Package 'EstimationTools'

September 9, 2025

Type Package

Title Maximum Likelihood Estimation for Probability Functions from Data Sets

Version 4.3.1

Depends R (>= 3.5.0), survival

Imports DEoptim, BBmisc, GA, Rdpack (>= 0.7), utils, stats, numDeriv, boot, autoimage, graphics, stringr, gaussquad, car

RdMacros Rdpack

Suggests Rmpfr, gamlss.dist, knitr, rmarkdown, AdequacyModel, readr, covr, testthat (>= 3.0.0), vdiffr, spelling, lifecycle, matrixStats, lintr, V8

VignetteBuilder knitr, utils

Description Total Time on Test plot and routines for parameter estimation of any lifetime distribution implemented in R via maximum likelihood (ML) given a data set. It is implemented thinking on parametric survival analysis, but it feasible to use in parameter estimation of probability density or mass functions in any field. The main routines 'maxlogL' and 'maxlogLreg' are wrapper functions specifically developed for ML estimation. There are included optimization procedures such as 'nlminb' and 'optim' from base package, and 'DEoptim' Mullen (2011) <doi:10.18637/jss.v040.i06>. Standard errors are estimated with 'numDeriv' Gilbert (2011) <https://CRAN.R-project.org/package=numDeriv> or the option 'Hessian = TRUE' of 'optim' function.

License GPL-3

URL https://jaimemosg.github.io/EstimationTools/, https://github.com/Jaimemosg/EstimationTools

BugReports https://github.com/Jaimemosg/EstimationTools/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Config/testthat/edition 3

Language en-US

Collate 'ALL-data.R' 'Cells-data.R' 'custom_optimizer.R'
'Fibers-data.R' 'Hazard_Shape.R' 'head-neck-cancer-data.R'
'LinkFunc.R' 'TTTE_Analytical.R' 'TTT_hazard_shape.R'
'bootstrap_maxlogL.R' 'coef.maxlogL.R' 'expected_value.R'
'gauss_quad.R' 'generic_funcs_EstimationTools.R' 'globals.R'
'hazard_fun.R' 'inherit_methods.R' 'integration.R' 'internal.R'
'interp.options.R' 'key_functions.R' 'legend.HazardShape.R'
'loess.options.R' 'maxlogL.R' 'maxlogLreg.R'
'plot.HazardShape.R' 'plot.empiricalTTT.R' 'residuals.R'
'plot.maxlogL.R' 'predict.maxlogL.R' 'print.HazardShape.R'
'summary.maxlogL.R' 'summate.R' 'utils.R' 'zzz.R'

NeedsCompilation no

Author Jaime Mosquera [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0002-1684-4756>),
Freddy Hernandez [ctb] (ORCID: <https://orcid.org/0000-0001-7459-3329>)

Maintainer Jaime Mosquera <jmosquerag@unal.edu.co>

Repository CRAN

Date/Publication 2025-09-09 06:50:35 UTC

Contents

`ALL_colosimo_table_4_1`

Acute Lymphoblastic Leukemia, table 4.1

Description

Survival times, in weeks, of 17 patients with acute leukemia. For these patients, their white blood cell counts (WBC) were recorded at the time of diagnosis. The variables are:

Usage

```
ALL_colosimo_table_4_1
```

Format

A data frame with 17 observations.

Details

- times: time-to-event (in weeks).
- status: censorship indicators.
- wbc: WBC, white blood cells counts.
- lwbc: base-10 logarithms of the WBC.

References

Colosimo EA, Ruiz Giolo S (2006). "Modelos de Regressao Parametricos." In *Análise de Sobrevivência Aplicada*, 123–134. Edgard Blucher, Sao Paulo. ISBN 978-8521203841.

Examples

```
data(ALL_colosimo_table_4_1)
hist(ALL_colosimo_table_4_1$time, main="", xlab="Time (Weeks)")
```

`ALL_colosimo_table_4_3`

Acute Lymphoblastic Leukemia, table 4.3

Description

Survival times, in weeks, of 17 patients with acute leukemia. For these patients, their white blood cell counts (WBC) were recorded at the time of diagnosis. The variables are:

Usage

```
ALL_colosimo_table_4_3
```

Format

A data frame with 33 observations.

Details

- times: time-to-event (in weeks).
- status: censorship indicators.
- wbc: WBC, white blood cells counts.
- lwbc: base-10 logarithms of the WBC.
- Ag_plus: whether the subject expresses the Calla antigen (`Ag_plus` = 1) or not (`Ag_plus` = 0).

References

Colosimo EA, Ruiz Giolo S (2006). "Modelos de Regressao Parametricos." In *Análise de Sobrevivência Aplicada*, 123–134. Edgard Blucher, Sao Paulo. ISBN 978-8521203841.

Examples

```
data(ALL_colosimo_table_4_3)
hist(ALL_colosimo_table_4_3$time, main="", xlab="Time (Weeks)")
```

bootstrap_maxlogL *Bootstrap computation of standard error for* maxlogL *class objects.*

Description

[Experimental]

bootstrap_maxlogL computes standard errors of maxlogL class objects by non-parametric bootstrap.

Usage

```
bootstrap_maxlogL(object, R = 2000, silent = FALSE, ...)
```

Arguments

object	an object of maxlogL class whose standard errors are going to be computed by bootstrap.
R	numeric. It is the number of resamples performed with the dataset in bootstrap computation. Default value is 2000.
silent	logical. If TRUE, notifications of bootstrap_maxlogL are suppressed.
...	arguments passed to boot used in this routine for estimation of standard errors.

Details

The computation performed by this function may be invoked when Hessian from optim and hessian fail in maxlogL or in maxlogLreg.

However, this function can be run even if Hessian matrix calculation does not fails. In this case, standard errors in the maxlogL class object is replaced.

Value

A modified object of class maxlogL.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

References

Canty A, Ripley BD (2017). *boot: Bootstrap R (S-Plus) Functions*.

See Also

maxlogL, maxlogLreg, boot

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------
# First example: Comparison between standard error computation via Hessian matrix
# and standard error computation via bootstrap

N <- rbinom(n = 100, size = 10, prob = 0.3)
phat1 <- maxlogL(x = N, dist = 'dbinom', fixed = list(size = 10),
                 link = list(over = "prob", fun = "logit_link"))

## Standard error computation method and results
print(phat1$outputs$StdE_Method)   # Hessian
summary(phat1)

## 'bootstrap_maxlogL' implementation
phat2 <- phat1                   # Copy the first 'maxlogL' object
bootstrap_maxlogL(phat2, R = 100)

## Standard error computation method and results
print(phat2$outputs$StdE_Method)   # Bootstrap
summary(phat2)


#---------------------------------------------------------------------------
```

`coef.maxlogL` *Extract Model Coefficients in a* `maxlogL` *Fits*

Description

[Maturing]

`coef.maxlogL` is the specific method for the generic function `coef` which extracts model coefficients from objects returned by `maxlogLreg`. `coefficients` is an alias for `coef`.

Usage

```
## S3 method for class 'maxlogL'
coef(object, parameter = object$outputs$par_names, ...)

coefMany(object, parameter = NULL, ...)
```

Arguments

`object`	an object of `maxlogL` class generated by `maxlogLreg` function.
`parameter`	a character which specifies the parameter is required. In `coefMany` this argument can be an atomic vector with two or more names of parameters.
`...`	other arguments.

Value

A named vector with coefficients of the specified distribution parameter.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

Examples

```
library(EstimationTools)

#-------------------------------------------------------------------------------
# Example 1: coefficients from a model using a simulated normal distribution
n <- 1000
x <- runif(n = n, -5, 6)
y <- rnorm(n = n, mean = -2 + 3 * x, sd = exp(1 + 0.3* x))
norm_data <- data.frame(y = y, x = x)

# It does not matter the order of distribution parameters
formulas <- list(sd.fo = ~ x, mean.fo = ~ x)

norm_mod <- maxlogLreg(formulas, y_dist = y ~ dnorm, data = norm_data,
                       link = list(over = "sd", fun = "log_link"))
coef(norm_mod)
coef(norm_mod, parameter = 'sd')
a <- coefMany(norm_mod, parameter = c('mean', 'sd'))
b <- coefMany(norm_mod)
identical(a, b)


#-------------------------------------------------------------------------------
# Example 2: Parameters in estimation with one fixed parameter
x <- rnorm(n = 10000, mean = 160, sd = 6)
theta_1 <- maxlogL(x = x, dist = 'dnorm', control = list(trace = 1),
                 link = list(over = "sd", fun = "log_link"),
                 fixed = list(mean = 160))
coef(theta_1)


#-------------------------------------------------------------------------------
```

`cum_hazard.maxlogL` *Cumuative hazard function of a* `maxlogLreg` *model.*

Description

[Experimental]

This function takes a `maxlogL` model and computes the cumulative hazard function (CHF) using the estimated parameters.

Usage

```
cum_hazard.maxlogL(object, ...)
```

Arguments

object	an object of maxlogL class obtained by fitting a model with maxlogLreg.
...	further arguments for cum_hazard_fun..

Details

The CHF is computed by default using the following expression

$$H(x) = -\log\left(S(x|\hat{\theta})\right),$$

where $S(x|\hat{\theta})$ is the survival function using the estimated parameters. This method relies on the cdf, i.e, the pXXX function stored in R environment, where xxx is the name of the distribution.

Notice that CHF can be computed by integration

$$H(x) = \int_0^t h(s)ds$$

Just set up a support and set method = "integration".

Value

the expected value of the fitted model corresponding to the distribution specified in the y_dist argument of maxlogLreg.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

See Also

Other maxlogL: expected_value.maxlogL(), maxlogL(), maxlogLreg()

Examples

```
library(EstimationTools)

#-------------------------------------------------------------------------
# Example 1: cumulative hazard function of a estimated model.
n <- 100
x <- runif(n = n, -5, 6)
y <- rnorm(n = n, mean = -2 + 3 * x, sd = 0.3)
norm_data <- data.frame(y = y, x = x)

formulas <- list(sd.fo = ~ 1, mean.fo = ~ x)
support <- list(interval = c(-Inf, Inf), type = "continuous")
```

```
norm_mod_maxlogL <- maxlogLreg(
  formulas, y_dist = y ~ dnorm,
  support = support,
  data = norm_data,
  link = list(over = "sd", fun = "log_link")
)

# Expected value
H <- cum_hazard.maxlogL(object = norm_mod_maxlogL)


#------------------------------------------------------------------------
```

`cum_hazard_fun` *Cumulative hazard functions for any distribution*

Description

[Experimental]

This function takes a `maxlogL` hazard function and computes the cumulative hazard function.

Usage

```
cum_hazard_fun(
  distr,
  support = NULL,
  method = c("log_sf", "integration"),
  routine = NULL
)
```

Arguments

`distr`	a length-one character vector with the name of density/mass function of interest.
`support`	a list with the following entries: • `interval`: a two dimensional atomic vector indicating the set of possible values of a random variable having the distribution specified in `y_dist`. • `type`: character indicating if distribution has a `discrete` or a `continous` random variable.
`method`	a character or function; if `"log_sf"`, the cumulative hazard function (CHF) is computed using the expression $H(t) = -\log(S(t))$; if `"integrate_hf"`, the CHF is computed with the integral of the hazard function.
`routine`	a character specifying the integration routine. `integrate` and `gauss_quad` are available for continuous distributions, and `summate` for discrete ones. Custom routines can be defined but they must be compatible with the `integration` API.

Value

A function with the following input arguments:

x	vector of (non-negative) quantiles.
...	Arguments of the probability density/mass function.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

See Also

Other distributions utilities: expected_value(), hazard_fun()

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------
# Example 1: Cumulative hazard function of the Weibull distribution.
support <- list(interval=c(0, Inf), type='continuous')

# Cumuative hazard function in the 'maxlogL' framework
Hweibull1 <- cum_hazard_fun(
  distr = 'dweibull',
  support = support,
  method = "integration"
 )

 Hweibull2 <- cum_hazard_fun(
  distr = 'dweibull',
  method = "log_sf"
 )

# Compute cumulative hazard function from scratch
# Recall h(x) = shape/scale * (x/scale)^(shape - 1), then
# H(x) = (x/scale)^shape

Hweibull3 <- function(x, scale, shape){
  (x/scale)^shape
}

# Comparison
Hweibull1(0.2, shape = 2, scale = 1)  # using H(t) = -log(S(t))
Hweibull2(0.2, shape = 2, scale = 1)  # integrating h(t)
Hweibull3(0.2, shape = 2, scale = 1)  # raw version


#---------------------------------------------------------------------------
```

`expected_value` *Expected value of a given function for any distribution*

Description

[Experimental]

This function takes the name of a probability density/mass function as an argument and creates a function to compute the expected value.

Usage

```
expected_value(f, parameters, support, g = identity, routine = NULL, ...)
```

Arguments

`f`	a character with the probability density/mass function name. The function must be availble in the `R` environment using the usual nomenclature (`d` prefix before the name).
`parameters`	a list with the input parameters for the distribution.
`support`	a list with the following entries: • `interval`: a two dimensional atomic vector indicating the set of possible values of a random variable having the distribution specified in `y_dist`. • `type`: character indicating if distribution has a `discrete` or a `continous` random variable.
`g`	a given function $g(x)$. If `g = identity`, then $g(x) = x$ and this is actually the mean of the distribution.
`routine`	a character specifying the integration routine. `integrate` and `gauss_quad` are available for continuous distributions, and `summate` for discrete ones. Custom routines can be defined but they must be compatible with the `integration` API.
`...`	further arguments for the integration routine.

Value

the expected value of the specified distribution.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

See Also

Other distributions utilities: `cum_hazard_fun()`, `hazard_fun()`

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------
# Example 1: mean of X ~ N(2, 1) using 'integrate' under the hood.
support <- list(interval=c(-Inf, Inf), type = "continuous")

expected_value(
  f = "dnorm",
  parameters = list(mean = 2, sd = 1),
  support = support
)

# Equivalent to
expected_value(
  f = "dnorm",
  parameters = list(mean = 2, sd = 1),
  support = support,
  g = identity,
  routine = "integrate"
)

# Example 1: mean of X ~ N(22, 1)

# 'integrate' fails because the mean is 22.
expected_value(
  f = "dnorm",
  parameters = list(mean = 22, sd = 1),
  support = support
)

# Let's compute with Monte Carlo integration
expected_value(
  f = "dnorm",
  parameters = list(mean = 22, sd = 1),
  support = support,
  routine = "monte-carlo"
)

# Compute Monte Carlo integration with more samples

expected_value(
  f = "dnorm",
  parameters = list(mean = 22, sd = 1),
  support = support,
  routine = "monte-carlo",
  n = 1e8
)


#---------------------------------------------------------------------------
```

expected_value.maxlogL *Expected value of a* maxlogLreg *model.*

Description

[Experimental]

This function takes a maxlogL model and computes the expected value using the estimated parameters. The expected value is computed using the following expression

$$E[\hat{g(X)}] = \int_{-\infty}^{\infty} x f(x|\hat{\theta}) dx,$$

where $f(x|\hat{\theta})$ is a probability density function using the estimated parameters.

Usage

```
expected_value.maxlogL(object, g = identity, routine, ...)
```

Arguments

object	an object of maxlogL class obtained by fitting a model with maxlogLreg.
g	a given function $g(x)$.
routine	a character specifying the integration routine. integrate and gauss_quad are available for continuous distributions, and summate for discrete ones. Custom routines can be defined but they must be compatible with the integration API.
...	further arguments for the integration routine.

Value

the expected value of the fitted model corresponding to the distribution specified in the y_dist argument of maxlogLreg.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

See Also

Other maxlogL: cum_hazard.maxlogL(), maxlogL(), maxlogLreg()

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------
# Example 1: mean value of a estimated model.
n <- 100
x <- runif(n = n, -5, 6)
y <- rnorm(n = n, mean = -2 + 3 * x, sd = 0.3)
norm_data <- data.frame(y = y, x = x)

formulas <- list(sd.fo = ~ 1, mean.fo = ~ x)
support <- list(interval = c(-Inf, Inf), type = "continuous")

norm_mod_maxlogL <- maxlogLreg(
  formulas, y_dist = y ~ dnorm,
  support = support,
  data = norm_data,
  link = list(over = "sd", fun = "log_link")
)

# Actual y values
y <- norm_mod_maxlogL$outputs$response

# Expected value
Ey <- expected_value.maxlogL(
  object = norm_mod_maxlogL,
  routine = "monte-carlo"
)

# Compare
plot(y, Ey)


#---------------------------------------------------------------------------
```

Fibers *Tensile strengths*

Description

Tensile strengths (in GPa) of 69 specimens of carbon fiber tested under tension at gauge lengths of 20 mm.

Usage

```
Fibers
```

Format

A data frame with 69 observations.

References

Ghitany ME, Al-Mutairi DK, Balakrishnan N, Al-Enezi LJ (2013). "Power Lindley distribution and associated inference." *Computational Statistics and Data Analysis*, **64**, 20–33. ISSN 01679473, doi:10.1016/j.csda.2013.02.026, http://dx.doi.org/10.1016/j.csda.2013.02.026.

Examples

```
data(Fibers)
hist(Fibers$Strenght, main="", xlab="Strength (GPa)")
```

---

gauss_quad *Numerical integration through Gaussian Quadrature*

Description

[Experimental]

This family of functions use quadratures for solving integrals. The user can create a custom integration routine, see *details* for further information.

Usage

```
gauss_quad(
  fun,
  lower,
  upper,
  kind = "legendre",
  n = 10,
  normalized = FALSE,
  ...
)
```

Arguments

fun	an **R** function which should take a numeric argument x and possibly some parameters. The function returns a numerical vector value for the given argument x.
lower	a numeric value for the lower limit of the integral.
upper	a numeric value for the upper limit of the integral.
kind	character specifying the weight (polynomial) function for the quadrature.
n	integer with the highest order of the polynomial of the selected rule.

normalized	logical. If TRUE, rules are for orthonormal polynomials, otherwise they are for orthogonal polynomials.
...	additional arguments to be passed to fun and to the quadrature routine specified in argument kind.

Details

gauss_quad uses the implementation of Gaussian quadratures from **gaussquad** package.This is a wrapper that implements rules and integration routine in the same place.

Value

The value of the integral of the function specified in fun argument.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

See Also

laguerre.quadrature, legendre.quadrature, chebyshev.c.quadrature, gegenbauer.quadrature, hermite.h.quadrature, etc.

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------
# Example 1: Mean of X ~ N(2,1) (Gauss-Hermitie quadrature).
g <- function(x, mu, sigma) sqrt(2)*sigma*x + mu
i2 <- gauss_quad(g, lower = -Inf, upper = Inf, kind = 'hermite.h',
                 normalized = FALSE, mu = 2, sigma = 1)
i2 <- i2/sqrt(pi)
i2


#---------------------------------------------------------------------------
```

---

half_norm_key *Half normal key function*

Description

[Experimental]

This function provides the half normal key function for model fitting in distance sampling.

Usage

```
half_norm_key(x, sigma)
```

Arguments

x	vector of perpendicular distances from the transect.
sigma	scale parameter.

Details

This is the half normal key function with parameter sigma. Its expression is given by

$g(x) = \exp(\frac{-x^2}{2*\sigma^2},$

for x > 0.

Value

A numeric value corresponding to a given value of x and sigma.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

See Also

Other key functions: hazard_rate_key(), uniform_key()

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------
# Example: Half normal function
half_norm_key(x=1, sigma=4.1058)
curve(half_norm_key(x, sigma=4.1058), from=0, to=20, ylab='g(x)')

#---------------------------------------------------------------------------
```

---

hazard_fun *Hazard functions for any distribution*

Description

[Experimental]

This function takes the name of a probability density/mass function as an argument and creates a hazard function.

Usage

```
hazard_fun(distr, log = FALSE)
```

Arguments

distr	a length-one character vector with the name of density/mass function of interest.
log	logical; if TRUE, the natural logarithm of the hazard values are returned.

Value

A function with the folling input arguments:

x	vector of (non-negative) quantiles.
...	Arguments of the probability density/mass function.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

See Also

Other distributions utilities: cum_hazard_fun(), expected_value()

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------
# Example 1: Hazard function of the Weibull distribution.

# Hazard function in the 'maxlogL' framework
hweibull1 <- hazard_fun('dweibull')

# Hazard function from scratch
hweibull2 <- function(x, shape, scale){
  shape/scale * (x/scale)^(shape - 1)
}

# Comparison
hweibull1(0.2, shape = 2, scale = 1)
hweibull2(0.2, shape = 2, scale = 1)


#---------------------------------------------------------------------------
```

hazard_rate_key *Hazard rate key function*

Description

[Experimental]

This function provides the hazard rate key function for model fitting in distance sampling.

Usage

```
hazard_rate_key(x, sigma, beta)
```

Arguments

x	vector of perpendicular distances from the transect.
sigma	scale parameter.
beta	shape parameter.

Details

This is the hazard rate key function with parameters sigma and beta. Its expression is given by

$g(x) = 1 - \exp((\frac{-x}{\sigma})^{-\beta}$,

for x > 0.

Value

A numeric value corresponding to a given value of x, sigma and beta.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

See Also

Other key functions: half_norm_key(), uniform_key()

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------
# Example: Hazard rate function
hazard_rate_key(x=1, sigma=2, beta=3)
curve(hazard_rate_key(x, sigma=2, beta=3), from=0, to=10, ylab='g(x)')

#---------------------------------------------------------------------------
```

head_neck_cancer *Head and neck cancer*

Description

Time-to-event data a randomized clinical trial to compare two therapies for head and neck cancer.51 patients were treated with radiation only and 45 patients treated with radiation plus chemotherapy. The variables are:

Usage

```
head_neck_cancer
```

Format

A data frame with 96 observations.

Details

- Time: time (in days) to recurrence of the cancer.
- Therapy: treatment applied to the patients.
- Status: censorship indicators.

References

Khan SA (2018). "Exponentiated Weibull regression for time-to-event data." *Lifetime Data Analysis*, **24**(2), 328–354. ISSN 1380-7870, doi:10.1007/s1098501793943, https://link.springer.com/article/10.1007/s10985-017-9394-3.

Examples

```
data(head_neck_cancer)
par(mfrow = c(1,2))
hist(head_neck_cancer$Time, main="", xlab="Time (Days)")
plot(head_neck_cancer$Time, xlab = "Patient (subjects)", lty = 3, type="h")
```

integration *Integration*

Description

[Experimental]

This is a wrapper routine for integration in maxlogL framework. It is used in integration for compute detectability density functions and in computation of mean values, but it is also a general purpose integrator.

Usage

```
integration(fun, lower, upper, routine = "integrate", ...)
```

Arguments

fun	an **R** function which should take a numeric argument x and possibly some parameters. The function returns a numerical vector value for the given argument x.
lower	a numeric value for the lower limit of the integral.
upper	a numeric value for the upper limit of the integral.
routine	a character specifying the integration routine. integrate and gauss_quad are available, but the custom routines can be defined.
...	additional arguments to be passed to fun and to the integration routine specified in routine argument.

Details

The user can create custom integration routines through implementation of a wrapper function using three arguments

fun: a function which should take a numeric argument x and possibly some parameters.

lower: a numeric value for the lower limit of the integral.

upper: a numeric value for the upper limit of the integral.

... furthermore, the user must define additional arguments to be passed to fun function.

The output must be a numeric atomic value with the result of the integral.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

See Also

integrate, gauss_quad

Examples

```
library(EstimationTools)


#---------------------------------------------------------------------------
# Example 1: Mean of X ~ N(2,1) using 'integrate'.
mynorm <- function(x, mu, sigma) x*dnorm(x, mean = mu, sd = sigma)
i1 <- integration(mynorm, lower = -Inf, upper = Inf, mu = 2, sigma = 1)
i1

#---------------------------------------------------------------------------
# Example 2: Mean of X ~ N(2,1) using 'gauss_quad' (Gauss-Hermitie
#            quadrature).
g <- function(x, mu, sigma) sqrt(2)*sigma*x + mu
i2 <- integration(g, lower = -Inf, upper = Inf, routine = 'gauss_quad',
                  kind = 'hermite.h', normalized = FALSE,
                  mu = 2, sigma = 1)
i2 <- i2/sqrt(pi)
i2


#---------------------------------------------------------------------------
# Example 3: replicating integrate
i3 <- integrate(dnorm, lower=-1.96, upper=1.96)
i4 <- integration(dnorm, lower=-1.96, upper=1.96)
identical(i3$value, i4)

#---------------------------------------------------------------------------
```

`interp.options` *Configure various aspects of interpolating function in* TTT_hazard_shape

---

Description

[Maturing]

This function allows the user to set the parameters of any of the following interpolating functions which can be used inside `TTT_hazard_shape`.

Usage

```
interp.options(interp.fun = "splinefun", length.out = 10, ...)
```

Arguments

`interp.fun`	character. This argument defines the interpolating function used. Default value is `"splinefun"`. Visit the **Details** section for further information.
`length.out`	numeric. Number of points interpolated. Default value is 10.
`...`	further arguments passed to the interpolating function.

Details

Each interpolating function has its particular arguments. The following interpolating functions are recommended:

- approxfun
- splinefun
- spline

The user can also implement a custom interpolating function.

Author(s)

Jaime Mosquera Gutiérrez <jmosquerag@unal.edu.co>

See Also

approxfun, splinefun, smooth, smooth.spline, loess, TTT_hazard_shape

---

is.maxlogL *Is return of any object of* EstimationTools*?*

Description

[Stable]

Checks if an object is any of the classes implemented in EstimationTools package.

Usage

```
is.maxlogL(x)

is.EmpiricalTTT(x)

is.HazardShape(x)

is.optimizer.config(x)
```

Arguments

x	an object of EstimationTools package.

Author(s)

Jaime Mosquera Gutiérrez <jmosquerag@unal.edu.co>

`legend.HazardShape` *Customize legend for* `plot.HazardShape` *outputs*

Description

[Maturing]

Put legend after run `plot.HazardShape` function.

Usage

```
legend.HazardShape(
  x,
  y = NULL,
  legend = c("Empirical TTT", "Spline curve"),
  fill = NULL,
  col = 1,
  border = "white",
  lty = NA,
  lwd = NA,
  pch = c(1, NA),
  angle = 45,
  density = NULL,
  bty = "o",
  bg = par("bg"),
  box.lwd = par("lwd"),
  box.lty = par("lty"),
  box.col = par("fg"),
  pt.bg = NA,
  cex = 1,
  pt.cex = cex,
  pt.lwd = lwd,
  xjust = 0,
  yjust = 1,
  x.intersp = 1,
  y.intersp = 1,
  adj = c(0, 0.5),
  text.width = NULL,
  text.col = par("col"),
  text.font = NULL,
  merge = TRUE,
  trace = FALSE,
  plot = TRUE,
  ncol = 1,
  horiz = FALSE,
  title = NULL,
  inset = 0,
```

```
  xpd = TRUE,
  title.col = text.col[1],
  title.adj = 0.5,
  title.cex = cex[1],
  title.font = text.font[1],
  seg.len = 2,
  curve_options = list(col = 2, lwd = 2, lty = 1)
)
```

Arguments

x, y	the x and y co-ordinates to be used to position the legend. They can be specified by keyword or in any way which is accepted by xy.coords: See 'Details'.
legend	a character or expression vector of length ≥ 1 to appear in the legend. Other objects will be coerced by as.graphicsAnnot.
fill	if specified, this argument will cause boxes filled with the specified colors (or shaded in the specified colors) to appear beside the legend text.
col	the color of points or lines appearing in the legend.
border	the border color for the boxes (used only if fill is specified).
lty, lwd	the line types and widths for lines appearing in the legend. One of these two *must* be specified for line drawing.
pch	the plotting symbols appearing in the legend, as numeric vector or a vector of 1-character strings (see points). Unlike points, this can all be specified as a single multi-character string. *Must* be specified for symbol drawing.
angle	angle of shading lines.
density	the density of shading lines, if numeric and positive. If NULL or negative or NA color filling is assumed.
bty	the type of box to be drawn around the legend. The allowed values are "o" (the default) and "n".
bg	the background color for the legend box. (Note that this is only used if bty != "n".)
box.lty, box.lwd, box.col	the line type, width and color for the legend box (if bty = "o").
pt.bg	the background color for the points, corresponding to its argument bg.
cex	character expansion factor **relative** to current par("cex"). Used for text, and provides the default for pt.cex.
pt.cex	expansion factor(s) for the points.
pt.lwd	line width for the points, defaults to the one for lines, or if that is not set, to par("lwd").
xjust	how the legend is to be justified relative to the legend x location. A value of 0 means left justified, 0.5 means centered and 1 means right justified.
yjust	the same as xjust for the legend y location.
x.intersp	character interspacing factor for horizontal (x) spacing between symbol and legend text.

y.intersp	vertical (y) distances (in lines of text shared above/below each legend entry). A vector with one element for each row of the legend can be used.
adj	numeric of length 1 or 2; the string adjustment for legend text. Useful for y-adjustment when labels are plotmath expressions.
text.width	the width of the legend text in x ("user") coordinates. (Should be positive even for a reversed x axis.) Can be a single positive numeric value (same width for each column of the legend), a vector (one element for each column of the legend), NULL (default) for computing a proper maximum value of strwidth(legend)), or NA for computing a proper column wise maximum value of strwidth(legend)).
text.col	the color used for the legend text.
text.font	the font used for the legend text, see text.
merge	logical; if TRUE, merge points and lines but not filled boxes. Defaults to TRUE if there are points and lines.
trace	logical; if TRUE, shows how legend does all its magical computations.
plot	logical. If FALSE, nothing is plotted but the sizes are returned.
ncol	the number of columns in which to set the legend items (default is 1, a vertical legend).
horiz	logical; if TRUE, set the legend horizontally rather than vertically (specifying horiz overrides the ncol specification).
title	a character string or length-one expression giving a title to be placed at the top of the legend. Other objects will be coerced by as.graphicsAnnot.
inset	inset distance(s) from the margins as a fraction of the plot region when legend is placed by keyword.
xpd	if supplied, a value of the graphical parameter xpd to be used while the legend is being drawn.
title.col	color for title, defaults to text.col[1].
title.adj	horizontal adjustment for title: see the help for par("adj").
title.cex	expansion factor(s) for the title, defaults to cex[1].
title.font	the font used for the legend title, defaults to text.font[1], see text.
seg.len	the length of lines drawn to illustrate lty and/or lwd (in units of character widths).
curve_options	a list whose names are curve graphical parameters, and whose values are the corresponding graphical parameters values.

Details

This function is a wrapper for the legend function. It just adds two features:

- legend has a default value, regarding that HazardShape objects produces the TTT plot and its LOESS estimation.
- curve_options is used to set legend parameters for the LOESS curve. We encourage you to define first a list with curve parameters, and then pass it to plot.HazardShape and legend.HazardShape (see example 2).

Author(s)

Jaime Mosquera Gutiérrez <jmosquerag@unal.edu.co>

Examples

```
library(EstimationTools)

data("reduction_cells")
TTT_IG <- TTTE_Analytical(Surv(days, status) ~ 1, data = reduction_cells,
                          method = 'censored')
HS_IG <- TTT_hazard_shape(TTT_IG, data = reduction_cells)

#-------------------------------------------------------------------------------
# Example 1: put legend to the TTT plot of the reduction cells data set.

# Run `plot.HazardShape` method.
par(
  cex.lab=1.8,
  cex.axis=1.8,
  mar=c(4.8,5.4,1,1),
  las = 1,
  mgp=c(3.4,1,0)
)
plot(HS_IG, pch = 16, cex = 1.8)

# Finally, put the legend
legend.HazardShape(
  x = "bottomright",
  box.lwd = NA,
  cex = 1.8,
  pt.cex = 1.8,
  bty = 'n',
  pch = c(16, NA)
)
#-------------------------------------------------------------------------------
# Example 2: example 1 with custom options for the curve

# This is the list with curve parameters
loess_options <- list(
  col = 3, lwd = 4, lty = 2
)
par(
  cex.lab=1.8,
  cex.axis=1.8,
  mar=c(4.8,5.4,1,1),
  las = 1,
  mgp=c(3.4,1,0)
)

plot(HS_IG, pch = 16, cex = 1.8, curve_options = loess_options)
legend.HazardShape(
  x = "bottomright",
```

```
  box.lwd = NA,
  cex = 1.8,
  pt.cex = 1.8,
  bty = 'n',
  pch = c(16, NA),
  curve_options = loess_options
)
```

`loess.options` *Configure various aspects of LOESS in* `TTT_hazard_shape`

Description

[Maturing]

This function allows the user to set the parameters of `loess` function used inside `TTT_hazard_shape`.

Usage

```
loess.options(span = 2/3, ...)
```

Arguments

`span`	the parameter which controls the degree of smoothing.
`...`	further arguments passed to `loess` function.

Details

Please, visit `loess` to know further possible arguments. The following arguments are not available for passing to the LOESS estimation:

data The only data handled inside `TTT_hazard_shape` is the computed empirical TTT.

subset This argument is used in `loess` to take a subset of data. In this context, it is not necessary.

Author(s)

Jaime Mosquera Gutiérrez <jmosquerag@unal.edu.co>

See Also

`loess`, `TTT_hazard_shape`

logit_link *Logit link function (for estimation with* maxlogL *object)*

Description

[Experimental]

log_link object provides a way to implement logit link function that maxlogL needs to perform estimation. See documentation for maxlogL for further information on parameter estimation and implementation of link objects.

Usage

```
logit_link()
```

Details

logit_link is part of a family of generic functions with no input arguments that defines and returns a list with details of the link function:

1. name: a character string with the name of the link function.
2. g: implementation of the link function as a generic function in R.
3. g_inv: implementation of the inverse link function as a generic function in R.

There is a way to add new mapping functions. The user must specify the details aforesaid.

Value

A list with logit link function, its inverse and its name.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

See Also

maxlogL

Other link functions: NegInv_link(), log_link()

Examples

```
#-------------------------------------------------------------------------------
# Estimation of proportion in binomial distribution with 'logit' function
# 10 trials, probability of success equals to 30%)
N <- rbinom(n = 100, size = 10, prob = 0.3)
phat <- maxlogL(x = N, dist = 'dbinom', fixed = list(size=10),
                link = list(over = "prob", fun = "logit_link"))
summary(phat)
```

```
# Link function name
fun <- logit_link()$name
print(fun)

# Link function
g <- logit_link()$g
curve(g(x), from = 0, to = 1)

# Inverse link function
ginv <- logit_link()$g_inv
curve(ginv(x), from = -10, to = 10)

#-------------------------------------------------------------------------------
```

---

log_link *Logarithmic link function (for estimation with* maxlogL *object)*

Description

log_link object provides a way to implement logarithmic link function that maxlogL needs to perform estimation. See documentation for maxlogL for further information on parameter estimation and implementation of link objects.

Usage

```
log_link()
```

Details

log_link is part of a family of generic functions with no input arguments that defines and returns a list with details of the link function:

1. name: a character string with the name of the link function.
2. g: implementation of the link function as a generic function in R.
3. g_inv: implementation of the inverse link function as a generic function in R.

There is a way to add new mapping functions. The user must specify the details aforesaid.

Value

A list with logit link function, its inverse and its name.

See Also

maxlogL

Other link functions: NegInv_link(), logit_link()

Examples

```
# One parameters of normal distribution mapped with logarithmic function
x <- rnorm(n = 10000, mean = 50, sd = 4)
theta_2 <- maxlogL( x = x, link = list(over = "sd",
                                       fun = "log_link") )
summary(theta_2)

# Link function name
fun <- log_link()$name
print(fun)

# Link function
g <- log_link()$g
curve(g(x), from = 0, to = 1)

# Inverse link function
ginv <- log_link()$g_inv
curve(ginv(x), from = -5, to = 5)
```

---

maxlogL *Maximum Likelihood Estimation for parametric distributions*

Description

[Maturing]

Wrapper function to compute maximum likelihood estimators (MLE) of any distribution implemented in R.

Usage

```
maxlogL(
  x,
  dist = "dnorm",
  fixed = NULL,
  link = NULL,
  start = NULL,
  lower = NULL,
  upper = NULL,
  optimizer = "nlminb",
  control = NULL,
  StdE_method = c("optim", "numDeriv"),
  silent = FALSE,
  ...
)
```

Arguments

x	a vector with data to be fitted. This argument must be a matrix with hierarchical distributions.
dist	a length-one character vector with the name of density/mass function of interest. The default value is 'dnorm', to compute maximum likelihood estimators of normal distribution.
fixed	a list with fixed/known parameters of distribution of interest. Fixed parameters must be passed with its name.
link	a list with names of parameters to be linked, and names of the link function object. For names of parameters, please visit documentation of density/mass function. There are three link functions available: log_link, logit_link and NegInv_link.
start	a numeric vector with initial values for the parameters to be estimated.
lower	a numeric vector with lower bounds, with the same length of argument start (for box-constrained optimization).
upper	a numeric vector with upper bounds, with the same length of argument start (for box-constrained optimization).
optimizer	a length-one character vector with the name of optimization routine. nlminb, optim, DEoptim and ga are available; custom optimization routines can also be implemented. nlminb is the default routine.
control	control parameters of the optimization routine. See, e.g., optim's control list, nlminb's control list, and DEoptim.control for DEoptim.
StdE_method	a length-one character vector with the routine for Hessian matrix computation. The This is needed for standard error estimation. The options available are "optim" and "numDeriv". For further information, visit optim or hessian.
silent	logical. If TRUE, warnings of maxlogL are suppressed.
...	further arguments to be supplied to the optimizer.

Details

maxlogL computes the likelihood function corresponding to the distribution specified in argument dist and maximizes it through optim, nlminb or DEoptim. maxlogL generates an S3 object of class maxlogL.

Noncentrality parameters must be named as ncp in the distribution.

Value

A list with class "maxlogL" containing the following lists:

fit	A list with output information about estimation.
inputs	A list with all input arguments.
outputs	A list with some output additional information:

- Number of parameters.
- Sample size
- Standard error computation method.

Note

The following generic functions can be used with a maxlogL object: summary, print, AIC, BIC, logLik.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

References

Nelder JA, Mead R (1965). "A Simplex Method for Function Minimization." *The Computer Journal*, **7**(4), 308–313. ISSN 0010-4620, doi:10.1093/comjnl/7.4.308, https://academic.oup.com/comjnl/article-lookup/doi/10.1093/comjnl/7.4.308.

Fox PA, Hall AP, Schryer NL (1978). "The PORT Mathematical Subroutine Library." *ACM Transactions on Mathematical Software*, **4**(2), 104–126. ISSN 00983500, doi:10.1145/355780.355783, https://dl.acm.org/doi/10.1145/355780.355783.

Nash JC (1979). *Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation*, 2nd Edition edition. Adam Hilger, Bristol.

Dennis JE, Gay DM, Walsh RE (1981). "An Adaptive Nonlinear Least-Squares Algorithm." *ACM Transactions on Mathematical Software*, **7**(3), 348–368. ISSN 00983500, doi:10.1145/355958.355965, https://dl.acm.org/doi/10.1145/355958.355965.

See Also

summary.maxlogL, optim, nlminb, DEoptim, DEoptim.control, maxlogLreg, bootstrap_maxlogL

Other maxlogL: cum_hazard.maxlogL(), expected_value.maxlogL(), maxlogLreg()

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------
# Example 1: estimation with one fixed parameter
x <- rnorm(n = 10000, mean = 160, sd = 6)
theta_1 <- maxlogL(x = x, dist = 'dnorm', control = list(trace = 1),
                link = list(over = "sd", fun = "log_link"),
                fixed = list(mean = 160))
summary(theta_1)


#---------------------------------------------------------------------------
# Example 2: both parameters of normal distribution mapped with logarithmic
# function
theta_2 <- maxlogL(x = x, dist = "dnorm",
                  link = list(over = c("mean","sd"),
                              fun = c("log_link","log_link")))
summary(theta_2)
```

```
#-------------------------------------------------------------------------------
# Example 3: parameter estimation in ZIP distribution
if (!require('gamlss.dist')) install.packages('gamlss.dist')
library(gamlss.dist)
z <- rZIP(n=1000, mu=6, sigma=0.08)
theta_3  <- maxlogL(x = z, dist = 'dZIP', start = c(0, 0),
                    lower = c(-Inf, -Inf), upper = c(Inf, Inf),
                    optimizer = 'optim',
                    link = list(over=c("mu", "sigma"),
                    fun = c("log_link", "logit_link")))
summary(theta_3)


#-------------------------------------------------------------------------------
# Example 4: parameter estimation with fixed noncentrality parameter.
y_2 <- rbeta(n = 1000, shape1 = 2, shape2 = 3)
theta_41 <- maxlogL(x = y_2, dist = "dbeta",
                     link = list(over = c("shape1", "shape2"),
                     fun = c("log_link","log_link")))
summary(theta_41)

# It is also possible define 'ncp' as fixed parameter
theta_42 <- maxlogL(x = y_2, dist = "dbeta", fixed = list(ncp = 0),
                     link = list(over = c("shape1", "shape2"),
                     fun = c("log_link","log_link")) )
summary(theta_42)


#-------------------------------------------------------------------------------
```

`maxlogLreg` *Maximum Likelihood Estimation for parametric linear regression models*

Description

[Experimental]

Function to compute maximum likelihood estimators (MLE) of regression parameters of any distribution implemented in R with covariates (linear predictors).

Usage

```
maxlogLreg(
  formulas,
  y_dist,
  support = NULL,
  data = NULL,
  subset = NULL,
```

```
  fixed = NULL,
  link = NULL,
  optimizer = "nlminb",
  start = NULL,
  lower = NULL,
  upper = NULL,
  inequalities = NULL,
  control = NULL,
  silent = FALSE,
  StdE_method = c("optim", "numDeriv"),
  ...
)
```

Arguments

formulas	a list of formula objects. Each element must have an ~, with the terms on the right separated by + operators. The response variable on the left side is optional. Linear predictor of each parameter must be specified with the name of the parameter followed by the suffix '.fo'. See the examples below for further illustration.
y_dist	a formula object that specifies the distribution of the response variable. On the left side of ~ must be the response, and in the right side must be the name o de probability density/mass function. See the section **Details** and the examples below for further illustration.
support	a list with the following entries: • interval: a two dimensional atomic vector indicating the set of possible values of a random variable having the distribution specified in y_dist. • type: character indicating if distribution has a discrete or a continous random variable.
data	an optional data frame containing the variables in the model. If data is not specified, the variables are taken from the environment from which maxlogLreg is called.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
fixed	a list with fixed/known parameters of distribution of interest. Fixed parameters must be passed with its name and its value (known).
link	a list with names of parameters to be linked, and names of the link function object. For names of parameters, please visit documentation of density/mass function. There are three link functions available: log_link, logit_link and NegInv_link. Take into account: the order used in argument over corresponds to the order in argument link.
optimizer	a length-one character vector with the name of optimization routine. nlminb, optim and DEoptim are available; nlminb is the default routine.
start	a numeric vector with initial values for the parameters to be estimated. Zero is the default value.

`lower`	a numeric vector with lower bounds, with the same lenght of argument `start` (for box-constrained optimization). `-Inf` is the default value.
`upper`	a numeric vector with upper bounds, with the same lenght of argument `start` (for box-constrained optimization). `Inf` is the default value.
`inequalities`	a character vector with the inequality constrains for the distribution parameters.
`control`	control parameters of the optimization routine. Please, visit documentation of selected optimizer for further information.
`silent`	logical. If TRUE, warnings of `maxlogL` are suppressed.
`StdE_method`	a length-one character vector with the routine for Hessian matrix computation. The This is needed for standard error estimation. The options available are `"optim"` and `"numDeriv"`. For further information, visit `optim` or `hessian`.
`...`	Further arguments to be supplied to the optimization routine.

Details

`maxlogLreg` computes programmatically the log-likelihood (log L) function corresponding for the following model:

$$y_i \overset{iid.}{\sim} \mathcal{D}(\theta_{i1}, \theta_{i2}, \ldots, \theta_{ij}, \ldots, \theta_{ik})$$
$$g(\boldsymbol{\theta}_j) = \boldsymbol{\eta}_j = \mathbf{X}_j^\top \boldsymbol{\beta}_j,$$

where,

- $g_k(\cdot)$ is the k-th link function.
- $\boldsymbol{\eta}_j$ is the value of the linear predictor for the j^{th} for all the observations.
- $\boldsymbol{\beta}_j = (\beta_{0j}, \beta_{1j}, \ldots, \beta_{(p_j-1)j})^\top$ are the fixed effects vector, where p_j is the number of parameters in linear predictor j and $\mathbf{X}_j$ is a known design matrix of order $n \times p_j$. These terms are specified in `formulas` argument.
- $\mathcal{D}$ is the distribution specified in argument `y_dist`.

Then, `maxlogLreg` maximizes the log L through `optim`, `nlminb` or `DEoptim`. `maxlogLreg` generates an S3 obj.

Estimation with censorship can be handled with `Surv` objects (see example 2). The output object stores the corresponding censorship matrix, defined as $r_{il} = 1$ if sample unit i has status l, or $r_{il} = 0$ in other case. $i = 1, 2, \ldots, n$ and $l = 1, 2, 3$ ($l = 1$: observation status, $l = 2$: right censorship status, $l = 3$: left censorship status).

Value

A list with class `maxlogL` containing the following lists:

`fit`	A list with output information about estimation and method used.
`inputs`	A list with all input arguments.
`outputs`	A list with additional information. The most important outputs are: • `npar`: number of parameters.

- n: sample size
- Stde_method: standard error computation method.
- b_lenght: a list with the number of regression parameters.
- design_matrix: a list with the $\mathbf{X}$ matrix for each parameter, the response values (called y) and the censorship matrix (called status). See the Details section for further information.

Note

- The following generic functions can be used with a maxlogL object: summary, print, logLik, AIC.
- Noncentrality parameters must be named as ncp in the distribution.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

References

Nelder JA, Mead R (1965). "A Simplex Method for Function Minimization." *The Computer Journal*, **7**(4), 308–313. ISSN 0010-4620, doi:10.1093/comjnl/7.4.308, https://academic.oup.com/comjnl/article-lookup/doi/10.1093/comjnl/7.4.308.

Fox PA, Hall AP, Schryer NL (1978). "The PORT Mathematical Subroutine Library." *ACM Transactions on Mathematical Software*, **4**(2), 104–126. ISSN 00983500, doi:10.1145/355780.355783, https://dl.acm.org/doi/10.1145/355780.355783.

Nash JC (1979). *Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation*, 2nd Edition edition. Adam Hilger, Bristol.

Dennis JE, Gay DM, Walsh RE (1981). "An Adaptive Nonlinear Least-Squares Algorithm." *ACM Transactions on Mathematical Software*, **7**(3), 348–368. ISSN 00983500, doi:10.1145/355958.355965, https://dl.acm.org/doi/10.1145/355958.355965.

See Also

summary.maxlogL, optim, nlminb, DEoptim, DEoptim.control, maxlogL, bootstrap_maxlogL

Other maxlogL: cum_hazard.maxlogL(), expected_value.maxlogL(), maxlogL()

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------
# Example 1: Estimation in simulated normal distribution
n <- 1000
x <- runif(n = n, -5, 6)
y <- rnorm(n = n, mean = -2 + 3 * x, sd = exp(1 + 0.3* x))
norm_data <- data.frame(y = y, x = x)

# It does not matter the order of distribution parameters
```

```
formulas <- list(sd.fo = ~ x, mean.fo = ~ x)
support <- list(interval = c(-Inf, Inf), type = 'continuous')

norm_mod <- maxlogLreg(formulas, y_dist = y ~ dnorm, support = support,
                       data = norm_data,
                       link = list(over = "sd", fun = "log_link"))
summary(norm_mod)


#------------------------------------------------------------------------------
# Example 2: Fitting with censorship
# (data from https://www.itl.nist.gov/div898/handbook/apr/section4/apr413.htm)

failures <- c(55, 187, 216, 240, 244, 335, 361, 373, 375, 386)
fails <- c(failures, rep(500, 10))
status <- c(rep(1, length(failures)), rep(0, 10))
Wei_data <- data.frame(fails = fails, status = status)

# Formulas with linear predictors
formulas <- list(scale.fo=~1, shape.fo=~1)
support <- list(interval = c(0, Inf), type = 'continuous')

# Bounds for optimization. Upper bound set with default values (Inf)
start <- list(
  scale = list(Intercept = 100),
  shape = list(Intercept = 10)
)
lower <- list(
  scale = list(Intercept = 0),
  shape = list(Intercept = 0)
)

mod_weibull <- maxlogLreg(formulas, y_dist = Surv(fails, status) ~ dweibull,
                          support = c(0, Inf), start = start,
                          lower = lower, data = Wei_data)
summary(mod_weibull)


#------------------------------------------------------------------------------
```

NegInv_link *Negative inverse link function (for estimation with* maxlogL *object)*

Description

NegInv_link object provides a way to implement negative inverse link function that maxlogL needs to perform estimation. See documentation for maxlogL for further information on parameter estimation and implementation of link objects.

Usage

```
NegInv_link()
```

Details

NegInv_link is part of a family of generic functions with no input arguments that defines and returns a list with details of the link function:

1. name: a character string with the name of the link function.
2. g: implementation of the link function as a generic function in R.
3. g_inv: implementation of the inverse link function as a generic function in R.

There is a way to add new mapping functions. The user must specify the details aforesaid.

Value

A list with negative inverse link function, its inverse and its name.

See Also

maxlogL

Other link functions: log_link(), logit_link()

Examples

```
# Estimation of rate parameter in exponential distribution
T <- rexp(n = 1000, rate = 3)
lambda <- maxlogL(x = T, dist = "dexp", start = 5,
                  link = list(over = "rate", fun = "NegInv_link"))
summary(lambda)

# Link function name
fun <- NegInv_link()$name
print(fun)

# Link function
g <- NegInv_link()$g
curve(g(x), from = 0.1, to = 1)

# Inverse link function
ginv <- NegInv_link()$g_inv
curve(ginv(x), from = 0.1, to = 1)
```

`plot.EmpiricalTTT` *Plot method for* `EmpiricalTTT` *objects*

Description

[Experimental]

Draws a TTT plot of an `EmpiricalTTT` object, one for each strata.

TTT plots are graphed in the same order in which they appear in the list element `strata` or in the list element `phi_n` of the `EmpiricalTTT` object.

Usage

```
## S3 method for class 'EmpiricalTTT'
plot(
  x,
  add = FALSE,
  grid = FALSE,
  type = "l",
  pch = 1,
  xlab = "i/n",
  ylab = expression(phi[n](i/n)),
  ...
)
```

Arguments

`x`	an object of class `EmpiricalTTT`.
`add`	logical. If TRUE, `plot.EmpiricalTTT` add a TTT plot to an already existing plot.
`grid`	logical. If `TRUE`, plot appears with grid.
`type`	character string (length 1 vector) or vector of 1-character strings indicating the type of plot for each TTT graph. See `plot`.
`pch`	numeric (integer). A vector of plotting characters or symbols when `type = "p"`. See `points`.
`xlab, ylab`	titles for x and y axes, as in `plot`.
`...`	further arguments passed to `matplot`. See the examples and **Details** section for further information.

Details

This method is based on `matplot`. Our function sets some default values for graphic parameters: `type = "l"`, `pch = 1`, `xlab = "i/n"` and `ylab = expression(phi[n](i/n))`. This arguments can be modified by the user.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

See Also

TTTE_Analytical, matplot

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------------
# First example: Scaled empirical TTT from 'mgus1' data from 'survival' package.

TTT_1 <- TTTE_Analytical(Surv(stop, event == 'pcm') ~1, method = 'cens',
                         data = mgus1, subset=(start == 0))
plot(TTT_1, type = "p")


#---------------------------------------------------------------------------------
# Second example: Scaled empirical TTT using a factor variable with 'aml' data
# from 'survival' package.

TTT_2 <- TTTE_Analytical(Surv(time, status) ~ x, method = "cens", data = aml)
plot(TTT_2, type = "l", lty = c(1,1), col = c(2,4))
plot(TTT_2, add = TRUE, type = "p", lty = c(1,1), col = c(2,4), pch = 16)

#---------------------------------------------------------------------------------
# Third example: Non-scaled empirical TTT without a factor (arbitrarily simulated
# data).

y <- rweibull(n=20, shape=1, scale=pi)
TTT_3 <- TTTE_Analytical(y ~ 1, scaled = FALSE)
plot(TTT_3, type = "s", col = 3, lwd = 3)


#---------------------------------------------------------------------------------
# Fourth example: TTT plot for 'carbone' data from 'AdequacyModel' package

if (!require('AdequacyModel')) install.packages('AdequacyModel')
library(AdequacyModel)
data(carbone)
TTT_4 <- TTTE_Analytical(response = carbone, scaled = TRUE)
plot(TTT_4, type = "l", col = "red", lwd = 2, grid = TRUE)


#---------------------------------------------------------------------------------
```

`plot.HazardShape` *Plot of* `HazardShape` *objects*

Description

[Maturing]

Draws the empirical total time on test (TTT) plot and its non-parametric (LOESS) estimated curve useful for identifying hazard shape.

Usage

```
## S3 method for class 'HazardShape'
plot(
  x,
  xlab = "i/n",
  ylab = expression(phi[n](i/n)),
  xlim = c(0, 1),
  ylim = c(0, 1),
  col = 1,
  lty = NULL,
  lwd = NA,
  main = "",
  curve_options = list(col = 2, lwd = 2, lty = 1),
  par_plot = lifecycle::deprecated(),
  legend_options = lifecycle::deprecated(),
  ...
)
```

Arguments

`x`	an object of class `initValOW`, generated with `TTT_hazard_shape`.
`xlab, ylab`	titles for x and y axes, as in `plot`.
`xlim`	the x limits (x1, x2) of the plot.
`ylim`	the y limits (x1, x2) of the plot.
`col`	the colors for lines and points. Multiple colors can be specified. This is the usual color argument of `plot.default`.
`lty`	a vector of line types, see `par` for further information.
`lwd`	a vector of line widths, see `par` for further information.
`main`	a main title for the plot.
`curve_options`	a list with further arguments useful for customization of non-parametric estimate plot.
`par_plot`	(deprecated) some graphical parameters which can be passed to the plot. See **Details** section for further information.

legend_options	(deprecated) a list with fur further arguments useful for customization. See **Details** section for further information. of the legend of the plot.
...	further arguments passed to empirical TTT plot.

Details

This plot complements the use of TTT_hazard_shape. It is always advisable to use this function in order to check the result of non-parametric estimate of TTT plot. See the first example in **Examples** section for an illustration.

Author(s)

Jaime Mosquera Gutiérrez <jmosquerag@unal.edu.co>

Examples

```
library(EstimationTools)

#-------------------------------------------------------------------------
# Example 1: Increasing hazard and its corresponding TTT plot with simulated
# data
hweibull <- function(x, shape, scale) {
  dweibull(x, shape, scale) / pweibull(x, shape, scale, lower.tail = FALSE)
}
curve(hweibull(x, shape = 2.5, scale = pi),
  from = 0, to = 42,
  col = "red", ylab = "Hazard function", las = 1, lwd = 2
)

y <- rweibull(n = 50, shape = 2.5, scale = pi)
my_initial_guess <- TTT_hazard_shape(formula = y ~ 1)

par(mar = c(3.7, 3.7, 1, 2), mgp = c(2.5, 1, 0))
plot(my_initial_guess)

#-------------------------------------------------------------------------
```

---

plot.maxlogL *Plot Residual Diagnostics for an* maxlogL *Object*

Description

[Experimental]

Provides plots of Cox-Snell, martingale Randomized quantile residuals.

Usage

```
## S3 method for class 'maxlogL'
plot(
  x,
  type = c("rqres", "cox-snell", "martingale", "right-censored-deviance"),
  parameter = NULL,
  which.plots = NULL,
  caption = NULL,
  xvar = NULL,
  ...
)
```

Arguments

x	a maxlogL object.
type	a character with the type of residuals to be plotted. The default value is type = "rqres", which is used to compute the normalized randomized quantile residuals.
parameter	which parameter fitted values are required for type = "rqres". The default is the first one defined in the pdf,e.g, in dnorm, the default parameter is mean.
which.plots	a subset of numbers to specify the plots. See details for further information.
caption	title of the plots. If caption = NULL, the default values are used.
xvar	an explanatory variable to plot the residuals against.
...	further parameters for the plot method.

Details

If type = "rqres", the available subset is 1:4, referring to:

- 1. Quantile residuals vs. fitted values (Tukey-Ascomb plot)
- 2. Quantile residuals vs. index
- 3. Density plot of quantile residuals
- 4. Normal Q-Q plot of the quantile residuals.

Value

Returns specified plots related to the residuals of the fitted maxlogL model.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------
# Example 1: Quantile residuals for a normal model
n <- 1000
x <- runif(n = n, -5, 6)
y <- rnorm(n = n, mean = -2 + 3 * x, sd = exp(1 + 0.3* x))
norm_data <- data.frame(y = y, x = x)

# It does not matter the order of distribution parameters
formulas <- list(sd.fo = ~ x, mean.fo = ~ x)
support <- list(interval = c(-Inf, Inf), type = 'continuous')

norm_mod <- maxlogLreg(formulas, y_dist = y ~ dnorm, support = support,
                       data = norm_data,
                       link = list(over = "sd", fun = "log_link"))

# Quantile residuals diagnostic plot
plot(norm_mod, type = "rqres")
plot(norm_mod, type = "rqres", parameter = "sd")

# Exclude Q-Q plot
plot(norm_mod, type = "rqres", which.plots = 1:3)


#---------------------------------------------------------------------------
# Example 2: Cox-Snell residuals for an exponential model
data(ALL_colosimo_table_4_1)
formulas <- list(scale.fo = ~ lwbc)
support <- list(interval = c(0, Inf), type = 'continuous')

ALL_exp_model <- maxlogLreg(
  formulas,
  fixed = list(shape = 1),
  y_dist = Surv(times, status) ~ dweibull,
  data = ALL_colosimo_table_4_1,
  support = support,
  link = list(over = "scale", fun = "log_link")
)

summary(ALL_exp_model)
plot(ALL_exp_model, type = "cox-snell")
plot(ALL_exp_model, type = "right-censored-deviance")

plot(ALL_exp_model, type = "martingale", xvar = NULL)
plot(ALL_exp_model, type = "martingale", xvar = "lwbc")


#---------------------------------------------------------------------------
```

predict.maxlogL *Predict Method for* maxlogL *Fits*

Description

[Maturing]

This function computes predictions and optionally the estimated standard errors of those predictions from a model fitted with maxlogLreg.

Usage

```
## S3 method for class 'maxlogL'
predict(
  object,
  parameter = NULL,
  newdata = NULL,
  type = c("link", "response", "terms"),
  se.fit = FALSE,
  terms = NULL,
  ...
)
```

Arguments

object	an object of maxlogL class generated by maxlogLreg function.
parameter	a character which specifies the parameter to predict.
newdata	a data frame with covariates with which to predict. It is an optional argument, if omitted, the fitted linear predictors or the (distribution) parameter predictions are used.
type	a character with the type of prediction required. The default (type = "link") is on the scale of the linear predictors; the alternative type = "response" is on the scale of the distribution parameter.
se.fit	logical switch indicating if standard errors of predictions are required.
terms	A character vector that specifies which terms are required if type = "terms". All terms are returned by default.
...	further arguments passed to or from other methods.

Details

This predict method computes predictions for values of any distribution parameter in link or original scale.

Value

If `se.fit = FALSE`, a vector of predictions is returned. For `type = "terms"`, a matrix with a column per term and an attribute "constant" is returned.

If `se.fit = TRUE`, a list with the following components is obtained:

1. `fit`: Predictions.
2. `se.fit`: Estimated standard errors.

Note

Variables are first looked for in `newdata` argument and then searched in the usual way (which will include the environment of the formula used in the fit). A warning will be given if the variables found are not of the same length as those in `newdata` if it is supplied.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------------
# Example 1: Predictions from a model using a simulated normal distribution
n <- 1000
x <- runif(n = n, -5, 6)
y <- rnorm(n = n, mean = -2 + 3 * x, sd = exp(1 + 0.3* x))
norm_data <- data.frame(y = y, x = x)

# It does not matter the order of distribution parameters
formulas <- list(sd.fo = ~ x, mean.fo = ~ x)

norm_mod <- maxlogLreg(formulas, y_dist = y ~ dnorm, data = norm_data,
                       link = list(over = "sd", fun = "log_link"))
predict(norm_mod)


#---------------------------------------------------------------------------------
# Example 2: Predictions using new values for covariates
predict(norm_mod, newdata = data.frame(x=0:6))


#---------------------------------------------------------------------------------
# Example 3: Predictions for another parameter
predict(norm_mod, newdata = data.frame(x=0:6), param = "sd",
       type = "response")

#---------------------------------------------------------------------------------
# Example 4: Model terms
predict(norm_mod, param = "sd", type = "terms")
```

```
#-------------------------------------------------------------------------
```

`print.HazardShape` *Print method for* `HazardShape` *objects*

Description

[Experimental]

Displays the estimated hazard shape given a `HazardShape` object.

Usage

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

Arguments

`x`	an object of class `HazardShape`, generated with `TTT_hazard_shape`.
`...`	further arguments passed to or from other methods.

Author(s)

Jaime Mosquera Gutiérrez <jmosquerag@unal.edu.co>

Examples

```
#-------------------------------------------------------------------------
# Example 1: Increasing hazard and its corresponding TTT plot with simulated data

hweibull <- function(x, shape, scale){
  dweibull(x, shape, scale)/pweibull(x, shape, scale, lower.tail = FALSE)
  }
curve(hweibull(x, shape = 2.5, scale = pi), from = 0, to = 42,
               col = "red", ylab = "Hazard function", las = 1, lwd = 2)

y <- rweibull(n = 50, shape = 2.5, scale = pi)
my_initial_guess <- TTT_hazard_shape(formula = y ~ 1)
print(my_initial_guess)


#-------------------------------------------------------------------------
```

reduction_cells *Reduction cells*

Description

Times to failure (in units of 1000 days) of 20 aluminum reduction cells.

Usage

```
reduction_cells
```

Format

A data frame with 20 observations.

References

Whitmore GA (1983). "A regression method for censored inverse-Gaussian data." *Canadian Journal of Statistics*, **11**(4), 305–315. ISSN 03195724, doi:10.2307/3314888.

Examples

```
data(reduction_cells)
par(mfrow = c(1,2))
hist(reduction_cells$days, main="", xlab="Time (Days)")
plot(reduction_cells$days, xlab = "Cell (subjects)", lty = 3, type="h")
```

residuals.maxlogL *Extract Residuals from* maxlogL *model.*

Description

[Experimental]

`residuals.maxlogL` is the `maxlogLreg` specific method for the generic function residuals which extracts the residuals from a fitted model.

Usage

```
## S3 method for class 'maxlogL'
residuals(object, parameter = NULL, type = "rqres", routine, ...)
```

Arguments

object	an object of maxlogL class obtained by fitting a model with maxlogLreg.
parameter	a character which specifies residuals for a specific parameter.
type	a character with the type of residuals to be computed. The default value is type = "rqres", which is used to compute the normalized randomized quantile residuals.
routine	a character specifying the integration routine. integrate and gauss_quad are available. Custom routines can be defined but they must be compatible with the integration API.
...	further arguments for the integration routine.

Details

For type = "deviance", the residuals are computed using the following expression

$$r_i^D = \text{sign}(y_i - \hat{\mu}_i) d_i^{1/2},$$

where d_i is the residual deviance of each data point. In this context, $\hat{\mu}$ is the estimated mean, computed as the expected value using the estimated parameters of a fitted maxlogLreg model.

On the other hand, for type = "response" the computation is simpler

$$r_i^R = (y_i - \hat{\mu}_i).$$

Value

a vector with the specified residuals of a maxlogLreg model.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------
# Example 1: Test deviance residuals
set.seed(123)
n <- 500
x <- runif(n = n, min = 0, max = 1)
y <- rweibull(n = n, shape = 1, scale = exp(4.5 + 0.5*x))
status <- rep(1, n) # sample(0:1, size = n, replace = TRUE)

distribution <- Surv(y, status) ~ dweibull

formulas <- list(
  scale.fo = ~ x
)
```

```
fixed <- list(shape = 1)

links <- list(
  over = "scale",
  fun = "log_link"
)

model <- maxlogLreg(
  formulas = formulas,
  y_dist = distribution,
  fixed = fixed,
  link = links
)

# Using `residuals` method
cox_snell_residuals_test <- residuals(model, type = "cox-snell")
martingale_residuals_test <- residuals(model, type = "martingale")
deviance_residuals_test <- residuals(model, type = "right-censored-deviance")

# From scratch
cox_snell_residuals_ref <- -pweibull(
  q = y,
  shape = 1,
  scale = exp(cbind(rep(1, n), x) %*% cbind(coef(model))),
  lower.tail = FALSE,
  log.p = TRUE
)
martingale_residuals_ref <- status - cox_snell_residuals_ref
deviance_residuals_ref <- sign(martingale_residuals_ref) * (
  -2 * (martingale_residuals_ref + status*log(status - martingale_residuals_ref))
)^ 0.5


plot(cox_snell_residuals_test, cox_snell_residuals_ref)
plot(martingale_residuals_test, martingale_residuals_ref)
plot(deviance_residuals_test, deviance_residuals_ref)


#------------------------------------------------------------------------
```

`set_optimizer` *Set custom optimizer in* `maxlogLreg`

Description

This function allows the selection of a custom S3 optimizer and additional arguments to be passed to it.

Usage

```
set_optimizer(
  optimizer_name,
  objective_name,
  start_name,
  lower_name,
  upper_name,
  optim_vals_name,
  objective_vals_name,
  ...
)
```

Arguments

optimizer_name	a one-length character specifying the name of the optimizer to be used.
objective_name	a one-length character with the name of the optimizer.
start_name	a one-length character with the name of the start/initial values input argument.
lower_name	a one-length character with the name of the lower bounds input argument.
upper_name	a one-length character with the name of the upper bounds input argument.
optim_vals_name	a one-length character with the name of the optimum values output argument.
objective_vals_name	a one-length character with the name of the objective function name output argument.
...	Further arguments to be passed to the selected optimizer. Do not include lower bounds, upper bounds and start/initial values. It must be included in the maxlogLreg function definition

Value

a list containing the selected optimizer name and the additional, the names of the start values argument, upper and lower bounds, optimum output values, objective function output value and the arguments passed to it.

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------
# Example 1: Estimation in simulated normal distribution (alternative
# implementation using `nlminb` as a custom optimizer)
n <- 1000
x <- runif(n = n, -5, 6)
y <- rnorm(n = n, mean = -2 + 3 * x, sd = exp(1 + 0.3* x))
norm_data <- data.frame(y = y, x = x)

formulas <- list(sd.fo = ~ x, mean.fo = ~ x)
support <- list(interval = c(-Inf, Inf), type = 'continuous')
```

```
# Default optimizer
norm_mod1 <- maxlogLreg(formulas, y_dist = y ~ dnorm, support = support,
                        data = norm_data,
                        link = list(over = "sd", fun = "log_link"))
summary(norm_mod1)

# Use the default optimizer as a custom one
optimizer <- set_optimizer(
  optimizer_name = "nlminb",
  objective_name = "objective",
  start_name = "start",
  lower_name = "lower",
  upper_name = "upper",
  optim_vals_name = "par",
  objective_vals_name = "objective"
)

norm_mod2 <- maxlogLreg(formulas, y_dist = y ~ dnorm, support = support,
                        data = norm_data,
                        link = list(over = "sd", fun = "log_link"),
                        optimizer = optimizer, start= NULL, lower = NULL,
                        upper = NULL)
summary(norm_mod2)
```

`summary.maxlogL` *Summarize Maximum Likelihood Estimation*

Description

[Maturing]

Displays maximum likelihood estimates computed with `maxlogL` with its standard errors, AIC and BIC. This is a `summary` method for `maxlogL` object.

Usage

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

Arguments

`object`	an object of `maxlogL` class which summary is desired.
`...`	additional arguments affecting the summary produced.

Details

This `summary` method computes and displays AIC, BIC, estimates and standard errors from a estimated model stored i a `maxlogL` class object. It also displays and computes Z-score and p values of significance test of parameters.

Value

A list with information that summarize results of a maxlogL class object.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

See Also

maxlogL, maxlogLreg, bootstrap_maxlogL

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------
### First example: One known parameter

x <- rnorm(n = 10000, mean = 160, sd = 6)
theta_1 <- maxlogL(x = x, dist = 'dnorm', control = list(trace = 1),
                   link = list(over = "sd", fun = "log_link"),
                   fixed = list(mean = 160))
summary(theta_1)


#---------------------------------------------------------------------------
# Second example: Binomial probability parameter estimation with variable
# creation

N <- rbinom(n = 100, size = 10, prob = 0.3)
phat <- maxlogL(x = N, dist = 'dbinom', fixed = list(size = 10),
                  link = list(over = "prob", fun = "logit_link"))

## Standard error calculation method
print(phat$outputs$StdE_Method)

## 'summary' method
summary(phat)

#---------------------------------------------------------------------------
# Third example: Binomial probability parameter estimation with no variable
# creation

N <- rbinom(n = 100, size = 10, prob = 0.3)
summary(maxlogL(x = N, dist = 'dbinom', fixed = list(size = 10),
                  link = list(over = "prob", fun = "logit_link")))

#---------------------------------------------------------------------------
# Fourth example: Estimation in a regression model with simulated normal data
n <- 1000
x <- runif(n = n, -5, 6)
y <- rnorm(n = n, mean = -2 + 3 * x, sd = exp(1 + 0.3* x))
```

```
norm_data <- data.frame(y = y, x = x)
formulas <- list(sd.fo = ~ x, mean.fo = ~ x)

norm_mod <- maxlogLreg(formulas, y_dist = y ~ dnorm, data = norm_data,
                       link = list(over = "sd", fun = "log_link"))

## 'summary' method
summary(norm_mod)


#------------------------------------------------------------------------------
```

---

summate *Summation of One-Dimensional Functions*

Description

[Experimental]

Discrete summation of functions of one variable over a finite or semi-infinite interval.

Usage

```
summate(fun, lower, upper, tol = 1e-10, ...)
```

Arguments

fun	an **R** function which should take a numeric argument x and possibly some parameters. The function returns a numerical vector value for the given argument x.
lower	a numeric value for the lower limit of the integral.
upper	a numeric value for the upper limit of the integral.
tol	a numeric value indicating the accuracy of the result (useful in infinite summations).
...	additional arguments to be passed to fun.

Details

Arguments after ... must be matched exactly. If both limits are infinite, the function fails. For semi-infinite intervals, the summation must be convergent. This is accomplished in manny probability mass functions.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

Examples

```
library(EstimationTools)


#-------------------------------------------------------------------------
# Example 1: Poisson expected value computation, X ~ Poisson(lambda = 15)
Poisson_integrand <- function(x, lambda) {
  x * lambda^x * exp(-lambda)/factorial(x)
}

summate(fun = Poisson_integrand, lower = 0, upper = Inf, lambda = 15)


#-------------------------------------------------------------------------
```

`TTTE_Analytical` *Empirical Total Time on Test (TTT), analytic version.*

Description

[Experimental]

This function allows to compute the TTT curve from a formula containing a factor type variable (classification variable).

Usage

```
TTTE_Analytical(
  formula,
  response = NULL,
  scaled = TRUE,
  data = NULL,
  method = c("Barlow", "censored"),
  partition_method = NULL,
  silent = FALSE,
  ...
)
```

Arguments

`formula`	an object of class `formula` with the response on the left of an operator `~`. The right side can be a factor variable as term or an `1` if a classification by factor levels is not desired.
`response`	an optional numeric vector with data of the response variable. Using this argument is equivalent to define a formula with the right side such as `~ 1`. See the fourth example below.
`scaled`	logical. If `TRUE` (default value), scaled TTT is computed.

`data`	an optional data frame containing the variables (response and the factor, if it is desired). If data is not specified, the variables are taken from the environment from which `TTT_analytical` is called.
`method`	a character specifying the method of computation. There are two options available: `'Barlow'` and `'censored'`. Further information can be found in the **Details** section.
`partition_method`	a list specifying cluster formation when the covariate in `formula` is numeric, or when the data has several covariates. 'quantile-based' method is the only one currently available (See the last example).
`silent`	logical. If TRUE, warnings of `TTTE_Analytical` are suppressed.
`...`	further arguments passing to `survfit`.

Details

When `method` argument is set as `'Barlow'`, this function uses the original expression of empirical TTT presented by Barlow (1979) and used by Aarset (1987):

$$\phi_n\left(\frac{r}{n}\right) = \frac{\left(\sum_{i=1}^{r} T_{(i)}\right) + (n-r)T_{(r)}}{\sum_{i=1}^{n} T_i}$$

where $T_{(r)}$ is the r^{th} order statistic, with $r = 1, 2, \ldots, n$, and n is the sample size. On the other hand, the option *'censored'* is an implementation based on integrals presented in Westberg and Klefsjö (1994), and using `survfit` to compute the Kaplan-Meier estimator:

$$\phi_n\left(\frac{r}{n}\right) = \sum_{j=1}^{r}\left[\prod_{i=1}^{j}\left(1 - \frac{d_i}{n_i}\right)\right]\left(T_{(j)} - T_{(j-1)}\right)$$

Value

A list with class object `Empirical.TTT` containing a list with the following information:

`i/n‘`	A matrix containing the empirical quantiles. This matrix has the number of columns equals to the number of levels of the factor considered (number of strata).
`phi_n`	A matrix containing the values of empirical TTT. his matrix has the number of columns equals to the number of levels of the factor considered (number of strata).
`strata`	A numeric named vector storing the number of observations per strata, and the name of each strata (names of the levels of the factor).

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

References

Barlow RE (1979). "Geometry of the total time on test transform." *Naval Research Logistics Quarterly*, **26**(3), 393–402. ISSN 00281441, doi:10.1002/nav.3800260303.

Aarset MV (1987). "How to Identify a Bathtub Hazard Rate." *IEEE Transactions on Reliability*, **R-36**(1), 106–108. ISSN 15581721, doi:10.1109/TR.1987.5222310.

Klefsjö B (1991). "TTT-plotting - a tool for both theoretical and practical problems." *Journal of Statistical Planning and Inference*, **29**(1-2), 99–110. ISSN 03783758, doi:10.1016/0378-3758(92)90125C, https://linkinghub.elsevier.com/retrieve/pii/037837589290125C.

Westberg U, Klefsjö B (1994). "TTT-plotting for censored data based on the piecewise exponential estimator." *International Journal of Reliability, Quality and Safety Engineering*, **01**(01), 1–13. ISSN 0218-5393, doi:10.1142/S0218539394000027, https://www.worldscientific.com/doi/abs/10.1142/S0218539394000027.

See Also

plot.EmpiricalTTT

Examples

```
library(EstimationTools)

#-------------------------------------------------------------------------------
# Example 1: Scaled empirical TTT from 'mgus1' data from 'survival' package.

TTT_1 <- TTTE_Analytical(Surv(stop, event == 'pcm') ~1, method = 'cens',
                         data = mgus1, subset=(start == 0))
head(TTT_1$`i/n`)
head(TTT_1$phi_n)
print(TTT_1$strata)


#-------------------------------------------------------------------------------
# Example 2: Scaled empirical TTT using a factor variable with 'aml' data
# from 'survival' package.

TTT_2 <- TTTE_Analytical(Surv(time, status) ~ x, method = "cens", data = aml)
head(TTT_2$`i/n`)
head(TTT_2$phi_n)
print(TTT_2$strata)

#-------------------------------------------------------------------------------
# Example 3: Non-scaled empirical TTT without a factor (arbitrarily simulated
# data).

set.seed(911211)
y <- rweibull(n=20, shape=1, scale=pi)
TTT_3 <- TTTE_Analytical(y ~ 1, scaled = FALSE)
head(TTT_3$`i/n`)
head(TTT_3$phi_n)
print(TTT_3$strata)
```

```
#-------------------------------------------------------------------------------
# Example 4: non-scaled empirical TTT without a factor (arbitrarily simulated
# data) using the 'response' argument (this is equivalent to Third example).

set.seed(911211)
y <- rweibull(n=20, shape=1, scale=pi)
TTT_4 <- TTTE_Analytical(response = y, scaled = FALSE)
head(TTT_4$`i/n`)
head(TTT_4$phi_n)
print(TTT_4$strata)

#-------------------------------------------------------------------------------
# Eample 5: empirical TTT with a continuously variant term for the shape
# parameter in Weibull distribution.

x <- runif(50, 0, 10)
shape <- 0.1 + 0.1*x
y <- rweibull(n = 50, shape = shape, scale = pi)

partitions <- list(method='quantile-based',
                   folds=5)
TTT_5 <- TTTE_Analytical(y ~ x, partition_method = partitions)
head(TTT_5$`i/n`)
head(TTT_5$phi_n)
print(TTT_5$strata)
plot(TTT_5) # Observe changes in Empirical TTT

#-------------------------------------------------------------------------------
```

TTT_hazard_shape *Hazard Shape estimation from TTT plot*

Description

[Experimental]

This function can be used so as to estimate hazard shape corresponding to a given data set. This is a wrapper for TTTE_Analytical.

Usage

```
TTT_hazard_shape(object, ...)

## S3 method for class 'formula'
TTT_hazard_shape(
  formula,
  data = NULL,
```

```
  local_reg = loess.options(),
  interpolation = interp.options(),
  silent = FALSE,
  ...
)

## S3 method for class 'EmpiricalTTT'
TTT_hazard_shape(
  object,
  local_reg = loess.options(),
  interpolation = interp.options(),
  silent = FALSE,
  ...
)
```

Arguments

object	An alternative way for getting the hazard shape estimation in passing directly the EmpiricalTTT object generated with TTTE_Analytical.
...	further arguments passed to TTTE_Analytical.
formula	An object of class formula with the response on the left of an operator ~. The right side must be 1.
data	an optional data frame containing the response variables. If data is not specified, the variables are taken from the environment from which TTT_hazard_shape is called.
local_reg	a list of control parameters for LOESS. See loess.options.
interpolation	a list of control parameters for interpolation function. See interp.options.
silent	logical. If TRUE, warnings of TTT_hazard_shape are suppressed.

Details

This function performs a non-parametric estimation of the empirical total time on test (TTT) plot. Then, this estimated curve can be used so as to get suggestions about initial values and the search region for parameters based on hazard shape associated to the shape of empirical TTT plot.

Use Hazard_Shape function to get the results for shape estimation.

Author(s)

Jaime Mosquera Gutiérrez <jmosquerag@unal.edu.co>

See Also

print.HazardShape, plot.HazardShape, TTTE_Analytical

Examples

```
#-------------------------------------------------------------------------------
# Example 1: Increasing hazard and its corresponding TTT statistic with
#            simulated data

hweibull <- function(x, shape, scale){
  dweibull(x, shape, scale)/pweibull(x, shape, scale, lower.tail = FALSE)
  }
curve(hweibull(x, shape = 2.5, scale = pi), from = 0, to = 42,
               col = "red", ylab = "Hazard function", las = 1, lwd = 2)

y <- rweibull(n = 50, shape = 2.5, scale = pi)
status <- c(rep(1, 48), rep(0, 2))
my_initial_guess1 <- TTT_hazard_shape(Surv(y, status) ~ 1)
my_initial_guess1$hazard_type


#-------------------------------------------------------------------------------
# Example 2: Same example using an 'EmpiricalTTT' object

y <- rweibull(n = 50, shape = 2.5, scale = pi)
TTT_wei <- TTTE_Analytical(y ~ 1)
my_initial_guess2 <- TTT_hazard_shape(TTT_wei)
my_initial_guess2$hazard_type


#-------------------------------------------------------------------------------
# Example 3: Increasing hazard with simulated censored data

hweibull <- function(x, shape, scale){
  dweibull(x, shape, scale)/pweibull(x, shape, scale, lower.tail = FALSE)
  }
curve(hweibull(x, shape = 2.5, scale = pi), from = 0, to = 42,
               col = "red", ylab = "Hazard function", las = 1, lwd = 2)

y <- rweibull(n = 50, shape = 2.5, scale = pi)
y <- sort(y)
status <- c(rep(1, 45), rep(0, 5))
my_initial_guess1 <- TTT_hazard_shape(Surv(y, status) ~ 1)
my_initial_guess1$hazard_type


#-------------------------------------------------------------------------------
```

`uniform_key` *Uniform key function*

Description

[Experimental]

This function provides the uniform key function for model fitting in distance sampling.

Usage

```
uniform_key(x, w)
```

Arguments

x	vector of perpendicular distances from the transect.
w	half width of the strip transect.

Details

This is the uniform key function with parameter sigma. Its expression is given by

$g(x) = \frac{1}{w}$,

for x > 0.

Value

A numeric value corresponding to a given value of x and w.

Author(s)

Jaime Mosquera Gutiérrez, <jmosquerag@unal.edu.co>

See Also

Other key functions: half_norm_key(), hazard_rate_key()

Examples

```
library(EstimationTools)

#---------------------------------------------------------------------------
# Example: Uniform function
uniform_key(x=1, w=100)
curve(uniform_key(x, w=100), from=0, to=10, ylab='g(x)')

#---------------------------------------------------------------------------
```

Index