Package 'HQM'

December 3, 2025

Type Package

Title Superefficient Estimation of Future Conditional Hazards Based on Marker Information

Version 1.1

Maintainer Dimitrios Bagkavos <dimitrios.bagkavos@gmail.com>

Description Provides univariate and indexed (multivariate) nonparametric smoothed kernel estimators for the future conditional hazard rate function when time-dependent covariates are present, a bandwidth selector for the estimator's implementation and pointwise and uniform confidence bands. Methods used in the package refer to Bagkavos, Isakson, Mammen, Nielsen and Proust-Lima (2025) <doi:10.1093/biomet/asaf008>.

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Imports stats, utils, survival, pec, timeROC, nlme, JM

License GPL (>= 2)

NeedsCompilation no

RoxygenNote 6.1.0

Author Dimitrios Bagkavos [aut, cre],
Alex Isakson [ctb],
Enno Mammen [ctb],
Jens Nielsen [ctb],
Cecile Proust-Lima [ctb]

Repository CRAN

Date/Publication 2025-12-03 21:00:10 UTC

Contents

`auc.hqm` *AUC for the High Quality Marker estimator*

Description

Calculates the AUC for the HQM estimator.

Usage

```
auc.hqm(xin, est, landm, th, event_time_name, status_name)
```

Arguments

xin	A data frame containing event times and the patient status.
est	The HQM estimator values, typically the output of get_h_x.
landm	Landmark time.
th	Time horizon.
event_time_name	The column name of the event times in the xin data frame.
status_name	The column name of the status variable in the xin frame.

Details

The function auc.hqm implements the AUC calculation for the HQM estimator estimator.

Value

A vector of two values: the landmark time of the calculation and the AUC value.

See Also

bs.hqm

Examples

```
library(timeROC)
library(survival)
 Landmark <- 2
 pbcT1 <- pbc2[which(pbc2$year< Landmark  & pbc2$years> Landmark),]
 timesS2 <- seq(Landmark,14,by=0.5)
 b=0.9
 arg1<- get_h_x(pbcT1, 'albumin', event_time_name = 'years',
                time_name = 'year', event_name = 'status2', 2, 0.9)
 br_s2  = seq(Landmark, 14,  length=99)
 sfalb2<- make_sf(    (br_s2[2]-br_s2[1])/4 , arg1)
 tHor <- 1.5
 auc.hq.use<-auc.hqm(pbcT1, sfalb2, Landmark,tHor,
              event_time_name = 'years', status_name = 'status2')
 auc.hq.use
```

Boot.hqm *Indexed HQM hazard estimator for one bootstrap sample*

Description

Compute the bootstrap estimate of the indexed HQM hazard estimate for a single (bootstrap) sample. The function is meant to be used internally for the calculation of confidence intervals

Usage

```
Boot.hqm(in.par, data, data.id, X1, XX1, X2, XX2, event_time_name = 'years',
                 time_name = 'year', event_name = 'status2', b, t)
```

Arguments

in.par	Numeric vector, the values of the indexing parameters.
data	Bootstrap data.frame.
data.id	Id-level data.frame (result of to_id).
X1, XX1, X2, XX2	Vectors, longitudinaal and id level marker values for indexing.
event_time_name	Name of event time column.
time_name	Name of observation time column.
event_name	Name of event indicator column.
b	Bandwidth parameter.
t	Conditioning level.

Value

Numeric vector with estimated hazard on the grid.

Examples

```
#Single instance of the bootstrap version of the bootstrap version
#of the indexed HQM estimator

b.alb = 0.9
b.bil = 4

t.alb = 1 # refers to zero mean variables - slightly high
t.bil = 1.9 # refers to zero mean variable - high

par.alb  <- 0.0702 #0.149
par.bil <- 0.0856 #0.10


b = 0.42 # The result, on the indexed marker 'indmar' of
      #\code{b_selection(pbc2, 'indmar', 'years', 'year', 'status2', I=26, seq(0.2,0.4,by=0.01))}
t = t.alb * par.alb + t.bil *par.bil

marker_name1 <- 'albumin'
marker_name2 <-  'serBilir'
event_time_name <- 'years'
time_name <- 'year'
event_name <- 'status2'
id<-'id'

data.use<-pbc2
```

```
data.use.id<-to_id(data.use)
data.use.id<-data.use.id[complete.cases(data.use.id), ]

# mean adjust the data:
X1=data.use[,marker_name1] -mean(data.use[, marker_name1])
XX1=data.use.id[,marker_name1] -mean(data.use.id[, marker_name1])
X2=data.use[,marker_name2]  -mean(data.use[, marker_name2])
XX2=data.use.id[,marker_name2] -mean(data.use.id[, marker_name2])

boot.haz<-Boot.hqm (c(par.alb,par.bil), data.use, data.use.id, X1, XX1, X2, XX2,
              event_time_name = 'years', time_name = 'year', event_name = 'status2',  b, t)
```

`Boot.hrandindex.param` *Bootstrap Estimation of Hazard Function and Index Parameters*

Description

Performs bootstrap estimation of hazard rates and corresponding index parameters for a given set of bootstrap samples. For each bootstrap replicate, the function re-estimates the index parameters via optimisation and computes hazard estimates using `Boot.hqm`. The output is used as input in functions `Quantile.Index.CIs` and `Pivot.Index.CIs`

Usage

```
Boot.hrandindex.param(B, Boot.samples, marker_name1, marker_name2,event_time_name,
                     time_name, event_name, b, t, true.haz, v.param, n.est.points)
```

Arguments

`B`	Integer. Number of bootstrap iterations.
`Boot.samples`	A list of bootstrap datasets. Each element corresponds to one replicate.
`marker_name1`	Character string. Name of the first longitudinal marker.
`marker_name2`	Character string. Name of the second longitudinal marker.
`event_time_name`	Name of the event time variable in the data.
`time_name`	Name of the time variable for the longitudinal marker measurements.
`event_name`	Name of the event indicator variable.
`b`	Numeric. Bandwidth parameter used in hazard estimation.
`t`	Numeric. Evaluation point for the conditional hazard.
`true.haz`	Numeric vector. The true or reference hazard used in the optimisation criterion.
`v.param`	Numeric vector. Starting values of the indexing parameters for the optimisation of the index coefficients.
`n.est.points`	Integer. Number of estimation grid points for the hazard curve.

Details

For each bootstrap iteration $k = 1, \ldots, B$, the function:

1. Extracts the bootstrap sample `data.use`.
2. Computes centred marker values at the subject and observation level.
3. Estimates index parameters by minimising `index_optim` using `optim`.
4. Computes the bootstrap hazard estimate via `Boot.hqm`.

The outputs are matrices collecting the hazard estimates and estimated index parameter vectors across bootstrap replicates.

Value

A matrix of dimension `n.est.points` × `B` containing the bootstrap hazard estimates.

See Also

`Boot.hqm`, `index_optim`, `to_id`

Examples

```
marker_name1 <- 'albumin'
marker_name2 <-  'serBilir'
event_time_name <- 'years'
time_name <- 'year'
event_name <- 'status2'
id<-'id'

par.x1  <- 0.0702
par.x2 <- 0.0856
t.x1 = 0 # refers to zero mean variables - slightly high
t.x2 = 1.9 # refers to zero mean variable - high
b = 0.42
t = par.x1 * t.x1 + par.x2 *t.x2

# first simulate true HR function:
xin <- pbc2[,c(id, marker_name1, marker_name2, event_time_name, time_name, event_name)]
n <- length(xin$id)
nn<-max(  as.double(xin[,'id']) )

#  Create bootstrap samples by group:
set.seed(1)
B<-100
Boot.samples<-list()
for(j in 1:B)
{
  i.use<-c()
  id.use<-c()
  index.nn <- sample (nn, replace = TRUE)
  for(l in 1:nn)
  {
```

```
    i.use2<-which(xin[,id]==index.nn[l])
    i.use<-c(i.use, i.use2)
    id.use2<-rep(index.nn[l], times=length(i.use2))
    id.use<-c(id.use, id.use2)
  }
  xin.i<-xin[i.use,]
  xin.i<-xin[i.use,]
  Boot.samples[[j]]<- xin.i[order(xin.i$id),] #xin[i.use,]
}
true.hazard<- Sim.True.Hazard(Boot.samples, id='id', marker_name1=marker_name1,
                    marker_name2= marker_name2, event_time_name = event_time_name,
                    time_name = time_name,  event_name = event_name,
                    in.par = c(par.x1,  par.x2), b)

res <- Boot.hrandindex.param( B,  Boot.samples, marker_name1, marker_name2,
          event_time_name,  time_name,  event_name ,  b = 0.4, t = 1.0,
          true.haz = true.hazard, v.param = c(0.07, 0.08), n.est.points = 100)

#return bootstrap hazard rate estimators in marix format:
res
```

`bs.hqm` *Brier score for the High Quality Marker estimator*

Description

Calculates the Brier score for the HQM estimator.

Usage

```
bs.hqm(xin, est, landm, th, event_time_name, status_name)
```

Arguments

`xin`	A data frame containing event times and the patient status.
`est`	The HQM estimator values, typically the output of `get_h_x`.
`landm`	Landmark time.
`th`	Time horizon.
`event_time_name`	The column name of the event times in the data frame `xin`.
`status_name`	The column name of the status variable in the data frame `xin`.

Details

The function `bs.hqm` implements the Brier score calculation for the HQM estimator estimator.

Value

Scalar: the Brier score of the HQM estimator.

See Also

auc.hqm

Examples

```
library(pec)
library(survival)
Landmark <- 2

pbcT1 <- pbc2[which(pbc2$year< Landmark  & pbc2$years> Landmark),]
timesS2 <- seq(Landmark,14,by=0.5)


b=0.9
arg1<- get_h_x(pbcT1, 'albumin', event_time_name = 'years',
               time_name = 'year', event_name = 'status2', 2, 0.9)
br_s2  = seq(Landmark, 14,  length=99)
sfalb2<- make_sf(    (br_s2[2]-br_s2[1])/4 , arg1)


tHor <- 1.5
bs.use<-bs.hqm(pbcT1, sfalb2, Landmark,tHor,
                event_time_name = 'years', status_name = 'status2')
bs.use
```

BwB.HRandIndex.param *Bootstrap Estimation of Hazard Function and Index Parameters*

Description

Performs bootstrap estimation of hazard rates and their standard deviation at each grid point (double boostrap) together with the corresponding index parameters for a given set of bootstrap samples. The output of the function is used as input in StudentizedBwB.Index.CIs.

Usage

```
BwB.HRandIndex.param(B, B1, Boot.samples, marker_name1, marker_name2,
                         event_time_name, time_name, event_name, b, t, true.haz,
                            v.param,  hqm.est, id, xin)
```

Arguments

B	Integer. Number of bootstrap samples.
B1	Integer. Number of bootstrap re-samples.
Boot.samples	A list of bootstrap datasets. Each element corresponds to one replicate.
marker_name1	Character string. Name of the first longitudinal marker.
marker_name2	Character string. Name of the second longitudinal marker.
event_time_name	Name of the event time variable in the data.
time_name	Name of the time variable for the longitudinal marker measurements.
event_name	Name of the event indicator variable.
b	Numeric. Bandwidth parameter used in hazard estimation.
t	Numeric. Evaluation point for the conditional hazard.
true.haz	Numeric vector. The true or reference hazard used in the optimisation criterion.
v.param	Numeric vector. Starting values of the indexing parameters for the optimisation of the index coefficients.
hqm.est	HQM estimator on the original sample.
id	label of id variable of dataset.
xin	original sample.

Details

For each bootstrap iteration $k = 1, \ldots, B$, the function:

1. Extracts the bootstrap sample `data.use`.
2. Computes centred marker values at the subject and observation level.
3. Estimates index parameters by minimising `index_optim` using `optim`.
4. Computes the bootstrap hazard estimate via `Boot.hqm`.

The outputs are matrices collecting the hazard estimates and estimated index parameter vectors across bootstrap replicates.

Value

A list of matrices of dimension `n.est.points` × `B` containing the bootstrap hazard estimates, the logarithm of the hazard rate estimates and and two vectors of the estimate's standard deviations at each grid point.

See Also

`Boot.hqm`, `index_optim`, `to_id`

Examples

```
# See the example for  function: StudentizedBwB.Index
```

b_selection *Cross validation bandwidth selection*

Description

Implements the bandwidth selection for the future conditional hazard rate $\hat{h}_x(t)$ based on K-fold cross validation.

Usage

```
b_selection(data, marker_name, event_time_name = 'years',
            time_name = 'year', event_name = 'status2', I, b_list)
```

Arguments

data	A data frame of time dependent data points. Missing values are allowed.
marker_name	The column name of the marker values in the data frame data.
event_time_name	The column name of the event times in the data frame data.
time_name	The column name of the times the marker values were observed in the data frame data.
event_name	The column name of the events in the data frame data.
I	Number of observations leave out for a K cross validation.
b_list	Vector of bandwidths that need to be tested.

Details

The function b_selection implements the cross validation bandwidth selection for the future conditional hazard rate $\hat{h}_x(t)$ given by

$$b_{CV} = argmin_b \sum_{i=1}^{N} \int_0^T \int_s^T Z_i(t) Z_i(s) (\hat{h}_{X_i(s)}(t-s) - h_{X_i(s)}(t-s))^2 dt ds,$$

where $\hat{h}_x(t)$ is a smoothed kernel density estimator of $h_x(t)$ and Z_i the exposure process of individual i. Note that $\hat{h}_x(t)$ is dependent on b.

Value

A list with the tested bandwidths and its cross validation scores.

References

Bagkavos, I., Isakson, R., Mammen, E., Nielsen, J., and Proust–Lima, C. (2025). Biometrika, 112(2), asaf008. doi:10.1093/biomet/asaf008

See Also

b_selection_prep_g, Q1, R_K, prep_cv, dataset_split

Examples

```
I = 26
# For Albumin marker:
b_list = seq(0.9, 1.7, 0.1)

b_scores_alb = b_selection(pbc2, 'albumin', 'years', 'year', 'status2', I, b_list)
b_scores_alb[[2]][which.min(b_scores_alb[[1]])]

# For Bilirubin marker:
b_list = seq(3, 4, 0.1)
b_scores_bil = b_selection(pbc2, 'serBilir', 'years', 'year', 'status2', I, b_list)
b_scores_bil[[2]][which.min(b_scores_bil[[1]])]
b_scores_bil
```

---

b_selection_index_optim

Cross validation index parameter selection

Description

Implements the index parameter selection for two markers based on K-fold cross validation.

Usage

```
b_selection_index_optim(in.par, data, marker_name1, marker_name2,
       event_time_name = 'years', time_name = 'year', event_name = 'status2', I, b)
```

Arguments

in.par	Vector of candidate values for the index parameters.
data	A data frame of time dependent data points. Missing values are allowed.
marker_name1	The column name of the first marker values in the data frame data.
marker_name2	The column name of the second marker values in the data frame data.
event_time_name	The column name of the event times in the data frame data.
time_name	The column name of the times the marker values were observed in the data frame data.
event_name	The column name of the events in the data frame data.
I	Number of observations leave out for a K cross validation.
b	scalar bandwidth for the HQM estimator.

Details

The function b_selection_index_optim implements the cross validation index parameter selection for the indexing of two markers and given by

$$\hat{\theta} = argmin_{\theta_1,\theta_2} \sum_{i=1}^{N} \int_0^T \int_s^T Z_i(t) Z_i(s) (\hat{h}_{\theta_0^T X_i(s)}(t-s) - h_{\theta_0^T X_i(s)}(t-s))^2 dt ds,$$

where $\hat{h}_x(t)$ is the HQM estimator of $h_x(t)$, see doi:10.1093/biomet/asaf008, and Z_i the exposure process of individual i. Note that $\hat{h}_x(t)$ is dependent on b.

Value

A list with the tested parameters and its cross validation scores.

References

Bagkavos, I., Isakson, R., Mammen, E., Nielsen, J., and Proust–Lima, C. (2025). Biometrika, 112(2), asaf008. doi:10.1093/biomet/asaf008

See Also

b_selection_prep_g, Q1, R_K, prep_cv, dataset_split

Examples

```
# Obtain the indexing parameters for the combination of albumin and bilirubin markers
# These are the values used in the example of function h_xt_vec
# and yielded the values:  (par.alb, par.bil) = ( 0.0702, 0.0856 ) that were used there.

I = 26
b_list = seq(1.1, 1.2, by=0.1)

for(i in 1:length(b_list))
{
  res<- optim(c(0.5, 0.5), b_selection_index_optim,  data=pbc2, marker_name1='albumin',
              marker_name2= 'serBilir', event_time_name = 'years', time_name = 'year',
              event_name = 'status2', I=26, b=b_list[i], method="Nelder-Mead")
 cat("i= ", i, " ", res$par, " ", res$value, "count calls to fn = ", res$counts, " converge? ",
       res$convergence,   "\n")
  res
}
```

b_selection_prep_g *Preparations for bandwidth selection*

Description

Calculates an intermediate part for the K-fold cross validation.

Usage

```
b_selection_prep_g(h_mat, int_X, size_X_grid, n, Yi)
```

Arguments

h_mat	A matrix of the estimator for the future conditional hazard rate for all values x and t.
int_X	Vector of the position of the observed marker values in the grid for marker values.
size_X_grid	Numeric value indicating the number of grid points for marker values.
n	Number of individuals.
Yi	A matrix made by make_Yi indicating the exposure.

Details

The function b_selection_prep_g calculates a key component for the bandwidth selection

$$\hat{g}_i^{-I_j}(t) = \int_0^t Z_i(s)\hat{h}^{-I_j}_{X_i(s)}(t-s)ds,$$

where $\hat{h}^{-I_j}$ is estimated without information from all counting processes i with $i \in I_j$ and Z is the exposure.

Value

A matrix with $\hat{g}_i^{-I_j}(t)$ for all individuals i and time grid points t.

See Also

b_selection

Examples

```
pbc2_id = to_id(pbc2)
size_s_grid <- size_X_grid <- 100
n = max(as.numeric(pbc2$id))
s = pbc2$year
X = pbc2$serBilir
XX = pbc2_id$serBilir
```

```
ss <- pbc2_id$years
delta <- pbc2_id$status2
br_s = seq(0, max(s), max(s)/( size_s_grid-1))
br_X = seq(min(X), max(X), (max(X)-min(X))/( size_X_grid-1))

X_lin = lin_interpolate(br_s, pbc2_id$id, pbc2$id, X, s)

int_X <- findInterval(X_lin, br_X)
int_s = rep(1:length(br_s), n)

N <- make_N(pbc2, pbc2_id, breaks_X=br_X, breaks_s=br_s, ss, XX, delta)
Y <- make_Y(pbc2, pbc2_id, X_lin, br_X, br_s, size_s_grid, size_X_grid,
            int_s, int_X, 'years', n)

b = 1.7
alpha<-get_alpha(N, Y, b, br_X, K=Epan )

Yi <- make_Yi(pbc2, pbc2_id, X_lin, br_X, br_s, size_s_grid, size_X_grid, int_s,
              int_X,'years', n)
Ni  <- make_Ni(breaks_s=br_s, size_s_grid, ss, delta, n)

t = 2

h_xt_mat = t(sapply(br_s[1:99], function(si){
          h_xt_vec(br_X, br_s, size_s_grid, alpha, t, b, Yi, int_X, n)}))

b_selection_prep_g(h_xt_mat, int_X, size_X_grid, n, Yi)
```

`Conf_bands` *Confidence bands*

Description

Implements the uniform and pointwise confidence bands for the future conditional hazard rate based on the last observed marker measure.

Usage

```
Conf_bands(data, marker_name, event_time_name = 'years',
            time_name = 'year', event_name = 'status2', x, b)
```

Arguments

`data`	A data frame of time dependent data points. Missing values are allowed.
`marker_name`	The column name of the marker values in the data frame `data`.
`event_time_name`	The column name of the event times in the data frame `data`.
`time_name`	The column name of the times the marker values were observed in the data frame `data`.

event_name	The column name of the events in the data frame data.
x	Numeric value of the last observed marker value.
b	Bandwidth.

Details

The function Conf_bands implements the pointwise and uniform confidence bands for the estimator of the future conditional hazard rate $\hat{h}_x(t)$. The confidence bands are based on a wild bootstrap approach ${h^*}_{x_*,B}(t)$.

Pointwise: For a given $t \in (0,T)$ generate ${h^*}^{(1)}_{x_*,B}(t), ..., {h^*}^{(N)}_{x_*,B}(t)$ for $N = 1000$ and order it ${h^*}^{[1]}_{x_*,B}(t) \leq ... \leq {h^*}^{[N]}_{x_*,B}(t)$. Then

$$\hat{I}^1_{n,N} = \left[\hat{h}_{x_*}(t) - \hat{\sigma}_{G_{x_*}}(t)\frac{h^{*[N(1-\frac{\alpha}{2})]}_{x_*,B}(t)}{\sqrt{n}}, \hat{h}_{x_*}(t) - \hat{\sigma}_{G_x}(t)\frac{h^{*[N\frac{\alpha}{2}]}_{x_*,B}(t)}{\sqrt{n}}\right]$$

is a $1-\alpha$ pointwise confidence band for $h_{x_*}(t)$, where $\hat{\sigma}_{G_{x_*}}(t)$ is a bootrap estimate of the variance. For more details on the wild bootstrap approach, please see prep_boot and g_xt.

Uniform: Generate $\bar{h}^{(1)}_{x_*,B}(t), ..., \bar{h}^{(N)}_{x_*,B}(t)$ for $N = 1000$ for all $t \in [\delta_T, T - \delta_T]$ and define $W^{(i)} = \sup_{t\in[0,T]} |\bar{h}^{(i)}_{x_*,B}(t)|$ for $i = 1, ..., N$. Order $W^{[1]} \leq ... \leq W^{[N]}$. Then

$$\hat{I}^2_{n,N} = \left[\hat{h}_{x_*}(t) \pm \hat{\sigma}_{G_{x_*}}(t)\frac{W^{[N(1-\alpha)]}}{\sqrt{n}}\right]$$

is a $1-\alpha$ uniform confidence band for $h_{x_*}(t)$.

Value

A list with pointwise, uniform confidence bands and the estimator $\hat{h}_x(t)$ for all possible time points t.

See Also

g_xt, prep_boot

Examples

```
b = 10
x = 3
size_s_grid <- 100
s = pbc2$year
br_s = seq(0, max(s), max(s)/( size_s_grid-1))


c_bands = Conf_bands(pbc2, 'serBilir', event_time_name = 'years',
                     time_name = 'year', event_name = 'status2', x, b)

J = 60
plot(br_s[1:J], c_bands$h_hat[1:J], type = "l", ylim = c(0,1), ylab = 'Hazard', xlab = 'Years')
```

```
lines(br_s[1:J], c_bands$I_p_up[1:J], col = "red")
lines(br_s[1:J], c_bands$I_p_do[1:J], col = "red")
lines(br_s[1:J], c_bands$I_nu[1:J], col = "blue")
lines(br_s[1:J], c_bands$I_nd[1:J], col = "blue")
```

`dataset_split` *Split dataset for K-fold cross validation*

Description

Creates multiple splits of a dataset which is then used in the bandwidth selection with K-fold cross validation.

Usage

```
dataset_split(I, data)
```

Arguments

`data`	A data frame of time dependent data points. Missing values are allowed.
`I`	The number of individuals that should be left out. Optimally, $K = n/I$ should be an integer, where n is the number of individuals.

Details

The function `dataset_split` takes a data frame and transforms it into $K = n/I$ data frames with I individuals missing from each data frame. Let I_j be sets of indices with $\cup_{j=1}^{K} I_j = \{1, ..., n\}$, $I_k \cap I_j = \emptyset$ and $|I_j| = |I_k| = I$ for all $j, k \in \{1, ..., K\}$. Then data frames with $\{1, ..., n\}/I_j$ individuals are created.

Value

A list of data frames with `I` individuals missing in the above way.

See Also

`b_selection`

Examples

```
splitted_dataset = dataset_split(26, pbc2)
```

dij — *D matrix entries, used for the implementation of the local linear kernel*

Description

Calculates the entries of the D matrix in the definition of the local linear kernel

Usage

```
dij(b,x,y, K)
```

Arguments

x	A vector of design points where the kernel will be evaluated.
y	A vector of sample data points.
b	The bandwidth to use (a scalar).
K	The kernel function to use.

Details

Implements the caclulation of all $d \times d$ entries of matrix D, which is part of the definition of the local linear kernel. The actual calculation is performed by

$$d_{jk} = \sum_{i=1}^{n} \int_0^T K_b(x - X_i(s))\{x - X_{ij}(s)\}\{x - X_{ik}(s)\}Z_i(s)ds,$$

Value

scalar value, the result of d_{jk}.

Epan — *Epanechnikov kernel*

Description

Implements the Epanechnikov kernel function

Usage

```
Epan(x)
```

Arguments

x	A vector of design points where the kernel will be evaluated.

Details

Implements the Epanechnikov kernel function

$$K(x) = \frac{3}{4}(1 - x^2) * (|x| < 1)),$$

Value

Scalar, the value of the Epanechnikov kernel at x.

`get_alpha` *Marker-only hazard rate*

Description

Calculates the marker-only hazard rate for time dependent data.

Usage

```
get_alpha(N, Y, b, br_X, K=Epan )
```

Arguments

`N`	A matrix made by `make_N` indicating the occurences of events.
`Y`	A matrix made by `make_Y` indicating the exposure.
`b`	Bandwidth.
`br_X`	Vector of grid points for the marker values X.
`K`	Used kernel function.

Details

The function `get_alpha` implements the marker-only hazard estimator

$$\hat{\alpha}_i(z) = \frac{\sum_{k \neq i} \int_0^T K_{b_1}(z - X_k(s)) \mathrm{d}N_k(s)}{\sum_{k \neq i} \int_0^T K_{b_1}(z - X_k(s)) Z_k(s) \mathrm{d}s},$$

where X is the marker and Z is the exposure. The marker-only hazard is defined as the underlying hazard which is not dependent on time

$$\alpha(X(t), t) = \alpha(X(t))$$

.

Value

A vector of marker-only values for `br_X`.

See Also

h_xt

Examples

```
pbc2_id = to_id(pbc2)
size_s_grid <- size_X_grid <- 100
n = max(as.numeric(pbc2$id))
s = pbc2$year
X = pbc2$serBilir
XX = pbc2_id$serBilir
ss <- pbc2_id$years
delta <- pbc2_id$status2
br_s = seq(0, max(s), max(s)/( size_s_grid-1))
br_X = seq(min(X), max(X), (max(X)-min(X))/( size_X_grid-1))

X_lin = lin_interpolate(br_s, pbc2_id$id, pbc2$id, X, s)

int_X <- findInterval(X_lin, br_X)
int_s = rep(1:length(br_s), n)

N  <- make_N(pbc2, pbc2_id, breaks_X=br_X, breaks_s=br_s, ss, XX, delta)
Y <- make_Y(pbc2, pbc2_id, X_lin, br_X, br_s, size_s_grid,
            size_X_grid, int_s, int_X, event_time = 'years', n)

b = 1.7
alpha<-get_alpha(N, Y, b, br_X, K=Epan )
```

---

get_h_x *Local constant future conditional hazard rate estimator*

Description

Calculates the (indexed) local constant future hazard rate function, conditional on a marker value `x`, across across a set of time values `t`.

Usage

```
get_h_x(data, marker_name, event_time_name, time_name, event_name, x, b)
```

Arguments

`data`	A data frame of time dependent data points. Missing values are allowed.
`marker_name`	The column name of the marker values in the data frame `data`.
`event_time_name`	The column name of the event times in the data frame `data`.
`time_name`	The column name of the times the marker values were observed in the data frame `data`.

event_name	The column name of the events in the data frame data.
x	Numeric value of the last observed marker value.
b	Bandwidth parameter.

Details

The function get_h_x implements the indexed local linear future conditional hazard estimator

$$\hat{h}_x(t) = \frac{\sum_{i=1}^{n} \int_0^T \hat{\alpha}_i(\hat{\theta}^T X_i(t+s)) Z_i(t+s) Z_i(s) K_b(x - \hat{\theta}^T X_i(s)) \mathrm{d}s}{\sum_{i=1}^{n} \int_0^T Z_i(t+s) Z_i(s) K_b(x - \hat{\theta}^T X_i(s)) \mathrm{d}s},$$

across a grid of possible time values t, where, for a positive integer p, $\hat{\theta}^T = (\hat{\theta}_1, \ldots, \hat{\theta}_p)$ is the vector of the estimated indexing parameters, $X_i = (X_{1,i}, \ldots, X_{i,p})$ is a vector of markers for indexing, Z_i is the exposure and $\alpha(z)$ is the marker-only hazard, see get_alpha for more details and $K_b = b^{-1}K(./b)$ is an ordinary kernel function, e.g. the Epanechnikov kernel. For $p = 1$ and $\hat{\theta} = 1$ the above estimator becomes the HQM hazard rate estimator conditional on one covariate,

$$\hat{h}_x(t) = \frac{\sum_{i=1}^{n} \int_0^T \hat{\alpha}_i(X_i(t+s)) Z_i(t+s) Z_i(s) K_b(x - X_i(s)) \mathrm{d}s}{\sum_{i=1}^{n} \int_0^T Z_i(t+s) Z_i(s) K_b(x - X_i(s)) \mathrm{d}s},$$

defined in equation (2) of doi:10.1093/biomet/asaf008.

Value

A vector of $\hat{h}_x(t)$ for a grid of possible time values t.

References

Bagkavos, I., Isakson, R., Mammen, E., Nielsen, J., and Proust–Lima, C. (2025). Biometrika, 112(2), asaf008. doi:10.1093/biomet/asaf008

See Also

get_alpha, h_xt, get_h_xll

Examples

```
library(survival)
b = 10
x = 3
Landmark <- 2
pbcT1 <- pbc2[which(pbc2$year< Landmark  & pbc2$years> Landmark),]
b=0.9

arg1ll<-get_h_xll(pbcT1,'albumin',event_time_name='years',
                  time_name='year',event_name='status2',2,0.9)
arg1lc<-get_h_x(pbcT1,'albumin',event_time_name='years',
                time_name='year',event_name='status2',2,0.9)

#Caclulate the local contant and local linear survival functions
```

```
br_s  = seq(Landmark, 14,  length=99)
sfalb2ll<- make_sf(    (br_s[2]-br_s[1])/4 , arg1ll)
sfalb2lc<- make_sf(    (br_s[2]-br_s[1])/4 , arg1lc)

#For comparison, also calculate the Kaplan-Meier
kma2<- survfit(Surv(years , status2) ~ 1, data = pbcT1)

#Plot the survival functions:
plot(br_s, sfalb2ll,  type="l", col=1, lwd=2, ylab="Survival probability",
     xlab="Marker level")
lines(br_s, sfalb2lc,  lty=2, lwd=2, col=2)
lines(kma2$time, kma2$surv, type="s",  lty=2, lwd=2, col=3)

legend("topright", c(  "Local linear HQM", "Local constant HQM",
        "Kaplan-Meier"), lty=c(1, 2, 2), col=1:3, lwd=2, cex=1.7)


## Not run:
#Example of get_h_x with two indexed covariates:
#First, estimate the joint model for Albumin and Bilirubin combined:
library(JM)
lmeFit <- lme(albumin + serBilir~ year, random = ~ year | id, data = pbc2)
coxFit <- coxph(Surv(years, status2) ~ albumin + serBilir, data = pbc2.id, x = TRUE)
jointFit <- jointModel(lmeFit, coxFit, timeVar = "year", method = "piecewise-PH-aGH")

Landmark <- 1
pbcT1 <- pbc2[which(pbc2$year< Landmark  & pbc2$years> Landmark),]

# Index Albumin and Bilirubin:
t.alb = 3  # slightly low albumin value
t.bil = 1.9  # slightly  high bilirubin value

par.alb  <- 0.0702
par.bil <- 0.0856
X = par.alb * pbcT1$albumin + par.bil *pbcT1$serBilir # X is now the indexed marker
t = par.alb * t.alb + par.bil *t.bil #conditioning value

pbcT1$drug<- X ## store the combine variable in place of 'drug' column which is redundant
## i.e. 'drug' corresponds to indexed bilirubin and albumin

timesS2 <- seq(Landmark,14,by=0.5)
predT1 <- survfitJM(jointFit, newdata = pbcT1, survTimes = timesS2)
nm<-length(predT1$summaries)

mat.out1<-matrix(nrow=length(timesS2), ncol=nm)
for(r in 1:nm)
{
  SurvLand <- predT1$summaries[[r]][,"Mean"][1] #obtain mean predictions
  mat.out1[,r] <- predT1$summaries[[r]][,"Mean"]/SurvLand
}
JM.surv.est<-rowMeans(mat.out1, na.rm=TRUE) #average the resulting JM estimates

# calculate indexed local linear HQM estimator for bilirubin and albumin
```

```
b.alb = 1.5
b.bil = 4
b.hqm   =  par.alb * b.alb + par.bil *b.bil # bandwidth for HQM estimator
arg1<- get_h_x(pbcT1, 'drug', event_time_name = 'years',  time_name = 'year',
                event_name = 'status2', t, b.hqm)
br_s2  = seq(Landmark, 14,  length=99) #grid points for HMQ estimator
hqm.surv.est<- make_sf((br_s2[2]-br_s2[1])/5, arg1) # transform HR to Survival func.

km.land<- survfit(Surv(years , status2) ~ 1, data = pbcT1) #KM estimate

#Plot the survival functions:
plot(br_s2, hqm.surv.est, type="l", ylim=c(0,1), xlim=c(Landmark,14),
           ylab="Survival probability", xlab="years",lwd=2)
lines(timesS2, JM.surv.est, col=2,  lwd=2, lty=2)
lines(km.land$time, km.land$surv, type="s",lty=2, lwd=2, col=3)
legend("bottomleft", c("HQM est.", "Joint Model est.", "Kaplan-Meier"),
      lty=c(1,2,2),  col=1:3, lwd=2, cex=1.7)


## End(Not run)
```

`get_h_xll` *Local linear future conditional hazard rate estimator*

Description

Calculates the (indexed) local linear future hazard rate function, conditional on a marker value `x`, across a set of time values `t`.

Usage

```
get_h_xll(data, marker_name, event_time_name, time_name, event_name, x, b)
```

Arguments

`data`	A data frame of time dependent data points. Missing values are allowed.
`marker_name`	The column name of the marker values in the data frame `data`.
`event_time_name`	The column name of the event times in the data frame `data`.
`time_name`	The column name of the times the marker values were observed in the data frame `data`.
`event_name`	The column name of the events in the data frame `data`.
`x`	Numeric value of the last observed marker value.
`b`	Bandwidth parameter.

Details

The function `get_h_xll` uses a local linear kernel to implement the indexed local linear future conditional hazard estimator

$$\hat{h}_x(t) = \frac{\sum_{i=1}^n \int_0^T \hat{\alpha}_i(\hat{\theta}^T X_i(t+s)) Z_i(t+s) Z_i(s) K_b(x - \hat{\theta}^T X_i(s)) \mathrm{d}s}{\sum_{i=1}^n \int_0^T Z_i(t+s) Z_i(s) K_b(x - \hat{\theta}^T X_i(s)) \mathrm{d}s},$$

across a grid of possible time values t, where, for a positive integer p, $\hat{\theta}^T = (\hat{\theta}_1, \ldots, \hat{\theta}_p)$ is the vector of the estimated indexing parameters, $X_i = (X_{1,i}, \ldots, X_{i,p})$ is a vector of markers for indexing, Z_i is the exposure and $\alpha(z)$ is the marker-only hazard, see `get_alpha` for more details. For $p = 1$ and $\hat{\theta} = 1$ the above estimator becomes the HQM hazard rate estimator conditional on one covariate,

$$\hat{h}_x(t) = \frac{\sum_{i=1}^n \int_0^T \hat{\alpha}_i(X_i(t+s)) Z_i(t+s) Z_i(s) K_b(x - X_i(s)) \mathrm{d}s}{\sum_{i=1}^n \int_0^T Z_i(t+s) Z_i(s) K_b(x - X_i(s)) \mathrm{d}s},$$

defined in equation (2) of doi:10.1093/biomet/asaf008. In the place of $K_b()$, `get_h_xll` uses the kernel

$$K_{x,b}(u) = \frac{K_b(u) - K_b(u) u^T D^{-1} c_1}{c_0 - c_1^T D^{-1} c_1},$$

where $K_b() = b^{-1} K(./b)$ with K being an ordinary kernel, e.g. the Epanechnikov kernel, $c_1 = (c_{11}, \ldots, c_{1d})^T$, $D = (d_{ij})_{(d+1)\times(d+1)}$ with

$$c_0 = \sum_{i=1}^n \int_0^T K_b(x - \hat{\theta}^T X_i(s)) Z_i(s) ds,$$

$$c_{ij} = \sum_{i=1}^n \int_0^T K_b(x - \hat{\theta}^T X_i(s)) \{x - \hat{\theta}^T X_{ij}(s)\} Z_i(s) ds,$$

$$d_{jk} = \sum_{i=1}^n \int_0^T K_b(x - \hat{\theta}^T X_i(s)) \{x - \hat{\theta}^T X_{ij}(s)\} \{x - \hat{\theta}^T X_{ik}(s)\} Z_i(s) ds,$$

see also doi:10.1080/03461238.1998.10413997.

Value

A vector of $\hat{h}_x(t)$ for a grid of possible time values t.

References

Bagkavos, I., Isakson, R., Mammen, E., Nielsen, J., and Proust–Lima, C. (2025). Biometrika, 112(2), asaf008. doi:10.1093/biomet/asaf008

Nielsen (1998), Marker dependent kernel hazard estimation from local linear estimation, Scandinavian Actuarial Journal, pp. 113-124. doi:10.1080/03461238.1998.10413997

See Also

`get_alpha`, `h_xt`, `get_h_x`

Examples

```
library(survival)
library(JM)

# Compare Local constant and local linear estimator for a single covariate,
# use KM for reference.
# Albumin marker, use landmarking
Landmark <- 2
pbcT1 <- pbc2[which(pbc2$year< Landmark  & pbc2$years> Landmark),]
b=0.9

arg1ll<-get_h_xll(pbcT1, 'albumin', event_time_name = 'years', time_name = 'year',
                  event_name = 'status2', 2, 0.9)
arg1lc<-get_h_x(pbcT1, 'albumin', event_time_name = 'years', time_name = 'year',
                event_name = 'status2', 2, 0.9)

#Calculate the local contant and local linear survival functions
br_s  = seq(Landmark, 14,  length=99)
sfalb2ll<- make_sf((br_s[2]-br_s[1])/4 , arg1ll)
sfalb2lc<- make_sf((br_s[2]-br_s[1])/4 , arg1lc)

#For comparison, also calculate the Kaplan-Meier
kma2<- survfit(Surv(years , status2) ~ 1, data = pbcT1)

#Plot the survival functions:
plot(br_s, sfalb2ll,  type="l", col=1, lwd=2, ylab="Survival probability",
                                                    xlab="Marker level")
lines(br_s, sfalb2lc,  lty=2, lwd=2, col=2)
lines(kma2$time, kma2$surv, type="s",  lty=2, lwd=2, col=3)
legend("topright", c(  "Local linear HQM", "Local constant HQM", "Kaplan-Meier"),
        lty=c(1, 2, 2), col=1:3, lwd=2, cex=1.7)


## Not run:
#Example of get_h_xll with a single covariate (no indexing):
#Compare JM, HQM and KM for Bilirubin
b = 10
Landmark <- 1
lmeFit <- lme(serBilir ~ year, random = ~ year | id, data = pbc2)
coxFit <- coxph(Surv(years, status2) ~ serBilir, data = pbc2.id, x = TRUE)

jointFit0 <- jointModel(lmeFit, coxFit, timeVar = "year",
                                method = "piecewise-PH-aGH")
pbcT1 <- pbc2[which(pbc2$year< Landmark  & pbc2$years> Landmark),]

timesS1 <- seq(1,14,by=0.5)
predT1 <- survfitJM(jointFit0, newdata = pbcT1,survTimes = timesS1)
nm<-length(predT1$summaries)

mat.out1<-matrix(nrow=length(timesS1), ncol=nm)
for(r in 1:nm)
{
```

```
  SurvLand <- predT1$summaries[[r]][,"Mean"][1]
  mat.out1[,r] <- predT1$summaries[[r]][,"Mean"]/SurvLand
}
sfit1y<-rowMeans(mat.out1, na.rm=TRUE)

arg1<- get_h_xll(pbcT1, 'serBilir', event_time_name = 'years',
        time_name = 'year', event_name = 'status2', 1, 10)
br_s1  = seq(Landmark, 14,  length=99)
sfbil1<- make_sf((br_s1[2]-br_s1[1])/5.4 , arg1)
kma1<- survfit(Surv(years , status2) ~ 1, data = pbcT1)

plot(br_s1, sfbil1, type="l", ylim=c(0,1), xlim=c(Landmark,14),
                    ylab="Survival probability", xlab="years",lwd=2)
lines(timesS1, sfit1y, col=2,  lwd=2, lty=2)
lines(kma1$time, kma1$surv, type="s", lty=2, lwd=2, col=3 )
legend("bottomleft", c("HQM est.", "Joint Model est.", "Kaplan-Meier"),
                                    lty=c(1,2,2), col=1:3, lwd=2, cex=1.7)

#Example of get_h_xll with two indexed covariates:

#First, estimate the joint model for Albumin and Bilirubin combined:
lmeFit <- lme(albumin + serBilir~ year, random = ~ year | id, data = pbc2)
coxFit <- coxph(Surv(years, status2) ~ albumin + serBilir, data = pbc2.id,
                                                          x = TRUE)
jointFit <- jointModel(lmeFit, coxFit, timeVar = "year",
                                             method = "piecewise-PH-aGH")

Landmark <- 1
pbcT1 <- pbc2[which(pbc2$year< Landmark  & pbc2$years> Landmark),]

# Index Albumin and Bilirubin:
t.alb = 3  # slightly low albumin value
t.bil = 1.9  # slightly  high bilirubin value

par.alb  <- 0.0702
par.bil <- 0.0856
X = par.alb * pbcT1$albumin + par.bil *pbcT1$serBilir # X is now the indexed marker
t = par.alb * t.alb + par.bil *t.bil #conditioning value

pbcT1$drug<- X ## store X in place of 'drug' column which is redundant here
## i.e. 'drug' corresponds to indexed bilirubin and albumin

timesS2 <- seq(Landmark,14,by=0.5)
predT1 <- survfitJM(jointFit, newdata = pbcT1,survTimes = timesS2)
nm<-length(predT1$summaries)

mat.out1<-matrix(nrow=length(timesS2), ncol=nm)
for(r in 1:nm)
{
  SurvLand <- predT1$summaries[[r]][,"Mean"][1] #obtain mean predictions
  mat.out1[,r] <- predT1$summaries[[r]][,"Mean"]/SurvLand
}
JM.surv.est<-rowMeans(mat.out1, na.rm=TRUE) #average the resulting JM estimates
```

```
# calculate indexed local linear HQM estimator for bilirubin and albumin
b.alb = 1.5
b.bil = 4
b.hqm   =  par.alb * b.alb + par.bil *b.bil # bandwidth for HQM estimator
arg1<- get_h_xll(pbcT1, 'drug', event_time_name = 'years',  time_name = 'year',
                                                event_name = 'status2', t, b.hqm)
br_s2  = seq(Landmark, 14,  length=99) #grid points for HMQ estimator
hqm.surv.est<- make_sf((br_s2[2]-br_s2[1])/5  ,arg1) # transform HR to Survival func.

km.land<- survfit(Surv(years , status2) ~ 1, data = pbcT1) #KM estimate

#Plot the survival functions:
plot(br_s2, hqm.surv.est, type="l", ylim=c(0,1), xlim=c(Landmark,14),
    ylab="Survival probability", xlab="years",lwd=2)
lines(timesS2, JM.surv.est, col=2,  lwd=2, lty=2)
lines(km.land$time, km.land$surv, type="s",lty=2, lwd=2, col=3)
legend("bottomleft", c("HQM est.", "Joint Model est.", "Kaplan-Meier"),
        lty=c(1,2,2),  col=1:3, lwd=2, cex=1.7)

## End(Not run)
```

---

g_xt *Computation of a key component for wild bootstrap*

Description

Implements a key part for the wild bootstrap of the hqm estimator.

Usage

```
g_xt(br_X, br_s, size_s_grid, int_X, x, t, b, Yi, Y, n)
```

Arguments

br_X	Marker value grid points that will be used in the evaluatiuon.
br_s	Time value grid points that will be used in the evaluatiuon.
size_s_grid	Size of the time grid.
int_X	Position of the linear interpolated marker values on the marker grid.
x	Numeric value of the last observed marker value.
t	Numeric value of the time the function should be evaluated.
b	Bandwidth.
Yi	A matrix made by make_Yi indicating the exposure.
Y	A matrix made by make_Y indicating the exposure.
n	Number of individuals.

Details

The function implements

$$\hat{g}_{t,x}(z) = \frac{1}{n}\sum_{j=1}^{n}\int_0^{T-t} \hat{E}(X_j(t+s))^{-1} K_b(z, X_j(t+s)) Z_j(t+s) Z_j(s) K_b(x, X_j(s)) ds,$$

for every value z on the marker grid, where $\hat{E}(x) = \frac{1}{n}\sum_{j=1}^{n}\int_0^T K_b(x, X_j(s)) Z_j(s) ds$, Z the exposure and X the marker.

Value

A vector of $\hat{g}_{t,x}(z)$ for all values z on the marker grid.

Examples

```
pbc2_id = to_id(pbc2)
size_s_grid <- size_X_grid <- 100
n = max(as.numeric(pbc2$id))
X = pbc2$serBilir
s = pbc2$year
br_s = seq(0, max(s), max(s)/( size_s_grid-1))
br_X = seq(min(X), max(X), (max(X)-min(X))/( size_X_grid-1))

X_lin = lin_interpolate(br_s, pbc2_id$id, pbc2$id, X, s)

int_X <- findInterval(X_lin, br_X)
int_s = rep(1:length(br_s), n)

Yi<-make_Yi(pbc2, pbc2_id, X_lin, br_X, br_s, size_s_grid, size_X_grid, int_s, int_X,
            'years', n)
Y<-make_Y(pbc2, pbc2_id, X_lin, br_X, br_s,size_s_grid,size_X_grid, int_s,int_X,
            'years', n)

t = 2
x = 2
b = 10

g_xt(br_X, br_s, size_s_grid, int_X, x, t, b, Yi, Y, n)
```

---

h_xt *Local constant future conditional hazard rate estimation at a single time point*

Description

Calculates the (indexed) future conditional hazard rate for a marker value `x` and a time value `t`.

Usage

```
h_xt(br_X, br_s, int_X, size_s_grid, alpha, x,t, b, Yi,n)
```

Arguments

br_X	Vector of grid points for the marker values X.
br_s	Vector of grid points for the time values s.
int_X	Position of the linear interpolated marker values on the marker grid.
size_s_grid	Size of the time grid.
alpha	Marker-hazard obtained from get_alpha.
x	Numeric value of the last observed marker value.
t	Numeric time value.
b	Bandwidth.
Yi	A matrix made by make_Yi indicating the exposure.
n	Number of individuals.

Details

Function h_xt implements the future conditional hazard estimator

$$\hat{h}_x(t) = \frac{\sum_{i=1}^n \int_0^T \hat{\alpha}_i(\hat{\theta}^T X_i(t+s)) Z_i(t+s) Z_i(s) K_b(x - \hat{\theta}^T X_i(s)) \mathrm{d}s}{\sum_{i=1}^n \int_0^T Z_i(t+s) Z_i(s) K_b(x - \hat{\theta}^T X_i(s)) \mathrm{d}s},$$

where, for a positive integer p, $\hat{\theta}^T = (\hat{\theta}_1, \ldots, \hat{\theta}_p)$ is the vector of the estimated indexing parameters, $X_i = (X_{1,i}, \ldots, X_{i,p})$ is a vector of markers for indexing, Z_i is the exposure and $\alpha(z)$ is the marker-only hazard, see get_alpha for more details and $K_b = b^{-1}K(./b)$ is an ordinary kernel function, e.g. the Epanechnikov kernel. For $p = 1$ and $\hat{\theta} = 1$ the above estimator becomes the HQM hazard rate estimator conditional on one covariate,

$$\hat{h}_x(t) = \frac{\sum_{i=1}^n \int_0^T \hat{\alpha}_i(X_i(t+s)) Z_i(t+s) Z_i(s) K_b(x - X_i(s)) \mathrm{d}s}{\sum_{i=1}^n \int_0^T Z_i(t+s) Z_i(s) K_b(x - X_i(s)) \mathrm{d}s},$$

defined in equation (2) of doi:10.1093/biomet/asaf008.

The future conditional hazard is defined as

$$h_{x,T}(t) = P\left(T_i \in (t+T, t+T+dt) | \theta_0^T X_i(T) = x, T_i > t+T\right),$$

where T_i is the survival time and X_i the marker of individual i observed in the time frame $[0, T]$.

Value

A single numeric value of $\hat{h}_x(t)$.

References

Bagkavos, I., Isakson, R., Mammen, E., Nielsen, J., and Proust–Lima, C. (2025). Biometrika, 112(2), asaf008. doi:10.1093/biomet/asaf008

See Also

get_alpha

Examples

```
pbc2_id = to_id(pbc2)
size_s_grid <- size_X_grid <- 100
n = max(as.numeric(pbc2$id))
s = pbc2$year
X = pbc2$serBilir
XX = pbc2_id$serBilir
ss <- pbc2_id$years
delta <- pbc2_id$status2
br_s = seq(0, max(s), max(s)/( size_s_grid-1))
br_X = seq(min(X), max(X), (max(X)-min(X))/( size_X_grid-1))

X_lin = lin_interpolate(br_s, pbc2_id$id, pbc2$id, X, s)

int_X <- findInterval(X_lin, br_X)
int_s = rep(1:length(br_s), n)

N <- make_N(pbc2, pbc2_id, breaks_X=br_X, breaks_s=br_s, ss, XX, delta)
Y <- make_Y(pbc2, pbc2_id, X_lin, br_X, br_s, size_s_grid, size_X_grid, int_s, int_X,
            'years', n)

b = 1.7
alpha<-get_alpha(N, Y, b, br_X, K=Epan )

Yi <- make_Yi(pbc2, pbc2_id, X_lin, br_X, br_s, size_s_grid, size_X_grid, int_s, int_X,
              'years', n)

x = 2
t = 2
h_hat = h_xt(br_X, br_s, int_X, size_s_grid, alpha, x, t, b, Yi, n)
```

---

h_xtll *Local linear future conditional hazard rate estimation at a single time point*

Description

Calculates the (indexed) local linear future conditional hazard rate for a marker value `x` and a time value `t`.

Usage

```
h_xtll(br_X, br_s, int_X, size_s_grid, alpha, x,t, b, Yi,n, Y)
```

Arguments

br_X	Vector of grid points for the marker values X.
br_s	Vector of grid points for the time values s.
int_X	Position of the linear interpolated marker values on the marker grid.
size_s_grid	Size of the time grid.
alpha	Marker-hazard obtained from get_alpha.
x	Numeric value of the last observed marker value.
t	Numeric time value.
b	Bandwidth.
Yi	A matrix made by make_Yi indicating the exposure.
n	Number of individuals.
Y	A matrix made by make_Y indicating the exposure.

Details

Function h_xtll implements the future local linear conditional hazard estimator

$$\hat{h}_x(t) = \frac{\sum_{i=1}^n \int_0^T \hat{\alpha}_i(\hat{\theta}^T X_i(t+s)) Z_i(t+s) Z_i(s) K_b(x - \hat{\theta}^T X_i(s)) \mathrm{d}s}{\sum_{i=1}^n \int_0^T Z_i(t+s) Z_i(s) K_b(x - \hat{\theta}^T X_i(s)) \mathrm{d}s},$$

for every x on the marker grid, where, for a positive integer p, $\hat{\theta}^T = (\hat{\theta}_1, \ldots, \hat{\theta}_p)$ is the vector of the estimated indexing parameters, $X_i = (X_{1,i}, \ldots, X_{i,p})$ is a vector of markers for indexing, Z_i is the exposure and $\alpha(z)$ is the marker-only hazard, see get_alpha for more details. For $p = 1$ and $\hat{\theta} = 1$ the above estimator becomes the HQM hazard rate estimator conditional on a single covariate,

$$\hat{h}_x(t) = \frac{\sum_{i=1}^n \int_0^T \hat{\alpha}_i(X_i(t+s)) Z_i(t+s) Z_i(s) K_b(x - X_i(s)) \mathrm{d}s}{\sum_{i=1}^n \int_0^T Z_i(t+s) Z_i(s) K_b(x - X_i(s)) \mathrm{d}s},$$

defined in equation (2) of doi:10.1093/biomet/asaf008. In the place of $K_b()$, h_xtll uses the kernel

$$K_{x,b}(u) = \frac{K_b(u) - K_b(u) u^T D^{-1} c_1}{c_0 - c_1^T D^{-1} c_1},$$

where $K_b() = b^{-1} K(./b)$ with K being an ordinary kernel, e.g. the Epanechnikov kernel, $c_1 = (c_{11}, \ldots, c_{1d})^T$, $D = (d_{ij})_{(d+1)\times(d+1)}$ with

$$c_0 = \sum_{i=1}^n \int_0^T K_b(x - \hat{\theta}^T X_i(s)) Z_i(s) ds,$$

$$c_{ij} = \sum_{i=1}^n \int_0^T K_b(x - \hat{\theta}^T X_i(s)) \{x - \hat{\theta}^T X_{ij}(s)\} Z_i(s) ds,$$

$$d_{jk} = \sum_{i=1}^n \int_0^T K_b(x - \hat{\theta}^T X_i(s)) \{x - \hat{\theta}^T X_{ij}(s)\} \{x - \hat{\theta}^T X_{ik}(s)\} Z_i(s) ds,$$

see also doi:10.1080/03461238.1998.10413997.

The future conditional hazard is defined as

$$h_{x,T}(t) = P\left(T_i \in (t+T, t+T+dt) | \theta_0^T X_i(T) = x, T_i > t+T\right),$$

where T_i is the survival time and X_i the marker of individual i observed in the time frame $[0, T]$.

Value

A single numeric value of $\hat{h}_x(t)$.

References

Bagkavos, I., Isakson, R., Mammen, E., Nielsen, J., and Proust–Lima, C. (2025). Biometrika, 112(2), asaf008. doi:10.1093/biomet/asaf008

Nielsen (1998), Marker dependent kernel hazard estimation from local linear estimation, Scandinavian Actuarial Journal, pp. 113-124. doi:10.1080/03461238.1998.10413997

See Also

get_alpha, dij

Examples

```
pbc2_id = to_id(pbc2)
size_s_grid <- size_X_grid <- 100
n = max(as.numeric(pbc2$id))
s = pbc2$year
X = pbc2$serBilir
XX = pbc2_id$serBilir
ss <- pbc2_id$years
delta <- pbc2_id$status2
br_s = seq(0, max(s), max(s)/( size_s_grid-1))
br_X = seq(min(X), max(X), (max(X)-min(X))/( size_X_grid-1))

X_lin = lin_interpolate(br_s, pbc2_id$id, pbc2$id, X, s)

int_X <- findInterval(X_lin, br_X)
int_s = rep(1:length(br_s), n)

N <- make_N(pbc2, pbc2_id, breaks_X=br_X, breaks_s=br_s, ss, XX, delta)
Y <- make_Y(pbc2, pbc2_id, X_lin, br_X, br_s, size_s_grid, size_X_grid, int_s, int_X,
            'years', n)

b = 1.7
alpha<-get_alpha(N, Y, b, br_X, K=Epan )

Yi <- make_Yi(pbc2, pbc2_id, X_lin, br_X, br_s, size_s_grid, size_X_grid, int_s, int_X,
              'years', n)

x = 2
t = 2
```

```
h_hat = h_xtll(br_X, br_s, int_X, size_s_grid, alpha, x, t, b, Yi, n, Y)
```

`h_xt_vec` *Hqm estimator on the marker grid*

Description

Computes the (indexed) hqm estimator on the marker grid.

Usage

```
h_xt_vec(br_X, br_s, size_s_grid, alpha, t, b, Yi, int_X, n)
```

Arguments

`br_X`	Marker value grid points that will be used in the evaluatiuon.
`br_s`	Time value grid points that will be used in the evaluatiuon.
`size_s_grid`	Size of the time grid.
`alpha`	Marker-hazard obtained from `get_alpha`.
`t`	Numeric value of the time the function should be evaluated.
`b`	Bandwidth.
`Yi`	A matrix made by `make_Yi` indicating the exposure.
`int_X`	Position of the linear interpolated marker values on the marker grid.
`n`	Number of individuals.

Details

The function implements the future conditional hazard estimator

$$\hat{h}_x(t) = \frac{\sum_{i=1}^{n} \int_0^T \hat{\alpha}_i(\hat{\theta}^T X_i(t+s)) Z_i(t+s) Z_i(s) K_b(x - \hat{\theta}^T X_i(s)) \mathrm{d}s}{\sum_{i=1}^{n} \int_0^T Z_i(t+s) Z_i(s) K_b(x - \hat{\theta}^T X_i(s)) \mathrm{d}s},$$

for every x on the marker grid, where, for a positive integer p, $\hat{\theta}^T = (\hat{\theta}_1, \ldots, \hat{\theta}_p)$ is the vector of the estimated indexing parameters, $X_i = (X_{1,i}, \ldots, X_{i,p})$ is a vector of markers for indexing, Z_i is the exposure and $\alpha(z)$ is the marker-only hazard, see `get_alpha` for more details and $K_b = b^{-1}K(./b)$ is an ordinary kernel function, e.g. the Epanechnikov kernel. For $p = 1$ and $\hat{\theta} = 1$ the above estimator becomes the HQM hazard rate estimator conditional on one covariate,

$$\hat{h}_x(t) = \frac{\sum_{i=1}^{n} \int_0^T \hat{\alpha}_i(X_i(t+s)) Z_i(t+s) Z_i(s) K_b(x - X_i(s)) \mathrm{d}s}{\sum_{i=1}^{n} \int_0^T Z_i(t+s) Z_i(s) K_b(x - X_i(s)) \mathrm{d}s},$$

defined in equation (2) of doi:10.1093/biomet/asaf008.

Value

A vector of $\hat{h}_x(t)$ for all values x on the marker grid.

References

Bagkavos, I., Isakson, R., Mammen, E., Nielsen, J., and Proust–Lima, C. (2025). Biometrika, 112(2), asaf008. doi:10.1093/biomet/asaf008

Examples

```
# Longitudinal data example

pbc2_id = to_id(pbc2)
size_s_grid <- size_X_grid <- 100
n = max(as.numeric(pbc2$id))
s = pbc2$year
X = pbc2$serBilir
XX = pbc2_id$serBilir
ss <- pbc2_id$years
delta <- pbc2_id$status2
br_s = seq(0, max(s), max(s)/( size_s_grid-1))
br_X = seq(min(X), max(X), (max(X)-min(X))/( size_X_grid-1))

X_lin = lin_interpolate(br_s, pbc2_id$id, pbc2$id, X, s)

int_X <- findInterval(X_lin, br_X)
int_s = rep(1:length(br_s), n)

N <- make_N(pbc2, pbc2_id, br_X, br_s, ss, XX, delta)
Y <- make_Y(pbc2, pbc2_id, X_lin, br_X, br_s,
            size_s_grid, size_X_grid, int_s, int_X, event_time = 'years', n)

b = 1.7
alpha<-get_alpha(N, Y, b, br_X, K=Epan )

Yi <- make_Yi(pbc2, pbc2_id, X_lin, br_X, br_s,
              size_s_grid, size_X_grid, int_s, int_X, event_time = 'years', n)

t = 2

h_xt_vec(br_X, br_s, size_s_grid, alpha, t, b, Yi, int_X, n)

# Time-invariant data example:
# pbc2 dataset, single event per individual version:
# 312 observations, most recent event per individual.
# Use landmarking to produce comparable curve with KM.
library(survival)
Landmark <- 3 #set the landmark to 3 years
pbc2.use<- to_id(pbc2) # keep only the most recent row per patient
pbcT1 <- pbc2.use[which(pbc2.use$year< Landmark  & pbc2.use$years> Landmark),]

timesS2 <- seq(Landmark,14,by=0.5)
b=0.9
arg1<- get_h_x(pbcT1, 'albumin', event_time_name = 'years',  time_name = 'year',
                                                    event_name = 'status2', 2, b)
br_s2  = seq(Landmark, 14,  length=99)
```

```
sfalb2<- make_sf(    (br_s2[2]-br_s2[1])/1.35 , arg1)
kma2<- survfit(Surv(years , status2) ~ 1, data = pbcT1)

#Plot the survival functions:
plot(br_s2, sfalb2, type="l", ylim=c(0,1), xlim=c(Landmark,14), ylab="Survival probability",
        xlab="years",lwd=2, main="HQM and KM survival functions, conditional on albumin=2,
            for the time-invariant pbc dataset")
lines(kma2$time, kma2$surv, type="s",lty=2, lwd=2, col=2)
legend("bottomleft", c("HQM est.", "Kaplan-Meier"), lty=c(1,2), col=1:2, lwd=2, cex=1.7)


############## Indexed HR estimator example 1: ###############
### The example is based on indexing two covariates: albumin and bilirubin

index.use<-84
x.grid<-br_s[1:index.use] #non need to plot the last year

b.alb = 1.5 # results from CV rule of Bagkavos et al. (2025), i.e.:
 # b_list = seq(1.5, 3, 0.05)
 # b_scores_alb = b_selection(pbc2, 'albumin', 'years', 'year', 'status2', I, b_list)
 # b.alb =  b_scores_alb[[2]][which.min(b_scores_alb[[1]])]

b.bil = 4 # results from CV rule of Bagkavos et al. (2025) :
  # b_list.bil = seq(3, 4, 0.05)
 # b_scores_bil = b_selection(pbc2, 'serBilir', 'years', 'year', 'status2', I, b_list.bil)
  # b.bil =  b_scores_alb[[2]][which.min(b_scores_alb[[1]])] - result = 4

t.alb = 1 # refers to zero mean variables
          #corresponds to approximately normal albumin level
t.bil = 1.9 # refers to zero mean variable - corresponds to
          #elevated bilirubin levelm approximately 7mg/dl

par.alb  <- 0.0702 # results from the indexing process
par.bil <- 0.0856 # results from the indexing process

# First, mean adjust the albumin covariate
Xalb = pbc2$albumin - mean(pbc2$albumin)
XXalb = pbc2_id$albumin - mean(pbc2_id$albumin)

br_Xalb = seq(min(Xalb), max(Xalb), (max(Xalb)-min(Xalb))/( size_X_grid-1))
X_linalb = lin_interpolate(br_s, pbc2_id$id, pbc2$id, Xalb, s)
int_Xalb <- findInterval(X_linalb, br_Xalb)
N.alb <- make_N(pbc2, pbc2_id, br_Xalb, br_s, ss, XXalb, delta)
Y.alb <- make_Y(pbc2, pbc2_id, X_linalb, br_Xalb, br_s,
            size_s_grid, size_X_grid, int_s, int_Xalb, event_time = 'years', n)

alpha.alb<-get_alpha(N.alb, Y.alb, b.alb, br_Xalb, K=Epan )
Yi.alb <- make_Yi(pbc2, pbc2_id, X_linalb, br_Xalb, br_s, size_s_grid, size_X_grid,
                                          int_s, int_Xalb, event_time = 'years', n)

#calculate HQM estimator for original albumin covariate:
arg.alb<- h_xt_vec(br_Xalb, br_s, size_s_grid, alpha.alb, t.alb, b.alb, Yi.alb,
                                                                  int_Xalb, n)
```

```
y.grid.alb<-arg.alb[1:index.use]

# Now, mean adjust the bilirubon covariate:
Xbil = pbc2$serBilir - mean(pbc2$serBilir)
XXbil = pbc2_id$serBilir - mean(pbc2_id$serBilir)

br_Xbil = seq(min(Xbil), max(Xbil), (max(Xbil)-min(Xbil))/( size_X_grid-1))
X_linbil = lin_interpolate(br_s, pbc2_id$id, pbc2$id, Xbil, s)
int_Xbil <- findInterval(X_linbil, br_Xbil)
N.bil <- make_N(pbc2, pbc2_id, br_Xbil, br_s, ss, XXbil, delta)
Y.bil <- make_Y(pbc2, pbc2_id, X_linbil, br_Xbil, br_s,
            size_s_grid, size_X_grid, int_s, int_Xbil, event_time = 'years', n)

alpha.bil<-get_alpha(N.bil, Y.bil, b.bil, br_Xbil, K=Epan )
Yibil <- make_Yi(pbc2, pbc2_id, X_linbil, br_Xbil, br_s, size_s_grid, size_X_grid,
                                      int_s, int_Xbil, event_time = 'years', n)

#calculate HQM estimator for original bilirubin covariate:
arg1.bil<- h_xt_vec(br_Xbil, br_s, size_s_grid, alpha.bil, t.bil, b.bil, Yibil,
                                                                    int_Xbil, n)
y.grid.bil<-arg1.bil[1:index.use]

# calculate the indexed variables:
X = par.alb * Xalb +  par.bil *Xbil
XX = par.alb * XXalb + par.bil *XXbil
b = 0.4 # results from CV rule in Bagkavos et al (2025)
t = par.alb * t.alb + par.bil *t.bil

br_X = seq(min(X), max(X), (max(X)-min(X))/( size_X_grid-1))

X_lin = lin_interpolate(br_s, pbc2_id$id, pbc2$id, X, s)
int_X <- findInterval(X_lin, br_X)
N <- make_N(pbc2, pbc2_id, breaks_X=br_X, breaks_s=br_s, ss, XX, delta)
Y <- make_Y(pbc2, pbc2_id, X_lin, br_X, br_s, size_s_grid, size_X_grid, int_s,
                                                            int_X, 'years', n)
alpha<-get_alpha(N, Y, b, br_X, K=Epan )
Yi <- make_Yi(pbc2, pbc2_id, X_lin, br_X, br_s, size_s_grid, size_X_grid,
                                                    int_s, int_X,'years', n)

# Now calculate indexed HR estimator:
arg2<- h_xt_vec(br_X, br_s, size_s_grid, alpha, t, b, Yi, int_X, n)
y.grid2<-arg2[1:index.use]


############### Plot the results:
plot(x.grid, y.grid2, type="l", ylim=c(0,2), xlab="Years", ylab="Hazard rate", lwd=2 )
lines(x.grid, y.grid.bil , lty=2, col=4, lwd=2)
lines(x.grid, y.grid.alb, lty=2, col=2, lwd=2)

legend("topleft", lty=c(1, 2, 2), lwd=c(2,2,2),
       c("Indexed (Bilirubin and Albumin) HQM hazard rate est.",
         "HQM hazard rate est. conditional on Albumin=2mg/dl",
```

```
         "HQM hazard rate est. conditional on Bilirubin=7mg/dl" ),
       col=c(1,2,4), cex=2, bty="n")

############# Indexed HR estimator example 2: ##############
### The example is based on indexing two covariates:  bilirubin and prothrombin time

index.use<-84
x.grid<-br_s[1:index.use]

b.pro = 3
b.bil = 4

t.pro = 1.5 # refers to zero mean variables - slightly high
t.bil = 1.9 # refers to zero mean variable - high

par.pro  <- 0.349 #0.5 #0.149
par.bil <- 0.0776 #0.10

############ prothrombin time #############
ind1 <- which( is.na( pbc2$prothrombin ) )
Xpro1 =   pbc2$prothrombin
Xpro1[ind1]<-mean(pbc2$prothrombin, na.rm=TRUE)
Xpro1 <- Xpro1 - mean(Xpro1)

ind2 <- which( is.na( pbc2_id$prothrombin ) )
XXpro1 =   pbc2_id$prothrombin
XXpro1[ind2]<-mean(pbc2_id$prothrombin, na.rm=TRUE)
XXpro1 <- XXpro1 - mean(XXpro1)

br_Xbil = seq(min(Xpro1), max(Xpro1), (max(Xpro1)-min(Xpro1))/( size_X_grid-1))
X_linbil = lin_interpolate(br_s, pbc2_id$id, pbc2$id, Xpro1, s)
int_Xbil <- findInterval(X_linbil, br_Xbil)
N.bil <- make_N(pbc2, pbc2_id, br_Xbil, br_s, ss, XXpro1, delta)
Y.bil <- make_Y(pbc2, pbc2_id, X_linbil, br_Xbil, br_s,
                size_s_grid, size_X_grid, int_s, int_Xbil, event_time = 'years', n)

alpha.bil<-get_alpha(N.bil, Y.bil, b.pro, br_Xbil, K=Epan )
Yibil <- make_Yi(pbc2, pbc2_id, X_linbil, br_Xbil, br_s, size_s_grid, size_X_grid, int_s,
                                                    int_Xbil, event_time = 'years', n)
arg1.pro<- h_xt_vec(br_Xbil, br_s, size_s_grid, alpha.bil, t.pro, b.pro, Yibil, int_Xbil, n)
y.grid.pro<-arg1.pro[1:index.use]


############ Bilirubin #############
Xbil = pbc2$serBilir - mean(pbc2$serBilir)
XXbil = pbc2_id$serBilir - mean(pbc2_id$serBilir)

br_Xbil = seq(min(Xbil), max(Xbil), (max(Xbil)-min(Xbil))/( size_X_grid-1))
X_linbil = lin_interpolate(br_s, pbc2_id$id, pbc2$id, Xbil, s)
int_Xbil <- findInterval(X_linbil, br_Xbil)
N.bil <- make_N(pbc2, pbc2_id, br_Xbil, br_s, ss, XXbil, delta)
Y.bil <- make_Y(pbc2, pbc2_id, X_linbil, br_Xbil, br_s,
            size_s_grid, size_X_grid, int_s, int_Xbil, event_time = 'years', n)
```

```
alpha.bil<-get_alpha(N.bil, Y.bil, b.bil, br_Xbil, K=Epan )
Yibil <- make_Yi(pbc2, pbc2_id, X_linbil, br_Xbil, br_s, size_s_grid, size_X_grid, int_s,
                                                    int_Xbil, event_time = 'years', n)
arg1.bil<- h_xt_vec(br_Xbil, br_s, size_s_grid, alpha.bil, t.bil, b.bil, Yibil, int_Xbil, n)
y.grid.bil<-arg1.bil[1:index.use]

######### Index prothrombin time and bilirubin
X = par.pro * Xpro1 +  par.bil *Xbil
XX = par.pro * XXpro1 + par.bil *XXbil

b = 1.5
t = par.pro * t.pro + par.bil *t.bil

br_X = seq(min(X), max(X), (max(X)-min(X))/( size_X_grid-1))

X_lin = lin_interpolate(br_s, pbc2_id$id, pbc2$id, X, s)
int_X <- findInterval(X_lin, br_X)
N <- make_N(pbc2, pbc2_id, breaks_X=br_X, breaks_s=br_s, ss, XX, delta)
Y <- make_Y(pbc2, pbc2_id, X_lin, br_X, br_s, size_s_grid, size_X_grid, int_s, int_X,
                                                                        'years', n)
alpha<-get_alpha(N, Y, b, br_X, K=Epan )
Yi <- make_Yi(pbc2, pbc2_id, X_lin, br_X, br_s, size_s_grid, size_X_grid, int_s, int_X,
                                                                        'years', n)
arg2<- h_xt_vec(br_X, br_s, size_s_grid, alpha, t, b, Yi, int_X, n)
y.grid2<-arg2[1:index.use]



###############Plot the results
plot(x.grid, y.grid2, type="l", ylim=c(0,2), xlab="Years", ylab="Hazard rate", lwd=2 )
lines(x.grid, y.grid.bil , lty=2, col=4, lwd=2)

lines(x.grid, y.grid.pro, lty=2, col=2, lwd=2)
legend("topleft", lty=c(1, 2, 2), lwd=c(2,2,2),
       c("Indexed (Bilirubin and PT) HQM hazard rate est.",
        "HQM hazard rate est. conditional on PT=15.5s",
        "HQM hazard rate est. conditional on Bilirubin=7mg/dl" ),
        col=c(1,2,4), cex=2, bty="n")
```

---

index_optim *Indexing parameter objective function*

Description

Objective used for optimizing indexing parameters via optim.

Usage

```
index_optim(in.par, data, data.id, X1, XX1, X2, XX2, event_time_name = 'years',
            time_name = 'year', event_name = 'status2', b, t, true.haz)
```

Arguments

in.par	Numeric vector length 2.
data	Data frame.
data.id	Id-level data frame.
X1, XX1, X2, XX2	Numeric vectors.
event_time_name	Name of event time column.
time_name	Name of observation time column.
event_name	Name of event indicator column.
b	Bandwidth.
t	Evaluation point or grid.
true.haz	Numeric vector of true hazard values for CV.

Value

Scalar cross-validation score.

Examples

```
marker_name1 <- 'albumin'
marker_name2 <-  'serBilir'
event_time_name <- 'years'
time_name <- 'year'
event_name <- 'status2'
id<-'id'


par.x1  <- 0.0702 #0.149
par.x2 <- 0.0856 #0.10
t.x1 = 0 # refers to zero mean variables - slightly high
t.x2 = 1.9 # refers to zero mean variable - high
b = 0.42
t = par.x1 * t.x1 + par.x2 *t.x2


# first simulate true HR function:
xin <- pbc2[,c(id, marker_name1, marker_name2, event_time_name, time_name, event_name)]
n <- length(xin$id)
nn<-max(  as.double(xin[,'id']) )
###################  Create bootstrap samples by group ####################
set.seed(1)
B<-100
```

```
Boot.samples<-list()
for(j in 1:B)
{
  i.use<-c()
  id.use<-c()
  index.nn <- sample (nn, replace = TRUE)
  for(l in 1:nn)
  {
    i.use2<-which(xin[,id]==index.nn[l])
    i.use<-c(i.use, i.use2)
    id.use2<-rep(index.nn[l], times=length(i.use2))
    id.use<-c(id.use, id.use2)
  }
  xin.i<-xin[i.use,]
  xin.i<-xin[i.use,]
  Boot.samples[[j]]<- xin.i[order(xin.i$id),] #xin[i.use,]
}
true.hazard<- Sim.True.Hazard(Boot.samples, id='id', marker_name1=marker_name1,
              marker_name2= marker_name2, event_time_name = event_time_name,
              time_name = time_name,  event_name = event_name,
              in.par = c(par.x1,  par.x2), b)
#############################################################################

# Then run the optimization for the indexing parameters:

data.use<-Boot.samples[[1]]
data.use.id<-to_id(data.use)
data.use.id<-data.use.id[complete.cases(data.use.id), ]
X1=data.use[,marker_name1] -mean(data.use[, marker_name1])
XX1=data.use.id[,marker_name1] -mean(data.use.id[, marker_name1])
X2=data.use[,marker_name2]  -mean(data.use[, marker_name2])
XX2=data.use.id[,marker_name2] -mean(data.use.id[, marker_name2])

res<- optim(par=c(par.x1, par.x2), fn=index_optim,  data=data.use, data.id=data.use.id,
            X1=X1, XX1=XX1,X2 = X2, XX2 =XX2, event_time_name = event_time_name,
            time_name = time_name, event_name = event_name, b=b, t=t,
            true.haz=true.hazard,  method="Nelder-Mead")
```

`Kernels` *Classical (unmodified) kernel and related functionals*

Description

Implements the classical kernel function and related functionals

Usage

```
K_b(b,x,y, K)
```

```
xK_b(b,x,y, K)
K_b_mat(b,x,y, K)
```

Arguments

x	A vector of design points where the kernel will be evaluated.
y	A vector of sample data points.
b	The bandwidth to use (a scalar).
K	The kernel function to use.

Details

The function K_b implements the classical kernel function calculation

$$h^{-1}K\left(\frac{x-y}{h}\right)$$

for scalars x and y while xK_b implements the functional

$$h^{-1}K\left(\frac{x-y}{h}\right)(x-y)$$

again for for scalars x and y. The function K_b_mat is the vectorized version of K_b. It uses as inputs the vectors $(X_1,\ldots,X_n)$ and $(Y_1,\ldots,Y_n)$ and returns a $n \times n$ matrix with entries

$$h^{-1}K\left(\frac{X_i-Y_j}{h}\right)$$

Value

Scalar values for K_b and xK_b and matrix outputs for K_b_mat.

---

lin_interpolate *Linear interpolation*

Description

Implements a linear interpolation between observered marker values.

Usage

```
lin_interpolate(t, i, data_id, data_marker, data_time)
```

Arguments

t	A vector of time values where the function should be evaluated.
i	A vector of ids of individuals for whom the marker values should be interpolated.
data_id	The vector of ids from a data frame of time dependent variables.
data_marker	The vector of marker values from a data frame of time dependent variables.
data_time	The vector of time values from a data frame of time dependent variables.

Details

Given time points $t_1, ..., t_K$ and marker values $m_1, ..., m_J$ at different time points $t_1^m, ..., t_J^m$, the function calculates a linear interpolation f with $f(t_i^m) = m_i$ at the time points $t_1, ..., t_K$ for all indicated individuals. Returned are then $(f(t_1), ..., f(t_K))$. Note that the first value is always observed at time point 0 and the function f is extrapolated constantly after the last observed marker value.

Value

A matrix with columns $(f(t_1), ..., f(t_K))$ as described above for every individual in the vector `i`.

Examples

```
size_s_grid <- 100
X = pbc2$serBilir
s = pbc2$year
br_s = seq(0, max(s), max(s)/( size_s_grid-1))
pbc2_id = to_id(pbc2)

X_lin = lin_interpolate(br_s, pbc2_id$id, pbc2$id, X, s)
```

`lin_interpolate_plus` *Interpolate marker values onto a time grid*

Description

Interpolate marker values at grid times for each subject id, slightly adjusted version of `lin_interpolate`.

Usage

```
lin_interpolate_plus(t, i, data_id, data_marker, data_time)
```

Arguments

`t`	Numeric vector of target times.
`i`	Vector of ids for which to compute interpolation.
`data_id`	Vector of ids aligned with observations.
`data_marker`	Observed marker values.
`data_time`	Observation times aligned with `data_marker`.

Value

Matrix of interpolated values.

Examples

```
size_s_grid <- 100
X = pbc2$serBilir
s = pbc2$year
br_s = seq(0, max(s), max(s)/( size_s_grid-1))
pbc2_id = to_id(pbc2)

X_lin = lin_interpolate_plus(br_s, pbc2_id$id, pbc2$id, X, s)
```

`llK_b` *Local linear kernel*

Description

Implements the local linear kernel function.

Usage

```
llK_b(b,x,y, K)
```

Arguments

`x`	A vector of design points where the kernel will be evaluated.
`y`	A vector of sample data points.
`b`	The bandwidth to use (a scalar).
`K`	The kernel function to use.

Details

Implements the local linear kernel

$$K_{x,b}(u) = \frac{K_b(u) - K_b(u)u^T D^{-1} c_1}{c_0 - c_1^T D^{-1} c_1},$$

where $c_1 = (c_{11}, \ldots, c_{1d})^T$, $D = (d_{ij})_{(d+1)\times(d+1)}$ with

$$c_0 = \sum_{i=1}^{n} \int_0^T K_b(x - X_i(s)) Z_i(s) ds,$$

$$c_{ij} = \sum_{i=1}^{n} \int_0^T K_b(x - X_i(s)) \{x - X_{ij}(s)\} Z_i(s) ds,$$

$$d_{jk} = \sum_{i=1}^{n} \int_0^T K_b(x - X_i(s)) \{x - X_{ij}(s)\} \{x - X_{ik}(s)\} Z_i(s) ds,$$

see also doi:10.1080/03461238.1998.10413997.

Value

Matrix output with entries the values of the kernel function at each point.

References

Nielsen (1998), Marker dependent kernel hazard estimation from local linear estimation, Scandinavian Actuarial Journal, pp. 113-124. doi:10.1080/03461238.1998.10413997

`llweights` *Local linear weight functions*

Description

Implements the weights to be used in the local linear HQM estimator.

Usage

```
sn.0(xin, xout, h, kfun)
sn.1(xin, xout, h, kfun)
sn.2(xin, xout, h, kfun)
```

Arguments

`xin`	Sample values.
`xout`	Grid points where the estimator will be evaluated.
`h`	Bandwidth parameter.
`kfun`	Kernel function.

Details

The function implements the local linear weights in the definition of the estimator $\hat{h}_x(t)$, see also `h_xt`

Value

A vector of $s_n(x)$ for all values x on the marker grid.

`make_N, make_Ni, make_Y, make_Yi` *Occurance and Exposure on grids*

Description

Auxiliary functions that help automate the process of calculating integrals with occurances or exposure processes.

Usage

```
make_N(data, data.id, breaks_X, breaks_s, ss, XX, delta)
make_Ni(breaks_s, size_s_grid, ss, delta, n)
make_Y(data, data.id, X_lin, breaks_X, breaks_s,
        size_s_grid, size_X_grid, int_s, int_X, event_time = 'years', n)
make_Yi(data, data.id, X_lin, breaks_X, breaks_s,
        size_s_grid, size_X_grid, int_s,int_X, event_time = 'years', n)
```

Arguments

`data`	A data frame of time dependent data points. Missing values are allowed.
`data.id`	An id data frame obtained from `to_id`.
`breaks_X`	Marker value grid points where the function will be evaluated.
`breaks_s`	Time value grid points where the function will be evaluated.
`ss`	Vector with event times.
`XX`	Vector of last observed marker values.
`delta`	0-1 vector of whether events happened.
`size_s_grid`	Size of the time grid.
`size_X_grid`	Size of the marker grid.
`n`	Number of individuals.
`X_lin`	Linear interpolation of observed marker values evaluated on the marker grid.
`int_s`	Position of the observed time values on the time grid.
`int_X`	Position of the linear interpolated marker values on the marker grid.
`event_time`	String of the column name with the event times.

Details

Implements matrices for the computation of integrals with occurences and exposures of the form

$$\int f(s)Z(s)Z(s+t)ds, \int f(s)Z(s)ds, \int f(s)dN(s).$$

where N is a 0-1 counting process, Z the exposure and f an arbitrary function.

Value

The functions make_N and make_Y return a matrix on the time grid and marker grid for occurence and exposure, respectively, while make_Ni and make_Yi return a matrix on the time grid for evey individual again for occurence and exposure, respectively.

See Also

h_xt, g_xt, get_alpha

Examples

```
pbc2_id = to_id(pbc2)
size_s_grid <- size_X_grid <- 100
n = max(as.numeric(pbc2$id))
s = pbc2$year
X = pbc2$serBilir
XX = pbc2_id$serBilir
ss <- pbc2_id$years
delta <- pbc2_id$status2
br_s = seq(0, max(s), max(s)/( size_s_grid-1))
br_X = seq(min(X), max(X), (max(X)-min(X))/( size_X_grid-1))

X_lin = lin_interpolate(br_s, pbc2_id$id, pbc2$id, X, s)

int_X <- findInterval(X_lin, br_X)
int_s = rep(1:length(br_s), n)

N <- make_N(pbc2, pbc2_id, br_X, br_s, ss, XX, delta)
Y <- make_Y(pbc2, pbc2_id, X_lin, br_X, br_s,
            size_s_grid, size_X_grid, int_s, int_X, event_time = 'years', n)
Yi <- make_Yi(pbc2, pbc2_id, X_lin, br_X, br_s,
              size_s_grid, size_X_grid, int_s, int_X, event_time = 'years', n)
Ni  <- make_Ni(br_s, size_s_grid, ss, delta, n)
```

---

make_sf *Survival function from a hazard*

Description

Creates a survival function from a hazard rate which was calculated on a grid.

Usage

```
make_sf(step_size_s_grid, haz)
```

Arguments

step_size_s_grid	Numeric value indicating the distance between two grid continuous grid points.
haz	Vector of hazard values. Hazard rate must have been calculated on a time grid.

Details

The function make_sf calculates the survival function

$$S(t) = \exp(-\int_0^t h(t)dt),$$

where h is the hazard rate. Here, a discritisation via an equidistant grid $\{t_i\}$ on $[0, t]$ is used to calculate the integral and it is assumed that h has been calculated for exactly these time points t_i.

Value

A vector of values $S(t_i)$.

Examples

```
make_sf(0.1, rep(0.1,10))
```

---

pbc2 *Mayo Clinic Primary Biliary Cirrhosis Data*

Description

Followup of 312 randomised patients with primary biliary cirrhosis, a rare autoimmune liver disease, at Mayo Clinic.

Usage

```
pbc2
```

Format

A data frame with 1945 observations on the following 20 variables.

`id` patients identifier; in total there are 312 patients.

`years` number of years between registration and the earlier of death, transplantion, or study analysis time.

`status` a factor with levels `alive`, `transplanted` and `dead`.

`drug` a factor with levels `placebo` and `D-penicil`.

`age` at registration in years.

`sex` a factor with levels `male` and `female`.

`year` number of years between enrollment and this visit date, remaining values on the line of data refer to this visit.

`ascites` a factor with levels `No` and `Yes`.

`hepatomegaly` a factor with levels `No` and `Yes`.

`spiders` a factor with levels `No` and `Yes`.

`edema` a factor with levels `No edema` (i.e., no edema and no diuretic therapy for edema), `edema no diuretics` (i.e., edema present without diuretics, or edema resolved by diuretics), and `edema despite diuretics` (i.e., edema despite diuretic therapy).

`serBilir` serum bilirubin in mg/dl.

`serChol` serum cholesterol in mg/dl.

`albumin` albumin in gm/dl.

`alkaline` alkaline phosphatase in U/liter.

`SGOT` SGOT in U/ml.

`platelets` platelets per cubic ml / 1000.

`prothrombin` prothrombin time in seconds.

`histologic` histologic stage of disease.

`status2` a numeric vector with the value 1 denoting if the patient was dead, and 0 if the patient was alive or transplanted.

References

Fleming, T. and Harrington, D. (1991) *Counting Processes and Survival Analysis*. Wiley, New York.

Therneau, T. and Grambsch, P. (2000) *Modeling Survival Data: Extending the Cox Model*. Springer-Verlag, New York.

Examples

```
summary(pbc2)
```

`Pivot.Index.CIs` *Compute Pivot Pointwise Confidence Intervals for the Indexed Hazard Rate Estimate*

Description

Computes bootstrap pivot and symmetric bootstrap-pivot confidence intervals for the hazard, log-hazard, and back-transformed (log-scale) hazard rate functions, using the indexed hazard estimator.

Usage

```
Pivot.Index.CIs(B, n.est.points, Mat.boot.haz.rate, time.grid, hqm.est, a.sig)
```

Arguments

B	Integer. Number of bootsrap replications
n.est.points	Integer. Number of estimation points at which the indexed hazard estimates and confidence intervals are evaluated.
Mat.boot.haz.rate	A matrix of bootstrap estimated hazard rates with dimensions n.est.points × B, where each column corresponds to one bootstrap replicate.
time.grid	Numeric vector of length n.est.points: the grid points at which the indexed hazard estimates and confidence intervals are calculated.
hqm.est	Indexed hazard estimator, calculated at the grid points time.grid and using the original sample.
a.sig	The significance level (e.g., 0.05) which will be used in computing the confidence intervals.

Details

This function computes several forms of pivot confidence intervals for indexed hazard rate estimates. Let $\hat{\sigma}$ be the sample standard deviation of the bootstrap estimators $\hat{h}_x^{(j)}(t), j = 1, \ldots, B$ and let $k^p_{\alpha/2}, k^p_{1-\alpha/2}$ and $\bar{k}^p_{1-\alpha}$ be the $\alpha/2, 1-\alpha/2$ and $1-\alpha$ quantiles respectively of

$$\frac{\hat{h}_x^{(j)}(t) - \hat{h}_x(t)}{\hat{\sigma}},\ j = 1, \ldots, B.$$

Then, the pivot bootstrap confidence interval (CI) for $\hat{h}_x(t)$ is given by

$$\left(\hat{h}_x(t) - \hat{\sigma} k^p_{1-\alpha/2}, \hat{h}_x(t) - \hat{\sigma} k^p_{\alpha/2}\right).$$

The symmetric pivot bootstrap CI for $\hat{h}_x(t)$ is

$$\left(\hat{h}_x(t) - \hat{\sigma} \bar{k}^p_{1-\alpha}, \hat{h}_x(t) + \hat{\sigma} \bar{k}^p_{1-\alpha}\right).$$

For the confidence intervals for the logarithm of the hazard rate function, first let

$$L_x(t) = \log\left\{h_x(t)\right\}, \hat{L}_x(t) = \log\left\{\hat{h}_x(t)\right\}, \hat{L}_x^{(j)}(t) = \log\left\{\hat{h}_x^{(j)}(t)\right\},$$

and $k^{L,p}_{\alpha/2}, k^{L,p}_{1-\alpha/2}$ and $\bar{k}^{L,p}_{1-\alpha}$ be respectively the $\alpha/2, 1-\alpha/2$ and $1-\alpha$ quantile of

$$\frac{|\hat{L}_x^{(j)}(t) - \hat{L}_x(t)|}{\hat{\sigma}}, j = 1, \ldots, B.$$

For the log hazard function $L_x(t)$ a pivotal bootstrap confidence interval is

$$\left(\hat{L}_x(t) - \hat{\sigma} k^{L,p}_{1-\alpha/2}, \hat{L}_x(t) - \hat{\sigma} k^{L,p}_{\alpha/2}\right).$$

A symmetric pivotal bootstrap CI for the log hazard is

$$\left(\hat{L}_x(t) - \hat{\sigma}\bar{k}^{L,p}_{1-\alpha}, \hat{L}_x(t) + \hat{\sigma}\bar{k}^{L,p}_{1-\alpha}\right).$$

These last two confidence intervals can be transformed back to yield confidence intervals for the hazard rate function $h_x(t)$. These are:

$$\left(\hat{h}_x(t)e^{-\hat{\sigma}k^{L,p}_{1-\alpha/2}}, \hat{h}_x(t)e^{-\hat{\sigma}k^{L,p}_{\alpha/2}}\right),$$

and

$$\left(\hat{h}_x(t)e^{-\hat{\sigma}\bar{k}^{L,p}_{1-\alpha}}, \hat{h}_x(t)e^{\hat{\sigma}\bar{k}^{L,p}_{1-\alpha}}\right)$$

respectively.

Note: The bootstrap matrix `Mat.boot.haz.rate` is assumed to contain estimates produced using the same time grid as `time.grid` and the same estimator used to generate `hqm.est`.

Value

A data frame with the following columns:

`time`	The evaluation grid points.
`est`	Indexed hazard rate estimator `hqm.est`.
`downci`, `upci`	Lower and upper endpoints of basic pivot CIs.
`docisym`, `upcisym`	Lower and upper endpoints of symmetric pivot CIs.
`logdoci`, `logupci`	Lower and upper endpoints of pivot CIs on the log-scale.
`logdocisym`, `logupcisym`	Symmetric pivot CIs on the log-scale.
`log.est`	The logarithm of the indexed hazard rate estimate, `log(hqm.est)`.
`tLogDoCI`, `tLogUpCI`	Transformed-log CIs based on `2*log(hqm.est) - log-quantiles`.
`tSymLogDoCI`, `tSymLogUpCI`	Symmetric transformed-log CIs.

See Also

`Boot.hrandindex.param`, `Boot.hqm`

Examples

```
marker_name1 <- 'albumin'
marker_name2 <-  'serBilir'
event_time_name <- 'years'
time_name <- 'year'
event_name <- 'status2'
id<-'id'

par.x1  <- 0.0702 #0.149
par.x2 <- 0.0856 #0.10
t.x1 = 0 # refers to zero mean variables - slightly high
t.x2 = 1.9 # refers to zero mean variable - high
b = 0.42#par.alb * b.alb + par.bil *b.bil # 7
t = par.x1 * t.x1 + par.x2 *t.x2

# Calculate the indexed HQM estimator on the original sample:
arg2<-SingleIndCondFutHaz(pbc2, id, marker_name1, marker_name2, event_time_name = 'years',
        time_name = 'year', event_name = 'status2', in.par= c(par.x1,  par.x2), b, t)
hqm.est<-arg2[,2] # Indexed HQM estimator on original sample
time.grid<-arg2[,1] # evaluation grid points
n.est.points<-length(hqm.est)

#Store original sample values:
xin <- pbc2[,c(id, marker_name1, marker_name2, event_time_name, time_name, event_name)]
n <- length(xin$id)
nn<-max(  as.double(xin[,'id']) )

#  Create bootstrap samples by group
set.seed(1)
B<-50  #for display purposes only; for sensible results use B=1000 (slower)
Boot.samples<-list()
for(j in 1:B)
{
  i.use<-c()
  id.use<-c()
  index.nn <- sample (nn, replace = TRUE)
  for(l in 1:nn)
  {
    i.use2<-which(xin[,id]==index.nn[l])
    i.use<-c(i.use, i.use2)
    id.use2<-rep(index.nn[l], times=length(i.use2))
    id.use<-c(id.use, id.use2)
  }
  xin.i<-xin[i.use,]
  xin.i<-xin[i.use,]
  Boot.samples[[j]]<- xin.i[order(xin.i$id),]
}

# Simulate true hazard rate function:
true.hazard<- Sim.True.Hazard(Boot.samples, id='id', marker_name1=marker_name1,
         marker_name2= marker_name2, event_time_name = event_time_name, time_name = time_name,
              event_name = event_name, in.par = c(par.x1,  par.x2), b)
```

```
# Bootstrap the original indexed HQM estimator:
res <- Boot.hrandindex.param(B, Boot.samples, marker_name1, marker_name2, event_time_name,
                           time_name,  event_name, b = 0.4, t = t, true.haz = true.hazard,
                               v.param = c(0.07, 0.08), n.est.points = 100)

# Construct Ci's:
a.sig<-0.05
all.cis.pivot<- Pivot.Index.CIs(B, n.est.points, res, time.grid, hqm.est, a.sig)

# extract Plain   + symmetric CIs
UpCI<-all.cis.pivot[,"UpCI"]
DoCI<-all.cis.pivot[,"DoCI"]
SymUpCI<-all.cis.pivot[,"SymUpCI"]
SymDoCI<-all.cis.pivot[,"SymDoCI"]

J <- 80 #select the first 80 grid points (for display purposes only)
#Plot the selected CIs
plot(time.grid[1:J],   hqm.est[1:J], type="l", ylim=c(0,2), ylab="Hazard rate",
      xlab="time", lwd=2)
polygon(x = c(time.grid[1:J], rev(time.grid[1:J])), y = c(UpCI[1:J], rev(DoCI[1:J])),
        col =  adjustcolor("red", alpha.f = 0.50), border = NA )
lines(time.grid[1:J], SymUpCI[1:J], lty=2, lwd=2 )
lines(time.grid[1:J],  SymDoCI[1:J], lty=2, lwd=2)

# extract transformed from Log HR + symmetric CIs
LogUpCI<-all.cis.pivot[,"LogUpCI"]
LogDoCI<-all.cis.pivot[,"LogDoCI"]
SymLogUpCI<-all.cis.pivot[,"LogSymUpCI"]
SymLogDoCI<-all.cis.pivot[,"LogSymDoCI"]

#Plot the selected CIs
plot(time.grid[1:J],   hqm.est[1:J] , type="l", ylim=c(0,2), ylab="Log Hazard rate",
      xlab="time", lwd=2)
polygon(x = c(time.grid[1:J], rev(time.grid[1:J])), y = c(LogUpCI[1:J], rev(LogDoCI[1:J])),
        col =  adjustcolor("red", alpha.f = 0.50), border = NA )
lines(time.grid[1:J], SymLogUpCI[1:J], lty=2, lwd=2 )
lines(time.grid[1:J],  SymLogDoCI[1:J], lty=2, lwd=2)

# extract Log HR + symmetric CIs
tLogUpCI<-all.cis.pivot[,"LogtUpCI"]
tLogDoCI<-all.cis.pivot[,"LogTDoCI"]
tSymLogUpCI<-all.cis.pivot[,"SymLogtUpCI"]
tSymLogDoCI<-all.cis.pivot[,"SymLogTDoCI"]

#Plot the selected CIs
plot(time.grid[1:J], log(hqm.est[1:J] ), type="l", ylim=c(-5,4), ylab="Hazard rate",
      xlab="time",  lwd=2)
polygon(x = c(time.grid[1:J], rev(time.grid[1:J])), y = c(tLogUpCI[1:J], rev(tLogDoCI[1:J])),
        col =  adjustcolor("red", alpha.f = 0.50), border = NA )
lines(time.grid[1:J], tSymLogUpCI[1:J], lty=2, lwd=2 )
lines(time.grid[1:J],  tSymLogDoCI[1:J], lty=2, lwd=2)
```

prep_boot *Precomputation for wild bootstrap*

Description

Implements key components for the wild bootstrap of the hqm estimator in preparation for obtaining confidence bands.

Usage

```
prep_boot(g_xt, alpha, Ni, Yi, size_s_grid, br_X, br_s, t, b, int_X, x, n)
```

Arguments

g_xt	A vector obtained by g_xt.
alpha	A vector of the marker only hazard on the marker grid obtained by get_alpha.
Ni	A matrix made by make_Ni indicating the occurence.
Yi	A matrix made by make_Yi indicating the exposure.
size_s_grid	Size of the time grid.
br_X	Vector of grid points for the marker values.
br_s	Time value grid points that will be used in the evaluatiuon.
t	Numeric value of the time the function should be evaluated.
b	Bandwidth.
int_X	Position of the linear interpolated marker values on the marker grid.
x	Numeric value of the last observed marker value.
n	Number of individuals.

Details

The function implements

$$A_B(t) = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} \int_0^T \hat{g}_{i,t,x_*}(X_i(s)) V_i \{dN_i(s) - \hat{\alpha}_i(X_i(s)) Z_i(s) ds\},$$

and

$$B_B(t) = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} V_i \{\hat{\Gamma}(t, x_*)^{-1} W_i(t, x_*) - \hat{h}_{x_*}(t)\},$$

where $V \sim N(0,1)$,

$$W_i(t) = \int_0^T \hat{\alpha}_i(X_i(t+s)) Z_i(t+s) Z_i(s) K_b(x_*, X_i(s)) \mathrm{d}s,$$

and

$$\hat{\Gamma}(t,x) = \frac{1}{n} \sum_{i=1}^{n} \int_0^{T-t} Z_i(t+s) Z_i(s) K_b(x, X_i(s)) ds,$$

with Z being the exposure and X the marker.

Value

A list of 5 items. The first two are vectors for calculating A_B and the third one a vector for B_B. The 4th one is the value of the hqm estimator that can also be obtained by h_xt and the last one is the value of Γ.

See Also

Conf_bands

Examples

```
pbc2_id = to_id(pbc2)
size_s_grid <- size_X_grid <- 100
n = max(as.numeric(pbc2$id))
s = pbc2$year
X = pbc2$serBilir
XX = pbc2_id$serBilir
ss <- pbc2_id$years
delta <- pbc2_id$status2
br_s = seq(0, max(s), max(s)/( size_s_grid-1))
br_X = seq(min(X), max(X), (max(X)-min(X))/( size_X_grid-1))

X_lin = lin_interpolate(br_s, pbc2_id$id, pbc2$id, X, s)

int_X <- findInterval(X_lin, br_X)
int_s = rep(1:length(br_s), n)

N <- make_N(pbc2, pbc2_id, br_X, br_s, ss, XX, delta)
Y <- make_Y(pbc2, pbc2_id, X_lin, br_X, br_s,
            size_s_grid, size_X_grid, int_s, int_X, event_time = 'years', n)

b = 1.7
alpha<-get_alpha(N, Y, b, br_X, K=Epan )

Yi <- make_Yi(pbc2, pbc2_id, X_lin, br_X, br_s,
              size_s_grid, size_X_grid, int_s, int_X, event_time = 'years', n)
Ni  <- make_Ni(br_s, size_s_grid, ss, delta, n)

t = 2
x = 2

g = g_xt(br_X, br_s, size_s_grid, int_X, x, t, b, Yi, Y, n)

Boot_all = prep_boot(g, alpha, Ni, Yi, size_s_grid, br_X, br_s, t, b, int_X, x, n)
Boot_all
```

---

prep_cv *Prepare for Cross validation bandwidth selection*

Description

Implements the calculation of the hqm estimator on cross validation data sets. This is a preparation for the cross validation bandwidth selection technique for future conditional hazard rate estimation based on marker information data.

Usage

```
prep_cv(data, data.id, marker_name, event_time_name = 'years',
        time_name = 'year',event_name = 'status2', n, I, b)
```

Arguments

data	A data frame of time dependent data points. Missing values are allowed.
data.id	An id data frame obtained from to_id.
marker_name	The column name of the marker values in the data frame data.
event_time_name	The column name of the event times in the data frame data.
time_name	The column name of the times the marker values were observed in the data frame data.
event_name	The column name of the events in the data frame data.
n	Number of individuals.
I	Number of observations leave out for a K cross validation.
b	Bandwidth.

Details

The function splits the data set via dataset_split and calculates for every splitted data set the hqm estimator

$$\hat{h}_x(t) = \frac{\sum_{i=1}^{n} \int_0^T \hat{\alpha}_i(X_i(t+s)) Z_i(t+s) Z_i(s) K_b(x - X_i(s)) \mathrm{d}s}{\sum_{i=1}^{n} \int_0^T Z_i(t+s) Z_i(s) K_b(x - X_i(s)) \mathrm{d}s},$$

for all x on the marker grid and t on the time grid, where X is the marker, Z is the exposure and $\alpha(z)$ is the marker-only hazard, see get_alpha for more details.

Value

A list of matrices for every cross validation data set with $\hat{h}_x(t)$ for all x on the marker grid and t on the time grid.

See Also

b_selection

Examples

```
pbc2_id = to_id(pbc2)
n = max(as.numeric(pbc2$id))
b = 1.5
I = 26
h_xt_mat_list = prep_cv(pbc2, pbc2_id, 'serBilir', 'years', 'year', 'status2', n, I, b)
```

---

prep_cv2 *Prepare for Cross validation index parameter selection*

Description

Implements the calculation of the hqm estimator on cross validation data sets. This is a preparation for the cross validation index selection technique for future conditional hazard rate estimation based on marker information data.

Usage

```
prep_cv2(in.par, data, data.id, marker_name1, marker_name2, event_time_name = 'years',
        time_name = 'year',event_name = 'status2', n, I, b)
```

Arguments

in.par	Vector of candidate indexing values.
data	A data frame of time dependent data points. Missing values are allowed.
data.id	An id data frame obtained from to_id.
marker_name1	The column name of the marker values in the data frame data.
marker_name2	The column name of the marker values in the data frame data.
event_time_name	The column name of the event times in the data frame data.
time_name	The column name of the times the marker values were observed in the data frame data.
event_name	The column name of the events in the data frame data.
n	Number of individuals.
I	Number of observations leave out for a K cross validation.
b	Bandwidth.

Details

The function splits the data set via dataset_split and calculates for every splitted data set the hqm estimator

$$\hat{h}_x(t) = \frac{\sum_{i=1}^{n} \int_0^T \hat{\alpha}_i(\theta_0^T X_i(t+s)) Z_i(t+s) Z_i(s) K_b(x - \theta_0^T X_i(s)) \mathrm{d}s}{\sum_{i=1}^{n} \int_0^T Z_i(t+s) Z_i(s) K_b(x - \theta_0^T X_i(s)) \mathrm{d}s},$$

for all x on the marker grid and t on the time grid, where X is the marker, Z is the exposure and $\alpha(z)$ is the marker-only hazard, see get_alpha for more details.

Value

A list of matrices for every cross validation data set with $\hat{h}_x(t)$ for all x on the marker grid and t on the time grid.

See Also

b_selection_index_optim

Q1 *Bandwidth selection score Q1*

Description

Calculates a part for the K-fold cross validation score.

Usage

```
Q1(h_xt_mat, int_X, size_X_grid, n, Yi)
```

Arguments

h_xt_mat	A matrix of the estimator for the future conditional hazard rate for all values x and t.
int_X	Vector of the position of the observed marker values in the grid for marker values.
size_X_grid	Numeric value indicating the number of grid points for marker values.
n	Number of individuals.
Yi	A matrix made by make_Yi indicating the exposure.

Details

The function implements

$$Q_1 = \sum_{i=1}^{N} \int_0^T \int_s^T Z_i(t) Z_i(s) \hat{h}^2_{X_i(s)}(t-s) dt ds,$$

where $\hat{h}$ is the hqm estimator, Z the exposure and X the marker.

Value

A value of the score Q1.

See Also

b_selection

Examples

```
pbc2_id = to_id(pbc2)
size_s_grid <- size_X_grid <- 100
n = max(as.numeric(pbc2$id))
s = pbc2$year
X = pbc2$serBilir
XX = pbc2_id$serBilir
ss <- pbc2_id$years
delta <- pbc2_id$status2
br_s = seq(0, max(s), max(s)/( size_s_grid-1))
br_X = seq(min(X), max(X), (max(X)-min(X))/( size_X_grid-1))

X_lin = lin_interpolate(br_s, pbc2_id$id, pbc2$id, X, s)

int_X <- findInterval(X_lin, br_X)
int_s = rep(1:length(br_s), n)

N <- make_N(pbc2, pbc2_id, br_X, br_s, ss, XX, delta)
Y <- make_Y(pbc2, pbc2_id, X_lin, br_X, br_s,
            size_s_grid, size_X_grid, int_s, int_X, event_time = 'years', n)

b = 1.7
alpha<-get_alpha(N, Y, b, br_X, K=Epan )

Yi <- make_Yi(pbc2, pbc2_id, X_lin, br_X, br_s,
              size_s_grid, size_X_grid, int_s, int_X, event_time = 'years', n)
Ni  <- make_Ni(br_s, size_s_grid, ss, delta, n)

t = 2

h_xt_mat = t(sapply(br_s[1:99],
            function(si){h_xt_vec(br_X, br_s, size_s_grid, alpha, t, b, Yi, int_X, n)}))

Q = Q1(h_xt_mat, int_X, size_X_grid, n, Yi)
```

`Quantile.Index.CIs` *Compute Quantile Pointwise Confidence Intervals for for the Indexed Hazard Rate Estimate*

Description

Computes quantile-bootstrap confidence intervals and its symmetric counterparts for the hazard rate, log-hazard rate, and back-transformed (from log scale) hazard rate functions, based on the indexed hazard estimator.

Usage

```
Quantile.Index.CIs(B, n.est.points, Mat.boot.haz.rate, time.grid, hqm.est, a.sig)
```

Arguments

B	Integer. Number of bootsrap replications.
n.est.points	Integer. Number of estimation points at which the indexed hazard estimates and confidence intervals are evaluated.
Mat.boot.haz.rate	A matrix of bootstrap estimated hazard rates with dimensions n.est.points × B, where each column corresponds to one bootstrap replicate.
time.grid	Numeric vector of length n.est.points: the grid points at which the indexed hazard estimates and confidence intervals are calculated.
hqm.est	Indexed hazard estimator, calculated at the grid points time.grid and using the original sample.
a.sig	The significance level (e.g., 0.05) which will be used in computing the confidence intervals.

Details

This function computes several forms of pivot confidence intervals for indexed hazard rate estimates. Set $k_{\alpha/2} = \hat{h}_x^{[\alpha/2]}(t) - \hat{h}_x(t)$ and $k_{1-\alpha/2} = \hat{h}_x^{[1-\alpha/2]}(t) - \hat{h}_x(t)$ where $\hat{h}_x^{[\alpha/2]}(t)$ is the $\alpha/2$ quantile of $\hat{h}_x^{(j)}(t), j = 1, \ldots, B$, obtained by ordering the bootstrap estimators in ascending order and selecting the $\alpha/2$-th ordered value. For example, for $B = 1000$ bootstrap iterations and $\alpha = 0.05$, $\hat{h}_x^{[\alpha/2]}(t)$ will be the 25th smallest out of the 1000 values $\hat{h}_x^{(j)}(t), j = 1, \ldots, 1000$. Also denote with $\bar{k}_{1-\alpha}$ the $1 - \alpha$ quantile of

$$|\hat{h}_x^{(j)}(t) - \hat{h}_x(t)|, j = 1, \ldots, B.$$

Then, the quantile bootstrap CI for $\hat{h}_x(t)$ is given by

$$\left(\hat{h}_x(t) - k_{1-\alpha/2}, \hat{h}_x(t) - k_{\alpha/2}\right).$$

The symmetric quantile CI (basic CI) is

$$\left(\hat{h}_x(t) - \bar{k}_{1-\alpha}, \hat{h}_x(t) + \bar{k}_{1-\alpha}\right).$$

The confidence intervals for the logarithm of the hazard rate function are defined as follows. First set $k^L_{\alpha/2} = \hat{L}_x^{[\alpha/2]}(t) - \hat{L}_x(t)$ and $k^L_{1-\alpha/2} = \hat{L}_x^{[1-\alpha/2]}(t) - \hat{L}_x(t)$.

Also denote with $\bar{k}^L_{1-\alpha}$ the $1 - \alpha$ quantile of $|\hat{L}_x^{(j)}(t) - \hat{L}_x(t)|, j = 1, \ldots, B$. For the log hazard function $L_x(t)$ we have the quantile confidence interval is

$$\left(\hat{L}_x(t) - k^L_{1-\alpha/2}, \hat{L}_x(t) - k^L_{\alpha/2}\right).$$

The corresponding symmetric quantile CI for the log hazard is

$$\left(\hat{L}_x(t) - \bar{k}^L_{1-\alpha}, \hat{L}_x(t) + \bar{k}^L_{1-\alpha}\right).$$

These confidence intervals are transformed back to confidence intervals for the hazard rate function $h_x(t)$:

$$\left(\hat{h}_x(t)e^{-k^L_{1-\alpha/2}}, \hat{h}_x(t)e^{-k^L_{\alpha/2}}\right).$$

The corresponding symmetric confidence interval is

$$\left(\hat{h}_x(t)e^{-\bar{k}^L_{1-\alpha}}, \hat{h}_x(t)e^{\bar{k}^L_{1-\alpha}}\right).$$

Note: The bootstrap matrix `Mat.boot.haz.rate` is assumed to contain estimates produced using the same time grid as `time.grid` and the same estimator used to generate `hqm.est`.

Value

A data frame with the following columns:

`time`	The evaluation grid points.
`est`	Indexed hazard rate estimator `hqm.est`.
`downci`, `upci`	Lower and upper endpoints of basic quantile CIs.
`docisym`, `upcisym`	Lower and upper endpoints of symmetric quantile CIs.
`logdoci`, `logupci`	Lower and upper endpoints of quantile CIs on the log-scale.
`logdocisym`, `logupcisym`	Symmetric log-scale CIs.
`log.est`	The logarithm of the indexed hazard rate estimate, `log(hqm.est)`.
`tLogDoCI`, `tLogUpCI`	Transformed-log CIs based on `2*log(hqm.est) - log-quantiles`.
`tSymLogDoCI`, `tSymLogUpCI`	Symmetric transformed-log CIs.

See Also

`Boot.hrandindex.param`, `Boot.hqm`

Examples

```
marker_name1 <- 'albumin'
marker_name2 <-  'serBilir'
event_time_name <- 'years'
time_name <- 'year'
event_name <- 'status2'
id<-'id'

par.x1  <- 0.0702 #0.149
par.x2 <- 0.0856 #0.10
t.x1 = 0 # refers to zero mean variables - slightly high
```

```
t.x2 = 1.9 # refers to zero mean variable - high
b = 0.42# The result, on the indexed marker 'indmar' of
      #\code{b_selection(pbc2, 'indmar', 'years', 'year', 'status2', I=26, seq(0.2,0.4,by=0.01))}
t = par.x1 * t.x1 + par.x2 *t.x2

# Calculate the indexed HQM estimator on the original sample:
arg2<-SingleIndCondFutHaz(pbc2, id, marker_name1, marker_name2, event_time_name = 'years',
        time_name = 'year', event_name = 'status2', in.par= c(par.x1,  par.x2), b, t)
hqm.est<-arg2[,2] # Indexed HQM estimator on original sample
time.grid<-arg2[,1] # evaluation grid points
n.est.points<-length(hqm.est)

# Store original sample values:
xin <- pbc2[,c(id, marker_name1, marker_name2, event_time_name, time_name, event_name)]
n <- length(xin$id)
nn<-max(  as.double(xin[,'id']) )

#  Create bootstrap samples by group:
set.seed(1)
B<-50   #for display purposes only; for sensible results use B=1000 (slower)
Boot.samples<-list()
for(j in 1:B)
{
  i.use<-c()
  id.use<-c()
  index.nn <- sample (nn, replace = TRUE)
  for(l in 1:nn)
  {
    i.use2<-which(xin[,id]==index.nn[l])
    i.use<-c(i.use, i.use2)
    id.use2<-rep(index.nn[l], times=length(i.use2))
    id.use<-c(id.use, id.use2)
  }
  xin.i<-xin[i.use,]
  xin.i<-xin[i.use,]
  Boot.samples[[j]]<- xin.i[order(xin.i$id),]
}

# Simulate true hazard rate function:
true.hazard<- Sim.True.Hazard(Boot.samples, id='id', marker_name1=marker_name1,
          marker_name2= marker_name2, event_time_name = event_time_name,
          time_name = time_name, event_name = event_name, in.par = c(par.x1, par.x2), b)

# Bootstrap the original indexed HQM estimator:
res <- Boot.hrandindex.param(B, Boot.samples, marker_name1, marker_name2, event_time_name,
                           time_name,  event_name, b = 0.4, t = t, true.haz = true.hazard,
                             v.param = c(0.07, 0.08), n.est.points = 100 )
J <- 80
a.sig<-0.05

# Construct Ci's:
all.cis.quant<- Quantile.Index.CIs(B, n.est.points, res, time.grid, hqm.est, a.sig)
```

```
# extract Plain   + symmetric CIs and plot them:
UpCI<-all.cis.quant[,"upci"]
DoCI<-all.cis.quant[,"downci"]
SymUpCI<-all.cis.quant[,"upcisym"]
SymDoCI<-all.cis.quant[,"docisym"]

#Plot the selected CIs
plot(time.grid[1:J],   hqm.est[1:J], type="l", ylim=c(0,2), ylab="Hazard rate",
      xlab="time", lwd=2)
polygon(x = c(time.grid[1:J], rev(time.grid[1:J])), y = c(UpCI[1:J], rev(DoCI[1:J])),
        col =  adjustcolor("red", alpha.f = 0.50), border = NA )
lines(time.grid[1:J], SymUpCI[1:J], lty=2, lwd=2 )
lines(time.grid[1:J],  SymDoCI[1:J], lty=2, lwd=2)

# extract transformed from Log HR + symmetric CIs
LogUpCI<-all.cis.quant[,"logupci"]
LogDoCI<-all.cis.quant[,"logdoci"]
SymLogUpCI<-all.cis.quant[,"logupcisym"]
SymLogDoCI<-all.cis.quant[,"logdocisym"]

#Plot the selected CIs
plot(time.grid[1:J],   hqm.est[1:J], type="l", ylim=c(0,2), ylab="Hazard rate",
      xlab="time", lwd=2)
polygon(x = c(time.grid[1:J], rev(time.grid[1:J])), y = c(LogUpCI[1:J], rev(LogDoCI[1:J])),
        col =  adjustcolor("red", alpha.f = 0.50), border = NA )
lines(time.grid[1:J], SymLogUpCI[1:J], lty=2, lwd=2 )#, lwd=3
lines(time.grid[1:J],  SymLogDoCI[1:J], lty=2, lwd=2)

# extract Log HR + symmetric CIs
tLogUpCI<-all.cis.quant[,"tLogUpCI"]
tLogDoCI<-all.cis.quant[,"tLogDoCI"]
tSymLogUpCI<-all.cis.quant[,"tSymLogUpCI"]
tSymLogDoCI<-all.cis.quant[,"tSymLogDoCI"]

#Plot the selected CIs
plot(time.grid[1:J],  log(hqm.est[1:J]), type="l", ylim=c(-5,4), ylab="Log Hazard rate",
     xlab="time", lwd=2)
polygon(x = c(time.grid[1:J], rev(time.grid[1:J])), y = c(tLogUpCI[1:J], rev(tLogDoCI[1:J])),
      col =  adjustcolor("red", alpha.f = 0.50), border = NA )
lines(time.grid[1:J], tSymLogUpCI[1:J], lty=2, lwd=2 )
lines(time.grid[1:J],  tSymLogDoCI[1:J], lty=2, lwd=2)
```

R_K *Bandwidth selection score R*

Description

Calculates a part for the K-fold cross validation score.

Usage

```
R_K(h_xt_mat_list, int_X, size_X_grid, Yi, Ni, n)
```

Arguments

h_xt_mat_list	A list of matrices for all cross validation data sets. Each matrix contains the estimator with the future conditional hazard rate for all values x and t and the respected data set.
int_X	Vector of the position of the observed marker values in the grid for marker values.
size_X_grid	Numeric value indicating the number of grid points for marker values.
Yi	A matrix made by make_Yi indicating the exposure.
Ni	A matrix made by make_Ni indicating the occurence.
n	Number of individuals.

Details

The function implements the estimator

$$\hat{R}_K = \sum_{j=1}^{K} \sum_{i \in I_j} \int_0^T g_i^{-I_j}(t) dN_i(t),$$

where $\hat{g}_i^{-I_j}(t) = \int_0^t Z_i(s) \hat{h}_{X_i(s)}^{-I_j}(t-s)ds$, and $\hat{h}^{-I_j}$ is estimated without information from all counting processes i with $i \in I_j$. This function estimates

$$R = \sum_{i=1}^{N} \int_0^T \int_s^T Z_i(t) Z_i(s) \hat{h}_{X_i(s)}(t-s) h_{X_i(s)}(t-s) dt ds.$$

where $\hat{h}$ is the hqm estimator, Z the exposure and X the marker.

Value

A matrix with $\hat{g}_i^{-I_j}(t)$ for all individuals i and time grid points t.

See Also

b_selection

Examples

```
pbc2_id = to_id(pbc2)
n = max(as.numeric(pbc2$id))
b = 1.5
I = 104
h_xt_mat_list = prep_cv(pbc2, pbc2_id, 'serBilir', 'years', 'year', 'status2', n, I, b)
```

```
size_s_grid <- size_X_grid <- 100
s = pbc2$year
X = pbc2$serBilir
br_s = seq(0, max(s), max(s)/( size_s_grid-1))
br_X = seq(min(X), max(X), (max(X)-min(X))/( size_X_grid-1))

ss <- pbc2_id$years
delta <- pbc2_id$status2

X_lin = lin_interpolate(br_s, pbc2_id$id, pbc2$id, X, s)
int_X <- findInterval(X_lin, br_X)
int_s = rep(1:length(br_s), n)

Yi <- make_Yi(pbc2, pbc2_id, X_lin, br_X, br_s,
              size_s_grid, size_X_grid, int_s, int_X, 'years', n)
Ni  <- make_Ni(br_s, size_s_grid, ss, delta, n)

R = R_K(h_xt_mat_list, int_X, size_X_grid, Yi, Ni, n)
R
```

`Sim.True.Hazard` *Simulated true hazard (bootstrap average)*

Description

Compute the Monte Carlo / bootstrap averaged alpha (true hazard) across a list of datasets.

Usage

```
Sim.True.Hazard(data.use, id, marker_name1 = marker_name1, marker_name2 = marker_name2,
    event_time_name = event_time_name, time_name = time_name, event_name = event_name,
      in.par, b)
```

Arguments

`data.use`	List of data.frames (bootstrap samples).
`id`	Id column name.
`marker_name1`	First marker name.
`marker_name2`	Second marker name.
`event_time_name`	Name of event time column.
`time_name`	Name of observation time column.
`event_name`	Name of event indicator column.
`in.par`	Numeric indexing parameters.
`b`	Bandwidth.

Value

Numeric vector (row means of alpha estimates).

Examples

```
marker_name1 <- 'albumin'
marker_name2 <-  'serBilir'
event_time_name <- 'years'
time_name <- 'year'
event_name <- 'status2'
id<-'id'

par.x1  <- 0.0702  #indexing parameter for marker 1
par.x2 <- 0.0856  #indexing parameter for marker 2

t.x1 = 0 # conditioning level for marker 1, assumes zero mean variable
t.x2 = 1.9 # conditioning level for marker 2, assumes zero mean variable
b = 0.42 # The result, on the indexed marker 'indmar' of
      #\code{b_selection(pbc2, 'indmar', 'years', 'year', 'status2', I=26, seq(0.2,0.4,by=0.01))}
t = par.x1 * t.x1 + par.x2 *t.x2

xin <- pbc2[,c(id, marker_name1, marker_name2, event_time_name, time_name, event_name)]
n <- length(xin$id)
nn<-max(  as.double(xin[,'id']) )

#  Create bootstrap samples by group
set.seed(1)
B<-100 #set bootstrap iterations here
Boot.samples<-list()
for(j in 1:B)
{
  i.use<-c()
  id.use<-c()
  index.nn <- sample (nn, replace = TRUE)
  for(l in 1:nn)
  {
    i.use2<-which(xin[,id]==index.nn[l])
    i.use<-c(i.use, i.use2)
    id.use2<-rep(index.nn[l], times=length(i.use2))
    id.use<-c(id.use, id.use2)
  }
  xin.i<-xin[i.use,]
  xin.i<-xin[i.use,]
  Boot.samples[[j]]<- xin.i[order(xin.i$id),]
}

#compute the bootstrap simulated true HR function:
true.hazard<- Sim.True.Hazard(Boot.samples, id='id', marker_name1=marker_name1,
    marker_name2= marker_name2, event_time_name = event_time_name, time_name = time_name,
              event_name = event_name, in.par = c(par.x1,  par.x2), b)
```

SingleIndCondFutHaz *Local linear future conditional hazard estimator (wrapper)*

Description

Compute the indexed local linear future conditional hazard rate estimator on a grid.

Usage

```
SingleIndCondFutHaz(datain, id, marker_name1, marker_name2, event_time_name = 'years',
       time_name = 'year', event_name = 'status2', in.par, b, t)
```

Arguments

datain	Data frame with longitudinal observations.
id	Name of id column (string).
marker_name1	Name of first marker (string).
marker_name2	Name of second marker (string).
event_time_name	Name of event time column.
time_name	Name of observation time column.
event_name	Name of event indicator column.
in.par	Numeric vector length 2 with indexing parameters.
b	Bandwidth parameter.
t	conditioning parameter.

Value

A data frame with columns time and est.

Examples

```
marker_name1 <- 'albumin'
marker_name2 <-  'serBilir'
event_time_name <- 'years'
time_name <- 'year'
event_name <- 'status2'
id<-'id'

par.x1  <- 0.0702
par.x2 <- 0.0856
t.x1 = 0 # refers to zero mean variables - slightly high
t.x2 = 1.9 # refers to zero mean variable - high
```

```
b = 0.42 # The result, on the indexed marker 'indmar' of
      #\code{b_selection(pbc2, 'indmar', 'years', 'year', 'status2', I=26, seq(0.2,0.4,by=0.01))}
t = par.x1 * t.x1 + par.x2 *t.x2

# Calculate the indexed HQM estimator on the original sample:
arg2<-SingleIndCondFutHaz(pbc2, id, marker_name1, marker_name2, event_time_name = 'years',
      time_name = 'year',  event_name = 'status2', in.par= c(par.x1,  par.x2), b, t)
hqm.est<-arg2[,2]
time.grid<-arg2[,1]
n.est.points<-length(hqm.est)
```

`StudentizedBwB.Index.CIs` *Compute Studentized Double Bootstrap Pointwise Confidence Intervals for the Indexed Hazard Rate Estimate*

Description

Computes bootstrap confidence intervals—studentized (double bootstrap) and its symmetric versions for the hazard rate, log-hazard rate, and back-transformed (from the log scale) hazard rate functions, based on the indexed hazard estimator.

Usage

```
StudentizedBwB.Index.CIs(n.est.points, all.mat, time.grid, hqm.est, a.sig)
```

Arguments

`n.est.points`	Integer. Number of estimation points at which the indexed hazard estimates and confidence intervals are evaluated.
`all.mat`	A list of matrices of bootstrap estimated hazard and log hazard rates with dimensions `n.est.points` × `B`, where each column corresponds to one bootstrap replicate.
`time.grid`	Numeric vector of length `n.est.points`: the grid points at which the indexed hazard estimates and confidence intervals are calculated.
`hqm.est`	Indexed hazard estimator, calculated at the grid points `time.grid` and using the original sample.
`a.sig`	The significance level (e.g., 0.05) which will be used in computing the confidence intervals.

Details

This function computes several forms of studentized confidence intervals for the indexed hazard rate function. First, for each bootstrap iteration $j = 1, \ldots, B$ we construct estimators of the standard deviation, denoted by $\hat{\sigma}_j^2$ as follows: each bootstrap sample is itself bootstrapped, say B_1 times, we estimate the corresponding B_1 index parameters $\hat{\theta}_{j,k} = (\hat{\theta}_{1,j,k}, \hat{\theta}_{2,j,k}), k = 1, \ldots, B_1$, and use them

to calculate the corresponding hazard estimators $\hat{h}_{x,}^{(j,k)}(t), k = 1, \ldots, B_1$. Finally we calculate $\hat{\sigma}_j^2$ as the sample variance of $\hat{h}_x^{(j,1)}(t), \ldots, \hat{h}_x^{(j,B_1)}(t)$. Let $k_{\alpha/2}^s, k_{1-\alpha/2}^s$ and $\bar{k}_{1-\alpha}^s$ be respectively the $\alpha/2, 1-\alpha/2$ and $1-\alpha$ quantiles of

$$\frac{|\hat{h}_x^{(j)}(t) - \hat{h}_x(t)|}{\hat{\sigma}_j}, \ j = 1, \ldots, B.$$

Given the bootstrap estimators $\hat{h}_x^{(j)}(t), j = 1, \ldots, B$, the estimator of the original data set $\hat{h}_x(t)$ and the quantiles $k_{\alpha/2}^s, k_{1-\alpha/2}^s$ and $\bar{k}_{1-\alpha}^s$, the studentized (double bootstrap) confidence interval (CI) for $\hat{h}_x(t)$ is given by

$$\left(\hat{h}_x(t) - \hat{\sigma} k_{1-\alpha/2}^s, \hat{h}_x(t) - \hat{\sigma} k_{\alpha/2}^s\right).$$

The symmetric studentized confidence interval (CI) for $\hat{h}_x(t)$ defined by

$$\left(\hat{h}_x(t) - \hat{\sigma} \bar{k}_{1-\alpha}^s, \hat{h}_x(t) + \hat{\sigma} \bar{k}_{1-\alpha}^s\right).$$

For the confidence intervals for the logarithm of the hazard rate function first set $k_{\alpha/2}^{L,s}, k_{1-\alpha/2}^{L,s}$ and $\bar{k}_{1-\alpha}^{L,s}$ be the $\alpha/2, 1-\alpha/2$ and $1-\alpha$ quantile of $|\hat{L}_x^{(j)}(t) - \hat{L}_x(t)|/\hat{\sigma}_j, j = 1, \ldots, B$.

The studentized (double bootstrap) CI confidence interval for the logarithm of the hazard rate function is

$$\left(\hat{L}_x(t) - \hat{\sigma} k_{1-\alpha/2}^{L,s}, \hat{L}_x(t) - \hat{\sigma} k_{\alpha/2}^{L,s}\right).$$

Its symmetric studentized (double bootstrap) CI for the log hazard is

$$\left(\hat{L}_x(t) - \hat{\sigma} \bar{k}_{1-\alpha}^{L,s}, \hat{L}_x(t) + \hat{\sigma} \bar{k}_{1-\alpha}^{L,s}\right).$$

These confidence intervals are transformed back to yield the following confidence intervals for the hazard rate function $h_x(t)$:

$$\left(\hat{h}_x(t) e^{-\hat{\sigma} k_{1-\alpha/2}^{L,s}}, \hat{h}_x(t) e^{-\hat{\sigma} k_{\alpha/2}^{L,s}}\right),$$

and the symmetric version

$$\left(\hat{h}_x(t) e^{-\hat{\sigma} \bar{k}_{1-\alpha}^{L,s}}, \hat{h}_x(t) e^{\hat{\sigma} \bar{k}_{1-\alpha}^{L,s}}\right).$$

Note: The bootstrap matrix `Mat.boot.haz.rate` is assumed to contain estimates produced using the same time grid as `time.grid` and the same estimator used to generate `hqm.est`.

Value

A data frame with the following columns:

`time`	The evaluation grid points.
`est`	Indexed hazard rate estimator `hqm.est`.
`downci`, `upci`	Lower and upper endpoints of basic studentized CIs.
`docisym`, `upcisym`	Lower and upper endpoints of symmetric CIs.
`logdoci`, `logupci`	Lower and upper endpoints of studentized CIs on the log-scale.
`logdocisym`, `logupcisym`	Symmetric log-scale CIs.
`log.est`	The logarithm of the indexed hazard rate estimate, `log(hqm.est)`.
`tLogDoCI`, `tLogUpCI`	Transformed-log CIs based on `2*log(hqm.est) - log-quantiles`.
`tSymLogDoCI`, `tSymLogUpCI`	Symmetric transformed-log CIs.

See Also

`Boot.hrandindex.param`, `Boot.hqm`

Examples

```
marker_name1 <- 'albumin'
marker_name2 <-  'serBilir'
event_time_name <- 'years'
time_name <- 'year'
event_name <- 'status2'
id<-'id'

par.x1  <- 0.0702 #0.149
par.x2 <- 0.0856 #0.10
t.x1 = 0 # refers to zero mean variables - slightly high
t.x2 = 1.9 # refers to zero mean variable - high
b = 0.42#par.alb * b.alb + par.bil *b.bil # 7
t = par.x1 * t.x1 + par.x2 *t.x2

xin <- pbc2[,c(id, marker_name1, marker_name2, event_time_name, time_name, event_name)]
n <- length(xin$id)
nn<-max(  as.double(xin[,'id']) )

#  Create bootstrap samples by group
set.seed(1)
B<-5 #for display purposes only; for sensible results use B=200 (slower)
B1<-5  #for display purposes only; use B1=20 (slower)
Boot.samples<-list()
for(j in 1:B)
{
```

```
  i.use<-c()
  id.use<-c()
  index.nn <- sample (nn, replace = TRUE)
  for(l in 1:nn)
  {
    i.use2<-which(xin[,id]==index.nn[l])
    i.use<-c(i.use, i.use2)
    id.use2<-rep(index.nn[l], times=length(i.use2))
    id.use<-c(id.use, id.use2)
  }
  xin.i<-xin[i.use,]
  xin.i<-xin[i.use,]
  Boot.samples[[j]]<- xin.i[order(xin.i$id),]
}

# Calculate the indexed HQM estimator on the original sample:
arg2<-SingleIndCondFutHaz(pbc2, id, marker_name1, marker_name2, event_time_name = 'years',
             time_name = 'year',  event_name = 'status2', in.par= c(par.x1,  par.x2), b, t)
hqm.est<-arg2[,2] # Indexed HQM estimator on original sample
time.grid<-arg2[,1] # evaluation grid points
n.est.points<-length(hqm.est)

# Simulate true hazard rate function:
true.hazard<- Sim.True.Hazard(Boot.samples, id='id', marker_name1=marker_name1,
              marker_name2= marker_name2, event_time_name = event_time_name,
              time_name = time_name, event_name = event_name,
              in.par = c(par.x1,  par.x2), b)

# Bootstrap the original indexed HQM estimator:
all.mat.use<-BwB.HRandIndex.param(B, B1, Boot.samples, marker_name1, marker_name2,
             event_time_name, time_name, event_name, b, t, true.haz=true.hazard,
             v.param=c(par.x1,  par.x2), hqm.est=hqm.est, id= 'id', xin=xin)

# Construct Ci's:
a.sig<-0.05
st.ci.data<-StudentizedBwB.Index.CIs(n.est.points, all.mat.use, time.grid, hqm.est, a.sig)

# extract Plain + symmetric CIs
UpCI<-st.ci.data[,"UpCI"]
DoCI<-st.ci.data[,"DoCI"]
SymUpCI<-st.ci.data[,"SymUpCI"]
SymDoCI<-st.ci.data[,"SymDoCI"]

#Plot the selected CIs
J<-80 #select the first 80 grid points (for display purposes only)
plot(time.grid[1:J], hqm.est[1:J], type="l", ylim=c(0,2), ylab="Hazard rate", xlab="time",
           lwd=2)
polygon(x = c(time.grid[1:J], rev(time.grid[1:J])), y = c(UpCI[1:J],  rev(DoCI[1:J])),
      col =  adjustcolor("red", alpha.f = 0.50), border = NA )
lines(time.grid[1:J], SymUpCI[1:J], lty=2, lwd=2 )
lines(time.grid[1:J],  SymDoCI[1:J], lty=2, lwd=2)

# extract transformed from Log HR + symmetric CIs
```

```
LogUpCI<-st.ci.data[,"LogUpCI"]
LogDoCI<-st.ci.data[,"LogDoCI"]
SymLogUpCI<-st.ci.data[,"LogSymUpCI"]
SymLogDoCI<-st.ci.data[,"LogSymDoCI"]

#Plot the selected CI's
plot(time.grid[1:J], log(hqm.est[1:J]), type="l", ylim=c(-5,4), ylab="Log Hazard rate",
      xlab="time",  lwd=2)
polygon(x = c(time.grid[1:J], rev(time.grid[1:J])), y = c(LogUpCI[1:J], rev(LogDoCI[1:J])),
      col =  adjustcolor("red", alpha.f = 0.50), border = NA )
lines(time.grid[1:J], SymLogUpCI[1:J], lty=2, lwd=2 )
lines(time.grid[1:J],  SymLogDoCI[1:J], lty=2, lwd=2)

# extract Log HR + symmetric CIs
tLogUpCI<-st.ci.data[,"LogtUpCI"]
tLogDoCI<-st.ci.data[,"LogTDoCI"]
tSymLogUpCI<-st.ci.data[,"SymLogtUpCI"]
tSymLogDoCI<-st.ci.data[,"SymLogTDoCI"]

#Plot the selected CIs
plot(time.grid[1:J],  hqm.est[1:J] , type="l", ylim=c(0,2), ylab="Hazard rate", xlab="time",
      lwd=2)
polygon(x = c(time.grid[1:J], rev(time.grid[1:J])), y = c(tLogUpCI[1:J], rev(tLogDoCI[1:J])),
    col =  adjustcolor("red", alpha.f = 0.50), border = NA )
lines(time.grid[1:J], tSymLogUpCI[1:J], lty=2, lwd=2 )
lines(time.grid[1:J],  tSymLogDoCI[1:J], lty=2, lwd=2)
```

`to_id` *Event data frame*

Description

Creates a data frame with only one entry per individual from a data frame with time dependent data. The resulting data frame focusses on the event time and the last observed marker value.

Usage

```
to_id(data_set)
```

Arguments

`data_set`	A data frame of time dependent data points. Missing values are allowed.

Details

The function `to_id` uses a data frame of time dependent marker data to create a smaller data frame with only one entry per individual, the last observed marker value and the event time. Note that the column indicating the individuals must have the name `id`. Note also that this data frame is similar to `pbc2.id` from the `JM` package with the difference that the last observed marker value instead of the first one is captured.

Value

A data frame with only one entry per individual.

Examples

```
data_set.id = to_id(pbc2)
```

Index