Package ‘coxinterval’

May 2, 2015

Title Cox-Type Models for Interval-Censored Data

Version 1.2

Date 2015-04-30

Author Audrey Boruvka and Richard J. Cook

Maintainer Audrey Boruvka <audrey.boruvka@gmail.com>

Description Fits Cox-type models based on interval-censored data from a survival or illness-death process.

SystemRequirements GNU make

Depends R (>= 2.15.2), survival, timereg

Imports Matrix

LazyData yes

LazyLoad yes

NeedsCompilation yes

License GPL (>= 2)

URL https://github.com/aboruvka/coxinterval

BugReports https://github.com/aboruvka/coxinterval/issues

Repository CRAN

Date/Publication 2015-05-02 14:14:36

R topics documented:

`const2lin` *Accumulate values from a piecewise constant function of time*

Description

A utility function for `coxdual` that integrates values from a piecewise constant function of time.

Usage

```
const2lin(const, time = "time", stratum = NULL)
```

Arguments

`const`	a matrix whose columns give the (possibly multivariate) piecewise constant function values, corresponding time points and strata.
`time`	an integer or character value indicating the column index or name of the time variable in the matrix `const`. Defaults to `"time"`.
`stratum`	integer or character value for the column index or name of the stratifying variable in the matrix `const`. If `stratum` is `NULL` (default), `const` is presumed unstratified.

Value

A matrix of the same dimension as `const`.

See Also

`lin2const`

Examples

```
fit <- coxdual(Surv(start, stop, status) ~ cluster(id)
               + trans(from, to) + z, data = dualrc, init.coxph = TRUE)
haz <- lin2const(fit$basehaz, stratum = 3)
Haz <- const2lin(haz, stratum = 3)
all(Haz == fit$basehaz)
```

`cosmesis` *Breast cosmesis data*

Description

Interval-censored times to cosmetic deterioration for breast cancer patients undergoing radiation or radiation plus chemotherapy.

Usage

```
data(cosmesis)
```

Format

A data frame with 94 observations on the following 3 variables.

`left` left endpoint of the censoring interval in months

`right` right endpoint of the censoring interval in months

`treat` a factor with levels `RT` and `RCT` representing radiotherapy-only and radiation plus chemotherapy treatments, respectively

Source

Finkelstein, D. M. and Wolfe, R. A. (1985) A semiparametric model for regression analysis of interval-censored failure time data. *Biometrics* **41**, 933–945.

References

Finkelstein, D. M. (1986) A proportional hazards model for interval-censored failure time data. *Biometrics* **42**, 845–854.

Examples

```
data(cosmesis)
```

`coxaalen` *Cox-Aalen model for interval-censored survival data*

Description

Fit a Cox-Aalen model to interval-censored survival data with fixed covariates.

Usage

```
coxaalen(formula, data = parent.frame(), subset, init = NULL,
         formula.timereg = NULL, init.timereg = FALSE, control, ...)
```

Arguments

`formula`	an expression of the form `response ~ terms`, where `response` is an object returned by the `Surv` function and `terms` contains at least one multiplicative term identified by the `prop` function from the **timereg** package.
`data`	an optional data frame in which to interpret the variables named in the arguments `formula` and `formula.timereg`.
`subset`	expression specifying which rows of `data` should be used in the fit. All observations are included by default.
`init`	a list with elements named `coef` and `basehaz`. The `coef` element should be a scalar or vector specifying the initial values of the multiplicative regression coefficient. If `init = NULL` or `coef = NULL`, this coefficient will be initialized to zero. The element `basehaz` should be a matrix or data frame whose columns represent time and the corresponding value for the cumulative baseline hazard function and any remaining additive cumulative coefficients. Initial values are obtained by linear interpolation or extrapolation at observation times relevant in the data. If `init = NULL` or `basehaz = NULL`, the cumulative baseline hazard function is initialized to a linear function of time and any cumulative coefficients are started at zero.
`formula.timereg`	an optional formula object specifying a model to fit with the **timereg** package's `cox.aalen` function using right-censored observations. Here the shorthand `~ .` refers to the same `terms` given in `formula`. Multiple formula objects can be provided as a list.
`init.timereg`	a logical value indicating that `init` should be overrided by estimates based on the `cox.aalen` fit to the first model in `formula.timereg`.
`control`	a named list of parameters controlling the model fit, as returned by the function `coxaalen.control`. This defaults to `coxaalen.control()`.
`...`	additional arguments to be passed to `coxaalen.control`.

Details

A valid response in the `formula` argument can be expressed as

```
Surv(<left>, <right>, type = "interval2")
```

where (`<left>`, `<right>`] is the censoring interval for the survival time. Following the **survival** package's `type = "interval2"` censoring for the `Surv` function, we use the convention that any right-censoring times are provided in the variable `<left>` and `<right>` is set to the `NA` value.

Terms in `formula` have either time-varying additive effects on the survival hazard as in Aalen's additive regression model, or fixed multiplicative effects as in the Cox model. Multiplicative terms are distinguished by applying **timereg**'s `prop` function to each corresponding variable.

`coxaalen` depends on libraries that are loaded only if `coxinterval` is installed from source on a system with IBM ILOG CPLEX Optimization Studio. Refer to the package's INSTALL file for detailed instructions.

Value

An object of the class "coxinterval" and "coxaalen", which is a list with the following components.

`call`	the matched call to `coxaalen`.
`n`	size of the sample used in the model fit.
`p`	number of (multiplicative) regression coefficients.
`coef`	a named `p` vector of regression coefficients.
`var`	a named `p` by `p` covariance matrix of the regression coefficients.
`basehaz`	a data frame giving the cumulative regression functions evaluated at time points given by the right endpoints of the maximal intersections among the censoring intervals.
`init`	list of initial values used in the model fit.
`loglik`	a vector giving the log-likelihood at initiation and each iteration.
`iter`	number of iterations needed to meet the stopping criteria.
`maxnorm`	the maximum norm of the difference between the penultimate and final parameter values.
`gradnorm`	the inner product between the final parameter value and the score function.
`cputime`	the processing time for parameter and variance estimation.
`fit.timereg`	the `cox.aalen` fit to any models specified by the `formula.timereg` argument. If `formula.timereg` is a list of formula objects, `fit.timereg` is an unnamed list following the same order.
`na.action`	the "`na.action`" attribute of the model frame.
`censor.rate`	a vector giving the rates of exact, left-censored, interval-censored and right-censored observations used in the model fit.
`control`	a named list of arguments passed to `coxaalen.control`.
`data`	a list containing the maximal intersections among the censoring intervals and model matrices for the multiplicative and additive terms in `formula`. This component is returned only if the `coxaalen.control` argument `data` is true.

References

Boruvka, A. and Cook, R. J. (2015) A Cox-Aalen model for interval-censored data. *Scandinavian Journal of Statistics* **42**, 414–426.

Martinussen, T. and Scheike, T. H. (2006) Dynamic Regression Models for Survival Data. New York: Springer.

Scheike, T. H. and Zhang, M.-J. (2002) An additive-multiplicative Cox-Aalen regression model. *Scandinavian Journal of Statistics* **29**, 75–88.

See Also

`cox.aalen`, `prop`, `Surv`

Examples

```
# Fit a Cox model to the breast cosmesis dataset
if (is.loaded("coxaalen", "coxinterval")) {
  fit <- coxaalen(Surv(left, right, type = "interval2") ~ prop(treat),
                  data = cosmesis, init.timereg = TRUE,
                  formula.timereg = list(Surv(pmax(left, right, na.rm = TRUE),
                  !is.na(right)) ~ .))
  summary(fit)
}
```

`coxaalen.control` *Control a Cox-Aalen model fit*

Description

Set parameters controlling the model fit returned by `coxaalen`.

Usage

```
coxaalen.control(eps = 1e-07, eps.norm = c("max", "grad"),
                 iter.max = 5000, armijo = 1/3, var.coef = TRUE,
                 coef.typ = 1, coef.max = 10, trace = FALSE,
                 thread.max = 1, data = FALSE)
```

Arguments

`eps`	threshold value for the norm used to measure convergence in the parameter estimates.
`eps.norm`	a character string identifying the norm to use in the convergence criteria—either the maximum norm between the current and previous parameter values (`eps.norm = "max"`) or the absolute inner product between the current value and the score (`eps.norm = "grad"`).
`iter.max`	maximum number of iterations to attempt. This ensures that `coxaalen` will eventually exit, even when the convergence criteria is not met. A warning is issued whenever the estimation routine has stopped before converging on a final parameter value.
`armijo`	a scale factor in (0, 1/2) for Armijo's (1966) rule—a line search used to ensure that each iteration achieves an adequate increase in the log-likelihood. The model fit is typically not very sensitive to this value.
`var.coef`	a logical value indicating that standard errors for the multiplicative regression coefficients should be estimated. This is done via profile likelihood—an approach that can require an inordinate amount of processing time under many regression coefficients and larger sample size.
`coef.typ`	a scalar or vector of typical (absolute) values for the multiplicative regression coefficient.

`coef.max`	a scalar or vector of probable upper bounds for the multiplicative regression coefficient. This and the `coef.typ` arguments tune variance estimation via the curvature in the profile log-likelihood.
`trace`	a logical value indicating that CPLEX should print its results to the screen.
`thread.max`	maximum number of CPU threads to allocate to CPLEX. The default value disables multithreading. A value of zero allows CPLEX to set the number of threads automatically. The actual number of threads used is limited by the number of available processors and the CPLEX license.
`data`	a logical value indicating that the object returned by `coxaalen` should contain an element `data` that gives the maximal intersections and the model matrix split into multiplicative and additive terms.

Value

A list of the above arguments with their final values.

References

Boruvka, A. and Cook, R. J. (2015) A Cox-Aalen model for interval-censored data. *Scandinavian Journal of Statistics* **42**, 414–426.

Armijo, L. (1966) Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics* **16**, 1–3.

See Also

`coxaalen`

Examples

```
if (is.loaded("coxaalen", "coxinterval"))
coxaalen(Surv(left, right, type = "interval2") ~ prop(treat),
         data = cosmesis, control = coxaalen.control(iter.max = 2,
         trace = TRUE))
```

`coxdual` *Cox model for a Markov illness-death process under dual censoring*

Description

Fit a Cox model to a progressive Markov illness-death process observed under right-censored survival times and interval- or right-censored progression times.

Usage

```
coxdual(formula, data = parent.frame(), subset, init = NULL,
        formula.coxph = NULL, init.coxph = FALSE, control, ...)
```

Arguments

`formula`	an expression of the form `response ~ terms`, where `response` is anobject returned by the `Surv` function and `terms` contain clustering and state-transition variables identified by the `cluster` and `trans` functions, respectively.
`data`	an optional data frame in which to interpret the variables named in the arguments `formula` and `formula.coxph`.
`subset`	expression specifying which rows of `data` should be used in the fit. All observations are included by default.
`init`	a named list of the vector `coef`, specifying the initial coefficient values, and matrix or data frame `basehaz`, used to the initialize the cumulative baseline transition intensities. The `basehaz` element should contain columns representing the cumulative transition intensity, time and transition type. If the columns do not appear in this order, they should be indicated by the column names `"hazard"`, `"time"` and `"trans"`, respectively. The sorted values used to represent the transition types should respectively denote the initial to intermediate, initial to terminal, and intermediate to terminal state transitions. The `NULL` value for `init` or its components enables default values. For `coef` the default is zero. For `basehaz` the default corresponds to linear functions of time with an upper bound of one. Under any alternatives arising from this or the `init.coxph` arguments, `basehaz` is interpreted as a step function of time and the initial value is its piecewise linear approximation.
`formula.coxph`	an optional formula specifying a model to fit with `coxph` using singly right-censored observations. The shorthand `~ .` indicates the same terms given in `formula`, with the function `strata` in place by `trans`. Under dual right censoring and `init.coxph = TRUE`, `formula.coxph` defaults to same model in `formula`, which is fit to observations singly-right–censored at the earlier censoring time. Multiple formula objects can be provided as a list.
`init.coxph`	a logical value indicating that `init` should be overrided by estimates based on the `coxph` fit to the (first) model specified in `formula.coxph`.
`control`	a named list of parameters controlling the model fit, returned by the function `coxdual.control`. This defaults to `coxdual.control()`.
`...`	additional arguments to be passed to `coxdual.control`.

Details

A valid `formula` argument can be expressed as

```
Surv(<start>, <stop>, <status>)
    ~ cluster(<id>) + trans(<from>, <to>) + <covariate terms>
```

where (`<start>`, `<stop>`] is largest *known* time interval over which individual `<id>` is at risk for a transition between the states `<from>` and `<to>`. The variable `<status>` indicates whether or not a transition is observed to occur at `<stop>`.

Under dual censoring (Boruvka and Cook, 2014), both the originating state and the left endpoint of an at-risk interval may be unknown. This case is handled with `<start> = NA`, `<from> = NA`, `<to>` equal to the index of the terminal state, and any transition-type–specific covariates taking on the values assumed when `<from>` is equal to the intermediate state index. Under discrete observation of non-terminal events, the right-endpoint of some at-risk intervals may be unknown. For these `<start>` is the initial observation time (zero, unless left-truncated), `<stop> = NA` and `<from>` is equal to the initial state index. Missing values are retained by the `NA` action `na.coxdual`. The default `NA` action is used to handle any missing values passed to `coxph` via the arguments `formula.coxph` or `init.coxph`.

Dual censoring typically arises in two scenarios: (1) dual right-censoring, where intermediate events are right-censored before terminal events, and (2) interval-censored intermediate events. For examples of these refer to `dualrc` and `dualic`, respectively.

A consequence of dual censoring is that any discrete maximum likelihood estimator has ambiguous support at any failure times associated with these `NA` values. To resolve this, the cumulative baseline transition intensities are restricted to piecewise linear functions on a sieve partition with size controlled by arguments passed to `coxdual.control`. This approach requires that both types of transitions to the terminal state are, at least for some subjects, observed exactly.

Value

An object of the classes `"coxinterval"` and `"coxdual"`, which is a list with the following components.

`call`	the matched call to `coxdual`.
`censor`	a string indicating the dual censoring type. The value "right" corresponds to strictly dual-right–censored data. All other cases return "interval".
`n`	size of the sample used in the model fit.
`m`	number of at-risk intervals used in the model fit.
`p`	number of regression coefficients.
`coef`	a named `p` vector of regression coefficients.
`var`	a named `p` by `p` covariance matrix of the regression coefficients.
`basehaz`	a data frame giving the cumulative baseline transition intensities evaluated over the sieve partition.
`init`	list of initial values used in the model fit.
`loglik`	a vector giving the log-likelihood at initiation and each iteration.
`iter`	number of iterations needed to reach the stopping criteria.
`gradnorm`	the maximum norm of the score scaled by the parameter value at the final iteration.
`maxnorm`	the maximum norm of the difference between the penultimate and final parameter values.
`cputime`	the processing time used for parameter and variance estimation.
`fit.coxph`	the `coxph` fit to any models specified by the `formula.coxph` argument. If `formula.coxph` is a list of formula objects, `fit.coxph` is an unnamed list following the same order.
`na.action`	the `"na.action"` attribute of the model frame. Here this corresponds to the result from the custom `NA` action `na.coxdual`.
`censor.rate`	a named vector of censoring rates.

control	a list of arguments passed to coxdual.control.
data	a list containing the data relevant to the model fit, if the coxdual.control argument data is true.

References

Boruvka, A. and Cook, R. J. (2014) Sieve estimation in a Markov illness-death process under dual censoring.

See Also

cluster, dualic, dualrc, Surv, trans

Examples

```
# Fit Cox model to dual-right--censored data
fit <- coxdual(Surv(start, stop, status) ~ cluster(id) + trans(from, to)
               + I(z * (to == 1)) + I(z * (from %in% 0 & to == 2))
               + I(z * (from %in% c(NA, 1) & to == 2)), data = dualrc,
               sieve.rate = 2/5)
fit
par(mfrow = c(1, 3))
by(fit$basehaz, fit$basehaz$trans, function(x) plot(x[, 2:1],
   type = "l", main = paste(x[1, 3]), xlim = c(0, 2), ylim = c(0, 4)))

# Fit Cox model to data with interval-censored progression times
fit <- coxdual(Surv(start, stop, status) ~ cluster(id) + trans(from, to)
               + I(z * (to == 1)) + I(z * (from %in% 0 & to == 2))
               + I(z * (from %in% c(NA, 1) & to == 2)), data = dualic)
fit
par(mfrow=c(1, 3))
by(fit$basehaz, fit$basehaz$trans, function(x) plot(x[, 2:1],
   type = "l", main = paste(x[1, 3]), xlim = c(0, 2), ylim = c(0, 4)))
```

---

coxdual.control *Control Cox model fit*

Description

Set parameters controlling the model fit returned by coxdual.

Usage

```
coxdual.control(eps = 1e-07, iter.max = 50000, coef.typ = 1,
                coef.max = 10, sieve = TRUE, sieve.const = 1,
                sieve.rate = 1/3, risk.min = 1, data = FALSE)
```

Arguments

`eps`	threshold value for the norm used to measure convergence in the parameter estimates.
`iter.max`	maximum number of iterations to attempt. This ensures that `coxdual` will eventually exit, even when the convergence criteria are not met.
`coef.typ`	a scalar or vector of typical (absolute) values for the regression coefficient.
`coef.max`	a scalar or vector of probable upper bounds for the regression coefficient. This and the `coef.typ` arguments tune variance estimation via the curvature in the profile log-likelihood.
`sieve`	a logical value indicating that the sieve rather than the semiparametric maximum likelihood estimator should be fit to the data. The default `TRUE` is recommended to avoid issues with support finding and convergence.
`sieve.const`	a constant factor that, in part, determines the sieve size. The factor can be made specific to the transition type with `sieve.const` a vector of length three. Indexing the states from zero, this vector's components correspond to the state 0 to state 1, 0 to 2, and 1 to 2 transition types, respectively.
`sieve.rate`	a scalar in (1/8, 1/2) determining the rate at which the sieve increases with the sample size.
`risk.min`	a positive integer giving the minimum size of risk set for support points defining the sieve.
`data`	a logical value indicating that the object returned by `coxdual` should contain an element `data` that gives the known support points, corresponding size of the risk set, left and right endpoints of censoring intervals for the progression time, first and last observation times, likelihood contribution type (`0` progression status unknown, `1` positive status, `2` negative status), survival time observed, and type-specific covariates.

Details

For a given sample size n, the resulting sieve has size at most `sieve.const*`n`^sieve.rate`. Any reduction in size from this value is applied to ensure that each subinterval in the sieve's time partition captures at least one support point from the semiparametric maximum likelihood estimator based on the subsample with known progression status (Boruvka and Cook, 2014).

Value

A list of the above arguments with their final values.

References

Boruvka, A. and Cook, R. J. (2014) Sieve estimation in a Markov illness-death process under dual censoring.

See Also

`coxdual`

Examples

```
coxdual(Surv(start, stop, status) ~ cluster(id) + trans(from, to)
        + I(z * (to == 1)) + I(z * (from %in% 0 & to == 2))
        + I(z * (from %in% c(NA, 1) & to == 2)), data = dualrc,
        control = coxdual.control(eps = 1e-5, sieve.rate = 2/5))
```

`dualic` *Simulated dual-censored data from an illness-death process*

Description

Data from a Markov illness-death process with interval-censored progression times, simulated according to the initial scenario described in Boruvka and Cook (2014).

Usage

```
data(dualic)
```

Format

A data frame with 723 observations on the following 7 variables.

`id` subject identifier.

`from` originating state index with `0` denoting the initial state, `1` the intermediate state, `2` the terminal state and `NA` an unknown state.

`to` subsequent state index.

`start` left endpoint of the time interval at which the subject is known to be at risk for a transition between state `from` and state `to`.

`stop` right endpoint of the at-risk interval.

`status` indicator that a transition between state `from` and state `to` was observed at `stop`.

`z` a binary covariate.

References

Boruvka, A. and Cook, R. J. (2014) Sieve estimation in a Markov illness-death process under dual censoring.

See Also

`coxdual`

Examples

```
data(dualic)
```

`dualrc` *Simulated dual-right–censored data from an illness-death process*

Description

Data from a Markov illness-death process with progression and death observed under dual right censoring, simulated according to the initial scenario described in Boruvka and Cook (2014).

Usage

```
data(dualrc)
```

Format

A data frame with 644 observations on the following 7 variables.

- `id` subject identifier.
- `from` originating state index with `0` denoting the initial state, `1` the intermediate state, `2` the terminal state and `NA` an unknown state.
- `to` subsequent state index.
- `start` left endpoint of the time interval at which the subject is known to be at risk for a transition between state `from` and state `to`.
- `stop` right endpoint of the at-risk interval.
- `status` indicator that a transition between state `from` and state `to` was observed at `stop`.
- `z` a binary covariate.

References

Boruvka, A. and Cook, R. J. (2014) Sieve estimation in a Markov illness-death process under dual censoring.

See Also

`coxdual`

Examples

```
data(dualrc)
```

`jump2step` *Accumulate increments in values from a step function of time*

Description

A utility function for `coxdual` that returns values from a step function given its increments over time.

Usage

```
jump2step(jump, time = "time", stratum = NULL)
```

Arguments

`jump`	a matrix whose columns give increments in the (possibly multivariate) step function, corresponding time points and strata.
`time`	an integer or character value indicating the column index or name of the time variable in the matrix `jump`. Defaults to `"time"`.
`stratum`	integer or character value for the column index or name of the stratifying variable in the matrix `jump`. If `stratum` is `NULL` (default), `jump` is presumed unstratified.

Value

A matrix of the same dimension as `jump`.

See Also

`step2jump`

Examples

```
fit <- coxdual(Surv(start, stop, status) ~ cluster(id)
               + trans(from, to) + z, data = dualrc, init.coxph = TRUE)
haz <- step2jump(fit$coxph$basehaz, stratum = 3)
Haz <- jump2step(haz, stratum = 3)
all(Haz == fit$coxph$basehaz)
```

lin2const *Slopes from values of a piecewise linear function of time*

Description

A utility function for coxdual that gives the slope corresponding to values from a piecewise linear function of time.

Usage

```
lin2const(lin, time = "time", stratum = NULL)
```

Arguments

lin	a matrix whose columns give the (possibly multivariate) piecewise linear function values, time points and strata.
time	an integer or character value indicating the column index or name of the time variable in the matrix lin. Defaults to "time".
stratum	integer or character value for the column index or name of the stratifying variable in the matrix lin. If stratum is NULL (default), lin is presumed unstratified.

Details

This is a utility function for coxdual.

Value

A matrix of the same dimension as lin.

See Also

const2lin

Examples

```
data(dualrc)
fit <- coxdual(Surv(start, stop, status) ~ cluster(id)
               + trans(from, to) + z, data = dualrc, init.coxph = TRUE)
fit$basehaz
haz <- lin2const(fit$basehaz, stratum = 3)
Haz <- const2lin(haz, stratum = 3)
all(Haz == fit$basehaz)
```

`linapprox` *Linear approximation*

Description

Perform linear interpolation or extrapolation

Usage

```
linapprox(xyin, xout)
```

Arguments

`xyin`	a matrix whose rows give the coordinate pairs of the points to be interpolated or extrapolated.
`xout`	a vector of numeric values at which interpolation or extrapolation should take place.

Value

A vector of interpolated or extrapolated values.

See Also

`approx` `const2lin` `lin2const`

Examples

```
fit <- coxdual(Surv(start, stop, status) ~ cluster(id)
               + trans(from, to) + z, data = dualrc, init.coxph = TRUE)
head(basehaz)
Haz01 <- with(fit, split(basehaz[, 1:2], basehaz[, 3]))[[1]]
all(Haz01$hazard == with(Haz01, linapprox(cbind(time, hazard), time)))
```

`maximalint` *Find maximal intersections from an interval-type survival object*

Description

A utility function that returns the "maximal intersections" from a set of censoring intervals in a given `type = "interval"` or `type = "interval2"` `Surv` object.

Usage

```
maximalint(x, eps = 1e-7)
```

Arguments

`x`	a two-column matrix or data frame giving the left- and right-endpoints of the censoring intervals or, alternatively, a `type = "interval"` or `type = "interval2"` `Surv` object. Following the `type = "interval2"` format, any `NA`-valued right endpoints are considered the same as `Inf`.
`eps`	a small value used to break ties in maximal intersection endpoints. Note that a large time scale requires a larger epsilon in order to ensure that `t != t + eps` for a given time value `t`.

Details

Each censoring interval is assumed to exclude its left endpoint. Analogous to Maathuis (2005, Section 2.1) ties between the unique (open) left and (closed) right endpoints among the censoring intervals are broken by subtracting a small value from the right endpoint value before constructing the maximal intersections.

Value

`int`	a two-column matrix whose rows give the maximal intersections (Wong and Yu, 1999) of the censoring intervals represented in `x`
`ind`	a `nrow(x)` by `nrow(int)` matrix that indicates overlap between each maximal intersection and censoring interval

References

Maathuis, MH (2005) Reduction algorithm for the NPMLE for the distribution function of bivariate interval-censored data. *Journal of Computational and Graphical Statistics* **14**, 352–362.

Wong, GYC. and Yu, Q. (1999) Generalized MLE of a joint distribution function with multivariate interval-censored data. *Journal of Multivariate Analysis* **69**, 155–166.

Examples

```
s <- with(cosmesis[1:10, ], Surv(left, right, type = "interval2"))
s
maximalint(s)
```

`print.coxinterval` *Print method for model fit*

Description

Prints an object the class `"coxinterval"`.

Usage

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

Arguments

`x`	an object returned by `coxaalen` or `coxdual`.
`...`	further arguments for other methods.

See Also

`summary.coxinterval`

`step2jump` *Increments from a step function of time*

Description

A utility function for `coxdual` that returns increments from values of a step function over time.

Usage

```
step2jump(step, time = "time", stratum = NULL)
```

Arguments

`step`	a matrix whose columns give the (possibly multivariate) step function values, corresponding time points and strata.
`time`	an integer or character value indicating the column index or name of the time variable in the matrix `step`. Defaults to `"time"`.
`stratum`	integer or character value for the column index or name of the stratifying variable in the matrix `step`. If `stratum` is `NULL` (default), `step` is presumed unstratified.

Value

A matrix of the same dimension as `step`.

See Also

`jump2step`

Examples

```
fit <- coxdual(Surv(start, stop, status) ~ cluster(id)
               + trans(from, to) + z, data = dualrc, init.coxph = TRUE)
haz <- step2jump(fit$coxph$basehaz, stratum = 3)
Haz <- jump2step(haz, stratum = 3)
all(Haz == fit$coxph$basehaz)
```

step2stepfun *Step function from cumulative increments over time*

Description

A utility function for that returns a step function given cumulative increments over time.

Usage

```
step2stepfun(step, time = "time", stratum = NULL)
```

Arguments

`step`	a matrix whose columns give the (possibly multivariate) step function values, corresponding time points and strata.
`time`	an integer or character value indicating the column index or name of the time variable in the matrix `step`. Defaults to `"time"`.
`stratum`	integer or character value for the column index or name of the stratifying variable in the matrix `step`. If `stratum` is `NULL` (default), `step` is presumed unstratified.

Value

A function that returns cumulative increments for given times. If stratified, a list of such functions given in the same order as the strata.

See Also

`jump2step` `step2jump`

Examples

```
fit <- coxdual(Surv(start, stop, status) ~ cluster(id)
               + trans(from, to) + z, data = dualrc, init.coxph = TRUE)
head(fit$coxph$basehaz)
Hazfun <- step2stepfun(fit$coxph$basehaz, stratum = 3)
Haz01 <- with(fit$coxph, split(basehaz[, 1:2], basehaz[, 3]))[[1]]
all(Hazfun[[1]](Haz01$time) == Haz01$hazard)
```

summary.coxinterval — *Summary method for Cox model fit*

Description

summary method for the class "coxinterval".

Usage

```
## S3 method for class 'coxinterval'
summary(object, conf.int = 0.95, scale = 1, ...)
```

Arguments

object	an object returned by coxaalen or coxdual.
conf.int	level for confidence intervals. If FALSE, no confidence intervals are provided.
scale	scale factor for the confidence intervals, whose limits represent the change in risk associated with one scale unit increase in the corresponding variable.
...	further arguments for other methods.

Value

An object of the class "summary.coxinterval".

Examples

```
# Fit Cox model to dual-right--censored data
fit <- coxdual(Surv(start, stop, status) ~ cluster(id) + trans(from, to)
               + I(z * (to == 1)) + I(z * (from %in% 0 & to == 2))
               + I(z* (from %in% c(NA, 1) & to == 2)), data = dualrc,
               sieve.rate = 2/5)
fit
```

trans — *Identify transition type in model terms*

Description

A special function for coxdual that identifies formula terms specifying the state-transition type.

Usage

```
trans(from, to)
```

Arguments

`from`	a variable representing the originating state.
`to`	a variable representing the subsequent state.

Value

A combination of the `from` and `to` arguments by column with two attributes:

`"states"`	a vector giving the unique non-missing values in the `from` and `to` arguments ordered so that the initial state appears first, the intermediate state second, and the terminal state last.
`"types"`	a vector of transition type labels in terms of the values in the `from` and `to` arguments ordered so that the intermediate transition appears first, the terminal transition directly from the initial state second, and the terminal transition from the intermediate state last.

See Also

`coxdual`

Examples

```
with(dualrc[1:10, ], trans(from, to))
```

Index