Package 'reda'

July 8, 2020

Title Recurrent Event Data Analysis

Version 0.5.2

Date 2020-07-07

Description Contains implementations of recurrent event data analysis routines including (1) survival and recurrent event data simulation from stochastic process point of view by the thinning method proposed by Lewis and Shedler (1979) <doi:10.1002/nav.3800260304> and the inversion method introduced in Cinlar (1975, ISBN:978-0486497976), (2) the mean cumulative function (MCF) estimation by the Nelson-Aalen estimator of the cumulative hazard rate function, (3) two-sample recurrent event responses comparison with the pseudo-score tests proposed by Lawless and Nadeau (1995) <doi:10.2307/1269617>, (4) gamma frailty model with spline rate function following Fu, et al. (2016) <doi:10.1080/10543406.2014.992524>.

Imports Rcpp, ggplot2, graphics, grDevices, methods, splines2, stats

LinkingTo Rcpp, RcppArmadillo

Suggests knitr, rmarkdown, tinytest

Depends R (>= 3.2.3)

License GPL (>= 3)

LazyData true

VignetteBuilder knitr

Collate 'RcppExports.R' 'class.R' 'Recur.R' 'Survr.R' 'aic.R' 'baseline.R' 'coef.R' 'data.R' 'mcf-generic.R' 'mcf-formula.R' 'mcf-rateReg.R' 'mcfDiff.R' 'misc.R' 'plot.R' 'rateReg.R' 'reda.R' 'show.R' 'simEvent.R' 'summary.R' 'zzz.R'

URL https://github.com/wenjie2wang/reda

BugReports https://github.com/wenjie2wang/reda/issues

Encoding UTF-8

RoxygenNote 7.1.1

NeedsCompilation yes

Author Wenjie Wang [aut, cre] (<https://orcid.org/0000-0003-0363-3180>),
Haoda Fu [aut],
Jun Yan [ctb] (<https://orcid.org/0000-0003-4401-7296>)

Maintainer Wenjie Wang <wenjie.2.wang@uconn.edu>

Repository CRAN

Date/Publication 2020-07-08 06:30:02 UTC

R topics documented:

reda-package *Recurrent Event Data Analysis*

Description

The R package **reda** provides functions for simulating, exploring and modeling recurrent event data.

Details

The main functions are summarized as follows:

- `simEventData`: Simulating survival, recurrent event, and multiple event data from stochastic process point of view.
- `mcf`: Estimating the mean cumulative function (MCF) from a fitted gamma frailty model, or from a sample recurrent event data by using the nonparametic MCF estimator (the Nelson-Aelen estimator of the cumulative hazard function).
- `mcfDiff`: Comparing two-sample MCFs by the pseudo-score tests and estimating their difference over time.
- `rateReg`: Fitting Gamma fraitly model with spline baseline rate function.

See the package vignettes for more introduction and demonstration.

AIC,rateReg-method *Akaike Information Criterion (AIC)*

Description

`AIC,rateReg-method` is an S4 class method calculating Akaike information criterion (AIC) for one or several `rateReg` objects, according to the formula - 2 * log-likelihood + 2 * nPar, where nPar represents the number of parameters in the fitted model.

Usage

```
## S4 method for signature 'rateReg'
AIC(object, ..., k = 2)
```

Arguments

`object`	An object used to dispatch a method.
`...`	Optionally more fitted model objects.
`k`	An optional numeric value used as the penalty per parameter. The default `k = 2` is the classic AIC.

Details

When comparing models fitted by maximum likelihood to the same data, the smaller the AIC, the better the fit. A friendly warning will be thrown out if the numbers of observation were different in the model comparison. `help(AIC,stats)` for other details.

Value

If just one object is provided, a numeric value representing calculated AIC. If multiple objects are provided, a data frame with rows corresponding to the objects and columns `df` and `AIC`, where `df` means degree of freedom, which is the number of parameters in the fitted model.

See Also

`rateReg` for model fitting; `summary,rateReg-method` for summary of a fitted model; `BIC,rateReg-method` for BIC.

Examples

```
## See examples given in function rateReg.
```

---

as.character,Recur-method

Convert An Recur Object to A Character Vector

Description

Summarize and convert the recurrent episodes for each subjects into character strings.

Usage

```
## S4 method for signature 'Recur'
as.character(x, ...)
```

Arguments

x	An Recur object.
...	Other arguments for future usage.

Details

This function is intended to be a helper function for the ‘show()‘ method of ‘Recur‘ objects. To be precise, the function set the maximum number of recurrent episodes for each subject to be ‘max(2L, as.integer(getOption("reda.Recur.maxPrint")))‘. By default, at most three recurrent episodes will be summarized for printing. When subjects having more than three recurrent episodes, the first ‘getOption("reda.Recur.maxPrint") - 1‘ number of recurrent episodes and the last one will be summarized. One may use ‘options()‘ to adjust the setting. For example, the default value is equivalent to ‘options(reda.Recur.maxPrint = 3)‘.

baseRate *Estimated Baseline Rate Function*

Description

An S4 class generic function that returns the estimated baseline rate function.

Usage

```
baseRate(object, ...)

## S4 method for signature 'rateReg'
baseRate(object, level = 0.95, control = list(), ...)
```

Arguments

object	An object used to dispatch a method.
...	Other arguments for future usage.
level	An optional numeric value indicating the confidence level required. The default value is 0.95.
control	An optional list to specify the time grid where the baseline rate function is estimated. The availble elements of the control list include grid, length.out, from and to. The time grid can be directly specified via element grid. A dense time grid is suggested. Element length.out represents the length of grid points. The dafault value is 1,000. Element from means the starting point of grid with default 0. Element to represnts the endpoint of grid with the right boundary knot as default. When grid is missing, the grid will be generated by seq (from package **base**) with arguments from, to and length.out.

Value

A baseRate object.

Functions

- baseRate,rateReg-method: Estiamted baseline rate from a fitted model.

See Also

rateReg for model fitting; summary,rateReg-method for summary of a fitted model; plot,baseRate.rateReg-method for ploting method.

Examples

```
## See examples given in function rateReg.
```

baseRate.rateReg-class *An S4 Class Representing Estimated Baseline Rate Function*

Description

An S4 class that represents the estimated baseline rate function from model. The function `baseRate` produces objects of this class.

Slots

`baseRate` A data frame.

`level` A numeric value.

See Also

`baseRate,rateReg-method`

BIC,rateReg-method *Bayesian Information Criterion (BIC)*

Description

`BIC,rateReg-method` is an S4 class method calculating Bayesian information criterion (BIC) or so-called Schwarz's Bayesian criterion (SBC) for one or several `rateReg` objects, according to the formula - 2 * log-likelihood + ln(nObs) * nPar, where nPar represents the number of parameters in the fitted model and nObs is the number of observations.

Usage

```
## S4 method for signature 'rateReg'
BIC(object, ...)
```

Arguments

`object`	An object used to dispatch a method.
`...`	More fitted model objects.

Details

When comparing models fitted by maximum likelihood to the same data, the smaller the BIC, the better the fit. `help(BIC,stats)` for other details.

Value

If just one object is provided, a numeric value representing calculated BIC. If multiple objects are provided, a data frame with rows corresponding to the objects and columns `df` and `BIC`, where `df` means degree of freedom, which is the number of parameters in the fitted model.

See Also

`rateReg` for model fitting; `summary,rateReg-method` for summary of a fitted model; `AIC,rateReg-method` for AIC.

Examples

```
## See examples given in function rateReg.
```

`check_Recur` *Checks for Recurrent Event Data*

Description

Perform several checks for recurrent event data and update object attributions if some rows of the contained data (in the `.Data` slot) have been removed by such as `na.action`.

Usage

```
check_Recur(x, check = c("hard", "soft", "none"))
```

Arguments

`x`	An `Recur` object.
`check`	A character value specifying how to perform the checks for recurrent event data. Errors or warnings will be thrown, respectively, if the `check` is specified to be `"hard"` (by default) or `"soft"`. If `check = "none"` is specified, no data checking procedure will be run.

Value

An `Recur` object.

coef,rateReg-method *Estimated Coefficients of Covariates*

Description

coef,rateReg-method is an S4 class method that extracts estimated coefficients of covariates from rateReg object produced by function rateReg.

Usage

```
## S4 method for signature 'rateReg'
coef(object, ...)
```

Arguments

object	A rateReg object.
...	Other arguments for future usage.

Value

A named numeric vector.

See Also

rateReg for model fitting; confint,rateReg-method for confidence intervals for covariate coefficients; summary,rateReg-method for summary of a fitted model.

Examples

```
## See examples given in function rateReg.
```

confint,rateReg-method *Confidence Intervals for Covariate Coefficients*

Description

confint,rateReg-method is an S4 class method for rateReg object, which returns approximate confidence intervals for all or specified covariates.

Usage

```
## S4 method for signature 'rateReg'
confint(object, parm, level = 0.95, ...)
```

Arguments

object	A rateReg object.
parm	A specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
level	An optional numeric value to specify the confidence level required. By default, the value is 0.95, which produces 95% confidence intervals.
...	Other arguments for future usage.

Details

Under regularity condition (Shao 2003, Theorem 4.16 and Theorem 4.17, page 287, 290), the approximate confidence intervals are constructed loosely based on Fisher information matrix and estimates of coefficients.

Value

A numeric matrix with row names and column names.

References

Shao, J. (2003), *Mathematical statistics*, Springer texts in statistics, New York: Springer, 2nd Edition.

See Also

rateReg for model fitting; coef,rateReg-method for point estimates of covariate coefficients; summary,rateReg-method for summary of a fitted model.

Examples

```
## See examples given in function rateReg.
```

---

is.Recur *Is the xect from the Recur class?*

Description

Return TRUE if the specified xect is from the Recur class, FALSE otherwise.

Usage

```
is.Recur(x)
```

Arguments

x	An R xect.

Value

A logical value.

---

mcf *Mean Cumulative Function (MCF)*

Description

An S4 class generic function that returns the mean cumulative function (MCF) estimates from a fitted model or returns the nonparametric MCF estimates (by Nelson-Aalen estimator or Cook-Lawless cumulative sample mean estimator) from the sample data.

Usage

```
mcf(object, ...)

## S4 method for signature 'formula'
mcf(
  object,
  data,
  subset,
  na.action,
  variance = c("LawlessNadeau", "Poisson", "bootstrap", "CSV", "none"),
  logConfInt = FALSE,
  adjustRiskset = TRUE,
  level = 0.95,
  control = list(),
  ...
)

## S4 method for signature 'rateReg'
mcf(
  object,
  newdata,
  groupName,
  groupLevels,
  level = 0.95,
  na.action,
  control = list(),
  ...
)
```

Arguments

object	An object used to dispatch a method.
...	Other arguments for future usage.

`data` A data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from `environment(formula)`, usually the environment from which the function is called.

`subset` An optional vector specifying a subset of observations to be used in the fitting process.

`na.action` A function that indicates what should the procedure do if the data contains `NA`s. The default is set by the na.action setting of `options`. The "factory-fresh" default is `na.omit`. Other possible values inlcude `na.fail`, `na.exclude`, and `na.pass`. `help(na.fail)` for details.

`variance` A character specifying the method for variance estimates. The available options are `"LawlessNadeau"` (the default) for Lawless and Nadeau (1995) method, `"Poisson"` for Poisson process method, `"bootstrap"` for bootstrap method, `"CSV"` for variance estimates of the corresponding cumulative sample mean function (CSM) by the cumulative sample variance method (Cook and Lawless, 2007), and `"none"` for no variance estimates. Partial matching on the names is allowed.

`logConfInt` A logical value. If `FALSE` (the default), the confidence interval are constructed based on the normality of the MCF estimates. Otherwise, the confidence intervals of given level are constucted based on the normality of logarithm of the MCF estimates.

`adjustRiskset` A logical value indicating whether to adjust the size of risk-set. If `TRUE` by default, the size of risk-set will be adjusted based on at-risk indicators and Nelson-Aalen estimator will be returned. Otherwise, the cumulative sample mean (CSM) function given by Cook and Lawless (2007) will be returned without adjustment on size of risk-set.

`level` An optional numeric value indicating the confidence level required. The default value is 0.95.

`control` An optional named list specifying other options. For `rateReg` object, it can be used to specify the time grid where the MCF is estimated. The available named elements are given as follows:

- `grid`: The time grid where MCF is estimated. A dense grid is suggested for further using the plot method.
- `length.out`: The length of grid points. The dafault value is 1,000.
- `from`: The starting point of grid. The default value is the left boundary knots (for `rateReg` object).
- `to`: The endpoint of grid. The default value is the right boundary knots (for `rateReg` object).

The option `length.out`, `from`, `to` will be ignored if `grid` is specified directly. Otherwise, the grid will be generated by function `seq.int` with specified `from`, `to` and `length.out`.

For formula method, the available named elements are given as follows:

- `B`: The number of bootstrap replicates for using bootstrap method for variance estimates of sample MCF estimates. The default value is 200.
- `se.method`: The method used for SE estimates for bootstrap. The available methods include `"sample.se"` (the default) and `"normality"`. The former

takes the sample SE of point estimates from bootstrap samples; The latter estimates SE based on interquantile and normality assumption.

- `ci.method`: The method used for confidence interval (CI) for bootstrap. The available options include `"normality"` (the default) and `"percentile"`. The former estimates the CI based on SE estimates and normality assumption; The latter takes percentiles of the bootstrap estimates.
- `keep.data`: A logical value specifying whether to keep the processed data in the output object. If `TRUE` (the default), the processed data will be kept in the output and available for later usage. Otherwise, an empty data frame object will be returned in the `data` slot. `FALSE` may be set when the memory consumption is of concern and we only need MCF estimates. For example, the function `mcfDiff` and `mcfDiff.test` will not be applicable for the `mcf.formula` object with an empty `data` slot.
- `verbose`: A logical value. The default value is `TRUE`. If `FALSE`, possible data checking messages (not including warnings or errors) will be suppressed.

`newdata` An optional data frame. If specified, the data frame should have the same column names as the covariate names appearing in the formula of original fitting.

`groupName` An optional length-one charactor vector to specify the name for grouping each unique row in `newdata`, such as "gender" for "male" and "female". The default value is "Group".

`groupLevels` An optional charactor vector to specify the levels for each unique row in `newdata`, such as "treatment" and "control". The default values are `"Level"` followed by a numeric sequence with length of number of levels.

Details

For `formula` object with `Recur` object as response, the covariate specified at the right hand side of the formula should be either `1` or any "linear" conbination of categorical variable in the data. The former computes the overall sample MCF. The latter computes the sample MCF for each level of the combination of the categorical variable(s) specified, respectively.

The MCF estimates are computed on each unique time point of the sample data. By default, the size of risk set is adjusted over time based on the at-risk indicators, which results in the Nelson-Aalen nonparametric estimator (Nelson 2003). If the size of risk set remains a constant (total number of processes) over time (specified by `adjustRiskset = FALSE`), the cumulative sample mean (CSM) function introduced in Chapter 1 of Cook and Lawless (2007) will be computed instead. The point estimate of sample MCF at each time point does not assume any particular underlying model. The variance estimates at each time point is computed following the Lawless and Nadeau method (Lawless and Nadeau 1995), the Poisson process method, or the bootstrap methods. The approximate confidence intervals are provided as well, which are constructed based on the asymptotic normality of the MCF itself (by default) or the logarithm of MCF.

For `rateReg` object, `mcf` estimates the baseline MCF and its confidence interval at each time grid if argument `newdata` is not specified. Otherwise, `mcf` estimates MCF and its confidence interval for the given `newdata` based on Delta-method.

Value

A `mcf.formula` or `mcf.rateReg` object.

A brief description of the slots of a `mcf.formula` object is given as follows:

- `formula`: Model Formula.
- `data`: Processed data based on the model formula or an empty data frame if `keep.data` is set to be `FALSE`.
- `MCF`: A data frame containing estimates for sample MCF.
- `origin`: Time origins.
- `multiGroup`: A logical value indicating whether MCF is estimated for different groups respectively.
- `logConfInt`: A logical value indicating whether the variance estimates are based on the normality of logarithm of the MCF estimates.
- `level`: Confidence level specified.

Most slots of a `mcf.rateReg` object are inherited from the input `rateReg` object. A brief description of other slots is given as follows:

- `newdata`: Given dataset used to estimate MCF.
- `MCF`: A data frame containing MCF estimates.
- `level`: Confidence level specified.
- `na.action`: The way handling missing values.
- `control`: The control list.
- `multiGroup`: A logical value indicating whether MCF is estimated for different groups respectively.

Functions

- `mcf,formula-method`: Sample MCF from data.
- `mcf,rateReg-method`: Estimated MCF from a fitted model.

References

Cook, R. J., and Lawless, J. (2007). *The statistical analysis of recurrent events*, Springer Science & Business Media.

Lawless, J. F. and Nadeau, C. (1995). Some Simple Robust Methods for the Analysis of Recurrent Events. *Technometrics*, 37, 158–168.

Nelson, W. B. (2003). *Recurrent Events Data Analysis for Product Repairs, Disease Recurrences, and Other Applications* (Vol. 10). SIAM.

See Also

`rateReg` for model fitting; `mcfDiff` for comparing two-sample MCFs. `plot-method` for plotting MCF.

Examples

```
library(reda)

### sample MCF
## Example 1. valve-seat data
## the default variance estimates by Lawless and Nadeau (1995) method
valveMcf0 <- mcf(Recur(Days, ID, No.) ~ 1, data = valveSeats)
plot(valveMcf0, conf.int = TRUE, mark.time = TRUE, addOrigin = TRUE) +
    ggplot2::xlab("Days") + ggplot2::theme_bw()

## variance estimates following Poisson process model
valveMcf1 <- mcf(Recur(Days, ID, No.) ~ 1,
                 data = valveSeats, variance = "Poisson")
## variance estimates by bootstrap method (with 1,000 bootstrap samples)
set.seed(123)
valveMcf2 <- mcf(Recur(Days, ID, No.) ~ 1,
                 data = valveSeats, variance = "bootstrap",
                 control = list(B = 200))

## comparing the variance estimates from different methods
library(ggplot2)
ciDat <- rbind(cbind(valveMcf0@MCF, Method = "Lawless & Nadeau"),
               cbind(valveMcf1@MCF, Method = "Poisson"),
               cbind(valveMcf2@MCF, Method = "Bootstrap"))
ggplot(ciDat, aes(x = time, y = se)) +
    geom_step(aes(color = Method, linetype = Method)) +
    xlab("Days") + ylab("SE estimates") + theme_bw()

## comparing the confidence interval estimates from different methods
ggplot(ciDat, aes(x = time)) +
    geom_step(aes(y = MCF)) +
    geom_step(aes(y = lower, color = Method, linetype = Method)) +
    geom_step(aes(y = upper, color = Method, linetype = Method)) +
    xlab("Days") + ylab("Confidence intervals") + theme_bw()


## Example 2. the simulated data
simuMcf <- mcf(Recur(time, ID, event) ~ group + gender,
               data = simuDat, ID %in% 1 : 50)
plot(simuMcf, conf.int = TRUE, lty = 1 : 4,
     legendName = "Treatment & Gender")

### estimate MCF difference between two groups
## one sample MCF object of two groups
mcf0 <- mcf(Recur(time, ID, event) ~ group, data = simuDat)
## two-sample pseudo-score tests
mcfDiff.test(mcf0)
## difference estimates over time
mcf0_diff <- mcfDiff(mcf0, testVariance = "none")
plot(mcf0_diff)

## or explicitly ask for the difference of two sample MCF
```

```
mcf1 <- mcf(Recur(time, ID, event) ~ 1, data = simuDat,
            subset = group %in% "Contr")
mcf2 <- mcf(Recur(time, ID, event) ~ 1, data = simuDat,
            subset = group %in% "Treat")
## perform two-sample tests and estimate difference at the same time
mcf12_diff1 <- mcfDiff(mcf1, mcf2)
mcf12_diff2 <- mcf1 - mcf2   # or equivalently using the `-` method
stopifnot(all.equal(mcf12_diff1, mcf12_diff2))
mcf12_diff1
plot(mcf12_diff1)

### For estimated MCF from a fitted model,
### see examples given in function rateReg.
```

`mcf.formula-class` *An S4 Class Representing Sample MCF*

Description

An S4 class that represents sample mean cumulative function (MCF) from data. The function `mcf` produces objects of this class.

Slots

`formula` Formula.

`data` A data frame.

`MCF` A data frame.

`origin` A named numeric vector.

`multiGroup` A logical value.

`variance` A character vector.

`logConfInt` A logical value.

`level` A numeric value.

See Also

`mcf,formula-method`.

mcf.rateReg-class *An S4 Class Respresenting Estimated MCF from a Fitted Model*

Description

An S4 class that represents estimated mean cumulative function (MCF) from Models. The function `mcf` produces objects of this class.

Slots

`call` Function call.

`formula` Formula.

`spline` A character.

`knots` A numeric vector.

`degree` A nonnegative integer.

`Boundary.knots` A numeric vector.

`newdata` A numeric matrix.

`MCF` A data frame.

`level` A numeric value between 0 and 1.

`na.action` A length-one character vector.

`control` A list.

`multiGroup` A logical value.

See Also

`mcf,rateReg-method`

mcfDiff *Comparing Two-Sample MCFs*

Description

This function estimates the sample MCF difference between two groups. Both the point estimates and the confidence intervals are computed (Lawless and Nadeau 1995). The two-sample pseudo-score test proposed by Cook, Lawless, and Nadeau (1996) is also performed by default.

Usage

```
mcfDiff(mcf1, mcf2 = NULL, level = 0.95, ...)

mcfDiff.test(
  mcf1,
  mcf2 = NULL,
  testVariance = c("robust", "Poisson", "none"),
  ...
)
```

Arguments

mcf1	A mcf.formula object representing the MCF for one or two groups.
mcf2	An optional second mcf.formula object or NULL.
level	A numeric value indicating the confidence level required. The default value is 0.95.
...	Other arguments passed to mcfDiff.test.
testVariance	A character string specifying the method for computing the variance estimate for the pseudo-score test statistic proposed by Cook, Lawless, and Nadeau (1996). The applicable options include "robust" (default) for an estimate robust to departures from Poisson assumptions, "Poisson" for an estimate for Poisson process, and "none" for not performing any test (if only the difference estimates are of interest in mcfDiff).

Details

The function mcfDiff estimates the two-sample MCFs' difference and internally calls function mcfDiff.test to perform the pseudo-score tests by default. A - method is available as a simple wrapper for the function mcfDiff for comparing two-sample MCFs from two mcf.formula objects. For instance, suppose mcf1 and mcf2 are mcf.formula objects, each of which represents the sample MCF estimates for one group. The function call mcf1 -mcf2 is equivalent to mcfDiff(mcf1,mcf2).

The null hypothesis of the two-sample pseudo-score test is that there is no difference between the two sample MCFs, while the alternative hypothesis suggests a difference. The test is based on a family of test statistics proposed by Lawless and Nadeau (1995). The argument testVariance specifies the method for computing the variance estimates of the test statistics under different model assumption. See the document of argument testVariance for all applicable options. For the variance estimates robust to departures from Poisson process assumption, both constant weight and the linear weight function (with scaling) suggested in Cook, Lawless, and Nadeau (1996) are implemented. The constant weight is powerful in cases where the two MCFs are approximately proportional to each other. The linear weight function is originally a(u) = t -u, where u represents the time variable and t is the first time point when the risk set of either group becomes empty. It is further scaled by 1 / t for test statistics invariant to the unit of measurement of the time variable. The linear weight function puts more emphasis on the difference at earily times than later times and is more powerful for cases where the MCFs are no longer proportional to each other, but not crossing. Also see Cook and Lawless (2007, Section 3.7.5) for more details.

Value

The function mcfDiff returns a mcfDiff object (of S4 class) that contains the following slots:

- call: Function call.
- MCF: Estimated Mean cumulative function Difference at each time point.
- origin: Time origins of the two groups.
- variance: The method used for variance estimates.
- logConfInt: A logical value indicating whether normality is assumed for log(MCF) instead of MCF itself. For mcfDiff object, it is always FALSE.
- level: Confidence level specified.

- `test`: A `mcfDiff.test` object for the hypothesis test results.

The function `mcfDiff.test` returns a `mcfDiff.test` object (of S4 class) that contains the following slots:

- `.Data`: A numeric matrix (of two rows and five columns) for hypothesis testing results.
- `testVariance`: A character string (or vector of length one) indicating the method used for the variance estimates of the test statistic.

References

Lawless, J. F., & Nadeau, C. (1995). Some Simple Robust Methods for the Analysis of Recurrent Events. *Technometrics*, 37(2), 158–168.

Cook, R. J., Lawless, J. F., & Nadeau, C. (1996). Robust Tests for Treatment Comparisons Based on Recurrent Event Responses. *Biometrics*, 52(2), 557–571.

Cook, R. J., & Lawless, J. (2007). *The Statistical Analysis of Recurrent Events*. Springer Science & Business Media.

Examples

```
## See examples given for function mcf.
```

`mcfDiff-class` *An S4 Class Representing Sample MCF Difference*

Description

An S4 class that represents the difference between two sample mean cumulative functions from data. The function `mcfDiff` produces objects of this class.

Slots

`call` A function call.

`MCF` A data frame.

`origin` A named numeric vector.

`variance` A character vector.

`logConfInt` A logical value.

`level` A numeric value.

`test` A `mcfDiff.test` class object.

See Also

`mcfDiff`

mcfDiff.test-class *An S4 Class Representing the Two-Sample Pseudo-Score Test Results*

Description

An S4 class that represents the results of the two-sample pseudo-score tests between two sample mean cumulative functions. The function `mcfDiff.test` produces objects of this class.

Slots

`.Data` A numeric matrix.

`testVariance` A character vector.

See Also

`mcfDiff.test`

parametrize *Parametrizations of Covariates and Covariate Coefficients*

Description

This function helps the parametrizations of covariates and covariate coeffcients when users specify a general hazard rate function in function `simEvent` and `simEventData`. It applies the specified function (or the built-in option) `FUN` to the i_{th} row of the covariate matrix `z` and the i_{th} row of the coefficient matrix, iteratively, for i from one to the number of rows of the covariate matrix `z`.

Usage

```
parametrize(z, zCoef, FUN = c("exponential", "linear", "excess"), ...)
```

Arguments

`z`	A numeric matrix, each row of which represents the covariate vector at one perticular time point.
`zCoef`	A numeric matrix, each row of which represents the covariate coeffcient vector at one perticular time point.
`FUN`	The parametrization of the model parameter(s) with covariates and covariate coefficients. The built-in options include `"exponential"`, `"linear"`, `"excess"` for parametrization in the exponential, linear, excess relative risk model form, respectively. It can also be a function that at least has argument `z` and `zCoef` for incorporating the covariates and covariate coefficients into the model. The user-specified function should expect that both the input `z` and `zCoef` are numeric vectors and return a numeric value (or can be convected to a numeric value by `as.numeric`).
`...`	Other arguments that can be passed to the function `FUN`.

Value

A numeric vector.

See Also

simEvent

Examples

```
## time points
timeVec <- c(0.5, 2)
## time-variant covariates
zMat <- cbind(0.5, ifelse(timeVec > 1, 1, 0))
## time-varying coefficients
zCoefMat <- cbind(sin(timeVec), timeVec)

## the following three ways are equivalent for the exponential form,
## where the first one (using the built-in option) has the best performance
parametrize(zMat, zCoefMat, FUN = "exponential")
parametrize(zMat, zCoefMat, function(z, zCoef) exp(z %*% zCoef))
sapply(1 : 2, function(i) as.numeric(exp(zMat[i, ] %*% zCoefMat[i, ])))
```

---

plot-method *Plot Baseline Rate or Mean Cumulative Function (MCF)*

Description

S4 class methods plotting sample MCF from data, estimated MCF, or estimated baseline hazard rate function from a fitted model by using ggplot2 plotting system. The plots generated are thus able to be further customized properly.

Usage

```
## S4 method for signature 'mcf.formula,missing'
plot(
  x,
  y,
  lty,
  col,
  legendName,
  legendLevels,
  conf.int = FALSE,
  mark.time = FALSE,
  addOrigin = FALSE,
  ...
)
```

```
## S4 method for signature 'mcf.rateReg,missing'
plot(x, y, conf.int = FALSE, lty, col, ...)

## S4 method for signature 'baseRate.rateReg,missing'
plot(x, y, conf.int = FALSE, lty, col, ...)

## S4 method for signature 'mcfDiff,missing'
plot(
  x,
  y,
  lty,
  col,
  legendName,
  legendLevels,
  conf.int = TRUE,
  addOrigin = FALSE,
  ...
)
```

Arguments

x	An object used to dispatch a method.
y	An argument that should be missing and ignored now. Its existence is just for satisfying the definition of generaic function plot in package graphics for methods' dispatching.
lty	An optional numeric vector indicating line types specified to different groups: 0 = blank, 1 = solid, 2 = dashed, 3 = dotted, 4 = dotdash, 5 = longdash, 6 = twodash.
col	An optional character vector indicating line colors specified to different groups.
legendName	An optional length-one charactor vector to specify the name for grouping each unique row in newdata, such as "gender" for "male" and "female". The default value is generated from the object.
legendLevels	An optional charactor vector to specify the levels for each unique row in newdata, such as "treatment" and "control". The default values are generated from the object.
conf.int	A logical value indicating whether to plot confidence interval. The default value is FALSE.
mark.time	A logical value with default value FALSE. If TRUE, each censoring time is marked by "+" on the MCF curves. Otherwise, the censoring time would not be marked.
addOrigin	A logical value indicating whether the MCF curves start from origin time. The default value is FALSE.
...	Other arguments for further usage.

Value

A ggplot object.

See Also

mcf for estimation of MCF; rateReg for model fitting.

Examples

```
## See examples given in function mcf and rateReg.
```

---

rateReg *Recurrent Events Regression Based on Counts and Rate Function*

Description

This function fits recurrent event data (event counts) by gamma frailty model with spline rate function. The default model is the gamma frailty model with one piece constant baseline rate function, which is equivalent to negative binomial regression with the same shape and rate parameter in the gamma prior. Spline (including piecewise constant) baseline hazard rate function can be specified for the model fitting.

Usage

```
rateReg(
  formula,
  data,
  subset,
  df = NULL,
  knots = NULL,
  degree = 0L,
  na.action,
  spline = c("bSplines", "mSplines"),
  start = list(),
  control = list(),
  contrasts = NULL,
  ...
)
```

Arguments

formula	Recur object produced by function Recur. The terminal events and risk-free episodes specified in Recur will be ignored since the model does not support them.
data	An optional data frame, list or environment containing the variables in the model. If not found in data, the variables are taken from environment(formula), usually the environment from which function rateReg is called.
subset	An optional vector specifying a subset of observations to be used in the fitting process.

df	An optional nonnegative integer to specify the degree of freedom of baseline rate function. If argument knots or degree are specified, df will be neglected whether it is specified or not.
knots	An optional numeric vector that represents all the internal knots of baseline rate function. The default is NULL, representing no any internal knots.
degree	An optional nonnegative integer to specify the degree of spline bases.
na.action	A function that indicates what should the procedure do if the data contains NAs. The default is set by the na.action setting of options. The "factory-fresh" default is na.omit. Other possible values inlcude na.fail, na.exclude, and na.pass. help(na.fail) for details.
spline	An optional character that specifies the flavor of splines. The possible option is bSplines for B-splines or mSplines for M-splines.
start	An optional list of starting values for the parameters to be estimated in the model. See more in Section details.
control	An optional list of parameters to control the maximization process of negative log likelihood function and adjust the baseline rate function. See more in Section details.
contrasts	An optional list, whose entries are values (numeric matrices or character strings naming functions) to be used as replacement values for the contrasts replacement function and whose names are the names of columns of data containing factors. See contrasts.arg of model.matrix.default for details.
...	Other arguments for future usage.

Details

Function Recur in the formula response by default first checks the dataset and will report an error if the dataset does not fall into recurrent event data framework. Subject's ID will be pinpointed if its observation violates any checking rule. See Recur for all the checking rules.

Function rateReg first constructs the design matrix from the specified arguments: formula, data, subset, na.action and constrasts before model fitting. The constructed design matrix will be checked again to fit the recurrent event data framework if any observation with missing covariates is removed.

The model fitting process involves minimization of negative log likelihood function, which calls function constrOptim internally. help(constrOptim) for more details.

The argument start is an optional list that allows users to specify the initial guess for the parameter values for the minimization of negative log likelihood function. The available numeric vector elements in the list include

- beta: Coefficient(s) of covariates, set to be all 0.1 by default.
- theta: Parameter in Gamma(theta, 1 / theta) for frailty random effect, set to be 0.5 by default.
- alpha: Coefficient(s) of baseline rate function, set to be all 0.05 by default.

The argument control is an optional list that allows users to control the process of minimization of negative log likelihood function passed to constrOptim and specify the boundary knots of baseline rate function. The available options additional to those that can be passed from control to constrOptim include

- `Boundary.knots`: A length-two numeric vector to specify the boundary knots for baseline rate funtion. By default, the left boundary knot is the smallest origin time and the right one takes the largest censoring time from data.
- `verbose`: A optional logical value with default `TRUE`. Set it to be `FALSE` to supress any possible message from this function.

Value

A `rateReg` object, whose slots include

- `call`: Function call of `rateReg`.
- `formula`: Formula used in the model fitting.
- `nObs`: Number of observations.
- `spline`: A list contains
  - `spline`: The name of splines used.
  - `knots`: Internal knots specified for the baseline rate function.
  - `Boundary.knots`: Boundary knots specified for the baseline rate function.
  - `degree`: Degree of spline bases specified in baseline rate function.
  - `df`: Degree of freedom of the model specified.
- `estimates`: Estimated coefficients of covariates and baseline rate function, and estimated rate parameter of gamma frailty variable.
- `control`: The control list specified for model fitting.
- `start`: The initial guess specified for the parameters to be estimated.
- `na.action`: The procedure specified to deal with missing values in the covariate.
- `xlevels`: A list that records the levels in each factor variable.
- `contrasts`: Contrasts specified and used for each factor variable.
- `convergCode`: code returned by function `optim`, which is an integer indicating why the optimization process terminated. `help(optim)` for details.
- `logL`: Log likelihood of the fitted model.
- `fisher`: Observed Fisher information matrix.

References

Fu, H., Luo, J., & Qu, Y. (2016). Hypoglycemic events analysis via recurrent time-to-event (HEART) models. *Journal Of Biopharmaceutical Statistics*, 26(2), 280–298.

See Also

`summary,rateReg-method` for summary of fitted model; `coef,rateReg-method` for estimated covariate coefficients; `confint,rateReg-method` for confidence interval of covariate coefficients; `baseRate,rateReg-method` for estimated coefficients of baseline rate function; `mcf,rateReg-method` for estimated MCF from a fitted model; `plot,mcf.rateReg-method` for plotting estimated MCF.

Examples

```
library(reda)

## constant rate function
(constFit <- rateReg(Recur(time, ID, event) ~ group + x1, data = simuDat))

## six pieces' piecewise constant rate function
(piecesFit <- rateReg(Recur(time, ID, event) ~ group + x1,
                      data = simuDat, subset = ID %in% 1:50,
                      knots = seq.int(28, 140, by = 28)))

## fit rate function with cubic spline
(splineFit <- rateReg(Recur(time, ID, event) ~ group + x1, data = simuDat,
                      knots = c(56, 84, 112), degree = 3))

## more specific summary
summary(constFit)
summary(piecesFit)
summary(splineFit)

## model selection based on AIC or BIC
AIC(constFit, piecesFit, splineFit)
BIC(constFit, piecesFit, splineFit)

## estimated covariate coefficients
coef(piecesFit)
coef(splineFit)

## confidence intervals for covariate coefficients
confint(piecesFit)
confint(splineFit, "x1", 0.9)
confint(splineFit, 1, 0.975)

## estimated baseline rate function
splinesBase <- baseRate(splineFit)
plot(splinesBase, conf.int = TRUE)

## estimated baseline mean cumulative function (MCF) from a fitted model
piecesMcf <- mcf(piecesFit)
plot(piecesMcf, conf.int = TRUE, col = "blueviolet")

## estimated MCF for given new data
newDat <- data.frame(x1 = rep(0, 2), group = c("Treat", "Contr"))
splineMcf <- mcf(splineFit, newdata = newDat, groupName = "Group",
                 groupLevels = c("Treatment", "Control"))
plot(splineMcf, conf.int = TRUE, lty = c(1, 5))

## example of further customization by ggplot2
library(ggplot2)
plot(splineMcf) +
    geom_ribbon(aes(x = time, ymin = lower,
                    ymax = upper, fill = Group),
```

```
                data = splineMcf@MCF, alpha = 0.2) +
    xlab("Days")
```

`rateReg-class` *An S4 Class Representing a Fitted Model*

Description

The class `rateReg` is an S4 class that represents a fitted model. The function `rateReg` produces objects of this class. See "Slots" for details.

Slots

`call` Function call.

`formula` Formula.

`nObs` A positive integer

`spline` A list.

`estimates` A list.

`control` A list.

`start` A list.

`na.action` A character vector (of length one).

`xlevels` A list.

`contrasts` A list.

`convergCode` A nonnegative integer.

`logL` A numeric value.

`fisher` A numeric matrix.

See Also

`rateReg`

Recur *Formula Response for Recurrent Event Data*

Description

Create an S4 class object that represents formula response for recurrent event data with optional checking procedures embedded.

Usage

```
Recur(
  time,
  id,
  event,
  terminal,
  origin,
  check = c("hard", "soft", "none"),
  ...
)
```

Arguments

time	A numerical vector representing the time of reccurence event or censoring, or a list with elements named "time1" and "time2" for specifying the follow-up of recurrent events. In the latter case, function %to% (or %2%) can be used for ease of typing. In addition to numeric values, Date and difftime are allowed and converted to numeric values. An error will be thrown if this argument is not specified.
id	Subject identificators. It can be numeric vector, character vector, or a factor vector. If it is left unspecified, Recur will assume that each row represents a subject.
event	A numeric vector that may represent the status, costs, or types of the recurrent events. Logical vector is allowed and converted to numeric vector. Non-positive values are internally converted to zero indicating censoring status.
terminal	A numeric vector that may represent the status, costs, or types of the terminal events. Logival vector is allowed and converted to numeric vector. Non-positive values are internally converted to zero indicating censoring status. If a scalar value is specified, all subjects will have the same status of terminal events at their last recurrent episodes. The length of the specified terminal should be equal to the number of subjects, or number of data rows. In the latter case, each subject may have at most one positive entry of terminal at the last recurrent episode.
origin	The time origin of each subject. If a scalar value is specified, all subjects will have the same origin at the specified value. The length of the specified origin should be equal to the number of subjects, or number of data rows. In the latter case, different subjects may have different origins. However, one subject must

have the same origin. In addition to numeric values, `Date` and `difftime` are also supported and converted to numeric values.

`check` A character value specifying how to perform the checks for recurrent event data. Errors or warnings will be thrown, respectively, if the `check` is specified to be `"hard"` (by default) or `"soft"`. If `check = "none"` is specified, no data checking procedure will be run.

`...` Other arguments for future usage. A warning will be thrown if any invalid argument is specified.

Details

This is a successor function of the deprecated function `Survr`. See the vignette by 'vignette("reda-Recur")' for details.

Value

An `Recur` object.

Examples

```
library(reda)
with(valveSeats, Recur(Days, ID))
with(valveSeats, Recur(Days, ID, No.))
with(valveSeats, Recur(Days, ID, No., terminal = 1))
with(valveSeats, Recur(Days, ID, No., origin = 10))
```

Recur-class *An S4 Class Representing Formula Response for Recurrent Event Data*

Description

The class `Recur` is an S4 that represents a formula response for recurrent event data model. The function `Recur` produces objects of this class. See "Slots" for details.

Slots

`.Data` A numeric matrix that consists of the following columns:

- `time1`: the beginning of time segements;
- `time2`: the end of time segements;
- `id`: Identificators of subjects;
- `event`: Event indicators;
- : `terminal`: Indicators of terminal events.

`ID` A charactrer vector for original identificators of subjects.

`ord` An integer vector for increasingly ordering data by `id`, `time2`, and `-event`. Sorting is often done in the model-fitting steps, where the indices stored in this slot can be used directly.

`rev_ord` An integer vector for reverting the ordering of the sorted data (by `ord`) to its original ordering. This slot is provided to easily revert the sorting.

`first_idx` An integer vector indicating the first record of each subject in the sorted matrix. It helps in the data checking produce and may be helpful in model-fitting step, such as getting the origin time.

`last_idx` An integer vector indicating the last record of each subject in the sorted data. Similar to `first_idx`, it helps in the data checking produce and may be helpful in the model-fitting step, such as locating the terminal events.

`check` A character string indicating how the data checking is performed. It just records the option that users specified on data checking.

See Also

`Recur`

`Recur-to` *Recurrent Episodes*

Description

Specify time segements or recurrent episodes by endpoints.

Usage

```
time1 %to% time2

time1 %2% time2
```

Arguments

`time1`	The left end-points of the recurrent episodes.
`time2`	The right end-points of the recurrent episodes.

Details

This function is intended to be used for specifying the argument `time` in function `Recur`.

Value

A list that consists of two elements named `"time1"` and `"time2"`.

show-method *Show an object.*

Description

S4 class methods that display objects produced from this package (similar to S3 class `print` methods).

Usage

```
## S4 method for signature 'Recur'
show(object)

## S4 method for signature 'rateReg'
show(object)

## S4 method for signature 'summary.rateReg'
show(object)

## S4 method for signature 'mcf.formula'
show(object)

## S4 method for signature 'mcf.rateReg'
show(object)

## S4 method for signature 'simEvent'
show(object)

## S4 method for signature 'mcfDiff'
show(object)

## S4 method for signature 'mcfDiff.test'
show(object)
```

Arguments

`object`	An object used to dispatch a method.

simEvent *Simulated Survival times or Recurrent Events*

Description

The function `simEvent` generates simulated recurrent events or survival time (the first event time) from one stochastic process. The function `simEventData` provides a simple wrapper that calls `simEvent` internally and collects the generated survival data or recurrent events into a data frame. More examples are available in one of the package vignettes in addition to the function documentation.

Usage

```
simEvent(
  z = 0,
  zCoef = 1,
  rho = 1,
  rhoCoef = 1,
  origin = 0,
  endTime = 3,
  frailty = 1,
  recurrent = TRUE,
  interarrival = "rexp",
  relativeRisk = c("exponential", "linear", "excess", "none"),
  method = c("thinning", "inversion"),
  arguments = list(),
  ...
)

simEventData(nProcess = 1, z = 0, origin = 0, endTime = 3, frailty = 1, ...)
```

Arguments

`z`	Time-invariant or time-varying covariates. The default value is `0` for no covariate effect. This argument should be a numeric vector for time-invariant covariates or a function of times that returns a numeric matrix for time-varying covariates, where each row represents the covariate vector at one perticular time point.
`zCoef`	Time-invariant or time-varying coefficients of covariates. The default value is `1`. Similar to the argument `z`, this argument should be a numeric vector for time-invariant coefficients or a function of times that returns a numeric matrix for time-varying coefficients, where each row represents the coefficient vector at one perticular time point. The dimension of the `z` and `zCoef` (either specified or generated) has to match with each other.
`rho`	Baseline rate (or intensity) function for the Poisson process. The default is `1` for a homogenous process of unit intensity. This argument can be either a non-negative numeric value for a homogenous process or a function of times for a non-homogenous process. In the latter case, the function should be able to take a vector of time points and return a numerical matrix (or vector) with each row representing the baseline hazard rate vector (or scalar value) at each time point.
`rhoCoef`	Coefficients of baseline rate function. The default value is `1`. It can be useful when `rho` is a function generating spline bases.

`origin` The time origin set to be `0` by default. It should be either a numeric value less than `endTime` or a function that returns such a numeric value.

`endTime` The end of follow-up time set to be `3` by default. Similar to `origin`, `endTime` should be either a numeric value greater than `origin` or a function that returns such a numeric value.

`frailty` A positive number or a function for frailty effect. The default value is `1` for no frailty effect. Other positive value can be specified directly for a shared frailty effect within a cluster. Similar to `z`, `zCoef`, and `rho`, a function can be specified for other distribution of the frailty effect. The specified function should randomly return a positive numeric value. The functions that generate random numbers following a certain distribution from `stats` package can be directly used. The arguments of the function can be specified through a list named `frailty` in `arguments`. For example, if we consider Gamma distribution with mean one as the distribution of frailty effect, we may specify `frailty = "rgamma"`. The shape and scale parameter needs to be specified through a list named `frailty` in `arguments`, such as `arguments = list(frailty = list(shape = 2,scale = 0.5))`.

`recurrent` A logical value with default value `TRUE` indicating whether to generate recurrent event data or survival data.

`interarrival` A function object for randomly generating (positive) interarrival time between two successive arrivals/events. The default value is `"rexp"` (i.e., function `stats::rexp`) for generating interarrival times following exponential distribution, which leads to a Poisson process. If the assumption of exponential interarrival times cannot be justified, we may consider a renewal process, (a generalization of Poisson process), in which interarrival times between events independently follows an identical distribution. A customized function can be specified in this case. It must have at least one argument named `rate` for the expected number of arrivals/events in unit time and returns one positive numerical value. If the function contains an argument named `n`, it is assumed that the function returns `n` interarrival times in one function call to possibly speed up the random number generation procedure. Other arguments can be specified through a named list inside `arguments`.

`relativeRisk` Relateive risk function for incorporating the covariates and the covariate coefficients into the intensity function. The applicable choices include `exponential` (the default) for the regular Cox model or Andersen-Gill model, `linear` for linear model (including an intercept term), `excess` for excess model, and `none` for not incorporating the covariates through a relative risk function. A customized function can be specified. The specified function must have at least one argument named `z` for the covariate vector and another argument named zCoef for covariate coefficient vector. The function should return a numeric value for given `z` vector and `zCoef` vector. Other arguments can be specified through a named list inside `arguments`.

`method` A character string specifying the method for generating simulated recurrent or survival data. The default method is thinning method (Lewis and Shedler 1979). Another available option is the inversion method (Cinlar 1975). When the rate function may go to infinite, the inversion method is used and a warning will be thrown out if the thinning method is initially specified.

arguments	A list that consists of named lists for specifying other arguments in the corresponding functions. For example, if a function of time named foo with two arguments, x (for time) and y, is specified for the time-varying covariates, the value of its second argument, y, can be specified by arguments = list(z = list(y = 1). A partial matching on names is not allowed to avoid possible misspecification. The input arguments will be evaluated within function simEvent, which can be useful for randomly setting function parameters for each process in function simEventData. See examples and vignettes for details.
...	Additional arguements passed from function simEventData to fucntion simEvent. For function simEvent, ... is not used.
nProcess	Number of stochastic processes. If missing, the value will be the number of row of the specified matrix z. Otherwise, a positive number should be speicified.

Details

For each process, a time-invariant or time-varying baseline hazard rate (intensity) function of failure can be specified. Covariates and their coefficients can be specified and incorporated by the specified relative risk functions. The default is the exponential relative risk function, which corresponds to the Cox proportional hazard model (Cox 1972) for survival data or Andersen-Gill model (Andersen and Gill 1982) for recurrent events. Other relative risk function can be specified through the argument relativeRisk. In addition, a frailty effect can be considered. Conditional on predictors (or covariates) and the unobserved frailty effect, the process is by default a Poisson process, where the interarrival times between two successive arrivals/events follow exponential distribution. A general renewal process can be specified through interarrival for other distributions of the interarrival times in addition to the exponential distribution.

The thinning method (Lewis and Shedler 1979) is applied for bounded hazard rate function by default. The inversion method (Cinlar 1975) is also available for possibly unbounded but integrable rate function over the given time period. The inversion method will be used when the rate function may go to infinite and a warning will be thrown out if the thinning method is specified originally.

For the covariates z, the covariate coefficients zCoef, and the baseline hazard rate function rho, a function of time can be specified for time-varying effect. The first argument of the input function has to be the time variable (not need to be named as "time" though). Other arguments of the function can be specified through a named list in arguments, while the first argument should not be specified.

For the frailty effect frailty, the starting point origin, and the end point of the process endTime, functions that generate random numbers can be specified. An argument n = 1 will be implicitly specified if the function has an argument named n, which is designed for those common functions generating random numbers from **stats** package. Similar to z, zCoef, and rho, other arguments of the function can be specified through a named list in arguments.

For time-varying covariates, the function simEventData assumes covariates can be observed only at event times and censoring times. Thus, covariate values are returned only at these time points. If we want other observed covariate values to be recorded, we may write a simple wrapper function for simEvent similar to simEventData.

Value

The function simEvent returns a simEvent S4 class object and the function simEventData returns a data.frame.

References

Andersen, P. K., & Gill, R. D. (1982). Cox's regression model for counting processes: A large sample study. *The annals of statistics*, 10(4), 1100–1120.

Cinlar, Erhan (1975). *Introduction to stochastic processes*. Englewood Cliffs, NJ: Printice-Hall.

Cox, D. R. (1972). Regression models and life-tables. *Journal of the Royal Statistical Society. Series B (Methodological)*, 34(2), 187–220.

Lewis, P. A., & G. S. Shedler. (1979). Simulation of Nonhomogeneous Poisson Processes by Thinning. *Naval Research Logistics Quarterly*, 26(3), Wiley Online Library: 403–13.

Examples

```
library(reda)
set.seed(123)

### time-invariant covariates and coefficients
## one process
simEvent(z = c(0.5, 1), zCoef = c(1, 0))
simEvent(z = 1, zCoef = 0.5, recurrent = FALSE)

## simulated data
simEventData(z = c(0.5, 1), zCoef = c(1, 0), endTime = 2)
simEventData(z = cbind(rnorm(3), 1), zCoef = c(1, 0))
simEventData(z = matrix(rnorm(5)), zCoef = 0.5, recurrent = FALSE)


### time-varying covariates and time-varying coefficients
zFun <- function(time, intercept) {
    cbind(time / 10 + intercept, as.numeric(time > 1))
}
zCoefFun <- function(x, shift) {
    cbind(sqrt(x + shift), 1)
}
simEvent(z = zFun, zCoef = zCoefFun,
         arguments = list(z = list(intercept = 0.1),
                          zCoef = list(shift = 0.1)))

## same function of time for all processes
simEventData(3, z = zFun, zCoef = zCoefFun,
             arguments = list(z = list(intercept = 0.1),
                              zCoef = list(shift = 0.1)))

## same function within one process but different between processes
## use quote function in the arguments
simDat <- simEventData(3, z = zFun, zCoef = zCoefFun,
                       arguments = list(
                           z = list(intercept = quote(rnorm(1) / 10)),
                           zCoef = list(shift = 0.1)
                       ))
## check the intercept randomly generated,
## which should be the same within each ID but different between IDs.
unique(with(simDat, cbind(ID, intercept = round(X.1 - time / 10, 6))))
```

```
### non-negative time-varying baseline hazard rate function
simEvent(rho = function(timeVec) { sin(timeVec) + 1 })
simEventData(3, origin = rnorm(3), endTime = rnorm(3, 5),
             rho = function(timeVec) { sin(timeVec) + 1 })
## specify other arguments
simEvent(z = c(rnorm(1), rbinom(1, 1, 0.5)) / 10,
         rho = function(a, b) { sin(a + b) + 1 },
         arguments = list(rho = list(b = 0.5)))
simEventData(z = cbind(rnorm(3), rbinom(3, 1, 0.5)) / 10,
             rho = function(a, b) { sin(a + b) + 1 },
             arguments = list(rho = list(b = 0.5)))

## quadratic B-splines with one internal knot at "time = 1"
## (using function 'bSpline' from splines2 package)
simEvent(rho = splines2::bSpline, rhoCoef = c(0.8, 0.5, 1, 0.6),
         arguments = list(rho = list(degree = 2, knots = 1,
                                     intercept = TRUE,
                                     Boundary.knots = c(0, 3))))


### frailty effect
## Gamma distribution with mean one
simEvent(z = c(0.5, 1), zCoef = c(1, 0), frailty = rgamma,
         arguments = list(frailty = list(shape = 2, scale = 0.5)))

## lognormal with mean zero (on the log scale)
set.seed(123)
simEvent(z = c(0.5, 1), zCoef = c(1, 0), frailty = "rlnorm",
         arguments = list(frailty = list(sdlog = 1)))
## or equivalently
set.seed(123)
logNorm <- function(a) exp(rnorm(n = 1, mean = 0, sd = a))
simEvent(z = c(0.5, 1), zCoef = c(1, 0), frailty = logNorm,
         arguments = list(frailty = list(a = 1)))


### renewal process
## interarrival times following uniform distribution
rUnif <- function(n, rate, min) runif(n, min, max = 2 / rate)
simEvent(interarrival = rUnif,
         arguments = list(interarrival = list(min = 0)))

## interarrival times following Gamma distribution with scale one
set.seed(123)
simEvent(interarrival = function(n, rate) rgamma(n, shape = 1 / rate))
## or equivalently
set.seed(123)
simEvent(interarrival = function(rate) rgamma(n = 1, shape = 1 / rate))


### relative risk functioin
```

```
set.seed(123)
simEvent(relativeRisk = "linear")
## or equivalently
rriskFun <- function(z, zCoef, intercept) {
    as.numeric(z %*% zCoef) + intercept
}
set.seed(123)
simEvent(relativeRisk = rriskFun,
         arguments = list(relativeRisk = list(intercept = 1)))
```

`simEvent-class` *An S4 Class for Simulated Recurrent Event or Survival Times*

Description

An S4 class that represents the simulated recurrent event or survival time from one stochastic process. The function `simEvent` produces objects of this class.

Slots

`.Data` A numerical vector of possibly length zero.

`call` A function call.

`z` A list.

`zCoef` A list.

`rho` A list.

`rhoCoef` A numerical vector.

`frailty` A list.

`origin` A list.

`endTime` A list.

`censoring` A list.

`recurrent` A logical vector.

`interarrival` A list.

`relativeRisk` A list.

`method` A character vector.

See Also

`simEvent`

simuDat *Simulated Sample Dataset for Demonstration*

Description

A simulated data frame with covariates named `ID`, `time`, `event`, `group`, `x1`, and `gender`, where

- `ID`: Subjects identification;
- `time`: Event or censoring time;
- `event`: Event indicator, 1 = event, 0 = censored;
- `group`: Treatment group indicator;
- `x1`: Continuous variable.
- `gender`: Gender of subjects.

Format

A data frame with 500 rows and 6 variables.

Details

The sample dataset is originally simulated by the thinning method developed by Lewis and Shedler (1979) and further processed for a better demonstration purpose. See Fu et al. (2016) for details also.

References

Lewis, P. A., & Shedler, G. S. (1979). Simulation of nonhomogeneous Poisson processes by thinning. *Naval Research Logistics Quarterly*, 26(3), 403–413.

Fu, H., Luo, J., & Qu, Y. (2016). Hypoglycemic events analysis via recurrent time-to-event (HEART) models. *Journal Of Biopharmaceutical Statistics*, 26(2), 280–298.

summary,rateReg-method *Summarizing a Fitted Model*

Description

Summary of estimated coefficients of covariates, rate function bases, and estimated rate parameter of frailty random variable, etc.

Usage

```
## S4 method for signature 'rateReg'
summary(object, showCall = TRUE, showKnots = TRUE, ...)
```

Arguments

object	A rateReg object.
showCall	A logic value with dafault TRUE, indicating whether function show prints out the original call information of rateReg. It may be helpful for a more concise printout.
showKnots	A logic value with default TRUE, indicating whether function show prints out the internal and boundary knots. Similar to argument showCall, It may be helpful for a more concise printout.
...	Other arguments for future usage.

Details

summary,rateReg-method returns a summary.rateReg object, whose slots include

- covarCoef: Estimated covariate coefficients.
- frailtyPar: Estimated rate parameter of gamma frailty.
- baseRateCoef: Estimated coeffcients of baseline rate function.

For the meaning of other slots, see rateReg.

Value

summary.rateReg object

See Also

rateReg for model fitting; coef,rateReg-method for point estimates of covariate coefficients; confint,rateReg-method for confidence intervals of covariate coeffcients; baseRate,rateReg-method for coefficients of baseline rate function.

Examples

```
## See examples given in function rateReg.
```

---

summary.rateReg-class *An S4 Class Representing Summary of a Fitted Model*

Description

The class summary.rateReg is an S4 class with selective slots of rateReg object. See "Slots" for details. The function summary,rateReg-method produces objects of this class.

Slots

`call` Function call.

`spline` A character.

`knots` A numeric vector.

`Boundary.knots` A numeric vector.

`covarCoef` A numeric matrix.

`frailtyPar` A numeric matrix.

`degree` A nonnegative integer.

`baseRateCoef` A numeric matrix.

`logL` A numeric value.

See Also

`summary,rateReg-method`

Survr *Formula Response for Recurrent Event Data*

Description

Create an S4 class that represents formula response for recurrent event data modeled by methods based on counts and rate function. Note that the function is deprecated since version 0.5.0 and will be removed in future.

Usage

```
Survr(ID, time, event, origin = 0, check = TRUE, ...)
```

Arguments

`ID`	Subject identificators. It can be numeric vector, character vector, or a factor vector.
`time`	Time of reccurence event or censoring. In addition to numeric values, `Date` and `difftime` are supported and converted to numeric values.
`event`	A numeric vector indicating failure cost or event indicator taking positive values as costs (`1` as events), and non-positive values as censoring. Logical vector is allowed and will be converted to numeric vector.
`origin`	The time origin of each subject or process. In addition to numeric values, `Date` and `difftime` are also supported and converted to numeric values. Different subjects may have different origins. However, one subject must have the same origin.
`check`	A logical value suggesting whether to perform data checking procedure. The default value is `TRUE`. `FALSE` should be set with caution and only for processed data already in recerruent event data framework.
`...`	Other arguments for future usage.

Details

This is a similar function to `Survr` in package **survrec** but with a more considerate checking procedure embedded for recurrent event data modeled by methods based on counts and rate function. The checking rules apply to each subject respectively and include that

- Subject identification, event times, censoring time, and event indicator cannot be missing or contain missing values.
- There has to be only one censoring time not earlier than any event time.
- The time origin has to be the same and not later than any event time.

`Survr-class` *An S4 Class Representing Formula Response*

Description

The class `Survr` is an S4 that represents a formula response for recurrent event data model. The function `Survr` produces objects of this class. See "Slots" for details.

Slots

`.Data` A numeric matrix object.

`ID` A charactrer vector for original subject identificator.

`check` A logical value indicating whether to performance data checking.

`ord` An integer vector for increasingly ordering data by `ID`, `time`, and `1 -event`.

See Also

`Survr`

`valveSeats` *Valve Seats Dataset*

Description

Valve seats wear out in certain diesel engines, each with 16 valve seats. The dataset served as an example of recurrence data in Nelson (1995), which consists of valve-seat replacements on 41 engines in a fleet. The covariates are named `ID`, `Days`, and `No.`, where

- `ID`: The engine number;
- `Days`: Engine age in days;
- `No.`: Event indicator, '1' for a valve-seat replacement and, '0' for the censoring age of an engine.

Format

A data frame with 89 rows and 3 variables.

References

Nelson, W. (1995), Confidence Limits for Recurrence Data-Applied to Cost or Number of Product Repairs, *Technometrics*, 37, 147–157.

Index