined in `print` (factor) is compared to the constant model. Option `"trend"` tests if the linear trend of the continuous variable in `print` is significant (using model comparison).

EAR: Excess Absolute Risk

Excess Absolute Risk is a simple way to quantify the absolute difference between cohort risk and population risk. Make sure that the person-years are calculated accordingly before using EAR. (when using mstate)

Formula for EAR:

$$EAR = \frac{observed - expected}{personyears} \times 1000.$$

Data format

The data should be given in tabulated format. That is the number of observations and person-years are represented for each stratum. Note that also individual data is allowed as long as each observations, person-years, and print and adjust variables are presented in columns. The extra variables and levels are reduced automatically before estimating SIRs. Example of data format:

sex	age	period	obs	pyrs
0	1	2010	0	390
0	2	2010	5	385
1	1	2010	3	308
1	2	2010	12	315

Value

A sir-object that is a `data.table` with meta information in the attributes.

Author(s)

Matti Rantanen, Joonas Miettinen

See Also

`[lexpand]` A SIR calculation vignette

Other sir functions: `lines.sirspline()`, `plot.sirspline()`, `sir_exp()`, `sir_ratio()`, `sirspline()`

Other main functions: `Surv()`, `rate()`, `relpois()`, `relpois_ag()`, `sirspline()`, `survmean()`, `survtab()`, `survtab_ag()`

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
data(popmort)
data(sire)
c <- lexpand( sire, status = status, birth = bi_date, exit = ex_date, entry = dg_date,
              breaks = list(per = 1950:2013, age = 1:100, fot = c(0,10,20,Inf)),
              aggre = list(fot, agegroup = age, year = per, sex) )
## SMR due other causes: status = 2
se <- sir( coh.data = c, coh.obs = 'from0to2', coh.pyrs = 'pyrs',
           ref.data = popmort, ref.rate = 'haz',
           adjust = c('agegroup', 'year', 'sex'), print = 'fot')
se
## for examples see: vignette('sir')
```

---

sir_exp *Calculate SMR*

Description

Calculate Standardized Mortality Ratios (SMRs) using a single data set that includes observed and expected cases and additionally person-years.

`sir_lex` solves SMR from an `[Epi::Lexis]` object calculated with `[lexpand]`.

`sir_ag` solves SMR from a `[aggre]` object calculated using `[lexpand]`.

Usage

```
sir_exp(
  x,
  obs,
  exp,
  pyrs = NULL,
  print = NULL,
  conf.type = "profile",
  test.type = "homogeneity",
  conf.level = 0.95,
  subset = NULL
)

sir_lex(x, print = NULL, breaks = NULL, ...)

sir_ag(
```

```
  x,
  obs = "from0to1",
  print = attr(x, "aggre.meta")$by,
  exp = "d.exp",
  pyrs = "pyrs",
  ...
)
```

Arguments

x	Data set e.g. aggre or Lexis object (see: [lexpand])
obs	Variable name of the observed cases in the data set
exp	Variable name or expression for expected cases
pyrs	Variable name for person-years (optional)
print	Variables or expression to stratify the results
conf.type	select confidence interval type: (default=) profile, wald, univariate
test.type	Test for equal SIRs. Test available are 'homogeneity' and 'trend'
conf.level	Level of type-I error in confidence intervals, default 0.05 is 95% CI
subset	a logical vector for subsetting data
breaks	a named list to split age group (age), period (per) or follow-up (fot).
...	pass arguments to sir_exp

Details

These functions are intended to calculate SMRs from a single data set that includes both observed and expected number of cases. For example utilizing the argument pop.haz of the [lexpand].

sir_lex automatically exports the transition fromXtoY using the first state in lex.Str as 0 and all other as 1. No missing values is allowed in observed, pop.haz or person-years.

Value

A sir object

Author(s)

Matti Rantanen

See Also

[lexpand] A SIR calculation vignette

Other sir functions: lines.sirspline(), plot.sirspline(), sir(), sir_ratio(), sirspline()

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)


BL <- list(fot = 0:5, per = c("2003-01-01","2008-01-01", "2013-01-01"))

## Aggregated data
x1 <- lexpand(sire, breaks = BL, status = status != 0,
              birth = bi_date, entry = dg_date, exit = ex_date,
              pophaz=popmort,
              aggre=list(sex, period = per, surv.int = fot))
sir_ag(x1, print = 'period')


# no aggreate or breaks
x2 <- lexpand(sire, status = status != 0,
              birth = bi_date, entry = dg_date, exit = ex_date,
              pophaz=popmort)
sir_lex(x2, breaks = BL, print = 'per')
```

---

sir_ratio *Confidence intervals for the ratio of two SIRs/SMRs*

Description

Calculate ratio of two SIRs/SMRs and the confidence intervals of the ratio.

Usage

```
sir_ratio(
  x,
  y,
  digits = 3,
  alternative = "two.sided",
  conf.level = 0.95,
  type = "exact"
)
```

Arguments

x	a sir-object or a vector of two; observed and expected cases.
y	a sir-object or a vector of two; observed and expected cases.

digits	number of digits in the output
alternative	The null-hypothesis test: (default:) two.sided, less, greater
conf.level	the type-I error in confidence intervals, default 0.95 for 95% CI.
type	How the binomial confidence intervals are calculated (default:) exact or asymptotic.

Details

Function works with pooled sir-objects i.e. the print argument in sir is ignored. Also x and y can be a vector of two where first index is the observed cases and second is expected cases (see examples). Note that the ratio of two SIR's is only applicable when the age distributions are similar in both populations.

Formula

The observed number of first sir 01 is considered as a Binomial variable with sample size of 01+02. The confidence intervals for Binomial proportion A is solved using exact or asymptotic method. Now the CI for ratio 01/02 is B = A/(1 - A). And further the CI for SIR/SMR is B*E2/E1. (Ederer and Mantel)

Value

A vector length of three: sir_ratio, and lower and upper confidence intervals.

Note

Parameter alternative is always two.sided when parameter type is set to asymptotic.

Author(s)

Matti Rantanen

References

Statistics with Confidence: Confidence Intervals and Statistical Guidelines, Douglas Altman, 2000. ISBN: 978-0-727-91375-3

See Also

[sir] A SIR calculation vignette

Other sir functions: lines.sirspline(), plot.sirspline(), sir(), sir_exp(), sirspline()

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
## Ratio for sir-object and the same values given manually:
```

```
## create example dataset
dt1 <- data.frame(obs = rep(c(5,7), 10),
                  pyrs = rep(c(250,300,350,400), 5),
                  var = 1:20)
Ref <- data.frame(obs = rep(c(50,70,80,100), 5),
                 pyrs = rep(c(2500,3000,3500,4000), 5),
                 var = 1:20)
## sir using the function
s1 <- sir(coh.data = dt1, coh.obs = obs, coh.pyrs = pyrs,
          ref.data = Ref, ref.obs = obs, ref.pyrs = pyrs,
          adjust = var)

## Ratio is simply 1:
sir_ratio(s1, c(120, 150))
```

`sire` *sire - a simulated cohort of Finnish female rectal cancer patients*

Description

`sire` is a simulated cohort pertaining female Finnish rectal cancer patients diagnosed between 1993-2012. Instead of actual original dates, the dates are masked via modest randomization within several time windows.

Format

data.table with columns

- sex - gender of the patient (1 = female)
- bi_date - date of birth
- dg_date - date of cancer diagnosis
- ex_date - date of exit from follow-up (death or censoring)
- status - status of the person at exit; 0 alive; 1 dead due to pertinent cancer; 2 dead due to other causes
- dg_age - age at diagnosis expressed as fractional years

Details

The closing date for the pertinent data was 2012-12-31, meaning status information was available only up to that point — hence the maximum possible `ex_date` is `2012-12-31`.

Author(s)

Karri Seppa

Source

The Finnish Cancer Registry

See Also

Other popEpi data: ICSS, meanpop_fi, popmort, sibr, stdpop101, stdpop18

Other survival data: sibr

`sirspline` *Estimate splines for SIR or SMR*

Description

Splines for standardised incidence or mortality ratio. A useful tool to e.g. check whether a constant SIR can be assumed for all calendar periods, age groups or follow-up intervals. Splines can be fitted for these time dimensions separately or in the same model.

Usage

```
sirspline(
  coh.data,
  coh.obs,
  coh.pyrs,
  ref.data = NULL,
  ref.obs = NULL,
  ref.pyrs = NULL,
  ref.rate = NULL,
  subset = NULL,
  print = NULL,
  adjust = NULL,
  mstate = NULL,
  spline,
  knots = NULL,
  reference.points = NULL,
  dependent.splines = TRUE
)
```

Arguments

`coh.data`	cohort data with observations and at risk time variables
`coh.obs`	variable name for observed cases
`coh.pyrs`	variable name for person-years in cohort data
`ref.data`	aggregated population data
`ref.obs`	variable name for observed cases
`ref.pyrs`	variable name for person-years in population data

`ref.rate` population rate observed/expected. This overwrites the parameters `ref.pyrs` and `ref.obs`.

`subset` logical condition to subset `coh.data` before any computations

`print` variable names for which to estimate SIRs/SMRs and associated splines separately

`adjust` variable names for adjusting the expected cases

`mstate` set column names for cause specific observations. Relevant only when coh.obs length is two or more. See help for `sir`.

`spline` variable name(s) for the splines

`knots` number knots (vector), pre-defined knots (list of vectors) or for optimal number of knots left NULL

`reference.points` fixed reference values for rate ratios. If left `NULL` the smallest value is the reference point (where SIR = 1). Ignored if `dependent.splines = FALSE`

`dependent.splines` logical; if TRUE, all splines are fitted in same model.

Details

See `[sir]` for help on SIR/SMR estimation in general; usage of splines is discussed below.

The spline variables

The model can include one, two or three splines variables. Variables can be included in the same model selecting `dependent.splines = TRUE` and SIR ratios are calculated (first one is the SIR, others SIR ratios). Reference points vector can be set via `reference.points` where first element of the vector is the reference point for first ratio.

Variable(s) to fit splines are given as a vector in argument `spline`. Order will affect the results.

dependent.splines

By default dependent.splines is FALSE and all splines are fitted in separate models. If TRUE, the first variable in `spline` is a function of a SIR and other(s) are ratios.

knots

There are three options to set knots to splines:

Set the number of knots for each spline variable with a **vector**. The knots are automatically placed to the quantiles of observed cases in cohort data. The first and last knots are always the maximum and minimum values, so knot value needs to be at least two.

Predefined knot places can be set with a **list** of vectors. The vector for each spline in the list specifies the knot places. The lowest and the largest values are the boundary knots and these should be checked beforehand.

If `knots` is left **NULL**, the model searches the optimal number of knots by model AIC by fitting models iteratively from 2 to 15 knots and the one with smallest AIC is selected. If `dependent.splines = TRUE`, the number of knots is searched by fitting each spline variable separately.

print

Splines can be stratified by the levels of variable given in `print`. If `print` is a vector, only the first variable is accounted for. The knots are placed globally for all levels of `print`. This also ensures

that the likelihood ratio test is valid. Splines are also fitted independently for each level of `print`. This allows for searching interactions, e.g. by fitting spline for period (`splines='period'`) for each age group (`print = 'agegroup'`).

p-values

The output p-value is a test of whether the splines are equal (homogenous) at different levels of `print`. The test is based on the likelihood ratio test, where the full model includes `print` and is compared to a null model without it. When `(dependent.splines = TRUE)` the p-value returned is a global p-value. Otherwise the p-value is spline-specific.

Value

A list of data.frames and vectors. Three spline estimates are named as `spline.est.A/B/C` and the corresponding values in `spline.seq.A/B/C` for manual plotting

Author(s)

Matti Rantanen, Joonas Miettinen

See Also

`[splitMulti]` A SIR calculation vignette

Other sir functions: `lines.sirspline()`, `plot.sirspline()`, `sir()`, `sir_exp()`, `sir_ratio()`

Other main functions: `Surv()`, `rate()`, `relpois()`, `relpois_ag()`, `sir()`, `survmean()`, `survtab()`, `survtab_ag()`

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
## for examples see: vignette('sir')
```

---

splitLexisDT *Split case-level observations*

Description

Split a `Lexis` object along one time scale (as `[Epi::splitLexis]`) with speed

Usage

```
splitLexisDT(lex, breaks, timeScale, merge = TRUE, drop = TRUE)
```

Arguments

lex	a Lexis object, split or not
breaks	a vector of [a,b) breaks to split data by
timeScale	a character string; name of the time scale to split by
merge	logical; if TRUE, retains all variables from the original data - i.e. original variables are repeated for all the rows by original subject
drop	logical; if TRUE, drops all resulting rows after expansion that reside outside the time window defined by the given breaks

Details

splitLexisDT is in essence a **data.table** version of splitLexis or survSplit for splitting along a single time scale. It requires a Lexis object as input, which may have already been split along some time scale.

Unlike splitLexis, splitLexisDT drops observed time outside the roof and floor of breaks by default - with drop = FALSE the functions have identical behaviour.

The Lexis time scale variables can be of any arbitrary format, e.g. Date, fractional years (see [Epi::cal.yr]) and [get.yrs].

Value

A data.table or data.frame (depending on options("popEpi.datatable"); see ?popEpi) object expanded to accommodate split observations.

Author(s)

Joonas Miettinen

See Also

Other splitting functions: lexpand(), splitMulti()

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
library(Epi)
data("sire", package = "popEpi")
x <- Lexis(data=sire[1000:1100, ],
           entry = list(fot=0, per=get.yrs(dg_date), age=dg_age),
           exit=list(per=get.yrs(ex_date)), exit.status=status)
BL <- list(fot=seq(0, 5, by = 3/12), per=c(2008, 2013))

x2 <- splitMulti(x, breaks = BL, drop = FALSE)
```

```
x3 <- splitLexisDT(x, breaks = BL$fot, timeScale = "fot", drop = FALSE)
x3 <- splitLexisDT(x3, breaks = BL$per, timeScale = "per", drop = FALSE)

x4 <- splitLexis(x,  breaks = BL$fot, time.scale = "fot")
x4 <- splitLexis(x4, breaks = BL$per, time.scale = "per")
## all produce identical results

## using Date variables
x <- Lexis(
  data=sire[1000:1100, ],
  entry = list(fot = 0L, per = dg_date, age = as.integer(dg_date - bi_date)),
  duration = as.integer(ex_date - dg_date),
  entry.status = 0L,
  exit.status = status
)
BL <- list(fot = 0:5*365.25, per = as.Date(c("2008-01-01", "2013-01-01")))

x2 <- splitMulti(x, breaks = BL, drop = FALSE)

x3 <- splitLexisDT(x, breaks = BL$fot, timeScale = "fot", drop = FALSE)
x3 <- splitLexisDT(x3, breaks = BL$per, timeScale = "per", drop = FALSE)

## splitLexis may not work when using Dates
```

`splitMulti` *Split case-level observations*

Description

Split a `Lexis` object along multiple time scales with speed and ease

Usage

```
splitMulti(
  data,
  breaks = NULL,
  ...,
  drop = TRUE,
  merge = TRUE,
  verbose = FALSE
)
```

Arguments

`data`	a Lexis object with event cases as rows
`breaks`	a list of named numeric vectors of breaks; see Details and Examples

...	alternate way of supplying breaks as named vectors; e.g. `fot = 0:5` instead of `breaks = list(fot = 0:5)`; if `breaks` is not `NULL`, `breaks` is used and any breaks passed through ... are NOT used; note also that due to partial matching of argument names in R, if you supply e.g. `dat = my_breaks` and you do not pass argument `data` explicitly (`data = my_data`), then R interprets this as `data = my_breaks` — so choose the names of your time scales wisely
drop	logical; if `TRUE`, drops all resulting rows after expansion that reside outside the time window defined by the given breaks
merge	logical; if `TRUE`, retains all variables from the original data - i.e. original variables are repeated for all the rows by original subject
verbose	logical; if `TRUE`, the function is chatty and returns some messages along the way

Details

`splitMulti` is in essence a **data.table** version of `splitLexis` or `survSplit` for splitting along multiple time scales. It requires a Lexis object as input.

The `breaks` must be a list of named vectors of the appropriate type. The breaks are fully explicit and left-inclusive and right exclusive, e.g. `fot=c(0,5)` forces the data to only include time between `[0,5)` for each original row (unless `drop = FALSE`). Use `Inf` or `-Inf` for open-ended intervals, e.g. `per=c(1990,1995,Inf)` creates the intervals `[1990,1995), [1995, Inf)`.

Instead of specifying `breaks`, one may make use of the ... argument to pass breaks: e.g.

`splitMulti(x, breaks = list(fot = 0:5))`

is equivalent to

`splitMulti(x, fot = 0:5)`.

Multiple breaks can be supplied in the same manner. However, if both `breaks` and ... are used, only the breaks in `breaks` are utilized within the function.

The `Lexis` time scale variables can be of any arbitrary format, e.g. `Date`, fractional years (see `[Epi::cal.yr]`) and `[get.yrs]`, or other.

Value

A `data.table` or `data.frame` (depending on `options("popEpi.datatable")`; see `?popEpi`) object expanded to accommodate split observations.

News for version 0.5.0

Fixed `popEpi::splitMulti` when `merge = FALSE`. Did not include `lex.dur` previously which caused an error.

Author(s)

Joonas Miettinen

See Also

`[Epi::splitLexis]`, `[Epi::Lexis]`, `[survival::survSplit]`

Other splitting functions: `lexpand()`, `splitLexisDT()`

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
#### let's prepare data for computing period method survivals
#### in case there are problems with dates, we first
#### convert to fractional years.

library("Epi")
library("data.table")
data("sire", package = "popEpi")
x <- Lexis(data=sire[dg_date < ex_date, ],
           entry = list(fot=0, per=get.yrs(dg_date), age=dg_age),
           exit=list(per=get.yrs(ex_date)), exit.status=status)
x2 <- splitMulti(x, breaks = list(fot=seq(0, 5, by = 3/12), per=c(2008, 2013)))
# equivalently:
x2 <- splitMulti(x, fot=seq(0, 5, by = 3/12), per=c(2008, 2013))

## using dates; note: breaks must be expressed as dates or days!
x <- Lexis(
  data = sire[dg_date < ex_date, ],
  entry = list(fot = 0L, per = dg_date, age = as.integer(dg_date - bi_date)),
  duration = as.integer(ex_date - dg_date),
  entry.status = 0L,
  exit.status = status
)
BL <- list(fot = seq(0, 5, by = 3/12)*365.242199,
           per = as.Date(paste0(c(1980:2014),"-01-01")),
           age = c(0,45,85,Inf)*365.242199)
x2 <- splitMulti(x, breaks = BL, verbose=TRUE)


## multistate example (healty - sick - dead)
sire2 <- data.frame(sire)
sire2 <- sire2[sire2$dg_date < sire2$ex_date, ]

set.seed(1L)
not_sick <- sample.int(nrow(sire2), 6000L, replace = FALSE)
sire2$dg_date[not_sick] <- NA
sire2$status[!is.na(sire2$dg_date) & sire2$status == 0] <- -1

sire2$status[sire2$status==2] <- 1
sire2$status <- factor(sire2$status, levels = c(0, -1, 1),
                       labels = c("healthy", "sick", "dead"))

xm <- Lexis(data = sire2,
            entry = list(fot=0, per=get.yrs(bi_date), age=0),
            exit = list(per=get.yrs(ex_date)), exit.status=status)
xm2 <- cutLexis(xm, cut = get.yrs(xm$dg_date),
                timescale = "per",
```

```
                new.state = "sick")
xm2[xm2$lex.id == 6L, ]

xm2 <- splitMulti(xm2, breaks = list(fot = seq(0,150,25)))
xm2[xm2$lex.id == 6L, ]
```

`stdpop101` *World standard population by 1 year age groups from 1 to 101. Sums to 100 000.*

Description

World standard population by 1 year age groups from 1 to 101. Sums to 100 000.

Format

data.table with columns

- `world_std` weight that sums to 100000 (numeric)
- `agegroup` age group from 1 to 101 (numeric)

Source

Standard population is from: world standard population "101of1"

See Also

Other popEpi data: `ICSS`, `meanpop_fi`, `popmort`, `sibr`, `sire`, `stdpop18`

Other weights: `ICSS`, `direct_standardization`, `stdpop18`

`stdpop18` *Standard populations from 2000: world, Europe and Nordic.*

Description

World, European, and Nordic standard populations by 18 age categories. Sums to 100000.

Format

data.table with columns

- `agegroup`, age group in 18 categories (character)
- `world`, World 2000 standard population (numeric)
- `europe`, European standard population (numeric)
- `nordic`, Nordic standard population (numeric)

Source

Nordcan, 2000

See Also

Other popEpi data: ICSS, meanpop_fi, popmort, sibr, sire, stdpop101

Other weights: ICSS, direct_standardization, stdpop101

---

summary.aggre *Summarize an* aggre *Object*

Description

summary method function for aggre objects; see [as.aggre] and [aggre].

Usage

```
## S3 method for class 'aggre'
summary(object, by = NULL, subset = NULL, ...)
```

Arguments

object	an aggre object
by	list of columns to summarize by - e.g. list(V1, V2) where V1 and V2 are columns in the data.
subset	a logical condition to subset results table by before summarizing; use this to limit to a certain stratum. E.g. subset = sex == "male"
...	unused

Value

Returns a data.table — a further aggregated version of object.

Author(s)

Joonas Miettinen

See Also

Other aggregation functions: aggre(), as.aggre(), lexpand(), setaggre()

summary.survtab *Summarize a survtab Object*

Description

Summary method function for survtab objects; see [survtab_ag]. Returns estimates at given time points or all time points if t and q are both NULL.

Usage

```
## S3 method for class 'survtab'
summary(object, t = NULL, subset = NULL, q = NULL, ...)
```

Arguments

object	a survtab object
t	a vector of times at which time points (actually intervals that contain t) to print summary table of survival function estimates by strata; values not existing in any interval cause rows containing only NAs to be returned.
subset	a logical condition to subset results table by before printing; use this to limit to a certain stratum. E.g. subset = sex == "male"
q	a named list of quantiles to include in returned data set, where names must match to estimates in object; returns intervals where the quantiles are reached first; e.g. list(surv.obs = 0.5) finds the interval where surv.obs is 0.45 and 0.55 at the beginning and end of the interval, respectively; returns rows with NA values for quantiles not reached in estimates (e.g. if q = list(surv.obs = 0.5) but lowest estimate is 0.6); see Examples.
...	unused; required for congruence with other summary methods

Details

Note that this function returns the intervals and NOT the time points corresponding to quantiles / estimates corresponding to time points. If you want precise estimates at time points that are not interval breaks, add the time points as breaks and re-estimate the survival time function. In interval-based estimation, the estimates denote e.g. probability of dying *during* the interval, so time points within the intervals are not usually considered at all. See e.g. Seppa, Dyba, and Hakulinen (2015).

Value

A data.table: a slice from object based on t, subset, and q.

Author(s)

Joonas Miettinen

References

Seppa K., Dyba T. and Hakulinen T.: Cancer Survival, Reference Module in Biomedical Sciences. Elsevier. 08-Jan-2015. doi:10.1016/B9780128012383.027458

See Also

Other survtab functions: `Surv()`, `lines.survtab()`, `plot.survtab()`, `print.survtab()`, `survtab()`, `survtab_ag()`

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)

library(Epi)

## NOTE: recommended to use factor status variable
x <- Lexis(entry = list(FUT = 0, AGE = dg_age, CAL = get.yrs(dg_date)),
           exit = list(CAL = get.yrs(ex_date)),
           data = sire[sire$dg_date < sire$ex_date, ],
           exit.status = factor(status, levels = 0:2,
           labels = c("alive", "canD", "othD")),
           merge = TRUE)
## pretend some are male
set.seed(1L)
x$sex <- rbinom(nrow(x), 1, 0.5)
## observed survival
st <- survtab(Surv(time = FUT, event = lex.Xst) ~ sex, data = x,
                  surv.type = "cif.obs",
                  breaks = list(FUT = seq(0, 5, 1/12)))

## estimates at full years of follow-up
summary(st, t = 1:5)

## interval estimate closest to 75th percentile, i.e.
## first interval where surv.obs < 0.75 at end
## (just switch 0.75 to 0.5 for median survival, etc.)
summary(st, q = list(surv.obs = 0.75))
## multiple quantiles
summary(st, q = list(surv.obs = c(0.75, 0.90), CIF_canD = 0.20))

## if you want all estimates in a new data.frame, you can also simply do

x <- as.data.frame(st)
```

Surv *Survival Objects*

Description

popEpi::Surv simply calls [survival::Surv]. This wrapper was written simply to avoid doing e.g. library(survival) when Surv is used in a formula evaluated by popEpi.

Usage

```
Surv(
  time,
  time2,
  event,
  type = c("right", "left", "interval", "counting", "interval2"),
  origin = 0
)
```

Arguments

time	See [survival::Surv]
time2	See [survival::Surv]
event	See [survival::Surv]
type	See [survival::Surv]
origin	See [survival::Surv]

Value

See [survival::Surv].

See Also

Other main functions: rate(), relpois(), relpois_ag(), sir(), sirspline(), survmean(), survtab(), survtab_ag()

Other survtab functions: lines.survtab(), plot.survtab(), print.survtab(), summary.survtab(), survtab(), survtab_ag()

Other survmean functions: lines.survmean(), plot.survmean(), survmean()

survmean *Compute Mean Survival Times Using Extrapolation*

Description

Computes mean survival times based on survival estimation up to a point in follow-up time (e.g. 10 years), after which survival is extrapolated using an appropriate hazard data file (pophaz) to yield the "full" survival curve. The area under the full survival curve is the mean survival.

Usage

```
survmean(
  formula,
  data,
  adjust = NULL,
  weights = NULL,
  breaks = NULL,
  pophaz = NULL,
  e1.breaks = NULL,
  e1.pophaz = pophaz,
  r = "auto",
  surv.method = "hazard",
  subset = NULL,
  verbose = FALSE
)
```

Arguments

formula	a formula, e.g. FUT ~ V1 or Surv(FUT, lex.Xst) ~ V1. Supplied in the same way as to [survtab], see that help for more info.
data	a Lexis data set; see [Epi::Lexis].
adjust	variables to adjust estimates by, e.g. adjust = "agegr". Flexible input.
weights	weights to use to adjust mean survival times. See the dedicated help page for more details on weighting. survmean computes curves separately by all variables to adjust by, computes mean survival times, and computes weighted means of the mean survival times. See Examples.
breaks	a list of breaks defining the time window to compute observed survival in, and the intervals used in estimation. E.g. list(FUT = 0:10) when FUT is the follow-up time scale in your data.
pophaz	a data set of population hazards passed to [survtab] (see the dedicated help page and the help page of survtab for more information). Defines the population hazard in the time window where observed survival is estimated.
e1.breaks	NULL or a list of breaks defining the time window to compute **expected** survival in, and the intervals used in estimation. E.g. list(FUT = 0:100) when FUT is the follow-up time scale in your data to extrapolate up to 100 years from where

	the observed survival curve ends. **NOTE:** the breaks on the survival time scale MUST include the breaks supplied to argument `breaks`; see Examples. If `NULL`, uses decent defaults (maximum follow-up time of 50 years).
`e1.pophaz`	Same as `pophaz`, except this defines the population hazard in the time window where **expected** survival is estimated. By default uses the same data as argument `pophaz`.
`r`	either a numeric multiplier such as `0.995`, `"auto"`, or `"autoX"` where `X` is an integer; used to determine the relative survival ratio (RSR) persisting after where the estimated observed survival curve ends. See Details.
`surv.method`	passed to `survtab`; see that help for more info.
`subset`	a logical condition; e.g. `subset = sex == 1`; subsets the data before computations
`verbose`	`logical`; if `TRUE`, the function is returns some messages and results along the run, which may be useful in debugging

Details

Basics

`survmean` computes mean survival times. For median survival times (i.e. where 50 % of subjects have died or met some other event) use `[survtab]`.

The mean survival time is simply the area under the survival curve. However, since full follow-up rarely happens, the observed survival curves are extrapolated using expected survival: E.g. one might compute observed survival till up to 10 years and extrapolate beyond that (till e.g. 50 years) to yield an educated guess on the full observed survival curve.

The area is computed by trapezoidal integration of the area under the curve. This function also computes the "full" expected survival curve from T = 0 till e.g. T = 50 depending on supplied arguments. The expected mean survival time is the area under the mean expected survival curve. This function returns the mean expected survival time to be compared with the mean survival time and for computing years of potential life lost (YPLL).

Results can be formed by strata and adjusted for e.g. age by using the `formula` argument as in `survtab`. See also Examples.

Extrapolation tweaks

Argument `r` controls the relative survival ratio (RSR) assumed to persist beyond the time window where observed survival is computed (defined by argument `breaks`; e.g. up to `FUT = 10`). The RSR is simply `RSR_i = p_oi / p_ei` for a time interval `i`, i.e. the observed divided by the expected (conditional, not cumulative) probability of surviving from the beginning of a time interval till its end. The cumulative product of `RSR_i` over time is the (cumulative) relative survival curve.

If `r` is numeric, e.g. `0.995`, that RSR level is assumed to persist beyond the observed survival curve. Numeric `r` should be `> 0` and expressed at the annual level when using fractional years as the scale of the time variables. E.g. if RSR is known to be `0.95` at the month level, then the annualized RSR is `0.95^12`. This enables correct usage of the RSR with survival intervals of varying lengths. When using day-level time variables (such as `Dates`; see `as.Date`), numeric `r` should be expressed at the day level, etc.

If `r` is `"auto"` or `"auto1"`, this function computes RSR estimates internally and automatically uses the `RSR_i` in the last survival interval in each stratum (and adjusting group) and assumes that to

persist beyond the observed survival curve. Automatic determination of `r` is a good starting point, but in situations where the RSR estimate is uncertain it may produce poor results. Using `"autoX"` such as `"auto6"` causes `survmean` to use the mean of the estimated RSRs in the last X survival intervals, which may be more stable. Automatic determination will not use values `>1` but set them to 1. Visual inspection of the produced curves is always recommended: see Examples.

One may also tweak the accuracy and length of extrapolation and expected survival curve computation by using `e1.breaks`. By default this is whatever was supplied to `breaks` for the survival time scale, to which

```
c(seq(1/12, 1, 1/12), seq(1.2, 1.8, 0.2), 2:19, seq(20, 50, 5))
```

is added after the maximum value, e.g. with `breaks = list(FUT = 0:10)` we have

```
..., 10+1/12, ..., 11, 11.2, ..., 2, 3, ..., 19, 20, 25, ... 50
```

as the `e1.breaks`. Supplying `e1.breaks` manually requires the breaks over time survival time scale supplied to argument `breaks` to be reiterated in `e1.breaks`; see Examples. **NOTE**: the default extrapolation breaks assume the time scales in the data to be expressed as fractional years, meaning this will work extremely poorly when using e.g. day-level time scales (such as `Date` variables). Set the extrapolation breaks manually in such cases.

Value

Returns a `data.frame` or `data.table` (depending on `getOptions("popEpi.datatable")`; see `?popEpi`) containing the following columns:

- `est`: The estimated mean survival time
- `exp`: The computed expected survival time
- `obs`: Counts of subjects in data
- `YPLL`: Years of Potential Life Lost, computed as `((exp - est) * obs)` — though your time data may be in e.g. days, this column will have the same name regardless. The returned data also has columns named according to the variables supplied to the right-hand-side of the formula.

Author(s)

Joonas Miettinen

See Also

Other survmean functions: `Surv()`, `lines.survmean()`, `plot.survmean()`

Other main functions: `Surv()`, `rate()`, `relpois()`, `relpois_ag()`, `sir()`, `sirspline()`, `survtab()`, `survtab_ag()`

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)

library(Epi)
```

```
## take 500 subjects randomly for demonstration
data(sire)
sire <- sire[sire$dg_date < sire$ex_date, ]
set.seed(1L)
sire <- sire[sample(x = nrow(sire), size = 500),]

## NOTE: recommended to use factor status variable
x <- Lexis(entry = list(FUT = 0, AGE = dg_age, CAL = get.yrs(dg_date)),
           exit = list(CAL = get.yrs(ex_date)),
           data = sire,
           exit.status = factor(status, levels = 0:2,
                                labels = c("alive", "canD", "othD")),
           merge = TRUE)

## phony variable
set.seed(1L)
x$group <- rbinom(nrow(x), 1, 0.5)
## age group
x$agegr <- cut(x$dg_age, c(0,45,60,Inf), right=FALSE)

## population hazards data  set
pm <- data.frame(popEpi::popmort)
names(pm) <- c("sex", "CAL", "AGE", "haz")

## breaks to define observed survival estimation
BL <- list(FUT = seq(0, 10, 1/12))

## crude mean survival
sm1 <- survmean(Surv(FUT, lex.Xst != "alive") ~ 1,
                pophaz = pm, data = x, weights = NULL,
                breaks = BL)

sm1 <- survmean(FUT ~ 1,
                pophaz = pm, data = x, weights = NULL,
                breaks = BL)

## mean survival by group
sm2 <- survmean(FUT ~ group,
                pophaz = pm, data = x, weights = NULL,
                breaks = BL)

## ... and adjusted for age using internal weights (counts of subjects)
## note: need also longer extrapolation here so that all curves
## converge to zero in the end.
eBL <- list(FUT = c(BL$FUT, 11:75))
sm3 <- survmean(FUT ~ group + adjust(agegr),
                pophaz = pm, data = x, weights = "internal",
                breaks = BL, e1.breaks = eBL)

## visual inspection of how realistic extrapolation is for each stratum;
## solid lines are observed + extrapolated survivals;
## dashed lines are expected survivals
plot(sm1)
```

```
## plotting object with both stratification and standardization
## plots curves for each strata-std.group combination
plot(sm3)

## for finer control of plotting these curves, you may extract
## from the survmean object using e.g.
attributes(sm3)$survmean.meta$curves


#### using Dates

x <- Lexis(entry = list(FUT = 0L, AGE = dg_date-bi_date, CAL = dg_date),
           exit = list(CAL = ex_date),
           data = sire[sire$dg_date < sire$ex_date, ],
           exit.status = factor(status, levels = 0:2,
                                labels = c("alive", "canD", "othD")),
           merge = TRUE)
## phony group variable
set.seed(1L)
x$group <- rbinom(nrow(x), 1, 0.5)


## NOTE: population hazard should be reported at the same scale
## as time variables in your Lexis data.
data(popmort, package = "popEpi")
pm <- data.frame(popmort)
names(pm) <- c("sex", "CAL", "AGE", "haz")
## from year to day level
pm$haz <- pm$haz/365.25
pm$CAL <- as.Date(paste0(pm$CAL, "-01-01"))
pm$AGE <- pm$AGE*365.25

BL <- list(FUT = seq(0, 8, 1/12)*365.25)
eBL <- list(FUT = c(BL$FUT, c(8.25,8.5,9:60)*365.25))
smd <- survmean(FUT ~ group, data = x,
                pophaz = pm, verbose = TRUE, r = "auto5",
                breaks = BL, e1.breaks = eBL)
plot(smd)
```

---

survtab *Estimate Survival Time Functions*

Description

This function estimates survival time functions: survival, relative/net survival, and crude/absolute risk functions (CIF).

Usage

```
survtab(
  formula,
  data,
  adjust = NULL,
  breaks = NULL,
  pophaz = NULL,
  weights = NULL,
  surv.type = "surv.rel",
  surv.method = "hazard",
  relsurv.method = "e2",
  subset = NULL,
  conf.level = 0.95,
  conf.type = "log-log",
  verbose = FALSE
)
```

Arguments

formula	a formula; e.g. fot ~ sex, where fot is the time scale over which you wish to estimate a survival time function; this assumes that lex.Xst in your data is the status variable in the intended format (almost always right). To be explicit, use [survival::Surv]: e.g. Surv(fot, lex.Xst) ~ sex. Variables on the right-hand side of the formula separated by + are considered stratifying variables, for which estimates are computed separately. May contain usage of adjust() — see Details and Examples.
data	a Lexis object with at least the survival time scale
adjust	can be used as an alternative to passing variables to argument formula within a call to adjust(); e.g. adjust = "agegr". Flexible input.
breaks	a named list of breaks, e.g. list(FUT = 0:5). If data is not split in advance, breaks must at the very least contain a vector of breaks to split the survival time scale (mentioned in argument formula). If data has already been split (using e.g. [splitMulti]) along at least the used survival time scale, this may be NULL. It is generally recommended (and sufficient; see Seppa, Dyban and Hakulinen (2015)) to use monthly intervals where applicable.
pophaz	a data.frame containing expected hazards for the event of interest to occur. See the dedicated help page. Required when surv.type = "surv.rel" or "cif.rel". pophaz must contain one column named "haz", and any number of other columns identifying levels of variables to do a merge with split data within survtab. Some columns may be time scales, which will allow for the expected hazard to vary by e.g. calendar time and age.
weights	typically a list of weights or a character string specifying an age group standardization scheme; see the dedicated help page and examples. NOTE: weights = "internal" is based on the counts of persons in follow-up at the start of follow-up (typically T = 0)
surv.type	one of 'surv.obs', 'surv.cause', 'surv.rel', 'cif.obs' or 'cif.rel'; defines what kind of survival time function(s) is/are estimated; see Details

surv.method	either 'lifetable' or 'hazard'; determines the method of calculating survival time functions, where the former computes ratios such as p = d/(n - n.cens) and the latter utilizes subject-times (typically person-years) for hazard estimates such as d/pyrs which are used to compute survival time function estimates. The former method requires argument n.cens and the latter argument pyrs to be supplied.
relsurv.method	either 'e2' or 'pp'; defines whether to compute relative survival using the EdererII method or using Pohar-Perme weighting; ignored if surv.type != "surv.rel"
subset	a logical condition; e.g. subset = sex == 1; subsets the data before computations
conf.level	confidence level used in confidence intervals; e.g. 0.95 for 95 percent confidence intervals
conf.type	character string; must be one of "plain", "log-log" and "log"; defines the transformation used on the survival time function to yield confidence intervals via the delta method
verbose	logical; if TRUE, the function is chatty and returns some messages and timings along the process

Value

Returns a table of life time function values and other information with survival intervals as rows. Returns some of the following estimates of survival time functions:

- surv.obs - observed (raw, overall) survival
- surv.obs.K - observed cause-specific survival for cause K
- CIF_k - cumulative incidence function for cause k
- CIF.rel - cumulative incidence function using excess cases
- r.e2 - relative survival, EdererII
- r.pp - relative survival, Pohar-Perme weighted

The suffix .as implies adjusted estimates, and .lo and .hi imply lower and upper confidence limits, respectively. The prefix SE. stands for standard error.

Basics

This function computes interval-based estimates of survival time functions, where the intervals are set by the user. For product-limit-based estimation see packages **survival** and **relsurv**.

if surv.type = 'surv.obs', only 'raw' observed survival is estimated over the chosen time intervals. With surv.type = 'surv.rel', also relative survival estimates are supplied in addition to observed survival figures.

surv.type = 'cif.obs' requests cumulative incidence functions (CIF) to be estimated. CIFs are estimated for each competing risk based on a survival-interval-specific proportional hazards assumption as described by Chiang (1968). With surv.type = 'cif.rel', a CIF is estimated with using excess cases as the "cause-specific" cases. Finally, with surv.type = 'surv.cause', cause-specific survivals are estimated separately for each separate type of event.

In hazard-based estimation (`surv.method = "hazard"`) survival time functions are transformations of the estimated corresponding hazard in the intervals. The hazard itself is estimated using counts of events (or excess events) and total subject-time in the interval. Life table `surv.method = "lifetable"` estimates are constructed as transformations of probabilities computed using counts of events and counts of subjects at risk.

The vignette survtab_examples has some practical examples.

Relative survival

When `surv.type = 'surv.rel'`, the user can choose `relsurv.method = 'pp'`, whereupon Pohar-Perme weighting is used. By default `relsurv.method = 'e2'`, i.e. the Ederer II method is used to estimate relative survival.

Adjusted estimates

Adjusted estimates in this context mean computing estimates separately by the levels of adjusting variables and returning weighted averages of the estimates. For example, computing estimates separately by age groups and returning a weighted average estimate (age-adjusted estimate).

Adjusting requires specification of both the adjusting variables and the weights for all the levels of the adjusting variables. The former can be accomplished by using `adjust()` with the argument `formula`, or by supplying variables directly to argument `adjust`. E.g. the following are all equivalent:

```
formula = fot ~ sex + adjust(agegr) + adjust(area)
```

```
formula = fot ~ sex + adjust(agegr, area)
formula  = fot ~ sex, adjust = c("agegr", "area")
formula  = fot ~ sex, adjust = list(agegr, area)
```

The adjusting variables must match with the variable names in the argument `weights`; see the dedicated help page. Typically weights are supplied as a `list` or a `data.frame`. The former can be done by e.g.

```
weights = list(agegr = VEC1, area = VEC2),
```

where `VEC1` and `VEC2` are vectors of weights (which do not have to add up to one). See survtab_examples for an example of using a `data.frame` to pass weights.

Period analysis and other data selection schemes

To calculate e.g. period analysis (delayed entry) estimates, limit the data when/before supplying to this function.See survtab_examples.

References

Perme, Maja Pohar, Janez Stare, and Jacques Esteve. "On estimation in relative survival." Biometrics 68.1 (2012): 113-120. doi:10.1111/j.15410420.2011.01640.x

Hakulinen, Timo, Karri Seppa, and Paul C. Lambert. "Choosing the relative survival method for cancer survival estimation." European Journal of Cancer 47.14 (2011): 2202-2210. doi:10.1016/j.ejca.2011.03.011

Seppa, Karri, Timo Hakulinen, and Arun Pokhrel. "Choosing the net survival method for cancer survival estimation." European Journal of Cancer (2013). doi:10.1016/j.ejca.2013.09.019

CHIANG, Chin Long. Introduction to stochastic processes in biostatistics. 1968. ISBN-14: 978-0471155003

Seppa K., Dyba T. and Hakulinen T.: Cancer Survival, Reference Module in Biomedical Sciences. Elsevier. 08-Jan-2015. doi:10.1016/B9780128012383.027458

See Also

[splitMulti], [lexpand], [ICSS], [sire] The survtab_examples vignette

Other main functions: Surv(), rate(), relpois(), relpois_ag(), sir(), sirspline(), survmean(), survtab_ag()

Other survtab functions: Surv(), lines.survtab(), plot.survtab(), print.survtab(), summary.survtab(), survtab_ag()

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)

data("sire", package = "popEpi")
library(Epi)

## NOTE: recommended to use factor status variable
x <- Lexis(entry = list(FUT = 0, AGE = dg_age, CAL = get.yrs(dg_date)),
           exit = list(CAL = get.yrs(ex_date)),
           data = sire[sire$dg_date < sire$ex_date, ],
           exit.status = factor(status, levels = 0:2,
                                labels = c("alive", "canD", "othD")),
           merge = TRUE)

## phony group variable
set.seed(1L)
x$group <- rbinom(nrow(x), 1, 0.5)

## observed survival. explicit supplying of status:
st <- survtab(Surv(time = FUT, event = lex.Xst) ~ group, data = x,
              surv.type = "surv.obs",
              breaks = list(FUT = seq(0, 5, 1/12)))
## this assumes the status is lex.Xst (right 99.9 % of the time)
st <- survtab(FUT ~ group, data = x,
              surv.type = "surv.obs",
              breaks = list(FUT = seq(0, 5, 1/12)))

## relative survival (ederer II)
data("popmort", package = "popEpi")
pm <- data.frame(popmort)
names(pm) <- c("sex", "CAL", "AGE", "haz")
st <- survtab(FUT ~ group, data = x,
              surv.type = "surv.rel",
```

```
              pophaz = pm,
              breaks = list(FUT = seq(0, 5, 1/12)))

## ICSS weights usage
data("ICSS", package = "popEpi")
cut <- c(0, 30, 50, 70, Inf)
agegr <- cut(ICSS$age, cut, right = FALSE)
w <- aggregate(ICSS1~agegr, data = ICSS, FUN = sum)
x$agegr <- cut(x$dg_age, cut, right = FALSE)
st <- survtab(FUT ~ group + adjust(agegr), data = x,
              surv.type = "surv.rel",
              pophaz = pm, weights = w$ICSS1,
              breaks = list(FUT = seq(0, 5, 1/12)))

#### using dates with survtab
x <- Lexis(entry = list(FUT = 0L, AGE = dg_date-bi_date, CAL = dg_date),
           exit = list(CAL = ex_date),
           data = sire[sire$dg_date < sire$ex_date, ],
           exit.status = factor(status, levels = 0:2,
                                labels = c("alive", "canD", "othD")),
           merge = TRUE)
## phony group variable
set.seed(1L)
x$group <- rbinom(nrow(x), 1, 0.5)

st <- survtab(Surv(time = FUT, event = lex.Xst) ~ group, data = x,
              surv.type = "surv.obs",
              breaks = list(FUT = seq(0, 5, 1/12)*365.25))

## NOTE: population hazard should be reported at the same scale
## as time variables in your Lexis data.
data(popmort, package = "popEpi")
pm <- data.frame(popmort)
names(pm) <- c("sex", "CAL", "AGE", "haz")
## from year to day level
pm$haz <- pm$haz/365.25
pm$CAL <- as.Date(paste0(pm$CAL, "-01-01"))
pm$AGE <- pm$AGE*365.25

st <- survtab(Surv(time = FUT, event = lex.Xst) ~ group, data = x,
              surv.type = "surv.rel", relsurv.method = "e2",
              pophaz = pm,
              breaks = list(FUT = seq(0, 5, 1/12)*365.25))
```

---

survtab_ag *Estimate Survival Time Functions*

Description

This function estimates survival time functions: survival, relative/net survival, and crude/absolute risk functions (CIF).

Usage

```
survtab_ag(
  formula = NULL,
  data,
  adjust = NULL,
  weights = NULL,
  surv.breaks = NULL,
  n = "at.risk",
  d = "from0to1",
  n.cens = "from0to0",
  pyrs = "pyrs",
  d.exp = "d.exp",
  n.pp = NULL,
  d.pp = "d.pp",
  d.pp.2 = "d.pp.2",
  n.cens.pp = "n.cens.pp",
  pyrs.pp = "pyrs.pp",
  d.exp.pp = "d.exp.pp",
  surv.type = "surv.rel",
  surv.method = "hazard",
  relsurv.method = "e2",
  subset = NULL,
  conf.level = 0.95,
  conf.type = "log-log",
  verbose = FALSE
)
```

Arguments

formula	a formula; the response must be the time scale to compute survival time function estimates over, e.g. fot ~ sex. Variables on the right-hand side of the formula separated by + are considered stratifying variables, for which estimates are computed separately. May contain usage of adjust() — see Details and Examples.
data	since popEpi 0.4.0, a data.frame containing variables used in formula and other arguments. aggre objects are recommended as they contain information on any time scales and are therefore safer; for creating aggre objects see [as.aggre] when your data is already aggregated and aggre for aggregating split Lexis objects.
adjust	can be used as an alternative to passing variables to argument formula within a call to adjust(); e.g. adjust = "agegr". Flexible input.
weights	typically a list of weights or a character string specifying an age group standardization scheme; see the dedicated help page and examples. NOTE: weights

= "internal" is based on the counts of persons in follow-up at the start of follow-up (typically T = 0)

surv.breaks — a vector of breaks on the survival time scale. Optional if data is an aggre object and mandatory otherwise. Must define each intended interval; e.g. surv.breaks = 0:5 when data has intervals defined by breaks seq(0, 5, 1/12) will aggregate to wider intervals first. It is generally recommended (and sufficient; see Seppa, Dyban and Hakulinen (2015)) to use monthly intervals where applicable.

n — variable containing counts of subjects at-risk at the start of a time interval; e.g. n = "at.risk". Required when surv.method = "lifetable". Flexible input.

d — variable(s) containing counts of subjects experiencing an event. With only one type of event, e.g. d = "deaths". With multiple types of events (for CIF or cause-specific survival estimation), supply e.g. d = c("canD", "othD"). If the survival time function to be estimated does not use multiple types of events, supplying more than one variable to d simply causes the variables to be added together. Always required. Flexible input.

n.cens — variable containing counts of subjects censored during a survival time interval; E.g. n.cens = "alive". Required when surv.method = "lifetable". Flexible input.

pyrs — variable containing total subject-time accumulated within a survival time interval; E.g. pyrs = "pyrs". Required when surv.method = "hazard". Flexible input.

d.exp — variable denoting total "expected numbers of events" (typically computed pyrs * pop.haz, where pop.haz is the expected hazard level) accumulated within a survival time interval; E.g. pyrs = "pyrs". Required when computing EdererII relative survivals or CIFs based on excess counts of events. Flexible input.

n.pp — variable containing total Pohar-Perme weighted counts of subjects at risk in an interval, supplied as argument n is supplied. Computed originally on the subject level as analogous to pp * as.integer(status == "at-risk"). Required when relsurv.method = "pp". Flexible input.

d.pp — variable(s) containing Pohar-Perme weighted counts of events, supplied as argument d is supplied. Computed originally on the subject level as analogous to pp * as.integer(status == some_event). Required when relsurv.method = "pp". Flexible input.

d.pp.2 — variable(s) containing total Pohar-Perme "double-weighted" counts of events, supplied as argument d is supplied. Computed originally on the subject level as analogous to pp * pp * as.integer(status == some_event). Required when relsurv.method = "pp". Flexible input.

n.cens.pp — variable containing total Pohar-Perme weighted counts censorings, supplied as argument n.cens is supplied. Computed originally on the subject level as analogous to pp * as.integer(status == "censored"). Required when relsurv.method = "pp". Flexible input.

pyrs.pp — variable containing total Pohar-Perme weighted subject-times, supplied as argument pyrs is supplied. Computed originally on the subject level as analogous to pp * pyrs. Required when relsurv.method = "pp". Flexible input.

d.exp.pp	variable containing total Pohar-Perme weighted counts of excess events, supplied as argument pyrs is supplied. Computed originally on the subject level as analogous to pp * d.exp. Required when relsurv.method = "pp". Flexible input.
surv.type	one of 'surv.obs', 'surv.cause', 'surv.rel', 'cif.obs' or 'cif.rel'; defines what kind of survival time function(s) is/are estimated; see Details
surv.method	either 'lifetable' or 'hazard'; determines the method of calculating survival time functions, where the former computes ratios such as p = d/(n - n.cens) and the latter utilizes subject-times (typically person-years) for hazard estimates such as d/pyrs which are used to compute survival time function estimates. The former method requires argument n.cens and the latter argument pyrs to be supplied.
relsurv.method	either 'e2' or 'pp'; defines whether to compute relative survival using the EdererII method or using Pohar-Perme weighting; ignored if surv.type != "surv.rel"
subset	a logical condition; e.g. subset = sex == 1; subsets the data before computations
conf.level	confidence level used in confidence intervals; e.g. 0.95 for 95 percent confidence intervals
conf.type	character string; must be one of "plain", "log-log" and "log"; defines the transformation used on the survival time function to yield confidence intervals via the delta method
verbose	logical; if TRUE, the function is chatty and returns some messages and timings along the process

Value

Returns a table of life time function values and other information with survival intervals as rows. Returns some of the following estimates of survival time functions:

- surv.obs - observed (raw, overall) survival
- surv.obs.K - observed cause-specific survival for cause K
- CIF_k - cumulative incidence function for cause k
- CIF.rel - cumulative incidence function using excess cases
- r.e2 - relative survival, EdererII
- r.pp - relative survival, Pohar-Perme weighted

The suffix .as implies adjusted estimates, and .lo and .hi imply lower and upper confidence limits, respectively. The prefix SE. stands for standard error.

Basics

This function computes interval-based estimates of survival time functions, where the intervals are set by the user. For product-limit-based estimation see packages **survival** and **relsurv**.

if `surv.type = 'surv.obs'`, only 'raw' observed survival is estimated over the chosen time intervals. With `surv.type = 'surv.rel'`, also relative survival estimates are supplied in addition to observed survival figures.

`surv.type = 'cif.obs'` requests cumulative incidence functions (CIF) to be estimated. CIFs are estimated for each competing risk based on a survival-interval-specific proportional hazards assumption as described by Chiang (1968). With `surv.type = 'cif.rel'`, a CIF is estimated with using excess cases as the "cause-specific" cases. Finally, with `surv.type = 'surv.cause'`, cause-specific survivals are estimated separately for each separate type of event.

In hazard-based estimation (`surv.method = "hazard"`) survival time functions are transformations of the estimated corresponding hazard in the intervals. The hazard itself is estimated using counts of events (or excess events) and total subject-time in the interval. Life table `surv.method = "lifetable"` estimates are constructed as transformations of probabilities computed using counts of events and counts of subjects at risk.

The vignette survtab_examples has some practical examples.

Relative survival

When `surv.type = 'surv.rel'`, the user can choose `relsurv.method = 'pp'`, whereupon Pohar-Perme weighting is used. By default `relsurv.method = 'e2'`, i.e. the Ederer II method is used to estimate relative survival.

Adjusted estimates

Adjusted estimates in this context mean computing estimates separately by the levels of adjusting variables and returning weighted averages of the estimates. For example, computing estimates separately by age groups and returning a weighted average estimate (age-adjusted estimate).

Adjusting requires specification of both the adjusting variables and the weights for all the levels of the adjusting variables. The former can be accomplished by using `adjust()` with the argument `formula`, or by supplying variables directly to argument `adjust`. E.g. the following are all equivalent:

```
formula = fot ~ sex + adjust(agegr) + adjust(area)
```

```
formula = fot ~ sex + adjust(agegr, area)
formula  = fot ~ sex, adjust = c("agegr", "area")
formula  = fot ~ sex, adjust = list(agegr, area)
```

The adjusting variables must match with the variable names in the argument `weights`; see the dedicated help page. Typically weights are supplied as a `list` or a `data.frame`. The former can be done by e.g.

```
weights = list(agegr = VEC1, area = VEC2),
```

where `VEC1` and `VEC2` are vectors of weights (which do not have to add up to one). See survtab_examples for an example of using a `data.frame` to pass weights.

Period analysis and other data selection schemes

To calculate e.g. period analysis (delayed entry) estimates, limit the data when/before supplying to this function.See survtab_examples.

Data requirements

survtab_ag computes estimates of survival time functions using pre-aggregated data. For using subject-level data directly, use [survtab]. For aggregating data, see [lexpand] and [aggre].

By default, and if data is an aggre object (not mandatory), survtab_ag makes use of the exact same breaks that were used in splitting the original data (with e.g. lexpand), so it is not necessary to specify any surv.breaks. If specified, the surv.breaks must be a subset of the pertinent pre-existing breaks. When data is not an aggre object, breaks must always be specified. Interval lengths (delta in output) are also calculated based on whichever breaks are used, so the upper limit of the breaks should therefore be meaningful and never e.g. Inf.

References

Perme, Maja Pohar, Janez Stare, and Jacques Esteve. "On estimation in relative survival." Biometrics 68.1 (2012): 113-120. doi:10.1111/j.15410420.2011.01640.x

Hakulinen, Timo, Karri Seppa, and Paul C. Lambert. "Choosing the relative survival method for cancer survival estimation." European Journal of Cancer 47.14 (2011): 2202-2210. doi:10.1016/j.ejca.2011.03.011

Seppa, Karri, Timo Hakulinen, and Arun Pokhrel. "Choosing the net survival method for cancer survival estimation." European Journal of Cancer (2013). doi:10.1016/j.ejca.2013.09.019

CHIANG, Chin Long. Introduction to stochastic processes in biostatistics. 1968. ISBN-14: 978-0471155003

Seppa K., Dyba T. and Hakulinen T.: Cancer Survival, Reference Module in Biomedical Sciences. Elsevier. 08-Jan-2015. doi:10.1016/B9780128012383.027458

See Also

[splitMulti], [lexpand], [ICSS], [sire] The survtab_examples vignette

Other main functions: Surv(), rate(), relpois(), relpois_ag(), sir(), sirspline(), survmean(), survtab()

Other survtab functions: Surv(), lines.survtab(), plot.survtab(), print.survtab(), summary.survtab(), survtab()

Examples

```
# this data.table::setDTthreads call is included here only to
# conform to the CRAN submission requirement to only use at most 2
# threads. you do not need to set this to use popEpi.
# however some long calculations may benefit from using more threads.
data.table::setDTthreads(2L)
## see more examples with explanations in vignette("survtab_examples")

#### survtab_ag usage

data("sire", package = "popEpi")
## prepare data for e.g. 5-year "period analysis" for 2008-2012
## note: sire is a simulated cohort integrated into popEpi.
BL <- list(fot=seq(0, 5, by = 1/12),
           per = c("2008-01-01", "2013-01-01"))
```

```
x <- lexpand(sire, birth = bi_date, entry = dg_date, exit = ex_date,
             status = status %in% 1:2,
             breaks = BL,
             pophaz = popmort,
             aggre = list(fot))

## calculate relative EdererII period method
## NOTE: x is an aggre object here, so surv.breaks are deduced
## automatically
st <- survtab_ag(fot ~ 1, data = x)

summary(st, t = 1:5) ## annual estimates
summary(st, q = list(r.e2 = 0.75)) ## 1st interval where r.e2 < 0.75 at end

plot(st)


## non-aggre data: first call to survtab_ag would fail
df <- data.frame(x)
# st <- survtab_ag(fot ~ 1, data = x)
st <- survtab_ag(fot ~ 1, data = x, surv.breaks = BL$fot)

## calculate age-standardised 5-year relative survival ratio using
## Ederer II method and period approach

sire$agegr <- cut(sire$dg_age,c(0,45,55,65,75,Inf),right=FALSE)
BL <- list(fot=seq(0, 5, by = 1/12),
           per = c("2008-01-01", "2013-01-01"))
x <- lexpand(sire, birth = bi_date, entry = dg_date, exit = ex_date,
             status = status %in% 1:2,
             breaks = BL,
             pophaz = popmort,
             aggre = list(agegr, fot))

## age standardisation using internal weights (age distribution of
## patients diagnosed within the period window)
## (NOTE: what is done here is equivalent to using weights = "internal")
w <- aggregate(at.risk ~ agegr, data = x[x$fot == 0], FUN = sum)
names(w) <- c("agegr", "weights")

st <- survtab_ag(fot ~ adjust(agegr), data = x, weights = w)
plot(st, y = "r.e2.as", col = c("blue"))

## age standardisation using ICSS1 weights
data(ICSS)
cut <- c(0, 45, 55, 65, 75, Inf)
agegr <- cut(ICSS$age, cut, right = FALSE)
w <- aggregate(ICSS1~agegr, data = ICSS, FUN = sum)
names(w) <- c("agegr", "weights")

st <- survtab_ag(fot ~ adjust(agegr), data = x, weights = w)
lines(st, y = "r.e2.as", col = c("red"))
```

```
## cause-specific survival
sire$stat <- factor(sire$status, 0:2, c("alive", "canD", "othD"))
x <- lexpand(sire, birth = bi_date, entry = dg_date, exit = ex_date,
             status = stat,
             breaks = BL,
             pophaz = popmort,
             aggre = list(agegr, fot))
st <- survtab_ag(fot ~ adjust(agegr), data = x, weights = w,
                 d = c("fromalivetocanD", "fromalivetoothD"),
                 surv.type = "surv.cause")
plot(st, y = "surv.obs.fromalivetocanD.as")
lines(st, y = "surv.obs.fromalivetoothD.as", col = "red")
```

`try2int` *Attempt coercion to integer*

Description

Attempts to convert a numeric object to integer, but won't if loss of information is imminent (if values after decimal are not zero for even one value in `obj`)

Usage

```
try2int(obj, tol = .Machine$double.eps^0.5)
```

Arguments

`obj`	a numeric vector
`tol`	tolerance; if each numeric value in `obj` deviate from the corresponding integers at most the value of `tol`, they are considered to be integers; e.g. by default `1 + .Machine$double.eps` is considered to be an integer but `1 + .Machine$double.eps^0.49` is not.

Value

An `integer` vector if no information is lost in coercion; else `numeric` vector.

Author(s)

James Arnold

Source

Stackoverflow thread

Index