Package 'Countr'

January 18, 2026

Type Package

Title Flexible Univariate Count Models Based on Renewal Processes

Version 3.6.1

Description Flexible univariate count models based on renewal processes. The models may include covariates and can be specified with familiar formula syntax as in glm() and package 'flexsurv'. The methodology is described by Kharrat et all (2019) <doi:10.18637/jss.v090.i13> (included as vignette 'Countr_guide' in the package).

License GPL (>= 2)

URL https://geobosh.github.io/Countr/ (doc), https://CRAN.R-project.org/package=Countr

BugReports https://github.com/GeoBosh/Countr/issues

Depends R (>= 3.3.0)

Imports Rcpp (>= 0.11.3), flexsurv, Formula, VGAM (>= 1.1-1), optimx, numDeriv, boot, MASS, utils, Rdpack (>= 0.7-0), dplyr, standardize, pscl, car, Matrix

LinkingTo Rcpp, RcppArmadillo

Suggests testthat, knitr, xtable, lmtest, lattice, RColorBrewer

RdMacros Rdpack

VignetteBuilder knitr

LazyData true

Encoding UTF-8

RoxygenNote 7.3.3

NeedsCompilation yes

Collate 'Countr-package.R' 'RcppExports.R' 'anc.R' 'coefnames.R' 'convCount_loglik.R' 'convCount_moments.R' 'convCount_probs.R' 'dBivariateWeibull.R' 'dWeibull.R' 'dWeibullgamma.R' 'data.R' 'renewal_IV.R' 'renewal_tools.R' 'renewal_cstr.R' 'tools.R' 'renewal_methods.R'

Config/build/clean-inst-doc FALSE

Author Tarak Kharrat [aut] (ORCID: <https://orcid.org/0000-0001-9399-6174>),
Georgi N. Boshnakov [aut, cre] (ORCID:
<https://orcid.org/0000-0003-2839-346X>)

Maintainer Georgi N. Boshnakov <georgi.boshnakov@manchester.ac.uk>

Repository CRAN

Date/Publication 2026-01-18 15:40:02 UTC

Contents

`Countr-package` *Flexible Univariate Count Models Based on Renewal Processes*

Description

Flexible univariate count models based on renewal processes. The models may include covariates and can be specified with familiar formula syntax as in glm() and 'flexsurv'.

Details

The methodology is described by Kharrat et al. (2019). The paper is included in the package as vignette `vignette('Countr_guide_paper', package = "Countr")`.

The main function is `renewalCount`, see its documentation for examples.

Goodness of fit chi-square (likelihood ratio and Pearson) tests for glm and count renewal models are implemented in `chiSq_gof` and `chiSq_pearson`.

Author(s)

Maintainer: Georgi N. Boshnakov <`georgi.boshnakov@manchester.ac.uk`> (ORCID)

Authors:

- Tarak Kharrat <`tarak@realanalytics.co.uk`> (ORCID)
- Georgi N. Boshnakov <`georgi.boshnakov@manchester.ac.uk`> (ORCID)

References

Kharrat T, Boshnakov GN, McHale I, Baker R (2019). "Flexible Regression Models for Count Data Based on Renewal Processes: The Countr Package." *Journal of Statistical Software*, **90**(13), 1–35. doi:10.18637/jss.v090.i13.

Baker R, Kharrat T (2017). "Event count distributions from renewal processes: fast computation of probabilities." *IMA Journal of Management Mathematics*, **29**(4), 415-433. ISSN 1471-6798, doi:10.1093/imaman/dpx008, https://academic.oup.com/imaman/article-pdf/29/4/415/25693854/dpx008.pdf.

Boshnakov G, Kharrat T, McHale IG (2017). "A bivariate Weibull count model for forecasting association football scores." *International Journal of Forecasting*, **33**(2), 458–466.

Cameron AC, Trivedi PK (2013). *Regression analysis of count data*, volume 53. Cambridge university press.

Kharrat T, Boshnakov GN, McHale IG, Baker R (2018). "Flexible regression models for count data based on renewal processes: the Countr package." *Journal of Statistical Software (to appear)*.

McShane B, Adrian M, Bradlow ET, Fader PS (2008). "Count models based on Weibull interarrival times." *Journal of Business & Economic Statistics*, **26**(3), 369–378.

Winkelmann R (1995). "Duration dependence and dispersion in count-data models." *Journal of Business & Economic Statistics*, **13**(4), 467–474.

See Also

Useful links:

- https://geobosh.github.io/Countr/ (doc)
- https://CRAN.R-project.org/package=Countr
- Report bugs at https://github.com/GeoBosh/Countr/issues

---

addBootSampleObject — *Create a bootsrap sample for coefficient estimates*

Description

Create a boostrap sample from coefficient estimates.

Usage

```
addBootSampleObject(object, ...)
```

Arguments

object	an object to add boot object to.
...	extra parameters to be passed to the boot::boot() function other than data and statistic.

Details

The information in object is used to prepare the arguments and then boot is called to generate the bootstrap sample. The bootstrap sample is stored in object as component "boot". Arguments in "..." can be used customise the boot() call.

Value

object with additional component "boot"

See Also

renewal_methods

Examples

```
## see renewal_methods
```

`chiSq_gof` *Formal Chi-square goodness-of-fit test*

Description

Carry out the formal chi-square goodness-of-fit test described by Cameron (2013).

Usage

```
chiSq_gof(object, breaks, ...)

## S3 method for class 'renewal'
chiSq_gof(object, breaks, ...)

## S3 method for class 'negbin'
chiSq_gof(object, breaks, ...)

## S3 method for class 'glm'
chiSq_gof(object, breaks, ...)
```

Arguments

`object`	an object from class `renewal`.
`breaks`	integer values at which the breaks shoudl happen. The function will compute the observed frequencies in the intervals `[breaks[i],breaks[i + 1])`.
`...`	currently not used.

Details

The test is a conditional moment test described in details in Cameron (2013, Section 5.3.4). We compute the asymptotically equivalent outer product of the gradient version which is justified for renewal models (fully parametric + parameters based on MLE).

Value

data.frame

References

Cameron AC, Trivedi PK (2013). *Regression analysis of count data*, volume 53. Cambridge university press.

See Also

`chiSq_pearson`

`chiSq_pearson` *Pearson Chi-Square test*

Description

Carry out Pearson Chi-Square test and compute the Pearson statistic.

Usage

```
chiSq_pearson(object, ...)

## S3 method for class 'renewal'
chiSq_pearson(object, ...)

## S3 method for class 'glm'
chiSq_pearson(object, ...)
```

Arguments

`object`	an object from class `renewal`.
`...`	currently not used.

Details

The computation is inspired from Cameron(2013) Chapter 5.3.4. Observed and fitted frequencies are computed and the contribution of every observed cell to the Pearson's chi-square test statistic is reported. The idea is to check if the fitted model has a tendancy to over or under predict some ranges of data

Value

data.frame with 5 columns given the count values (`Counts`), observed frequencies (`Actual`), model's prediction (`Predicted`), the difference (`Diff`) and the contribution to the Pearson's statistic (`Pearson`).

References

Cameron AC, Trivedi PK (2013). *Regression analysis of count data*, volume 53. Cambridge university press.

See Also

`chiSq_gof`

`compareToGLM` *Compare renewals fit to glm models fit*

Description

Compare renewals fit to glm models fit on the same data.

Usage

```
compareToGLM(poisson_model, breaks, nbinom_model, ...)
```

Arguments

`poisson_model`	fitted Poisson glm model
`breaks`	integer values at which the breaks should happen. The function will compute the observed frequencies in the intervals `[breaks[i],breaks[i + 1])`.
`nbinom_model`	fitted negative binomial (fitted using `MASS::glm.nb()`). This argument is optional.
`...`	renewal models to be considered.

Details

This function computes a data.frame similar to Table 5.6 in Cameron(2013), using the observed frequencies and predictions from different models. Supported models accepted are Poisson and negative binomial (fitted using `MASS::glm.nb()`) from the glm family and any model from the renewal family (passed in `...`).

Value

data.frame with columns `Counts`, `Actual` (observed probability) and then 2 columns per model passed (predicted probability and pearson statistic) for the associated count value.

References

Cameron AC, Trivedi PK (2013). *Regression analysis of count data*, volume 53. Cambridge university press.

`CountrFormula` *Create a formula for renewalCount*

Description

Create a formula for renewalCount

Usage

```
CountrFormula(response, ...)
```

Arguments

`response`	the formula for the "main" parameter. It also specifies the response variable.
`...`	additional arguments for the ancilliary parameters.

Value

a Formula object suitable for argument `formula` of `renewalCount()`.

`count_table` *Summary of a count variable*

Description

Summary of a count variable.

Usage

```
count_table(count, breaks, formatChar = FALSE)
```

Arguments

`count`	integer, observed count value for every individual in the sample.
`breaks`	integer, values at which the breaks should happen. The function will compute the observed frequency in `[breaks[i], breaks[i + 1])`.
`formatChar`	logical, should the values be converted to character and formatted?

Details

The function does a similar job to `table()` with more flexibility introduced by the argument `breaks`. The user can decide how to break the count values and decide to merge some cells if needed.

Value

`matrix` with 2 rows and `length(breaks)` columns. The column names are the cells names. The rows are the observed frequencies and relative frequencies (probabilities).

`dBivariateWeibullCountFrankCopula`
Density and log-likelihood of the Bivariate Frank Copula Weibull Count model

---

Description

Compute density and log-likelihood of the Bivariate Frank Copula Weibull Count model.

Usage

```
dBivariateWeibullCountFrankCopula(
  x,
  y,
  shapeX,
  scaleX,
  shapeY,
  scaleY,
  theta,
  method = c("series_acc", "conv_dePril"),
  time = 1,
  log = FALSE,
  conv_steps = 100,
  conv_extrap = TRUE,
  series_terms = 50,
  series_acc_niter = 300,
  series_acc_eps = 1e-10
)

dBivariateWeibullCountFrankCopula_loglik(
  x,
  y,
  shapeX,
  scaleX,
  shapeY,
  scaleY,
  theta,
  method = c("series_acc", "conv_dePril"),
  time = 1,
  na.rm = TRUE,
  conv_steps = 100,
  conv_extrap = TRUE,
  series_terms = 50,
```

```
  series_acc_niter = 300,
  series_acc_eps = 1e-10,
  weights = NULL
)
```

Arguments

x, y	numeric, the desired counts.
shapeX, shapeY	numeric, shape parameters. Either length(x) or length(1).
scaleX, scaleY	numeric, scale parameters (length(x)).
theta	numeric, Frank copula parameter.
method	character method to be used. Choices are "series_acc" (accelerated series expansion) or "conv_dePril" (convolution by dePril algorithm).
time	numeric, length of the observation window (defaults to 1).
log	TODO
conv_steps	integer, number of steps to use in the computation of the integral.
conv_extrap	logical, if TRUE, Richardson extrapolation will be applied to improve accuracy.
series_terms	number of terms used in series expansion.
series_acc_niter	number of iterations in the acceleration algorithm.
series_acc_eps	double, tolerance to declare convergence in the acceleration algorithm.
na.rm	logical, should NAs (obtained from log of small probabilities) be replaced with the smallest allowed probability?
weights	numeric vector of weights to apply. If NULL, a vector of ones.

Details

dBivariateWeibullCountFrankCopula computes the probabilities $P(X(t) = x(t), Y(t) = y(t))$, where $X(t), Y(t)$ is a bivariate Weibull count process in which the bivariate distribution is modelled by Frank copulas.

Value

for dBivariateWeibullCountFrankCopula, a vector of the (log-)probabilities.

for dBivariateWeibullCountFrankCopula_loglik, the log-likelihood of the model, a number.

Examples

```
## first 10 cases from "estimationParams.RDS", rounded for presentation
gam_weiH <-  0.9530455
gam_weiA <-  1.010051
theta    <- -0.3703702
HG <- c(0, 0, 0, 2, 1, 0, 2, 0, 1, 2)
AG <- c(2, 2, 1, 1, 6, 1, 0, 2, 0, 1)
lambdaHome <- c(1.5, 1.0, 1.3, 1.8, 1.3, 1.2, 1.3, 1.0, 2.0, 1.4)
lambdaAway <- c(1.2, 2.4, 1.3, 0.7, 1.3, 1.4, 0.6, 1.6, 0.6, 1.3)
```

```
weiFrank0 <- dBivariateWeibullCountFrankCopula(
    HG, AG, gam_weiH, lambdaHome, gam_weiA, lambdaAway, theta,
    "series_acc", 1, TRUE)

weiFrank1 <- dBivariateWeibullCountFrankCopula(
    HG, AG, gam_weiH, lambdaHome, gam_weiA, lambdaAway, theta,
    "conv_dePril", 1, TRUE, conv_extrap = TRUE)


weights <- c(0.01355306, 0.01355306, 0.01355306, 0.01355306, 0.01355306,
             0.01355306, 0.01355306, 0.01355306, 0.01357825, 0.01357825)

weiFrank2 <- dBivariateWeibullCountFrankCopula_loglik(
    HG, AG, gam_weiH, lambdaHome, gam_weiA, lambdaAway, theta,
    "conv_dePril", 1, TRUE, conv_extrap = TRUE, weights = weights)

weiFrank3 <- dBivariateWeibullCountFrankCopula_loglik(
    HG, AG, gam_weiH, lambdaHome, gam_weiA, lambdaAway, theta,
    "series_acc", 1, TRUE, weights = weights)

cbind(weiFrank0, weiFrank1, weiFrank2, weiFrank3)
## rdname dRenewalFrankCopula_user
```

---

dCount_conv_bi *Compute count probabilities using convolution*

Description

Compute count probabilities using one of several convolution methods. dCount_conv_bi does the computations for the distributions with builtin support in this package.

dCount_conv_user does the same using a user defined survival function.

Usage

```
dCount_conv_bi(
  x,
  distPars,
  dist = c("weibull", "gamma", "gengamma", "burr"),
  method = c("dePril", "direct", "naive"),
  nsteps = 100,
  time = 1,
  extrap = TRUE,
  log = FALSE
)

dCount_conv_user(
  x,
```

```
  distPars,
  extrapolPars,
  survR,
  method = c("dePril", "direct", "naive"),
  nsteps = 100,
  time = 1,
  extrap = TRUE,
  log = FALSE
)
```

Arguments

x	integer (vector), the desired count values.
distPars	Rcpp::List with distribution specific slots, see section 'Details'.
dist	character name of the built-in distribution, see section 'Details'.
method	character string, the method to use, see section 'Details'.
nsteps	unsiged integer, number of steps used to compute the integral.
time	double, time at wich to compute the probabilities. Set to 1 by default.
extrap	logical, if TRUE, Richardson extrapolation will be applied to improve accuracy.
log	logical, if TRUE the log-probability will be returned.
extrapolPars	vector of length 2, the extrapolation values.
survR	function, user supplied survival function; should have signature function(t, distPars), where t is a positive real number (the time where the survival function is evaluated) and distPars is a list of distribution parameters. It should return a double value.

Details

dCount_conv_bi computes count probabilities using one of several convolution methods for the distributions with builtin support in this package.

The following convolution methods are implemented: "dePril", "direct", and "naive".

The builtin distributions currently are Weibull, gamma, generalised gamma and Burr.

Value

vector of probabilities $P(x(i),\ i = 1, ..., n)$ where n is the length of x.

Examples

```
x <- 0:10
lambda <- 2.56
p0 <- dpois(x, lambda)
ll <- sum(dpois(x, lambda, TRUE))

err <- 1e-6
## all-probs convolution approach
distPars <- list(scale = lambda, shape = 1)
```

```
pmat_bi <- dCount_conv_bi(x, distPars, "weibull", "direct",
                          nsteps = 200)

## user pwei
pwei_user <- function(tt, distP) {
    alpha <- exp(-log(distP[["scale"]]) / distP[["shape"]])
    pweibull(q = tt, scale = alpha, shape = distP[["shape"]],
             lower.tail = FALSE)
}

pmat_user <- dCount_conv_user(x, distPars, c(1, 2), pwei_user, "direct",
                              nsteps = 200)
max((pmat_bi - p0)^2 / p0)
max((pmat_user - p0)^2 / p0)

## naive convolution approach
pmat_bi <- dCount_conv_bi(x, distPars, "weibull", "naive",
                          nsteps = 200)
pmat_user <- dCount_conv_user(x, distPars, c(1, 2), pwei_user, "naive",
                              nsteps = 200)
max((pmat_bi- p0)^2 / p0)
max((pmat_user- p0)^2 / p0)

## dePril conv approach
pmat_bi <- dCount_conv_bi(x, distPars, "weibull", "dePril",
                          nsteps = 200)
pmat_user <- dCount_conv_user(x, distPars, c(1, 2), pwei_user, "dePril",
                              nsteps = 200)
max((pmat_bi- p0)^2 / p0)
max((pmat_user- p0)^2 / p0)
```

`dCount_conv_loglik_bi` *Log-likelihood of a count probability computed by convolution (bi)*

Description

Compute the log-likelihood of a count model using convolution methods to compute the probabilities. `dCount_conv_loglik_bi` is for the builtin distributions. `dCount_conv_loglik_user` is for user defined survival functions.

Usage

```
dCount_conv_loglik_bi(
  x,
  distPars,
  dist = c("weibull", "gamma", "gengamma", "burr"),
  method = c("dePril", "direct", "naive"),
  nsteps = 100,
```

```
  time = 1,
  extrap = TRUE,
  na.rm = TRUE,
  weights = NULL
)

dCount_conv_loglik_user(
  x,
  distPars,
  extrapolPars,
  survR,
  method = c("dePril", "direct", "naive"),
  nsteps = 100,
  time = 1,
  extrap = TRUE,
  na.rm = TRUE,
  weights = NULL
)
```

Arguments

x	integer (vector), the desired count values.
distPars	list of the same length as x with each slot being itself a named list containing the distribution parameters corresponding to x[i].
dist	character name of the built-in distribution, see section 'Details'.
method	character, convolution method to be used; choices are "dePril" (section 3.2), "direct" (section 2) or "naive" (section 3.1).
nsteps	unsiged integer number of steps used to compute the integral.
time	double time at wich to compute the probabilities. Set to 1 by default.
extrap	logical if TRUE, Richardson extrapolation will be applied to improve accuracy.
na.rm	logical, if TRUE, NAs (produced by taking the log of very small probabilities) will be replaced by the smallest allowed probability; default is TRUE.
weights	numeric, vector of weights to apply. If NULL, a vector of ones.
extrapolPars	list of same length as x where each slot is a vector of length 2 (the extrapolation values to be used) corresponding to x[i].
survR	a user defined survival function; should have the signature function(t, distPars) where t is a real number (>0) where the survival function is evaluated and distPars is a list of distribution parameters. It should return a double value.

Value

numeric, the log-likelihood of the count process

Examples

```
x <- 0:10
lambda <- 2.56
distPars <- list(scale = lambda, shape = 1)
distParsList <- lapply(seq(along = x), function(ind) distPars)
extrapolParsList <- lapply(seq(along = x), function(ind) c(2, 1))
## user pwei
pwei_user <- function(tt, distP) {
    alpha <- exp(-log(distP[["scale"]]) / distP[["shape"]])
    pweibull(q = tt, scale = alpha, shape = distP[["shape"]],
             lower.tail = FALSE)
}

## log-likehood allProbs Poisson
dCount_conv_loglik_bi(x, distParsList,
                      "weibull", "direct", nsteps = 400)

dCount_conv_loglik_user(x, distParsList, extrapolParsList,
                        pwei_user, "direct", nsteps = 400)

## log-likehood naive Poisson
dCount_conv_loglik_bi(x, distParsList,
                      "weibull", "naive", nsteps = 400)

dCount_conv_loglik_user(x, distParsList, extrapolParsList,
                        pwei_user, "naive", nsteps = 400)

## log-likehood dePril Poisson
dCount_conv_loglik_bi(x, distParsList,
                      "weibull", "dePril", nsteps = 400)

dCount_conv_loglik_user(x, distParsList, extrapolParsList,
                        pwei_user, "dePril", nsteps = 400)
## see dCount_conv_loglik_bi()
```

---

dmodifiedCount_bi — *Compute count probabilities based on modified renewal process (bi)*

Description

Compute count probabilities based on modified renewal process using dePril algorithm. `dmodifiedCount_bi` does it for the builtin distributions.

`dmodifiedCount_user` does the same for a user specified distribution.

Usage

```
dmodifiedCount_bi(
  x,
  distPars,
```

```
  dist,
  distPars0,
  dist0,
  nsteps = 100L,
  time = 1,
  extrap = TRUE,
  cdfout = FALSE,
  logFlag = FALSE
)

dmodifiedCount_user(
  x,
  distPars,
  survR,
  distPars0,
  survR0,
  extrapolPars,
  nsteps = 100L,
  time = 1,
  extrap = TRUE,
  cdfout = FALSE,
  logFlag = FALSE
)
```

Arguments

x	integer (vector), the desired count values.
distPars0, distPars	Rcpp::List with distribution specific slots for the first arrival and the rest of the process respectively.
dist0, dist	character, name of the first and following survival distributions.
nsteps	unsiged integer number of steps used to compute the integral.
time	double time at wich to compute the probabilities. Set to 1 by default.
extrap	logical if TRUE, Richardson extrapolation will be applied to improve accuracy.
cdfout	TODO
logFlag	logical if TRUE the log-probability will be returned.
survR0, survR	user supplied survival function; should have signature function(t, distPars), where t is a positive real number (the time at which the survival function is evaluated) and distPars is a list of distribution parameters. It should return a double value (first arrival and following arrivals respectively).
extrapolPars	list of same length as x, where each slot is a vector of length 2 (the extrapolation values to be used) corresponding to x[i].

Details

For the modified renewal process the first arrival is allowed to have a different distribution from the time between subsequent arrivals. The renewal assumption is kept.

Value

vector of probabilities P(x(i)) for i = 1, ..., n where n is the length of x.

---

dRenewalFrankCopula_user *Bivariate Count probability Using Frank copula (user)*

Description

Bivariate Count probability Using Frank copula to model dependence using user passed survival objects

Bivariate Count probability Using Frank copula to model dependence using built-in distributions

Usage

```
dRenewalFrankCopula_user(
  x,
  y,
  survX,
  survY,
  distParsX,
  distParsY,
  extrapolParsX,
  extrapolParsY,
  theta,
  time = 1,
  logFlag = FALSE,
  nsteps = 100L,
  extrap = TRUE
)

dRenewalFrankCopula_bi(
  x,
  y,
  distX,
  distY,
  distParsX,
  distParsY,
  theta,
  time = 1,
  logFlag = FALSE,
  nsteps = 100L,
  extrap = TRUE
)
```

Arguments

- `x, y` — numeric vector the desired counts.
- `survX, survY` — R functions: the survival functions.
- `distParsX, distParsY` — List of Lists. Each slot is a named vector of distribution parameters.
- `extrapolParsX, extrapolParsY` — list vec of length 2 values of the Richardson extrapolation parameters for the inputted distribution.
- `theta` — double Frank copula parameter.
- `time` — double time at wich to compute the probabilities. Set to 1 by default.
- `logFlag` — TODO
- `nsteps` — unsiged integer number of steps used to compute the integral.
- `extrap` — logical if `TRUE`, Richardson extrapolation will be applied to improve accuracy. TODO: (this is for arg. method, maybe!) param dePrilConv logical if `TRUE` the dePril method will be applied to compute convolution. Otherwise, the binary decomposition of section 3 will be used.
- `distX, distY` — character name of the survival distribution.

Details

We use Frank copula to model depepndence between 2 renewal count processes obtained from user passed inter-arrival distribution defined by `survPtr`, `distPars` and `extrapolPars`.

Value

(log) probability of the bivariate count $P(X(t) = x_i, Y(t) = y_i)$ where x_i and y_i are the ith component of the X and Y respectively.

(log) probability of the bivariate count $P(X(t) = x_i, Y(t) = y_i)$ where x_i and y_i are the ith component of the X and Y respectively.

`dWeibullCount` *Probability calculations for Weibull count models*

Description

Probability computations for the univariate Weibull count process. Several methods are provided. `dWeibullCount` computes probabilities.

`dWeibullCount_loglik` computes the log-likelihood.

`evWeibullCount` computes the expected value and variance.

Usage

```
dWeibullCount(
  x,
  shape,
  scale,
  method = c("series_acc", "series_mat", "conv_direct", "conv_naive", "conv_dePril"),
  time = 1,
  log = FALSE,
  conv_steps = 100,
  conv_extrap = TRUE,
  series_terms = 50,
  series_acc_niter = 300,
  series_acc_eps = 1e-10
)

dWeibullCount_loglik(
  x,
  shape,
  scale,
  method = c("series_acc", "series_mat", "conv_direct", "conv_naive", "conv_dePril"),
  time = 1,
  na.rm = TRUE,
  conv_steps = 100,
  conv_extrap = TRUE,
  series_terms = 50,
  series_acc_niter = 300,
  series_acc_eps = 1e-10,
  weights = NULL
)

evWeibullCount(
  xmax,
  shape,
  scale,
  method = c("series_acc", "series_mat", "conv_direct", "conv_naive", "conv_dePril"),
  time = 1,
  conv_steps = 100,
  conv_extrap = TRUE,
  series_terms = 50,
  series_acc_niter = 300,
  series_acc_eps = 1e-10
)
```

Arguments

x	integer (vector), the desired count values.
shape	numeric (length 1), shape parameter of the Weibull count.
scale	numeric (length 1), scale parameter of the Weibull count.

method	character, one of the available methods, see section 'Details'.
time	double, length of the observation window (defaults to 1).
log	logical, if TRUE, the log of the probability will be returned.
conv_steps	numeric, number of steps used for the extrapolation.
conv_extrap	logical, should Richardson extrappolation be applied ?
series_terms	numeric, number of terms in the series expansion.
series_acc_niter	numeric, number of iterations in the Euler-van Wijngaarden algorithm.
series_acc_eps	numeric, tolerance of convergence in the Euler-van Wijngaarden algorithm.
na.rm	logical, if TRUE NA's (produced by taking the log of very small probabilities) will be replaced by the smallest allowed probaility; default is TRUE.
weights	numeric, vector of weights to apply. If NULL, a vector of one's will be applied.
xmax	unsigned integer, maximum count to be used.

Details

Argument method can be used to specify the desired method, as follows:

"series_mat" - series expansion using matrix techniques,

"series_acc" - Euler-van Wijngaarden accelerated series expansion (default),

"conv_direc"t - direct convolution method of section 2,

"conv_naive" - naive convolurion described in section 3.1,

"conv_dePril" - dePril convolution described in section 3.2.

The arguments have sensible default values.

Value

for dWeibullCount, a vector of probabilities $P(x(i)), i = 1, \ldots n$, where $n =$ length(x).

for dWeibullCount_loglik, a double, the log-likelihood of the count process.

for evWeibullCount, a list with components:

ExpectedValue	expected value,
Variance	variance.

dWeibullgammaCount_mat_Covariates

Univariate Weibull Count Probability with gamma and covariate heterogeneity

Description

Univariate Weibull Count Probability with gamma and covariate heterogeneity

Usage

```
dWeibullgammaCount_mat_Covariates(
  x,
  cc,
  r,
  alpha,
  Xcovar,
  beta,
  t = 1,
  logFlag = FALSE,
  jmax = 100L
)
```

Arguments

x, cc, t, logFlag, jmax	TODO keywords internal
r	numeric shape of the gamma distribution
alpha	numeric rate of the gamma distribution
Xcovar	matrix covariates value
beta	numeric vector of slopes

evCount_conv_bi

Expected value and variance of a renewal count process

Description

Compute numerically expected values and variances of renewal count processes.

Usage

```
evCount_conv_bi(
  xmax,
  distPars,
  dist = c("weibull", "gamma", "gengamma", "burr"),
  method = c("dePril", "direct", "naive"),
  nsteps = 100,
  time = 1,
  extrap = TRUE
)

evCount_conv_user(
  xmax,
  distPars,
  extrapolPars,
  survR,
  method = c("dePril", "direct", "naive"),
  nsteps = 100,
  time = 1,
  extrap = TRUE
)
```

Arguments

`xmax`	unsigned integer maximum count to be used.
`distPars`	TODO
`dist`	TODO
`method`	TODO
`nsteps`	unsiged integer, number of steps used to compute the integral.
`time`	double, time at wich to compute the probabilities. Set to 1 by default.
`extrap`	logical, if `TRUE`, Richardson extrapolation will be applied to improve accuracy.
`extrapolPars`	ma::vec of length 2. The extrapolation values.
`survR`	function, user supplied survival function; should have signature `function(t, distPars)`, where `t` is a positive real number (the time where the survival function is evaluated) and `distPars` is a list of distribution parameters. It should return a double value.

Details

`evCount_conv_bi` computes the expected value and variance of renewal count processes for the builtin distirbutions of inter-arrival times.

`evCount_conv_user` computes the expected value and variance for a user specified distribution of the inter-arrival times.

Value

a named list with components `ExpectedValue` and `Variance`

Examples

```
pwei_user <- function(tt, distP) {
    alpha <- exp(-log(distP[["scale"]]) / distP[["shape"]])
    pweibull(q = tt, scale = alpha, shape = distP[["shape"]],
             lower.tail = FALSE)
}

## ev convolution Poisson count
lambda <- 2.56
beta <- 1
distPars <- list(scale = lambda, shape = beta)

evbi <- evCount_conv_bi(20, distPars, dist = "weibull")
evu <- evCount_conv_user(20, distPars, c(2, 2), pwei_user, "dePril")

c(evbi[["ExpectedValue"]], lambda)
c(evu[["ExpectedValue"]], lambda )
c(evbi[["Variance"]], lambda     )
c(evu[["Variance"]], lambda      )

## ev convolution weibull count
lambda <- 2.56
beta <- 1.35
distPars <- list(scale = lambda, shape = beta)

evbi <- evCount_conv_bi(20, distPars, dist = "weibull")
evu <- evCount_conv_user(20, distPars, c(2.35, 2), pwei_user, "dePril")

x <- 1:20
px <- dCount_conv_bi(x, distPars, "weibull", "dePril",
                     nsteps = 100)
ev <- sum(x * px)
var <- sum(x^2 * px) - ev^2

c(evbi[["ExpectedValue"]], ev)
c(evu[["ExpectedValue"]], ev )
c(evbi[["Variance"]], var    )
c(evu[["Variance"]], var     )
```

---

fertility *Fertility data*

Description

Fertility data analysed by Winkelmann(1995). The data comes from the second (1985) wave of German Socio-Economic Panel. The sample is formed by 1,243 women aged 44 or older in 1985. The response variable is the number of children per woman and explanatory variables are described in more details below.

Usage

```
fertility
```

Format

A data frame with 9 variables (5 factors, 4 integers) and 1243 observations:

`children` integer; response variable: number of children per woman (integer).
`german` factor; is the mother German? (yes or no).
`years_school` integer; education measured as years of schooling.
`voc_train` factor; vocational training ? (yes or no)
`university` factor; university education ? (yes or no)
`religion` factor; mother's religion: Catholic, Protestant, Muslim or Others (reference).
`rural` factor; rural (yes or no ?)
`year_birth` integer; year of birth (last 2 digits)
`age_marriage` integer; age at marriage

For further details, see Winkelmann (1995).

References

Winkelmann R (1995). "Duration dependence and dispersion in count-data models." *Journal of Business & Economic Statistics*, **13**(4), 467–474.

`football` *Football data*

Description

Final scores of all matches in the English Premier League from seasons 2009/2010 to 2016/2017.

Usage

```
football
```

Format

a data.frame with 6 columns and 1104 observations:

`seasonId` integer season identifier (year of the first month of competition).
`gameDate` POSIXct game date and time.
`homeTeam,awayTeam` character home and away team name.
`homeTeamGoals,awayTeamGoals` integer number of goals scored by the home and the away team.

Details

The data were collected from https://www.football-data.co.uk/ and slightly formatted and simplified.

`frequency_plot` *Plot a frequency chart*

Description

Plot a frequency chart to compare actual and predicted values.

Usage

```
frequency_plot(count_labels, actual, pred, colours = character(0))
```

Arguments

`count_labels`	character, labels to be used.
`actual`	numeric, the observed probabilities for the different count specified in `count_labels`.
`pred`	data.frame of predicted values. Should have the same number of rows as actual and one column per model. The columns' names will be used as labels for the different models.
`colours`	character vector of colour codes with `length ncol(pred) + 2`. If `colours` is missing or `length(colours) < ncol(pred) + 2`, the remaining colours are generated using `RColorBrewer::brewer.pal`.

Details

In order to compare actual and fitted values, a barchart plot is created. It is the user's responsibility to provide the count, observed and fitted values.

If argument `colour` is missing or not of sufficient length, the colours are set automatically using a function from package **RColorBrewer**.

The bar chart is created with `lattice::barchart`. If `frequency_plot` is called from the command line, the returned value is automatically 'printed' (i.e., the plot is produced). Otherwise, for example in scripts, you may need to use `print()` on the returned value.

Value

an object from class `"trellis"`

getParNames — *Return the names of distribution parameters*

Description

Return the names of the parameters of a count distribution.

Usage

```
getParNames(dist, ...)
```

Arguments

`dist`	character, name of the distribution.
`...`	parameters to pass when `dist == "custom"`.

Value

character vector with the names of the distribution parameters

predict.renewal — *Predict method for renewal objects*

Description

Compute predictions from renewal objects.

Usage

```
## S3 method for class 'renewal'
predict(
  object,
  newdata = NULL,
  type = c("response", "prob"),
  se.fit = FALSE,
  terms = NULL,
  na.action = na.pass,
  time = 1,
  ...
)
```

Arguments

`object`	Object of class inheriting from `"lm"`
`newdata`	An optional data frame in which to look for variables with which to predict. If omitted, the fitted values are used.
`type`	type of prediction. If equal to `"response"`, give the mean probability associated with the individual covariates. If `"prob"`, give the probability of the observed count.
`se.fit`	A switch indicating if standard errors are required.
`terms`	If `type = "terms"`, which terms (default is all terms), a `character` vector.
`na.action`	function determining what should be done with missing values in `newdata`. The default is to predict `NA`.
`time`	TODO
`...`	further arguments passed to or from other methods.

Examples

```
fn <- system.file("extdata", "McShane_Wei_results_boot.RDS", package = "Countr")
object <- readRDS(fn)
data <- object$data
## old data
predOld.response <- predict(object, type = "response", se.fit = TRUE)
predOld.prob <- predict(object, type = "prob", se.fit = TRUE)

## newData (extracted from old Data)
newData <- head(data)
predNew.response <- predict(object, newdata = newData,
                            type = "response", se.fit = TRUE)
predNew.prob <- predict(object, newdata = newData,
                        type = "prob", se.fit = TRUE)

cbind(head(predOld.response$values),
           head(predOld.response$se$scale),
           head(predOld.response$se$shape),
           predNew.response$values,
           predNew.response$se$scale,
           predNew.response$se$shape)

cbind(head(predOld.prob$values),
      head(predOld.prob$se$scale),
      head(predOld.prob$se$shape),
      predNew.prob$values,
      predNew.prob$se$scale,
      predNew.prob$se$shape)
```

renewalCoef *Get named vector of coefficients for renewal objects*

Description

Get named vector of coefficients for renewal objects.

Usage

```
renewalCoef(object, ...)
```

Arguments

`object`	an object, there are methods for several classes, see section 'Details'.
`...`	further arguments to be passed to `renewalNames`, usually something like `target = "weibull"`.

Details

This is a convenience function for constructing named vector of coefficients for renewal count models. Such vectors are needed, for example, for starting values in the model fitting procedures. The simplest way to get a suitably named vector is to take the coefficients of a fitted model but if the fitting procedure requires initial values, this is seemingly a circular situation.

The overall idea is to take coefficients specified by `object` and transform them to coefficients suitable for a renewal count model as specified by the arguments `"..."`. The provided methods eliminate the need for tedius manual preparation of such vectors and in the most common cases allow the user to do this in a single line.

The default method extracts the coefficients of `object` using `co <- coef(object)` (an error is raised if this fails). It prepares a named numeric vector with names requested by the arguments in `"..."` and assigns `co` to the first `length(co)` elements of the prepared vector. The net effect is that the coefficients of a model can be initialised from the coefficients of a nested model. For example a Poisson regression model can be used to initialise a Weibull count model. Of course the non-zero shape parameter(s) of the Weibull model need to be set separately.

If `object` is from class `glm`, the method is identical to the default method.

If object is from class `renewalCoefList`, its elements are simply concatenated in one long vector.

References

Kharrat T, Boshnakov GN, McHale I, Baker R (2019). "Flexible Regression Models for Count Data Based on Renewal Processes: The Countr Package." *Journal of Statistical Software*, **90**(13), 1–35. doi:10.18637/jss.v090.i13.

See Also

`renewalNames`

`renewalCoefList` *Split a vector using the prefixes of the names for grouping*

Description

Split a vector using the prefixes of the names for grouping.

Usage

```
renewalCoefList(coef)
```

Arguments

`coef`	a named vector

Details

The names of the coefficients of renewal regression models are prefixed with the names of the parameters to which they refer. This function splits such vectors into a list with one component for each parameter. For example, for a Weibull renewal regression model this will create a list with components `"scale"` and `"shape"`.

This is a convenience function allowing users to manipulate the coefficients related to a parameter more easily. `renewalCoef` can convert this list back to a vector.

See Also

`renewalNames`, `renewalCoef`

`renewalCount` *Fit renewal count processes regression models*

Description

Fit renewal regression models for count data via maximum likelihood.

Usage

```
renewalCount(
  formula,
  data,
  subset,
  na.action,
  weights,
  offset,
  dist = c("weibull", "weibullgam", "custom", "gamma", "gengamma"),
```

```
    anc = NULL,
    convPars = NULL,
    link = NULL,
    time = 1,
    control = renewal.control(...),
    customPars = NULL,
    seriesPars = NULL,
    weiMethod = NULL,
    computeHessian = TRUE,
    standardise = FALSE,
    standardise_scale = 1,
    model = TRUE,
    y = TRUE,
    x = FALSE,
    ...
)
```

Arguments

formula
: a formula object. If it is a standard formula object, the left hand side specifies the response variable and the right hand sides specifies the regression equation for the first parameter of the conditional distribution. `formula` can also be used to specify the ancilliary regressions, using the operator '|', see section 'Details'.

data, subset, na.action
: arguments controlling formula processing via `model.frame`.

weights
: optional numeric vector of weights.

offset
: optional numeric vector with an a priori known component to be included in the linear predictor of the count model. Currently not used.

dist
: character, built-in distribution to be used as the inter-arrival time distribution or "custom" for a user defined distribution, see section 'Details'. Currently the built-in distributions are "weibull", "weibullgam", "gamma", "gengamma" (generalized-gamma) and "burr".

anc
: a named list of formulas for ancillary regressions, if any, otherwise `NULL`. The formulas associated with the (exact) parameter names are used. The left-hand sides of the formulas in `anc` are ignored.

convPars
: a list of convolution parameters arguments with slots `nsteps`, `extrap` and `convMethod`, see `dCount_conv_bi`. If NULL, default parameters will be applied.

link
: named list of character strings specifying the name of the link functions to be used in the regression. If `NULL`, the canonical link function will be used, i.e, `log` if the parameter is supposed to be positive, identity otherwise.

time
: numeric, time at which the count is observed; default to unity (1).

control
: a list of control arguments specified via `renewal.control`.

customPars
: list, user inputs if `dist = "custom"`, see section 'Details'.

seriesPars
: list, series expansion input parameters with slots `terms` (number of terms in the series expansion), `iter` (number of iteration in the accelerated series ex-

pansion algorithm) and eps (tolerance in the accelerated series expansion algorithm), Only used if dist = "weibull" and weiMethod = c("series_mat", "series_acc").

weiMethod — character, computation method to be used if dist = "weibull" or "weibullgam", see dWeibullCount and dWeibullgammaCount.

computeHessian — logical, should the hessian (and hence the covariance matrix) be computed numerically at the fitted values.

standardise — logical, should the covariates be standardised using standardize::standardize() function.

standardise_scale — numeric the desired scale for the covariates; defaults to 1.

model, y, x — logicals. If TRUE the corresponding components of the fit (model frame, response, model matrix) are returned.

... — arguments passed to renewal.control in the default setup.

Details

renewal re-uses design and functionality of the basic R tools for fitting regression model (lm, glm) and is highly inspired by hurdle() and zeroinfl() from package pscl. Package Formula is used to handle formulas.

Argument formula is a formula object. In the simplest case its left-hand side (lhs) designates the response variable and the right-hand side the covariates for the first parameter of the distribution (as reported by getParNames. In this case, covariates for the ancilliary parameters are specified using argument anc.

The ancilliary regressions, can also be specified in argument formula by adding them to the righhand side, separated by the operator '|'. For example Y | shape ~ x + y | z can be used in place of the pair Y ~ x + y and anc = list(shape = ~z). In most cases, the name of the second parameter can be omitted, which for this example gives the equivalent Y ~ x + y | z. The actual rule is that if the parameter is missing from the left-hand side, it is inferred from the default parameter list of the distribution.

As another convenience, if the parameters are to to have the same covariates, it is not necessary to repeat the rhs. For example, Y | shape ~ x + y is equivalent to Y | shape ~ x + y | x + y. Note that this is applied only to parameters listed on the lhs, so Y ~ x + y specifies covariates only for the response variable and not any other parameters.

Distributions for inter-arrival times supported internally by this package can be chosen by setting argument "dist" to a suitable character string. Currently the built-in distributions are "weibull", "weibullgam", "gamma", "gengamma" (generalized-gamma) and "burr".

Users can also provide their own inter-arrival distribution. This is done by setting argument "dist" to "custom", specifying the initial values and giving argument customPars as a list with the following components:

parNames character, the names of the parameters of the distribution. The location parameter should be the first one.

survivalFct function object containing the survival function. It should have signature function(t, distPars) where t is the point where the survival function is evaluated and distPars is the list of the distribution parameters. It should return a double value.

extrapolFct function object computing the extrapolation values (numeric of length 2) from the value of the distribution parameters (in `distPars`). It should have signature `function(distPars)` and return a numeric vector of length 2. Only required if the extrapolation is set to `TRUE` in `convPars`.

Some checks are done to validate `customPars` but it is user's responsibility to make sure the the functions have the appropriate signatures.

Note: The Weibull-gamma distribution is an experimental version and should be used with care! It is very sensitive to initial values and there is no guarantee of convergence. It has also been reparameterized in terms of $(1/r, 1/\alpha, c)$ instead of (r, α, c), where r and α are the shape and scale of the gamma distribution and c is the shape of the Weibull distribution.

(2017-08-04(Georgi) experimental feature: probability residuals in component 'probResiduals'. I also added type 'prob' to the method for residuals() to extract them.

probResiduals[i] is currently 1 - Prob(Y[i] given the covariates). "one minus", so that values close to zero are "good". On its own this is probably not very useful but when comparing two models, if one of them has mostly smaller values than the other, there is some reason to claim that the former is superior. For example (see below), gamModel < poisModel in 3:1

Value

An `S3` object of class `"renewal"`, which is a list with components including:

coefficients values of the fitted coefficients.

residuals vector of weighted residuals $\omega * (observed - fitted)$.

fitted.values vector of fitted means.

optim data.frame output of `optimx`.

method optimisation algorithm.

control the control arguments, passed to `optimx`.

start starting values, passed to `optimx`.

weights weights to apply, if any.

n number of observations (with weights > 0).

iterations number of iterations in the optimisation algorithm.

execTime duration of the optimisation.

loglik log-likelihood of the fitted model.

df.residual residuals' degrees of freedom for the fitted model.

vcoc convariance matrix of all coefficients, computed numerically from the hessian at the fitted coefficients (if `computeHessian` is `TRUE`).

dist name of the inter-arrival distribution.

link list, inverse link function corresponding to each parameter in the inter-arrival distribution.

converged logical, did the optimisation algorithm converge?

data data used to fit the model.

formula the original formula.

call the original function call.

anc named list of formulas to model regression on ancillary parameters.

score_fct function to compute the vector of scores defined in Cameron and Trivedi (2013), equation 2.94.

convPars convolution inputs used.

customPars named list, user passed distribution inputs, see section 'Details'.

time observed window used, default is 1.0 (see inputs).

model the full model frame (if `model = TRUE`).

y the response count vector (if `y = TRUE`).

x the model matrix (if `x = TRUE`).

References

Kharrat T, Boshnakov GN, McHale I, Baker R (2019). "Flexible Regression Models for Count Data Based on Renewal Processes: The Countr Package." *Journal of Statistical Software*, **90**(13), 1–35. doi:10.18637/jss.v090.i13.

Cameron AC, Trivedi PK (2013). *Regression analysis of count data*, volume 53. Cambridge university press.

Examples

```
## Not run:
## may take some time to run depending on your CPU
data(football)
wei = renewalCount(formula = homeTeamGoals ~ 1,
                    data = football, dist = "weibull", weiMethod = "series_acc",
                    computeHessian = FALSE, control = renewal.control(trace = 0,
                    method = "nlminb"))

## End(Not run)
```

`renewalNames` *Get names of parameters of renewal regression models*

Description

Get names of parameters of renewal regression models

Usage

```
renewalNames(object, ...)
```

Arguments

`object`	an object.
`...`	further arguments.

Details

renewalNames gives the a character vector of names of parameters for renewal regression models. There are two main use scenarios: renewalNames(object, target = "dist") and renewalNames(object,...). In the first scenario target can be a count distribution, such as "weibull" or a parameter name, such as shape. In this case renewalNames transforms coefficient names of object to those specified by target. In the second cenario the argument list is the same that would be used to call renewalCount. In this case renewalNames returns the names that would be used by renewalCount for the coefficients of the fitted model.

See Also

renewalCoefList, renewalCoef

---

renewal_methods — *Methods for renewal objects*

Description

Methods for renewal objects.

Usage

```
## S3 method for class 'renewal'
coef(object, ...)

## S3 method for class 'renewal'
vcov(object, ...)

## S3 method for class 'renewal'
residuals(object, type = c("pearson", "response", "prob"), ...)

## S3 method for class 'renewal'
residuals_plot(object, type = c("pearson", "response", "prob"), ...)

## S3 method for class 'renewal'
fitted(object, ...)

## S3 method for class 'renewal'
confint(
  object,
  parm,
  level = 0.95,
  type = c("asymptotic", "boot"),
  bootType = c("norm", "bca", "basic", "perc"),
  ...
)
```

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

## S3 method for class 'renewal'
print(x, digits = max(3, getOption("digits") - 3), ...)

## S3 method for class 'summary.renewal'
print(
  x,
  digits = max(3, getOption("digits") - 3),
  width = getOption("width"),
  ...
)

## S3 method for class 'renewal'
model.matrix(object, ...)

## S3 method for class 'renewal'
logLik(object, ...)

## S3 method for class 'renewal'
nobs(object, ...)

## S3 method for class 'renewal'
extractAIC(fit, scale, k = 2, ...)

## S3 method for class 'renewal'
addBootSampleObject(object, ...)

## S3 method for class 'renewal'
df.residual(object, ...)
```

Arguments

`object`	an object from class "renewal".
`...`	further arguments for methods.
`type`, `parm`, `level`, `bootType`, `x`, `digits`	see the corresponding generics and section 'Details'.
`width`	numeric width length.
`fit`, `scale`, `k`	same as in the generic.

Details

Objects from class "renewal" represent fitted count renewal models and are created by calls to "renewalCount()". There are methods for this class for many of the familiar functions for interacting with fitted models.

Examples

```
fn <- system.file("extdata", "McShane_Wei_results_boot.RDS", package = "Countr")
object <- readRDS(fn)
class(object) # "renewal"

coef(object)
vcov(object)

## Pearson residuals: rescaled by sd
head(residuals(object, "pearson"))
## response residuals: not rescaled
head(residuals(object, "response"))

head(fitted(object))

## loglik, nobs, AIC, BIC
c(loglik = as.numeric(logLik(object)), nobs = nobs(object),
  AIC = AIC(object), BIC = BIC(object))

asym <- se.coef(object, , "asymptotic")
boot <- se.coef(object, , "boot")
cbind(asym, boot)
## CI for coefficients
asym <- confint(object, type = "asymptotic")
## Commenting out for now, see the nite in the code of confint.renewal():
## boot <- confint(object, type = "boot", bootType = "norm")
## list(asym = asym, boot = boot)
summary(object)
print(object)
## see renewal_methods
## see renewal_methods
```

---

residuals_plot *Method to visualise the residuals*

Description

A method to visualise the residuals

Usage

```
residuals_plot(object, type, ...)
```

Arguments

object	object returned by one of the count modeling functions.
type	character type of residuals to be used.
...	further arguments for methods.

se.coef *Extract Standard Errors of Model Coefficients*

Description

Extract standard errors of model coefficients from objects returned by count modeling functions.

Usage

```
se.coef(object, parm, type, ...)

## S3 method for class 'renewal'
se.coef(object, parm, type = c("asymptotic", "boot"), ...)
```

Arguments

`object`	an object returned by one of the count modeling functions.
`parm`	parameter's name or index.
`type`	type of standard error: asymtotic normal standard errors (`"asymptotic"`) or bootsrap (`"boot"`).
`...`	further arguments for methods.

Details

The method for class `"renewal"` extracts standard errors of model coefficients from objects returned by `renewal`. When bootsrap standard error are requested, the function checks for the bootsrap sample in `object`. If it is not found, the bootsrap sample is created and a warning is issued. Users can choose between asymtotic normal standard errors (`asymptotic`) or bootsrap (`boot`).

Value

a named numeric vector

Examples

```
## see examples for renewal_methods
```

surv *Wrapper to built-in survival functions*

Description

Wrapper to built-in survival functions

Usage

```
surv(t, distPars, dist)
```

Arguments

t	double, time point where the survival is to be evaluated at.
distPars	Rcpp::List with distribution specific slots, see section 'Details'.
dist	character name of the built-in distribution, see section 'Details'.

Details

The function wraps all builtin-survival distributions. User can choose between the weibull, gamma, gengamma(generalized gamma) and burr (Burr type XII distribution). It is the user responsibility to pass the appropriate list of parameters as follows:

weibull scale (the scale) and shape (the shape) parameters.

burr scale (the scale) and shape1 (the shape1) and shape2 (the shape2) parameters.

gamma scale (the scale) and shape (the shape) parameter.

gengamma mu (location), sigma (scale) and Q (shape) parameters.

Value

a double, giving the value of the survival function at time point t at the parameters' values.

Examples

```
tt <- 2.5
## weibull

distP <- list(scale = 1.2, shape = 1.16)
alpha <- exp(-log(distP[["scale"]]) / distP[["shape"]])
pweibull(q = tt, scale = alpha, shape = distP[["shape"]],
                      lower.tail = FALSE)
surv(tt, distP, "weibull") ## (almost) same

## gamma
distP <- list(shape = 0.5, rate = 1.0 / 0.7)
pgamma(q = tt, rate = distP[["rate"]], shape = distP[["shape"]],
                    lower.tail = FALSE)
```

```
surv(tt, distP, "gamma")  ## (almost) same

## generalized gamma
distP <- list(mu = 0.5, sigma = 0.7, Q = 0.7)
flexsurv::pgengamma(q = tt, mu = distP[["mu"]],
                    sigma = distP[["sigma"]],
                    Q = distP[["Q"]],
                    lower.tail = FALSE)
surv(tt, distP, "gengamma")  ## (almost) same
```

Index