Package ‘reReg’

July 22, 2020

Title Recurrent Event Regression

Version 1.3.0

Description A collection of regression models for recurrent event process and failure time data. Available methods include these from Xu et al. (2017) <doi:10.1080/01621459.2016.1173557>, Lin et al. (2000) <doi:10.1111/1467-9868.00259>, Wang et al. (2001) <doi:10.1198/016214501753209031>, Ghosh and Lin (2003) <doi:10.1111/j.0006-341X.2003.00102.x>, and Huang and Wang (2004) <doi:10.1198/016214504000001033>.

Depends R (>= 3.5.0)

License GPL (>= 3)

Encoding UTF-8

LazyData true

URL `http://github.com/stc04003/reReg`

BugReports `http://github.com/stc04003/reReg/issues`

Imports BB, nleqslv, SQUAREM, survival, ggplot2, MASS, methods, reda (>= 0.5.0), scam, Rcpp, rootSolve

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.1.1

NeedsCompilation yes

Author Sy Han (Steven) Chiou [aut, cre], Chiung-Yu Huang [aut]

Maintainer Sy Han (Steven) Chiou <schiou@utdallas.edu>

Repository CRAN

Date/Publication 2020-07-22 20:50:02 UTC

R topics documented:

`reReg-package` *reReg: Recurrent Event Regression*

Description

The package provides an easy access to fit semiparametric regression models to recurrent event data. The available implementations allow users to explore recurrent data through event plot and the cumulative sample mean function plot, simulate recurrent event data, and fit semiparametric regression models under different assumptions.

Author(s)

Maintainer: Sy Han (Steven) Chiou <`schiou@utdallas.edu`>

Authors:

- Chiung-Yu Huang <`ChiungYu.Huang@ucsf.edu`>

References

Lin, D., Wei, L., Yang, I. and Ying, Z. (2000). Semiparametric Regression for the Mean and Rate Functions of Recurrent Events. *Journal of the Royal Statistical Society: Series B (Methodological)*, **62**: 711–730.

Wang, M.-C., Qin, J., and Chiang, C.-T. (2001). Analyzing Recurrent Event Data with Informative Censoring. *Journal of the American Statistical Association*, **96**(455): 1057–1065.

Ghosh, D. and Lin, D.Y. (2002). Marginal Regression Models for Recurrent and Terminal Events. *Statistica Sinica*: 663–688.

Ghosh, D. and Lin, D.Y. (2003). Semiparametric Analysis of Recurrent Events Data in the Presence of Dependent Censoring. *Biometrics*, **59**: 877–885.

Huang, C.-Y. and Wang, M.-C. (2004). Joint Modeling and Estimation for Recurrent Event Processes and Failure Time Data. *Journal of the American Statistical Association*, **99**(468): 1153–1165.

Xu, G., Chiou, S.H., Huang, C.-Y., Wang, M.-C. and Yan, J. (2017). Joint Scale-change Models for Recurrent Events and Failure Time. *Journal of the American Statistical Association*, **112**(518): 796–805.

Xu, G., Chiou, S.H.,Yan, J., Marr, K., and Huang, C.-Y. (2019). Generalized Scale-Change Models for Recurrent Event Processes under Informative Censoring. *Statistica Sinica*: pre-print.

See Also

Useful links:

- http://github.com/stc04003/reReg
- Report bugs at http://github.com/stc04003/reReg/issues

`plot.Recur` *Produce Event Plot or Mean Cumulative Function Plot*

Description

Plot the event plot or the mean cumulative function (MCF) from an `Recur` object.

Usage

```
## S3 method for class 'Recur'
plot(
  x,
  mcf = FALSE,
  event.result = c("increasing", "decreasing", "asis"),
  mcf.adjrisk = TRUE,
  mcf.smooth = FALSE,
  control = list(),
  ...
)
```

Arguments

`x`	an object of class `Recur` returned by the `Recur()` function. See `?Recur` for creating `Recur` objects.
`mcf`	an optional logical value indicating whether the mean cumulative function (MCF) will be plotted instead of the event plot (default).
`event.result`	an optional character string that is passed to the `plotEvents()` function as the `result` argument. See `plotEvents`. This argument is used to specify whether the event plot is sorted by the subjects' terminal time. The available options are `increasing` sort the terminal time from in ascending order (default). This places longer terminal times on top. `decreasing` sort the terminal time from in descending order (default). This places shorter terminal times on top. `asis` present the as is, without sorting.
`mcf.adjrisk`	an optional logical value that is passed to the `plotMCF()` function as the `adjrisk` argument. See `plotMCF`. This argument indicates whether risk set size will be adjusted. If `mcf.adjrisk = TRUE`, subjects leave the risk set after terminal times as in the Nelson-Aalen estimator. If `mcf.adjrisk = FALSE`, subjects remain in the risk set after terminal time.

`mcf.smooth`	an optional logical value that is passed to the `plotMCF()` function as the `smooth` argument. See `plotMCF`. This argument indicates whether to add a smooth curve obtained from a monotone increasing P-splines implemented in package `scam`.
`control`	a list of control parameters. See **Details**.
`...`	additional graphical parameters to be passed to methods.

Details

The argument `control` consists of options with argument defaults to a list with the following values:

xlab customizable x-label, default value is "Time".

ylab customizable y-label, default value is "Subject" for event plot and "Cumulative mean" for MCF plot.

main customizable title, the default value is "Recurrent event plot" when `mcf = FALSE` and "Sample cumulative mean function plot" when `mcf = TRUE`.

terminal.name customizable label for terminal event, default value is "Terminal event".

recurrent.name customizable legend title for recurrent event, default value is "Recurrent events".

recurrent.types customizable label for recurrent event type, default value is `NULL`.

alpha between 0 and 1, controls the transparency of points.

The `xlab`, `ylab` and `main` parameters can also be passed down without specifying a `control` list. See **Examples**.

Value

A `ggplot` object.

References

Nelson, W. B. (1995) Confidence Limits for Recurrence Data-Applied to Cost or Number of Product Repairs. *Technometrics*, **37**(2): 147–157.

See Also

`Recur`, `plotEvents`, `plotMCF`

Examples

```
data(simDat)
reObj <- with(simDat, Recur(Time, id, event, status))

## Event plots:
plot(reObj)
plot(reObj, event.result = "decreasing")

## MCF plots
plot(reObj, mcf = TRUE)
plot(reObj, mcf = TRUE, mcf.adjrisk = FALSE)
```

```
## With (hypothetical) multiple event types
set.seed(1)
reObj2 <- with(simDat, Recur(Time, id, event * sample(1:3, nrow(simDat), TRUE), status))
plot(reObj2)
```

plot.reReg — *Plot the Baseline Cumulative Rate Function and the Baseline Cumulative Hazard Function*

Description

Plot the baseline cumulative rate function and the baseline cumulative hazard function (if applicable) for an `reReg` object.

Usage

```
## S3 method for class 'reReg'
plot(
  x,
  baseline = c("both", "rate", "hazard"),
  smooth = FALSE,
  control = list(),
  ...
)
```

Arguments

`x`	an object of class `reReg`, returned by the `reReg` function.
`baseline`	a character string specifying which baseline function to plot. `baseline = "both"` plot both the baseline cumulative rate and the baseline cumulative hazard function (if applicable) in separate panels within the same display (default). `baseline = "rate"` plot the baseline cumulative rate function. `baseline = "hazard"` plot the baseline cumulative hazard function.
`smooth`	an optional logical value indicating whether to add a smooth curve obtained from a monotone increasing P-splines implemented in package `scam`.
`control`	a list of control parameters. See **Details**.
`...`	additional graphical parameters to be passed to methods.

Details

The argument `control` consists of options with argument defaults to a list with the following values:

xlab customizable x-label, default value is "Time".

ylab customizable y-label, default value is empty.

main customizable title, default value are "Baseline cumulative rate and hazard function" when `baseline = "both"`, "Baseline cumulative rate function" when `baseline = "rate"`, and "Baseline cumulative hazard function" when `baseline = "hazard"`.

Value

A `ggplot` object.

See Also

reReg

Examples

```
data(simDat)
fm <- Recur(Time, id, event, status) ~ x1 + x2

fit <- reReg(fm, data = simDat, method = "cox", B = 0)
plot(fit)
plot(fit, baseline = "rate")
plot(fit, baseline = "rate", xlab = "Time (days)", smooth = TRUE)
```

`plotEvents` *Produce Event Plots*

Description

Plot the event plot for an `Recur` object. The usage of the function is similar to that of `plot.Recur()` but with more flexible options.

Usage

```
plotEvents(
  formula,
  data,
  result = c("increasing", "decreasing", "asis"),
  control = list(),
  ...
)
```

Arguments

`formula`	a formula object, with the response on the left of a "~" operator, and the predictors on the right. The response must be a recurrent event survival object as returned by function `Recur()`.
`data`	an optional data frame in which to interpret the variables occurring in the "`formula`".
`result`	an optional character string specifying whether the event plot is sorted by the subjects' terminal time. The available options are `increasing` sort the terminal time from in ascending order (default). This places longer terminal times on top.

`decreasing` sort the terminal time from in descending order (default). This places shorter terminal times on top.

`asis` present the as is, without sorting.

`control`	a list of control parameters.
`...`	graphical parameters to be passed to methods. These include `xlab`, `ylab`, `main`, and more. See **Details**.

Details

The argument `control` consists of options with argument defaults to a list with the following values:

xlab customizable x-label, default value is "Time".

ylab customizable y-label, default value is "Subject".

main customizable title, default value is "Recurrent event plot".

terminal.name customizable label for terminal event, default value is "Terminal event".

recurrent.name customizable legend title for recurrent event, default value is "Recurrent events".

recurrent.types customizable label for recurrent event type, default value is `NULL`.

alpha between 0 and 1, controls the transparency of points.

legend a character string specifying the position of the legend (if any). The available options are "right", "left", "top", "bottom", and "none". The default value is "right".

Value

A `ggplot` object.

See Also

`Recur`, `plot.Recur`

Examples

```
data(simDat)
plotEvents(Recur(Time, id, event, status) ~ 1, data = simDat,
           xlab = "Time in days", ylab = "Subjects arranged by terminal time")

## Separate plots by x1
plotEvents(Recur(Time, id, event, status) ~ x1, data = simDat)

## For multiple recurrent events
simDat$x3 <- ifelse(simDat$x2 < 0, "x2 < 0", "x2 > 0")
simDat$event <- simDat$event * sample(1:3, nrow(simDat), TRUE)
plotEvents(Recur(Time, id, event, status) ~ x1 + x3, data = simDat)
```

plotHaz *Plot the Baseline Cumulative Hazard Function for the Terminal Time*

Description

Plot the baseline cumulative hazard function for an reReg object. The 95% confidence interval on the baseline cumulative rate function

Usage

```
plotHaz(x, smooth = FALSE, control = list(), ...)
```

Arguments

x	an object of class reReg, returned by the reReg function.
smooth	an optional logical value indicating whether to add a smooth curve obtained from a monotone increasing P-splines implemented in package scam.
control	a list of control parameters.
...	graphical parameters to be passed to methods. These include xlab, ylab, main, and more. See **Details**.

Details

The argument control consists of options with argument defaults to a list with the following values:

xlab customizable x-label, default value is "Time".

ylab customizable y-label, default value is empty.

main customizable title, default value is "Baseline cumulative hazard function".

These arguments can also be passed down without specifying a control list. See **Examples**.

Value

A ggplot object.

See Also

reReg plot.reReg

Examples

```
data(simDat)
fm <- Recur(Time, id, event, status) ~ x1 + x2

fit <- reReg(fm, data = simDat, method = "cox", B = 0)
## Plot both the baseline cumulative rate and hazard function
plot(fit)
```

```
## Plot baseline cumulative hazard function
plotHaz(fit)
plotHaz(fit, smooth = TRUE)
```

---

plotMCF *Produce Cumulative Sample Mean Function Plots*

Description

Plot the mean cumulative function (MCF) for an `Recur` object. The usage of the function is similar to that of `plot.Recur()` but with more flexible options.

Usage

```
plotMCF(
  formula,
  data,
  onePanel = FALSE,
  adjrisk = TRUE,
  smooth = FALSE,
  control = list(),
  ...
)
```

Arguments

`formula`	a formula object, with the response on the left of a "~" operator, and the predictors on the right. The response must be a recurrent event survival object returned by the `Recur()` function.
`data`	an optional data frame in which to interpret the variables occurring in the "`formula`".
`onePanel`	an optional logical value indicating whether the mean cumulative functions will be plotted in the same panel. This is only useful when there are multiple recurrent event types or in the presence of (discrete) covariates.
`adjrisk`	an optional logical value that is passed to the `plotMCF()` function as the `adjrisk` argument. See `plotMCF`. This argument indicates whether risk set size will be adjusted. If `mcf.adjrisk = TRUE`, subjects leave the risk set after terminal times as in the Nelson-Aalen estimator. If `mcf.adjrisk = FALSE`, subjects remain in the risk set after terminal time.
`smooth`	an optional logical value indicating whether to add a smooth curve obtained from a monotone increasing P-splines implemented in package `scam`. This feature only works for data with one recurrent event type.
`control`	a list of control parameters.
`...`	graphical parameters to be passed to methods. These include `xlab`, `ylab`, `main`, and more. See **Details**.

Details

When `adjrisk = TRUE`, the `plotMCF()` is equivalent to the Nelson-Aalen estimator for the intensity function of the recurrent event process. When `adjrisk = FALSE`, the `plotMCF()` does not adjust for the risk set and assumes all subjects remain at risk after the last observed recurrent event. This is known as the survivor rate function. The argument `control` consists of options with argument defaults to a list with the following values:

xlab customizable x-label, default value is "Time".

ylab customizable y-label, default value is "Cumulative mean".

main customizable title, default value is "Sample cumulative mean function plot".

The `xlab`, `ylab` and `main` parameters can also be passed down without specifying a `control` list.

Value

A `ggplot` object.

See Also

`Recur`, `plot.Recur`

Examples

```
data(simDat)
plotMCF(Recur(Time, id, event, status) ~ 1, data = simDat)
plotMCF(Recur(Time, id, event, status) ~ x1, data = simDat, onePanel = TRUE)
```

`plotRate` *Plotting the Baseline Cumulative Rate Function for the Recurrent Event Process*

Description

Plot the baseline rate function for an `reReg` object.

Usage

```
plotRate(
  x,
  type = c("unrestricted", "scaled", "raw"),
  smooth = FALSE,
  control = list(),
  ...
)
```

Arguments

`x`	an object of class `reReg`, usually returned by the `reReg` function.
`type`	a character string specifying the type of rate function to be plotted. Options are "unrestricted", "scaled", "raw". See **Details**.
`smooth`	an optional logical value indicating whether to add a smooth curve obtained from a monotone increasing P-splines implemented in package `scam`.
`control`	a list of control parameters.
`...`	graphical parameters to be passed to methods. These include `xlab`, `ylab`, `main`, and more. See **Details**.

Details

The `plotRate()` plots the estimated baseline cumulative rate function depending on the identifiability assumption. When `type = "unrestricted"` (default), the baseline cumulative rate function is plotted under the assumption $E(Z) = 1$. When `type = "scaled"`, the baseline cumulative rate function is plotted under the assumption $\Lambda(\min(Y^*, \tau)) = 1$. When `type = "raw"`, the baseline cumulative rate function is plotted under the assumption $\Lambda(\tau) = 1$. See `?reReg` for the specification of the notations and underlying models.

The argument `control` consists of options with argument defaults to a list with the following values:

xlab customizable x-label, default value is "Time".

ylab customizable y-label, default value is empty.

main customizable title, default value is "Baseline cumulative rate function".

These arguments can also be passed down without specifying a `control` list. See **Examples**.

Value

A `ggplot` object.

See Also

`reReg` `plot.reReg`

Examples

```
data(simDat)
fm <- Recur(Time, id, event, status) ~ x1 + x2

fit <- reReg(fm, data = simDat, method = "cox", B = 0)
## Plot both the baseline cumulative rate and hazard function
plot(fit)
## Plot baseline cumulative rate function
plotRate(fit)
plotRate(fit, smooth = TRUE)
```

Recur *The* Recur *function is imported from* reda.

Description

Create a recurrent event survival object, used as a response variable in reReg. This function is replacing the original reSurv() in version 1.1.6. See ?reda::Recur for more details.

See Also

%2%

Examples

```
Recur(2:6, id = c(1, 1, 1, 2, 2))
Recur(2:6, id = c(1, 1, 1, 2, 2))
Recur(1:5 %2% 2:6, id = c(1, 1, 1, 2, 2))
```

Recur-pipe *The* %to% *function is imported from* reda

Description

This pipe operator specifies the time segments or recurrent episodes by endpoints. See reda for more details.

Examples

```
Recur(2:6, id = c(1, 1, 1, 2, 2))
Recur(2:6, id = c(1, 1, 1, 2, 2))
Recur(1:5 %2% 2:6, id = c(1, 1, 1, 2, 2))
```

reReg *Fits Semiparametric Regression Models for Recurrent Event Data*

Description

Fits a general (joint) semiparametric regression model for the recurrent event data, where the rate function of the underlying recurrent event process and the hazard function of the terminal event can be specified as a Cox-type model, an accelerated mean model, an accelerated rate model, or a generalized scale-change model. See details for model specifications.

Usage

```
reReg(
  formula,
  data,
  B = 200,
  method = "cox",
  se = c("resampling", "bootstrap", "NULL"),
  control = list()
)
```

Arguments

`formula`	a formula object, with the response on the left of a "~" operator, and the predictors on the right. The response must be a recurrent event survival object as returned by function `Recur`.
`data`	an optional data frame in which to interpret the variables occurring in the `"formula"`.
`B`	a numeric value specifies the number of resampling for variance estimation. When `B = 0`, variance estimation will not be performed.
`method`	a character string specifying the underlying model. See **Details**.
`se`	a character string specifying the method for standard error estimation. See **Details**.
`control`	a list of control parameters.

Details

Suppose the recurrent event process and the failure events are observed in the time interval $t \in [0, \tau]$, for some constant τ. We formulate the recurrent event rate function, $\lambda(t)$, and the terminal event hazard function, $h(t)$, in the form of

$$\lambda(t) = Z\lambda_0(te^{X^\top\alpha})e^{X^\top\beta}, h(t) = Zh_0(te^{X^\top\eta})e^{X^\top\theta},$$

where $\lambda_0(t)$ is the baseline rate function, $h_0(t)$ is the baseline hazard function, X is a n by p covariate matrix and α, Z is an unobserved shared frailty variable, and (α, η) and (β, θ) correspond to the shape and size parameters of the rate function and the hazard function, respectively. The model includes several popular semiparametric models as special cases, which can be specified via the `method` argument with the rate function and hazard function separated by "`|`". For examples, Wang, Qin and Chiang (2001) ($\alpha = \eta = \theta = 0$) can be called with `method = "cox|."`; Huang and Wang (2004) ($\alpha = \eta = 0$) can be called with `method = "cox|cox"`; Xu et al. (2017) ($\alpha = \beta$ and $\eta = \theta$) can be called with `method = "am|am"`; Xu et al. (2019) ($\eta = \theta = 0$) can be called with `method = "sc|."`. Users can mix the models depending on the application. For example, `method = "cox|ar"` postulate a Cox proportional model for the recurrent event rate function and an accelerated rate model for the terminal event hazard function ($\alpha = \theta = 0$). If only one method is specified without an "`|`", it is used for both the rate function and the hazard function. For example, specifying `method = "cox"` is equivalent to `method = "cox|cox"`. Some methods that assumes `Z = 1` and requires independent censoring are also implemented in `reReg`; these includes `method = "cox.LWYY"` for Lin et al. (2000), `method = "cox.GL"` for Ghosh and Lin (2002), and `method = "am.GL"` for Ghosh and Lin (2003).

The available methods for variance estimation are:

NULL variance estimation will not be performed. This is equivalent to setting `B = 0`.

resampling performs the efficient resampling-based variance estimation.

bootstrap performs nonparametric bootstrap.

The `control` list consists of the following parameters:

tol absolute error tolerance.

a0, b0 initial guesses used for root search.

solver the equation solver used for root search. The available options are `BB::BBsolve`, `BB::dfsane`, `BB:BBoptim`, and `optim`.

baseSE an logical value indicating whether the 95% confidence bounds for the baseline functions will be computed.

parallel an logical value indicating whether parallel computation will be applied when `se = "bootstrap"` is called.

parCl an integer value specifying the number of CPU cores to be used when `parallel = TRUE`. The default value is half the CPU cores on the current host.

References

Lin, D., Wei, L., Yang, I. and Ying, Z. (2000). Semiparametric Regression for the Mean and Rate Functions of Recurrent Events. *Journal of the Royal Statistical Society: Series B (Methodological)*, **62**: 711–730.

Wang, M.-C., Qin, J., and Chiang, C.-T. (2001). Analyzing Recurrent Event Data with Informative Censoring. *Journal of the American Statistical Association*, **96**(455): 1057–1065.

Ghosh, D. and Lin, D.Y. (2002). Marginal Regression Models for Recurrent and Terminal Events. *Statistica Sinica*: 663–688.

Ghosh, D. and Lin, D.Y. (2003). Semiparametric Analysis of Recurrent Events Data in the Presence of Dependent Censoring. *Biometrics*, **59**: 877–885.

Huang, C.-Y. and Wang, M.-C. (2004). Joint Modeling and Estimation for Recurrent Event Processes and Failure Time Data. *Journal of the American Statistical Association*, **99**(468): 1153–1165.

Xu, G., Chiou, S.H., Huang, C.-Y., Wang, M.-C. and Yan, J. (2017). Joint Scale-change Models for Recurrent Events and Failure Time. *Journal of the American Statistical Association*, **112**(518): 796–805.

Xu, G., Chiou, S.H.,Yan, J., Marr, K., and Huang, C.-Y. (2019). Generalized Scale-Change Models for Recurrent Event Processes under Informative Censoring. *Statistica Sinica*: pre-print.

See Also

`Recur`, `simSC`

Examples

```
fm <- Recur(Time, id, event, status) ~ x1 + x2

## Joint Cox/Accelerated Mean Model
set.seed(1)
dat <- simSC(50, c(-1, 1), c(-1, 1), type = "cox|am")
(fit <- reReg(Recur(Time, id, event, status) ~ x1 + x2,
              data = dat, method = "cox|am", B = 50))
summary(fit)

## Generalized Scale-Change Model
set.seed(1)
dat <- simSC(50, c(-1, 1), c(-1, 1), type = "sc")
(fit <- reReg(Recur(Time, id, event, status) ~ x1 + x2,
              data = dat, method = "sc|.", B = 50))
summary(fit)
```

`reSurv` *Create an* `reSurv` *Object*

Description

Create a recurrent event survival object, used as a response variable in `reReg`. This function is deprecated in Version 1.1.6. A recurrent event object is now being created with `Recur()`. See '?Recur()' for details.

Usage

```
reSurv(time1, time2, id, event, status, origin = 0)
```

Arguments

`time1`	when "`time2`" is provided, this vector is treated as the starting time for the gap time between two successive recurrent events. In the absence of "`time2`", this is the observation time of recurrence on calendar time scale, in which, the time corresponds to the time since entry/inclusion in the study.
`time2`	an optional vector for ending time for the gap time between two successive recurrent events.
`id`	subject's id.
`event`	a binary vector used as the recurrent event indicator. `event = 1` for recurrent times.
`status`	a binary vector used as the status indicator for the terminal event. `status = 0` for censored times.
`origin`	a numerical vector indicating the time origin of subjects. When `origin` is a scalar, `reSurv` assumes all subjects have the same origin. Otherwise, `origin` needs to be a numerical vector, with length equals to the number of subjects. In this case, each element corresponds to different origins for different subjects. This argument is only needed when "`time2`" is missing.

Examples

```
## Not run:
  data(simDat)
  ## being deprecated in Verson 1.1.7
  with(dat, reSurv(Time, id, event, status))
  ## Use Recur() instead
  with(dat, Recur(Time, id, event, status))

## End(Not run)
```

---

simDat — *Simulated dataset for demonstration*

Description

A simulated data frame with variables

id subjects identification

Time observation time

status terminal event indicator; 1 if a terminal event is recorded

event recurrent event indicator; 1 if a recurrent event is recorded

x1 baseline covariate generated from a standard uniform distribution

x2 baseline covariate generated from a standard uniform distribution (independent from `z1`

Usage

```
data(simDat)
```

Format

A data frame with 274 rows and 6 variables.

Details

The sample dataset is generated by `set.seed(1); simSC(30,c(-1,1),c(-1,1))`. See `simSC` for instruction on simulating recurrent event data from scale-change models.

`simSC` *Function to generate simulated recurrent event data*

Description

The function `simSC()` generates simulated recurrent event data from either a Cox-type model, an accelerated mean model, an accelerated rate model, or a scale-change model.

Usage

```
simSC(
  n,
  a1 = a2,
  b1 = b2,
  a2 = a1,
  b2 = b1,
  type = "cox",
  zVar = 0.25,
  tau = 60,
  summary = FALSE
)
```

Arguments

`n`	number of observation.
`a1, a2, b1, b2`	are numeric vectors of length 2. These correspond to the α, β, η, and θ in the joint model. See **Details**
`type`	is a character string specifying the underlying model. The rate function type and the hazard function type are separated by a vertical bar "|", with the rate function on the left. For example, `type = "cox|am"` generates the recurrent process from a Cox model and the terminal event from an accelerated mean model. Setting `type = "cox"` gives `type = "cox|cox"`.
`zVar`	a numeric variable specifying the variance of the fraility variable,Z, when `zVar` > 0. When `zVar` = 0, Z is set to a fixed constant 1. The default value is 0.25.
`tau`	a numeric value specifying the maximum observation time.
`summary`	a logical value indicating whether a brief data summary will be printed.

Details

The function `simSC()` generates simulated recurrent event data over the interval $(0, \tau)$ based on the specification of the recurrent process and the terminal events. Specifically, the rate function, $\lambda(t)$, of the recurrent process can be specified as one of the following model:

$$\lambda(t) = Z\lambda_0(te^{X^\top\alpha})e^{X^\top\beta}, h(t) = Zh_0(te^{X^\top\eta})e^{X^\top\theta},$$

where $\lambda_0(t)$ is the baseline rate function, $h_0(t)$ is the baseline hazard function, X is a n by p covariate matrix and α, Z is an unobserved shared frailty variable, and (α, η) and (β, θ) correspond to the shape and size parameters of the rate function and the hazard function, respectively.

For all scenarios, two covariates are considered; $X = (X_1, X_2)$, where X_1 follows a Bernoulli distribution with probability 0.5 and X_2 follows a standard normal distribution. The censoring time could be either independent (given covariates) or informative. The simulated data is used for illustration. An informative censoring time, C, is generated separately from an exponential distribution with a rate parameter of 1 / 60 if X_1 is 1, or $Z^2/30$ if X_1 is 0. The observed recurrent events is then observed up to the minimum of C, terminal event, and τ. Lastly, we assume the baseline functions

$$\lambda_0(t) = \frac{2}{1+t}, h_0(t) = \frac{1}{8(1+t)}.$$

See Also

reReg

Examples

```
set.seed(123)
simSC(100, c(-1, 1), c(-1, 1), summary = TRUE)
```

Index