

Package ‘lpl’

September 17, 2026

Type Package

Title Local Partial Likelihood Estimation and Simultaneous Confidence Band

Version 0.16

Date 2026-09-16

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](https://doi.org/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](https://doi.org/10.1002/sim.6563)>).

License GPL-2

LazyLoad yes

NeedsCompilation no

Repository CRAN

Date/Publication 2026-09-17 02:10:02 UTC

Contents

lpl-package	2
asymSCB	3
control	5
coxcumhaz	6
coxlogLik	8
coxScoreHess	10

csv	11
ibs	13
IPCW	15
K_func	16
lplb	18
lple	20
multiRoot	22
numHessian	23
numScore	24
oddsRatio	25
plot.lple	26
predict.lple	27
print.lplb	28
print.lple	29
rmst	30
rsurv	31
softmax	33
survfit.lple	34
Index	36

lpl-package	<i>Local Partial Likelihood Bootstrap test</i>
-------------	--

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 using 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 command

```
install_github("statapps/lpl")
```

"lpl" uses local partial likelihood to estimate 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)
```

asymSCB

Asymptotic Critical Value for Simultaneous Confidence Bands

Description

Computes an asymptotic critical value for simultaneous confidence bands (SCBs) based on the bandwidth, kernel function, and desired confidence level.

Usage

```
asymSCB(h, kernel, conf.int = 0.95)
```

Arguments

h	A positive numeric bandwidth parameter. The bandwidth should be sufficiently small for the asymptotic approximation to be appropriate.
kernel	Character string specifying the kernel function used to calculate the asymptotic critical value. Supported choices are "epanechnikov", "gaussian", "rectangular", "triangular", "biweight", "cosine", and "optcosine".
conf.int	Numeric value specifying the confidence level of the simultaneous confidence band. The default is 0.95.

Details

The function calculates an asymptotic critical value for simultaneous confidence bands.

First, an intermediate quantity d_n is calculated using the internal function `.dvn`. It is given by

$$d_n = \sqrt{-2 \log(h)} (-2 \log(h))^{-1/2} \log \left(\frac{v_1}{4\pi v_0} \right)$$

where v_0 and v_1 depend on the selected kernel function.

The final asymptotic critical value is calculated as

$$p_n = d_n + [\log(2) - \log - \log(\text{conf.int})] (-2 \log(h))^{-1/2}$$

The internal function `.dvn` calculates v_0 and v_1 numerically. The kernel is evaluated on a grid from -5 to 5 with spacing 0.001 . The quantity v_0 is obtained from the numerical integral of the squared kernel, while v_1 is obtained from the numerical integral of the squared derivative of the kernel.

The resulting value can be used as an asymptotic multiplier in the construction of simultaneous confidence bands.

Value

A numeric value `pn` giving the asymptotic critical value for the specified bandwidth, kernel, and confidence level.

See Also

[K_func](#)

Examples

```
#95 percent asymptotic critical value using the Epanechnikov kernel
asymSCB(h = 0.1, kernel = "epanechnikov")
```

```
#90 percent confidence level
asymSCB(h = 0.1, kernel = "epanechnikov", conf.int = 0.90)
```

```
#Compare different kernels
asymSCB(h = 0.1, kernel = "gaussian")
asymSCB(h = 0.1, kernel = "triangular")
asymSCB(h = 0.1, kernel = "biweight")
```

```
#Effect of the bandwidth
asymSCB(h = 0.05, kernel = "epanechnikov")
asymSCB(h = 0.10, kernel = "epanechnikov")
asymSCB(h = 0.20, kernel = "epanechnikov")
```

`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

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

Details

Control is used in model fitting of lpl.

Value

This function checks the internal consistency 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
```

coxcumhaz	<i>Baseline Hazard, Cumulative Hazard and Survival Functions for a Cox Model</i>
-----------	--

Description

Estimate the baseline hazard, cumulative hazard and survival functions for a Cox proportional hazards model using the Breslow estimator. This function can replace basehaz, which is hard to use due to the confusion of centered = TRUE / FALSE concerns.

Usage

```
coxcumhaz(y, linear.predictors = NULL, sorted = FALSE)
```

Arguments

y	A two-column numeric matrix containing the survival outcome. The first column gives the observed time and the second column the event indicator (1 = event, 0 = censored).
linear.predictors	Optional numeric vector of linear predictors $\eta_i = x_i^\top \beta$. If NULL, all relative risks are set to 1, corresponding to the baseline hazard estimate with no covariate effects.
sorted	Logical; if TRUE, the observations are assumed to be already sorted by increasing time. Use this to speed up the function when it is called repeatedly for the same y. Otherwise the function sorts the data internally.

Details

Let t_i denote the observed time and δ_i the event indicator. Define the relative risk

$$r_i = \exp(\eta_i),$$

where η_i is the supplied linear predictor. If no linear predictors are provided, $r_i = 1$.

Let $R(t)$ denote the risk set just prior to time t and define

$$S_0(t) = \sum_{j \in R(t)} r_j.$$

The Breslow estimator of the baseline hazard increment at time t_i is

$$\hat{h}(t_i) = \frac{\delta_i}{S_0(t_i)}.$$

The cumulative baseline hazard estimate is

$$\hat{H}(t) = \sum_{t_i \leq t} \hat{h}(t_i),$$

and the corresponding survival function is

$$\hat{S}(t) = \exp[-\hat{H}(t)].$$

The returned objects are interpolation functions created with `approxfun`, allowing evaluation of $\hat{H}(t)$ and $\hat{S}(t)$ at arbitrary time points.

Value

A list containing:

- `ht`: A function giving the estimated baseline hazard $\hat{H}(t)$.
- `Ht`: A function giving the estimated baseline cumulative hazard $\hat{H}(t)$.
- `St`: A function giving the estimated baseline survival function $\hat{S}(t)$.

Author(s)

Bingshu E. Chen

Examples

```
set.seed(29)

n <- 20
time <- rexp(n)
event <- rbinom(n, 1, 0.7)
y <- cbind(time, event)

# Estimate baseline cumulative hazard and survival
fit <- coxcumhaz(y)

fit$ht(1) # hazard at time 1
fit$Ht(1) # cumulative hazard at time 1
fit$St(1) # survival probability at time 1

# Example with linear predictors
lp = rnorm(n)
```

```
chz = coxcumhaz(y, linear.predictors = lp)
Ht = chz$Ht
```

coxlogLik

Compute the Log-Likelihood for a Cox Proportional Hazards Model

Description

Computes either the partial log-likelihood or the full log-likelihood for a Cox proportional hazards (PH) model.

The function evaluates the log-likelihood given a design matrix, survival outcome, regression coefficients, and optional offset and baseline hazard function. The data may optionally be pre-sorted by time to improve performance when the function is called repeatedly with the same outcome data.

Usage

```
coxlogLik(X, y, beta, offset = NULL, H0 = NULL, h0 = NULL, partial = TRUE, sorted = FALSE)
```

Arguments

X	A numeric design matrix of covariates with dimension $n \times p$.
y	Two-column numeric matrix containing survival data. The first column gives the observed time and the second column the event indicator (1 = event, 0 = censored).
beta	A numeric vector of regression coefficients of length p .
offset	Optional numeric vector of length n representing an offset added to the linear predictor. Default is NULL.
H0	Optional cumulative baseline hazard function. If provided, it should be a function that takes time values as input and returns the baseline cumulative hazard at those times. If NULL, the baseline cumulative hazard is estimated using the Breslow-type estimator when computing the full likelihood.
h0	Optional baseline hazard function. If provided, it should be a function that takes time values as input and returns the baseline hazard at those times. h0 cannot be NULL if H0 is not NULL.
partial	Logical value indicating whether to compute the Cox partial log-likelihood. If TRUE (default), the partial likelihood is returned. If FALSE, the full log-likelihood is computed.
sorted	Logical value indicating whether the observations are already sorted by decreasing time. If FALSE (default), the function sorts the data internally. If TRUE, sorting is skipped, which can improve performance when repeatedly evaluating the likelihood with the same outcome data.

Details

Let $\eta_i = x_i^\top \beta + \text{offset}_i$ denote the linear predictor and $\exp(\eta_i)$ the relative risk.

For the partial likelihood, the function computes

$$\ell(\beta) = \sum_{i=1}^n d_i \log \left(\frac{\exp(\eta_i)}{S_0(t_i)} \right),$$

where d_i is the event indicator and $S_0(t_i) = \sum_{j \in R(t_i)} \exp(\eta_j)$ is the risk set sum.

For the full likelihood, the function evaluates

$$\ell(\beta) = \sum_{i=1}^n d_i (\log h(t_i) + \eta_i) - H(t_i) \exp(\eta_i),$$

where $h(t)$ is the baseline hazard and $H(t)$ is the cumulative baseline hazard.

If `H0` is not supplied, a Breslow-type estimate based on the observed events is used.

Value

A numeric value representing the log-likelihood (partial or full) evaluated at `beta`.

Author(s)

Bingshu E. Chen

Examples

```
set.seed(29)

n <- 100
p <- 3

X <- matrix(rnorm(n * p), n, p)
beta <- c(0.5, -0.3, 0.2)

time <- rexp(n)
event <- rbinom(n, 1, 0.7)

y <- cbind(time, event)

# Partial log-likelihood
coxlogLik(X, y, beta)

# Full log-likelihood with estimated baseline hazard
coxlogLik(X, y, beta, partial = FALSE)

# Example with offset
offset <- rnorm(n)
coxlogLik(X, y, beta, offset = offset)
```

 coxScoreHess

Calculate the Score vector / Hessian matrix for the Cox model

Description

Calculate the partial log Likelihood, 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, sorted = FALSE)
```

Arguments

X	the covariate matrix from model.matrix, without the intercept term.
y	y is a survival object, $y = \text{Surv}(\text{time}, \text{event})$.
exb	exb is the relative risks with $\text{exb} = \exp(X \cdot \text{beta})$.
hess	output the Hessian matrix, with $\text{hess} = \text{FALSE}$ as the default, which outputs the score vector only.
sorted	data were sorted by time from the largest to the smallest, to speed up the algorithm, default is $\text{sorted} = \text{FALSE}$, sort by time is recommended 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.

$$\text{partial likelihood} = \sum(\text{event}(\exp(X \cdot \text{beta})/S_0))$$

$$\text{score} = \sum(\text{event} \cdot (X - S_1/S_0))$$

$$\text{Sigma} = \sum(S_1 \cdot t(S_1))$$

$$H = \sum(\text{event} \cdot (S_2/S_0 - S_1 \cdot t(S_1)/S_0))$$

the robust variance can be calculated by $\text{inv}(H) \cdot \text{Sigma} \cdot \text{inv}(H)$.

Value

An p by 1 vector of the score of the function calculated at the point relative $\exp(X \cdot \text{beta})$. If $\text{hess} = \text{TRUE}$, 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 variance of the score.
H	a p x p hessian matrix.

See Also

[numHessian](#) [numScore](#) [multiRoot](#)

Examples

```
n = 5
x = matrix(runif(n*3), nrow = n)
y = Surv(runif(n), rbinom(n, 1, 0.7))
### offset term
ot= runif(n)
### fit the coxph model
#fit = coxph(y~x+offset(ot))
#beta = fit$coefficientspartial
beta = c(0.3, 0.4, 0.5)
#cat("The Cox partial likelihood for the fitted coxph model:\n")
#print(logLik(fit))
cat("The Cox partial likelihood using the coxlogLik function:\n",
    coxlogLik(x, y, beta, offset = ot), "\n")
```

 csv

Print an Object in Comma-Separated (CSV) Style

Description

A generic function that prints an object to the console in a comma-separated style, so the output can be easily copied into a spreadsheet or a plain .csv file. Methods are provided for matrices, and for fitted model objects of class "coxph", "lm", and "survfit". There is no method for "glm" objects; such objects are handled by `csv.default`, which reports the class as unsupported.

Usage

```
csv(x, ...)

## Default S3 method:
csv(x, digits = 4, ...)

## S3 method for class 'matrix'
csv(x, digits = 4, ...)

## S3 method for class 'coxph'
csv(x, ...)

## S3 method for class 'lm'
csv(x, ...)

## S3 method for class 'survfit'
csv(x, time = NULL, digits = 3, title = FALSE, ...)
```

Arguments

<code>x</code>	an object to be printed in CSV style. For <code>csv.default</code> , <code>csv.matrix</code> , <code>csv.coxph</code> and <code>csv.lm</code> , this is the object whose values (or, for model objects, whose coefficient/summary table) will be printed.
<code>digits</code>	integer; the number of decimal places to which numeric values are rounded before printing. Defaults to 4 for the <code>default</code> and <code>matrix</code> methods, and to 3 for the <code>survfit</code> method.
<code>time</code>	a numeric vector of time points at which the survival estimate, and its confidence limits, should be reported. If <code>NULL</code> (the default), the event/censoring times stored in <code>fit\$time</code> are used.
<code>title</code>	logical; if <code>TRUE</code> , a header line ("Time, Survival percentage, 95% CI lower, 95% CI upper") is printed before the table of results. Defaults to <code>FALSE</code> .
<code>...</code>	additional arguments. For <code>csv.coxph</code> and <code>csv.lm</code> , these are passed on to <code>csv.matrix</code> (e.g. <code>digits</code>). Currently unused by <code>csv.default</code> and <code>csv.matrix</code> beyond generic method dispatch.

Details

`csv` is an S3 generic. Its intended use is quick, ad-hoc export of fitted-model summaries or numeric matrices to a comma-separated text format that prints directly to the console (via `cat`), rather than writing to a file.

`csv.default` does not print `x` at all; it simply lists the classes supported by `csv` (`coxph`, `glm`, `matrix`, `survfit`) and states that the class of the supplied object is not currently supported. Note that, despite this message, no explicit `csv.glm` method is defined, so a "glm" object will also be routed to `csv.default` unless a suitable method is added.

`csv.matrix` rounds `x` to `digits` decimal places and prints it row by row, with values separated by commas. If `x` has column names, a header row of column names is printed first (with a leading comma to align with the row-name column), and row names default to `1:n` if none are present. If `x` has no column names, no header or row names are printed and only the data values are printed for each row.

`csv.coxph` takes the coefficient table and confidence intervals from `summary(x)` for a fitted Cox proportional hazards model (as produced by `coxph`), column-binds them together, and passes the result to `csv.matrix` for printing.

`csv.lm` takes the coefficient table from `summary(x)` for a fitted linear model (as produced by `lm`) and passes it to `csv.matrix` for printing.

`csv.survfit` prints, for each requested time in `time`, the last available Kaplan–Meier estimate at or before that time: the time, the survival estimate, and the lower and upper confidence limits, each rounded to `digits` decimal places and separated by commas. Unlike the other methods, `csv.survfit` both prints its output via `cat` and **returns** a matrix (invisibly not enforced) with one column per requested time, containing the four rounded values described above.

Value

`csv.default`, `csv.matrix`, `csv.coxph`, and `csv.lm` are called for their side effect of printing to the console and do not return a useful value (they return `NULL`, invisibly, as a side effect of `cat`).

`csv.survfit` returns a numeric matrix, `sout`, with four rows (time, survival estimate, lower confidence limit, upper confidence limit) and one column for each element of `time`, in addition to printing the same information to the console.

Note

These functions print directly to the console with `cat` rather than writing to disk. To save the output as an actual `.csv` file, redirect the printed output (e.g.\ with `sink`) or use `write.csv` instead.

See Also

`cat`, `write.csv`, `coxph`, `survfit`, `lm`

Examples

```
## matrix method
m = matrix(1:6, nrow = 2, dimnames = list(c("r1", "r2"), c("a", "b", "c")))
csv(m)

## default method: unsupported class
csv(1:5)

## Not run:
## coxph method (requires the survival package)
library(survival)
fit = coxph(Surv(time, status) ~ age + sex, data = lung)
csv(fit)

## lm method
fit2 = lm(mpg ~ wt + hp, data = mtcars)
csv(fit2)

## survfit method
sfit = survfit(Surv(time, status) ~ 1, data = lung)
csv(sfit, time = c(100, 200, 300), title = TRUE)

## End(Not run)
```

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)
ibs(object, ...)
## 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 <code>ibs.Surv</code> and <code>ibs.default</code> , it is a survival object created by <code>Surv(time, event)</code> . For others, it is a model object returned by <code>coxph</code> , <code>lple</code> .
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 <code>NULL</code> .
St	the predicted survival function at time tau to calculate the Brier score.
survProb	the predicted survival function matrix. Row denotes each subject and column denotes each time points. <code>survProb[i,j]</code> denotes the predicted survival probability of the <i>i</i> th subject at the time <i>t</i> [<i>j</i>].
tau	the time point at which the Brier score is calculated.
...	additional arguments to be passed to the functions such as <code>ibs.coxph</code> , <code>ibs.lple</code> , <code>ibs.Surv</code> 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 $\int_0^{\tau} (I(T > t) - S(t))^2 dt$. 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
```

K_func	<i>Kernel Function</i>
--------	------------------------

Description

Computes a kernel function evaluated at the difference between a target value w and evaluation point u , with bandwidth h . Several commonly used kernel functions are supported, including Epanechnikov, Gaussian, rectangular, triangular, biweight, cosine, and optimal cosine kernels.

Usage

```
K_func(w, u, h,  
kernel = c("epanechnikov", "gaussian", "rectangular",  
"triangular", "biweight", "cosine", "optcosine"))
```

Arguments

- | | |
|---------------------|--|
| <code>w</code> | Numeric value or vector at which the kernel is evaluated. |
| <code>u</code> | Numeric value or vector representing the evaluation point or center of the kernel. |
| <code>h</code> | A positive numeric bandwidth parameter controlling the scale of the kernel. |
| <code>kernel</code> | Character string specifying the kernel function to use. One of "epanechnikov", "gaussian", "rectangular", "triangular", "biweight", "cosine", or "optcosine". The default is "epanechnikov". |

Details

The function computes $x = w - u$ and evaluates the selected kernel using bandwidth h .

For the Gaussian kernel, the function uses a normal density with standard deviation h and evaluates it only when $|w-u| < 5h$. Outside this range, the value is set to $1e-40$.

The other kernels have compact support on $|w-u| < h$. Their kernel functions are:

- epanechnikov: $K(x) = \frac{3}{4h} [1 - (x/h)^2]$.
- gaussian: $K(x) = \frac{1}{h} \phi(x/h)$, where ϕ is the standard normal density.
- rectangular: $K(x) = \frac{1}{2h}$.
- triangular: $K(x) = \frac{1-|x|/h}{h}$.
- biweight: $K(x) = \frac{15}{16h} [1 - (x/h)^2]^2$.
- cosine: $K(x) = \frac{1+\cos(\pi x/h)}{2h}$.
- optcosine: $K(x) = \frac{\pi}{4h} \cos(\pi x/(2h))$.

The implementation uses `match.arg` to validate the requested kernel type.

Value

A numeric value or vector containing the kernel density weights evaluated at $w - u$. Values outside the support of a compactly supported kernel are assigned a small positive value ($1e-40$) rather than zero.

See Also

[dnorm, density](#)

Examples

```
#Evaluate the Epanechnikov kernel

K_func(w = 0.5, u = 0, h = 1)

#Evaluate the Gaussian kernel

K_func(w = 0.5, u = 0, h = 1, kernel = "gaussian")

#Compare several kernel functions

x <- seq(-1, 1, length.out = 100)

K.epan <- K_func(x, 0, 1, kernel = "epanechnikov")
K.tri <- K_func(x, 0, 1, kernel = "triangular")
K.bi <- K_func(x, 0, 1, kernel = "biweight")

#Vectorized evaluation

w <- c(-0.5, 0, 0.5, 1)
K_func(w, u = 0, h = 1, kernel = "epanechnikov")
```

lplb	<i>Local partial likelihood bootstrap (LPLB) method to fit biomarker Models</i>
------	---

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

```
lplb(x, ...)
## 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 β_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	<i>Local partial likelihood estimate (LPLE) method to fit biomarker Models</i>
------	--

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 β_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	<i>m-Dimensional Root (Zero) Finding</i>
-----------	--

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

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

Details

The function `multiRoot` finds an numerical approximation to $\text{func}(\theta) = 0$ using Newton method: $\theta = \theta - \text{solve}(J, \text{func}(\theta))$ when $m = p$. This function can be used to solve the score function equations 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:

$$\theta = \theta - \text{solve}((t(J) \times J), J \times \text{func}(\theta))$$

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 $\text{sum}(t(U) \times U)$, where $U = \text{func}(\theta)$, will be minimized.

Value

A list with at least four components:

<code>root</code>	a vector of <code>theta</code> that solves $\text{func}(\theta) = 0$.
<code>f.root</code>	a vector of $f(\text{root})$ that evaluates at $\theta = \text{root}$.
<code>iter</code>	number of iterations used in the algorithm.
<code>convergence</code>	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.

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	<i>Calculate the Score / Jacobian Function</i>
----------	--

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)
```

oddsRatio

*Odds Ratios and Confidence Intervals for a Logistic Regression***Description**

Computes odds ratios, their confidence intervals, and the associated Wald test p-values for the coefficients of a fitted logistic regression model.

Usage

```
oddsRatio(x, conf.int = 0.95, ...)
```

Arguments

<code>x</code>	a fitted model object of class "glm", typically the result of a call to <code>glm</code> with <code>family = binomial</code> .
<code>conf.int</code>	numeric; the confidence level for the confidence interval on each odds ratio. Defaults to 0.95 (a 95% confidence interval).
<code>...</code>	additional arguments, currently unused.

Details

`oddsRatio` exponentiates the estimated coefficients of `x` to obtain odds ratios, and exponentiates the corresponding Wald-type confidence limits from `confint.default` (that is, limits based on asymptotic normality, estimate $\pm z * SE$, rather than profile-likelihood limits from `confint`) to obtain confidence intervals on the odds ratio scale. The two-sided Wald test p-value for each coefficient, taken from `summary(x)$coefficients[, "Pr(>|z|)"]`, is included alongside the odds ratio and its confidence interval.

The intercept, if present, is included in the output along with all other coefficients; its exponentiated value does not have an odds ratio interpretation in the usual sense and can be ignored or removed by the user if not needed.

Value

A numeric matrix with one row per coefficient in `x` (in the same order as `coef(x)`) and four columns:

<code>OR</code>	the odds ratio, <code>exp(coef(x))</code> .
<code>CI_lower</code>	the lower confidence limit for the odds ratio, at the level given by <code>conf.int</code> .
<code>CI_upper</code>	the upper confidence limit for the odds ratio, at the level given by <code>conf.int</code> .
<code>p_value</code>	the Wald test p-value for the corresponding coefficient, from <code>summary(x)</code> .

Note

Confidence intervals are computed with `confint.default` (Wald-based), not with the (generally preferred, but slower) profile-likelihood intervals from `confint`. For models with small samples or coefficients near a boundary, the two methods can disagree noticeably.

See Also

`glm`, `coef`, `confint.default`, `confint`, `summary.glm`

Examples

```
## a simple logistic regression
fit = glm(am ~ mpg + wt, data = mtcars, family = binomial)
oddsRatio(fit)

## with a 90% confidence interval instead of the 95% default
oddsRatio(fit, conf.int = 0.90)
```

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

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

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	<i>predict a lple object</i>
--------------	------------------------------

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 hazard 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](#)

<code>print.lplb</code>	<i>print a lplb object</i>
-------------------------	----------------------------

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	<i>print a lple object</i>
------------	----------------------------

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

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

Details

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

$$\text{RMST}(\tau) = E(\min(T, \tau)) = \int_0^\tau S(t) dt,$$

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 $h_0(t)$ or $H_0(t)$ is required. If $h_0(t)$ is provided, then $H_0(t)$ 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)
```

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 $h_0(x) = 1$ and $H_0(x) = x$, respectively.

The survival distribution function is given by,

$$S(x) = \exp(-H_0(x)),$$

where $H_0(x)$ is the cumulative hazard function. Only one of h_0 or H_0 can be specified, if h_0 is given, then $H_0(x) = \text{integrate}(h_0, 0, x, \text{subdivisions} = 500L)$

To calculate the restricted mean survival time for Weibull distribution with

$$H = \text{function}(x) \ x^2 \ h = \text{function}(x) \ 2*x$$

use

```
rmst(tua, h0 = h)
```

or

```
rmst(tua, H0 = H)
```

when both h_0 and H_0 are provided, only h_0 will be used and H_0 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)
```

softmax

Softmax Activation Function

Description

Computes the softmax transformation for a numeric vector or matrix. The softmax function converts real-valued inputs into probabilities that sum to 1. A numerically stable implementation is used by subtracting the maximum value before exponentiation.

For matrix input, the softmax transformation is applied row-wise, so each row is converted into a probability distribution whose elements sum to 1.

Usage

```
softmax(x)
```

Arguments

x A numeric vector or numeric matrix.

Details

The softmax function is defined as:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

To improve numerical stability, the implementation subtracts the maximum value from the input before exponentiation:

$$\frac{e^{x_i - \max(x)}}{\sum_j e^{x_j - \max(x)}}$$

Value

If `x` is a vector, returns a numeric vector of the same length containing softmax probabilities that sum to 1.

If `x` is a matrix, returns a numeric matrix of the same dimensions where each row contains softmax probabilities summing to 1.

Author(s)

Bingshu E. Chen

See Also

[exp](#)

Examples

```
# Vector input
x <- c(1.2, 0.3, 2.4)
softmax(x)

# Matrix input
m <- matrix(c(1, 2, 3,
              0, 1, 0),
            nrow = 2,
            byrow = TRUE)

softmax(m)

# Row sums equal 1
rowSums(softmax(m))
```

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 compute 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, $-\log(\text{surv})$.
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

- * **Brier Score**
 - ibs, [13](#)
 - IPCW, [15](#)
 - * **Cox PH random variable**
 - rsurv, [31](#)
 - * **IPCW**
 - ibs, [13](#)
 - IPCW, [15](#)
 - * **Restricted Mean Survival Time**
 - rmst, [30](#)
 - * **Survival Analysis**
 - rmst, [30](#)
 - * **Survival distribution**
 - rsurv, [31](#)
 - * **biomarker interaction**
 - lplb, [18](#)
 - lple, [20](#)
 - * **biomarker**
 - lpl-package, [2](#)
 - * **bootstrap**
 - lplb, [18](#)
 - * **confidence**
 - asymSCB, [3](#)
 - * **control**
 - control, [5](#)
 - * **kernel**
 - K_func, [16](#)
 - * **local linear model**
 - lpl-package, [2](#)
 - * **local partial likelihood**
 - lplb, [18](#)
 - lple, [20](#)
 - * **lple**
 - plot.lple, [26](#)
 - * **lpl**
 - lpl-package, [2](#)
 - * **math**
 - softmax, [33](#)
 - * **methods**
 - csv, [11](#)
 - * **models**
 - oddsRatio, [25](#)
 - * **nonparametric**
 - asymSCB, [3](#)
 - K_func, [16](#)
 - * **plot**
 - plot.lple, [26](#)
 - * **predict**
 - predict.lple, [27](#)
 - * **print**
 - csv, [11](#)
 - print.lplb, [28](#)
 - print.lple, [29](#)
 - * **regression**
 - oddsRatio, [25](#)
 - * **smooth**
 - asymSCB, [3](#)
 - K_func, [16](#)
 - * **survfit**
 - survfit.lple, [34](#)
- asymSCB, [3](#)
- brier, [15](#)
- brierScore, [16](#)
- brierScore (ibs), [13](#)
- cat, [12](#), [13](#)
- coef, [26](#)
- confint, [25](#), [26](#)
- confint.default, [25](#), [26](#)
- control, [5](#)
- coxcumhaz, [6](#)
- coxlogLik, [8](#)
- coxph, [12](#), [13](#), [15](#), [19](#), [21](#), [31](#)
- coxScoreHess, [10](#)
- csv, [11](#)
- density, [17](#)

Distributions, 33
dnorm, 17
dsurv (rsurv), 31
dweibull, 33

exp, 34

glm, 25, 26

ibs, 13, 16
IPCW, 14, 15, 15
ipcw (IPCW), 15

K_func, 4, 16

lm, 12, 13
lpl-doc (lpl-package), 2
lpl-package, 2
lpl.control, 19, 21, 26, 27
lpl.control (control), 5
lplb, 5, 18, 27–29
lplDemoData (lple), 20
lple, 5, 20, 26, 27, 29, 30, 35
lple_fit (lple), 20
lple_se (lple), 20

maxTest (lplb), 18
multiRoot, 10, 22, 24

nlm, 23
nlminb, 23
numHessian, 10, 23, 24
numJacobian, 23
numJacobian (numScore), 24
numScore, 10, 23, 24, 24

oddsRatio, 25
optim, 23
optimize, 23

plot.lple, 19, 21, 26
predict, 28
predict.coxph, 28
predict.lple, 27
print.default, 29, 30
print.lplb, 28
print.lple, 19, 21, 27, 29
psurv (rsurv), 31

qsurv (rsurv), 31

rcoxph (rsurv), 31
residuals.lple (predict.lple), 27
rmst, 30
rmstFit (rmst), 30
rsurv, 31

sink, 13
softmax, 33
summary.glm, 26
Surv, 31
survfit, 13, 16, 35
survfit.lple, 28, 34, 35

uniroot, 23

write.csv, 13