Package ‘QTOCen’

October 12, 2022

Type Package

Title Quantile-Optimal Treatment Regimes with Censored Data

Version 0.1.1

Author Yu Zhou [cre, aut],
Lan Wang [ctb]

Maintainer Yu Zhou <zhou0269@umn.edu>

Description Provides methods for estimation of mean- and quantile-optimal treatment regimes from censored data.
Specifically, we have developed distinct functions for three types of right censoring for static treatment using quantile criterion: (1) independent/random censoring, (2) treatment-dependent random censoring, and (3) covariates-dependent random censoring. It also includes a function to estimate quantile-optimal dynamic treatment regimes for independent censored data. Finally, this package also includes a simulation data generative model of a dynamic treatment experiment proposed in literature.

License GPL (>= 2)

LazyData TRUE

Imports survival, rgenoud (>= 5.8), quantreg (>= 5.18), stats,
grDevices, methods, Rdpack, MatrixModels

RdMacros Rdpack

Depends R (>= 3.3), utils

RoxygenNote 6.1.1

Suggests parallel, stringr, testthat, faraway, quantoptr (>= 0.1.3),
survminer

NeedsCompilation no

Encoding UTF-8

Repository CRAN

Date/Publication 2019-06-04 12:10:10 UTC

R topics documented:

`Bnk_func` *Generate biquadratic kernel weights for a univariate variable*

Description

This is the biquadratic kernel function, that weights observations by their distances to the target observation.

Usage

```
Bnk_func(x0k, Xk, bw.bnk)
```

Arguments

`x0k`	Numeric scalar. One univariate covariate value of interest from one observation.
`Xk`	Numerical vector. The vector of the same covariate from observations
`bw.bnk`	The bandwith scalar parameter.

Value

This function returns a list of kernel weights with the same length of input `Xk`.

Note

This function is widely used for generating kernel weights for nonparametrically estimating conditional survival functions. See Section 2.3 of (Wang and Wang 2009).

References

Wang HJ, Wang L (2009). "Locally weighted censored quantile regression." *Journal of the American Statistical Association*, **104**(487), 1117–1128.

Examples

```
Bnk_func(x0k=0, Xk=c(-5:5), bw.bnk=10)
```

---

est_mean_ipwe *Estimate the marginal mean response of a linear static treatment regime*

Description

Assume we have binary treatment options for each subject in the target population. This function evaluates a given treatment regime by the estimated marginal mean response. We assume the space of treatment regimes are linear decision functions indexed by parametric coefficients.

This R function is an empirical *value function* in the literature of optimal treatment regime estimation. Since the goal here is to maximize population's **marginal mean** response, this function, which estimates the performance of a set of parameters in terms of the **marginal mean**, is the objective function in a nonparametric policy-search method.

The user facing application which utilizes this function is `IPWE_mean_IndCen`.

Usage

```
est_mean_ipwe(beta, x, censor_y, delta, ph, a, ghat,
  check_complete = TRUE)
```

Arguments

`beta`	Numeric vector. A set of parameter that indexes the regime.
`x`	Numeric Matrix. The baseline covariates from all observed data.
`censor_y`	Numeric vector. The censored survival times from all observed data, i.e. `censor_y = min(Y, C)`
`delta`	Numeric vector. The censoring indicators from all observed data. We use 1 for uncensored, 0 for censored.
`ph`	Numeric vector. The estimated propensity score of being assigned treatment `A=1` by the original data generation mechanism for all observed data.
`a`	Numeric vector. The vector of observed treatment level for all observed data. Treatment levels should be coded as 0/1.

`ghat`	Numeric vector. The conditional/unconditional probabilities of event that the censoring variable is larger than the observed survival time given covariates for each observation. a.k.a $F(T > y_0 \mid x_0)$. This can be calculated by function `LocalKM`. Estimation of conditional cumulative function value at y_0 is implemented in `tauhat_func`.
`check_complete`	logical. Since this value estimation method is purely nonparametric, we need at least one unit in collected data such that the observed treatment assignment is the same what the regime parameter suggests. If `check_complete` is `TRUE`. It will check if any observation satisfies this criterion. When none observation satisfies, a message is printed to console to raise users awareness that the input regime parameter `beta` does not agree with any observed treatment level assignment. Then a sufficiently small number is returned from this function, to keep the genetic algorithm running smoothly.

Examples

```
GenerateData <- function(n)
{
  x1 <- runif(n, min=-0.5,max=0.5)
  x2 <- runif(n, min=-0.5,max=0.5)
  error <- rnorm(n, sd= 1)
  ph <- rep(0.5,n)
  a <- rbinom(n = n, size = 1, prob=ph)
  c <- 1.5 +  + runif(n = n, min=0, max=2)
  cmplt_y <-  pmin(2+x1+x2 +  a*(1 - x1 - x2) +  (0.2 + a*(1+x1+x2)) * error, 4.4)
  censor_y <- pmin(cmplt_y, c)
  delta <- as.numeric(c > cmplt_y)
  return(data.frame(x1=x1,x2=x2,a=a, censor_y = censor_y, delta=delta))
}
n <- 100
data <- GenerateData(n)

# here the value for argument ghat uses 0.5 vector for brevity.
mean_hat <- est_mean_ipwe(c(-1,0,2), x=cbind(1, data$x1, data$x2),
                          censor_y = data$censor_y, delta = data$delta, ph = rep(0.5,n),
                          a = data$a, ghat = rep(0.5,n))
```

est_quant_ipwe *Estimate the marginal quantile response of a linear static treatment regime*

Description

Assume we have binary treatment options for each subject in the target population. This function evaluates a given treatment regime by the estimated marginal mean response. We assume the space of treatment regimes are linear decision functions indexed by parametric coefficients.

This R function is an empirical *value function* in the literature of optimal treatment regime estimation. Since the goal here is to maximize population's **marginal quantile**, this function, which estimates the perforamce of a set of parameters in terms of **marginal quantile**, is the objective function in a nonparametric policy-search method.

The user facing application which utilizes this function is `IPWE_Qopt_IndCen`.

Usage

```
est_quant_ipwe(beta, sign_beta1, x, censor_y, delta, epsi, a, tau,
  check_complete = TRUE, Penalty.level = 0)
```

Arguments

`beta`	Numerical vector. Exclude the coefficient for the first nontrivial covariate. So if there are `k` covariates, the length of `beta` should equal `k+1-1=k` because the intercept needs one coefficient as well.
`sign_beta1`	logical. FALSE if the coefficient for the first continuous variable is fixed to be negative one; TRUE if positive one.
`x`	Numeric Matrix. The baseline covariates from all observed data.
`censor_y`	Numeric vector. The censored survival times from all observed data, i.e. `censor_y = min(Y, C)`
`delta`	Numeric vector. The censoring indicators from all observed data. We use 1 for uncensored, 0 for censored.
`epsi`	the product of (1) the probability of being assigned the observed treatment level through the original treatment assignment mechanism and (2) the conditional survival probability of the censoring variable at `censor_y`.
`a`	Numeric vector. The vector of observed treatment level for all observed data. Treatment levels should be coded as 0/1.
`tau`	a value between 0 and 1. This is the quantile of interest.
`check_complete`	logical. Since this value estimation method is purely nonparametric, we need at least one unit in collected data such that the observed treatment assignment is the same what the regime parameter suggests. If `check_complete` is `TRUE`. It will check if any observation satisfies this criterion. When none observation satisfies, a message is printed to console to raise users awareness that the input regime parameter `beta` does not agree with any observed treatment level assignment. Then a sufficiently small number is returned from this function, to keep the genetic algorithm running smoothly.
`Penalty.level`	the level that determines which objective function to use. `Penalty.level = 0` indicates no regularization; `Penalty.level = 1` indicates the value function estimation minus the means absolute average coefficient is the output, which is useful trick to achieve uniqueness of estimated optimal TR when resolution of input response is low.

Examples

```
GenerateData <- function(n)
{
  x1 <- runif(n, min=-0.5,max=0.5)
  x2 <- runif(n, min=-0.5,max=0.5)
  error <- rnorm(n, sd= 1)
  ph <- rep(0.5,n)
  a <- rbinom(n = n, size = 1, prob=ph)
  c <- 1.5 +  + runif(n = n, min=0, max=2)
  cmplt_y <-  pmin(2+x1+x2 +  a*(1 - x1 - x2) +  (0.2 + a*(1+x1+x2)) * error, 4.4)
  censor_y <- pmin(cmplt_y, c)
  delta <- as.numeric(c > cmplt_y)
  return(data.frame(x1=x1,x2=x2,a=a, censor_y = censor_y, delta=delta))
}
n <- 100
data <- GenerateData(n)

# here the value for argument epsi uses 0.5 vector for brevity in notation.
quant_hat <- est_quant_ipwe(beta=c(-1,2), sign_beta1=TRUE, x=cbind(1, data$x1, data$x2),
                            censor_y = data$censor_y, delta = data$delta, tau=0.5,
                            epsi = rep(0.5,n), a = data$a)
```

---

est_quant_TwoStg_ipwe *Estimate the marginal quantile response of a specific dynamic TR*

Description

Assume we have binary treatment options for two sequential stages with a fixed time duration between them. This means for each subject in the target population if the censored survival time or the time-to-event is beyond the timepoint of the second treatment.

This function evaluates a given dynamic treatment regime and returns the estimated marginal quantile response.

We assume the space of two-stage treatment regimes is a cartesian product of two single-stage linear treatment regime space.

The user facing function that applies this function is `IPWE_Qopt_DTR_IndCen`.

Usage

```
est_quant_TwoStg_ipwe(n, beta, sign_beta1.stg1, sign_beta1.stg2, txVec1,
  txVec2_na_omit, s_Diff_Time, nvars.stg1, nvars.stg2, p.data1, p.data2,
  censor_y, delta, ELG, w_di_vec, tau, check_complete = TRUE,
  Penalty.level = 0)
```

Arguments

`n`	the sample size
`beta`	the vector of coefficients indexing a two-stage treatment regime
`sign_beta1.stg1`	Is sign of the coefficient for the first non-intercept variable for the first stage known? Default is NULL, meaning user does not have contraint on the sign; FALSE if the coefficient for the first continuous variable is fixed to be `-1`; TRUE if `1`. We can make the search space discrete because we employ $\|\beta_1\| = 1$ scale normalizaion.
`sign_beta1.stg2`	Default is NULL. Similar to `sign_beta1.stg1`.
`txVec1`	the vector of treatment received at the first stage
`txVec2_na_omit`	the vector of second stage treatment for patients who indeed second stage treatment
`s_Diff_Time`	the length of time between the first stage treatment and the second stage treatment
`nvars.stg1`	number of coeffients for the decision rule of the first stage
`nvars.stg2`	number of coeffients for the decision rule of the second stage
`p.data1`	the design matrix to be used for decision in stage one
`p.data2`	the design matrix to be used for decision in stage two
`censor_y`	Numeric vector. The censored survival times from all observed data, i.e. `censor_y = min(Y, C)`
`delta`	Numeric vector. The censoring indicators from all observed data. We use 1 for uncensored, 0 for censored.
`ELG`	the boolean vector of whether patients get the second stage treatment
`w_di_vec`	the inverse probability weight for two stage experiments
`tau`	a value between 0 and 1. This is the quantile of interest.
`check_complete`	logical. Since this value estimation method is purely nonparametric, we need at least one unit in collected data such that the observed treatment assignment is the same what the regime parameter suggests. If `check_complete` is `TRUE`. It will check if any observation satisfies this criterion. When none observation satisfies, a message is printed to console to raise users awareness that the input regime parameter `beta` does not agree with any observed treatment level assignment. Then a sufficiently small number is returned from this function, to keep the genetic algorithm running smoothly.
`Penalty.level`	the level that determines which objective function to use. `Penalty.level = 0` indicates no regularization; `Penalty.level = 1` indicates the value function estimation minus the means absolute average coefficient is the output, which is useful trick to achieve uniqueness of estimated optimal TR when resolution of input response is low.

Examples

```
##########################################################################
# Note: the preprocessing steps prior to calling est_quant_TwoStg_ipwe() #
# are wrapped up in IPWE_Qopt_DTR_IndCen().                              #
# w_di_vec is the inverse probability weight for two stage experiments   #
# We recommend users to use function IPWE_Qopt_DTR_IndCen() directly.    #
# Below is a simple customized calculation of the weight that only works #
# for this example                                                       #
##########################################################################

library(survival)
# Simulate data
n=200
s_Diff_Time = 1
D <- simJLSDdata(n, case="a")

# give regime classes
regimeClass.stg1 <- as.formula(a0~x0)
regimeClass.stg2 <- as.formula(a1~x1)

# extract columns that matches each stage's treatment regime formula
p.data1 <- model.matrix(regimeClass.stg1, D)

# p.data2 would only contain observations with non-null value.
p.data2 <- model.matrix(regimeClass.stg2, D)

txVec1 <- D[, "a0"]
# get none-na second stage treatment levels in data
txVec2 <- D[, "a1"]
txVec2_na_omit <- txVec2[which(!is.na(txVec2))]

# Eligibility flag
ELG <- (D$censor_y  >  s_Diff_Time)

# Build weights
D$deltaC <- 1 - D$delta
survfit_all <- survfit(Surv(censor_y, event = deltaC)~1, data=D)
survest <- stepfun(survfit_all$time, c(1, survfit_all$surv))
D$ghat <- survest(D$censor_y)
g_s_Diff_Time <- survest(s_Diff_Time)
D$w_di_vec <- rep(-999, n)
for(i in 1:n){
  if (!ELG[i]) {
      D$w_di_vec[i] <- 0.5 * D$ghat[i]} else {
         D$w_di_vec[i] <- 0.5* D$ghat[i] * 0.5
 }
}


qhat <- est_quant_TwoStg_ipwe(n=n, beta=c(2.5,2.8),
             sign_beta1.stg1 = FALSE, sign_beta1.stg2=FALSE,
             txVec1=txVec1, txVec2_na_omit=txVec2_na_omit, s_Diff_Time=1,
             nvars.stg1=2, nvars.stg2=2,
             p.data1=p.data1,
             p.data2=p.data2,
             censor_y=D$censor_y,
             delta=D$delta,
             ELG=ELG, w_di_vec=D$w_di_vec,
             tau=0.3)
```

`Gene_Mean_CenIPWE` *A low-level function for the generic optimization step in estimating Mean-optimal treatment regime for censored data*

Description

This function supports the `IPWE_mean_IndCen` function. It does the genetic algorithm based method with inverse probability weighting for censored data. In the future, if more complicated applications/scenarios is sought after for mean optimality, users may create their own wrapper function based on `Gene_Mean_CenIPWE`.

Usage

```
Gene_Mean_CenIPWE(data_aug, ph, p_level, regimeClass, Domains = NULL,
  cluster = FALSE, s.tol = 1e-04, it.num = 8, pop.size = 3000)
```

Arguments

`data_aug`	a data.frame of the observed data after preprocessing. It should include be augmented with two new columns: `ph` for the enstimated propensity scores and `ghat` for the estimated conditional survival probabilities.
`ph`	propensity score estimates. For example, if the treatment is denoted by `A`, then `ph` should be P(A=1|X)
`p_level`	printing level
`regimeClass`	a formula indicating the form of treatment regimes
`Domains`	default is NULL. Otherwise, the object should be a `nvars *2` matrix used as the space of parameters, which will be supplied to `rgenoud::genoud`.
`cluster`	default is FALSE. This can also be an object of the 'cluster' class returned by one of the makeCluster commands in the parallel package or a vector of machine names so rgenoud::genoud can setup the cluster automatically.
`s.tol`	tolerance level for the GA algorithm. This is input for parameter `solution.tolerance` in function `rgenoud::genoud`.
`it.num`	the maximum GA iteration number
`pop.size`	an integer with the default set to be 3000. This is roughly the number individuals for the first generation in the genetic algorithm (`rgenoud::genoud`).

Examples

```
GenerateData <- function(n)
{
  x1 <- runif(n, min=-0.5,max=0.5)
  x2 <- runif(n, min=-0.5,max=0.5)
  error <- rnorm(n, sd= 1)
  ph <- rep(0.5,n)
  a <- rbinom(n = n, size = 1, prob=ph)
  c <- 1.5 +  + runif(n = n, min=0, max=2)
  cmplt_y <-  pmin(2+x1+x2 +  a*(1 - x1 - x2) +  (0.2 + a*(1+x1+x2)) * error, 4.4)
  censor_y <- pmin(cmplt_y, c)
  delta <- as.numeric(c > cmplt_y)
  return(data.frame(x1=x1,x2=x2,a=a, censor_y = censor_y, delta=delta))
}
n <- 100
data <- GenerateData(n)

# preprocessing
data_aug <- data
data_aug$ph <- rep(mean(data$a), n)
data_aug$deltaC <- 1 - data_aug$delta
library(survival)
survfit_all <- survfit(Surv(censor_y, event = deltaC)~1, data=data_aug)
survest <- stepfun(survfit_all$time, c(1, survfit_all$surv))
data_aug$ghat <- survest(data_aug$censor_y)

# estimate the mean-optimal treatment regime
meanopt_fit <- Gene_Mean_CenIPWE(data=data_aug, ph = data_aug$ph, p_level=1, regimeClass=a~x1*x2)
```

`Gene_Quantile_CenIPWE` *A low-level function for the generic optimization step in estimating Quanilte-optimal treatment regime for censored data*

Description

This function supports several user facing functions for Quantile-optimal treatment regime estimation, namely

`IPWE_Qopt_IndCen()`, `IPWE_Qopt_DTR_IndCen()`, `IPWE_Qopt_DepCen_trt()`, and `IPWE_Qopt_DepCen_general()`.

It implements the genetic algorithm based policy-search method with inverse probability weighting for censored data, such that the estimator is cube root consistent under the assumption that the propensity score model and the model for the survival distriution of the censoring time variable are both correct.

Usage

```
Gene_Quantile_CenIPWE(data_aug, tau, p_level, regimeClass,
  cluster = FALSE, s.tol = 1e-04, it.num = 8, pop.size = 5000,
  Domains = NULL, sign_beta1 = NULL, Penalty.level = 0)
```

Arguments

`data_aug`	a data.frame of the observed data after preprocessing. It should include be augmented with a new column: `epsi` for the composite weights.
`tau`	a value between 0 and 1. This is the quantile of interest.
`p_level`	choose between 0,1,2,3 to indicate different levels of output from the genetic function. Specifically, 0 (minimal printing), 1 (normal), 2 (detailed), and 3 (debug).
`regimeClass`	a formula specifying the class of treatment regimes to search, e.g. if `regimeClass = a~x1+x2`, and then this function will search the class of treatment regimes of the form $$d(x) = I\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0\right).$$ Polynomial arguments are also supported.
`cluster`	default is FALSE, meaning do not use parallel computing for the genetic algorithm(GA).
`s.tol`	tolerance level for the GA algorithm. This is input for parameter `solution.tolerance` in function `rgenoud::genoud`.
`it.num`	the maximum GA iteration number
`pop.size`	an integer with the default set to be 3000. This is roughly the number individuals for the first generation in the genetic algorithm (`rgenoud::genoud`).
`Domains`	default is NULL. Otherwise, the object should be a `nvars *2` matrix used as the space of parameters, which will be supplied to `rgenoud::genoud`.
`sign_beta1`	logical. FALSE if the coefficient for the first continuous variable is fixed to be negative one; TRUE if positive one.
`Penalty.level`	the level that determines which objective function to use. `Penalty.level = 0` indicates no regularization; `Penalty.level = 1` indicates the value function estimation minus the means absolute average coefficient is the output, which is useful trick to achieve uniqueness of estimated optimal TR when resolution of input response is low.

Examples

```
GenerateData <- function(n)
{
  x1 <- runif(n, min=-0.5,max=0.5)
  x2 <- runif(n, min=-0.5,max=0.5)
  error <- rnorm(n, sd= 1)
  ph <- rep(0.5,n)
  a <- rbinom(n = n, size = 1, prob=ph)
  c <- 1.5 +  + runif(n = n, min=0, max=2)
  cmplt_y <-  pmin(2+x1+x2 +  a*(1 - x1 - x2) +  (0.2 + a*(1+x1+x2)) * error, 4.4)
  censor_y <- pmin(cmplt_y, c)
  delta <- as.numeric(c > cmplt_y)
  return(data.frame(x1=x1,x2=x2,a=a, censor_y = censor_y, delta=delta))
}
n <- 100
data <- GenerateData(n)
```

```
# preprocessing
data_aug <- data
data_aug$ph <- rep(mean(data$a), n)
data_aug$deltaC <- 1 - data_aug$delta
library(survival)
survfit_all <- survfit(Surv(censor_y, event = deltaC)~1, data=data_aug)
survest <- stepfun(survfit_all$time, c(1, survfit_all$surv))
data_aug$ghat <- survest(data_aug$censor_y)
data_aug$epsi <- (data_aug$ph * data_aug$a + (1 - data_aug$ph) * (1 - data_aug$a)) * data_aug$ghat

# estimate the median-optimal treatment regime

quantopt_fit <- Gene_Quantile_CenIPWE(data_aug=data_aug,tau=0.5,
                                      p_level=1, regimeClass=a~x1+x2^2,
                                      sign_beta1=FALSE)
```

Gene_Quantile_CenIPWE_DTR

A low-level function for the generic optimization step in estimating dynamic Quanilte-optimal treatment regime for censored data

Description

This function supports wrapper functions for two stage Quantile-optimal treatment regime estimation, namely `IPWE_Qopt_DTR_IndCen`.

Usage

```
Gene_Quantile_CenIPWE_DTR(data, max, tau, regimeClass.stg1,
  regimeClass.stg2, s_Diff_Time, txVec1, txVec2, nvars.stg1, nvars.stg2,
  p.data1, p.data2, sign_beta1.stg1, sign_beta1.stg2, p_level, cluster,
  s.tol, it.num, pop.size, Domains1 = NULL, Domains2 = NULL,
  Penalty.level = 0)
```

Arguments

`data`	raw data.frame
`max`	Maximization (TRUE) or Minimizing (FALSE). Determines if genoud minimizes or maximizes the objective function.
`tau`	a quantile level of interest
`regimeClass.stg1`	the class of treatment regimes for stage one
`regimeClass.stg2`	the class of treatment regimes for stage two

`s_Diff_Time`	the length of time between the first stage treatment and the second stage treatment
`txVec1`	the vector of treatment received at the first stage
`txVec2`	the vector of treatment received at the second stage, it expects entries to be `NA` for patients who did not receive the second treatment
`nvars.stg1`	number of coeffients for the decision rule of the first stage
`nvars.stg2`	number of coeffients for the decision rule of the second stage
`p.data1`	the design matrix to be used for decision in stage one
`p.data2`	the design matrix to be used for decision in stage two
`sign_beta1.stg1`	Is sign of the coefficient for the first non-intercept variable for the first stage known? Default is NULL, meaning user does not have contraint on the sign; FALSE if the coefficient for the first continuous variable is fixed to be `-1`; TRUE if `1`. We can make the search space discrete because we employ $\|\beta_1\| = 1$ scale normalizaion.
`sign_beta1.stg2`	Default is NULL. Similar to `sign_beta1.stg1`.
`p_level`	choose between 0,1,2,3 to indicate different levels of output from the genetic function. Specifically, 0 (minimal printing), 1 (normal), 2 (detailed), and 3 (debug).
`cluster`	default is FALSE, meaning do not use parallel computing for the genetic algorithm(GA).
`s.tol`	tolerance level for the GA algorithm. This is input for parameter `solution.tolerance` in function `rgenoud::genoud`.
`it.num`	the maximum GA iteration number
`pop.size`	an integer with the default set to be 3000. This is roughly the number individuals for the first generation in the genetic algorithm (`rgenoud::genoud`).
`Domains1`	This is optional. If not NULL, please provide the two-column matrix for the searching range of coeffients in stage one. The coefficient taking value of positive/negative one should not be included.
`Domains2`	This is optional. If not NULL, please provide the two-column matrix for the searching range of coeffients in stage two. The coefficient taking value of positive/negative one should not be included.
`Penalty.level`	the level that determines which objective function to use. `Penalty.level = 0` indicates no regularization; `Penalty.level = 1` indicates the value function estimation minus the means absolute average coefficient is the output, which is useful trick to achieve uniqueness of estimated optimal TR when resolution of input response is low.

Examples

```
library(survival)
# Simulate data
n=200
```

```
s_Diff_Time = 1
D <- simJLSDdata(n, case="a")

# give regime classes
regimeClass.stg1 <- as.formula(a0~x0)
regimeClass.stg2 <- as.formula(a1~x1)

# extract columns that matches each stage's treatment regime formula
p.data1 <- model.matrix(regimeClass.stg1, D)

# p.data2 would only contain observations with non-null value.
p.data2 <- model.matrix(regimeClass.stg2, D)

txVec1 <- D[, "a0"]
txVec2 <- D[, "a1"]

# Eligibility flag
ELG <- (D$censor_y  >  s_Diff_Time)

# Build weights
D$deltaC <- 1 - D$delta
survfit_all <- survfit(Surv(censor_y, event = deltaC)~1, data=D)
survest <- stepfun(survfit_all$time, c(1, survfit_all$surv))
D$ghat <- survest(D$censor_y)
g_s_Diff_Time <- survest(s_Diff_Time)
D$w_di_vec <- rep(-999, n)
for(i in 1:n){
  if (!ELG[i]) {
      D$w_di_vec[i] <- 0.5 * D$ghat[i]} else {
         D$w_di_vec[i] <- 0.5* D$ghat[i] * 0.5
 }
}



fit1  <- Gene_Quantile_CenIPWE_DTR(data=D, max=TRUE,
  tau=0.3,
  regimeClass.stg1 = regimeClass.stg1,
  regimeClass.stg2 = regimeClass.stg2,
  s_Diff_Time = s_Diff_Time,
  txVec1 = txVec1,
  txVec2 = txVec2,
  nvars.stg1=2,
  nvars.stg2=2,
  p.data1=p.data1,
  p.data2=p.data2,
  sign_beta1.stg1=FALSE,
  sign_beta1.stg2=NULL,
  p_level=1,
  cluster=FALSE,
  s.tol=1e-6,
  it.num=5,
  pop.size=6000,
  Domains1 = NULL,
  Domains2 = NULL,
  Penalty.level = 0
  )
```

`IPWE_mean_IndCen` *Estimate the mean-optimal treatment regime for data with independently censored response*

Description

This function estimates the Mean-optimal Treatment Regime with censored response. The implemented function only works for scenarios in which treatment is binary and the censoring time is independent of baseline covariates, treatment group and all potential survival times.

Usage

```
IPWE_mean_IndCen(data, regimeClass, moPropen = "BinaryRandom",
  Domains = NULL, cluster = FALSE, p_level = 1, s.tol = 1e-04,
  it.num = 8, pop.size = 3000)
```

Arguments

`data`	a data.frame, containing variables in the `moPropen` and `RegimeClass` and also the response variables, namely `censor_y` as the censored response, and `delta` as the censoring indicator.
`regimeClass`	a formula specifying the class of treatment regimes to search, e.g. if `regimeClass = a~x1+x2`, and then this function will search the class of treatment regimes of the form $$d(x) = I\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0\right).$$ Polynomial arguments are also supported.
`moPropen`	The propensity score model for the probability of receiving treatment level 1. When `moPropen` equals the string "BinaryRandom", the proportion of observations receiving treatment level 1 in the sample will be plugged in as an estimate of the propensity. Otherwise, this argument should be a formula/string, based on which this function will fit a logistic regression on the treatment level. e.g. `a1~x1`.
`Domains`	default is NULL. Otherwise, the object should be a `nvars *2` matrix used as the space of parameters, which will be supplied to `rgenoud::genoud`. `nvars` is the total number of parameters.
`cluster`	default is FALSE, meaning do not use parallel computing for the genetic algorithm(GA).

`p_level`	choose between 0,1,2,3 to indicate different levels of output from the genetic function. Specifically, 0 (minimal printing), 1 (normal), 2 (detailed), and 3 (debug).
`s.tol`	tolerance level for the GA algorithm. This is input for parameter `solution.tolerance` in function `rgenoud::genoud`.
`it.num`	the maximum GA iteration number
`pop.size`	an integer with the default set to be 3000. This is roughly the number individuals for the first generation in the genetic algorithm (`rgenoud::genoud`).

Value

This function returns an object with 6 objects:

- `coefficients` the estimated parameter indexing the mean-optimal treatment regime. Since we focus the space of linear treatment regimes, the estimated decision rule cannot be uniquely identified without scale normalized. In this package, we normalized by $|\beta_1| = 1$, which was proposed in Horowitz (Horowitz 1992).
- `hatQ` the estimated optimal marginal mean response
- `moPropen` log of the input argument of `moPropen`
- `regimeClass` log of the input argument of `regimeClass`
- `data_aug` Training data with additional columns used in the algorithm. Note that `data_aug` is used for plotting of survival function of the censoring time
- `survfitCensorTime` the estimated survival function of the censoring time

References

Zhou Y (2018). *Quantile-Optimal Treatment Regimes with Censored Data*. Ph.D. thesis, University of Minnesota.

Horowitz JL (1992). “A smoothed maximum score estimator for the binary response model.” *Econometrica: journal of the Econometric Society*, 505–531.

Examples

```
GenerateData <- function(n)
{
  x1 <- runif(n, min=-0.5,max=0.5)
  x2 <- runif(n, min=-0.5,max=0.5)
  error <- rnorm(n, sd= 1)
  ph <- exp(-0.5+1*(x1+x2))/(1+exp(-0.5 + 1*(x1+x2)))
  a <- rbinom(n = n, size = 1, prob=ph)
  c <- 1.5 +  + runif(n = n, min=0, max=2)
  cmplt_y <-  pmin(2+x1+x2 +  a*(1 - x1 - x2) +  (0.2 + a*(1+x1+x2)) * error, 4.4)
  censor_y <- pmin(cmplt_y, c)
  delta <- as.numeric(c > cmplt_y)
  return(data.frame(x1=x1,x2=x2,a=a, censor_y = censor_y, delta=delta))
}
n <- 400
```

```
D <- GenerateData(n)
fit1 <- IPWE_mean_IndCen(data = D, regimeClass = a~x1+x2)
```

IPWE_Qopt_DepCen_general

Estimate Quantile-optimal Treatment Regime for covariates-dependent random censoring data

---

Description

This function estimates the Quantile-optimal Treatment Regime for a given quantile level of interest under the assumption that the distribution of censoring time is independent of the set of potential survival times given a set of baseline covariates and treatment actually received.

More specifically, we do stratification by treatment first and then used kernel smoothing to estimate local survival function of censoring time for each treatment group.

Usage

```
IPWE_Qopt_DepCen_general(data, regimeClass, tau, Domains = NULL, bw,
  moPropen = "BinaryRandom", DepCens = NULL, UseTrueG = FALSE,
  trueG_value = NULL, cluster = FALSE, p_level = 1, s.tol = 1e-05,
  it.num = 8, pop.size = 5000)
```

Arguments

`data`	raw data.frame
`regimeClass`	the class of treatment regimes. e.g., 'txname ~ x1+x2'.
`tau`	the quantile of interest
`Domains`	default is NULL.
`bw`	the bandwidth of local KM model (e.g. see Wang-wang 2008)
`moPropen`	an optional string for the working model of treatment assignment
`DepCens`	an optional vector of baseline variable names that the censoring variable depends on. Note that the treatment variable is always treated as dependent with the censoring time. If unspecified (`DepCens=NULL`), then all variables on the right side of `regimeClass` are used for `DepCens`
`UseTrueG`	logical. Whether the true survival probability of each patient is provided.
`trueG_value`	default is NULL. IF `UseTrueG=FALSE`, `trueG_value` should be `NULL`.
`cluster`	default is FALSE. This can also be an object of the 'cluster' class returned by one of the makeCluster commands in the parallel package or a vector of machine names so rgenoud::genoud can setup the cluster automatically.
`p_level`	print level
`s.tol`	tolerance level
`it.num`	the maximum iteration number
`pop.size`	the initial population size

Examples

```
GenerateData <- function(n)
{
  x1 <- runif(n, min=-0.5,max=0.5)
  x2 <- runif(n, min=-0.5,max=0.5)
  error <- rnorm(n, sd= 1)
  ph <- exp(-0.5+1*(x1+x2))/(1+exp(-0.5 + 1*(x1+x2)))
  a <- rbinom(n = n, size = 1, prob=ph)
  c <- 1.5 +  + runif(n = n, min=0, max=2)
  cmplt_y <-  pmin(2+x1+x2 +  a*(1 - x1 - x2) +  (0.2 + a*(1+x1+x2)) * error, 4.4)
  censor_y <- pmin(cmplt_y, c)
  delta <- as.numeric(c > cmplt_y)
  return(data.frame(x1=x1,x2=x2,a=a, censor_y = censor_y, delta=delta))
}




n <- 400
data <- GenerateData(n)
fit1 <- IPWE_Qopt_DepCen_general(data = data, regimeClass = a~x1+x2, moPropen = a~x1+x2,
                                 tau = 0.2, bw = 20/n,
                                 pop.size=3000, it.num = 3)
```

IPWE_Qopt_DepCen_trt — *Estimate the Quantile-opt Treatment Regime under the assumption that the censoring time's distribution only depends on treatment level*

Description

Here we assume the censoring variable is independent of covariates and potential outcomes given the treatment assignment. For example, if evidence shows that patients at certain treatment level are prone to experience censoring earlier.

Usage

```
IPWE_Qopt_DepCen_trt(data, regimeClass, tau, moPropen = "BinaryRandom",
  cluster = FALSE, p_level = 1, s.tol = 1e-04, it.num = 8,
  pop.size = 6000)
```

Arguments

`data`	raw data.frame
`regimeClass`	a formula specifying the class of treatment regimes to search, e.g. if `regimeClass = a~x1+x2`, and then this function will search the class of treatment regimes of the form $$d(x) = I(\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0).$$ Polynomial arguments are also supported.

`tau`	the quantile of interest
`moPropen`	The propensity score model for the probability of receiving treatment level 1. When `moPropen` equals the string "BinaryRandom", the proportion of observations receiving treatment level 1 in the sample will be plugged in as an estimate of the propensity. Otherwise, this argument should be a formula/string, based on which this function will fit a logistic regression on the treatment level. e.g. `a1~x1`.
`cluster`	default is FALSE, meaning do not use parallel computing for the genetic algorithm(GA).
`p_level`	choose between 0,1,2,3 to indicate different levels of output from the genetic function. Specifically, 0 (minimal printing), 1 (normal), 2 (detailed), and 3 (debug).
`s.tol`	tolerance level for the GA algorithm. This is input for parameter `solution.tolerance` in function `rgenoud::genoud`.
`it.num`	the maximum GA iteration number
`pop.size`	an integer with the default set to be 3000. This is roughly the number individuals for the first generation in the genetic algorithm (`rgenoud::genoud`).

Details

data is a dataframe that contains: a(observed treatment assignment), censor_y, and delta

Examples

```
GenerateData_DepCen_trt <- function(n)
{
  x1 <- runif(n, min=-0.5,max=0.5)
  x2 <- runif(n, min=-0.5,max=0.5)
  error <- rnorm(n, sd= 1)
  ph <- exp(-0.5+1*(x1+x2))/(1+exp(-0.5 + 1*(x1+x2)))
  a <- rbinom(n = n, size = 1, prob=ph)
  c <- 1 + 1*a + runif(n = n, min=0, max=2)
   # distribution of `c' depends on treatment level `a'
  cmplt_y <-  pmin(2+x1+x2 +  a*(1 - x1 - x2) +  (0.2 + a*(1+x1+x2)) * error, 4.4)
  censor_y <- pmin(cmplt_y, c)
  delta <- as.numeric(c > cmplt_y)
  return(data.frame(x1=x1,x2=x2,a=a, censor_y = censor_y, delta=delta))
}


n <- 400
data <- GenerateData_DepCen_trt(n)
fit1 <- IPWE_Qopt_DepCen_trt(data = data, regimeClass = a~x1+x2, moPropen = a~x1+x2,
                                 tau = 0.2)
```

IPWE_Qopt_DTR_IndCen *Function to estimate the two-stage quantile-optimal dynamic treatment regime for censored data: the independent censoring Case*

Description

This function inplements the estimator of two-stage quantile-optimal treatment regime with censored outcome by inverse probability of weighting, which is proposed in Chapter 3 of (Zhou 2018). We assume the censoring is independent of everything else, including the treatment covariates, and potential outcomes.

Specifically, we do grid search on the sign of the coefficient for the first non-intercept variables in stage 1 and stage 2 and apply genetic algorithm on the remaining coeffients simultaneously. So if stage one has d1 covariates excluding the intercept, stage two has d2, the resulting coefficient has dimension d1+d2+2.

Usage

```
IPWE_Qopt_DTR_IndCen(data, tau, regimeClass.stg1, regimeClass.stg2,
  s_Diff_Time = 1, moPropen1 = "BinaryRandom",
  moPropen2 = "BinaryRandom", sign_beta1.stg1 = NULL,
  sign_beta1.stg2 = NULL, Penalty.level = 0, s.tol = 1e-06,
  it.num = 4, max = TRUE, Domains1 = NULL, Domains2 = NULL,
  cluster = FALSE, p_level = 1, pop.size = 10000)
```

Arguments

`data`	a data.frame, containing variables in the `moPropen` and `RegimeClass` and also the response variables, namely `censor_y` as the censored response, and `delta` as the censoring indicator.
`tau`	a value between 0 and 1. This is the quantile of interest.
`regimeClass.stg1`	a formula specifying the class of treatment regimes for the first stage. For details of the general formulation of a linear treatment regime see `regimeClass` in `IPWE_Qopt_IndCen`.
`regimeClass.stg2`	a formula specifying the class of treatment regimes for the second stage
`s_Diff_Time`	Numeric. The fixed length of time between the first stage treatment and the second stage treatment
`moPropen1`	the first stage propensity score model. Default is "BinaryRandom".
`moPropen2`	the second stage propensity score model. Default is "BinaryRandom".
`sign_beta1.stg1`	Is sign of the coefficient for the first non-intercept variable for the first stage known? Default is NULL, meaning user does not have contraint on the sign; FALSE if the coefficient for the first continuous variable is fixed to be `-1`; TRUE if `1`. We can make the search space discrete because we employ $\|\beta_1\| = 1$ scale normalizaion.

`sign_beta1.stg2`	Default is NULL. Similar to `sign_beta1.stg1`.
`Penalty.level`	0: stop if the marginal quantiles cannot be further optimized; 1: continue the search among treatment regimes with with same value for the TR with the smallest intended proportion of treatment.
`s.tol`	tolerance level for the GA algorithm. This is input for parameter `solution.tolerance` in function `rgenoud::genoud`.
`it.num`	the maximum GA iteration number
`max`	logical. TRUE if the goal is maximization of the quantile. FALSE is the goal is minimization of the quantile.
`Domains1`	This is optional. If not NULL, please provide the two-column matrix for the searching range of coeffients in stage one. The coefficient taking value of positive/negative one should not be included.
`Domains2`	This is optional. If not NULL, please provide the two-column matrix for the searching range of coeffients in stage two. The coefficient taking value of positive/negative one should not be included.
`cluster`	default is FALSE, meaning do not use parallel computing for the genetic algorithm(GA).
`p_level`	choose between 0,1,2,3 to indicate different levels of output from the genetic function. Specifically, 0 (minimal printing), 1 (normal), 2 (detailed), and 3 (debug).
`pop.size`	an integer with the default set to be 3000. This is roughly the number individuals for the first generation in the genetic algorithm (`rgenoud::genoud`).

Details

In our setting, if a subject was censored or had experienced the event of interest before `s_Diff_Time` units of time had elapsed after the first stage of treatment, s/he would not be eligible to receive a second stage treatment.

Author(s)

Yu Zhou, `<zhou0269@umn.edu>`

References

Zhou Y (2018). *Quantile-Optimal Treatment Regimes with Censored Data*. Ph.D. thesis, University of Minnesota.

Examples

```
D <- simJLSDdata(400, case="a")
fit_2stage <-IPWE_Qopt_DTR_IndCen(data=D, tau= 0.3, regimeClass.stg1 = a0~x0,
                      regimeClass.stg2 = a1~x1,
                      sign_beta1.stg1 = FALSE,
                      sign_beta1.stg2 = FALSE)
```

`IPWE_Qopt_IndCen`	*Function to estimate the quantile-optimal treatment regime: the independent censoring Case*

Description

This function implements the estimation method proposed in Chapter 2 of (Zhou 2018). It estimates the quantile-optimal treatment regime for a given quantile level of interest from a single-stage clinical randomized experiment or a single-stage observational study under the independent censoring assumption. In other words, we estimate the parameters indexing the quantile-optimal treatment regime.

Our assumption of independent censoring means the distribution of the censoring time is the same conditional on baseline covariates, treatment group and the two potential survival times.

Usage

```
IPWE_Qopt_IndCen(data, regimeClass, tau, moPropen = "BinaryRandom",
  Domains = NULL, cluster = FALSE, p_level = 1, s.tol = 1e-04,
  it.num = 8, pop.size = 6000, sign_beta1 = NULL,
  Penalty.level = 0)
```

Arguments

`data` a data.frame, containing variables in the `moPropen` and `RegimeClass` and also the response variables, namely `censor_y` as the censored response, and `delta` as the censoring indicator.

`regimeClass` a formula specifying the class of treatment regimes to search, e.g. if `regimeClass = a~x1+x2`, and then this function will search the class of treatment regimes of the form

$$d(x) = I\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0\right).$$

Polynomial arguments are also supported.

`tau` a value between 0 and 1. This is the quantile of interest.

`moPropen` The propensity score model for the probability of receiving treatment level 1. When `moPropen` equals the string "BinaryRandom", the proportion of observations receiving treatment level 1 in the sample will be plugged in as an estimate of the propensity. Otherwise, this argument should be a formula/string, based on which this function will fit a logistic regression on the treatment level. e.g. `a1~x1`.

`Domains` default is NULL. Otherwise, the object should be a `nvars *2` matrix used as the space of parameters, which will be supplied to `rgenoud::genoud`.

`cluster`	default is FALSE, meaning do not use parallel computing for the genetic algorithm(GA).
`p_level`	choose between 0,1,2,3 to indicate different levels of output from the genetic function. Specifically, 0 (minimal printing), 1 (normal), 2 (detailed), and 3 (debug).
`s.tol`	tolerance level for the GA algorithm. This is input for parameter `solution.tolerance` in function `rgenoud::genoud`.
`it.num`	the maximum GA iteration number
`pop.size`	an integer with the default set to be 3000. This is roughly the number individuals for the first generation in the genetic algorithm (`rgenoud::genoud`).
`sign_beta1`	logical. Default is NULL. FALSE if the coefficient for the first continuous variable is fixed to be negative one; TRUE if positive one.
`Penalty.level`	0: stop if the marginal quantiles cannot be further optimized; 1: continue the search among treatment regimes with with same value for the TR with the smallest intended proportion of treatment.

Details

The input argument `data` is the dataframe that contains:

1. `a` observed treatment assignment
2. `censor_y` the censored response variable
3. `delta` the censoring indicator

The naming of these three columns should be strict.

Note that this function currently only works for scenarios in which treatment is binary.

References

Zhou Y (2018). *Quantile-Optimal Treatment Regimes with Censored Data*. Ph.D. thesis, University of Minnesota.

Horowitz JL (1992). "A smoothed maximum score estimator for the binary response model." *Econometrica: journal of the Econometric Society*, 505–531.

Examples

```
GenerateData <- function(n)
{
  x1 <- runif(n, min=-0.5,max=0.5)
  x2 <- runif(n, min=-0.5,max=0.5)
  error <- rnorm(n, sd= 1)
  ph <- exp(-0.5+1*(x1+x2))/(1+exp(-0.5 + 1*(x1+x2)))
  a <- rbinom(n = n, size = 1, prob=ph)
  c <- 1 + 1*a + runif(n = n, min=0, max=2)
  cmplt_y <-  pmin(2+x1+x2 +  a*(1 - x1 - x2) +  (0.2 + a*(1+x1+x2)) * error, 4.4)
  censor_y <- pmin(cmplt_y, c)
  delta <- as.numeric(c > cmplt_y)
  return(data.frame(x1=x1,x2=x2,a=a, censor_y = censor_y, delta=delta))
}
n <- 400

data <- GenerateData(n)
fit1 <- IPWE_Qopt_IndCen(data = data, regimeClass = a~x1+x2, tau=0.25)

# We can used the returned model to visualize the Kaplan-meier
# estimate of survival function of the censoring time variable,
# justified by the independent censoring assumption.
library(survminer)
ggsurvplot(fit1$survfitCensorTime, data=fit1$data_aug, risk.table = TRUE)
```

LocalKM *Kernel-based Local Kaplan-Meier Estimator*

Description

This is the local KM estimator customized for this library to run in batch mode. It returns the estimated conditional survival probabilities given a user specified set of covariate names that the survival time depends on, a.k.a $F(T > y_0 \mid x_0)$.

More specifically, for uncensored data points, we return `(1 - tauhat_func())` . If the observed data point is censored, then this function returns value -1 as a flag meaning we cannot .

Usage

```
LocalKM(D, bw, NamesCov)
```

Arguments

D	a data.frame with column `censor_y`, column `delta`, and additional covaraites.
bw	the bandwidth parameter
NamesCov	the vector of column names in data.frame `D` such that the survival time depends on.

Value

A vector of estimated conditional survival probability evaluated at the observed actual survival time on the same individual

Examples

```
GenerateData <- function(n)
{
  x1 <- runif(n, min=-0.5,max=0.5)
  x2 <- runif(n, min=-0.5,max=0.5)
  error <- rnorm(n, sd= 1)
  ph <- exp(-0.5+1*(x1+x2))/(1+exp(-0.5 + 1*(x1+x2)))
  a <- rbinom(n = n, size = 1, prob=ph)
  c <- 1.5 +  + runif(n = n, min=0, max=2)
  cmplt_y <-  pmin(2+x1+x2 +  a*(1 - x1 - x2) +  (0.2 + a*(1+x1+x2)) * error, 4.4)
  censor_y <- pmin(cmplt_y, c)
  delta <- as.numeric(c > cmplt_y)
  return(data.frame(x1=x1,x2=x2,a=a, censor_y = censor_y, delta=delta))
}
n <- 20
D <- GenerateData(n)
mean_hat <- LocalKM(D, 5, c("x1","x2"))
```

`simJLSDdata` *Function to generate simulation data from a sequentially randomized experiment designed in (Jiang et al. 2017)*

Description

Function to generate simulation data from a sequentially randomized experiment designed in (Jiang et al. 2017)

Usage

```
simJLSDdata(n, case = "a", s_Diff_Time = 1, C_max = 5,
  Censored = TRUE, fix_x0_value = NULL)
```

Arguments

`n`	sample size
`case`	string. One of "a", "b", "c", corresponding to three models.
`s_Diff_Time`	Numeric. Default is 1. This is the length of time between two stages of treatment
`C_max`	Numeric. Default is 5. This the upper bound of the uniform distribution of the censoring time variable. Changing this value shifts the overall censoring rate easily.
`Censored`	Boolean. Default is TRUE. Whether the data has censoring or not. If TRUE, all survival time would not be censored at all in the returned data.
`fix_x0_value`	Numeric. Default is Null. If supplied, it will generate simulated data with a fixed value, `fix_x0_value`, of the univariate baseline covarate.

Details

This generative model is proposed in (Jiang et al. 2017), Section 5, the second example. It uniformly defined three sets of conditional distributions of the survival times given the observable covariates at each stage within the same framework.

All three models satisfy the independent censoring assumption.

Value

This function returns a data.frame with simulated subject trajectories.

- `x0` the baseline covariate, always observable at relative time point 0;
- `a0` the observed first-stage treatment level at relative time point 0;
- `x1` an updated covariate observable to the relative time point `s_Diff_Time`, when the a second stage treatment is scheduled
- `a1` the observed second-stage treatment level at relative time point `s_Diff_Time`.

References

Jiang R, Lu W, Song R, Davidian M (2017). "On estimation of optimal treatment regimes for maximizing t-year survival probability." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **79**(4), 1165–1185.

Examples

```
dataA <- simJLSDdata(500,case="a")
dataB <- simJLSDdata(500,case="b")
dataC <- simJLSDdata(500,case="c")
```

`tauhat_func` *Kernel-based Local Kaplan-Meier Estimator for the Conditional Probability of the Survival Time*

Description

This function estimates the value of

$$F(T <= y_0 \mid x_0),$$

the conditional cumulative distribution function of a survival time T given covaraites vector x_0 at value y_0. This estimator is described in detail in (Wang and Wang 2009).

Usage

```
tauhat_func(y0, x0, z, x, delta, bw)
```

Arguments

`y0`	the vector of censored outcome of a single observation
`x0`	the vector of given covariate of a single observation
`z`	observed vector of response variable from observed data
`x`	the observed matrix of covariates, the dimension is # of observations by number of covariates. Note that the vector of ones should NOT be included in `x`.
`delta`	the vector of censoring indicators
`bw`	the scalar bandwidth parameter in kernel

Details

For cases with multivariate covariates, we adopted a product kernel. For example, in the bivariate case we use

$$K(x_1, x_2) = K_1(x_1)K_2(x_2),$$

where K_1 and K_2 are both biquadratickernel functions.

References

Wang HJ, Wang L (2009). "Locally weighted censored quantile regression." *Journal of the American Statistical Association*, **104**(487), 1117–1128.

Examples

```
tauhat_func(y0=10, x0=c(2,3), z=c(10, 12, 11),
           x=matrix(c(1,1,2,2,3,3), nrow=3, byrow=TRUE),
           delta=c(1,1,0), bw=10)
```

Index