Package ‘depCensoring’

December 12, 2024

Type Package

Title Statistical Methods for Survival Data with Dependent Censoring

Version 0.1.5

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 Keile-
gom (2024) <doi:10.1080/01621459.2022.2161387> and Willems et al. (2024+) <doi:10.48550/arXiv.2403.11860> and Ding and Van Keilegom (2024).

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

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Ilias Willems [aut] (<https://orcid.org/0009-0001-9463-9942>),
Gilles Crommen [aut] (<https://orcid.org/0000-0001-8380-1900>),
Negera Wakgari Deresa [aut, cre]
(<https://orcid.org/0000-0002-1302-3725>),
Jie Ding [aut] (<https://orcid.org/0000-0002-6083-7529>),
Claudia Czado [aut] (<https://orcid.org/0000-0002-6329-5438>),
Ingrid Van Keilegom [aut] (<https://orcid.org/0000-0001-8827-7642>)

Repository CRAN

Date/Publication 2024-12-12 14:10:02 UTC

Contents

`boot.fun` *Nonparametric bootstrap approach for the dependent censoring model*

Description

This function estimates the bootstrap standard errors for the finite-dimensional model parameters and for the non-parametric cumulative hazard function. Parallel computing using foreach has been used to speed up the estimation of standard errors.

Usage

```
boot.fun(
  init,
  resData,
  X,
  W,
  lhat,
  cumL,
  dist,
  k,
  lb,
  ub,
  Obs.time,
  cop,
  n.boot,
  n.iter,
  ncore,
  eps
)
```

Arguments

`init`	Initial values for the finite dimensional parameters obtained from the fit of `fitDepCens`
`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
`lhat`	Initial values for the hazard function obtained from the fit of `fitDepCens` based on the original data.
`cumL`	Initial values for the cumulative hazard function obtained from the fit of `fitDepCens` based on the original data.
`dist`	The distribution to be used for the dependent censoring time C. Only two distributions are allowed, i.e, Weibull and lognormal distributions. With the value `"Weibull"` as the default.
`k`	Dimension of X
`lb`	lower boundary for finite dimensional parameters
`ub`	Upper boundary for finite dimensional parameters
`Obs.time`	Observed survival time, which is the minimum of T, C and A, where A is the administrative censoring time.
`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")`.
`n.boot`	Number of bootstraps to use in the estimation of bootstrap standard errors.
`n.iter`	Number of iterations; the default is `n.iter = 20`. The larger the number of iterations, the longer the computational time.
`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-3`

Value

Bootstrap standard errors for parameter estimates and for estimated cumulative hazard function.

boot.funI *Nonparametric bootstrap approach for the independent censoring model*

Description

This function estimates the bootstrap standard errors for the finite-dimensional model parameters and for the non-parametric cumulative hazard function under the assumption of independent censoring. Parallel computing using foreach has been used to speed up the computation.

Usage

```
boot.funI(
  init,
  resData,
  X,
  W,
  lhat,
  cumL,
  dist,
  k,
  lb,
  ub,
  Obs.time,
  n.boot,
  n.iter,
  ncore,
  eps
)
```

Arguments

`init`	Initial values for the finite dimensional parameters obtained from the fit of `fitIndepCens`
`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
`lhat`	Initial values for the hazard function obtained from the fit of `fitIndepCens` based on the original data
`cumL`	Initial values for the cumulative hazard function obtained from the fit of `fitIndepCens` based on the original data
`dist`	The distribution to be used for the dependent censoring time C. Only two distributions are allowed, i.e, Weibull and lognormal distributions. With the value `"Weibull"` as the default

`k`	Dimension of X
`lb`	lower boundary for finite dimensional parameters
`ub`	Upper boundary for finite dimensional parameters
`Obs.time`	Observed survival time, which is the minimum of T, C and A, where A is the administrative censoring time.
`n.boot`	Number of bootstraps to use in the estimation of bootstrap standard errors.
`n.iter`	Number of iterations; the default is `n.iter = 20`. The larger the number of iterations, the longer the computational time
`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-3`

Value

Bootstrap standard errors for parameter estimates and for estimated cumulative hazard function.

`boot.nonparTrans` *Nonparametric bootstrap approach for a Semiparametric transformation model under dependent censpring*

Description

This function estimates the bootstrap standard errors for the finite-dimensional model parameters and for the non-parametric transformation function. Parallel computing using foreach has been used to speed up the estimation of standard errors.

Usage

```
boot.nonparTrans(init, resData, X, W, n.boot, n.iter, eps)
```

Arguments

`init`	Initial values for the finite dimensional parameters obtained from the fit of `NonParTrans`
`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.
`n.boot`	Number of bootstraps to use in the estimation of bootstrap standard errors.
`n.iter`	Number of iterations; the default is `n.iter = 15`. The larger the number of iterations, the longer the computational time.
`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

Bootstrap standard errors for parameter estimates and for estimated cumulative hazard function.

`Bvprob` *Compute bivariate survival probability*

Description

This function calculates a bivariate survival probability based on multivariate normal distribution.

Usage

```
Bvprob(lx, ly, rho)
```

Arguments

`lx`	The first lower bound of integration
`ly`	The second lower bound
`rho`	Association parameter

`chol2par` *Transform Cholesky decomposition to covariance matrix*

Description

This function transforms the parameters of the Cholesky de- composition to the covariance matrix, represented as a the row-wise con- catenation of the upper-triangular elements.

Usage

```
chol2par(par.chol1)
```

Arguments

`par.chol1`	The vector of Cholesky parameters.

Value

Covariance matrix corresponding to the provided Cholesky decomposition.

`chol2par.elem` *Transform Cholesky decomposition to covariance matrix parameter element.*

Description

This function transforms the parameters of the Cholesky de- composition to a covariance matrix element. This function is used in chol2par.R.

Usage

```
chol2par.elem(a, b, par.chol1)
```

Arguments

`a`	The row index of the covariance matrix element to be computed.
`b`	The column index of the covariance matrix element to be computed.
`par.chol1`	The vector of Cholesky parameters.

Value

Specified element of the covariance matrix resulting from the provided Cholesky decomposition.

`CompC` *Compute phi function*

Description

This function estimates phi function at fixed time point t

Usage

```
CompC(theta, t, X, W, ld, cop, dist)
```

Arguments

`theta`	Estimated parameter values/initial values for finite dimensional parameters
`t`	A fixed time point
`X`	Data matrix with covariates related to T
`W`	Data matrix with covariates related to C. First column of W should be ones
`ld`	Output of `SolveL` function at a fixed time t
`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.

`control.arguments` *Prepare initial values within the control arguments*

Description

Prepare initial values within the control arguments

Usage

```
control.arguments(maxit = 300, eps = 1e-06, trace = TRUE, ktau.inits = NULL)
```

Arguments

`maxit`	a positive integer that denotes the maximum iteration number in optimization.
`eps`	a positive small numeric value that denotes the tolerance for convergence.
`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.

`copdist.Archimedean` *The distribution function of the Archimedean copula*

Description

The distribution function of the Archimedean copula

Usage

```
copdist.Archimedean(x, copfam, ktau, coppar = NULL)
```

Arguments

`x`	the value at which the distribution function will be calculated at.
`copfam`	a character string that denotes the copula family.
`ktau`	a numeric value that denotes the Kendall's tau.
`coppar`	a numeric value that denotes the copula parameter.

`cophfunc` *The h-function of the copula*

Description

The h-function of the copula

Usage

```
cophfunc(x, coppar, copfam, condvar = 1)
```

Arguments

`x`	the value at which the h-function will be calculated at.
`coppar`	a numeric value that denotes the copula parameter.
`copfam`	a character string that denotes the copula family.
`condvar`	a numeric value that specifies the type of the h-function.

`coppar.to.ktau` *Convert the copula parameter the Kendall's tau*

Description

Convert the copula parameter the Kendall's tau

Usage

```
coppar.to.ktau(coppar, copfam)
```

Arguments

`coppar`	a numeric value that denotes the copula parameter.
`copfam`	a character string that denotes the copula family.

cr.lik *Competing risk likelihood function.*

Description

This function implements the second step likelihood function of the competing risk model defined in Willems et al. (2024+).

Usage

```
cr.lik(
  n,
  s,
  Y,
  admin,
  cens.inds,
  M,
  Sigma,
  beta.mat,
  sigma.vct,
  rho.mat,
  theta.vct
)
```

Arguments

`n`	The sample size.
`s`	The number of competing risks.
`Y`	The observed times.
`admin`	Boolean value indicating whether or not administrative censoring should be taken into account.
`cens.inds`	matrix of censoring indicators (each row corresponds to a single observation).
`M`	Design matrix, consisting of [intercept, exo.cov, Z, cf]. Note that `cf` represents the multiple ways of 'handling' the endogenous covariate Z, see also the documentation of 'estimate.cmprsk.R'. When there is no confounding, `M` will be [intercept, exo.cov].
`Sigma`	The covariance matrix.
`beta.mat`	Matrix containing all of the covariate effects.
`sigma.vct`	Vector of standard deviations. Should be equal to `sqrt(diag(Sigma))`.
`rho.mat`	The correlation matrix.
`theta.vct`	Vector containing the parameters of the Yeo-Johnsontrans- formations.

Value

Evaluation of the log-likelihood function

References

Willems et al. (2024+). Flexible control function approach under competing risks (in preparation).

---

dat.sim.reg.comp.risks — *Data generation function for competing risks data*

Description

This function generates competing risk data that can be used in simulation studies.

Usage

```
dat.sim.reg.comp.risks(
  n,
  par,
  iseed,
  s,
  conf,
  Zbin,
  Wbin,
  type.cov,
  A.upper = 15
)
```

Arguments

n	The sample size of the generated data set.
par	List of parameter vectors for each component of the transformation model.
iseed	Random seed.
s	The number of competing risks. Note that the given parameter vector could specify the parameters for the model with more than s competing risks, but in this case only the first s sets of parameters will be considered.
conf	Boolean value indicating whether the data set should contain confounding.
Zbin	Indicator whether the confounded variable is binary `Zbin = 1` or not `Zbin = 0`. If `conf = FALSE`, this variable is ignored.
Wbin	Indicator whether the instrument is binary (`Zbin = 1`) or not `Zbin = 0`.
type.cov	Vector of characters "c" and "b", indicating which exogenous covariates should be continuous `"c"` or binary `"b"`.
A.upper	The upper bound on the support of the administrative censoring distribution. This can be used to control for the amount of administrative censoring in the data. Default is `A.upper = 15`. `A.upper = NULL` will return a data set without administrative censoring.

Value

A generated data set

`dchol2par` *Derivative of transform Cholesky decomposition to covariance matrix.*

Description

This function defines the derivative of the transformation function that maps Cholesky parameters to the full covariance matrix.

Usage

```
dchol2par(par.chol1)
```

Arguments

`par.chol1`	The vector of Cholesky parameters.

Value

Derivative of the function that transforms the cholesky parameters to the full covariance matrix.

`dchol2par.elem` *Derivative of transform Cholesky decomposition to covariance matrix element.*

Description

This function defines the derivative of the transformation function that maps Cholesky parameters to a covariance matrix element. This function is used in dchol2par.R.

Usage

```
dchol2par.elem(k, q, a, b, par.chol1)
```

Arguments

`k`	The row index of the parameter with respect to which to take the derivative.
`q`	the column index of the parameter with respect to which to take the derivative.
`a`	The row index of the covariance matrix element to be computed.
`b`	The column index of the covariance matrix element to be computed.
`par.chol1`	The vector of Cholesky parameters.

Value

Derivative of the function that transforms the cholesky parameters to the specified element of the covariance matrix, evaluated at the specified arguments.

`Distance` *Distance between vectors*

Description

This function computes distance between two vectors based on L2-norm

This function computes distance between two vectors based on L2-norm

Usage

```
Distance(b, a)
```

```
Distance(b, a)
```

Arguments

`b`	Second vector
`a`	First vector

`DYJtrans` *Derivative of the Yeo-Johnson transformation function*

Description

Evaluates the derivative of the Yeo-Johnson transformation at the provided argument.

Usage

```
DYJtrans(y, theta)
```

Arguments

`y`	The argument to be supplied to the derivative of the Yeo-Johnson transformation.
`theta`	The parameter of the Yeo-Johnson transformation. This should be a number in the range [0,2].

Value

The transformed value of y.

`estimate.cf` *Estimate the control function*

Description

This function estimates the control function for the endogenous variable based on the provided covariates. This function is called inside estimate.cmprsk.R.

Usage

```
estimate.cf(XandW, Z, Zbin, gammaest = NULL)
```

Arguments

`XandW`	Design matrix of exogenous covariates.
`Z`	Endogenous covariate.
`Zbin`	Boolean value indicating whether endogenous covariate is binary.
`gammaest`	Vector of pre-estimated parameter vector. If `NULL`, this function will first estimate `gammaest`. Default value is `gammaest = NULL`.

Value

List containing the vector of values for the control function and the regression parameters of the first step.

`estimate.cmprsk` *Estimate the competing risks model of Rutten, Willems et al. (20XX).*

Description

This function estimates the parameters in the competing risks model described in Willems et al. (2024+). 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,
  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.
`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. (2024+). Flexible control function approach under competing risks (in preparation).

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
# 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)
```

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 non-parametric 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")
```

`generator.Archimedean` *The generator function of the Archimedean copula*

Description

The generator function of the Archimedean copula

Usage

```
generator.Archimedean(x, coppar, copfam, inverse = FALSE)
```

Arguments

`x`	the value at which the generator function will be calculated at.
`coppar`	a numeric value that denotes the copula parameter.
`copfam`	a character string that denotes the copula family.
`inverse`	a logical value that specifies whether the inverse function will be used.

`IYJtrans` *Inverse Yeo-Johnson transformation function*

Description

Computes the inverse Yeo-Johnson transformation of the provided argument.

Usage

```
IYJtrans(y, theta)
```

Arguments

`y`	The argument to be supplied to the inverse Yeo-Johnson transformation.
`theta`	The parameter of the inverted Yeo-Johnson transformation. This should be a number in the range [0,2].

Value

The transformed value of y.

`Kernel` *Calculate the kernel function*

Description

Calculate the kernel function

Usage

```
Kernel(u, name = "Gaussian")
```

Arguments

`u`	the value in which the kernel function will be calculated at.
`name`	a character used to specify the type of kernel function.

`ktau.to.coppar` *Convert the Kendall's tau into the copula parameter*

Description

Convert the Kendall's tau into the copula parameter

Usage

```
ktau.to.coppar(ktau, copfam)
```

Arguments

`ktau`	a numeric value that denotes the Kendall's tau.
`copfam`	a character string that denotes the copula family.

`LikCopInd` *Loglikehood function under independent censoring*

Description

Loglikehood function under independent censoring

Usage

```
LikCopInd(theta, resData, X, W, lhat, cumL, dist)
```

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
`lhat`	The estimated hazard function obtained from the output of `SolveLI`.
`cumL`	The estimated cumulative hazard function from the output of `SolveLI`.
`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. @importFrom stats nlminb pnorm qnorm sd

Value

Maximized log-likelihood value

`Likelihood.Parametric` *Calculate the likelihood function for the fully parametric joint distribution*

Description

Calculate the likelihood function for the fully parametric joint distribution

Usage

```
Likelihood.Parametric(param, yobs, delta, copfam, margins, cure = FALSE)
```

Arguments

`param`	a vector contains all parametric parameters.
`yobs`	a numeric vector that indicated the observed survival times.
`delta`	a numeric vector that stores the right-censoring indicators.
`copfam`	a character string that specifies the copula family.
`margins`	a list used to define the distribution structures of both the survival and censoring margins.
`cure`	a logical value that indicates whether the existence of a cured fraction should be considered.

`Likelihood.Profile.Kernel` *Calculate the profiled likelihood function with kernel smoothing*

Description

Calculate the profiled likelihood function with kernel smoothing

Usage

```
Likelihood.Profile.Kernel(param, yobs, delta, copfam, margins, cure = FALSE)
```

Arguments

`param`	a vector contains all parametric parameters.
`yobs`	a numeric vector that indicated the observed survival times.
`delta`	a numeric vector that stores the right-censoring indicators.
`copfam`	a character string that specifies the copula family.
`margins`	a list used to define the distribution structures of both the survival and censoring margins.
`cure`	a logical value that indicates whether the existence of a cured fraction should be considered.

Likelihood.Profile.Solve

Solve the profiled likelihood function

Description

Solve the profiled likelihood function

Usage

```
Likelihood.Profile.Solve(
  yobs,
  delta,
  copfam,
  margins,
  ktau.init,
  parapar.init,
  cure,
  curerate.init,
  constraints,
  maxit,
  eps
)
```

Arguments

yobs	a numeric vector that indicated the observed survival times.
delta	a numeric vector that stores the right-censoring indicators.
copfam	a character string that specifies the copula family.
margins	a list used to define the distribution structures of both the survival and censoring margins.
ktau.init	initial value of Kendall's tau.
parapar.init	initial value of parametric parameters.
cure	a logical value that indicates whether the existence of a cured fraction should be considered.
curerate.init	initial value of cure rate.
constraints	constraints of parameters.
maxit	a positive integer that denotes the maximum iteration number in optimization.
eps	a positive small numeric value that denotes the tolerance for convergence.

`Likelihood.Semiparametric` *Calculate the semiparametric version of profiled likelihood function*

Description

Calculate the semiparametric version of profiled likelihood function

Usage

```
Likelihood.Semiparametric(
  param,
  Syobs,
  yobs,
  delta,
  copfam,
  margins,
  cure = FALSE
)
```

Arguments

`param`	a vector contains all parametric parameters.
`Syobs`	values of survival function at observed time points.
`yobs`	a numeric vector that indicated the observed survival times.
`delta`	a numeric vector that stores the right-censoring indicators.
`copfam`	a character string that specifies the copula family.
`margins`	a list used to define the distribution structures of both the survival and censoring margins.
`cure`	a logical value that indicates whether the existence of a cured fraction should be considered.

`LikF.cmprsk` *Second step log-likelihood function.*

Description

This function defines the log-likelihood used to estimate the second step in the competing risks extension of the model described in Willems et al. (2024+).

Usage

```
LikF.cmprsk(par, data, admin, conf, cf)
```

Arguments

`par`	Vector of all second step model parameters, consisting of the regression parameters, variance-covariance matrix elements and transformation parameters.
`data`	Data frame resulting from the 'uniformize.data.R' function.
`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 W.
`cf`	"Control function" to be used. This can either be the (i) estimated control function, (ii) the true control function, (iii) the instrumental variable, or (iv) nothing (`cf = NULL`). Option (ii) is used when comparing the two-step estimator to the oracle estimator, and option (iii) is used to compare the two-step estimator with the naive estimator.

Value

Log-likelihood evaluation of the second step.

References

Willems et al. (2024+). Flexible control function approach under competing risks (in preparation).

`likF.cmprsk.Cholesky` *Wrapper implementing likelihood function using Cholesky factorization.*

Description

This function parametrizes the covariance matrix using its Cholesky decomposition, so that optimization of the likelihood can be done based on this parametrization, and positive-definiteness of the covariance matrix is guaranteed at each step of the optimization algorithm.

Usage

```
likF.cmprsk.Cholesky(par.chol, data, admin, conf, cf, eps = 0.001)
```

Arguments

`par.chol`	Vector of all second step model parameters, consisting of the regression parameters, Cholesky decomposition of the variance-covariance matrix elements and transformation parameters.
`data`	Data frame resulting from the 'uniformize.data.R' function.
`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 W.

`cf`	"Control function" to be used. This can either be the (i) estimated control function, (ii) the true control function, (iii) the instrumental variable, or (iv) nothing (`cf = NULL`). Option (ii) is used when comparing the two-step estimator to the oracle estimator, and option (iii) is used to compare the two-step estimator with the naive estimator.
`eps`	Minimum value for the diagonal elements in the covariance matrix. Default is `eps = 0.001`.

Value

Log-likelihood evaluation of the second step.

`LikGamma1` *First step log-likelihood function for Z continuous*

Description

This function defines the maximum likelihood used to estimate the control function in the case of a continuous endogenous variable.

Usage

```
LikGamma1(gamma, Z, M)
```

Arguments

`gamma`	Vector of coefficients in the linear model used to estimate the control function.
`Z`	Endogenous covariate.
`M`	Design matrix, containing an intercept, the exogenous covariates and the instrumental variable.

Value

Returns the log-likelihood function corresponding to the data, evaluated at the point `gamma`.

`LikGamma2` *First step log-likelihood function for Z binary.*

Description

This function defines the maximum likelihood used to estimate the control function in the case of a binary endogenous variable.

Usage

```
LikGamma2(gamma, Z, M)
```

Arguments

`gamma`	Vector of coefficients in the logistic model used to estimate the control function.
`Z`	Endogenous covariate.
`M`	Design matrix, containing an intercept, the exogenous covariates and the instrumental variable.

Value

Returns the log-likelihood function corresponding to the data, evaluated at the point `gamma`.

`LikI.bis` *Second likelihood function needed to fit the independence model in the second step of the estimation procedure.*

Description

This function defines the log-likelihood used in estimating the second step in the competing risks extension of the model described in Willems et al. (2024+). The results of this function will serve as starting values for subsequent optimizations (LikI.comprsk.R and LikF.cmprsk.R)

Usage

```
LikI.bis(par, data, admin, conf, cf)
```

Arguments

`par`	Vector of all second step model parameters, consisting of the regression parameters, variance-covariance matrix elements and transformation parameters.
`data`	Data frame resulting from the 'uniformize.data.R' function.
`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 W
`cf`	"Control function" to be used. This can either be the (i) estimated control function, (ii) the true control function, (iii) the instrumental variable, or (iv) nothing (`cf = NULL`). Option (ii) is used when comparing the two-step estimator to the oracle estimator, and option (iii) is used to compare the two-step estimator with the naive estimator.

Value

Starting values for subsequent optimization function used in the second step of the estimation procedure.

References

Willems et al. (2024+). Flexible control function approach under competing risks (in preparation).

`LikI.cmprsk` *Second step log-likelihood function under independence assumption.*

Description

This function defines the log-likelihood used to estimate the second step in the competing risks extension assuming independence of some of the competing risks in the model described in Willems et al. (2024+).

Usage

```
LikI.cmprsk(par, data, eoi.indicator.names, admin, conf, cf)
```

Arguments

`par`	Vector of all second step model parameters, consisting of the regression parameters, variance-covariance matrix elements and transformation parameters.
`data`	Data frame resulting from the 'uniformize.data.R' function.
`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.
`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 W
`cf`	"Control function" to be used. This can either be the (i) estimated control function, (ii) the true control function, (iii) the instrumental variable, or (iv) nothing (`cf = NULL`). Option (ii) is used when comparing the two-step estimator to the oracle estimator, and option (iii) is used to compare the two-step estimator with the naive estimator.

Value

Log-likelihood evaluation for the second step in the esimation procedure.

References

Willems et al. (2024+). Flexible control function approach under competing risks (in preparation).

LikI.cmprsk.Cholesky *Wrapper implementing likelihood function assuming independence between competing risks and censoring using Cholesky factorization.*

Description

This function does the same as LikI.cmprsk (in fact, it even calls said function), but it parametrizes the covariance matrix using its Cholesky decomposition in order to guarantee positive definiteness. This function is never used, might not work and could be deleted.

Usage

```
LikI.cmprsk.Cholesky(
  par.chol,
  data,
  eoi.indicator.names,
  admin,
  conf,
  cf,
  eps = 0.001
)
```

Arguments

`par.chol`	Vector of all second step model parameters, consisting of the regression parameters, Cholesky decomposition of the variance-covariance matrix elements and transformation parameters.
`data`	Data frame resulting from the 'uniformize.data.R' function.
`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.
`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 W.

`cf`	"Control function" to be used. This can either be the (i) estimated control function, (ii) the true control function, (iii) the instrumental variable, or (iv) nothing (`cf = NULL`). Option (ii) is used when comparing the two-step estimator to the oracle estimator, and option (iii) is used to compare the two-step estimator with the naive estimator.
`eps`	Minimum value for the diagonal elements in the covariance matrix. Default is `eps = 0.001`.

Value

Log-likelihood evaluation for the second step in the estimation procedure.

`likIFG.cmprsk.Cholesky` *Full likelihood (including estimation of control function).*

Description

This function defines the 'full' likelihood of the model. Specifically, it includes the estimation of the control function in the computation of the likelihood. This function is used in the estimation of the variance of the estimates (variance.cmprsk.R).

Usage

```
likIFG.cmprsk.Cholesky(
  parhatG,
  data,
  eoi.indicator.names,
  admin,
  conf,
  Zbin,
  inst
)
```

Arguments

`parhatG`	The full parameter vector.
`data`	Data frame.
`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 modelled dependently.
`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 W.
Zbin	Boolean value indicating whether the confounding variable is binary.
inst	Type of instrumental function to be used.

Value

Full model log-likelihood evaluation.

`loglike.clayton.unconstrained`

Log-likelihood function for the Clayton copula.

Description

This likelihood function is maximized to estimate the model parameters under the Clayton copula.

Usage

```
loglike.clayton.unconstrained(para, Y, Delta, Dist.T, Dist.C)
```

Arguments

para	Estimated parameter values/initial values.
Y	Follow-up time.
Delta	Censoring indicator.
Dist.T	The distribution to be used for the survival time T. This argument can take one of the values from `c("lnorm", "weibull")` and has to be the same as Dist.C.
Dist.C	The distribution to be used for the censoring time C. This argument can take one of the values from `c("lnorm", "weibull")` and has to be the same as Dist.T.

Value

Maximized log-likelihood value.

`loglike.frank.unconstrained` *Log-likelihood function for the Frank copula.*

Description

This likelihood function is maximized to estimate the model parameters under the Frank copula.

Usage

```
loglike.frank.unconstrained(para, Y, Delta, Dist.T, Dist.C)
```

Arguments

`para`	Estimated parameter values/initial values.
`Y`	Follow-up time.
`Delta`	Censoring indicator.
`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")`.

Value

Maximized log-likelihood value.

`loglike.gaussian.unconstrained` *Log-likelihood function for the Gaussian copula.*

Description

This likelihood function is maximized to estimate the model parameters under the Gaussian copula.

Usage

```
loglike.gaussian.unconstrained(para, Y, Delta, Dist.T, Dist.C)
```

Arguments

`para`	Estimated parameter values/initial values.
`Y`	Follow-up time.
`Delta`	Censoring indicator.
`Dist.T`	The distribution to be used for the survival time T. This argument can only the value `"lnorm"`.
`Dist.C`	The distribution to be used for the censoring time C. This argument can only the value `"lnorm"`.

Value

Maximized log-likelihood value.

`loglike.gumbel.unconstrained` *Log-likelihood function for the Gumbel copula.*

Description

This likelihood function is maximized to estimate the model parameters under the Gumbel copula.

Usage

```
loglike.gumbel.unconstrained(para, Y, Delta, Dist.T, Dist.C)
```

Arguments

`para`	Estimated parameter values/initial values.
`Y`	Follow-up time.
`Delta`	Censoring indicator.
`Dist.T`	The distribution to be used for the survival time T. This argument can take one of the values from `c("lnorm", "weibull")` and has to be the same as Dist.C.
`Dist.C`	The distribution to be used for the censoring time C. This argument can take one of the values from `c("lnorm", "weibull")` and has to be the same as Dist.T.

Value

Maximized log-likelihood value.

`loglike.indep.unconstrained` *Log-likelihood function for the independence copula.*

Description

This likelihood function is maximized to estimate the model parameters under the independence copula.

Usage

```
loglike.indep.unconstrained(para, Y, Delta, Dist.T, Dist.C)
```

Arguments

`para`	Estimated parameter values/initial values.
`Y`	Follow-up time.
`Delta`	Censoring indicator.
`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")`.

Value

Maximized log-likelihood value.

`log_transform` *Logarithmic transformation function.*

Description

Computes the logarithm of a number.

Usage

```
log_transform(y)
```

Arguments

`y`	Numerical value of which the logarithm is computed.

Value

This function returns the logarithm of the provided argument `y` if it is greater than zero. If `y` is smaller than zero, it will return 0.

`Longfun` *Long format*

Description

Change hazard and cumulative hazard to long format

Usage

```
Longfun(Z, T1, lhat, Lhat)
```

Arguments

`Z`	Observed survival time, which is the minimum of T, C and A, where A is the administrative censoring time.
`T1`	Distinct observed survival time
`lhat`	Hazard function estimate
`Lhat`	Cumulative hazard function estimate

`LongNPT` *Change H to long format*

Description

Change a nonparametric transformation function to long format

Usage

```
LongNPT(Z, T1, H)
```

Arguments

`Z`	Observed survival time, which is the minimum of T, C and A, where A is the administrative censoring time.
`T1`	Distinct observed survival time
`H`	Nonparametric transformation function estimate

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
```

`optimlikelihood` *Fit the dependent censoring models.*

Description

Estimates the model parameters by maximizing the log-likelihood.

Usage

```
optimlikelihood(Y, Delta, Copula, Dist.T, Dist.C, start)
```

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 list 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.

`parafam.d` *Obtain the value of the density function*

Description

Obtain the value of the density function

Usage

```
parafam.d(x, parameter, distribution, truncation = NULL)
```

Arguments

`x`	the value in which the density function will be calculated at.
`parameter`	the parameter of the specified distribution
`distribution`	the specified distribution function.
`truncation`	a positive numeric value thats denotes the value of truncation for the assumed distribution.

`parafam.p` *Obtain the value of the distribution function*

Description

Obtain the value of the distribution function

Usage

```
parafam.p(x, parameter, distribution, truncation = NULL)
```

Arguments

`x`	the value in which the distribution function will be calculated at.
`parameter`	the parameter of the specified distribution
`distribution`	the specified distribution function.
`truncation`	a positive numeric value thats denotes the value of truncation for the assumed distribution.

`parafam.trunc` *Obtain the adjustment value of truncation*

Description

Obtain the adjustment value of truncation

Usage

```
parafam.trunc(truncation, parameter, distribution)
```

Arguments

`truncation`	a positive numeric value thats denotes the value of truncation for the assumed distribution.
`parameter`	the parameter of the specified distribution
`distribution`	the specified distribution function.

`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)
```

`Parameters.Constraints` *Generate constraints of parameters*

Description

Generate constraints of parameters

Usage

```
Parameters.Constraints(copfam, margins, cure)
```

Arguments

`copfam`	a character string that specifies the copula family.
`margins`	a list used to define the distribution structures of both the survival and censoring margins.
`cure`	a logical value that indicates whether the existence of a cured fraction should be considered.

`power_transform` *Power transformation function.*

Description

Computes a given power of a number.

Usage

```
power_transform(y, pw)
```

Arguments

`y`	The number which one wants to raise to a certain power `pw`.
`pw`	The power to which to raise `y`.

Value

This function returns the result of raising `y` to the power `pw` when `y > 0`. Otherwise, it will return 1.

`PseudoL` *Likelihood function under dependent censoring*

Description

The `PseudoL` function is maximized in order to estimate the finite dimensional model parameters, including the dependency parameter. This function assumes that the cumulative hazard function is known.

Usage

```
PseudoL(theta, resData, X, W, lhat, cumL, cop, dist)
```

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
`lhat`	The estimated hazard function obtained from the output of `SolveL`.
`cumL`	The estimated cumulative hazard function from the output of `SolveL`.
`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

maximized log-likelihood value

`ScoreEqn` *Score equations of finite parameters*

Description

This function computes the score vectors and the Jacobean matrix for finite model parameters.

Usage

```
ScoreEqn(theta, resData, X, W, H)
```

Arguments

`theta`	Vector of parameters in the semiparametric transformation model.
`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.
`H`	The estimated non-parametric transformation function for a given value of theta

SearchIndicate *Search function*

Description

Function to indicate position of t in observed survival time

Usage

```
SearchIndicate(t, T1)
```

Arguments

`t`	fixed time t
`T1`	distinct observed survival time

SolveH *Estimate a nonparametric transformation function*

Description

This function estimates the nonparametric transformation function H when the survival time and censoring time are dependent given covariates. The estimating equation of H was derived based on the martingale ideas. More details about the derivation of a nonparmaetric estimator of H and its estimation algorithm can be found in Deresa and Van Keilegom (2021).

Usage

```
SolveH(theta, resData, X, W)
```

Arguments

`theta`	Vector of parameters in the semiparametric transformation model.
`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.

Value

Returns the estimated transformation function H for a fixed value of parameters theta.

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.

SolveHt1 *Estimating equation for Ht1*

Description

This function obtains an estimating equation of H at the first observed survival time t1.

Usage

```
SolveHt1(Ht1, Z, nu, t, X, W, theta)
```

Arguments

`Ht1`	The solver solves for an optimal value of Ht1 by equating the estimating equation to zero.
`Z`	The observed survival time, which is the minimum of T, C and A.
`nu`	The censoring indicator for T or C
`t`	A fixed time point
`X`	Data matrix with covariates related to T.
`W`	Data matrix with covariates related to C.
`theta`	Vector of parameters

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")
```

SolveScore *Estimate finite parameters based on score equations*

Description

This function estimates the model parameters

Usage

```
SolveScore(theta, resData, X, W, H, eps = 0.001)
```

Arguments

`theta`	Vector of parameters in the semiparametric transformation model.
`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.
`H`	The estimated non-parametric transformation function for a given value of theta.
`eps`	Convergence error.

---

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.

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.

@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).

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
```

SurvDC.GoF *Calculate the goodness-of-fit test statistic*

Description

Calculate the goodness-of-fit test statistic

Usage

```
SurvDC.GoF(
  yobs,
  delta,
  copfam,
  margins,
  ktau,
  parapar,
  cure = FALSE,
  curerate = NULL
)
```

Arguments

yobs	a numeric vector that indicated the observed survival times.
delta	a numeric vector that stores the right-censoring indicators.

`copfam`	a character string that specifies the copula family.
`margins`	a list used to define the distribution structures of both the survival and censoring margins.
`ktau`	Kendall's tau.
`parapar`	parametric parameters.
`cure`	a logical value that indicates whether the existence of a cured fraction should be considered.
`curerate`	value of cure rate.

`SurvFunc.CG` *Estimated survival function based on copula-graphic estimator (Archimedean copula only)*

Description

Estimated survival function based on copula-graphic estimator (Archimedean copula only)

Usage

```
SurvFunc.CG(tm = NULL, yobs, delta, copfam, ktau, coppar = NULL)
```

Arguments

`tm`	a vector contains all time points that the survival function will be calculated at.
`yobs`	a numeric vector that indicated the observed survival times.
`delta`	a numeric vector that stores the right-censoring indicators.
`copfam`	a character string that denotes the copula family.
`ktau`	a numeric value that denotes the Kendall's tau.
`coppar`	a numeric value that denotes the copula parameter.

`SurvFunc.KM` *Estimated survival function based on Kaplan-Meier estimator*

Description

Estimated survival function based on Kaplan-Meier estimator

Usage

```
SurvFunc.KM(tm = NULL, yobs, delta, type = "right")
```

Arguments

`tm`	a vector contains all time points that the survival function will be calculated at.
`yobs`	a numeric vector that indicated the observed survival times.
`delta`	a numeric vector that stores the right-censoring indicators.
`type`	a character string that specifies the type of the step function. If `type="right"`, it will be a right-continuous function.

`SurvMLE` *Maximum likelihood estimator for a given parametric distribution*

Description

Maximum likelihood estimator for a given parametric distribution

Usage

```
SurvMLE(
  yobs,
  delta,
  distribution,
  truncation = NULL,
  cure = FALSE,
  maxit = 300
)
```

Arguments

`yobs`	a numeric vector that indicated the observed survival times.
`delta`	a numeric vector that stores the right-censoring indicators.
`distribution`	the specified distribution function.
`truncation`	a positive numeric value thats denotes the value of truncation for the assumed distribution.
`cure`	a logical value that indicates whether the existence of a cured fraction should be considered.
`maxit`	a positive integer that denotes the maximum iteration number in optimization.

SurvMLE.Likelihood *Likelihood for a given parametric distribution*

Description

Likelihood for a given parametric distribution

Usage

```
SurvMLE.Likelihood(
  param,
  yobs,
  delta,
  distribution,
  truncation = NULL,
  cure = FALSE
)
```

Arguments

param	a vector contains all parametric parameters.
yobs	a numeric vector that indicated the observed survival times.
delta	a numeric vector that stores the right-censoring indicators.
distribution	the specified distribution function.
truncation	a positive numeric value thats denotes the value of truncation for the assumed distribution.
cure	a logical value that indicates whether the existence of a cured fraction should be considered.

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)
```

`uniformize.data` *Standardize data format*

Description

Checks the required preconditions of the data and possibly restructures the data.

Usage

```
uniformize.data(
  data,
  admin = FALSE,
  conf = FALSE,
  comp.risks = FALSE,
  Zbin = NULL,
  Wbin = NULL
)
```

Arguments

data	A data frame that should contain columns named `Y` and `delta` (unless `comp.risks = TRUE`, see later).
admin	Boolean value indicating whether the provided data frame contains administrative (i.e. independent) censoring on top of the dependent censoring (in the column named `delta`). The default is `admin = FALSE`.
conf	Boolean value indicating whether the provided data frame contains a confounded variable and a corresponding instrument. If `cond = TRUE`, the provided data frame should contain columns named `Z` and `W`, corresponding to the confounded variable and instrument, respectively. Moreover, `Zbin` and `Wbin` should be specified. The default value is `conf = FALSE`.
comp.risks	Boolean value indicating whether the provided data frame contains competing risks. If `comp.risks = TRUE`, the given data frame should contain the columns `delta1`, `delta2`, etc., corresponding to the indicators I(Y = T1), I(Y = T2), etc. respectively. The default is `comp.risks = FALSE`.
Zbin	Boolean or integer value (0, 1) indicating whether the confounded variable is binary. `Zbin = TRUE` or `Zbin = 1` means that Z is binary. `Zbin = FALSE` or `Zbin = 0` means that Z is continuous.
Wbin	Boolean or integer value (0, 1) indicating whether the instrument is binary. `Wbin = TRUE` or `Wbin = 1` means that W is binary. `Wbin = FALSE` or `Wbin = 0` means that W is continuous.

Value

Returns the uniformized data set.

`variance.cmprsk` *Compute the variance of the estimates.*

Description

This function computes the variance of the estimates computed by the 'estimate.cmprsk.R' function.

Usage

```
variance.cmprsk(
  parhatc,
  gammaest,
  data,
  admin,
  conf,
  inst,
  cf,
  eoi.indicator.names,
  Zbin,
  use.chol,
  n.trans,
  totparl
)
```

Arguments

`parhatc`	Vector of estimated parameters, computed in the first part of `estimate.cmprsk.R`.
`gammaest`	Vector of estimated parameters in the regression model for the control function.
`data`	A data frame.
`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`.
`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"`.
`cf`	The control function used to estimate the second step.
`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`.
`use.chol`	Boolean value indicating whether the cholesky decomposition was used in estimating the covariance matrix.
`n.trans`	Number of competing risks in the model (and hence, number of transformation models).
`totparl`	Total number of covariate effects (including intercepts) in all of the transformation models combined.

Value

Variance estimates of the provided vector of estimated parameters.

`YJtrans` *Yeo-Johnson transformation function*

Description

Computes the Yeo-Johnson transformation of the provided argument.

Usage

```
YJtrans(y, theta)
```

Arguments

`y`	The argument to be supplied to the Yeo-Johnson transformation.
`theta`	The parameter of the Yeo-Johnson transformation. This should be a number in the range [0,2].

Value

The transformed value of y.

Index