Package 'depCensoring'

May 8, 2026

Type Package

Title Statistical Methods for Survival Data with Dependent Censoring

Version 0.1.10

Maintainer Negera Wakgari Deresa <negera.deresa@gmail.com>

Description Several statistical methods for analyzing survival data under various forms of dependent censoring are implemented in the package. In addition to accounting for dependent censoring, it offers tools to adjust for unmeasured confounding factors. The implemented approaches allow users to estimate the dependency between survival time and dependent censoring time, based solely on observed survival data. For more details on the methods, refer to Deresa and Van Keilegom (2021) <doi:10.1093/biomet/asaa095>, Czado and Van Keilegom (2023) <doi:10.1093/biomet/asac067>, Crommen et al. (2024) <doi:10.1007/s11749-023-00903-9>, Deresa and Van Keilegom (2024) <doi:10.1080/01621459.2022.2161387>, Willems et al. (2025) <doi:10.48550/arXiv.2403.11860>, Ding and Van Keilegom (2025) and D'Haen et al. (2025) <doi:10.1007/s10985-025-09647-0>.

Imports survival, foreach, parallel, doParallel, pbivnorm, stats, MASS, nleqslv, methods, Matrix, EnvStats, mvtnorm, rafalib, rvinecopulib, matrixcalc, nloptr, numDeriv, copula, R6, lubridate, splines2

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Suggests testthat (>= 3.0.0), rkriging, orthopolynom, OpenMx, stringr, LaplacesDemon

Config/testthat/edition 3

NeedsCompilation no

Author Ilias Willems [aut] (ORCID: <https://orcid.org/0009-0001-9463-9942>), Gilles Crommen [aut] (ORCID: <https://orcid.org/0000-0001-8380-1900>), Negera Wakgari Deresa [aut, cre] (ORCID: <https://orcid.org/0000-0002-1302-3725>), Jie Ding [aut] (ORCID: <https://orcid.org/0000-0002-6083-7529>), Claudia Czado [aut] (ORCID: <https://orcid.org/0000-0002-6329-5438>),

Myrthe D'Haen [aut] (ORCID: <https://orcid.org/0000-0003-0223-0267>),
Ingrid Van Keilegom [aut] (ORCID:
<https://orcid.org/0000-0001-8827-7642>)

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2026-04-13 18:10:08 UTC

Contents

`Chronometer` *Chronometer object*

Description

R6 object that mimics a chronometer. It can be started, paused, record legs and stopped.

Methods

Public methods:

- `Chronometer$show()`
- `Chronometer$reset()`
- `Chronometer$start()`
- `Chronometer$stop()`
- `Chronometer$record.leg()`
- `Chronometer$get.chronometer.data()`
- `Chronometer$get.total.time()`
- `Chronometer$accumulate.legs()`

- `Chronometer$clone()`

Method `show()`**:** Display the values stored in this chronometer object.

Usage:

```
Chronometer$show()
```

Method `reset()`**:** Reset the chronometer.

Usage:

```
Chronometer$reset()
```

Method `start()`**:** Start the chronometer

Usage:

```
Chronometer$start()
```

Method `stop()`**:** Stop the chronometer

Usage:

```
Chronometer$stop(leg.name = NULL)
```

Arguments:

`leg.name` (optional) Name for the stopped leg.

Method `record.leg()`**:** Record a leg time. The chronometer will continue running.

Usage:

```
Chronometer$record.leg(leg.name = NULL)
```

Arguments:

`leg.name` Name for the recorded leg.

Method `get.chronometer.data()`**:** Like `show` method, but more rudimentary.

Usage:

```
Chronometer$get.chronometer.data()
```

Method `get.total.time()`**:** Return the total time span between start and stop.

Usage:

```
Chronometer$get.total.time(force = FALSE)
```

Arguments:

`force` Boolean variable. If `TRUE`, avoids error when calling this function while chronometer has not been stopped yet.

Method `accumulate.legs()`**:** Return total time spent per leg category (using leg names).

Usage:

```
Chronometer$accumulate.legs(force = FALSE)
```

Arguments:

`force` force Boolean variable. If `TRUE`, avoids error when calling this function while chronometer has not been stopped yet.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Chronometer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`estimate.cmprsk` *Estimate the competing risks model of Willems et al. (2025).*

Description

This function estimates the parameters in the competing risks model described in Willems et al. (2025). Note that this model extends the model of Crommen, Beyhum and Van Keilegom (2024) and as such, this function also implements their methodology.

Usage

```
estimate.cmprsk(
  data,
  admin,
  conf,
  eoi.indicator.names = NULL,
  Zbin = NULL,
  inst = "cf",
  realV = NULL,
  compute.var = TRUE,
  eps = 0.001
)
```

Arguments

`data` A data frame, adhering to the following formatting rules:

- The first column, named `"y"`, contains the observed times.
- The next columns, named `"delta1"`, `delta2`, etc. contain the indicators for each of the competing risks.
- The next column, named `da`, contains the censoring indicator (independent censoring).
- The next column should be a column of all ones (representing the intercept), names `x0`.
- The subsequent columns should contain the values of the covariates, named `x1`, `x2`, etc.
- When applicable, the next column should contain the values of the endogenous variable. This column should be named `z`.
- When `z` is provided and an instrument for `z` is available, the next column, named `w`, should contain the values for the instrument.

admin — Boolean value indicating whether the data contains administrative censoring.

conf — Boolean value indicating whether the data contains confounding and hence indicating the presence of z and, possibly, w.

eoi.indicator.names — Vector of names of the censoring indicator columns pertaining to events of interest. Events of interest will be modeled allowing dependence between them, whereas all censoring events (corresponding to indicator columns not listed in eoi.indicator.names) will be treated as independent of every other event. If eoi.indicator.names == NULL, all events will be modeled dependently.

Zbin — Indicator value indicating whether (Zbin = TRUE) or not Zbin = FALSE the endogenous covariate is binary. Default is Zbin = NULL, corresponding to the case when conf == FALSE.

inst — Variable encoding which approach should be used for dealing with the confounding. inst = "cf" indicates that the control function approach should be used. inst = "W" indicates that the instrumental variable should be used 'as is'. inst = "None" indicates that Z will be treated as an exogenous covariate. Finally, when inst = "oracle", this function will access the argument realV and use it as the values for the control function. Default is inst = "cf".

realV — Vector of numerics with length equal to the number of rows in data. Used to provide the true values of the instrumental function to the estimation procedure.

compute.var — Boolean value indicating whether the variance of the parameter estimates should be computed as well (this can be very computationally intensive, so may want to be disabled). Default is estimate.var = TRUE.

eps — Value that will be added to the diagonal of the covariance matrix during estimation in order to ensure strictly positive variances.

Value

A list of parameter estimates in the second stage of the estimation algorithm (hence omitting the estimates for the control function), as well as an estimate of their variance and confidence intervals.

References

Willems et al. (2025). Flexible control function approach under competing risks (submitted).

Crommen, G., Beyhum, J., and Van Keilegom, I. (2024). An instrumental variable approach under dependent censoring. Test, 33(2), 473-495.

Examples

```
n <- 200

# Set parameters
gamma <- c(1, 2, 1.5, -1)
theta <- c(0.5, 1.5)
eta1 <- c(1, -1, 2, -1.5, 0.5)
eta2 <- c(0.5, 1, 1, 3, 0)
```

```
# Generate exogenous covariates
x0 <- rep(1, n)
x1 <- rnorm(n)
x2 <- rbinom(n, 1, 0.5)

# Generate confounder and instrument
w <- rnorm(n)
V <- rnorm(n, 0, 2)
z <- cbind(x0, x1, x2, w) %*% gamma + V
realV <- z - (cbind(x0, x1, x2, w) %*% gamma)

# Generate event times
err <- MASS::mvrnorm(n, mu = c(0, 0), Sigma =
matrix(c(3, 1, 1, 2), nrow = 2, byrow = TRUE))
bn <- cbind(x0, x1, x2, z, realV) %*% cbind(eta1, eta2) + err
Lambda_T1 <- bn[,1]; Lambda_T2 <- bn[,2]
x.ind = (Lambda_T1>0)
y.ind <- (Lambda_T2>0)
T1 <- rep(0,length(Lambda_T1))
T2 <- rep(0,length(Lambda_T2))
T1[x.ind] = ((theta[1]*Lambda_T1[x.ind]+1)^(1/theta[1])-1)
T1[!x.ind] = 1-(1-(2-theta[1])*Lambda_T1[!x.ind])^(1/(2-theta[1]))
T2[y.ind] = ((theta[2]*Lambda_T2[y.ind]+1)^(1/theta[2])-1)
T2[!y.ind] = 1-(1-(2-theta[2])*Lambda_T2[!y.ind])^(1/(2-theta[2]))
# Generate adminstrative censoring time
C <- runif(n, 0, 40)

# Create observed data set
y <- pmin(T1, T2, C)
delta1 <- as.numeric(T1 == y)
delta2 <- as.numeric(T2 == y)
da <- as.numeric(C == y)
data <- data.frame(cbind(y, delta1, delta2, da, x0, x1, x2, z, w))
colnames(data) <- c("y", "delta1", "delta2", "da", "x0", "x1", "x2", "z", "w")

# Estimate the model
admin <- TRUE                # There is administrative censoring in the data.
conf <- TRUE                 # There is confounding in the data (z)
eoi.indicator.names <- NULL  # We will not impose that T1 and T2 are independent
Zbin <- FALSE                # The confounding variable z is not binary
inst <- "cf"                 # Use the control function approach
compute.var <- TRUE          # Variance of estimates should be computed.
# Since we don't use the oracle estimator, this argument is ignored anyway
realV <- NULL
estimate.cmprsk(data, admin, conf, eoi.indicator.names, Zbin, inst, realV,
                compute.var)
```

`fitDepCens` *Fit Dependent Censoring Models*

Description

This function allows to estimate the dependency parameter along all other model parameters. First, estimates the cumulative hazard function, and then at the second stage it estimates other model parameters assuming that the cumulative hazard function is known. The details for implementing the dependent censoring methodology can be found in Deresa and Van Keilegom (2024).

Usage

```
fitDepCens(
  resData,
  X,
  W,
  cop = c("Frank", "Gumbel", "Normal"),
  dist = c("Weibull", "lognormal"),
  start = NULL,
  n.iter = 50,
  bootstrap = TRUE,
  n.boot = 150,
  ncore = 7,
  eps = 1e-04
)
```

Arguments

`resData`	Data matrix with three columns; Z = the observed survival time, d1 = the censoring indicator of T and d2 = the censoring indicator of C.
`X`	Data matrix with covariates related to T.
`W`	Data matrix with covariates related to C. First column of W should be a vector of ones.
`cop`	Which copula should be computed to account for dependency between T and C. This argument can take one of the values from `c("Gumbel", "Frank", "Normal")`.
`dist`	The distribution to be used for the censoring time C. Only two distributions are allowed, i.e, Weibull and lognormal distributions. With the value `"Weibull"` as the default.
`start`	Initial values for the finite dimensional parameters. If `start` is NULL, the initial values will be obtained by fitting a Cox model for survival time T and a Weibull model for dependent censoring C.
`n.iter`	Number of iterations; the default is `n.iter = 50`. The larger the number of iterations, the longer the computational time.
`bootstrap`	A boolean indicating whether to compute bootstrap standard errors for making inferences.

n.boot	Number of bootstrap samples to use in the estimation of bootstrap standard errors if bootstrap = TRUE. The default is n.boot = 150. But, higher values of n.boot are recommended for obtaining good estimates of bootstrap standard errors.
ncore	The number of cores to use for parallel computation in bootstrapping, with the default ncore = 7.
eps	Convergence error. This is set by the user in such away that the desired convergence is met; the default is eps = 1e-4.

Value

This function returns a fit of dependent censoring model; parameter estimates, estimate of the cumulative hazard function, bootstrap standard errors for finite-dimensional parameters, the nonparametric cumulative hazard function, etc.

References

Deresa and Van Keilegom (2024). Copula based Cox proportional hazards models for dependent censoring, Journal of the American Statistical Association, 119:1044-1054.

Examples

```
# Toy data example to illustrate implementation
n = 300
beta = c(0.5)
lambd = 0.35
eta = c(0.9,0.4)
X = cbind(rbinom(n,1,0.5))
W = cbind(rep(1,n),rbinom(n,1,0.5))
# generate dependency structure from Frank
frank.cop <- copula::frankCopula(param = 5,dim = 2)
U = copula::rCopula(n,frank.cop)
T1 = (-log(1-U[,1]))/(lambd*exp(X*beta))           # Survival time
T2 = (-log(1-U[,2]))^(1.1)*exp(W%*%eta)            # Censoring time
A = runif(n,0,15)                                  # administrative censoring time
Z = pmin(T1,T2,A)
d1 = as.numeric(Z==T1)
d2 = as.numeric(Z==T2)
resData = data.frame("Z" = Z,"d1" = d1, "d2" = d2)   # should be data frame
colnames(W) <- c("ones","cov1")
colnames(X) <- "cov.surv"

# Fit dependent censoring model

fit <- fitDepCens(resData = resData, X = X, W = W, bootstrap = FALSE)

# parameter estimates

fit$parameterEstimates

# summary fit results
```

```
summary(fit)

# plot cumulative hazard vs time

plot(fit$observedTime, fit$cumhazardFunction, type = "l",xlab = "Time",
ylab = "Estimated cumulative hazard function")
```

`fitIndepCens` *Fit Independent Censoring Models*

Description

This function allows to estimate all model parameters under the assumption of independent censoring. First, estimates the cumulative hazard function, and then at the second stage it estimates other model parameters assuming that the cumulative hazard is known.

Usage

```
fitIndepCens(
  resData,
  X,
  W,
  dist = c("Weibull", "lognormal"),
  start = NULL,
  n.iter = 50,
  bootstrap = TRUE,
  n.boot = 150,
  ncore = 7,
  eps = 1e-04
)
```

Arguments

`resData`	Data matrix with three columns; Z = the observed survival time, d1 = the censoring indicator of T and d2 = the censoring indicator of C.
`X`	Data matrix with covariates related to T.
`W`	Data matrix with covariates related to C. First column of W should be ones.
`dist`	The distribution to be used for the censoring time C. Only two distributions are allowed, i.e, Weibull and lognormal distributions. With the value `"Weibull"` as the default.
`start`	Initial values for the finite dimensional parameters. If `start` is NULL, the initial values will be obtained by fitting a Cox model for survival time T and a Weibull model for censoring time C.
`n.iter`	Number of iterations; the default is `n.iter = 50`. The larger the number of iterations, the longer the computational time.

bootstrap	A boolean indicating whether to compute bootstrap standard errors for making inferences.
n.boot	Number of bootstrap samples to use in the estimation of bootstrap standard errors if bootstrap = TRUE. The default is n.boot = 150. But, higher values of n.boot are recommended for obtaining good estimates of bootstrap standard errors.
ncore	The number of cores to use for parallel computation is configurable, with the default ncore = 7.
eps	Convergence error. This is set by the user in such away that the desired convergence is met; the default is eps = 1e-4.

Value

This function returns a fit of independent censoring model; parameter estimates, estimate of the cumulative hazard function, bootstrap standard errors for finite-dimensional parameters, the nonparametric cumulative hazard function, etc.

Examples

```
# Toy data example to illustrate implementation
n = 300
beta = c(0.5)
lambd = 0.35
eta = c(0.9,0.4)
X = cbind(rbinom(n,1,0.5))
W = cbind(rep(1,n),rbinom(n,1,0.5))
# generate dependency structure from Frank
frank.cop <- copula::frankCopula(param = 5,dim = 2)
U = copula::rCopula(n,frank.cop)
T1 = (-log(1-U[,1]))/(lambd*exp(X*beta))                  # Survival time'
T2 = (-log(1-U[,2]))^(1.1)*exp(W%*%eta)                   # Censoring time
A = runif(n,0,15)                                        # administrative censoring time
Z = pmin(T1,T2,A)
d1 = as.numeric(Z==T1)
d2 = as.numeric(Z==T2)
resData = data.frame("Z" = Z,"d1" = d1, "d2" = d2)     # should be data frame

colnames(W) <- c("ones","cov1")
colnames(X) <- "cov.surv"

# Fit independent censoring model

fitI <- fitIndepCens(resData = resData, X = X, W = W, bootstrap = FALSE)

# parameter estimates

fitI$parameterEstimates

# summary fit results
summary(fitI)
```

```
# plot cumulative hazard vs time

 plot(fitI$observedTime, fitI$cumhazardFunction, type = "l",xlab = "Time",
 ylab = "Estimated cumulative hazard function")
```

`liver` *Liver cirrhosis data set.*

Description

End stage liver disease data set, as for example analyzed by D'Haen et al. (2025).

Usage

```
liver
```

Format

A data frame with 281 rows and 7 variables:

patient patient ID.

time Survival time.

status Censoring indicator.

age Age of patient.

gender Gender of patient (1 - male, 0 - female).

bmi Body mass index.

ukeld UK End-stage Liver Disease score. Higher scores indicate more severe disease state.

`NonParTrans` *Fit a semiparametric transformation model for dependent censoring*

Description

This function allows to estimate the dependency parameter along all other model parameters. First, estimates a non-parametric transformation function, and then at the second stage it estimates other model parameters assuming that the non-parametric function is known. The details for implementing the dependent censoring methodology can be found in Deresa and Van Keilegom (2021).

Usage

```
NonParTrans(
  resData,
  X,
  W,
  start = NULL,
  n.iter = 15,
  bootstrap = FALSE,
  n.boot = 50,
  eps = 0.001
)
```

Arguments

resData	Data matrix with three columns; Z = the observed survival time, d1 = the censoring indicator of T and d2 = the censoring indicator of C.
X	Data matrix with covariates related to T
W	Data matrix with covariates related to C
start	Initial values for the finite dimensional parameters. If start is NULL, the initial values will be obtained by fitting an Accelerated failure time models.
n.iter	Number of iterations; the default is n.iter = 20. The larger the number of iterations, the longer the computational time.
bootstrap	A boolean indicating whether to compute bootstrap standard errors for making inferences.
n.boot	Number of bootstrap samples to use in the estimation of bootstrap standard errors if bootstrap = TRUE. The default is n.boot = 50. But, higher values of n.boot are recommended for obtaining good estimates of bootstrap standard errors.
eps	Convergence error. This is set by the user in such away that the desired convergence is met; the default is eps = 1e-3.

Value

This function returns a fit of a semiparametric transformation model; parameter estimates, estimate of the non-parametric transformation function, bootstrap standard errors for finite-dimensional parameters, the nonparametric cumulative hazard function, etc.

References

Deresa, N. and Van Keilegom, I. (2021). On semiparametric modelling, estimation and inference for survival data subject to dependent censoring, Biometrika, 108, 965–979.

Examples

```
# Toy data example to illustrate implementation
n = 300
beta = c(0.5, 1); eta = c(1,1.5); rho = 0.70
```

```
sigma = matrix(c(1,rho,rho,1),ncol=2)
err = MASS::mvrnorm(n, mu = c(0,0) , Sigma=sigma)
err1 = err[,1]; err2 = err[,2]
x1 = rbinom(n,1,0.5); x2 = runif(n,-1,1)
X = matrix(c(x1,x2),ncol=2,nrow=n); W = X   # data matrix
T1 = X%*%beta+err1
C =  W%*%eta+err2
T1 = exp(T1); C = exp(C)
A = runif(n,0,8); Y = pmin(T1,C,A)
d1 = as.numeric(Y==T1)
d2 = as.numeric(Y==C)
resData = data.frame("Z" = Y,"d1" = d1, "d2" = d2)   # should be data frame
colnames(X) = c("X1", "X2")
colnames(W) = c("W1","W2")

#  Bootstrap is false by default
output = NonParTrans(resData = resData, X = X, W = W, n.iter = 2)
output$parameterEstimates
```

`ParamCop` *Estimation of a parametric dependent censoring model without covariates.*

Description

Note that it is not assumed that the association parameter of the copula function is known, unlike most other papers in the literature. The details for implementing the methodology can be found in Czado and Van Keilegom (2023).

Usage

```
ParamCop(Y, Delta, Copula, Dist.T, Dist.C, start = c(1, 1, 1, 1))
```

Arguments

`Y`	Follow-up time.
`Delta`	Censoring indicator.
`Copula`	The copula family. This argument can take values from `c("frank","gumbel","clayton","gaussian",`
`Dist.T`	The distribution to be used for the survival time T. This argument can take one of the values from `c("lnorm", "weibull", "llogis")`.
`Dist.C`	The distribution to be used for the censoring time C. This argument can take one of the values from `c("lnorm", "weibull", "llogis")`.
`start`	Starting values.

Value

A table containing the minimized negative log-likelihood using the independence copula model, the estimated parameter values for the model with the independence copula, the minimized negative log-likelihood using the specified copula model and the estimated parameter values for the model with the specified copula.

References

Czado and Van Keilegom (2023). Dependent censoring based on parametric copulas. Biometrika, 110(3), 721-738.

Examples

```
tau = 0.75
Copula = "frank"
Dist.T = "weibull"
Dist.C = "lnorm"
par.T = c(2,1)
par.C = c(1,2)
n=1000

simdata<-TCsim(tau,Copula,Dist.T,Dist.C,par.T,par.C,n)

Y = simdata[[1]]
Delta = simdata[[2]]

ParamCop(Y,Delta,Copula,Dist.T,Dist.C)
```

---

pi.surv *Estimate the model of Willems et al. (2025).*

Description

This function estimates bounds on the coefficients the single- index model $\Lambda(x^\top \beta(t))$ for the conditional cumulative distribution function of the event time.

Usage

```
pi.surv(
  data,
  idx.param.of.interest,
  idxs.c,
  t,
  par.space,
  search.method = "GS",
  add.options = list(),
  verbose = 0,
```

```
  picturose = FALSE,
  parallel = FALSE
)
```

Arguments

`data`	Data frame containing the data on which to fit the model. The columns should be named as follows: 'Y' = observed timed, 'Delta' = censoring indicators, 'X0' = intercept column, 'X1' - 'Xp' = covariates.
`idx.param.of.interest`	Index of element in the covariate vector for which the identified interval should be estimated. It can also be specified as `idx.param.of.interest = "all"`, in which case identified intervals will be computed for all elements in the parameter vector. Note that `idx.param.of.interest = 1` corresponds to the intercept parameter.
`idxs.c`	Vector of indices of the continuous covariates. Suppose the given data contains 5 covariates, of which 'X2' and 'X5' are continuous, this argument should be specified as `idxs.c = c(2, 5)`.
`t`	Time point for which to estimate the identified set of $\beta(t)$.
`par.space`	Matrix containing bounds on the space of the parameters. The first column corresponds to lower bounds, the second to upper bounds. The i'th row corresponds to the bounds on the i'th element in the parameter vector.
`search.method`	The search method to be used to find the identified interval. Default is `search.method = "GS"`.
`add.options`	List of additional options to be specified to the method. Notably, it can be used to select the link function $\Lambda(t))$ that should be considered. Currently, the link function leading to an accelerated failure time model (`"AFT_ll"`, default) and the link function leading to a Cox proportional hazards model (`"Cox_wb"`) are implemented. Other options can range from 'standard' hyperparameters such as the confidence level of the test and number of instrumental functions to be used, to technical hyperparameters regarding the search method and test implementation. General hyperparameters: **cov.ranges:** known bounds on each of the covariates in the data set. **norm.func.name:** Name of the normalization function to be used. Can be either "normalize.covariates1" or "normalize.covariates2" (default). The former is a simple elementwise rescaling. The latter uses the PCA approach. **inst.func.family:** Family of instrumental functions to be used for all covariates. Options are "box", "spline" and "cd". The former two are only applicable for continuous covariates. The latter can also handle discrete covariates. Default is "cd". **G.c:** The class of instrumental functions used for the continuous covariates in the model, in case "cd" is selected as `inst.func.family:`. Options are "box" and "spline". Default is "spline". **degree:** The degree of the B-spline functions, should they be used as instrumental functions for the continuous covariates. Default is 3.

link.function: Name of the link function to be used. Options are "AFT_ll" for the AFT model with log-logistic baseline, or "Cox_wb" for the Cox PH model (originally with Weibull baseline, but now for a general) baseline hazard).

K.bar: Number of refinement steps when obtaining the critical value. See Bei (2024).

B: Number of bootstrap samples to be used when obtaining the bootstrap distribution of the test statistic.

ignore.empty.IF: Boolean value indicating whether instrumental functions with empty support should be ignored. Default is FALSE. The feature `ignore.empty.IF = TRUE` is experimental, so there might exist edge cases for which the implementation will fail to run.

c: User-specified projection vector. If supplied, the method will estimate the identified interval of $c^\top \beta$ as opposed to the identified interval of β_k, where k is specified as the argument `idx.param.of.interest`.

Hyperparameters specific to the EAM implementation:

min.dist/max.dist: The minimum/maximum distance of sampled points from the current best value for the coefficient of interest.

min.eval/max.eval: The minimum/maximum number of points evaluated in the initial feasible point search.

nbr.init.sample.points: The total number of drawn points required in the initial drawing process.

nbr.init.unif: The total number of uniformly drawn points in the initial set of starting values.

nbr.points.per.iter.init: Number of points sampled per iteration in the initial drawing process.

nbr.start.vals: Number of starting values for which to run the optimization algorithm for the expected improvement.

nbr.opt.EI: Number of optimal theta values found by the optimization algorithm to return.

nbr.extra: Number of extra randomly drawn points to add to the set of optimal theta values (to be supplied to the next E-step).

min.improvement: Minimum amount that the current best root of the violation curve should improve by wrt. the its previous value.

min.possible.improvement: Minimum amount that the next iteration should be able to improve upon the current best value of the root.

EAM.min.iter: Minimum amount of EAM iterations to run.

max.iter: Maximum amount of EAM iterations to run.

Hyperparameters specific to the gridsearch implementation:

min.eval/max.eval: Minimum and maximum number of evaluations.

next.gs.point: Function that determines the next point in the grid search sequence.

step.size: Step size of the grid.

bin.search.tol: Binary search tolerance.

max.iter: Maximum number of iterations that the algorithm can run.

Other (hidden) options can also be overwritten, though we highly discourage this. If necessary, you can consult the source code of this functions to find the names of the desired parameters and add their name alongside their desired value as an entry in `options` (e.g. `options$min.var <- 1e-4`. Again, not recommended!).

`verbose` Verbosity level. The higher the value, the more verbose the method will be. Default is `verbose = 0`.

`picturose` Picturosity flag. If `TRUE`, a plot illustrating the workings of the algorithm will updated during runtime. Default is `picturose = FALSE`.

`parallel` Flag for whether or not parallel computing should be used. Default is `parallel = FALSE`. When `parallel = TRUE`, this implementation will use `min(detectCores() - 1, 10)` cores to construct the parallel back-end.

Value

Matrix containing the identified intervals of the specified coefficients, as well as corresponding convergence information of the estimation algorithm.

References

Willems, I., Beyhum, J. and Van Keilegom, I. (2025). Partial identification for a class of survival models under dependent censoring. (Submitted).

Examples

```
# Clear workspace
rm(list = ls())

# Load the survival package
library(survival)

# Set random seed
set.seed(123)

# Load and preprocess data
data <- survival::lung
data[, "intercept"] <- rep(1, nrow(data))
data[, "status"] <- data[, "status"] - 1
data <- data[, c("time", "status", "intercept", "age", "sex")]
colnames(data) <- c("Y", "Delta", "X0", "X1", "X2")

# Standardize age variable
data[, "X1"] <- scale(data[, "X1"])

## Example:
## - Link function: AFT link function (default setting)
## - Number of IF: 5 IF per continuous covariate (default setting)
## - Search method: Binary search
```

```
## - Type of IF: Cubic spline functions for continuous covariate, indicator
##   function for discrete covariate (default setting).

# Settings for main estimation function
idx.param.of.interest <- 2 # Interest in effect of age
idxs.c <- 1                # X1 (age) is continuous
t <- 200                   # Model imposed at t = 200
search.method <- "GS"      # Use binary search
par.space <- matrix(rep(c(-10, 10), 3), nrow = 3, byrow = TRUE)
add.options <- list()
picturose <- TRUE
parallel <- FALSE

# Estimate the identified intervals
pi.surv(data, idx.param.of.interest, idxs.c, t, par.space,
        search.method = search.method, add.options = add.options,
        picturose = picturose, parallel = parallel)
```

`QRdepCens` *Estimate the model of D'Haen et al. (2025).*

Description

This function estimates the parameters in the model of D'Haen et al. (2025).

Usage

```
QRdepCens(data, hp, var.estimate = FALSE, verbose = TRUE)
```

Arguments

`data`	Data on which the model should be estimated. Note that the data should be structured in a specific form: The observed times should be put in a column named `"Y"`, the censoring indicators in a column named `"Delta"`, and the covariates in columns named `"X1"`, `"X2"`, ... (in increasing order). The given data set cannot contain any other columns.
`hp`	List of hyperparameters to be used, the elements of which will overwrite the default settings. In particular, consider changing: **test_cop_name:** Copula to be used. Should be one of `"frank"`, `"gumbel"`, `"clayton"` or `"indep"`. **homoscedastic:** Boolean flag indicating whether homoscedasticity can be assumed. **variance.bootstrap.iterations:** Number of bootstrap resamples to use during variance estimation. Consider increase if more precision is needed; consider decreasing to reduce computation time.

Other hyperparameters can be changed though it is not recommended. We refer to the source code for the available options.

`var.estimate` Boolean value indicating whether the variance should be estimated (via bootstrap). This can take a considerable amount of time. Default is `var.estimate = FALSE`.

`verbose` Verbosity flag (boolean) indicating whether the results should be printed to the console. Default is `verbose = TRUE`.

Note

The variance estimation procedure could easily be paralelized. However, this is currently not implemented.

References

D'Haen, M., Van Keilegom, I. and Verhasselt, A. (2025). Quantile regression under dependent censoring with unknown association. Lifetime Data Analysis 31(2):253-299.

Examples

```
# Load the data
data(liver)

# Give standard column names (required!)
colnames(liver) <- c("patient", "Y", "Delta", "X1", "X2", "X3", "X4")
liver <- liver[, c(-1, -6, -7)]

# Run the model
hp <- list(
  homoscedastic = FALSE,
  test_cop_name = "frank"
)
QRdepCens(liver, hp, var.estimate = FALSE)
# Takes a while if var.estimate = TRUE...
```

`SolveL` *Cumulative hazard function of survival time under dependent censoring*

Description

This function estimates the cumulative hazard function of survival time (T) under dependent censoring (C). The estimation makes use of the estimating equations derived based on martingale ideas.

Usage

```
SolveL(
  theta,
  resData,
  X,
  W,
  cop = c("Frank", "Gumbel", "Normal"),
  dist = c("Weibull", "lognormal")
)
```

Arguments

theta	Estimated parameter values/initial values for finite dimensional parameters
resData	Data matrix with three columns; Z = the observed survival time, d1 = the censoring indicator of T and d2 = the censoring indicator of C.
X	Data matrix with covariates related to T
W	Data matrix with covariates related to C. First column of W should be ones
cop	Which copula should be computed to account for dependency between T and C. This argument can take one of the values from `c("Gumbel", "Frank", "Normal")`. The default copula model is "Frank".
dist	The distribution to be used for the dependent censoring C. Only two distributions are allowed, i.e, Weibull and lognormal distributions. With the value `"Weibull"` as the default.

Value

This function returns an estimated hazard function, cumulative hazard function and distinct observed survival times;

Examples

```
n = 200
beta = c(0.5)
lambd = 0.35
eta = c(0.9,0.4)
X = cbind(rbinom(n,1,0.5))
W = cbind(rep(1,n),rbinom(n,1,0.5))
frank.cop <- copula::frankCopula(param = 5,dim = 2)
U = copula::rCopula(n,frank.cop)
T1 = (-log(1-U[,1]))/(lambd*exp(X*beta))         # Survival time'
T2 = (-log(1-U[,2]))^(1.1)*exp(W%*%eta)          # Censoring time
A = runif(n,0,15)                                # administrative censoring time
Z = pmin(T1,T2,A)
d1 = as.numeric(Z==T1)
d2 = as.numeric(Z==T2)
resData = data.frame("Z" = Z,"d1" = d1, "d2" = d2)
theta = c(0.3,1,0.3,1,2)

# Estimate cumulative hazard function
```

```
cumFit <- SolveL(theta, resData,X,W)
cumhaz = cumFit$cumhaz
time = cumFit$times

# plot hazard vs time

plot(time, cumhaz, type = "l",xlab = "Time",
ylab = "Estimated cumulative hazard function")
```

SolveLI *Cumulative hazard function of survival time under independent censoring*

Description

This function estimates the cumulative hazard function of survival time (T) under the assumption of independent censoring. The estimating equation is derived based on martingale ideas.

Usage

```
SolveLI(theta, resData, X)
```

Arguments

theta	Estimated parameter values/initial values for finite dimensional parameters
resData	Data matrix with three columns; Z = the observed survival time, d1 = the censoring indicator of T and d2 = the censoring indicator of C.
X	Data matrix with covariates related to T

Value

This function returns an estimated hazard function, cumulative hazard function and distinct observed survival times;

Examples

```
n = 200
beta = c(0.5)
lambd = 0.35
eta = c(0.9,0.4)
X = cbind(rbinom(n,1,0.5))
W = cbind(rep(1,n),rbinom(n,1,0.5))
frank.cop <- copula::frankCopula(param = 5,dim = 2)
U = copula::rCopula(n,frank.cop)
T1 = (-log(1-U[,1]))/(lambd*exp(X*beta))           # Survival time'
T2 = (-log(1-U[,2]))^(1.1)*exp(W%*%eta)            # Censoring time
```

```
A = runif(n,0,15)                                    # administrative censoring time
Z = pmin(T1,T2,A)
d1 = as.numeric(Z==T1)
d2 = as.numeric(Z==T2)
resData = data.frame("Z" = Z,"d1" = d1, "d2" = d2)
theta = c(0.3,1,0.3,1)

# Estimate cumulative hazard function

cumFit_ind <- SolveLI(theta, resData,X)

cumhaz = cumFit_ind$cumhaz
time = cumFit_ind$times

# plot hazard vs time

plot(time, cumhaz, type = "l",xlab = "Time",
ylab = "Estimated cumulative hazard function")
```

`summary.depFit` *Summary of* `depCensoringFit` *object*

Description

Summary of `depCensoringFit` object

Usage

```
## S3 method for class 'depFit'
summary(object, ...)
```

Arguments

`object`	Output of `fitDepCens` function
`...`	Further arguments

Value

Summary of dependent censoring model fit in the form of table

summary.indepFit *Summary of* indepCensoringFit *object*

Description

Summary of indepCensoringFit object

Usage

```
## S3 method for class 'indepFit'
summary(object, ...)
```

Arguments

object	Output of fitIndepCens function
...	Further arguments

Value

Summary of independent censoring model fit in the form of table

SurvDC *Semiparametric Estimation of the Survival Function under Dependent Censoring*

Description

Provide semiparametric approaches that can be used to model right-censored survival data under dependent censoring (without covariates). The copula-based approach is adopted and there is no need to explicitly specify the association parameter. One of the margins can be modeled nonparametrically. As a byproduct, both marginal distributions of survival and censoring times can be considered as fully parametric. The existence of a cured fraction concerning survival time can also be taken into consideration.

@references Czado and Van Keilegom (2023). Dependent censoring based on parametric copulas. Biometrika, 110(3), 721-738. @references Delhelle and Van Keilegom (2024). Copula based dependent censoring in cure models. TEST (to appear). @references Ding and Van Keilegom (2024). Semiparametric estimation of the survival function under dependent censoring (in preparation).

Usage

```
SurvDC(
  yobs,
  delta,
  tm = NULL,
  copfam = "frank",
  margins = list(survfam = NULL, censfam = "lnorm"),
  cure = FALSE,
  Var = list(do = TRUE, nboot = 200, level = 0.05),
  control = list(maxit = 300, eps = 1e-06, trace = TRUE, ktau.inits = NULL)
)
```

Arguments

yobs	a numeric vector that indicated the observed survival times.
delta	a numeric vector that stores the right-censoring indicators.
tm	a numeric vector that contains interested non-negative time points at which the survival probabilities will be evluated. Note that if we omit the definition of this argument (the default value becomes NULL), our function will automatically output survival probabilities at all oberserved time points, that is, yobs.
copfam	a character string that specifies the copula family. Currently, it supports Archimedean copula families, including "frank" (the default value), "clayton", "gumbel", and "joe". The degenerated independent censoring case can be considered as well by setting "indep". (other options will be added in the near future!)
margins	a list used to define the distribution structures of both the survival and censoring margins. Specifically, it contains the following elements:

survfam a character string that defines the assumed distribution for the survival time random variable, including "lnorm" for log-normal distribution, "weibull" for weibull distribution (other options will be added in the near future).

censfam a character string that defines the assumed distribution for the censoring time random variable, and the details are the same as those shown in survfam.

survtrunc a positive numeric value thats denotes the value of truncation for the assumed distribution, that is, survfam.

censtrunc a positive numeric value thats denotes the value of truncation for the assumed distribution, that is, censfam.

Note if one of the marginal distributions should be modeled nonparametrically, one can let the corresponding argument to be NULL directly. For example if a semiparametric framework that defines the survival margin to be nonparametric and the censoring margin to be parametric, say log-normal, is desired, we can let survfam = NULL and censfam = "lnorm", which is indeed the default value. Furthermore, if no truncation is imposed in survfam (or censfam), one can directly omit the specification of survtrunc (or censtrunc), which is the default specification. We also remark here that when a cured fraction is included

(cure = TRUE), if survfam is not NULL and survtrunc = NULL, we will automatically let survtrunc to be max(yobs). If we wants to model the data with a non-truncated survival distribution when there is a cured fraction, we can set survtrunc = Inf.

cure — a logical value that indicates whether the existence of a cured fraction should be considered.

Var — a list that controls the execution of the bootstrap for variance estimation, and it contains two elements: do is a logical value with default FALSE to tell the function whether the boostrap-based variances should be calculated; nboot is a numeric integer that specifies the number of bootstrap samples.

control — indicates more detailed control of the underlying model fitting procedures. It is a list of the following three arguments:

- maxit a positive integer that denotes the maximum iteration number in optimization. The default value is 300.
- eps a positive small numeric value that denotes the tolerance for convergence. The default value is 1e-6.
- trace a logical value that judges whereh the tracing information on the progress of the model fitting should be produced. The default value if TRUE.
- ktau.inits a numeric vector that contains initial values of the Kendall's tau. The default value is NULL, meaning that a grids of initial values will be automatically generated within our function.

Details

This unified function provide approaches that can be used to model right-censored survival data under dependent censoring (without covariates). Various specifications of marginal distributions can be considered by choosing different combinations of the provided arguments. Generally speaking, the following two scenarios are what we mainly focused on:

nonparametric survival margin and parametric censoring margin (without cure) survfam = NULL, censfam is not NULL and cure = FALSE.

nonparametric survival margin and parametric censoring margin (with cure) survfam = NULL, censfam is not NULL and cure = TRUE.

As byproducts, several other scenarios (the distribution of the underlying survival time is not nonparametric but fully parametric) can also be considered by this R function:

parametric survival and censoring margins (without cure) both survfam and censfam are not NULL and cure = FALSE.

parametric survival and censoring margins (with cure) both survfam and censfam are not NULL and cure = TRUE.

parametric survival margin and nonparametric censoring margin (without cure) survfam is not NULL, censfam = NULL and cure = FALSE.

Furthermore, one might expect that a scenario with "parametric survival margin and nonparametric censoring margin (with cure)" can also be included. Indeed, it can be done based on: survfam is not NULL, censfam = NULL and cure = TRUE. However, from a theoretical perspective of view, whether this type of modeling is reasonable or not still needs further investigations.

We emphasize that the first scenario (in byproducts) has also be considered in another function of this package. Specifically, the scenario of "parametric survival margin and nonparametric censoring margin (without cure)" can be fitted based on `ParamCop()`. However, the default joint modeling of survival and censoring times are based on their joint survival function in line with the semiparametric case (instead of modeling joint distribution function directly as in Czado and Van Keilegom (2023) <doi:10.1093/biomet/asac067>), but the idea of estimation methodology are exactly the same.

Value

A list of fitted results is returned. Within this outputted list, the following elements can be found:

`probs` survival probabilities of the survial margin at `tm`.

`ktau` Kendall's tau.

`parapar` estimation of all parameters (except Kendall's tau) contained in the parametric part.

`GoF` goodness-of-test results.

`curerate` cure rate. If `cure = FALSE`, it is `NULL`.

Examples

```
#-----------------------------------------------------------#
# Basic preparations before running subsequent examples ####
#-----------------------------------------------------------#

# library necessary packages

#-------------------------------------------------------------------------#
# simulated data from Frank copula log-Normal margins (without cure)
#-------------------------------------------------------------------------#

# generate the simulated data

# - the sample size of the generated data
n <- 1000

# information on the used copula
copfam.true <- "frank"
ktau.true <- 0.5
coppar.true <- 5.74

# parameters of the underlying log-normal marginal distributions
survpar.true <- c(2.20,1.00)
censpar.true <- c(2.20,0.25)

# - true underlying survival and censoring times
set.seed(1)
u.TC <- copula::rCopula(
  n        = n,
  copula   = copula::archmCopula(
    family = copfam.true,
```

```
    param  = coppar.true,
    dim    = 2
  )
)
yobs.T <- qlnorm(1-u.TC[,1],survpar.true[1],survpar.true[2])
yobs.C <- qlnorm(1-u.TC[,2],censpar.true[1],censpar.true[2])

# observations
yobs  <- pmin(yobs.T,yobs.C)
delta <- as.numeric(yobs.T<=yobs.C)
cat("censoring rate is", mean(1-delta))

# model the data under different scenarios

# scenario 1: nonparametric survival margin and parametric censoring margin
set.seed(1)
sol.scenario1 <- SurvDC(
  yobs    = yobs,
  delta   = delta,
  tm      = quantile(yobs, c(0.25,0.50,0.75)),
  copfam  = copfam.true,
  margins = list(survfam = NULL, censfam = "lnorm"),
  Var     = list(do = FALSE, nboot = 50)
)
sol.scenario1$probs
sol.scenario1$ktau
sol.scenario1$parapar

# scenario 2: parametric survival and censoring margins
set.seed(1)
sol.scenario2 <- SurvDC(
  yobs    = yobs,
  delta   = delta,
  tm      = quantile(yobs, c(0.25,0.50,0.75)),
  copfam  = copfam.true,
  margins = list(survfam = "lnorm", censfam = "lnorm"),
  Var     = list(do = FALSE, nboot = 50)
)
sol.scenario2$probs
sol.scenario2$ktau
sol.scenario2$parapar

# scenario 3: parametric survival margin and nonparametric censoring margin
set.seed(1)
sol.scenario3 <- SurvDC(
  yobs    = yobs,
  delta   = delta,
  tm      = quantile(yobs, c(0.25,0.50,0.75)),
  copfam  = copfam.true,
  margins = list(survfam = "lnorm", censfam = NULL),
  Var     = list(do = FALSE, nboot = 50)
)
sol.scenario3$probs
```

```
sol.scenario3$ktau
sol.scenario3$parapar

#-----------------------------------------------------------------------
# simulated data from Frank copula log-Normal margins (with cure)
#-----------------------------------------------------------------------

# generate the simulated data

#  true underlying cure rate
curerate.true <- 0.2

# true underlying survival and censoring times
set.seed(1)
u.TC <- copula::rCopula(
  n        = n,
  copula   = copula::archmCopula(
    family = copfam.true,
    param  = coppar.true,
    dim    = 2
  )
)
yobs.T <- sapply(u.TC[,1],function(uT){
  if(uT<=curerate.true){ val <- Inf }else{
   val <- EnvStats::qlnormTrunc((1-uT)/(1-curerate.true),survpar.true[1],survpar.true[2],0,15)
  }
  return(val)
})
yobs.C <- qlnorm(1-u.TC[,2],censpar.true[1],censpar.true[2])
cat("cure rate is",mean(yobs.T==Inf))

#  observations
yobs  <- pmin(yobs.T,yobs.C)
delta <- as.numeric(yobs.T<=yobs.C)
cat("censoring rate is",mean(1-delta))

# model the data under different scenarios (with cure)

# scenario 4: parametric survival and censoring margins
set.seed(1)
sol.scenario4 <- SurvDC(
  yobs    = yobs,
  delta   = delta,
  tm      = quantile(yobs, c(0.25,0.50,0.75)),
  copfam  = copfam.true,
  margins = list(survfam = "lnorm", censfam = "lnorm"),
  Var     = list(do = FALSE, nboot = 50),
  cure    = TRUE
)
sol.scenario4$probs
sol.scenario4$ktau
sol.scenario4$parapar
sol.scenario4$curerate
```

```
# scenario 5: nonparametric survival margin and parametric censoring margin
set.seed(1)
sol.scenario5 <- SurvDC(
  yobs    = yobs,
  delta   = delta,
  tm      = quantile(yobs, c(0.25,0.50,0.75)),
  copfam  = copfam.true,
  margins = list(survfam = NULL, censfam = "lnorm"),
  Var     = list(do = FALSE, nboot = 50),
  cure    = TRUE
)
sol.scenario5$probs
sol.scenario5$ktau
sol.scenario5$parapar
sol.scenario5$curerate
```

`TCsim` *Function to simulate (Y,Delta) from the copula based model for (T,C).*

Description

Generates the follow-up time and censoring indicator according to the specified model.

Usage

```
TCsim(
  tau = 0,
  Copula = "frank",
  Dist.T = "lnorm",
  Dist.C = "lnorm",
  par.T = c(0, 1),
  par.C = c(0, 1),
  n = 10000
)
```

Arguments

`tau`	Value of Kendall's tau for (T,C). The default value is 0.
`Copula`	The copula family. This argument can take values from `c("frank","gumbel","clayton","gaussian",` The default copula model is "frank".
`Dist.T`	Distribution of the survival time T. This argument can take one of the values from `c("lnorm", "weibull", "llogis")`. The default distribution is "lnorm".
`Dist.C`	Distribution of the censoring time C. This argument can take one of the values from `c("lnorm", "weibull", "llogis")`. The default distribution is "lnorm".

par.T	Parameter values for the distribution of T.
par.C	Parameter values for the distribution of C.
n	Sample size.

Value

A list containing the generated follow-up times and censoring indicators.

Examples

```
tau = 0.5
Copula = "gaussian"
Dist.T = "lnorm"
Dist.C = "lnorm"
par.T = c(1,1)
par.C = c(2,2)
n=1000

simdata <- TCsim(tau,Copula,Dist.T,Dist.C,par.T,par.C,n)
Y = simdata[[1]]
Delta = simdata[[2]]
hist(Y)
mean(Delta)
```

Index