Package 'depCensoring'

March 12, 2025

Type Package

Title Statistical Methods for Survival Data with Dependent Censoring

Version 0.1.7

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>, Rutten 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, R6, lubridate, splines2

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0), rkriging

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 2025-03-11 23:10:10 UTC

Contents

A_step *A-step in the EAM algorithm described in KMS19*

Description

This function performs the approximation step in the EAM algorithm. More specifically, it fits a Gaussian-process regression model (Kriging) to the evaluated data points $(\theta, c(\theta))$.

Usage

```
A_step(evaluations, verbose = 0)
```

Arguments

evaluations	Matrix containing each point that was already evaluated, alongside the corresponding test statistic and critical value, as its rows.
verbose	Verbosity parameter.

Value

Results of the A-step.

See Also

Package **rkriging**.

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.

Bspline.unit.interval *Evaluate the specified B-spline, defined on the unit interval*

Description

This function evaluates the specified B-spline defined on the unit interval, when considering `n.if.per.cov` B-splines. Currently, the implementation is based on the one in Andrews, Shi 2013 (supplementary materials).

Usage

```
Bspline.unit.interval(x, spline.index, n.if.per.cov, degree = 3)
```

Arguments

`x`	value inside the unit interval at which to evaluate the spline.
`spline.index`	Index of the spline to evaluate.
`n.if.per.cov`	Number of B-splines to consider over the unit interval.
`degree`	Degree of the B-splines. Default is `degree = 3`.

References

Andrews, D.W.K. and Shi, X. (2013). Inference based on confitional moment inequalities. Econometrica. 81(2):609-666.

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

cbMV *Combine bounds based on majority vote.*

Description

This function combines a list of individual identified intervals to a single one based on majority vote. Note that the intersection of all intervals can be viewed as a majority vote as well, so that it is included as a special case.

Usage

```
cbMV(results.list, threshold)
```

Arguments

results.list	List object containing the individual identified intervals.
threshold	Threshold proportion of identified intervals a given value should be contained in in order for it to be included in the combined identified interval. For intersection bounds, set this value to 1.

Value

The combined identified interval.

check.args.pisurv *Check argument consistency.*

Description

This function checks whether the arguments supplied to the main estimation function pi.surv are valid. When arguments are invalid, the an exception is thrown.

Usage

```
check.args.pisurv(
  data,
  idx.param.of.interest,
  idxs.c,
  t,
  par.space,
  search.method,
  add.options
)
```

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. For the latter, we refer to the documentations of `set.hyperparameters`, `set.EAM.hyperparameters` and `set.GS.hyperparameters`. We recommend to use the default parameters, unless you really know what you are doing.

`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.

`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.

`clear.plt.wdw` *Clear plotting window*

Description

This function clears the plotting window

Usage

```
clear.plt.wdw()
```

`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).

`D.hat` *Obtain the diagonal matrix of sample variances of moment functions*

Description

This function computes the diagonal matrix of the sample variance-covariance matrix.

Usage

```
D.hat(input, beta = NULL, t = NULL, hp = NULL, m.avg = NULL, mi.mat = NULL)
```

Arguments

`input`	Can either be the variance-covariance matrix obtained from the function Sigma.hat, or the data frame.
`beta`	The coefficient vector. Only needs to be supplied when the argument for `input` is the data frame.
`t`	The time point of interest. Only needs to be supplied when the argument for `input` is the data frame.
`hp`	List of hyperparameters. Only needs to be supplied when the argument for `input` is the data frame.
`m.avg`	See documentation of `Sigma.hat`. Only needs to be supplied when the argument for `input` is the data frama.
`mi.mat`	See documentation of `Sigma.hat`. Only needs to be supplied when the argument for `input` is the data frama.

`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.

dD.hat *Obtain the matrix of partial derivatives of the sample variances.*

Description

This function computes the matrix of sample derivatives of the sample variances.

Usage

```
dD.hat(
  data,
  beta,
  t,
  hp,
  mi.mat = NULL,
  m.avg = NULL,
  dm.avg = NULL,
  dmi.tens = NULL
)
```

Arguments

data	Data frame.
beta	Vector of coefficients.
t	Time point at which to evaluate the (derivatives of) the moment functions. Also allowed to be a vector of time points (used in estimating the model under assumed time- independent coefficients).
hp	List of hyperparamerers.
mi.mat	A precomputed matrix of moment function evaluations at each observation. If supplied, some computations can be skipped. Default is mi.mat = NULL.
m.avg	A precomputed vector of the sample average of the moment functions. If not supplied, this vector is computed. Default is m.avg = NULL.
dm.avg	Matrix of precomputed sample averages of the derivatives of the moment functions. Default is dm.avg = NULL.
dmi.tens	3D tensor of precomputed evaluations of the derivatives of the moment functions. Default is dmi.tens = NULL.

Value

A matrix containing the partial derivatives of the variances of the moment functions. Each row corresponds to a moment function, each column corresponds to a covariate.

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

dLambda_AFT_ll *Derivative of link function (AFT model)*

Description

This function defines the derivative of the AFT link function.

Usage

```
dLambda_AFT_ll(t)
```

Arguments

t	time parameter.

dLambda_Cox_wb *Derivative of link function (Cox model)*

Description

This function defines the derivative of the Cox PH link function.

Usage

```
dLambda_Cox_wb(t)
```

Arguments

t	time parameter.

dm.bar *Vector of sample average of each moment function* $(\bar{m}_n(\theta))$.

Description

This function computes the matrix containing the sample average of the partial derivatives of the moment functions.

Usage

```
dm.bar(data, beta, t, hp, dmi.tens = NULL)
```

Arguments

data	Data frame.
beta	Vector of coefficients.
t	Time point at which to compute the derivative of the moment functions. Also allowed to be a vector of time points (used in estimating the model under assumed time- independent coefficients).
hp	List of hyperparameters.
dmi.tens	Tensor of derivative moment function evaluations. Can be used to avoid some computation. Default is dmi.tens = NULL.

Value

A matrix containing the sample average of the partial derivatives of the moment functions. Each row corresponds to a moment function, each column corresponds to a coefficient.

do.optimization.Mstep *Optimize the expected improvement*

Description

This function finds the point for which the expected improvement is optimal, based on a given set of starting values. (M_step.R)

Usage

```
do.optimization.Mstep(start.vals, EI.Mstep, hyperparams)
```

Arguments

start.vals	Starting values for optimization.
EI.Mstep	Function to compute expected improvements.
hyperparams	List of hyperparameters.

Value

Maximum of the expected imrpovement function.

draw.sv.init *Draw initial set of starting values for optimizing the expected improvement.*

Description

Used in the M-step (get.starting.values.R). ToDo: Adapt this code so as to also perform sample space contractions as in the MatLab implementation of Bei (2024).

Usage

```
draw.sv.init(theta.hash, dir, hyperparams, EI.fun)
```

Arguments

theta.hash	Tentative optimal value for theta, i.e., the largest or smallest feasible value for theta (if dir = 1 or dir = -1, respectively). A 'feasible value' is one that satisfies all moment restrictions.
dir	Search direction. dir = 1 corresponds to looking for an upper bound. dir = -1 corresponds to looking for a lower bound.
hyperparams	List of hyperparameters.
EI.fun	Function used to compute the expected improvement. See also EI.

Value

Initial set of starting values.

References

Bei, X. (2024). Local linearieation based subvector inference in moment inequality models. Journal of Econometrics. 238:105549-

---

`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.

`EAM` *Main function to run the EAM algorithm*

Description

This function implements the EAM search strategy as described in Kaido, Molinari and Stoye (2019). This is a generic function, requiring the specification of a test function (`test.fun`), as well as the specification of the parameter space (`hyperparams`).

Usage

```
EAM(
  dir,
  test.fun,
  hyperparams,
  evaluations = NULL,
  time.run.duration = FALSE,
  verbose = 0,
  picturose = FALSE
)
```

Arguments

dir	The direction of the test. dir = 1 corresponds to finding the upper bound of the identified set, dir = -1 corresponds to finding the lower bound.
test.fun	The test function to be inverted in order to obtain the identified set. It should take a scalar parameter as argument (i.e. the specified value of a component of the full parameter vector) and return a list with named elements list(theta, t.stat, crit.val), where theta is the scalar value that was tested, t.stat is the value of the test statistic and crit.val is the critical value to be used in determining whether to reject or not reject.
hyperparams	A list of hyperparameters needed in order for this method to run (see set.EAM.hyperparameters.R).
evaluations	Matrix of already evaluated points, of which at least one is feasible. When evaluations = NULL (default), the initial feasible point search will be executed first.
time.run.duration	Boolean value indicating whether to time each step in the EAM algorithm. Requires chronometer.R. Default is time.run.duration = FALSE.
verbose	Boolean value indicating whether or not to print run time updates to the console. Default is verbose = FALSE.
picturose	Boolean value indicating whether or not to visualize the identified set search. Default is picturose = FALSE.

Value

List containing the evaluations of the test statistic and critical values, convergence information, and run times.

References

Kaido et al. (2019). Confidence intervals for projections of partially identified parameters. Econometrica. 87(4):1397-1432.

`EAM.converged` *Check convergence of the EAM algorithm.*

Description

This function checks the convergence of the EAM algorithm. ToDo: Get rid of hard coding stop of algorithm when no more improvement of theta values (maybe related to parameter space contraction, since the problem is that the given points to check in the E-step of the following iteration can always be the same and always be rejected (except of course for the randomly chosen one), while the most promising theta value continues to be the same, infeasible value. In this way, it is possible that theta.hash - mp.theta.next at some point will never decrease).

Usage

```
EAM.converged(
  opt.val.prev,
  evaluations,
  mp.theta.next,
  iter.nbr,
  dir,
  hyperparams,
  verbose
)
```

Arguments

`opt.val.prev`	Previous optimal theta value.
`evaluations`	Matrix of violation curve evaluations.
`mp.theta.next`	Most promising value of theta for which to run the E-step in the following iteration
`iter.nbr`	Number of iterations of the EAM algorithm run so far.
`dir`	Search direction.
`hyperparams`	List of hyperparameters used in the EAM algorithm.
`verbose`	Verbosity parameter.

Value

Boolean value whether or not algorithm has converged.

EI *Expected improvement*

Description

Used in the M-step (M_step.R). Note: predict(fit.krige, ...) has weird beheviour when making predictions for a single value in terms of standard error. We work around this issue in this implementation.

Usage

```
EI(theta, test.fun, fit.krige, theta.hash, dir)
```

Arguments

theta	Vector of coefficients.
test.fun	Test function (cf. EstimationAlgorithmBei.R).
fit.krige	Fitted Kriging model.
theta.hash	Tentative optimal value for theta, i.e., the largest or smallest feasible value for theta (if dir = 1 or dir = -1, respectively). A 'feasible value' is one that satisfies all moment restrictions.
dir	Search direction. dir = 1 corresponds to looking for an upper bound. dir = -1 corresponds to looking for a lower bound.

Value

The expected improvement.

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,
  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. (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
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)
```

`E_step` *E-step in the EAM algorithm as described in KMS19.*

Description

This function performs the estimation step in the EAM algorithm.

Usage

```
E_step(thetas, test.fun, dir, evaluations, verbose)
```

Arguments

`thetas`	Points at which to perform the E-step. Usually the result of the M-step.
`test.fun`	Function returning the test statistic, as well as the critical value.
`dir`	Direction in which to optimize. For finding upper bounds, set `dir = 1`, for finding lower bounds, set `dir = -1`.
`evaluations`	Matrix containing each point that was already evaluated, alongside the corresponding test statistic and critical value, as its rows.
`verbose`	Verbosity parameter.

Value

Results of the E-step.

`feasible_point_search` *Method for finding initial points of the EAM algorithm*

Description

Also called the 'initialization' step in KMS19, this method tries to find at least one initial feasible point, which is required to run the EAM algorithm. ToDo: Investigate whether the feasible point search of Bei (2024) is better. If so, implement it.

Usage

```
feasible_point_search(
  test.fun,
  hyperparams,
  verbose,
  picturose = FALSE,
  parallel = FALSE
)
```

Arguments

test.fun	Function that takes a parameter vector as a first argument and returns the test statistic, as well as the critical value.
hyperparams	List of hyperparameters.
verbose	Verbosity parameter.
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.

Value

Results of the initial feasible point search.

References

Kaido et al. (2019). Confidence intervals for projections of partially identified parameters. Econometrica. 87(4):1397-1432.

---

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")
```

G.box *Family of box functions*

Description

This function defined the class of box functions as defined in Willems et al. (2024+).

Usage

```
G.box(
  x,
  g.idx,
  data,
  n.box.per.cov,
  norm.func,
  cov.ranges = NULL,
  norm.cov.out = NULL,
  ...
)
```

Arguments

x	Vector of covariates to be normalized alongside the data. Default is x = NULL.
g.idx	Index of the instrumental function, in {1, ..., n.inst.func}.
data	Data frame.
n.box.per.cov	Number of box functions to consider per continuous covariate.
norm.func	Function to be used to normalize the covariates.
cov.ranges	Matrix of ranges of the covariates. Used for normalizing the data to the unit interval before applying the instrumental functions. Default is cov.ranges = NULL.
norm.cov.out	Output of a preliminary call to the supplied covariate normalization function.
...	Additional arguments will be ignored. Useful for allowing compatibility with the implementations of other instrument function families. Specifically, it allows to ignore the degree argument used in 'G.spline.R' and 'G.cd.R'.

G.cd *Family of continuous/discrete instrumental function*

Description

The function normalizes the continuous covariates to lie in the unit interval and then evaluates the subvector of continuous covariates on the specified family of instrumental function. For the discrete elements, indicator functions are used for each level.

Usage

```
G.cd(
  x,
  g.idx,
  data,
  n.if.per.cov,
  idxs.c,
```

```
  G.c,
  norm.func,
  discrete.covariate.levels = NULL,
  cov.ranges = NULL,
  norm.cov.out = NULL,
  degree = 3
)
```

Arguments

x	The vector of covariates at which to evaluate the B-splines
g.idx	The index of the instrumental function.
data	Data frame containing the data.
n.if.per.cov	Number of instrumental functions per continuous covariate.
idxs.c	Vector of indices of the continuous elements in the vector of covariates.
G.c	Family of instrumental functions to use for the subvector of continuous covariates.
norm.func	Function to be used to normalize the covariates.
discrete.covariate.levels	Matrix containing as rows all possible 'combined' levels of the discrete covariates. Default is discrete.covariate.levels = NULL.
cov.ranges	Matrix containing as its rows the lower and upper bounds for each continuous covariate. Default is cov.ranges = NULL.
norm.cov.out	Output of a preliminary call to a covariate normalization function (defined above). This is used to speed up computations. Note that this argument should only apply to continuous covariates!! Default is norm.cov.out = NULL.
degree	Degree of the spline functions to be used as instrumental functions for the continuous covariates (if applicable). Default is degree = 3.

G.cd.mc *Family of discrete/continuous instrumental functions, in the case of many covariates.*

Description

This function defines the family of discrete/continuous instrumental functions in the case of many covariates. It does so by considering a instrumental functions for each pair of entries in the given covariate vector.

Usage

```
G.cd.mc(
  x,
  g.idx,
  data,
  n.if.per.cov,
  idxs.c,
  G.c,
  norm.func,
  info.manycov = NULL,
  cov.ranges = NULL,
  norm.cov.out = NULL,
  degree = 3,
  ...
)
```

Arguments

x	The vector of covariates at which to evaluate the B-splines
g.idx	The index of the instrumental function.
data	Data frame containing the data.
n.if.per.cov	Number of instrumental functions per continuous covariate.
idxs.c	Vector of indices of the continuous elements in the vector of covariates.
G.c	Family of instrumental functions to use for the subvector of continuous covariates.
norm.func	Function to be used to normalize the covariates.
info.manycov	Data frame containing some information about the global structure of the instrumental functions of this class. If info.manycov = NULL, it will be computed during execution. Default is info.manycov = NULL.
cov.ranges	Matrix containing as its rows the lower and upper bounds for each continuous covariate. Default is cov.ranges = NULL.
norm.cov.out	Output of function that normalizes the covariates.
degree	Degree of the spline functions to be used as instrumental functions for the continuous covariates (if applicable). Default is degree = 3.
...	Arguments specified here will be ignored. Used for compatibility with other instrumental function classes.

G.hat *Compute the Gn matrix in step 3b of Bei (2024).*

Description

Compute the Gn matrix in step 3b of Bei (2024).

Usage

```
G.hat(
  data,
  beta,
  t,
  hp,
  mi.mat = NULL,
  m.avg = NULL,
  dm.avg = NULL,
  dmi.tens = NULL,
  D = NULL
)
```

Arguments

data	Data frame.
beta	Vector of coefficients.
t	Time point at which to evaluate the (derivatives of) the moment functions.
hp	List of hyperparamerers.
mi.mat	A precomputed matrix of moment function evaluations at each observation. If supplied, some computations can be skipped. Default is mi.mat = NULL.
m.avg	A precomputed vector of the sample average of the moment functions. If not supplied, this vector is computed. Default is m.avg = NULL.
dm.avg	Matrix of precomputed sample averages of the derivatives of the moment functions. Default is dm.avg = NULL.
dmi.tens	3D tensor of precomputed evaluations of the derivatives of the moment functions. Default is dmi.tens = NULL.
D	Diagonal of D-matrix.

Value

A matrix containing the partial derivatives of the variances of the moment functions. Each row corresponds to a moment function, each column corresponds to a covariate.

References

Bei, X. (2024). Local linearieation based subvector inference in moment inequality models. Journal of Econometrics. 238:105549-

G.spline *Family of spline instrumental functions*

Description

This function normalizes the covariates to lie in the unit interval and then evaluates each B-spline at each observation, multiplying together the results per observation.

Usage

```
G.spline(
  x,
  g.idx,
  data,
  n.if.per.cov,
  norm.func,
  cov.ranges = NULL,
  norm.cov.out = NULL,
  degree = 3
)
```

Arguments

x	The vector of covariates at which to evaluate the B-splines
g.idx	The index of the instrumental function. Note that g.idx ranges between 1 and n.if.per.cov^n.cov, as an instrumental function is the product of the appropriate B-spline evaluation for each element in the covariate vector.
data	Data frame containing the data.
n.if.per.cov	Number of instrumental variables to be used per covariate.
norm.func	Function to be used to normalize the covariates.
cov.ranges	Matrix of ranges of the covariates. Used for normalizing the covariates. If cov.ranges = NULL, the data will be normalized in a data-dependent way. Default is cov.ranges = NULL.
norm.cov.out	Output of a preliminary call to the given covariate normalization function. Default is norm.cov.out = NULL.
degree	Degree of B-splines to use. Default value is degree = 3.

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.

get.anchor.points *Get anchor points on which to base the instrumental functions*

Description

The points returned by this function can be used as corner points in the family of box functions, or as knots in the family of B-spline functions.

Usage

```
get.anchor.points(data, n.if.per.cov, normalized = FALSE)
```

Arguments

data	Data set.
n.if.per.cov	Number of instrumental functions to use per continuous covariate.
normalized	Boolean value indicating whether the covariates in the given data frame have been normalized. Default is normalized = FALSE.

get.cond.moment.evals *Compute the conditional moment evaluations*

Description

This function computes the 1(Y <= t) - Lambda(X^T beta(t)) and Lambda(X^T beta(t)) - 1(Y <= t, Delta = 1) parts of the moment functions. (Used in function get.mi.mat.R)

Usage

```
get.cond.moment.evals(data, beta, t, hp)
```

Arguments

data	Data frame.
beta	Vector of coefficients.
t	Time point of interest.
hp	List of hyperparameters.

Value

A vector of 2n elements containing in the first n positions the evaluations of 1(Y <= t) - Lambda(X^T beta(t)) and in the last n positions the evaluations of Lambda(X^T beta(t)) - 1(Y <= t, Delta = 1).

get.cvLLn *Compute the critical value of the test statistic.*

Description

This function computes the critical value following the algorithm of Section 4.3 in Bei (2024).

Usage

```
get.cvLLn(
  BetaI.r,
  data,
  t,
  hp,
  c,
  r,
  par.space,
  inst.func.evals = NULL,
  alpha = 0.95
)
```

Arguments

BetaI.r	Matrix containing in its columns the minimizers of the S-function leading to the test statistic.
data	Data frame.
t	Time point of interest. Also allowed to be a vector of time points (used in estimating the model under assumed time- independent coefficients).
hp	List of hyperparameters.
c	Projection vector.
r	Result of projection of parameter vector onto c.
par.space	Bounds on the parameter space.
inst.func.evals	Matrix of precomputed instrumental function evaluations for each observation in the data set. If NULL, the evaluations will be computed during execution of this function. Default is inst.func.evals = NULL.
alpha	Confidence level.

Value

The critical value for the test statistic.

References

Bei, X. (2024). Local linearieation based subvector inference in moment inequality models. Journal of Econometrics. 238:105549-

---

get.deriv.mom.func *Matrix of derivatives of conditional moment functions*

Description

This function evaluates the derivatives of the conditional moment function at each observation. Used in get.dmi.tens.R

Usage

```
get.deriv.mom.func(data, beta, t, hp)
```

Arguments

data	Data frame.
beta	Parameter vector.
t	Time point of interest.
hp	List of hyperparameters.

`get.dmi.tens` *Faster implementation to obtain the tensor of the evaluations of the derivatives of the moment functions at each observation.*

Description

This function provides a faster implementation of obtaining the evaluations of the derivative of the moment functions at each observation (wrt the previous implementation using 'dm.comp' and 'dm.R'). Used in the function G.hat.R

Usage

```
get.dmi.tens(data, beta, t, hp, inst.func.evals = NULL)
```

Arguments

`data`	Data frame.
`beta`	Vector of coefficients.
`t`	Time point of interest. Also allowed to be a vector of time points (used in estimating the model under assumed time- independent coefficients).
`hp`	List of hyperparameters.
`inst.func.evals`	Precomputed matrix of instrumental function evaluations. Defaults is `inst.func.evals = NULL`, in which case the evaluations will be done inside this function.

`get.extra.Estep.points` *Get extra evaluation points for E-step*

Description

Function used to obtain extra theta values to be supplied to the E-step in the next iteration (M_step.R). Note: this function should be changed when implementing the sample space contractions (see comment made in documentation of `M_step`).

Usage

```
get.extra.Estep.points(dir, theta.hash, maxviol.hash, hyperparams)
```

Arguments

dir	Search direction. dir = 1 corresponds to looking for an upper bound. dir = -1 corresponds to looking for a lower bound.
theta.hash	Tentative optimal value for theta, i.e., the largest or smallest feasible value for theta (if dir = 1 or dir = -1, respectively). A 'feasible value' is one that satisfies all moment restrictions.
maxviol.hash	Violation curve evaluated at theta.hash.
hyperparams	List of hyperparameters.

Value

Points to evaluate in E-step.

---

get.instrumental.function.evals *Evaluate each instrumental function at each of the observations.*

Description

Obtain the evaluations of each observation on each of the instrumental functions. (Used in function get.mi.mat.R)

Usage

```
get.instrumental.function.evals(data, hp)
```

Arguments

data	Data frame.
hp	List of hyperparameters. Notably, it contains the instrumental function to be used in an element named G.

---

get.mi.mat *Faster implementation of vector of moment functions.*

Description

This function obtains the moment function evaluations.

Usage

```
get.mi.mat(data, beta, t, hp, inst.func.evals = NULL)
```

Arguments

data	Data frame.
beta	Vector of coefficients.
t	Time point at which to compute the moment function. Also allowed to be a vector of time points (used in estimating the model under assumed time- independent coefficients).
hp	List of hyperparameters
inst.func.evals	Matrix of instrumental function evaluations. If NULL, it will be computed during execution of this function. Default value is inst.func.evals = NULL.

---

get.next.point *Obtain next point for feasible point search.*

Description

Function to obtain the next point to evaluate in the search for a feasible point. This function evaluates the entire parameter space of the component of theta as evenly as possible. Used in the initialization step (feasible_point_search.R)

Usage

```
get.next.point(evaluations, lb.theta, ub.theta)
```

Arguments

evaluations	Matrix of evaluations performed so far.
lb.theta	Lower bound on the parameter of interest.
ub.theta	Upper bound on the parameter of interest.

Value

Next point in the feasible point search.

get.starting.values *Main function for obtaining the starting values of the expected improvement maximization step.*

Description

Obtain starting values used in the M-step (M_step.R).

Usage

```
get.starting.values(theta.hash, dir, EI.Mstep, hyperparams)
```

Arguments

theta.hash	Tentative optimal value for theta, i.e., the largest or smallest feasible value for theta (if dir = 1 or dir = -1, respectively). A 'feasible value' is one that satisfies all moment restrictions.
dir	Search direction. `dir = 1` corresponds to looking for an upper bound. `dir = -1` corresponds to looking for a lower bound.
EI.Mstep	Function to compute expected improvements.
hyperparams	List of hyperparameters.

Value

Vector of starting values

get.test.statistic *Obtain the test statistic by minimizing the S-function over the feasible region $\beta(r)$.*

Description

Obtain the test statistic by minimizing the S-function over the feasible region $\beta(r)$.

Usage

```
get.test.statistic(
  beta.init,
  data,
  par.space,
  t,
  hp,
  c,
  r,
  inst.func.evals = NULL
)
```

Arguments

beta.init	Starting value of minimization algorithm.
data	Data frame.
par.space	Matrix containing the bounds on the parameter space.
t	Time point at which to evaluate beta. Also allowed to be a vector of time points (used in estimating the model under assumed time- independent coefficients).
hp	List of hyperparameters.
c	Projection vector
r	hypothesised value of the projection.
inst.func.evals	Matrix of precomputed instrumental function evaluations for each observation in the data set. If NULL, the evaluations will be computed during execution of this function. Default is inst.func.evals = NULL.

Value

A list containing the value of the test statistic and the parameter at which this value was attained.

---

gridSearch *Grid search algorithm for finding the identified set*

Description

This function implements the gridsearch and binary search algorithms used to compute the roots of the violation curve and hence in estimating the identified intervals.

Usage

```
gridSearch(
  dir,
  test.fun,
  hyperparams,
  evaluations = NULL,
  time.run.duration = FALSE,
  verbose = 0,
  picturose = FALSE
)
```

Arguments

dir	Search direction.

test.fun	The test function to be inverted in order to obtain the identified set. It should take a scalar parameter as argument (i.e. the specified value of a component of the full parameter vector) and return a list with named elements list(theta, t.stat, crit.val), where theta is the scalar value that was tested, t.stat is the value of the test statistic and crit.val is the critical value to be used in determining whether to reject or not reject.
hyperparams	List of hyperparameters.
evaluations	Matrix of already evaluated points, of which at least one is feasible. When evaluations = NULL (default), the initial feasible point search will be executed first.
time.run.duration	Boolean value indicating whether to time each step in the EAM algorithm. Requires chronometer.R. Default is time.run.duration = FALSE.
verbose	Boolean value indicating whether or not to print run time updates to the console. Default is verbose = FALSE.
picturose	Boolean value indicating whether or not to visualize the identified set search. Default is FALSE.

Value

List containing the evaluations of the test statistic and critical values, convergence information, and run times.

---

gs.algo.bidir *Rudimentary, bidirectional 1D grid search algorithm.*

Description

This function implements a rudimentary, bidirectional search algorithm. It works by expanding a grid with given step.size in both directions, starting from an initial feasible point.

Usage

```
gs.algo.bidir(test.results, max.iter, step.size)
```

Arguments

test.results	Matrix containing the evaluations of the test statistic and critical value.
max.iter	Maximum number of iterations.
step.size	Step size based on which the grid is constructed.

Value

The next point to evaluate in the grid search.

gs.binary *Return the next point to evaluate when doing binary search*

Description

This function implements the binary search algorithm, that starts from a given feasible point and looks in the given direction for the root of the violation curve.

Usage

```
gs.binary(evaluations, dir, iter.nbr, hp)
```

Arguments

evaluations	Matrix of evaluated test statistics and critical values.
dir	Search direction.
iter.nbr	Iteration number.
hp	List of hyperparameters.

Value

The next point to evaluate.

gs.interpolation *Return the next point to evaluate when doing interpolation search*

Description

This function implements the interpolation search algorithm, that starts from a given feasible point and looks in the given direction for the root of the violation curve.

Usage

```
gs.interpolation(evaluations, dir, iter.nbr, hp)
```

Arguments

evaluations	Matrix of evaluated test statistics and critical values.
dir	Search direction.
iter.nbr	Iteration number.
hp	List of hyperparameters.

Value

The next point to evaluate.

gs.regular *Return the next point to evaluate when doing regular grid search*

Description

This function implements a unidirectional grid search, that works by expanding a grid starting from a given feasible point in the given direction.

Usage

```
gs.regular(evaluations, dir, iter.nbr, hp)
```

Arguments

evaluations	Matrix of evaluated test statistics and critical values.
dir	Search direction.
iter.nbr	Iteration number.
hp	List of hyperparameters.

Value

Next point to evaluate in the search algorithm.

insert.row *Insert row into a matrix at a given row index*

Description

Used in initalization step (feasible_point_search.R).

Usage

```
insert.row(evaluations, row, idx.after)
```

Arguments

evaluations	Matrix of violation function evaluations.
row	Row (evaluations) to be added to the evaluation matrix.
idx.after	Index of the row of evaluations after which the given row should be placed.

Value

Evaluation matrix.

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.

Lambda_AFT_ll *Link function (AFT model)*

Description

This function defines the AFT link function.

Usage

```
Lambda_AFT_ll(t)
```

Arguments

t	time parameter.

Lambda_Cox_wb *Link function (Cox model)*

Description

This function defines the Cox PH link function.

Usage

```
Lambda_Cox_wb(t)
```

Arguments

t	time parameter.

Lambda_inverse_AFT_ll *Inverse of link function (AFT model)*

Description

This function defines the inverse of the AFT link function.

Usage

```
Lambda_inverse_AFT_ll(p)
```

Arguments

p	probability.

Lambda_inverse_Cox_wb *Inverse of link function (Cox model)*

Description

This function defines the inverse of the Cox PH link function.

Usage

```
Lambda_inverse_Cox_wb(p)
```

Arguments

p	probability.

lf.delta.beta1 *Loss function to compute Delta(beta).*

Description

This function defines the loss function used in computing the penalized local linear approximation of the test statistic in order to construct the bootstrap distribution of the test statistic.

Usage

```
lf.delta.beta1(
  Delta.sub,
  vnb,
  phi,
  Gn,
  Omegan,
  beta,
  c,
  r,
  data,
  par.space,
  epsilon.n,
  lambda.n
)
```

Arguments

Delta.sub	Subvector of Delta.
vnb	Bootstrapped stochastic process.
phi	Moment selection functions.
Gn	First-order approximation matrix.
Omegan	Correlation matrix of sample moment functions.
beta	Coefficient vector.
c	Projection vector.
r	Value of projected coefficient vector.
data	Data frame.
par.space	Matrix containing the bounds on the parameter space.
epsilon.n	Parameter used in constructing the feasible region as in Example 4.1 in Bei (2024). Not used in this function.
lambda.n	Weight of penalty term.

Value

Loss function evaluation evaluated at the given Delta.

References

Bei, X. (2024). Local linearieation based subvector inference in moment inequality models. Journal of Econometrics. 238:105549-

lf.ts *'Loss function' of the test statistic.*

Description

This function implements the loss function used in computing the test statistic.

Usage

```
lf.ts(beta.sub, data, t, hp, c, r, inst.func.evals = NULL)
```

Arguments

beta.sub	Subvector of coefficient vector.
data	Data frame.
t	Time point of interest. Also allowed to be a vector of time points (used in estimating the model under assumed time- independent coefficients).
hp	List of hyperparameters.
c	Unit vector containing unity at the location of the parameter of interest.
r	Value of the parameter of interest that is tested.
inst.func.evals	Pre-computed matrix of insturmental function evaluations. If not supplied, it will be computed during execution of this function.

Value

S-functions evaluation for the specified parameter vector.

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

`m.bar` *Vector of sample average of each moment function* $(\bar{m}\_n(\theta))$.

Description

This function obtains the vector of sample averages of each moment function.

Usage

```
m.bar(data, beta, t, hp, mi.mat = NULL)
```

Arguments

`data`	Data frame.
`beta`	Vector of coefficients.
`t`	Time point at which to compute the moment functions. Also allowed to be a vector of time points (used in estimating the model under assumed time- independent coefficients).
`hp`	List of hyperparameters.
`mi.mat`	Matrix of moment function evaluations. Can be used to avoid some computation. Default is `mi.mat = NULL`.

`MSpoint` *Analogue to KMS_AUX4_MSpoints(...) in MATLAB code of Bei (2024).*

Description

Create starting values for EI maximization. Used in the M-step (get.starting.values.R).

Usage

```
MSpoint(draws.init)
```

Arguments

`draws.init`	Initial draws.

References

Bei, X. (2024). Local linearieation based subvector inference in moment inequality models. Journal of Econometrics. 238:105549-

M_step *M-step in the EAM algorithm described in KMS19.*

Description

This function performs the maximization step in the EAM algorithm. More specifically, it maximizes the expected improvement. ToDo: implement sample space contractions (see comment made in documentation of `draw.sv.init`).

Usage

```
M_step(
  dir,
  evaluations,
  theta.hash,
  fit.krige,
  test.fun,
  c,
  par.space,
  hyperparams,
  verbose
)
```

Arguments

`dir`	Direction to search in. `dir = 1` corresponds to finding the upper bound of the confidence interval. `dir = -1` corresponds to finding the lower bound.
`evaluations`	Matrix containing each point that was already evaluated, alongside the corresponding test statistic and critical value, as its rows.
`theta.hash`	Tentative best value of theta. Obtained from the E-step.
`fit.krige`	Kriging model obtained from the A-step.
`test.fun`	The test function to be inverted in order to obtain the identified set.
`c`	Projection vector.
`par.space`	Bounds of the parameter space.
`hyperparams`	Parameters used in obtaining initial values for the maximization algorithm. If `NULL`, default values are used. Default is `hyperparams = NULL`.
`verbose`	Verbosity parameter.

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
```

`normalize.covariates` *Normalize the covariates of a data set to lie in the unit interval by scaling based on the ranges of the covariates.*

Description

This function normalized the covariates in the data to lie in the unit interval based on either the empirical or known ranges of the covariates. It is useful to perform this step when defining the instrumental functions later on. This function is used in `G.box.R`, `G.spline.R` and by extension in `G.cd.R`.

Usage

```
normalize.covariates(
  data = NULL,
  x = NULL,
  cov.ranges = NULL,
  idxs.c = "all",
  norm.cov.out = NULL,
```

```
  ...
)
```

Arguments

`data`	(optional) Data set to be used to construct the normalizing transformation. Default is `data = NULL`.
`x`	(optional) Vector of covariates to be normalized alongside the data. Default is `x = NULL`.
`cov.ranges`	(optional) Matrix that specifies the range of each of the covariates in the data set. Each column corresponds to a covariate. The first row contains the lower bound, the second row contains the upper bound. If not supplied, the data will be normalized based on the minimum and maximum detected values. If supplied, the non data-dependent transformation function listed in the appendix of Andrews, Shi 2013 will be used. Default is `cov.ranges = NULL`.
`idxs.c`	(optional) Vector of indices of covariates that are continuous. Note that that indices are relative to the covariate vector, not the full data set. Default value is `idxs.c = "all"`, which indicates that all elements should be regarded as continuous. If `idxs.c = NULL`, all elements are regarded as discrete.
`norm.cov.out`	(optional) The output of a previous call to this function. Can be used to speed up computation. If both `data` and `norm.cov.out` are supplied to the function, this method will throw an error. Default is `norm.cov.out = NULL`.
`...`	Allows easier interchangeability between covariate normalization functions. All arguments specified under `...` will be ignored.

References

Andrews, D.W.K. and Shi, X. (2013). Inference based on confitional moment inequalities. Econometrica. 81(2):609-666.

---

normalize.covariates2 *Normalize the covariates of a data set to lie in the unit interval by transforming based on PCA.*

Description

This function normalized the covariates in the data to lie in the unit interval based on a principal component analysis. It is useful to perform this step when defining the instrumental functions later on. This function is used in `G.box`, `G.spline` and by extension `G.cd`.

Usage

```
normalize.covariates2(
  data = NULL,
  x = NULL,
  idxs.c = "all",
```

```
  norm.cov.out = NULL,
  ...
)
```

Arguments

data	(optional) Data set to be used to construct the normalizing transformation. Default is data = NULL.
x	(optional) Vector of covariates to be normalized alongside the data. Default is x = NULL.
idxs.c	(optional) Vector of indices of covariates that are continuous. Note that that indices are relative to the covariate vector, not the full data set. Default value is idxs.c = "all", which indicates that all elements should be regarded as continuous. If idxs.c = NULL, all elements are regarded as discrete.
norm.cov.out	(optional) The output of a previous call to this function. Can be used to speed up computation. If both data and norm.cov.out are supplied to the function, the function will throw an error. Default is norm.cov.out = NULL
...	Allows easier interchangeability between covariate normalization functions. All arguments specified under ... will be ignored.

Omega.hat *Obtain the correlation matrix of the moment functions*

Description

This function computes the correlation matrix corresponding to the variance-covariance matrix as returned by Sigma.hat.R

Usage

```
Omega.hat(Sigma)
```

Arguments

Sigma	The output of the function Sigma.hat

`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.

---

pi.surv *Estimate the model of Willems et al. (2024+).*

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. For the latter, we refer to the documentations of `set.hyperparameters`, `set.EAM.hyperparameters` and `set.GS.hyperparameters`. We recommend to use the default parameters, unless you really know what you are doing.
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. (2024+). Partial identification for a class of survival models under dependent censoring. (In preparation).

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)
```

plot_addpte *Draw points to be evaluated*

Description

This function draws the points to be evaluated.

Usage

```
plot_addpte(pte, col = "orange")
```

Arguments

pte	Vector of points to be evaluated.
col	Color of the points.

plot_addpte.eval *Draw evaluated points.*

Description

This function draws evaluated points. Feasible points are indicated in green, red points correspond to infeasible points.

Usage

```
plot_addpte.eval(evaluations)
```

Arguments

evaluations	Matrix of evaluations to be drawn.

plot_base *Draw base plot*

Description

This functon draws the base plot, used when `picturose = TRUE`.

Usage

```
plot_base(c, hp)
```

Arguments

c	Projection vector
hp	List of hyperparameters

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

S.func *S-function*

Description

This function computes the loss function at a given point.

Usage

```
S.func(m, Sigma)
```

Arguments

m	Vector of averages of moment functions.
Sigma	Sample variance-covariance matrix of moment functions.

Value

S(m, Sigma).

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

---

set.EAM.hyperparameters *Set default hyperparameters for EAM algorithm*

Description

This function returns a list with the (default) hyperparameters used in the EAM algorithm

Usage

```
set.EAM.hyperparameters(options)
```

Arguments

`options` A list of user-specified values for (some of) the hyperparameters. These hyperparameters can include:

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.

Value

List of hyperparameters for the EAM algotithm.

`set.GS.hyperparameters` *Set default hyperparameters for grid search algorithm*

Description

This function returns a list with the (default) hyperparameters used in the grid search algorithm

Usage

```
set.GS.hyperparameters(options)
```

Arguments

options	A list of user-specified values for (some of) the hyperparameters. These hyperparameters could include: **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.

Value

List of hyperparameters for the gridsearch and binary search algorithms.

---

set.hyperparameters *Define the hyperparameters used for finding the identified interval*

Description

This function defines all the necessary hyperparameters used to run the methodology.

Usage

```
set.hyperparameters(data, par.space, c, search.method, options)
```

Arguments

data	Data frame.
par.space	Bounds on the parameter space.
c	Projection vector.
search.method	Search method to use ("EAM" or "GS")
options	List of user specified hyperparameters that will substitute the corresponding default values. This list can contain the entries: **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 as discussed in Willems et al. (2024+). **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 (cf. Willems et al., 2024). 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.

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!).

Value

The list of hyperparameters.

`Sigma.hat` *Compute the variance-covariance matrix of the moment functions.*

Description

This function comptutes the empricical variance-covariance matrix of the moment functions.

Usage

```
Sigma.hat(data, beta, t, hp, m.avg = NULL, mi.mat = NULL)
```

Arguments

`data`	Data frame.
`beta`	Coefficient vector.
`t`	Time point of interest.
`hp`	List of hyperparameters.

m.avg	A precomputed vector of the sample average of the moment functions. If not supplied, this vector is computed. Default is m.avg = NULL.
mi.mat	A precomputed matrix of moment function evaluations at each observation. If supplied, some computations can be skipped. Default is mi.mat = NULL.

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)
```

---

test.point_Bei *Perform the test of Bei (2024) for a given point*

Description

This function performs the unconditional moment restriction test as described in Bei (2024).

Usage

```
test.point_Bei(
  r,
  c,
  t,
```

```
  par.space,
  data,
  hp,
  verbose = FALSE,
  inst.func.evals = NULL,
  alpha = 0.95,
  parallel = FALSE
)
```

Arguments

r	Result of the projection for which the test should be carried out.
c	The projection matrix. For now, c is restricted to being an elementary vector, i.e. c = (0, ...,0, 1, 0, ..., 0).
t	The time point at which to evaluate theta.
par.space	Matrix containing 2 columns and d_θ rows, where d_θ is the dimension of the parameter space. The first column represents the lower left corner of the parameter space, the second column represents the upper right corner. At least for the time being, only rectangular parameter spaces are allowed.
data	Data frame on which to base the test.
hp	List of hyperparameters needed.
verbose	Boolean variable indicating whether to print updates of the estimation process to the console.
inst.func.evals	Matrix of precomputed instrumental function evaluations for each observation in the data set. Used to speed up the simulations. If NULL, the evaluations will be computed during execution of this function. Default is inst.func.evals = NULL.
alpha	The significance level at which to perform the test. Default is alpha = 0.95.
parallel	Flag for whether or not parallel computing should be used. Default is parallel = FALSE.

References

Bei, X. (2024). Local linearization based subvector inference in moment inequality models. Journal of Econometrics, 238(1), 105549-. https://doi.org/10.1016/j.jeconom.2023.10554

---

test.point_Bei_MT *Perform the test of Bei (2024) simultaneously for multiple time points.*

Description

This function performs the unconditional moment restriction test as described in Bei (2024). This function directly extends test.point_Bei by allowing for pairs of moment restrictions over a grid of time points.

Usage

```
test.point_Bei_MT(
  r,
  c,
  t,
  par.space,
  data,
  hp,
  verbose = FALSE,
  inst.func.evals = NULL,
  alpha = 0.95,
  parallel = FALSE
)
```

Arguments

r	Result of the projection for which the test should be carried out.
c	The projection matrix. For now, c is restricted to being an elementary vector, i.e. c = (0, ...,0, 1, 0, ..., 0).
t	The time point at which to evaluate theta. Also allowed to be a vector of time points (used in estimating the model under assumed time- independent coefficients).
par.space	Matrix containing 2 columns and d_θ rows, where d_θ is the dimension of the parameter space. The first column represents the lower left corner of the parameter space, the second column represents the upper right corner. At least for the time being, only rectangular parameter spaces are allowed.
data	Data frame on which to base the test.
hp	List of hyperparameters needed.
verbose	Boolean variable indicating whether to print updates of the estimation process to the console.
inst.func.evals	Matrix of precomputed instrumental function evaluations for each observation in the data set. Used to speed up the simulations. If NULL, the evaluations will be computed during execution of this function. Default is inst.func.evals = NULL.
alpha	The significance level at which to perform the test. Default is alpha = 0.95.
parallel	Flag for whether or not parallel computing should be used. Default is parallel = FALSE.

References

Bei, X. (2024). Local linearization based subvector inference in moment inequality models. Journal of Econometrics, 238(1), 105549-. https://doi.org/10.1016/j.jeconom.2023.10554

`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