Package 'lpl'

May 8, 2026

Type Package

Title Local Partial Likelihood Estimation and Simultaneous Confidence Band

Version 0.13

Date 2025-08-19

Author Bingshu E. Chen [aut, cre],
Yicong Liu [aut],
Siwei Zhang [aut],
Teng Wen [aut],
Wenyu Jiang [aut]

Maintainer Bingshu E. Chen <bingshu.chen@queensu.ca>

Depends R (>= 3.5.0), MASS, methods, parallel, survival

Description Local partial likelihood estimation by Fan, Lin and Zhou(2006)<doi:10.1214/009053605000000796> and simultaneous confidence band is a set of tools to test the covariates-biomarker interaction for survival data. Test for the covariates-biomarker interaction using the bootstrap method and the asymptotic method with simultaneous confidence band (Liu, Jiang and Chen (2015)<doi:10.1002/sim.6563>).

License GPL-2

LazyLoad yes

NeedsCompilation no

Repository CRAN

Date/Publication 2025-08-20 23:00:01 UTC

Contents

`lpl-package` *Local Partial Likelihood Boostrap test*

Description

This package fits a multivariable local partial likelihood model for covariate-biomarker interaction with survival data.

Details

"lpl" is a R package for multivariate covariate-biomarker interaction uisng local partial likelihood method.

Please use the following steps to install 'lpl' package:

1. First, you need to install the 'devtools' package. You can skip this step if you have 'devtools' installed in your R. Invoke R and then type

install.packages("devtools")

2. Load the devtools package.

library(devtools)

3. Install "lpl" package with R commond

install_github("statapps/lpl")

"lpl" uses local partial likelihood to etimate covariate-biomarker interactions and bootstrap method to test the significance of the interactions.

Author(s)

Siwei Zhang and Bingshu E. Chen

Maintainer: Bingshu E. Chen <bingshu.chen@queensu.ca>

References

1. Fan, J., Lin, H,, Zhou, Y. (2006). Local partial-likelihood estimation for lifetime data. The Annals of Statistics. 34, 290-325.

2. Liu, Y., Jiang, W. and Chen, B. E. (2015). Testing for treatment-biomarker interaction based on local partial-likelihood. Statistics in Medicine. 34, 3516-3530.

3. Zhang, S., Jiang, W. and Chen, B. E. (2016). Estimate and test of multivariate covariates and biomarker interactions for survival data based on local partial likelihood. Manuscript in preparation.

See Also

`coxph, survival`

Examples

```
# fit = lpl(y~trt+age+biomarker)
```

`control` *Auxiliary function for lpl fitting*

Description

Auxiliary function for `lple` fitting. Typically only used internally by 'lpl', but may be used to construct a control argument to either function.

Usage

```
#   lpl.control(h, kernel = 'gaussian', B, w0, p1, pctl)
```

Arguments

`h`	bandwidth of kernel function. The default value is h = 0.2
`kernel`	kernel funtion types, including "gaussian", "epanechnikov", "rectangular", "triangular", "biweight", "cosine", "optcosine". The default value is 'gaussian'
`B`	number of bootstrap times. The default value is 200
`w0`	the estimated points in the interval of (0,1), select arbitrarily. The default value is seq(0.05, 0.95, 0.025)
`p1`	the number of dependend variables that make interactions with the biomarker w. The default value is 1
`pctl`	the estimated points that want to be shown in the output. The default value is seq(0.2, 0.8, 0.1)

Details

Control is used in model fitting of lpl.

Value

This function checks the internal consisitency and returns a list of value as inputed to control model fit of lpl.

Author(s)

Siwei Zhang and Bingshu E. Chen

See Also

lplb, lple

Examples

```
## The default control values are: h = 0.2, kernel = 'gaussian', B = 200,
## w0 = seq(0.05, 0.95, 0.025), p1 = 1, pctl = seq(0.2, 0.8, 0.1)
##
## To fit the lpl model with some control variables changed,

w0=seq(0.05, 0.95, by=0.05)
ctl = lpl.control(w0=w0, h=0.3, p1=2, B=100)

## then fit the lple model
```

---

coxScoreHess *Calculate the Score vector / Hessian matrix for the Cox model*

Description

Calculate the Score vector or the Hessian matrix for the Cox proportional hazards model with inputs of covariates, survival outcomes and the relative risks

Usage

```
coxScoreHess(X, y, exb, hess = FALSE)
coxpl(X, y, beta, sorted = FALSE)
```

Arguments

X	the covariate matrix from model.matrix, without the interecpt term.
y	y is a survival object, y = Surv(time, event).
exb	exb is the relative risks with exb = exp(X*beta).
hess	output the Hessian matrix, with hess = FALSE as the default, which outputs the score vector only.
beta	the p x 1 regression coefficient to be used in calculation of the partial likelihood.
sorted	data were sorted by time from the largest to the smallest, to speed up the algorithm, default is sorted = FALSE, sort by time is recommand when the function will be called multiple times for the same y.

Details

The survival time shall be sorted from the largest to the smallest, an error will occur if y is not sorted.

partial likelihood = sum(event(exp(X*beta)/S0))

score = sum(event*(X - S1/S0))

Sigma = sum(S1*t(S1))

H = sum(event*(S2/S0 - S1*t(S1)/S0))

the robust varaince can be calculated by inv(H)*Sigma*inv(H).

Value

An p by 1 vector of the score of the function calculated at the point relative exp(X*beta). If hess = TRUE, then a list with the following three components is returned:

`score`	a 1 x p score vector.
`Sigma`	a p x p matrix for the empirical varaince of the score.
`H`	a p x p hessian matrix.

See Also

`numHessian` `numScore` `multiRoot`

`ibs` *The Brier Score and Integrated Brier Score (IBS)*

Description

Calculate the Brier score and the integration of the Brier score (IBS) using the Inverse Probability of Censoring Weighting (IPCW) method.

Usage

```
brierScore(object, St, tau)
## Default S3 method:
ibs(object, ...)
## S3 method for class 'coxph'
ibs(object, newdata = NULL, newy = NULL, ...)
## S3 method for class 'lple'
ibs(object, newdata = NULL, newy = NULL, ...)
## S3 method for class 'Surv'
ibs(object, survProb, ...)
```

Arguments

object	for ibs.Surv and ibs.default, it is a survival object created by Surv(time, event). For others, it is a model object returned by coxph, lple.
newdata	optional new data at which the IBS is calculated. If absent, IBS is for the dataframe used in the original model fit.
newy	optional new survival object data. Default is NULL.
St	the predicted survival function at time tau to calcuate the Brier score.
survProb	the predicted survival function matrix. Row denotes each subject and column denotes each time points. survProb[i,j] denotes the predicted survival probability of the ith subject at the time t[j].
tau	the time point at which the Brier score is calculated.
...	additional arguments to be passed to the functions such as ibs.coxph, ibs.lple, ibs.Surv etc.

Details

The Brier score is the mean square difference between the true survival status and the predicted survival function. The Brier score is defined as,

bs(tau) = 1/n*I(T_i>tau, delta_i = 1) S(t)^2/G(T_i) + (1-S(tau))^2/G(tau),

where G = IPCW(Surv(time, event)), and IPCW is called to fit a KM model for the censoring time.

The IBS is an integrated Brier Score over time. That is an integrated weighted squared distance between the estimated survival function and the empirical survival function int_0 ^ 2 (I(T > t) - S(t))^2dt. The inverse probability censoring weighting(IPCW) is used to adjust for censoring.

Value

A value of the Brier score or integration of the Brier score is returned.

Author(s)

Bingshu E. Chen

References

1. Brier, G. W. (1950). Verification of forecasts expressed in terms of probability. Monthly Weather Review, 78.

2. Graf, Erika, Schmoor, Claudia, Sauerbrei, & Willi, et al. (1999). Assessment and comparison of prognostic classification schemes for survival data. Statistics in Medicine, 18, 2529-2545.

See Also

The IPCW method is used calculate the Brier score and the integrated Brier score. A Cox proportional hazards (PH) model (coxph) shall be fitted to calculate Brier and IBS for the Cox PH model.

The Brier score for the Cox model can also be calculated by brier.

Examples

```
set.seed(29)
n     = 25
time  = rexp(n, 1)
event = rbinom(n, 1, 0.75)

### calculate the Brier score at time tau
tau   = 0.5
St    = pexp(rep(tau, n), 1, lower.tail = FALSE)
bs    = brierScore(Surv(time, event), St, tau)

### calculate the integrated Brier score
#fit   = coxph(Surv(time, event)~1)
#IBS  = ibs(fit)
```

IPCW *Inverse probability of censoring weighting (IPCW)*

Description

Create the Inverse Probability of Censoring Weighting (IPCW) using the Kaplan-Meier (KM) method. print are used to provide a short summary of lple outputs.

Usage

```
IPCW(object)
ipcw(time, event)
```

Arguments

object	a survival object created by Surv(time, event).
time	the survival time.
event	the status indicator, normally 0=alive, 1=dead.

Details

survfit is called to fit a KM model for the censoring time.

Value

A vector for the survival function of the censoring time is returned.

Author(s)

Bingshu E. Chen

See Also

The IPCW function is used in brierScore to calculate the brier score and ibs to calculate the integrated brier score.

Examples

```
# See example in brier ibs
```

---

lplb *Local partial likelihood bootstrap (LPLB) method to fit biomarker Models*

Description

{lplb} is a R package for local partial likelihood estimation (LPLE) (Fan et al., 2006) of the coefficients of covariates with interactions of the biomarker W, and hypothesis test of whether the relationships between covariates and W are significant, by using bootstrap method.

Usage

```
## Default S3 method:
lplb(x, y, control, ...)
## S3 method for class 'formula'
lplb(formula, data=list(...), control = list(...), ...)

# use
#          lplb(y ~ X1+X2+...+Xp+w, data=data, control)
#
# to fit a model with interactions between biomarker (w) with the first p1
# terms of dependent variables.
# p1 is included in 'control'. p1<p. See 'lplb.control' for details
#
# use
#         lplb(x, y, control)
#
# to fit a model without the formula
#
# Biomarker w should be the 'LAST' dependend variable
```

Arguments

formula	an object of class "formula"(or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
data	an optional data frame, list or environment (or object coercible by 'as.data.frame' to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula).

x, y	For 'lplb.default', x is a design matrix of dimension n * (p+1) and y is a vector of observations of length n for a "Surv" object for "coxph".
control	a list of parameters for controlling the fitting process. See 'lplb.control' for details
...	additional arguments to be passed to the low level regression fitting functions (see below).

Details

Here 'w' is a Biomarker variable. This variable is required and shall be the last dependent variable in the formula.

'x.cdf' is a function that maps biomarker values to interval (0, 1) using its empirical cumulative distribution function.

Value

lplb returns an object of class inheriting from "lplb" which inherits from the class 'coxph'. See later in this section.

The function "print" (i.e., "print.lplb") can be used to obtain or print a summary of the results.

An object of class "lplb" is a list containing at least the following components:

beta_w	a matrix of m * p1, the estimated coefficients at each of the m estimated points, for the first p1 dependent variables with interactions of the biomarker w
Q1	the test statistic of the data
mTstar	a vector of the test statistics from B times' bootstrap
pvalue	the p-value of the hypothesis that beta_w is a constant

Note

This package was build on code developed by Yicong Liu for simple treatment-biomaker interaction model.

Author(s)

Siwei Zhang and Bingshu E. Chen (bingshu.chen@queensu.ca)

References

Zhang, S., Jiang, W. and Chen, B. E. (2016). Estimate and test of multivariate covariates and biomarker interactions for survival data based on local partial likelihood. Manuscript in preparation.

See Also

coxph, lpl.control print.lple plot.lple

Examples

```
dat = lplDemoData(50)
fit = lplb(Surv(time, status)~z1 + z2 + w, data = dat, B = 3, p1 = 2)
print(fit)
```

`lple` *Local partial likelihood estimate (LPLE) method to fit biomarker Models*

Description

{lple} is a R package for local partial likelihood estimation (LPLE) (Fan et al., 2006) of the coefficients of covariates with interactions of the biomarker W, and hypothesis test of whether the relationships between covariates and W are significant, by using bootstrap method.

Usage

```
## Default S3 method:
lple(x, y, control, ...)
## S3 method for class 'formula'
lple(formula, data=list(...), control = list(...), ...)

# use
#          lple(y ~ X1+X2+...+Xp+w, data=data, control)
#
# to fit a model with interactions between biomarker (w) with the first p1
# terms of dependent variables.
# p1 is included in 'control'. p1<p. See 'lplb.control' for details
#
# use
#         lple(x, y, control)
#
# to fit a model without the formula
#
# Biomarker w should be the 'LAST' dependend variable
```

Arguments

`formula`	an object of class "formula"(or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`data`	an optional data frame, list or environment (or object coercible by 'as.data.frame' to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula).
`x, y`	For 'lple.default', x is a design matrix of dimension n * (p+1) and y is a vector of observations of length n for a "Surv" object for "coxph".

control	a list of parameters for controlling the fitting process. See 'lplb.control' for details
...	additional arguments to be passed to the low level regression fitting functions (see below).

Details

Here 'w' is a Biomarker variable. This variable is required and shall be the last dependent variable in the formula.

'x.cdf' is a function that maps biomarker values to interval (0, 1) using its empirical cumulative distribution function.

Value

lple returns an object of class inheriting from "lple" which inherits from the class 'coxph'. See later in this section.

The function "print" (i.e., "print.lple") can be used to obtain or print a summary of the results.

An object of class "lple" is a list containing at least the following components:

beta_w	a matrix of m * p1, the estimated coefficients at each of the m estimated points, for the first p1 dependent variables with interactions of the biomarker w
Q1	the test statistic of the data
mTstar	a vector of the test statistics from B times' bootstrap
pvalue	the p-value of the hypothesis that beta_w is a constant

Note

This package was build on code developed by Yicong Liu for simple treatment-biomaker interaction model.

Author(s)

Siwei Zhang and Bingshu E. Chen (bingshu.chen@queensu.ca)

References

Zhang, S., Jiang, W. and Chen, B. E. (2016). Estimate and test of multivariate covariates and biomarker interactions for survival data based on local partial likelihood. Manuscript in preparation.

See Also

coxph, lpl.control print.lple plot.lple

Examples

```
dat = lplDemoData(50)
fit = lple(Surv(time, status)~z1 + w, data = dat, p1 = 1)
print(fit)
predict(fit)
survfit(fit, se.fit = FALSE)
```

`multiRoot` *m-Dimensional Root (Zero) Finding*

Description

The function `multiRoot` searches for root (i.e, zero) of the vector-valued function `func` with respect to its first argument using the Gauss-Newton algorithm.

Usage

```
multiRoot(func, theta, ..., verbose = FALSE, maxIter = 50,
       thetaUp = NULL, thetaLow = NULL, tol = .Machine$double.eps^0.25)
```

Arguments

`func`	a m-vector function for which the root is sought.
`theta`	the parameter vector first argument to func.
`thetaLow`	the lower bound of theta.
`thetaUp`	the upper bound of theta.
`verbose`	print out the verbose, default is FALSE.
`maxIter`	the maximum number of iterations, default is 20.
`tol`	the desired accuracy (convergence tolerance), default is .Machine$double.eps^0.25.
`...`	an additional named or unmaned arguments to be passed to `func`.

Details

The function `multiRoot` finds an numerical approximation to func(theta) = 0 using Newton method: theta = theta - solve(J, func(theta)) when m = p. This function can be used to solve the score function euqations for a maximum log likelihood estimate.

This function make use of `numJacobian` calculates an numerical approximation to the m by p first order derivative of a m-vector valued function. The parameter theta is updated by the Gauss-Newton method:

theta = theta - solve((t(J) x J), J x func(theta))

When m > p, if the nonlinear system has not solution, the method attempts to find a solution in the non-linear least squares sense (Gauss-Newton algorithm). The sum of square sum(t(U)xU), where U = func(theta), will be minimized.

Value

A list with at least four components:

root	a vector of theta that solves func(theta) = 0.
f.root	a vector of f(root) that evaluates at theta = root.
iter	number of iteratins used in the algorithm.
convergence	1 if the algorithm converges, 0 otherwise.

Author(s)

Bingshu E. Chen (bingshu.chen@queensu.ca)

References

Gauss, Carl Friedrich(1809). Theoria motus corporum coelestium in sectionibus conicis solem ambientum.

See Also

optim (which is preferred) and nlm, nlminb, numJacobian, numScore, optimize and uniroot for one-dimension optimization.

Examples

```
g = function(x, a) (c(x[1]+2*x[2]^3, x[2] - x[3]^3, a*sin(x[1]*x[2])))
theta = c(1, 2, 3)
multiRoot(g, theta, a = -3)
```

---

numHessian *Calculate Hessian or Information Matrix*

Description

Calculate a numerical approximation to the Hessian matrix of a function at a parameter value.

Usage

```
numHessian(func, theta, h = 0.0001, method=c("fast", "easy"), ...)
```

Arguments

func	a function for which the first (vector) argument is used as a parameter vector.
theta	the parameter vector first argument to func.
h	the step used in the numerical calculation.
method	one of "fast" or "easy" indicating the method to use for the approximation.
...	additional named or unmaned arguments to be passed to func.

Details

The function numHessian calculates an numerical approximation to the p by p second order derivative of a scalar real valued function with p-vector argument theta. This function can be used to check if the information matrix of a log likelihood is correct or not.

Value

An p by p matrix of the Hessian of the function calculated at the point theta. If the func is a log likelihood function, then the negative of the p by p matrix is the information matrix.

See Also

numScore

Examples

```
g = function(x, a) (x[1]+2*x[2]^3 - x[3]^3 + a*sin(x[1]*x[2]))
x0= c(1, 2, 3)
numHessian(g, theta = x0, a = 9)
numHessian(g, theta = x0, method = 'easy', a = 9)
```

---

numScore *Calculate the Score / Jacobian Function*

Description

Calculate a numerical approximation to the Score function of a function at a parameter value.

Usage

```
numScore(func, theta, h = 0.0001, ...)
numJacobian(func, theta, m, h = 0.0001, ...)
```

Arguments

func	a function for which the first (vector) argument is used as a parameter vector.
theta	the parameter vector first argument to func.
h	the step used in the numerical calculation.
m	the dimension of the function f(theta), default is 2.
...	additional named or unmaned arguments to be passed to func.

Details

The function numScore calculates an numerical approximation to the p by 1 first order derivative of a scalar real valued function with p-vector argument theta. This function can be used to check if the score function of a log likelihood is correct or not.

The function numJacobian calculates an numerical approximation to the m by p first order derivative of a m-vector real valued function with p-vector argument theta. This function can be used to find the solution of score functions for a log likelihood using the multiRoot function.

Value

An p by 1 vector of the score of the function calculated at the point `theta`. If the `func` is a log likelihood function, then the p by 1 vector is the score function.

See Also

`numHessian` `multiRoot`

Examples

```
g = function(x, a) (x[1]+2*x[2]^3 - x[3]^3 + a*sin(x[1]*x[2]))
x0 = c(1, 2, 3)
numScore(g, x0, a = -3)
```

`plot.lple` *The Plot Function of lple*

Description

Draw a series of plots of beta_w vs. w_est for each dependent variable with interactions with the biomarker w. See also: `lple`, `lpl.control`

Usage

```
## S3 method for class 'lple'
plot(x, ..., scale = c('original', 'transformed'))
```

Arguments

`x`	a lple class returned from lple fit.
`scale`	choose the scale of biomarker variable, 'original' or 'o' for the original biomarker scale. 'transformed' or 't' for transformed scale that mapps biomarker to interval (0, 1). The default is to plot in the original scale.
`...`	other options used in plot().

Details

plot.lple is called to plot the relationships between beta_w and w_est for each dependent variable with interactions with the biomarker w, from the `lple` fit model.

The number of interaction terms can be set in `lpl.control`.

The default method, print.default has its own help page. Use methods("print") to get all the methods for the print generic.

Value

No return value, called for plot model fit

Author(s)

Bingshu E. Chen and Siwei Zhang

See Also

lplb, lple, lpl.control, print.lple

Examples

```
dat = lplDemoData(50)
fit = lple(Surv(time, status)~z1 + w, data = dat, p1 = 1)
plot(fit)
```

---

predict.lple *predict a lple object*

Description

Compute fitted values and prediction error for a model fitted by lple

Usage

```
## S3 method for class 'lple'
## S3 method for class 'lple'
predict(object, newdata, newy=NULL, ...)
## S3 method for class 'lple'
residuals(object, type=c("martingale", "deviance"), ...)
```

Arguments

object	a model object from the lple fit
newdata	optional new data at which to do predictions. If absent, predictions are for the dataframe used in the original fit
newy	optional new response data. Default is NULL
type	type of residuals, the default is a martingale residual
...	additional arguments affecting the predictions produced

Details

predict.lple is called to predict object from the lple model lple.

The default method, predict has its own help page. Use methods("predict") to get all the methods for the predict generic.

Value

predict.lple returns a list of predicted values, prediction error and residuals.

`lp`	linear predictor of beta(w)*Z, where beta(w) is the fitted regression coefficient and Z is covariance matrix.
`risk`	risk score, exp(lp). When new y is provided, both lp and risk will be ordered by survival time of the new y.
`residuals`	martingale residuals of the prediction, if available.
`pe.mres`	prediction error based on martingale residual, if both new data and new y is provided.
`cumhaz`	cumulative hzard function.
`time`	time for cumulative hazard function. Time from new y will be used is provided

Author(s)

Bingshu E. Chen

See Also

The default method for predict `predict`,

For the Cox model prediction: `predict.coxph`. #`survfit.lple`

`print.lplb` *print a lplb object*

Description

print are used to provide a short summary of lplb outputs.

Usage

```
## S3 method for class 'lplb'
print(x, ...)
```

Arguments

`x`	a lplb class returned from lplb fit
`...`	other options used in print()

Details

print.lplb is called to print object or summary of object from the lplb model `lplb`.

The default method, print.default has its own help page. Use methods("print") to get all the methods for the print generic.

Value

No return value, called for printing model fit

Author(s)

Siwei Zhand and Bingshu E. Chen

See Also

The default method for print `print.default`, `lplb`

Examples

```
#
#  See examples in lplb and lple
#
```

`print.lple` *print a lple object*

Description

print are used to provide a short summary of lple outputs.

Usage

```
## S3 method for class 'lple'
print(x, ...)
```

Arguments

`x`	the results of a lple fit
`...`	other options used in print()

Details

print.lple is called to print object or summary of object from the lple model `lple`.

The default method, print.default has its own help page. Use methods("print") to get all the methods for the print generic.

Value

No return value, called for printing model fit

Author(s)

Siwei Zhand and Bingshu E. Chen

See Also

The default method for print `print.default`, `lple`

Examples

```
#
# see example in lple
#
```

`rmst` *The restricted mean survival time (RMST)*

Description

Calculate the restricted mean survival time (RMST) for Surv object, Cox proportional model and other survival objects.

Usage

```
rmst(object, ...)
rmstFit(tau, h0 = NULL, H0 = function(x){x})
## Default S3 method:
rmst(object, ...)
## S3 method for class 'coxph'
rmst(object, newdata = NULL, linear.predictors = NULL, tau=NULL, ...)
## S3 method for class 'Surv'
rmst(object, tau = NULL, ...)
```

Arguments

`object`	for rmst.Surv and rmst.default, it is a survival object created by Surv(time, event). For others, it is a model object returned by coxph, lple.
`h0`	a hazard function to be used for restricted mean survival time calculation. If h0(t) is provided, then H0(t) will be ignored.
`H0`	a cumulative hazard function to be used for restricted mean survival time calculation. The default is H0(t) = t for t>0
`linear.predictors`	the linear predictor from the Cox PH model.
`newdata`	optional new data at which the RMST is calculated. If absent, RMST is for the dataframe used in the original model fit.
`tau`	the time point at which the restricted mean survival time is calculated.
`...`	additional arguments to be passed to the functions such as rmst.coxph, rmst.lple, rmst.Surv etc.

Details

The restricted mean survival time (RMST) is the mean of the truncated survival time at some finite value tau. The RMST is defined as,

RMST(tau) = E(min(T, tau)) = int_0 ^tau S(t)dt,

where S(t) = P(T>t) is the survival function of the random variable T.

rmstFit(tau, h0, H0) calculates the restricted mean survival time based on a hazard (or cumulative hazard) function. Only one function of either h0(t) or H0(t) is required. If h0(t) is provided, then H0(t) will be ignored.

Value

A value of the Brier score or integration of the Brier score is returned.

Author(s)

Bingshu E. Chen

See Also

coxph, Surv

Examples

```
set.seed(29)
n     = 25
time  = rexp(n, 1)
event = rbinom(n, 1, 0.75)
x     = rnorm(n)
y     = Surv(time, event)

### calculate the restricted mean survival time at tau = 0.5
rms   = rmst(y, tau = 0.5)

### calculate the integrated brier score
#fit   = coxph(y~x)
#RMST  = rmst(fit, tau = 2)
```

---

rsurv *The Survival Distribution*

Description

Density, distribution function, quantile function and random variable generation for a survival distribution with a provided hazard function or cumulative hazard function

Usage

```
dsurv(x, h0 = NULL, H0 = function(x){x}, log=FALSE)
psurv(q, h0 = NULL, H0 = function(x){x}, low.tail=TRUE, log.p=FALSE)
qsurv(p, h0 = NULL, H0 = function(x){x}, low.tail=TRUE)
rsurv(n, h0 = NULL, H0 = function(x){x})
rcoxph(n, h0 = NULL, H0 = function(x){x}, lp = 0)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations.
h0	hazard function, default is h0 = NULL.
H0	cumulative hazard function, default is H0(x) = x.
lp	linear predictor for rcoxph, H(x) = H0(x)exp(lp).
log, log.p	logical; if TRUE, probabilities p are give as log(p).
low.tail	logical; if TRUE, probabilities are P[X < or = x] otherwise, S(x) = P[X>x].

Details

If { h0 } or { H0 } are not specified, they assume the default values of h0(x) = 1 and H0(x) = x, respectively.

The survival distribution function is given by,

S(x) = exp(-H0(x)),

where H0(x) is the cumulative hazard function. Only one of h0 or H0 can be specified, if h0 is given, then H0(x) = integrate(h0, 0, x, subdivisions = 500L)

To calculate the restricted mean survival time for Weibull distribution with

H = function(x) x^2 h = function(x) 2*x

use

rmst(tua, h0 = h)

or

rmst(tua, H0 = H)

when both h0 and H0 are provided, only h0 will be used and H0 will be ignored.

To generate Cox PH survival time, use

u = exp(-H(t)*exp(lp))

then, -log(u)*exp(-lp) = H(t). Find t such that H(t) = -log(u)exp(-lp).

Value

{ dsurv } gives the density h(x)/S(x), { psurv } gives the distribution function, { qsurv } gives the quantile function, { rsurv } generates random survival time, and { rcoxph } generates random survival time with Cox proportional hazards model.

The length of the result is determined by n for rsurv and rcoxph.

Author(s)

Bingshu E. Chen

References

Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995). Continuous Univariate Distributions, volume 1. Wiley, New York.

See Also

Distributions for other standard distributions, including dweibull for the Weibull distribution.

Examples

```
#### use qsurv to generate quantiles for weibull distribution
H1 = function(x) x^3
qsurv(seq(0.1, 0.9, 0.2), H0 = H1) ### shall be the same as
qweibull(seq(0.1, 0.9, 0.2), 3)
#### to get random survival time from the cumulative hazard function H1(t)
rsurv(15, H0 = H1)
```

---

survfit.lple *Compute a Survival Curve from a Local Linear Partial Likelihood Estimate.*

Description

Computes the predicted survival function for a model fitted by (lple).

Usage

```
## S3 method for class 'lple'
## S3 method for class 'lple'
survfit(formula, se.fit=TRUE, conf.int=.95, ...)
```

Arguments

formula	a fitted model from (lple) fit
se.fit	a logical value indicating whether standard errors shall be computed. Default is TRUE
conf.int	The level for a two-sided confidence interval on the survival curve. Default is 0.95
...	other arguments to the specific method

Details

survfit.lple is called to compuate baseline survival function from the lple model `lple`.

The default method, survfit has its own help page. Use methods("survfit") to get all the methods for the survfit generic.

Value

survfit.lple returns a list of predicted baseline survival function, cumulative hazard function and residuals.

`surv`	Predicted baseline survival function when beta(w) = 0.
`cumhaz`	Baseline cumulative hazard function, -log(surv).
`hazard`	Baseline hazard function.
`varhaz`	Variance of the baseline hazard.
`residuals`	Martingale residuals of the (lple) model.
`std.err`	Standard error for the cumulative hazard function, if se.fit = TRUE.

See `survfit` for more detail about other output values such as upper, lower, conf.type. Confidence interval is based on log-transformation of survival function.

Author(s)

Bingshu E. Chen

See Also

The default method for survfit `survfit`, #`survfit.lple`

Examples

```
#
# See example in lple
#
```

Index