Package: DiscreteDists (via r-universe)

October 11, 2026

Title Discrete Statistical Distributions

Version 1.1.3

Description Implementation of new discrete statistical distributions. Each distribution includes the traditional functions as well as an additional function called the family function, which can be used to estimate parameters within the 'gamlss' framework.

License MIT + file LICENSE

Imports gamlss, pracma, Rcpp, COMPoissonReg, nleqslv

LinkingTo Rcpp

Encoding UTF-8

Roxygen list(markdown = TRUE)

Suggests knitr, rmarkdown

URL https://github.com/fhernanb/DiscreteDists

BugReports https://github.com/fhernanb/DiscreteDists/issues

Depends R (>= 4.1)

LazyData true

Config/roxygen2/version 8.1.0

Repository https://fhernanb.r-universe.dev

Date/Publication 2026-09-06 23:57:39 UTC

RemoteUrl https://github.com/fhernanb/discretedists

RemoteRef HEAD

RemoteSha 0cf4beece20f7729552346e64efca47a959e5848

Contents

add 3
azpro 4
BerG 5
checklink 7

COMPO . . . 7
COMPO2 . . . 10
dBerG . . . 12
DBH . . . 15
dCOMPO . . . 17
dCOMPO2 . . . 19
dDBH . . . 22
dDGEII . . . 25
dDIKUM . . . 27
dDLD . . . 30
dDMOLBE . . . 33
dDPERKS . . . 35
dDsPA . . . 38
DGEII . . . 40
dGGEO . . . 42
dHYPERPO . . . 45
dHYPERPO2 . . . 48
DIKUM . . . 50
DLD . . . 52
DMOLBE . . . 55
dNPGL . . . 58
dNPGL2 . . . 61
DPERKS . . . 64
dPOISXL . . . 66
dPTRTE . . . 69
DsPA . . . 71
estim_mu_sigma_NPGL . . . 74
estim_mu_sigma_NPGL_MM . . . 75
estim_mu_sigma_NPGL2 . . . 75
GGEO . . . 76
grazing . . . 79
HYPERPO . . . 80
HYPERPO2 . . . 81
logLik_NPGL . . . 83
logLik_NPGL2 . . . 84
make.link.gamlss . . . 84
mean_var_hp . . . 86
NMES1988 . . . 87
NPGL . . . 88
NPGL2 . . . 90
plot_discrete_cdf . . . 93
POISXL . . . 94
PTRTE . . . 96

Index **99**

add *Sum of One-Dimensional Functions*

Description

Sum of One-Dimensional Functions

Usage

```
add(f, lower, upper, ..., abs.tol = .Machine$double.eps)
```

Arguments

f	an R function taking a numeric first argument and returning a numeric vector of the same length.
lower	the lower limit of sum. Can be infinite.
upper	the upper limit of sum. Can be infinite.
...	additional arguments to be passed to f.
abs.tol	absolute accuracy requested.

Value

This function returns the sum value.

Author(s)

Freddy Hernandez, <fhernanb@unal.edu.co>

Examples

```
# Poisson expected value
add(f=function(x, lambda) x*dpois(x, lambda), lower=0, upper=Inf,
    lambda=7.5)

# Binomial expected value
add(f=function(x, size, prob) x*dbinom(x, size, prob), lower=0, upper=20,
    size=20, prob=0.5)

# Examples with infinite series
add(f=function(x) 0.5^x, lower=0, upper=100) # Ans=2
add(f=function(x) (1/3)^(x-1), lower=1, upper=Inf) # Ans=1.5
add(f=function(x) 4/(x^2+3*x+2), lower=0, upper=Inf, abs.tol=0.001) # Ans=4.0
add(f=function(x) 1/(x*(log(x)^2)), lower=2, upper=Inf, abs.tol=0.000001) # Ans=2.02
add(f=function(x) 3*0.7^(x-1), lower=1, upper=Inf) # Ans=10
add(f=function(x, a, b) a*b^(x-1), lower=1, upper=Inf, a=3, b=0.7) # Ans=10
add(f=function(x, a=3, b=0.7) a*b^(x-1), lower=1, upper=Inf) # Ans=10
```

azpro *azpro dataset*

Description

Data come from the 1991 Arizona cardiovascular patient files. A subset of the fields was selected to model the differential length of stay for patients entering the hospital to receive one of two standard cardiovascular procedures: CABG and PTCA. CABG is the standard acronym for Coronary Artery Bypass Graft, where the flow of blood in a diseased or blocked coronary artery or vein has been grafted to bypass the diseased sections. PTCA, or Percutaneous Transluminal Coronary Angioplasty, is a method of placing a balloon in a blocked coronary artery to open it to blood flow. It is a much less severe method of treatment for those having coronary blockage, with a corresponding reduction in risk.

Usage

```
azpro
```

Format

azpro**:**

A data frame with 3589 observations on the following 6 variables:

los length of hospital stay

procedure 1=CABG; 0=PTCA

sex 1=Male; 0=female

admit 1=Urgent/Emerg; 0=elective (type of admission)

age75 1=Age>75; 0=Age<=75

hospital encrypted facility code (string)

Author(s)

COUNT R package.

Source

This dataset comes from the COUNT R package.

BerG *The Bernoulli-geometric distribution*

Description

The function `BerG()` defines the Bernoulli-geometric distribution, a two parameter distribution, for a `gamlss.family` object to be used in GAMLSS fitting using the function `gamlss()`.

Usage

```
BerG(mu.link = "log", sigma.link = "log")
```

Arguments

`mu.link`	defines the mu.link, with "log" link as the default for the mu parameter.
`sigma.link`	defines the sigma.link, with "log" link as the default for the sigma.

Details

The BerG distribution with parameters μ and σ has a support 0, 1, 2, ... and mass function given by

$f(x|\mu,\sigma) = \frac{(1-\mu+\sigma)}{(1+\mu+\sigma)}$ if $x = 0$,

$f(x|\mu,\sigma) = 4\mu\frac{(\mu+\sigma-1)^{x-1}}{(\mu+\sigma+1)^{x+1}}$ if $x = 1, 2, ...,$

with $\mu > 0$, $\sigma > 0$ and $\sigma > |\mu - 1|$.

Value

Returns a `gamlss.family` object which can be used to fit a BerG distribution in the `gamlss()` function.

Author(s)

Hermes Marques, <hermes.marques@ufrn.br>

References

Bourguignon, M., & de Medeiros, R. M. (2022). A simple and useful regression model for fitting count data. Test, 31(3), 790-827.

See Also

dBerG.

Examples

```
# Example 1
# Generating some random values with
# known mu and sigma
y <- rBerG(n=500, mu=0.75, sigma=0.5)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y~1, family=BerG,
               control=gamlss.control(n.cyc=500, trace=FALSE))

# Extracting the fitted values for mu and sigma
exp(coef(mod1, what="mu"))
exp(coef(mod1, what="sigma"))

# Example 2
# Generating random values under some model

# A function to simulate a data set with Y ~ BerG
gendat <- function(n) {
  x1 <- runif(n)
  x2 <- runif(n)
  x3 <- runif(n)
  x4 <- runif(n)
  mu    <- exp(1 + 1.2*x1 + 0.2*x2)
  sigma <- exp(2 + 1.5*x3 + 1.5*x4)
  y <- rBerG(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1, x2=x2, x3=x3, x4=x4)
}

set.seed(16494786)
datos <- gendat(n=500)

mod2 <- gamlss(y~x1+x2, sigma.fo=~x3+x4, family=BerG, data=datos,
               control=gamlss.control(n.cyc=500, trace=TRUE))

summary(mod2)

# Example using the dataset grazing from the bergreg package
# https://github.com/rdmatheus/bergreg

# This example corresponds to example 5.1
# presented by Bourguignon & Medeiros (2022)
# A simple and useful regression model for fitting count data

data("grazing")
hist(grazing$birds)

mod3 <- gamlss(birds ~ when + grazed,
               sigma.fo=~1,
               family=BerG, data=grazing,
               control=gamlss.control(n.cyc=500, trace=TRUE))
```

```
summary(mod3)
```

---

checklink *Set the Right Link Function for Specified Parameter and Distribution*

Description

This function is used within the distribution family specification of a GAMLSS model to define the right link for each of the parameters of the distribution. This function should not be called by the user unless he/she specify a new distribution family or wishes to change existing link functions in the parameters.

Usage

```
checklink(which.link = NULL, which.dist = NULL, link = NULL, link.List = NULL)
```

Arguments

`which.link`	which parameter link e.g. `which.link="mu.link"`
`which.dist`	which distribution family e.g. `which.dist="Cole.Green"`
`link`	a repetition of `which.link` e.g. `link=substitute(mu.link)`
`link.List`	what link function are required e.g. `link.List=c("inverse", "log", "identity")`

Details

This function is the same `checklink` fuction from gamlss.dist package.

Value

Defines the right link for each parameter.

COMPO *The COMPO family*

Description

The function `COMPO()` defines the Conway-Maxwell-Poisson distribution, a two parameter distribution, for a `gamlss.family` object to be used in GAMLSS fitting using the function `gamlss()`.

Usage

```
COMPO(mu.link = "log", sigma.link = "log")
```

Arguments

`mu.link`	defines the mu.link, with "log" link as the default for the mu parameter.
`sigma.link`	defines the sigma.link, with "log" link as the default for the sigma.

Details

The COMPO distribution with parameters μ and σ has a support 0, 1, 2, ... and mass function given by

$f(x|\mu,\sigma) = \frac{\mu^x}{(x!)^\sigma Z(\mu,\sigma)}$

with $\mu > 0$, $\sigma \geq 0$ and

$Z(\mu,\sigma) = \sum_{j=0}^{\infty} \frac{\mu^j}{(j!)^\sigma}$.

The proposed functions here are based on the functions from the COMPoissonReg package.

Value

Returns a `gamlss.family` object which can be used to fit a COMPO distribution in the `gamlss()` function.

Author(s)

Freddy Hernandez, <fhernanb@unal.edu.co>

References

Shmueli, G., Minka, T. P., Kadane, J. B., Borle, S., & Boatwright, P. (2005). A useful distribution for fitting discrete data: revival of the Conway–Maxwell–Poisson distribution. Journal of the Royal Statistical Society Series C: Applied Statistics, 54(1), 127-142.

See Also

dCOMPO.

Examples

```
# Example 1
# Generating some random values with
# known mu and sigma
## Not run:
set.seed(12)
y <- rCOMPO(n=100, mu=10, sigma=3)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y~1, sigma.fo=~1, family=COMPO,
               control=gamlss.control(n.cyc=500, trace=TRUE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
exp(coef(mod1, what="mu"))
```

```
exp(coef(mod1, what="sigma"))

## End(Not run)

# Example 2
# Generating random values under some model

## Not run:
# A function to simulate a data set with Y ~ COMPO
gendat <- function(n) {
  x1 <- runif(n)
  x2 <- runif(n)
  mu    <- exp(2 + 1 * x1) # 12 approximately
  sigma <- exp(2 - 2 * x2) # 2.71 approximately
  y <- rCOMPO(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1, x2=x2)
}

set.seed(123)
dat <- gendat(n=100)

# Fitting the model
mod2 <- NULL
mod2 <- gamlss(y~x1, sigma.fo=~x2, family=COMPO, data=dat,
               control=gamlss.control(n.cyc=500, trace=TRUE))

summary(mod2)

## End(Not run)

# Example 3
# Using the data from Shmueli et al. (2005) page 134
# The dataset consists of quarterly sales of a well-known brand of a
# particular article of clothing at stores of a large national retailer.
## Not run:
values <- 0:30
freq <- c(514, 503, 457, 423, 326, 233, 195, 139, 101, 77, 56, 40,
          37, 22, 9, 7, 10, 9, 3, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1)

y <- rep(x=values, times=freq)

mod3 <- gamlss(y~1, sigma.fo=~1, family=COMPO,
               control=gamlss.control(n.cyc=500, trace=TRUE))

exp(coef(mod3, what="mu"))
exp(coef(mod3, what="sigma"))

estim_mu_sigma_COMPO(y)

library(COMPoissonReg)
fit <- glm.cmp(y ~ 1)
res <- exp(fit$opt.res$par)
res
```

```
## End(Not run)

# Example 4
# Using the data from Shmueli et al. (2005) page 134
# The dataset contains lengths of words (numbers of syllables)
# in a Hungarian dictionary

## Not run:
# Slovak dictionary
y <- rep(x=1:5, times=c(7, 33, 49, 22, 6))

# Hungarian dictionary
y <- rep(x=1:9, times=c(1421, 12333, 20711, 15590, 5544, 1510, 289, 60, 1))

mod4 <- gamlss(y~1, sigma.fo=~1, family=COMPO,
               control=gamlss.control(n.cyc=500, trace=TRUE))

exp(coef(mod4, what="mu"))
exp(coef(mod4, what="sigma"))

estim_mu_sigma_COMPO(y)

library(COMPoissonReg)
fit <- glm.cmp(y ~ 1)
res <- exp(fit$opt.res$par)
res


## End(Not run)
```

COMPO2 *The COMPO2 family (with mu as mean)*

Description

The function `COMPO2()` defines the Conway-Maxwell-Poisson distribution a two parameter distribution, for a `gamlss.family` object to be used in GAMLSS fitting using the function `gamlss()`. This parameterization was proposed by Ribeiro et al. (2020) and the main characteristic is that $E(X) = \mu$.

Usage

```
COMPO2(mu.link = "log", sigma.link = "identity")
```

Arguments

`mu.link`	defines the mu.link, with "log" link as the default for the mu parameter.
`sigma.link`	defines the sigma.link, with "identity" link as the default for the sigma.

Details

The COMPO2 distribution with parameters μ and σ has a support 0, 1, 2, ... and mass function given by

$f(x|\mu,\sigma) = \left(\mu + \frac{\exp(\sigma)-1}{2\exp(\sigma)}\right)^{x\exp(\sigma)} \frac{(x!)^{\exp(\sigma)}}{Z(\mu,\sigma)}$

with $\mu > 0$, $\sigma \in \Re$ and

$Z(\mu,\sigma) = \sum_{j=0}^{\infty} \frac{\mu^j}{(j!)^\sigma}$.

The proposed functions here are based on the functions from the COMPoissonReg package.

Value

Returns a `gamlss.family` object which can be used to fit a COMPO2 distribution in the `gamlss()` function.

Author(s)

Freddy Hernandez, <fhernanb@unal.edu.co>

References

Ribeiro Jr, Eduardo E., et al. "Reparametrization of COM–Poisson regression models with applications in the analysis of experimental data." Statistical Modelling 20.5 (2020): 443-466.

See Also

dCOMPO2.

Examples

```
# Example 1
# Generating some random values with
# known mu and sigma
y <- rCOMPO2(n=500, mu=10, sigma=-1)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y~1, sigma.fo=~1, family=COMPO2,
               control=gamlss.control(n.cyc=500, trace=TRUE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
exp(coef(mod1, what="mu"))
coef(mod1, what="sigma")

# Example 2
# Generating random values under some model

## Not run:
# A function to simulate a data set with Y ~ COMPO2
gendat <- function(n) {
```

```
  x1 <- runif(n)
  x2 <- runif(n)
  mu    <- exp(2 + 1 * x1) # 12 approximately
  sigma <- 1 - 2 * x2      # 0 approximately
  y <- rCOMPO2(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1, x2=x2)
}

set.seed(123)
dat <- gendat(n=200)

# Fitting the model
mod2 <- NULL
mod2 <- gamlss(y~x1, sigma.fo=~x2, family=COMPO2, data=dat)

summary(mod2)

## End(Not run)
```

---

dBerG *Bernoulli-geometric distribution*

Description

These functions define the density, distribution function, quantile function and random generation for the Bernoulli-geometric distribution with parameters μ and σ.

Usage

```
dBerG(x, mu, sigma, log = FALSE)

pBerG(q, mu, sigma, lower.tail = TRUE, log.p = FALSE)

rBerG(n, mu, sigma)

qBerG(p, mu, sigma, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x, q	vector of (non-negative integer) quantiles.
mu	vector of the mu parameter.
sigma	vector of the sigma parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
n	number of random values to return.
p	vector of probabilities.

Details

The BerG distribution with parameters μ and σ has a support 0, 1, 2, ... and mass function given by

$f(x|\mu,\sigma) = \frac{(1-\mu+\sigma)}{(1+\mu+\sigma)}$ if $x = 0$,

$f(x|\mu,\sigma) = 4\mu\frac{(\mu+\sigma-1)^{x-1}}{(\mu+\sigma+1)^{x+1}}$ if $x = 1, 2, ...,$

with $\mu > 0$, $\sigma > 0$ and $\sigma > |\mu - 1|$.

Value

`dBerG` gives the density, `pBerG` gives the distribution function, `qBerG` gives the quantile function, `rBerG` generates random deviates.

Author(s)

Hermes Marques, <hermes.marques@ufrn.br>

References

Bourguignon, M., & de Medeiros, R. M. (2022). A simple and useful regression model for fitting count data. Test, 31(3), 790-827.

See Also

BerG.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

x_max <- 20
probs1 <- dBerG(x=0:x_max, mu=0.7, sigma=0.5)
probs2 <- dBerG(x=0:x_max, mu=0.3, sigma=1)
probs3 <- dBerG(x=0:x_max, mu=2, sigma=3)

# To plot the first k values
plot(x=0:x_max, y=probs1, type="o", lwd=2, col="dodgerblue", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for BerG",
     ylim=c(0, 0.80))
points(x=0:x_max, y=probs2, type="o", lwd=2, col="tomato")
points(x=0:x_max, y=probs3, type="o", lwd=2, col="green4")
legend("topright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=0.7, sigma=0.5",
                "mu=0.3, sigma=1",
                "mu=2, sigma=3"))

# Example 2
# Checking if the cumulative curves converge to 1

#plot1
```

```
x_max <- 10
plot_discrete_cdf(x=0:x_max,
                  fx=dBerG(x=0:x_max, mu=1, sigma=2),
                  col="dodgerblue",
                  main="CDF for BerG",
                  lwd=3)
legend("bottomright", legend="mu=1, sigma=2",
       col="dodgerblue", lty=1, lwd=2, cex=0.8)


#plot2
plot_discrete_cdf(x=0:x_max,
                  fx=dBerG(x=0:x_max, mu=3, sigma=3),
                  col="tomato",
                  main="CDF for BerG",
                  lwd=3)
legend("bottomright", legend="mu=3, sigma=3",
       col="tomato", lty=1, lwd=2, cex=0.8)


#plot3
plot_discrete_cdf(x=0:x_max,
                  fx=dBerG(x=0:x_max, mu=5, sigma=5),
                  col="green4",
                  main="CDF for BerG",
                  lwd=3)
legend("bottomright", legend="mu=5, sigma=5",
       col="green4", lty=1, lwd=2, cex=0.8)


# Example 3
# Comparing the random generator output with
# the theoretical probabilities

x_max <- 15
probs1 <- dBerG(x=0:x_max, mu=0.5, sigma=5)
names(probs1) <- 0:x_max

x <- rBerG(n=1000, mu=0.5, sigma=5)
probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
mp <- barplot(height, beside=TRUE, names.arg=cn,
              col=c("dodgerblue3","firebrick3"), las=1,
              xlab="X", ylab="Proportion")
legend("topright",
       legend=c("Theoretical", "Simulated"),
       bty="n", lwd=3,
       col=c("dodgerblue3","firebrick3"), lty=1)

# Example 4
# Checking the quantile function
```

```
mu <- 1
sigma <- 2
p <- seq(from=0, to=1, by=0.01)
qxx <- qBerG(p=p, mu=mu, sigma=sigma, lower.tail=TRUE, log.p=FALSE)
plot(p, qxx, type="s", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles of DBerG(mu=1, sigma=2)")
```

DBH *The Discrete Burr Hatke family*

Description

The function `DBH()` defines the Discrete Burr Hatke distribution a single parameter distribution, for a `gamlss.family` object to be used in GAMLSS fitting using the function `gamlss()`.

Usage

```
DBH(mu.link = "logit")
```

Arguments

`mu.link`	defines the mu.link, with "logit" link as the default for the mu parameter. Other links are "probit" and "cloglog"'(complementary log-log)

Details

The Discrete Burr-Hatke distribution with parameter μ has a support 0, 1, 2, ... and its probability mass function (pmf) is given by

$f(x|\mu) = (\frac{1}{x+1} - \frac{\mu}{x+2})\mu^x$

The pmf is log-convex for all values of $0 < \mu < 1$, where $\frac{f(x+1;\mu)}{f(x;\mu)}$ is an increasing function in x for all values of the parameter μ.

Note: in this implementation we changed the original parameters λ for μ, we did it to implement this distribution within gamlss framework.

Value

Returns a `gamlss.family` object which can be used to fit a Discrete Burr-Hatke distribution in the `gamlss()` function.

Author(s)

Valentina Hurtado Sepulveda, <vhurtados@unal.edu.co>

References

El-Morshedy, M., Eliwa, M. S., & Altun, E. (2020). Discrete Burr-Hatke distribution with properties, estimation methods and regression model. IEEE access, 8, 74359-74370.

See Also

dDBH.

Examples

```
# Example 1
# Generating some random values with
# known mu
y <- rDBH(n=1000, mu=0.74)

library(gamlss)
mod1 <- gamlss(y~1, family=DBH,
               control=gamlss.control(n.cyc=500, trace=FALSE))

# Extracting the fitted values for mu
# using the inverse logit function
inv_logit <- function(x) exp(x) / (1+exp(x))
inv_logit(coef(mod1, parameter="mu"))

# Example 2
# Generating random values under some model

# A function to simulate a data set with Y ~ DBH
gendat <- function(n) {
  x1 <- runif(n)
  mu    <- inv_logit(-3 + 5 * x1)
  y <- rDBH(n=n, mu=mu)
  data.frame(y=y, x1=x1)
}

datos <- gendat(n=150)

mod2 <- NULL
mod2 <- gamlss(y~x1, family=DBH, data=datos,
               control=gamlss.control(n.cyc=500, trace=FALSE))

summary(mod2)

# Example 3
# Number of carious teeth among the four deciduous molars.
# Taken from EL-MORSHEDY (2020) page 74364.

y <- rep(0:4, times=c(64, 17, 10, 6, 3))

mod3 <- gamlss(y~1, family=DBH,
               control=gamlss.control(n.cyc=500, trace=FALSE))
```

```
# Extracting the fitted values for mu
# using the inverse link function
inv_logit <- function(x) 1/(1 + exp(-x))
inv_logit(coef(mod3, what="mu"))

# Example 4
# Counts of cysts of kidneys using steroids.
# Taken from EL-MORSHEDY (2020) page 74365.

y <- rep(0:11, times=c(65, 14, 10, 6, 4, 2, 2, 2, 1, 1, 1, 2))

mod4 <- gamlss(y~1, family=DBH,
               control=gamlss.control(n.cyc=500, trace=FALSE))

# Extracting the fitted values for mu
# using the inverse link function
inv_logit <- function(x) 1/(1 + exp(-x))
inv_logit(coef(mod4, what="mu"))
```

---

dCOMPO *The COMPO distribution*

Description

These functions define the density, distribution function, quantile function and random generation for the Conway-Maxwell-Poisson distribution with parameters μ and σ.

Usage

```
dCOMPO(x, mu, sigma, log = FALSE)

pCOMPO(q, mu, sigma, lower.tail = TRUE, log.p = FALSE)

qCOMPO(p, mu, sigma, lower.tail = TRUE, log.p = FALSE)

rCOMPO(n, mu, sigma)
```

Arguments

x, q	vector of (non-negative integer) quantiles.
mu	vector of the mu parameter.
sigma	vector of the sigma parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
p	vector of probabilities.
n	number of random values to return.

Details

The COMPO distribution with parameters μ and σ has a support 0, 1, 2, ... and mass function given by

$f(x|\mu,\sigma) = \frac{\mu^x}{(x!)^\sigma Z(\mu,\sigma)}$

with $\mu > 0$, $\sigma \geq 0$ and

$Z(\mu,\sigma) = \sum_{j=0}^{\infty} \frac{\mu^j}{(j!)^\sigma}$.

The proposed functions here are based on the functions from the COMPoissonReg package.

Value

`dCOMPO` gives the density, `pCOMPO` gives the distribution function, `qCOMPO` gives the quantile function, `rCOMPO` generates random deviates.

Author(s)

Freddy Hernandez, <fhernanb@unal.edu.co>

References

Shmueli, G., Minka, T. P., Kadane, J. B., Borle, S., & Boatwright, P. (2005). A useful distribution for fitting discrete data: revival of the Conway–Maxwell–Poisson distribution. Journal of the Royal Statistical Society Series C: Applied Statistics, 54(1), 127-142.

See Also

COMPO.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

x_max <- 20
probs1 <- dCOMPO(x=0:x_max, mu=2, sigma=0.5)
probs2 <- dCOMPO(x=0:x_max, mu=8, sigma=1.0)
probs3 <- dCOMPO(x=0:x_max, mu=15, sigma=1.5)

# To plot the first k values
plot(x=0:x_max, y=probs1, type="o", lwd=2, col="dodgerblue", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for COMPO",
     ylim=c(0, 0.30))
points(x=0:x_max, y=probs2, type="o", lwd=2, col="tomato")
points(x=0:x_max, y=probs3, type="o", lwd=2, col="green4")
legend("topright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=2, sigma=0.5",
                "mu=8, sigma=1.0",
                "mu=15, sigma=1.5"))

# Example 2
```

```
# Checking if the cumulative curves converge to 1

x_max <- 20
cumulative_probs1 <- pCOMPO(q=0:x_max, mu=2, sigma=0.5)
cumulative_probs2 <- pCOMPO(q=0:x_max, mu=8, sigma=1.0)
cumulative_probs3 <- pCOMPO(q=0:x_max, mu=15, sigma=1.5)

plot(x=0:x_max, y=cumulative_probs1, col="dodgerblue",
     type="o", las=1, ylim=c(0, 1),
     main="Cumulative probability for COMPO",
     xlab="X", ylab="Probability")
points(x=0:x_max, y=cumulative_probs2, type="o", col="tomato")
points(x=0:x_max, y=cumulative_probs3, type="o", col="green4")
legend("bottomright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=2, sigma=0.5",
                "mu=8, sigma=1.0",
                "mu=15, sigma=1.5"))

# Example 3
# Comparing the random generator output with
# the theoretical probabilities

x_max <- 50
probs1 <- dCOMPO(x=0:x_max, mu=5, sigma=0.5)
names(probs1) <- 0:x_max

x <- rCOMPO(n=1000, mu=5, sigma=0.5)
probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
mp <- barplot(height, beside = TRUE, names.arg = cn,
              col=c("dodgerblue3","firebrick3"), las=1,
              xlab="X", ylab="Proportion")
legend("topright",
       legend=c("Theoretical", "Simulated"),
       bty="n", lwd=3,
       col=c("dodgerblue3","firebrick3"), lty=1)

# Example 4
# Checking the quantile function

mu <- 3
sigma <- 1.5
p <- seq(from=0.01, to=0.99, by=0.01)
qxx <- qCOMPO(p=p, mu=mu, sigma=sigma, lower.tail=TRUE, log.p=FALSE)
plot(p, qxx, type="s", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles of COMPO(mu = 3, sigma = 1.5)")
```

---

dCOMPO2 *The COMPO2 distribution (with mu as mean)*

Description

These functions define the density, distribution function, quantile function and random generation for the Comway-Maxwell-Poisson distribution with parameters μ and σ. This parameterization was proposed by Ribeiro et al. (2020) and the main characteristic is that $E(X) = \mu$.

Usage

```
dCOMPO2(x, mu, sigma, log = FALSE)

pCOMPO2(q, mu, sigma, lower.tail = TRUE, log.p = FALSE)

qCOMPO2(p, mu, sigma, lower.tail = TRUE, log.p = FALSE)

rCOMPO2(n, mu, sigma)
```

Arguments

x, q	vector of (non-negative integer) quantiles.
mu	vector of the mu parameter.
sigma	vector of the sigma parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
p	vector of probabilities.
n	number of random values to return.

Details

The COMPO2 distribution with parameters μ and σ has a support 0, 1, 2, ... and mass function given by

$f(x|\mu,\sigma) = \left(\mu + \frac{\exp(\sigma)-1}{2\exp(\sigma)}\right)^{x\exp(\sigma)} \frac{(x!)^{\exp(\sigma)}}{Z(\mu,\sigma)}$

with $\mu > 0$, $\sigma \in \Re$ and

$Z(\mu,\sigma) = \sum_{j=0}^{\infty} \frac{\mu^j}{(j!)^\sigma}$.

The proposed functions here are based on the functions from the COMPoissonReg package.

Value

dCOMPO2 gives the density, pCOMPO2 gives the distribution function, qCOMPO2 gives the quantile function, rCOMPO2 generates random deviates.

Author(s)

Freddy Hernandez, <fhernanb@unal.edu.co>

References

Ribeiro Jr, Eduardo E., et al. "Reparametrization of COM–Poisson regression models with applications in the analysis of experimental data." Statistical Modelling 20.5 (2020): 443-466.

See Also

COMPO2.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

x_max <- 20
probs1 <- dCOMPO2(x=0:x_max, mu=2, sigma=-0.7)
probs2 <- dCOMPO2(x=0:x_max, mu=8, sigma=0)
probs3 <- dCOMPO2(x=0:x_max, mu=15, sigma=0.7)

# To plot the first k values
plot(x=0:x_max, y=probs1, type="o", lwd=2, col="dodgerblue", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for COMPO2",
     ylim=c(0, 0.30))
points(x=0:x_max, y=probs2, type="o", lwd=2, col="tomato")
points(x=0:x_max, y=probs3, type="o", lwd=2, col="green4")
legend("topright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=2, sigma=-0.7",
                "mu=8, sigma=0",
                "mu=15, sigma=0.7"))

# Example 2
# Checking if the cumulative curves converge to 1

x_max <- 20
cumulative_probs1 <- pCOMPO2(q=0:x_max, mu=2, sigma=-0.7)
cumulative_probs2 <- pCOMPO2(q=0:x_max, mu=8, sigma=0)
cumulative_probs3 <- pCOMPO2(q=0:x_max, mu=15, sigma=0.7)

plot(x=0:x_max, y=cumulative_probs1, col="dodgerblue",
     type="o", las=1, ylim=c(0, 1),
     main="Cumulative probability for COMPO2",
     xlab="X", ylab="Probability")
points(x=0:x_max, y=cumulative_probs2, type="o", col="tomato")
points(x=0:x_max, y=cumulative_probs3, type="o", col="green4")
legend("bottomright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=2, sigma=-0.7",
                "mu=8, sigma=0",
                "mu=15, sigma=0.7"))

# Example 3
# Comparing the random generator output with
# the theoretical probabilities
```

```
x_max <- 15
probs1 <- dCOMPO2(x=0:x_max, mu=5, sigma=0.5)
names(probs1) <- 0:x_max

x <- rCOMPO2(n=1000, mu=5, sigma=0.5)
probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
mp <- barplot(height, beside = TRUE, names.arg = cn,
              col=c("dodgerblue3","firebrick3"), las=1,
              xlab="X", ylab="Proportion")
legend("topright",
       legend=c("Theoretical", "Simulated"),
       bty="n", lwd=3,
       col=c("dodgerblue3","firebrick3"), lty=1)

# Example 4
# Checking the quantile function

mu <- 3
sigma <- 0.15
p <- seq(from=0.01, to=0.99, by=0.01)
qxx <- qCOMPO2(p=p, mu=mu, sigma=sigma, lower.tail=TRUE, log.p=FALSE)
plot(p, qxx, type="s", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles of COMPO2(mu = 3, sigma = 0.15)")
```

---

dDBH *The Discrete Burr Hatke distribution*

Description

These functions define the density, distribution function, quantile function and random generation for the Discrete Burr Hatke distribution with parameter μ.

Usage

```
dDBH(x, mu, log = FALSE)

pDBH(q, mu, lower.tail = TRUE, log.p = FALSE)

qDBH(p, mu = 1, lower.tail = TRUE, log.p = FALSE)

rDBH(n, mu = 1)
```

Arguments

x, q	vector of (non-negative integer) quantiles.

mu	vector of the mu parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
p	vector of probabilities.
n	number of random values to return

Details

The Discrete Burr-Hatke distribution with parameters μ has a support 0, 1, 2, ... and density given by

$f(x|\mu) = (\frac{1}{x+1} - \frac{\mu}{x+2})\mu^x$

The pmf is log-convex for all values of $0 < \mu < 1$, where $\frac{f(x+1;\mu)}{f(x;\mu)}$ is an increasing function in x for all values of the parameter μ.

Note: in this implementation we changed the original parameters λ for μ, we did it to implement this distribution within gamlss framework.

Value

dDBH gives the density, pDBH gives the distribution function, qDBH gives the quantile function, rDBH generates random deviates.

Author(s)

Valentina Hurtado Sepulveda, <vhurtados@unal.edu.co>

References

El-Morshedy, M., Eliwa, M. S., & Altun, E. (2020). Discrete Burr-Hatke distribution with properties, estimation methods and regression model. IEEE access, 8, 74359-74370.

See Also

DBH.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

plot(x=0:5, y=dDBH(x=0:5, mu=0.1),
     type="h", lwd=2, col="dodgerblue", las=1,
     ylab="P(X=x)", xlab="X", ylim=c(0, 1),
     main="Probability mu=0.1")

plot(x=0:10, y=dDBH(x=0:10, mu=0.5),
     type="h", lwd=2, col="tomato", las=1,
     ylab="P(X=x)", xlab="X", ylim=c(0, 1),
     main="Probability mu=0.5")
```

```
plot(x=0:15, y=dDBH(x=0:15, mu=0.9),
     type="h", lwd=2, col="green4", las=1,
     ylab="P(X=x)", xlab="X", ylim=c(0, 1),
     main="Probability mu=0.9")

# Example 2
# Checking if the cumulative curves converge to 1

x_max <- 15
cumulative_probs1 <- pDBH(q=0:x_max, mu=0.1)
cumulative_probs2 <- pDBH(q=0:x_max, mu=0.5)
cumulative_probs3 <- pDBH(q=0:x_max, mu=0.9)

plot(x=0:x_max, y=cumulative_probs1, col="dodgerblue",
     type="o", las=1, ylim=c(0, 1),
     main="Cumulative probability for Burr-Hatke",
     xlab="X", ylab="Probability")
points(x=0:x_max, y=cumulative_probs2, type="o", col="tomato")
points(x=0:x_max, y=cumulative_probs3, type="o", col="green4")
legend("bottomright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=0.1",
                "mu=0.5",
                "mu=0.9"))

# Example 3
# Comparing the random generator output with
# the theoretical probabilities

mu <- 0.4
x_max <- 10
probs1 <- dDBH(x=0:x_max, mu=mu)
names(probs1) <- 0:x_max

x <- rDBH(n=1000, mu=mu)
probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
mp <- barplot(height, beside = TRUE, names.arg = cn,
              col=c("dodgerblue3","firebrick3"), las=1,
              xlab="X", ylab="Proportion")
legend("topright",
       legend=c("Theoretical", "Simulated"),
       bty="n", lwd=3,
       col=c("dodgerblue3","firebrick3"), lty=1)

# Example 4
# Checking the quantile function

mu <- 0.97
p <- seq(from=0, to=1, by = 0.01)
qxx <- qDBH(p, mu, lower.tail = TRUE, log.p = FALSE)
```

```
plot(p, qxx, type="s", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles of BH(mu=0.97)")
```

---

dDGEII *Discrete generalized exponential distribution - a second type*

Description

These functions define the density, distribution function, quantile function and random generation for the Discrete generalized exponential distribution a second type with parameters μ and σ.

Usage

```
dDGEII(x, mu = 0.5, sigma = 1.5, log = FALSE)

pDGEII(q, mu = 0.5, sigma = 1.5, lower.tail = TRUE, log.p = FALSE)

rDGEII(n, mu = 0.5, sigma = 1.5)

qDGEII(p, mu = 0.5, sigma = 1.5, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x, q	vector of (non-negative integer) quantiles.
mu	vector of the mu parameter.
sigma	vector of the sigma parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
n	number of random values to return.
p	vector of probabilities.

Details

The DGEII distribution with parameters μ and σ has a support 0, 1, 2, ... and mass function given by

$f(x|\mu,\sigma) = (1-\mu^{x+1})^{\sigma} - (1-\mu^{x})^{\sigma}$

with $0 < \mu < 1$ and $\sigma > 0$. If $\sigma = 1$, the DGEII distribution reduces to the geometric distribution with success probability $1 - \mu$.

Note: in this implementation we changed the original parameters p to μ and α to σ, we did it to implement this distribution within gamlss framework.

Value

dDGEII gives the density, pDGEII gives the distribution function, qDGEII gives the quantile function, rDGEII generates random deviates.

Author(s)

Valentina Hurtado Sepulveda, <vhurtados@unal.edu.co>

References

Nekoukhou, V., Alamatsaz, M. H., & Bidram, H. (2013). Discrete generalized exponential distribution of a second type. Statistics, 47(4), 876-887.

See Also

DGEII.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

x_max <- 40
probs1 <- dDGEII(x=0:x_max, mu=0.1, sigma=5)
probs2 <- dDGEII(x=0:x_max, mu=0.5, sigma=5)
probs3 <- dDGEII(x=0:x_max, mu=0.9, sigma=5)

# To plot the first k values
plot(x=0:x_max, y=probs1, type="o", lwd=2, col="dodgerblue", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for DGEII",
     ylim=c(0, 0.60))
points(x=0:x_max, y=probs2, type="o", lwd=2, col="tomato")
points(x=0:x_max, y=probs3, type="o", lwd=2, col="green4")
legend("topright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=0.1, sigma=5",
                "mu=0.5, sigma=5",
                "mu=0.9, sigma=5"))

# Example 2
# Checking if the cumulative curves converge to 1

#plot1
x_max <- 10
plot_discrete_cdf(x=0:x_max,
                  fx=dDGEII(x=0:x_max, mu=0.3, sigma=15),
                  col="dodgerblue",
                  main="CDF for DGEII",
                  lwd=3)
legend("bottomright", legend="mu=0.3, sigma=15",
       col="dodgerblue", lty=1, lwd=2, cex=0.8)


#plot2
plot_discrete_cdf(x=0:x_max,
                  fx=dDGEII(x=0:x_max, mu=0.5, sigma=30),
                  col="tomato",
                  main="CDF for DGEII",
```

```
                  lwd=3)
legend("bottomright", legend="mu=0.5, sigma=30",
       col="tomato", lty=1, lwd=2, cex=0.8)


#plot3
plot_discrete_cdf(x=0:x_max,
                  fx=dDGEII(x=0:x_max, mu=0.5, sigma=50),
                  col="green4",
                  main="CDF for DGEII",
                  lwd=3)
legend("bottomright", legend="mu=0.5, sigma=50",
       col="green4", lty=1, lwd=2, cex=0.8)


# Example 3
# Comparing the random generator output with
# the theoretical probabilities

x_max <- 15
probs1 <- dDGEII(x=0:x_max, mu=0.5, sigma=5)
names(probs1) <- 0:x_max

x <- rDGEII(n=1000, mu=0.5, sigma=5)
probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
mp <- barplot(height, beside=TRUE, names.arg=cn,
              col=c("dodgerblue3","firebrick3"), las=1,
              xlab="X", ylab="Proportion")
legend("topright",
       legend=c("Theoretical", "Simulated"),
       bty="n", lwd=3,
       col=c("dodgerblue3","firebrick3"), lty=1)

# Example 4
# Checking the quantile function

mu <- 0.5
sigma <- 5
p <- seq(from=0, to=1, by=0.01)
qxx <- qDGEII(p=p, mu=mu, sigma=sigma, lower.tail=TRUE, log.p=FALSE)
plot(p, qxx, type="s", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles of DDGEII(mu=0.5, sigma=5)")
```

---

dDIKUM *The discrete Inverted Kumaraswamy distribution*

Description

These functions define the density, distribution function, quantile function and random generation for the discrete Inverted Kumaraswamy, DIKUM(), distribution with parameters μ and σ.

Usage

```
dDIKUM(x, mu = 1, sigma = 5, log = FALSE)

pDIKUM(q, mu = 1, sigma = 5, lower.tail = TRUE, log.p = FALSE)

rDIKUM(n, mu = 1, sigma = 5)

qDIKUM(p, mu = 1, sigma = 5, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x, q	vector of (non-negative integer) quantiles.
mu	vector of the mu parameter.
sigma	vector of the sigma parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
n	number of random values to return.
p	vector of probabilities.

Details

The discrete Inverted Kumaraswamy distribution with parameters μ and σ has a support 0, 1, 2, ... and density given by

$f(x|\mu,\sigma) = (1-(2+x)^{-\mu})^{\sigma} - (1-(1+x)^{-\mu})^{\sigma}$

with $\mu > 0$ and $\sigma > 0$.

Note: in this implementation we changed the original parameters α and β for μ and σ respectively, we did it to implement this distribution within gamlss framework.

Value

dDIKUM gives the density, pDIKUM gives the distribution function, qDIKUM gives the quantile function, rDIKUM generates random deviates.

Author(s)

Daniel Felipe Villa Rengifo, <dvilla@unal.edu.co>

References

El-Helbawy, A. A., Hegazy, M. A., Al-Dayian, G. R., & Abd EL-Kader, R. E. (2022). A discrete analog of the inverted Kumaraswamy distribution: properties and estimation with application to COVID-19 data. Pakistan Journal of Statistics and Operation Research, 18(1), 297-328.

See Also

DIKUM.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

x_max <- 30

probs1 <- dDIKUM(x=0:x_max, mu=1, sigma=5)
probs2 <- dDIKUM(x=0:x_max, mu=1, sigma=20)
probs3 <- dDIKUM(x=0:x_max, mu=1, sigma=50)

# To plot the first k values
plot(x=0:x_max, y=probs1, type="o", lwd=2, col="dodgerblue", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for Inverted Kumaraswamy Distribution",
     ylim=c(0, 0.12))
points(x=0:x_max, y=probs2, type="o", lwd=2, col="tomato")
points(x=0:x_max, y=probs3, type="o", lwd=2, col="green4")
legend("topright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=1, sigma=5",
                "mu=1, sigma=20",
                "mu=1, sigma=50"))

# Example 2
# Checking if the cumulative curves converge to 1

x_max <- 500

cumulative_probs1 <- pDIKUM(q=0:x_max, mu=1, sigma=5)
cumulative_probs2 <- pDIKUM(q=0:x_max, mu=1, sigma=20)
cumulative_probs3 <- pDIKUM(q=0:x_max, mu=1, sigma=50)

plot(x=0:x_max, y=cumulative_probs1, col="dodgerblue",
     type="o", las=1, ylim=c(0, 1),
     main="Cumulative probability for Inverted Kumaraswamy Distribution",
     xlab="X", ylab="Probability")
points(x=0:x_max, y=cumulative_probs2, type="o", col="tomato")
points(x=0:x_max, y=cumulative_probs3, type="o", col="green4")
legend("bottomright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=1, sigma=5",
                "mu=1, sigma=20",
                "mu=1, sigma=50"))

# Example 3
# Comparing the random generator output with
# the theoretical probabilities

x_max <- 20
probs1 <- dDIKUM(x=0:x_max, mu=3, sigma=20)
names(probs1) <- 0:x_max
```

```
x <- rDIKUM(n=1000, mu=3, sigma=20)
probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
nombres <- cn
mp <- barplot(height, beside = TRUE, names.arg = nombres,
              col=c("dodgerblue3","firebrick3"), las=1,
              xlab="X", ylab="Proportion")
legend("topright",
       legend=c("Theoretical", "Simulated"),
       bty="n", lwd=3,
       col=c("dodgerblue3","firebrick3"), lty=1)

# Example 4
# Checking the quantile function

mu <- 1
sigma <- 5
p <- seq(from=0.01, to=0.99, by=0.1)
qxx <- qDIKUM(p=p, mu=mu, sigma=sigma, lower.tail=TRUE, log.p=FALSE)
plot(p, qxx, type="s", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles of HP(mu = sigma = 3)")
```

---

dDLD *The Discrete Lindley distribution*

Description

These functions define the density, distribution function, quantile function and random generation for the Discrete Lindley distribution with parameter μ.

Usage

```
dDLD(x, mu, log = FALSE)

pDLD(q, mu, lower.tail = TRUE, log.p = FALSE)

qDLD(p, mu, lower.tail = TRUE, log.p = FALSE)

rDLD(n, mu = 0.5)
```

Arguments

x, q	vector of (non-negative integer) quantiles.
mu	vector of positive values of this parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).

lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
p	vector of probabilities.
n	number of random values to return.

Details

The Discrete Lindley distribution with parameters μ has a support 0, 1, 2, ... and density given by

$f(x|\mu) = \frac{e^{-\mu x}}{1+\mu}\,[\mu(1-2e^{-\mu}) + (1-e^{-\mu})(1+\mu x)]$

Note: in this implementation we changed the original parameters θ for μ, we did it to implement this distribution within gamlss framework.

Value

dDLD gives the density, pDLD gives the distribution function, qDLD gives the quantile function, rDLD generates random deviates.

Author(s)

Yojan Andrés Alcaraz Pérez, <yalcaraz@unal.edu.co>

References

Bakouch, H. S., Jazi, M. A., & Nadarajah, S. (2014). A new discrete distribution. Statistics, 48(1), 200-240.

See Also

DLD.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

plot(x=0:25, y=dDLD(x=0:25, mu=0.2),
     type="h", lwd=2, col="dodgerblue", las=1,
     ylab="P(X=x)", xlab="X", ylim=c(0, 0.1),
     main="Probability mu=0.2")

plot(x=0:15, y=dDLD(x=0:15, mu=0.5),
     type="h", lwd=2, col="tomato", las=1,
     ylab="P(X=x)", xlab="X", ylim=c(0, 0.25),
     main="Probability mu=0.5")

plot(x=0:8, y=dDLD(x=0:8, mu=1),
     type="h", lwd=2, col="green4", las=1,
     ylab="P(X=x)", xlab="X", ylim=c(0, 0.5),
     main="Probability mu=1")

plot(x=0:5, y=dDLD(x=0:5, mu=2),
```

```
     type="h", lwd=2, col="red", las=1,
     ylab="P(X=x)", xlab="X", ylim=c(0, 1),
     main="Probability mu=2")

# Example 2
# Checking if the cumulative curves converge to 1

x_max <- 10
cumulative_probs1 <- pDLD(q=0:x_max, mu=0.2)
cumulative_probs2 <- pDLD(q=0:x_max, mu=0.5)
cumulative_probs3 <- pDLD(q=0:x_max, mu=1)
cumulative_probs4 <- pDLD(q=0:x_max, mu=2)

plot(x=0:x_max, y=cumulative_probs1, col="dodgerblue",
     type="o", las=1, ylim=c(0, 1),
     main="Cumulative probability for Lindley",
     xlab="X", ylab="Probability")
points(x=0:x_max, y=cumulative_probs2, type="o", col="tomato")
points(x=0:x_max, y=cumulative_probs3, type="o", col="green4")
points(x=0:x_max, y=cumulative_probs4, type="o", col="magenta")
legend("bottomright",
       col=c("dodgerblue", "tomato", "green4", "magenta"), lwd=3,
       legend=c("mu=0.2",
                "mu=0.5",
                "mu=1",
                "mu=2"))

# Example 3
# Comparing the random generator output with
# the theoretical probabilities

mu <- 0.6
x <- rDLD(n = 1000, mu = mu)
x_max <- max(x)
probs1 <- dDLD(x = 0:x_max, mu = mu)
names(probs1) <- 0:x_max

probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])

mp <- barplot(height, beside = TRUE, names.arg = cn,
              col=c("dodgerblue3","firebrick3"), las=1,
              xlab="X", ylab="Proportion")
legend("topright",
       legend=c("Theoretical", "Simulated"),
       bty="n", lwd=3,
       col=c("dodgerblue3","firebrick3"), lty=1)

# Example 4
# Checking the quantile function
```

```
mu <- 0.9
p <- seq(from=0, to=1, by=0.01)
qxx <- qDLD(p, mu, lower.tail = TRUE, log.p = FALSE)
plot(p, qxx, type="S", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles of DL(mu=0.9)")
```

---

dDMOLBE *The DMOLBE distribution*

Description

These functions define the density, distribution function, quantile function and random generation for the Discrete Marshall–Olkin Length Biased Exponential DMOLBE distribution with parameters μ and σ.

Usage

```
dDMOLBE(x, mu = 1, sigma = 1, log = FALSE)

pDMOLBE(q, mu = 1, sigma = 1, lower.tail = TRUE, log.p = FALSE)

rDMOLBE(n, mu = 1, sigma = 1)

qDMOLBE(p, mu = 1, sigma = 1, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x, q	vector of (non-negative integer) quantiles.
mu	vector of the mu parameter.
sigma	vector of the sigma parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
n	number of random values to return.
p	vector of probabilities.

Details

The DMOLBE distribution with parameters μ and σ has a support 0, 1, 2, ... and mass function given by

$f(x|\mu,\sigma) = \frac{\sigma((1+x/\mu)\exp(-x/\mu)-(1+(x+1)/\mu)\exp(-(x+1)/\mu))}{(1-(1-\sigma)(1+x/\mu)\exp(-x/\mu))((1-(1-\sigma)(1+(x+1)/\mu)\exp(-(x+1)/\mu))}$

with $\mu > 0$ and $\sigma > 0$

Value

dDMOLBE gives the density, pDMOLBE gives the distribution function, qDMOLBE gives the quantile function, rDMOLBE generates random deviates.

Author(s)

Olga Usuga, <olga.usuga@udea.edu.co>

References

Aljohani, H. M., Ahsan-ul-Haq, M., Zafar, J., Almetwally, E. M., Alghamdi, A. S., Hussam, E., & Muse, A. H. (2023). Analysis of Covid-19 data using discrete Marshall–Olkinin length biased exponential: Bayesian and frequentist approach. Scientific Reports, 13(1), 12243.

See Also

DMOLBE.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

x_max <- 20
probs1 <- dDMOLBE(x=0:x_max, mu=0.5, sigma=0.5)
probs2 <- dDMOLBE(x=0:x_max, mu=5, sigma=0.5)
probs3 <- dDMOLBE(x=0:x_max, mu=1, sigma=2)

# To plot the first k values
plot(x=0:x_max, y=probs1, type="o", lwd=2, col="dodgerblue", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for DMOLBE",
     ylim=c(0, 0.80))
points(x=0:x_max, y=probs2, type="o", lwd=2, col="tomato")
points(x=0:x_max, y=probs3, type="o", lwd=2, col="green4")
legend("topright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=0.5, sigma=0.5",
                "mu=5, sigma=0.5",
                "mu=1, sigma=2"))

# Example 2
# Checking if the cumulative curves converge to 1

x_max <- 20
cumulative_probs1 <- pDMOLBE(q=0:x_max, mu=0.5, sigma=0.5)
cumulative_probs2 <- pDMOLBE(q=0:x_max, mu=5, sigma=0.5)
cumulative_probs3 <- pDMOLBE(q=0:x_max, mu=1, sigma=2)

plot(x=0:x_max, y=cumulative_probs1, col="dodgerblue",
     type="o", las=1, ylim=c(0, 1),
     main="Cumulative probability for DMOLBE",
     xlab="X", ylab="Probability")
points(x=0:x_max, y=cumulative_probs2, type="o", col="tomato")
points(x=0:x_max, y=cumulative_probs3, type="o", col="green4")
legend("bottomright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=0.5, sigma=0.5",
                "mu=5, sigma=0.5",
                "mu=1, sigma=2"))
```

```
# Example 3
# Comparing the random generator output with
# the theoretical probabilities

x_max <- 50
probs1 <- dDMOLBE(x=0:x_max, mu=5, sigma=0.5)
names(probs1) <- 0:x_max

x <- rDMOLBE(n=1000, mu=5, sigma=0.5)
probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
mp <- barplot(height, beside = TRUE, names.arg = cn,
              col=c("dodgerblue3","firebrick3"), las=1,
              xlab="X", ylab="Proportion")
legend("topright",
       legend=c("Theoretical", "Simulated"),
       bty="n", lwd=3,
       col=c("dodgerblue3","firebrick3"), lty=1)

# Example 4
# Checking the quantile function

mu <- 3
sigma <-3
p <- seq(from=0, to=1, by=0.01)
qxx <- qDMOLBE(p=p, mu=mu, sigma=sigma, lower.tail=TRUE, log.p=FALSE)
plot(p, qxx, type="s", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles of DMOLBE(mu = 3, sigma = 3)")
```

---

dDPERKS *The Discrete Perks distribution*

Description

These functions define the density, distribution function, quantile function and random generation for the Discrete Perks, DPERKS(), distribution with parameters μ and σ.

Usage

```
dDPERKS(x, mu = 0.5, sigma = 0.5, log = FALSE)

pDPERKS(q, mu = 0.5, sigma = 0.5, lower.tail = TRUE, log.p = FALSE)

rDPERKS(n, mu = 0.5, sigma = 0.5)

qDPERKS(p, mu = 0.5, sigma = 0.5, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x, q	vector of (non-negative integer) quantiles.
mu	vector of the mu parameter.
sigma	vector of the sigma parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
n	number of random values to return.
p	vector of probabilities.

Details

The discrete Perks distribution with parameters $\mu > 0$ and $\sigma > 0$ has a support 0, 1, 2, ... and density given by

$f(x|\mu,\sigma) = \frac{\mu(1+\mu)(e^{\sigma}-1)e^{\sigma x}}{(1+\mu e^{\sigma x})(1+\mu e^{\sigma(x+1)})}$

Note: in this implementation we changed the original parameters λ for μ and β for σ, we did it to implement this distribution within gamlss framework.

Value

dDPERKS gives the density, pDPERKS gives the distribution function, qDPERKS gives the quantile function, rDPERKS generates random deviates.

Author(s)

Veronica Seguro Varela, <vseguro@unal.edu.co>

References

Tyagi, A., Choudhary, N., & Singh, B. (2020). A new discrete distribution: Theory and applications to discrete failure lifetime and count data. J. Appl. Probab. Statist, 15, 117-143.

See Also

DPERKS.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

x_max <- 25
probs1 <- dDPERKS(x=0:x_max, mu=0.001, sigma=0.52)
probs2 <- dDPERKS(x=0:x_max, mu=0.001, sigma=0.85)
probs3 <- dDPERKS(x=0:x_max, mu=0.001, sigma=1.5)

# To plot the first k values
plot(x=0:x_max, y=probs1, type="o", lwd=2, col="green4", las=1,
```

```
     ylab="P(X=x)", xlab="X", main="Probability for DPERKS",
     ylim=c(0, 0.40))
points(x=0:x_max, y=probs2, type="o", lwd=2, col="tomato")
points(x=0:x_max, y=probs3, type="o", lwd=2, col="black")
legend("topright", col=c("green4", "tomato", "black"), lwd=3,
       legend=c("mu=0.001, sigma=0.52 ",
                "mu=0.001, sigma=0.85",
                "mu=0.001, sigma=1.5"))

# Example 2
# Checking if the cumulative curves converge to 1

x_max <- 25
cumulative_probs1 <- pDPERKS(q=0:x_max, mu=0.001, sigma=0.52)
cumulative_probs2 <- pDPERKS(q=0:x_max, mu=0.001, sigma=0.85)
cumulative_probs3 <- pDPERKS(q=0:x_max, mu=0.001, sigma=1.5)

plot(x=0:x_max, y=cumulative_probs1, col="green4",
     type="o", las=1, ylim=c(0, 1),
     main="Cumulative probability for DPERKS",
     xlab="X", ylab="Probability")
points(x=0:x_max, y=cumulative_probs2, type="o", col="tomato")
points(x=0:x_max, y=cumulative_probs3, type="o", col="black")
legend("bottomright", col=c("green4", "tomato", "black"), lwd=3,
       legend=c("mu=0.001, sigma=0.52 ",
                "mu=0.001, sigma=0.85",
                "mu=0.001, sigma=1.5"))

# Example 3
# Comparing the random generator output with the theoretical probabilities

x_max <- 20
mu <- 2.5
sigma <- 0.4
probs1 <- dDPERKS(x=0:x_max, mu=mu, sigma=sigma)
names(probs1) <- 0:x_max

x <- rDPERKS(n=10000, mu=mu, sigma=sigma)
probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
nombres <- cn
mp <- barplot(height, beside = TRUE, names.arg = nombres,
              col=c("dodgerblue3","firebrick3"), las=1,
              xlab="X", ylab="Proportion")
legend("topright",
       legend=c("Theoretical", "Simulated"),
       bty="n", lwd=3,
       col=c("dodgerblue3","firebrick3"), lty=1)

# Example 4
# Checking the quantile function
```

```
mu <- 0.2
sigma <- 0.2
p <- seq(from=0, to=1, by=0.01)
qxx <- qDPERKS(p=p, mu=mu, sigma=sigma, lower.tail=TRUE, log.p=FALSE)
plot(p, qxx, type="s", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles of DPERKS(mu = sigma = 0.2)")
```

---

dDsPA *The DsPA distribution*

Description

These functions define the density, distribution function, quantile function and random generation for the discrete power-Ailamujia distribution with parameters μ and σ.

Usage

```
dDsPA(x, mu, sigma, log = FALSE)

pDsPA(q, mu, sigma, lower.tail = TRUE, log.p = FALSE)

qDsPA(p, mu = 1, sigma = 1, lower.tail = TRUE, log.p = FALSE)

rDsPA(n, mu, sigma)
```

Arguments

x, q	vector of (non-negative integer) quantiles.
mu	vector of the mu parameter.
sigma	vector of the sigma parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
p	vector of probabilities.
n	number of random values to return.

Details

The DsPA distribution with parameters μ and σ has a support 0, 1, 2, ...

Note:in this implementation we changed the original parameters β and λ for μ and σ respectively, we did it to implement this distribution within gamlss framework.

Value

dDsPA gives the density, pDsPA gives the distribution function, qDsPA gives the quantile function.

Author(s)

Maria Camila Mena Romana, <mamenar@unal.edu.co>

References

Alghamdi, A. S., Ahsan-ul-Haq, M., Babar, A., Aljohani, H. M., Afify, A. Z., & Cell, Q. E. (2022). The discrete power-Ailamujia distribution: properties, inference, and applications. AIMS Math, 7(5), 8344-8360.

See Also

DsPA.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

x_max <- 30
probs1 <- dDsPA(x=0:x_max, mu=1.2, sigma=0.5)
probs2 <- dDsPA(x=0:x_max, mu=1.2, sigma=0.7)
probs3 <- dDsPA(x=0:x_max, mu=1.2, sigma=0.9)

# To plot the first k values
plot(x=0:x_max, y=probs1, type="o", lwd=2, col="dodgerblue", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for DsPA",
     ylim=c(0, 0.40))
points(x=0:x_max, y=probs2, type="o", lwd=2, col="tomato")
points(x=0:x_max, y=probs3, type="o", lwd=2, col="green4")
legend("topright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=1.2, sigma=0.5",
                "mu=1.2, sigma=0.7",
                "mu=1.2, sigma=0.9"))

# Example 2
# Checking if the cumulative curves converge to 1

x_max <- 15
cumulative_probs1 <- pDsPA(q=0:x_max, mu=1.2, sigma=0.5)
cumulative_probs2 <- pDsPA(q=0:x_max, mu=1.2, sigma=0.7)
cumulative_probs3 <- pDsPA(q=0:x_max, mu=1.2, sigma=0.9)

plot(x=0:x_max, y=cumulative_probs1, col="dodgerblue",
     type="o", las=1, ylim=c(0, 1),
     main="Cumulative probability for DsPA",
     xlab="X", ylab="Probability")
points(x=0:x_max, y=cumulative_probs2, type="o", col="tomato")
points(x=0:x_max, y=cumulative_probs3, type="o", col="green4")
legend("bottomright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=1.2, sigma=0.5",
                "mu=1.2, sigma=0.7",
                "mu=1.2, sigma=0.9"))
```

```
# Example 3
# Comparing the random generator output with
# the theoretical probabilities

x_max <- 50
probs1 <- dDsPA(x=0:x_max, mu=1.2, sigma=0.9)
names(probs1) <- 0:x_max

x <- rDsPA(n=1000, mu=1.2, sigma=0.9)
probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
mp <- barplot(height, beside=TRUE, names.arg=cn,
              col=c("dodgerblue3","firebrick3"), las=1,
              xlab="X", ylab="Proportion")
legend("topright",
       legend=c("Theoretical", "Simulated"),
       bty="n", lwd=3,
       col=c("dodgerblue3","firebrick3"), lty=1)

# Example 4
# Checking the quantile function

mu <- 1.2
sigma <- 0.9
p <- seq(from=0, to=1, by=0.01)
qxx <- qDsPA(p=p, mu=mu, sigma=sigma,
                lower.tail=TRUE, log.p=FALSE)
plot(p, qxx, type="s", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles of DsPA(mu=1.2, sigma=0.9)")
```

DGEII *The DGEII distribution*

Description

The function DGEII() defines the Discrete generalized exponential distribution of a Second type a two parameter distribution, for a gamlss.family object to be used in GAMLSS fitting using the function gamlss().

Usage

```
DGEII(mu.link = "logit", sigma.link = "log")
```

Arguments

`mu.link`	defines the mu.link, with "logit" link as the default for the mu parameter. Other links are "probit" and "cloglog"'(complementary log-log).
`sigma.link`	defines the sigma.link, with "log" link as the default for the sigma.

Details

The DGEII distribution with parameters μ and σ has a support 0, 1, 2, ... and mass function given by

$f(x|\mu,\sigma) = (1-\mu^{x+1})^{\sigma} - (1-\mu^{x})^{\sigma}$

with $0 < \mu < 1$ and $\sigma > 0$. If $\sigma = 1$, the DGEII distribution reduces to the geometric distribution with success probability $1 - \mu$.

Note: in this implementation we changed the original parameters p to μ and α to σ, we did it to implement this distribution within gamlss framework.

Value

Returns a `gamlss.family` object which can be used to fit a DGEII distribution in the `gamlss()` function.

Author(s)

Valentina Hurtado Sepúlveda, <vhurtados@unal.edu.co>

References

Nekoukhou, V., Alamatsaz, M. H., & Bidram, H. (2013). Discrete generalized exponential distribution of a second type. Statistics, 47(4), 876-887.

See Also

dDGEII.

Examples

```
# Example 1
# Generating some random values with
# known mu and sigma

y <- rDGEII(n=100, mu=0.75, sigma=0.5)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y~1, family=DGEII,
               control=gamlss.control(n.cyc=500, trace=TRUE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
inv_logit <- function(x) 1/(1 + exp(-x))
```

```
inv_logit(coef(mod1, what="mu"))
exp(coef(mod1, what="sigma"))

# Example 2
# Generating random values under some model

# A function to simulate a data set with Y ~ DGEII
gendat <- function(n) {
  x1 <- runif(n)
  x2 <- runif(n)
  mu    <- inv_logit(1.7 - 2.8*x1)
  sigma <- exp(0.73 + 1*x2)
  y <- rDGEII(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1, x2=x2)
}

datos <- gendat(n=100)

mod2 <- gamlss(y~x1, sigma.fo=~x2, family=DGEII, data=datos,
               control=gamlss.control(n.cyc=500, trace=TRUE))

summary(mod2)

# Example 3
# Number of accidents to 647 women working on H. E. Shells
# for 5 weeks. Taken from
# Nekoukhou V, Alamatsaz MH, Bidram H (2013) page 886.

y <- rep(x=0:5, times=c(447, 132, 42, 21, 3, 2))

mod3 <- gamlss(y~1, family=DGEII,
               control=gamlss.control(n.cyc=500, trace=TRUE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
inv_logit <- function(x) 1/(1 + exp(-x))
inv_logit(coef(mod3, what="mu"))
exp(coef(mod3, what="sigma"))
```

---

dGGEO *The GGEO distribution*

Description

These functions define the density, distribution function, quantile function and random generation for the Generalized Geometric distribution with parameters μ and σ.

Usage

```
dGGEO(x, mu = 0.5, sigma = 1, log = FALSE)

pGGEO(q, mu = 0.5, sigma = 1, lower.tail = TRUE, log.p = FALSE)

rGGEO(n, mu = 0.5, sigma = 1)

qGGEO(p, mu = 0.5, sigma = 1, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x, q	vector of (non-negative integer) quantiles.
mu	vector of the mu parameter.
sigma	vector of the sigma parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
n	number of random values to return.
p	vector of probabilities.

Details

The GGEO distribution with parameters μ and σ has a support 0, 1, 2, ... and mass function given by

$f(x|\mu,\sigma) = \frac{\sigma\mu^x(1-\mu)}{(1-(1-\sigma)\mu^{x+1})(1-(1-\sigma)\mu^x)}$

with $0 < \mu < 1$ and $\sigma > 0$. If $\sigma = 1$, the GGEO distribution reduces to the geometric distribution with success probability $1 - \mu$.

Note: in this implementation we changed the original parameters θ for μ and α for σ, we did it to implement this distribution within gamlss framework.

Value

dGGEO gives the density, pGGEO gives the distribution function, qGGEO gives the quantile function, rGGEO generates random deviates.

Author(s)

Valentina Hurtado Sepulveda, <vhurtados@unal.edu.co>

References

Gómez-Déniz, E. (2010). Another generalization of the geometric distribution. Test, 19, 399-415.

See Also

GGEO.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

x_max <- 80
probs1 <- dGGEO(x=0:x_max, mu=0.5, sigma=10)
probs2 <- dGGEO(x=0:x_max, mu=0.7, sigma=30)
probs3 <- dGGEO(x=0:x_max, mu=0.9, sigma=50)

# To plot the first k values
plot(x=0:x_max, y=probs1, type="o", lwd=2, col="dodgerblue", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for GGEO",
     ylim=c(0, 0.20))
points(x=0:x_max, y=probs2, type="o", lwd=2, col="tomato")
points(x=0:x_max, y=probs3, type="o", lwd=2, col="green4")
legend("topright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=0.5, sigma=10",
                "mu=0.7, sigma=30",
                "mu=0.9, sigma=50"))

# Example 2
# Checking if the cumulative curves converge to 1

x_max <- 10
plot_discrete_cdf(x=0:x_max,
                  fx=dGGEO(x=0:x_max, mu=0.3, sigma=15),
                  col="dodgerblue",
                  main="CDF for GGEO",
                  lwd= 3)
legend("bottomright", legend="mu=0.3, sigma=15", col="dodgerblue",
       lty=1, lwd=2, cex=0.8)

plot_discrete_cdf(x=0:x_max,
                  fx=dGGEO(x=0:x_max, mu=0.5, sigma=30),
                  col="tomato",
                  main="CDF for GGEO",
                  lwd=3)
legend("bottomright", legend="mu=0.5, sigma=30",
       col="tomato", lty=1, lwd=2, cex=0.8)

plot_discrete_cdf(x=0:x_max,
                  fx=dGGEO(x=0:x_max, mu=0.5, sigma=50),
                  col="green4",
                  main="CDF for GGEO",
                  lwd=3)
legend("bottomright", legend="mu=0.5, sigma=50",
       col="green4", lty=1, lwd=2, cex=0.8)

# Example 3
# Comparing the random generator output with
# the theoretical probabilities
```

```
x_max <- 15
probs1 <- dGGEO(x=0:x_max, mu=0.5, sigma=5)
names(probs1) <- 0:x_max

x <- rGGEO(n=1000, mu=0.5, sigma=5)
probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
mp <- barplot(height, beside=TRUE, names.arg=cn,
              col=c("dodgerblue3", "firebrick3"), las=1,
              xlab="X", ylab="Proportion")
legend("topright",
       legend=c("Theoretical", "Simulated"),
       bty="n", lwd=3,
       col=c("dodgerblue3","firebrick3"), lty=1)

# Example 4
# Checking the quantile function

mu <- 0.5
sigma <- 5
p <- seq(from=0, to=1, by=0.01)
qxx <- qGGEO(p=p, mu=mu, sigma=sigma, lower.tail=TRUE, log.p=FALSE)
plot(p, qxx, type="s", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles of GGEO(mu=0.5, sigma=0.5)")
```

---

dHYPERPO *The hyper-Poisson distribution*

Description

These functions define the density, distribution function, quantile function and random generation for the hyper-Poisson, HYPERPO(), distribution with parameters μ and σ.

Usage

```
dHYPERPO(x, mu = 1, sigma = 1, log = FALSE)

pHYPERPO(q, mu = 1, sigma = 1, lower.tail = TRUE, log.p = FALSE)

rHYPERPO(n, mu = 1, sigma = 1)

qHYPERPO(p, mu = 1, sigma = 1, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x, q	vector of (non-negative integer) quantiles.

mu	vector of the mu parameter.
sigma	vector of the sigma parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
n	number of random values to return.
p	vector of probabilities.

Details

The hyper-Poisson distribution with parameters μ and σ has a support 0, 1, 2, ... and density given by

$f(x|\mu,\sigma) = \frac{\mu^x}{{}_1F_1(1;\mu;\sigma)} \frac{\Gamma(\sigma)}{\Gamma(x+\sigma)}$

where the function ${}_1F_1(a;c;z)$ is defined as

${}_1F_1(a;c;z) = \sum_{r=0}^{\infty} \frac{(a)_r}{(c)_r} \frac{z^r}{r!}$

and $(a)_r = \frac{\gamma(a+r)}{\gamma(a)}$ for $a > 0$ and r positive integer.

Note: in this implementation we changed the original parameters λ and γ for μ and σ respectively, we did it to implement this distribution within gamlss framework.

Value

dHYPERPO gives the density, pHYPERPO gives the distribution function, qHYPERPO gives the quantile function, rHYPERPO generates random deviates.

Author(s)

Freddy Hernandez, <fhernanb@unal.edu.co>

References

Sáez-Castillo, A. J., & Conde-Sánchez, A. (2013). A hyper-Poisson regression model for overdispersed and underdispersed count data. Computational Statistics & Data Analysis, 61, 148-157.

See Also

HYPERPO.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

x_max <- 30
probs1 <- dHYPERPO(x=0:x_max, mu=5, sigma=0.1)
probs2 <- dHYPERPO(x=0:x_max, mu=5, sigma=1.0)
probs3 <- dHYPERPO(x=0:x_max, mu=5, sigma=1.8)

# To plot the first k values
```

```
plot(x=0:x_max, y=probs1, type="o", lwd=2, col="dodgerblue", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for hyper-Poisson",
     ylim=c(0, 0.20))
points(x=0:x_max, y=probs2, type="o", lwd=2, col="tomato")
points(x=0:x_max, y=probs3, type="o", lwd=2, col="green4")
legend("topright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=5, sigma=0.1",
                "mu=5, sigma=1.0",
                "mu=5, sigma=1.8"))

# Example 2
# Checking if the cumulative curves converge to 1

x_max <- 15
cumulative_probs1 <- pHYPERPO(q=0:x_max, mu=5, sigma=0.1)
cumulative_probs2 <- pHYPERPO(q=0:x_max, mu=5, sigma=1.0)
cumulative_probs3 <- pHYPERPO(q=0:x_max, mu=5, sigma=1.8)

plot(x=0:x_max, y=cumulative_probs1, col="dodgerblue",
     type="o", las=1, ylim=c(0, 1),
     main="Cumulative probability for hyper-Poisson",
     xlab="X", ylab="Probability")
points(x=0:x_max, y=cumulative_probs2, type="o", col="tomato")
points(x=0:x_max, y=cumulative_probs3, type="o", col="green4")
legend("bottomright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=5, sigma=0.1",
                "mu=5, sigma=1.0",
                "mu=5, sigma=1.8"))

# Example 3
# Comparing the random generator output with
# the theoretical probabilities

x_max <- 15
probs1 <- dHYPERPO(x=0:x_max, mu=3, sigma=1.1)
names(probs1) <- 0:x_max

x <- rHYPERPO(n=1000, mu=3, sigma=1.1)
probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
nombres <- cn
mp <- barplot(height, beside = TRUE, names.arg = nombres,
              col=c("dodgerblue3","firebrick3"), las=1,
              xlab="X", ylab="Proportion")
legend("topright",
       legend=c("Theoretical", "Simulated"),
       bty="n", lwd=3,
       col=c("dodgerblue3","firebrick3"), lty=1)

# Example 4
# Checking the quantile function
```

```
mu <- 3
sigma <-3
p <- seq(from=0, to=1, by=0.01)
qxx <- qHYPERPO(p=p, mu=mu, sigma=sigma,
                lower.tail=TRUE, log.p=FALSE)
plot(p, qxx, type="s", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles of HP(mu=3, sigma=3)")
```

---

dHYPERPO2 *The hyper-Poisson distribution (with mu as mean)*

Description

These functions define the density, distribution function, quantile function and random generation for the hyper-Poisson in the second parameterization with parameters μ (as mean) and σ as the dispersion parameter.

Usage

```
dHYPERPO2(x, mu = 1, sigma = 1, log = FALSE)

pHYPERPO2(q, mu = 1, sigma = 1, lower.tail = TRUE, log.p = FALSE)

rHYPERPO2(n, mu = 1, sigma = 1)

qHYPERPO2(p, mu = 1, sigma = 1, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x, q	vector of (non-negative integer) quantiles.
mu	vector of positive values of this parameter.
sigma	vector of positive values of this parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
n	number of random values to return
p	vector of probabilities.

Details

The hyper-Poisson distribution with parameters μ and σ has a support 0, 1, 2, ...

Note: in this implementation the parameter μ is the mean of the distribution and σ corresponds to the dispersion parameter. If you fit a model with this parameterization, the time will increase because an internal procedure to convert μ to λ parameter.

Value

dHYPERPO2 gives the density, pHYPERPO2 gives the distribution function, qHYPERPO2 gives the quantile function, rHYPERPO2 generates random deviates.

Author(s)

Freddy Hernandez, <fhernanb@unal.edu.co>

References

Sáez-Castillo, A. J., & Conde-Sánchez, A. (2013). A hyper-Poisson regression model for overdispersed and underdispersed count data. Computational Statistics & Data Analysis, 61, 148-157.

See Also

HYPERPO2, HYPERPO.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

x_max <- 30
probs1 <- dHYPERPO2(x=0:x_max, sigma=0.01, mu=3)
probs2 <- dHYPERPO2(x=0:x_max, sigma=0.50, mu=5)
probs3 <- dHYPERPO2(x=0:x_max, sigma=1.00, mu=7)
# To plot the first k values
plot(x=0:x_max, y=probs1, type="o", lwd=2, col="dodgerblue", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for hyper-Poisson",
     ylim=c(0, 0.30))
points(x=0:x_max, y=probs2, type="o", lwd=2, col="tomato")
points(x=0:x_max, y=probs3, type="o", lwd=2, col="green4")
legend("topright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("sigma=0.01, mu=3",
                "sigma=0.50, mu=5",
                "sigma=1.00, mu=7"))

# Example 2
# Checking if the cumulative curves converge to 1

x_max <- 15
cumulative_probs1 <- pHYPERPO2(q=0:x_max, mu=1, sigma=1.5)
cumulative_probs2 <- pHYPERPO2(q=0:x_max, mu=3, sigma=1.5)
cumulative_probs3 <- pHYPERPO2(q=0:x_max, mu=5, sigma=1.5)

plot(x=0:x_max, y=cumulative_probs1, col="dodgerblue",
     type="o", las=1, ylim=c(0, 1),
     main="Cumulative probability for hyper-Poisson",
     xlab="X", ylab="Probability")
points(x=0:x_max, y=cumulative_probs2, type="o", col="tomato")
points(x=0:x_max, y=cumulative_probs3, type="o", col="green4")
legend("bottomright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
```

```
       legend=c("sigma=1.5, mu=1",
                "sigma=1.5, mu=3",
                "sigma=1.5, mu=5"))

# Example 3
# Comparing the random generator output with
# the theoretical probabilities

x_max <- 15
probs1 <- dHYPERPO2(x=0:x_max, mu=3, sigma=1.1)
names(probs1) <- 0:x_max

x <- rHYPERPO2(n=1000, mu=3, sigma=1.1)
probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
nombres <- cn
mp <- barplot(height, beside = TRUE, names.arg = nombres,
              col=c('dodgerblue3','firebrick3'), las=1,
              xlab='X', ylab='Proportion')
legend('topright',
       legend=c('Theoretical', 'Simulated'),
       bty='n', lwd=3,
       col=c('dodgerblue3','firebrick3'), lty=1)

# Example 4
# Checking the quantile function

mu <- 3
sigma <-3
p <- seq(from=0, to=1, by=0.01)
qxx <- qHYPERPO2(p=p, mu=mu, sigma=sigma, lower.tail=TRUE, log.p=FALSE)
plot(p, qxx, type="s", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles of HP2(mu = sigma = 3)")
```

DIKUM *The discrete Inverted Kumaraswamy family*

Description

The function `DIKUM()` defines the discrete Inverted Kumaraswamy distribution a two parameter distribution, for a `gamlss.family` object to be used in GAMLSS fitting using the function `gamlss()`.

Usage

```
DIKUM(mu.link = "log", sigma.link = "log")
```

Arguments

mu.link	defines the mu.link, with "log" link as the default for the mu parameter.
sigma.link	defines the sigma.link, with "log" link as the default for the sigma.

Details

The discrete Inverted Kumaraswamy distribution with parameters μ and σ has a support 0, 1, 2, ... and density given by

$f(x|\mu,\sigma) = (1-(2+x)^{-\mu})^{\sigma} - (1-(1+x)^{-\mu})^{\sigma}$

with $\mu > 0$ and $\sigma > 0$.

Note: in this implementation we changed the original parameters α and β for μ and σ respectively, we did it to implement this distribution within gamlss framework.

Value

Returns a gamlss.family object which can be used to fit a discrete Inverted Kumaraswamy distribution in the gamlss() function.

Author(s)

Daniel Felipe Villa Rengifo, <dvilla@unal.edu.co>

References

El-Helbawy, A. A., Hegazy, M. A., Al-Dayian, G. R., & Abd EL-Kader, R. E. (2022). A discrete analog of the inverted Kumaraswamy distribution: properties and estimation with application to COVID-19 data. Pakistan Journal of Statistics and Operation Research, 18(1), 297-328.

See Also

dDIKUM.

Examples

```
# Example 1
# Generating some random values with
# known mu and sigma
set.seed(150)
y <- rDIKUM(1000, mu=1, sigma=5)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y ~ 1, sigma.fo = ~1, family=DIKUM,
               control = gamlss.control(n.cyc=500, trace=FALSE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
exp(coef(mod1, what="mu"))
exp(coef(mod1, what="sigma"))
```

```
# Example 2
# Generating random values under some model

library(gamlss)

# A function to simulate a data set with Y ~ DIKUM
gendat <- function(n) {
  x1 <- runif(n, min=0.4, max=0.6)
  x2 <- runif(n, min=0.4, max=0.6)
  mu    <- exp(1.21 - 3 * x1) # 0.75 approximately
  sigma <- exp(1.26 - 2 * x2) # 1.30 approximately
  y <- rDIKUM(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1, x2=x2)
}

dat <- gendat(n=1000)

# Fitting the model
mod2 <- gamlss(y ~ x1, sigma.fo = ~x2, family = "DIKUM", data=dat,
               control=gamlss.control(n.cyc=500, trace=FALSE))

summary(mod2)
```

DLD *The Discrete Lindley family*

Description

The function `DLD()` defines the Discrete Lindley distribution a single parameter distribution, for a `gamlss.family` object to be used in GAMLSS fitting using the function `gamlss()`.

Usage

```
DLD(mu.link = "log")
```

Arguments

`mu.link`	defines the mu.link, with "log" link as the default for the mu parameter.

Details

The Discrete Lindley distribution with parameters $\mu > 0$ has a support 0, 1, 2, ... and density given by

$f(x|\mu) = \frac{e^{-\mu x}}{1+\mu}(\mu(1-2e^{-\mu}) + (1-e^{-\mu})(1+\mu x))$

The parameter μ can be interpreted as a strict upper bound on the failure rate function

The conventional discrete distributions (such as geometric, Poisson, etc.) are not suitable for various scenarios like reliability, failure times, and counts. Consequently, alternative discrete distributions have been created by adapting well-known continuous models for reliability and failure times.

Among these, the discrete Weibull distribution stands out as the most widely used. But models like these require two parameters and not many of the known discrete distributions can provide accurate models for both times and counts, which the Discrete Lindley distribution does.

Note: in this implementation we changed the original parameters θ for μ, we did it to implement this distribution within gamlss framework.

Value

Returns a `gamlss.family` object which can be used to fit a Discrete Lindley distribution in the `gamlss()` function.

Author(s)

Yojan Andrés Alcaraz Pérez, <yalcaraz@unal.edu.co>

References

Bakouch, H. S., Jazi, M. A., & Nadarajah, S. (2014). A new discrete distribution. Statistics, 48(1), 200-240.

See Also

dDLD.

Examples

```
# Example 1
# Generating some random values with
# known mu
y <- rDLD(n=100, mu=0.3)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y~1, family=DLD,
               control=gamlss.control(n.cyc=500, trace=FALSE))

# Extracting the fitted values for mu
# using the inverse link function
exp(coef(mod1, what="mu"))

# Example 2
# Generating random values under some model

# A function to simulate a data set with Y ~ DLD
gendat <- function(n) {
  x1 <- runif(n)
  mu    <- exp(2 - 4 * x1)
  y <- rDLD(n=n, mu=mu)
  data.frame(y=y, x1=x1)
}
```

```
set.seed(1235)
datos <- gendat(n=150)

mod2 <- NULL
mod2 <- gamlss(y~x1, sigma.fo=~x2, family=DLD, data=datos,
                control=gamlss.control(n.cyc=500, trace=FALSE))

summary(mod2)

# Example 3
# Survival times in days of 72 guinea pigs.
# Taken from Bakouch et al (2014) page 26.

y <- c(12, 15, 22, 24, 24, 32, 32, 33, 34, 38, 38, 43, 44, 48,
       52, 53, 54, 54, 55, 56, 57, 58, 58, 59, 60, 60, 60, 60,
       61, 62, 63, 65, 65, 67, 68, 70, 70, 72, 73, 75, 76, 76,
       81, 83, 84, 85, 87, 91, 95, 96, 98, 99, 109, 110, 121,
       127, 129, 131, 143, 146, 146, 175, 175, 211, 233, 258,
       258, 263, 297, 341, 341, 376)

mod3 <- gamlss(y~1, family=DLD,
               control=gamlss.control(n.cyc=500, trace=FALSE))

# Extracting the fitted values for mu
# using the inverse link function
exp(coef(mod3, what="mu"))


# Example 4
# Remission times in weeks for 20 leukaemia patients.
# Taken from Bakouch et al (2014) page 26.

y <- c(1, 3, 3, 6, 7, 7, 10, 12, 14, 15, 18, 19,
       22, 26, 28, 29, 34, 40, 48, 49)

mod4 <- gamlss(y~1, family=DLD,
               control=gamlss.control(n.cyc=500, trace=FALSE))

# Extracting the fitted values for mu
# using the inverse link function
exp(coef(mod4, what="mu"))

# Example 5
# Numbers of fires in Greece for the period from 1
# July 1998 to 31 August of the same year .
# Taken from Bakouch et al (2014) page 26.

y <- c(2, 4, 4, 3, 3, 1, 2, 4, 3, 1, 1, 0, 5, 5, 0, 3, 1,
       1, 0, 1, 0, 2, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 4, 2, 2, 1, 2, 1, 2, 0, 2, 2, 1, 0, 3, 2, 1, 2,
       2, 7, 3, 5, 2, 5, 4, 5, 6, 5, 4, 3, 8, 43, 8, 4, 4,
       3, 10, 5, 4, 5, 12, 3, 8, 12, 10, 11, 6, 1, 8, 9,
       12, 9, 4, 8, 12, 11, 8, 6, 4, 7, 9, 15, 12, 15, 15,
```

```
       12, 9, 16, 7, 11, 9, 11, 6, 5, 20, 9, 8, 8, 5, 7, 10,
       6, 6, 5, 5, 15, 6, 8, 5, 6)

mod5 <- gamlss(y~1, family=DLD,
               control=gamlss.control(n.cyc=500, trace=FALSE))

# Extracting the fitted values for mu
# using the inverse link function
exp(coef(mod5, what="mu"))

# Example 6
# Final examination marks of space students in
# mathematics in the Indian Institute of Technology at Kanpur.
# Taken from Bakouch et al (2014) page 26.

y <- c(29, 25, 50, 15, 13, 27, 15, 18, 7, 7, 8, 19, 12,
       18, 5, 21, 15, 86, 21, 15, 14, 39, 15, 14, 70,
       44, 6, 23, 58, 19, 50, 23, 11, 6, 34, 18, 28, 34,
       12, 37, 4, 60, 20, 23, 40, 65, 19, 31)

mod6 <- gamlss(y~1, family=DLD,
               control=gamlss.control(n.cyc=500, trace=FALSE))

# Extracting the fitted values for mu
# using the inverse link function
exp(coef(mod6, what="mu"))
```

DMOLBE *The DMOLBE family*

Description

The function `DMOLBE()` defines the Discrete Marshall-Olkin Length Biased Exponential distribution a two parameter distribution, for a `gamlss.family` object to be used in GAMLSS fitting using the function `gamlss()`.

Usage

```
DMOLBE(mu.link = "log", sigma.link = "log")
```

Arguments

`mu.link`	defines the mu.link, with "log" link as the default for the mu parameter.
`sigma.link`	defines the sigma.link, with "log" link as the default for the sigma.

Details

The DMOLBE distribution with parameters μ and σ has a support 0, 1, 2, ... and mass function given by

$f(x|\mu,\sigma) = \frac{\sigma((1+x/\mu)\exp(-x/\mu)-(1+(x+1)/\mu)\exp(-(x+1)/\mu))}{(1-(1-\sigma)(1+x/\mu)\exp(-x/\mu))((1-(1-\sigma)(1+(x+1)/\mu)\exp(-(x+1)/\mu))}$

with $\mu > 0$ and $\sigma > 0$

Value

Returns a `gamlss.family` object which can be used to fit a DMOLBE distribution in the `gamlss()` function.

Author(s)

Olga Usuga, <olga.usuga@udea.edu.co>

References

Aljohani, H. M., Ahsan-ul-Haq, M., Zafar, J., Almetwally, E. M., Alghamdi, A. S., Hussam, E., & Muse, A. H. (2023). Analysis of Covid-19 data using discrete Marshall–Olkinin length biased exponential: Bayesian and frequentist approach. Scientific Reports, 13(1), 12243.

See Also

dDMOLBE.

Examples

```
# Example 1
# Generating some random values with
# known mu and sigma
set.seed(1234)
y <- rDMOLBE(n=200, mu=3, sigma=7)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y~1, sigma.fo=~1, family=DMOLBE)

# Extracting the fitted values for mu and sigma
# using the inverse link function
exp(coef(mod1, what="mu"))
exp(coef(mod1, what="sigma"))

# Example 2
# Generating random values under some model

# A function to simulate a data set with Y ~ DMOLBE
gendat <- function(n) {
  x1 <- runif(n)
  x2 <- runif(n)
  mu    <- exp(1.21 - 3 * x1) # 0.75 approximately
```

```
  sigma <- exp(1.26 - 2 * x2) # 1.30 approximately
  y <- rDMOLBE(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1, x2=x2)
}

set.seed(123)
dat <- gendat(n=200)

# Fitting the model
mod2 <- NULL
mod2 <- gamlss(y~x1, sigma.fo=~x2, family=DMOLBE, data=dat,
                 control=gamlss.control(n.cyc=500, trace=FALSE))

summary(mod2)


# Example 3
# Data Set I (death due to coronavirus in China). The first data set is the number
# of deaths due to coronavirus in China from 23 January to 28 March.
# The data sets used in the paper was collected from 2020 year. The data set
# is reported in https://www.worldometers.info/coronavirus/country/china/.
# The data are:

y <- c(8, 16, 15, 24, 26, 26, 38, 43, 46, 45, 57, 64, 65, 73, 73, 86, 89, 97,
       108, 97, 146, 121, 143, 142, 105, 98, 136, 114, 118, 109, 97, 150, 71,
       52, 29, 44, 47, 35, 42, 31, 38, 31, 30, 28, 27, 22, 17, 22, 11, 7,
       13, 10, 14, 13, 11, 8, 3, 7, 6, 9, 7, 4, 6, 5, 3, 5)

# Fitting the model
mod3 <- gamlss(y~1, sigma.fo=~1, family=DMOLBE,
               control=gamlss.control(n.cyc=500, trace=FALSE))

summary(mod3)

# Extracting the fitted values for mu and sigma
# using the inverse link function
mu_hat <- exp(coef(mod3, what="mu"))
mu_hat
sigma_hat <- exp(coef(mod3, what="sigma"))
sigma_hat

# Example 4
# Data Set II (daily death due to coronavirus in Pakistan). The second data
# set is the daily deaths due to coronavirus in Pakistan from 18 March
# to 30 June. The data sets used in the paper was collected from 2020 year.
# The data is reported in
# https://www.worldometers.info/coronavirus/country/Pakistan.
# The data are:

y <- c(1, 6, 6, 4, 4, 4, 1, 20, 5, 2, 3, 15, 17, 7, 8, 25, 8, 25, 11,
       25, 16, 16, 12, 11, 20, 31, 42, 32, 23, 17, 19, 38, 50, 21, 14,
       37, 23, 47, 31, 24, 9, 64, 39, 30, 36, 46, 32, 50, 34, 32, 34,
       30, 28, 35, 57, 78, 88, 60, 78, 67, 82, 68, 97, 67, 65, 105,
```

```
       83, 101, 107, 88, 178, 110, 136, 118, 136, 153, 119, 89, 105,
       60, 148, 59, 73, 83, 49, 137, 91)

# Fitting the model
mod4 <- gamlss(y~1, sigma.fo=~1, family=DMOLBE,
               control=gamlss.control(n.cyc=500, trace=FALSE))

summary(mod4)

# Extracting the fitted values for mu and sigma
# using the inverse link function
mu_hat <- exp(coef(mod4, what="mu"))
mu_hat
sigma_hat <- exp(coef(mod4, what="sigma"))
sigma_hat
```

---

dNPGL *New Poisson-generalised Lindley distribution*

Description

These functions define the density, distribution function, quantile function and random generation for the Poisson-generalised Lindley (NPGL) distribution with parameters μ and σ.

Usage

```
dNPGL(x, mu = 0.1, sigma = 2, log = FALSE)

pNPGL(q, mu = 0.1, sigma = 2, lower.tail = TRUE, log.p = FALSE)

rNPGL(n, mu = 0.1, sigma = 2)

qNPGL(p, mu = 0.1, sigma = 2, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x, q	vector of (non-negative integer) quantiles.
mu	vector of the mu parameter.
sigma	vector of the sigma parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
n	number of random values to return.
p	vector of probabilities.

Details

The Poisson-generalised Lindley distribution with parameters μ and σ has support $x = 0, 1, 2, \ldots$ and probability mass function given by

$f(x \mid \mu, \sigma) = \frac{\mu^2 + \frac{\mu^\sigma (\mu+1)^{1-\sigma} \Gamma(x+\sigma)}{\Gamma(\sigma)\Gamma(x+1)}}{(\mu+1)^{x+2}}$

with $\mu > 0$ and $\sigma > 0$.

This distribution is useful for modeling over-dispersed count data.

Note: in this implementation we changed the original parameters θ and α for μ and σ respectively, we did it to implement this distribution within gamlss framework.

Value

dNPGL gives the density, pNPGL gives the distribution function, qNPGL gives the quantile function, rNPGL generates random deviates.

Author(s)

Tomas Mesa, <tomas.mesaz@udea.edu.co>

References

Altun, E. A new two-parameter discrete poisson-generalized Lindley distribution with properties and applications to healthcare data sets. Comput Stat 36, 2841–2861 (2021). https://doi.org/10.1007/s00180-021-01097-0

See Also

NPGL.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

x_max <- 50
probs1 <- dNPGL(x=0:x_max, mu = 0.1, sigma = 2)
probs2 <- dNPGL(x=0:x_max, mu = 0.5, sigma = 5)
probs3 <- dNPGL(x=0:x_max, mu = 0.2, sigma = 6)
probs4 <- dNPGL(x=0:x_max, mu = 20, sigma = 2)

plot(x=0:x_max, y=probs1, type="h", lwd=2, col="dodgerblue", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for dNPGL",
     ylim=c(0, 0.035))
legend("topright", legend="mu=0.1, sigma=2")

plot(x=0:x_max, y=probs2, type="h", lwd=2, col="tomato", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for dNPGL",
     ylim=c(0, 0.1))
legend("topright", legend="mu=0.5, sigma=5")
```

```
plot(x=0:x_max, y=probs3, type="h", lwd=2, col="green4", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for dNPGL",
     ylim=c(0, 0.03))
legend("topright", legend="mu=0.2, sigma=6")

plot(x=0:x_max, y=probs4, type="h", lwd=2, col="magenta", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for dNPGL",
     ylim=c(0, 1))
legend("topright", legend="mu=20, sigma=2")

# Example 2
# Checking if the cumulative curves converge to 1

x_max <- 100
cumulative_probs1 <- pNPGL(q=0:x_max, mu = 0.1, sigma = 2)
cumulative_probs2 <- pNPGL(q=0:x_max, mu = 0.5, sigma = 5)
cumulative_probs3 <- pNPGL(q=0:x_max, mu = 0.2, sigma = 6)
cumulative_probs4 <- pNPGL(q=0:x_max, mu = 20, sigma = 2)

plot(x=0:x_max, y=cumulative_probs1, col="dodgerblue",
     type="o", las=1, ylim=c(0, 1),
     main="Cumulative probability for NPGL",
     xlab="X", ylab="Probability")
points(x=0:x_max, y=cumulative_probs2, type="o", col="tomato")
points(x=0:x_max, y=cumulative_probs3, type="o", col="green4")
points(x=0:x_max, y=cumulative_probs4, type="o", col="magenta")
legend("bottomright", col=c("dodgerblue", "tomato", "green4", "magenta"), lwd=3,
       legend=c("mu=0.1, sigma=2",
                "mu=0.5, sigma=5",
                "mu=0.2, sigma=6",
                "mu=20, sigma=2"))

# Example 3
# Comparing the random generator output with
# the theoretical probabilities

x_max <- 100
mu <- 0.1
sigma <- 2
probs1 <- dNPGL(x=0:x_max, mu=mu, sigma=sigma)
names(probs1) <- 0:x_max

x <- rNPGL(n=10000, mu=mu, sigma=sigma)
probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
nombres <- cn
mp <- barplot(height, beside = TRUE, names.arg = nombres,
              col=c("dodgerblue3","firebrick3"), las=1,
              xlab="X", ylab="Proportion")
legend("topright",
       legend=c("Theoretical", "Simulated"),
```

```
       bty="n", lwd=3,
       col=c("dodgerblue3","firebrick3"), lty=1)

# Example 4
# Checking the quantile function

mu <- 0.1
sigma <- 2
p <- seq(from=0, to=1, by=0.01)
qxx <- qNPGL(p=p, mu=mu, sigma=sigma, lower.tail=TRUE, log.p=FALSE)
plot(p, qxx, type="s", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles of NPGL(mu=0.1, sigma=2)")
```

---

dNPGL2 *New Poisson-generalised Lindley distribution (with mu as mean)*

Description

These functions define the density, distribution function, quantile function and random generation for the Poisson-generalised Lindley in second parametrization with parameters μ (as mean) and σ.

Usage

```
dNPGL2(x, mu = 2, sigma = 2, log = FALSE)

pNPGL2(q, mu = 2, sigma = 2, lower.tail = TRUE, log.p = FALSE)

rNPGL2(n, mu = 2, sigma = 2)

qNPGL2(p, mu = 2, sigma = 2, lower.tail = TRUE, log.p = FALSE)
```

Arguments

x, q	vector of (non-negative integer) quantiles.
mu	vector of the mu parameter.
sigma	vector of the sigma parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
n	number of random values to return.
p	vector of probabilities.

Details

The Poisson-generalised Lindley distribution with parameters μ and σ has support $x = 0, 1, 2, \ldots$ and probability mass function given by

Note: in this implementation the parameter μ is the mean of the distribution and σ (equivalent to α in the original NPGL parametrization).

Value

dNPGL2 gives the density, pNPGL2 gives the distribution function, qNPGL2 gives the quantile function, rNPGL2 generates random deviates.

Author(s)

Tomás Mesa, <tomas.mesaz@udea.edu.co>

References

Altun, E. A new two-parameter discrete poisson-generalized Lindley distribution with properties and applications to healthcare data sets. Comput Stat 36, 2841–2861 (2021). https://doi.org/10.1007/s00180-021-01097-0

See Also

NPGL, NPGL2.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

# Original parameters and their corresponding means
casos <- data.frame(
  mu_orig = c(0.1, 0.5, 0.2, 20),
  sigma   = c(2,   5,   6,   2)
)
casos$mu_mean <- (casos$sigma + casos$mu_orig) /
  (casos$mu_orig * (1 + casos$mu_orig))

x_max <- 50

probs1 <- dNPGL2(x=0:x_max, mu=casos$mu_mean[1], sigma=casos$sigma[1])
probs2 <- dNPGL2(x=0:x_max, mu=casos$mu_mean[2], sigma=casos$sigma[2])
probs3 <- dNPGL2(x=0:x_max, mu=casos$mu_mean[3], sigma=casos$sigma[3])
probs4 <- dNPGL2(x=0:x_max, mu=casos$mu_mean[4], sigma=casos$sigma[4])

plot(x = 0:x_max, y = probs1, type = "h", lwd = 2, col = "dodgerblue", las = 1,
     ylab = "P(X=x)", xlab = "X", main = "Probability for dNPGL2",
     ylim = c(0, 0.035))
legend("topright", legend = paste0("mu=", round(casos$mu_mean[1], 2),
                                   ", sigma=", casos$sigma[1]))

plot(x = 0:x_max, y = probs2, type = "h", lwd = 2, col = "tomato", las = 1,
     ylab = "P(X=x)", xlab = "X", main = "Probability for dNPGL2",
     ylim = c(0, 0.1))
legend("topright", legend = paste0("mu=", round(casos$mu_mean[2], 2),
                                   ", sigma=", casos$sigma[2]))

plot(x = 0:x_max, y = probs3, type = "h", lwd = 2, col = "green4", las = 1,
```

```
     ylab = "P(X=x)", xlab = "X", main = "Probability for dNPGL2",
     ylim = c(0, 0.03))
legend("topright", legend = paste0("mu=", round(casos$mu_mean[3], 2),
                                   ", sigma=", casos$sigma[3]))

plot(x = 0:x_max, y = probs4, type = "h", lwd = 2, col = "magenta", las = 1,
     ylab = "P(X=x)", xlab = "X", main = "Probability for dNPGL2",
     ylim = c(0, 1))
legend("topright", legend = paste0("mu=", round(casos$mu_mean[4], 4),
                                   ", sigma=", casos$sigma[4]))

# Example 2
# Checking if the cumulative curves converge to 1

x_max <- 100
cumulative_probs1 <- pNPGL2(q = 0:x_max, mu = casos$mu_mean[1], sigma = casos$sigma[1])
cumulative_probs2 <- pNPGL2(q = 0:x_max, mu = casos$mu_mean[2], sigma = casos$sigma[2])
cumulative_probs3 <- pNPGL2(q = 0:x_max, mu = casos$mu_mean[3], sigma = casos$sigma[3])
cumulative_probs4 <- pNPGL2(q = 0:x_max, mu = casos$mu_mean[4], sigma = casos$sigma[4])

plot(x = 0:x_max, y = cumulative_probs1, col = "dodgerblue",
     type = "o", las = 1, ylim = c(0, 1),
     main = "Cumulative probability for NPGL2",
     xlab = "X", ylab = "Probability")
points(x = 0:x_max, y = cumulative_probs2, type = "o", col = "tomato")
points(x = 0:x_max, y = cumulative_probs3, type = "o", col = "green4")
points(x = 0:x_max, y = cumulative_probs4, type = "o", col = "magenta")
legend("bottomright",
       col = c("dodgerblue", "tomato", "green4", "magenta"), lwd = 3,
       legend = c(
         paste0("mu=", round(casos$mu_mean[1], 2), ", sigma=", casos$sigma[1]),
         paste0("mu=", round(casos$mu_mean[2], 2), ", sigma=", casos$sigma[2]),
         paste0("mu=", round(casos$mu_mean[3], 2), ", sigma=", casos$sigma[3]),
         paste0("mu=", round(casos$mu_mean[4], 4), ", sigma=", casos$sigma[4])
       ))

# Example 3
# Comparing the random generator output with
# the theoretical probabilities

x_max <- 60
mu    <- 5
sigma <- 2
probs1 <- dNPGL2(x = 0:x_max, mu = mu, sigma = sigma)
names(probs1) <- 0:x_max

set.seed(123)
x      <- rNPGL2(n = 10000, mu = mu, sigma = sigma)
probs2 <- prop.table(table(x))

cn     <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
mp     <- barplot(height, beside = TRUE, names.arg = cn,
```

```
                  col = c("dodgerblue3", "firebrick3"), las = 1,
                  xlab = "X", ylab = "Proportion",
                  main = paste0("NPGL2(mu=", mu, ", sigma=", sigma, ")"))
legend("topright",
       legend = c("Theoretical", "Simulated"),
       bty = "n", lwd = 3,
       col = c("dodgerblue3", "firebrick3"), lty = 1)

# Example 4
# Checking the quantile function
mu    <- 5
sigma <- 2
p     <- seq(from = 0, to = 1, by = 0.01)
qxx   <- qNPGL2(p = p, mu = mu, sigma = sigma, lower.tail = TRUE, log.p = FALSE)
plot(p, qxx, type = "s", lwd = 2, col = "green3", ylab = "quantiles",
     main = paste0("Quantiles of NPGL2(mu=", mu, ", sigma=", sigma, ")"))
```

DPERKS *The Discrete Perks family*

Description

The function `DPERKS()` defines the Discrete Perks distribution a two parameter distribution, for a `gamlss.family` object to be used in GAMLSS fitting using the function `gamlss()`.

Usage

```
DPERKS(mu.link = "log", sigma.link = "log")
```

Arguments

`mu.link`	defines the mu.link, with "log" link as the default for the mu parameter.
`sigma.link`	defines the sigma.link, with "log" link as the default for the sigma.

Details

The discrete Perks distribution with parameters $\mu > 0$ and $\sigma > 0$ has a support 0, 1, 2, ... and density given by

$f(x|\mu,\sigma) = \frac{\mu(1+\mu)(e^{\sigma}-1)e^{\sigma x}}{(1+\mu e^{\sigma x})(1+\mu e^{\sigma(x+1)})}$

Note: in this implementation we changed the original parameters λ for μ and β for σ, we did it to implement this distribution within gamlss framework.

Author(s)

Veronica Seguro Varela, <vseguro@unal.edu.co>

References

Tyagi, A., Choudhary, N., & Singh, B. (2020). A new discrete distribution: Theory and applications to discrete failure lifetime and count data. J. Appl. Probab. Statist, 15, 117-143.

See Also

dDPERKS.

Examples

```
# Example 1
# Generating some random values with
# known mu and sigma
set.seed(123)
y <- rDPERKS(n=1000, mu=2.5, sigma=0.4)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y~1, family=DPERKS,
               control=gamlss.control(n.cyc=500, trace=TRUE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
exp(coef(mod1, what="mu"))
exp(coef(mod1, what="sigma"))

# Example 2
# Generating random values under some model

# A function to simulate a data set with Y ~ DPERKS
gendat <- function(n) {
  x1 <- runif(n)
  x2 <- runif(n)
  mu    <- exp(-1.6 + 5 * x1)
  sigma <- exp(1.7 - 5 * x2)
  y <- rDPERKS(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1, x2=x2)
}

set.seed(12345)
datos <- gendat(n=1000)

mod2 <- NULL
mod2 <- gamlss(y~x1, sigma.fo=~x2, family=DPERKS, data=datos,
                control=gamlss.control(n.cyc=500, trace=TRUE))

summary(mod2)

# Example 3
# The dataset comes from Tyagi et al. (2020) page 21
# The dataset contains the number of outbreaks of strikes in the
# UK coal mining industries in four successive week periods
```

```
# in the year 1948-59.
y <- rep(0:4, times=c(46, 76, 24, 9, 1))

# Fitting the model
library(gamlss)
mod3 <- gamlss(y~1, family=DPERKS,
               control=gamlss.control(n.cyc=500, trace=TRUE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
exp(coef(mod3, what="mu"))
exp(coef(mod3, what="sigma"))

# Example 4
# The dataset comes from Tyagi et al. (2020) page 21
# The dataset contains the number fishes caught
# in traps (David, 1971, pg. 168).
y <- rep(0:9, times=c(1, 2, 11, 20, 29, 23, 10, 3, 1, 0))

# Fitting the model
library(gamlss)
mod4 <- gamlss(y~1, family=DPERKS,
               control=gamlss.control(n.cyc=500, trace=TRUE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
exp(coef(mod4, what="mu"))
exp(coef(mod4, what="sigma"))

# Example 5
# The dataset comes from Tyagi et al. (2020) page 24
# This dataset consists of remission times in weeks
# for 20 leukemia patients randomly assigned to a certain treatment
y <- c(1, 3, 3, 6, 7, 7, 10, 12, 14, 15, 18,
       19, 22, 26, 28, 29, 34, 40, 48, 49)

# Fitting the model
library(gamlss)
mod4 <- gamlss(y~1, family=DPERKS)

# Extracting the fitted values for mu and sigma
# using the inverse link function
exp(coef(mod4, what="mu"))
exp(coef(mod4, what="sigma"))
```

---

dPOISXL *The Discrete Poisson XLindley*

Description

These functions define the density, distribution function, quantile function and random generation for the Discrete Poisson XLindley distribution with parameter μ.

Usage

```
dPOISXL(x, mu = 0.3, log = FALSE)

pPOISXL(q, mu = 0.3, lower.tail = TRUE, log.p = FALSE)

qPOISXL(p, mu = 0.3, lower.tail = TRUE, log.p = FALSE)

rPOISXL(n, mu = 0.3)
```

Arguments

x, q	vector of (non-negative integer) quantiles.
mu	vector of the mu parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, P[X > x].
p	vector of probabilities.
n	number of random values to return

Details

The Discrete Poisson XLindley distribution with parameters μ has a support 0, 1, 2, ... and mass function given by

$f(x|\mu) = \frac{\mu^2(x+\mu^2+3(1+\mu))}{(1+\mu)^{4+x}}$; with $\mu > 0$.

Note: in this implementation we changed the original parameters α for μ, we did it to implement this distribution within gamlss framework.

Value

dPOISXL gives the density, pPOISXL gives the distribution function, qPOISXL gives the quantile function, rPOISXL generates random deviates.

Author(s)

Mariana Blandon Mejia, <mblandonm@unal.edu.co>

References

Ahsan-ul-Haq, M., Al-Bossly, A., El-Morshedy, M., & Eliwa, M. S. (2022). Poisson XLindley distribution for count data: statistical and reliability properties with estimation techniques and inference. Computational Intelligence and neuroscience, 2022(1), 6503670.

See Also

POISXL.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

x_max <- 20
probs1 <- dPOISXL(x=0:x_max, mu=0.2)
probs2 <- dPOISXL(x=0:x_max, mu=0.5)
probs3 <- dPOISXL(x=0:x_max, mu=1.0)

# To plot the first k values
plot(x=0:x_max, y=probs1, type="o", lwd=2, col="dodgerblue", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for Poisson XLindley",
     ylim=c(0, 0.50))
points(x=0:x_max, y=probs2, type="o", lwd=2, col="tomato")
points(x=0:x_max, y=probs3, type="o", lwd=2, col="green4")
legend("topright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=0.2", "mu=0.5", "mu=1.0"))

# Example 2
# Checking if the cumulative curves converge to 1

x_max <- 20

plot_discrete_cdf(x=0:x_max,
                  fx=dPOISXL(x=0:x_max, mu=0.2), col="dodgerblue",
                  main="CDF for Poisson XLindley with mu=0.2")

plot_discrete_cdf(x=0:x_max,
                  fx=dPOISXL(x=0:x_max, mu=0.5), col="tomato",
                  main="CDF for Poisson XLindley with mu=0.5")

plot_discrete_cdf(x=0:x_max,
                  fx=dPOISXL(x=0:x_max, mu=1.0), col="green4",
                  main="CDF for Poisson XLindley with mu=1.0")

# Example 3
# Comparing the random generator output with
# the theoretical probabilities

x_max <- 15
probs1 <- dPOISXL(x=0:x_max, mu=0.3)
names(probs1) <- 0:x_max

x <- rPOISXL(n=3000, mu=0.3)
probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
```

```
mp <- barplot(height, beside = TRUE, names.arg = cn,
              col=c("dodgerblue3","firebrick3"), las=1,
              xlab="X", ylab="Proportion")
legend("topright",
       legend=c("Theoretical", "Simulated"),
       bty="n", lwd=3,
       col=c("dodgerblue3","firebrick3"), lty=1)

# Example 4
# Checking the quantile function

mu <- 0.3
p <- seq(from=0, to=1, by = 0.01)
qxx <- qPOISXL(p, mu, lower.tail = TRUE, log.p = FALSE)
plot(p, qxx, type="s", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles for Poisson XLindley mu=0.3")
```

---

dPTRTE *The Poisson–transmuted record type exponential distribution*

Description

These functions define the density, distribution function, quantile function and random generation for the Poisson–transmuted record type exponential (PTRTE) distribution with parameters μ and σ.

This distribution was proposed by Erbayram and Akdogan (2025) as a new discrete distribution obtained from a mixed Poisson model, where the Poisson parameter follows a transmuted record type exponential distribution.

Usage

```
dPTRTE(x, mu, sigma, log = FALSE)

pPTRTE(q, mu, sigma, lower.tail = TRUE, log.p = FALSE)

qPTRTE(p, mu, sigma, lower.tail = TRUE, log.p = FALSE)

rPTRTE(n, mu, sigma)
```

Arguments

x, q	vector of (non-negative integer) quantiles.
mu	vector of the mu parameter.
sigma	vector of the sigma parameter.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
p	vector of probabilities.
n	number of random values to return.

Details

The Poisson–transmuted record type exponential distribution with parameters μ and σ has support $x = 0, 1, 2, \ldots$ and probability mass function given by

$$f(x|\mu,\sigma) = \frac{\mu}{(1+\mu)^{x+1}}\left(\frac{\sigma\mu(1+x)}{1+\mu} - (\sigma-1)\right)$$

with $\mu > 0$ and $0 < \sigma < 1$.

Value

`dPTRTE` gives the density, `pPTRTE` gives the distribution function, `qPTRTE` gives the quantile function, `rPTRTE` generates random deviates.

Author(s)

Rebeca Isabel Rodriguez Gonzalez, <rebeca.rodriguez@udea.edu.co>

References

Erbayram, T., & Akdogan, Y. (2025). A new discrete model generated from mixed Poisson transmuted record type exponential distribution. Ricerca di Matematica, 74, 1225–1247.

See Also

PTRTE.

Examples

```
# Example 1
# Plotting the mass function for different parameter values

x_max <- 20
probs1 <- dPTRTE(x=0:x_max, mu=0.5, sigma=0.5)
probs2 <- dPTRTE(x=0:x_max, mu=1, sigma=0.5)
probs3 <- dPTRTE(x=0:x_max, mu=2, sigma=0.5)

# To plot the first k values
plot(x=0:x_max, y=probs1, type="o", lwd=2, col="dodgerblue", las=1,
     ylab="P(X=x)", xlab="X", main="Probability for PTRTE",
     ylim=c(0, 0.80))
points(x=0:x_max, y=probs2, type="o", lwd=2, col="tomato")
points(x=0:x_max, y=probs3, type="o", lwd=2, col="green4")
legend("topright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=0.5, sigma=0.5",
                "mu=1, sigma=0.5",
                "mu=2, sigma=0.5"))

# Example 2
# Checking if the cumulative curves converge to 1
```

```
x_max <- 10

cumulative_probs1 <- pPTRTE(q=0:x_max, mu=1, sigma=0.5)
cumulative_probs2 <- pPTRTE(q=0:x_max, mu=1, sigma=0.7)
cumulative_probs3 <- pPTRTE(q=0:x_max, mu=1, sigma=0.9)

plot(x=0:x_max, y=cumulative_probs1, col="dodgerblue",
     type="o", las=1, ylim=c(0, 1),
     main="Cumulative for PTRTE",
     xlab="X", ylab="Probability")
points(x=0:x_max, y=cumulative_probs2, type="o", col="tomato")
points(x=0:x_max, y=cumulative_probs3, type="o", col="green4")
legend("bottomright", col=c("dodgerblue", "tomato", "green4"), lwd=3,
       legend=c("mu=1, sigma=0.5",
                "mu=1, sigma=0.7",
                "mu=1, sigma=0.9"))

# Example 3
# Comparing the random generator output with
# the theoretical probabilities

x_max <- 30
probs1 <- dPTRTE(x=0:x_max, mu=0.5, sigma=0.5)
names(probs1) <- 0:x_max

x <- rPTRTE(n=1000, mu=0.5, sigma=0.5)
probs2 <- prop.table(table(x))

cn <- union(names(probs1), names(probs2))
height <- rbind(probs1[cn], probs2[cn])
mp <- barplot(height, beside=TRUE, names.arg=cn,
              col=c("dodgerblue3","firebrick3"), las=1,
              xlab="X", ylab="Proportion")
legend("topright",
       legend=c("Theoretical", "Simulated"),
       bty="n", lwd=3,
       col=c("dodgerblue3","firebrick3"), lty=1)

# Example 4
# Checking the quantile function

mu <- 1
sigma <- 0.5
p <- seq(from=0, to=1, by=0.01)
qxx <- qPTRTE(p=p, mu=mu, sigma=sigma, lower.tail=TRUE, log.p=FALSE)
plot(p, qxx, type="s", lwd=2, col="green3", ylab="quantiles",
     main="Quantiles of DPTRTE(mu=1, sigma=0.5)")
```

DsPA *Discrete power-Ailamujia distribution*

Description

The function `DsPA()` defines the discrete power-Ailamujia distribution a two parameter distribution, for a `gamlss.family` object to be used in GAMLSS fitting using the function `gamlss()`.

Usage

```
DsPA(mu.link = "log", sigma.link = "logit")
```

Arguments

`mu.link`	defines the mu.link, with "log" link as the default for the mu parameter.
`sigma.link`	defines the sigma.link, with "logit" link as the default for the sigma parameter. Other links are "probit" and "cloglog"'(complementary log-log).

Details

The discrete power-Ailamujia distribution with parameters μ and σ has a support 0, 1, 2, ... and density given by

$f(x|\mu,\sigma) = (\sigma^x)^\mu(1 - x^\mu \log(\lambda)) - (\sigma^{(x+1)})^\mu(1 - (x+1)^\mu \log(\lambda))$

Note: in this implementation we changed the original parameters β and λ for μ and σ respectively, we did it to implement this distribution within gamlss framework.

Value

Returns a `gamlss.family` object which can be used to fit a discrete power-Ailamujia distribution in the `gamlss()` function.

Author(s)

Maria Camila Mena Romana, <mamenar@unal.edu.co>

References

Alghamdi, A. S., Ahsan-ul-Haq, M., Babar, A., Aljohani, H. M., Afify, A. Z., & Cell, Q. E. (2022). The discrete power-Ailamujia distribution: properties, inference, and applications. AIMS Math, 7(5), 8344-8360.

See Also

dDsPA.

Examples

```
# Example 1
# Generating some random values with
# known mu and sigma
y <- rDsPA(n=100, mu=1.2, sigma=0.5)

# Fitting the model
```

```
library(gamlss)
mod1 <- gamlss(y~1, family=DsPA,
               control=gamlss.control(n.cyc=500, trace=TRUE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
inv_logit <- function(x) 1/(1 + exp(-x))

exp(coef(mod1, what="mu"))
inv_logit(coef(mod1, what="sigma"))

# Example 2
# Generating random values under some model

# A function to simulate a data set with Y ~ DsPA
gendat <- function(n) {
  x1 <- runif(n)
  x2 <- runif(n)
  x3 <- runif(n)
  x4 <- runif(n)
  mu    <- exp(1 + 1.2*x1 + 0.2*x2)
  sigma <- inv_logit(2 + 1.5*x3 + 1.5*x4)
  y <- rDsPA(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1, x2=x2, x3=x3, x4=x4)
}

set.seed(123)
datos <- gendat(n=100)

mod2 <- gamlss(y~x1+x2, sigma.fo=~x3+x4, family=DsPA, data=datos,
               control=gamlss.control(n.cyc=500, trace=TRUE))

summary(mod2)

# Example 3
# failure times for a sample of 15 electronic components in an acceleration life test
# Taken from
# Alghamdi et. al (202) page 8354.

y <- c(1.0, 5.0, 6.0, 11.0, 12.0, 19.0, 20.0, 22.0,
       23.0, 31.0, 37.0, 46.0, 54.0, 60.0, 66.0)

mod3 <- gamlss(y~1, family=DsPA,
               control=gamlss.control(n.cyc=500, trace=FALSE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
inv_logit <- function(x) 1/(1 + exp(-x))

exp(coef(mod3, what="mu"))
inv_logit(coef(mod3, what="sigma"))

# Example 4
```

```
# number of fires in Greece from July 1, 1998 to August 31, 1998.
# Taken from
# Alghamdi et. al (202) page 8354.

y <- c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4,
       4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5,
       5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6,
       6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7,
       8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9,
       9, 9, 9, 9, 10, 10, 10, 11, 11,
       11, 11, 12, 12, 12, 12, 12, 12,
       15, 15, 15, 15, 16, 20, 43)

mod4 <- gamlss(y~1, family=DsPA,
               control=gamlss.control(n.cyc=500, trace=FALSE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
inv_logit <- function(x) 1/(1 + exp(-x))

exp(coef(mod4, what="mu"))
inv_logit(coef(mod4, what="sigma"))
```

`estim_mu_sigma_NPGL` *estim_mu_sigma_NPGL*

Description

This function generates initial values for NPGL distribution.

Usage

```
estim_mu_sigma_NPGL(y)
```

Arguments

y	vector with the random sample

Examples

```
y <- rNPGL(n=100, mu=0.1, sigma=2)
estim_mu_sigma_NPGL(y=y)
```

estim_mu_sigma_NPGL_MM *estim_mu_sigma_NPGL_MM*

Description

This function generates initial values for NPGL distribution using the method of moments estimators given in equations (19) and (20) of Altun (2021).

Usage

```
estim_mu_sigma_NPGL_MM(y)
```

Arguments

y	vector with the random sample

Examples

```
y <- rNPGL(n=100, mu=0.1, sigma=2)
estim_mu_sigma_NPGL_MM(y=y)
```

estim_mu_sigma_NPGL2 *estim_mu_sigma_NPGL2*

Description

estim_mu_sigma_NPGL2

Usage

```
estim_mu_sigma_NPGL2(y)
```

Arguments

y	vector with the random sample

Examples

```
y <- rNPGL2(n = 100, mu = 3, sigma = 2)
estim_mu_sigma_NPGL2(y = y)
```

GGEO *The GGEO family*

Description

The function GGEO() defines the Generalized Geometric distribution a two parameter distribution, for a gamlss.family object to be used in GAMLSS fitting using the function gamlss().

Usage

```
GGEO(mu.link = "logit", sigma.link = "log")
```

Arguments

mu.link	defines the mu.link, with "log" link as the default for the mu parameter.
sigma.link	defines the sigma.link, with "logit" link as the default for the sigma. Other links are "probit" and "cloglog"'(complementary log-log)

Details

The GGEO distribution with parameters μ and σ has a support 0, 1, 2, ... and mass function given by

$f(x|\mu,\sigma) = \frac{\sigma\mu^x(1-\mu)}{(1-(1-\sigma)\mu^{x+1})(1-(1-\sigma)\mu^x)}$

with $0 < \mu < 1$ and $\sigma > 0$. If $\sigma = 1$, the GGEO distribution reduces to the geometric distribution with success probability $1 - \mu$.

Value

Returns a gamlss.family object which can be used to fit a GGEO distribution in the gamlss() function.

Author(s)

Valentina Hurtado Sepúlveda, <vhurtados@unal.edu.co>

References

Gómez-Déniz, E. (2010). Another generalization of the geometric distribution. Test, 19, 399-415.

See Also

dGGEO.

Examples

```
# Example 1
# Generating some random values with
# known mu and sigma
set.seed(123)
y <- rGGEO(n=200, mu=0.95, sigma=1.5)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y~1, family=GGEO,
               control=gamlss.control(n.cyc=500, trace=TRUE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
inv_logit <- function(x) 1/(1 + exp(-x))

inv_logit(coef(mod1, what="mu"))
exp(coef(mod1, what="sigma"))

# Example 2
# Generating random values under some model

# A function to simulate a data set with Y ~ GGEO
## Not run:
gendat <- function(n) {
  x1 <- runif(n)
  x2 <- runif(n)
  mu    <- inv_logit(1.7 - 2.8*x1)
  sigma <- exp(0.73 + 1*x2)
  y <- rGGEO(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1, x2=x2)
}

set.seed(78353)
datos <- gendat(n=100)

mod2 <- gamlss(y~x1, sigma.fo=~x2, family=GGEO, data=datos,
               control=gamlss.control(n.cyc=500, trace=TRUE))

summary(mod2)

## End(Not run)

# Example 3
# Number of accidents to 647 women working on H. E. Shells
# for 5 weeks. Taken from Gomez-Deniz (2010) page 411.

y <- rep(x=0:5, times=c(447, 132, 42, 21, 3, 2))

mod3 <- gamlss(y~1, family=GGEO,
               control=gamlss.control(n.cyc=500, trace=TRUE))
```

```
# Extracting the fitted values for mu and sigma
# using the inverse link function
inv_logit <- function(x) 1/(1 + exp(-x))

inv_logit(coef(mod3, what="mu"))
exp(coef(mod3, what="sigma"))

# Example 4
# Total number of carious teeth among the four deciduous molars
# Taken from Gomez-Deniz (2010) page 412.

y <- rep(x=0:4, times=c(64, 17, 10, 6, 3))

mod4 <- gamlss(y~1, family=GGEO,
               control=gamlss.control(n.cyc=500, trace=TRUE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
inv_logit <- function(x) 1/(1 + exp(-x))

inv_logit(coef(mod4, what="mu"))
exp(coef(mod4, what="sigma"))

# Example 5
# Results of ten shots fired from a rifle at each of 100 targets.
# Taken from Gomez-Deniz (2010) page 412.

y <- rep(x=0:10, times=c(0, 2, 4, 10, 22, 26, 18, 12, 4, 2, 0))

mod5 <- gamlss(y~1, family=GGEO,
               control=gamlss.control(n.cyc=500, trace=TRUE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
inv_logit <- function(x) 1/(1 + exp(-x))

inv_logit(coef(mod5, what="mu"))
exp(coef(mod5, what="sigma"))

# Example 6
# Fish catch data in Kemp (1992).
# Taken from Gomez-Deniz (2010) page 412.

y <- rep(x=0:9, times=c(1, 2, 11, 20, 29, 23, 10, 3, 1, 0))

mod6 <- gamlss(y~1, family=GGEO,
               control=gamlss.control(n.cyc=500, trace=TRUE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
inv_logit <- function(x) 1/(1 + exp(-x))

inv_logit(coef(mod6, what="mu"))
```

```
exp(coef(mod6, what="sigma"))
```

`grazing` *grazing dataset*

Description

In this experiment, the density of understorey birds at a series of sites in two areas either side of a stockproof fence were compared. Once side had limited grazing (mainly from native herbivores), and the other was heavily grazed by feral herbivores, mostly horses. Bird counts were done at the sites either side of the fence (the Before measurements). Then the herbivores were removed, and bird counts done again (the After measurements). The measurements are the total number of understorey-foraging birds observed in three 20-minute surveys of two hectare quadrats.

Usage

```
grazing
```

Format

`grazing`**:**

A data frame with 62 rows and 3 variables:

when when the bird count was conducted

grazed a factor with levels Reference and Feral

birds the number of understorey birds ...

Author(s)

Rodrigo Matheus

Source

https://github.com/rdmatheus/bergreg

HYPERPO *The hyper Poisson family*

Description

The function HYPERPO() defines the hyper Poisson distribution a two parameter distribution, for a gamlss.family object to be used in GAMLSS fitting using the function gamlss().

Usage

```
HYPERPO(mu.link = "log", sigma.link = "log")
```

Arguments

mu.link	defines the mu.link, with "log" link as the default for the mu parameter.
sigma.link	defines the sigma.link, with "log" link as the default for the sigma.

Details

The hyper-Poisson distribution with parameters μ and σ has a support 0, 1, 2, ... and density given by

$f(x|\mu,\sigma) = \frac{\mu^x}{{}_1F_1(1;\mu;\sigma)} \frac{\Gamma(\sigma)}{\Gamma(x+\sigma)}$

where the function ${}_1F_1(a;c;z)$ is defined as

${}_1F_1(a;c;z) = \sum_{r=0}^{\infty} \frac{(a)_r}{(c)_r} \frac{z^r}{r!}$

and $(a)_r = \frac{\gamma(a+r)}{\gamma(a)}$ for $a > 0$ and r positive integer.

Note: in this implementation we changed the original parameters λ and γ for μ and σ respectively, we did it to implement this distribution within gamlss framework.

Value

Returns a gamlss.family object which can be used to fit a hyper-Poisson distribution in the gamlss() function.

Author(s)

Freddy Hernandez, <fhernanb@unal.edu.co>

References

Sáez-Castillo, A. J., & Conde-Sánchez, A. (2013). A hyper-Poisson regression model for overdispersed and underdispersed count data. Computational Statistics & Data Analysis, 61, 148-157.

See Also

dHYPERPO.

Examples

```
# Example 1
# Generating some random values with
# known mu and sigma
set.seed(1234)
y <- rHYPERPO(n=200, mu=10, sigma=1.5)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y~1, sigma.fo=~1, family=HYPERPO,
               control=gamlss.control(n.cyc=500, trace=TRUE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
exp(coef(mod1, what="mu"))
exp(coef(mod1, what="sigma"))

# Example 2
# Generating random values under some model

# A function to simulate a data set with Y ~ HYPERPO
gendat <- function(n) {
  x1 <- runif(n)
  x2 <- runif(n)
  mu    <- exp(1.21 - 3 * x1) # 0.75 approximately
  sigma <- exp(1.26 - 2 * x2) # 1.30 approximately
  y <- rHYPERPO(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1, x2=x2)
}

set.seed(1234)
dat <- gendat(n=100)

mod2 <- gamlss(y~x1, sigma.fo=~x2, family=HYPERPO, data=dat,
                control=gamlss.control(n.cyc=500, trace=TRUE))

summary(mod2)
```

HYPERPO2 *The hyper Poisson family (with mu as mean)*

Description

The function `HYPERPO2()` defines the hyper Poisson distribution (with mu as mean) a two parameter distribution, for a `gamlss.family` object to be used in GAMLSS fitting using the function `gamlss()`.

Usage

```
HYPERPO2(mu.link = "log", sigma.link = "log")
```

Arguments

`mu.link`	defines the mu.link, with "log" link as the default for the mu parameter.
`sigma.link`	defines the sigma.link, with "log" link as the default for the sigma.

Details

The hyper-Poisson distribution with parameters μ and σ has a support 0, 1, 2, ...

Note: in this implementation the parameter μ is the mean of the distribution and σ corresponds to the dispersion parameter. If you fit a model with this parameterization, the time will increase because an internal procedure to convert μ to λ parameter.

Value

Returns a `gamlss.family` object which can be used to fit a hyper-Poisson distribution version 2 in the `gamlss()` function.

Author(s)

Freddy Hernandez, <fhernanb@unal.edu.co>

References

Sáez-Castillo, A. J., & Conde-Sánchez, A. (2013). A hyper-Poisson regression model for overdispersed and underdispersed count data. Computational Statistics & Data Analysis, 61, 148-157.

See Also

dHYPERPO2, HYPERPO.

Examples

```
# Example 1
# Generating some random values with
# known mu and sigma
set.seed(1234)
y <- rHYPERPO2(n=100, mu=4, sigma=1.5)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y~1, sigma.fo=~1, family=HYPERPO2,
               control=gamlss.control(n.cyc=500, trace=TRUE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
exp(coef(mod1, what="mu"))
exp(coef(mod1, what="sigma"))

# Example 2
# Generating random values under some model
```

```
## Not run:
# A function to simulate a data set with Y ~ HYPERPO2
gendat <- function(n) {
  x1 <- runif(n)
  x2 <- runif(n)
  mu    <- exp(1.21 - 3 * x1) # 0.75 approximately
  sigma <- exp(1.26 - 2 * x2) # 1.30 approximately
  y <- rHYPERPO2(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1, x2=x2)
}

set.seed(12345)
dat <- gendat(n=200)

mod2 <- gamlss(y~x1, sigma.fo=~x2, family=HYPERPO2, data=dat,
               control=gamlss.control(n.cyc=500, trace=TRUE))

summary(mod2)

## End(Not run)
```

---

logLik_NPGL *logLik_NPGL*

Description

This is an auxiliar function to obtain the logLik for NPGL.

Usage

```
logLik_NPGL(param = c(0, 0), x)
```

Arguments

param	vector with the values for mu and sigma
x	vector with the data

Examples

```
y <- rNPGL(n=100, mu=0.1, sigma=2)
logLik_NPGL(param=c(0, 0), x=y)
```

logLik_NPGL2 *logLik_NPGL2*

Description

Auxiliary function to evaluate the log-likelihood of the mean-parametrized NPGL2 distribution.

Usage

```
logLik_NPGL2(param = c(log(3), log(2)), x)
```

Arguments

param	vector with the values for mu and sigma
x	vector with the data

Examples

```
y <- rNPGL2(n = 100, mu = 3, sigma = 2)
logLik_NPGL2(param = c(log(3), log(2)), x = y)
```

make.link.gamlss *Create a Link for GAMLSS families*

Description

The function `make.link.gamlss()` is used with `gamlss.family` distributions in package **gamlss()**.Given a link, it returns a link function, an inverse link function, the derivative dpar/deta where 'par' is the appropriate distribution parameter and a function for checking the domain. It differs from the usual `make.link` of `glm()` by having extra links as the `logshifto1`, and the own. For the use of the own link see the example bellow. `show.link` provides a way in which the user can identify the link functions available for each gamlss distribution. If your required link function is not available for any of the gamlss distributions you can add it in.

Usage

```
make.link.gamlss(link)
```

Arguments

link	character or numeric; one of `"logit"`, `"probit"`, `"cloglog"`, `"identity"`, `"log"`, `"sqrt"`, `"1/mu^2"`, `"inverse"`, `"logshifted"`, `"logitshifted"`, or number, say lambda resulting in power link μ^{λ}.

Details

The own link function is added to allow the user greater flexibility. In order to used the own link function for any of the parameters of the distribution the own link should appear in the available links for this parameter. You can check this using the function show.link. If the own do not appear in the list you can create a new function for the distribution in which own is added in the list. For example the first line of the code of the binomial distribution, BI, has change from

"mstats <- checklink("mu.link", "Binomial", substitute(mu.link), c("logit", "probit", "cloglog", "log")), in version 1.0-0 of gamlss, to

"mstats <- checklink("mu.link", "Binomial", substitute(mu.link), c("logit", "probit", "cloglog", "log", "own"))

in version 1.0-1. Given that the parameter has own as an option the user needs also to define the following four new functions in order to used an own link.

i) own.linkfun

ii) own.linkinv

iii) own.mu.eta and

iv) own.valideta.

An example is given below.

Only one parameter of the distribution at a time is allowed to have its own link, (unless the same four own functions above are suitable for more that one parameter of the distribution).

Note that from **gamlss** version 1.9-0 the user can introduce its own link function by define an appropriate function, (see the example below).

Value

For the make.link.gamlss a list with components

linkfun: Link function function(parameter)

linkinv: Inverse link function function(eta)

mu.eta: Derivative function(eta) dparameter/deta

valideta: function(eta) TRUE if all of eta is in the domain of linkinv.

For the show.link a list with components the available links for the distribution parameters

Note

For the links involving parameters as in logshifted and logitshifted the parameters can be passed in the definition of the distribution by calling the checklink function, for example in the definition of the tau parameter in BCPE distribution the following call is made: tstats <- checklink("tau.link","Box Cox Power Exponential", substitute(tau.link),c("logshifted", "log", "identity"), par.link = c(1))

This function is the same make.link.gamlss fuction from gamlss.dist package.

mean_var_hp *Mean and variance for hyper-Poisson distribution*

Description

This function calculates the mean and variance for the hyper-Poisson distribution with parameters μ and σ.

Usage

```
mean_var_hp(mu, sigma)

mean_var_hp2(mu, sigma)
```

Arguments

mu	value of the mu parameter.
sigma	value of the sigma parameter.

Details

The hyper-Poisson distribution with parameters μ and σ has a support 0, 1, 2, ... and density given by

$f(x|\mu,\sigma) = \frac{\mu^x}{{}_1F_1(1;\mu;\sigma)} \frac{\Gamma(\sigma)}{\Gamma(x+\sigma)}$

where the function ${}_1F_1(a;c;z)$ is defined as

${}_1F_1(a;c;z) = \sum_{r=0}^{\infty} \frac{(a)_r}{(c)_r} \frac{z^r}{r!}$

and $(a)_r = \frac{\gamma(a+r)}{\gamma(a)}$ for $a > 0$ and r positive integer.

This function calculates the mean and variance of this distribution.

Value

the function returns a list with the mean and variance.

Author(s)

Freddy Hernandez, <fhernanb@unal.edu.co>

References

Sáez-Castillo, A. J., & Conde-Sánchez, A. (2013). A hyper-Poisson regression model for overdispersed and underdispersed count data. Computational Statistics & Data Analysis, 61, 148-157.

See Also

HYPERPO.

Examples

```
# Example 1

# Theoretical values
mean_var_hp(mu=5.5, sigma=0.1)

# Using simulated values
y <- rHYPERPO(n=1000, mu=5.5, sigma=0.1)
mean(y)
var(y)


# Example 2

# Theoretical values
mean_var_hp2(mu=5.5, sigma=1.9)

# Using simulated values
y <- rHYPERPO2(n=1000, mu=5.5, sigma=1.9)
mean(y)
var(y)
```

NMES1988 *NMES1988 dataset*

Description

Cross-section data originating from the US National Medical Expenditure Survey (NMES) conducted in 1987 and 1988. The NMES is based upon a representative, national probability sample of the civilian non-institutionalized population and individuals admitted to long-term care facilities during 1987. The data are a subsample of individuals ages 66 and over all of whom are covered by Medicare (a public insurance program providing substantial protection against health-care costs).

Usage

```
NMES1988
```

Format

NMES1988**:**

A data frame containing 4,406 observations on 19 variables:

visits Number of physician office visits.

nvisits Number of non-physician office visits.

ovisits Number of physician hospital outpatient visits.

novisits Number of non-physician hospital outpatient visits.

emergency Emergency room visits.

hospital Number of hospital stays.

health Factor indicating self-perceived health status, levels are "poor", "average" (reference category), "excellent".

chronic Number of chronic conditions.

adl Factor indicating whether the individual has a condition that limits activities of daily living ("limited") or not ("normal").

region Factor indicating region, levels are northeast, midwest, west, other (reference category).

age Age in years (divided by 10).

afam Factor. Is the individual African-American?

gender Factor indicating gender.

married Factor. Is the individual married?

school Number of years of education.

income Family income in USD 10,000.

employed Factor. Is the individual employed?

insurance Factor. Is the individual covered by private insurance?

medicaid Factor. Is the individual covered by Medicaid?

Author(s)

AER R package.

Source

This dataset comes from the AER R package.

NPGL *New Poisson-generalised Lindley distribution*

Description

The function `NPGL()` defines the Poisson-generalised Lindley distribution, a two parameter distribution, for a `gamlss.family` object to be used in GAMLSS fitting using the function `gamlss()`.

Usage

```
NPGL(mu.link = "log", sigma.link = "log")
```

Arguments

`mu.link`	defines the mu.link, with "log" link as the default for the mu parameter.
`sigma.link`	defines the sigma.link, with "log" link as the default for the sigma.

Details

The Poisson-generalised Lindley distribution with parameters μ and σ has support $x = 0, 1, 2, \ldots$ and probability mass function given by

$f(x \mid \mu, \sigma) = \frac{\mu^2 + \frac{\mu^\sigma (\mu+1)^{1-\sigma} \Gamma(x+\sigma)}{\Gamma(\sigma)\Gamma(x+1)}}{(\mu+1)^{x+2}}$

with $\mu > 0$ and $\sigma > 0$.

This distribution is useful for modeling over-dispersed count data.

Note: in this implementation we changed the original parameters θ and α for μ and σ respectively, we did it to implement this distribution within gamlss framework.

Value

Returns a `gamlss.family` object which can be used to fit a NPGL distribution in the `gamlss()` function.

Author(s)

Tomás Mesa, <tomas.mesaz@udea.edu.co>

References

Altun, E. A new two-parameter discrete poisson-generalized Lindley distribution with properties and applications to healthcare data sets. Comput Stat 36, 2841–2861 (2021). https://doi.org/10.1007/s00180-021-01097-0

See Also

dNPGL.

Examples

```
# Example 1
# Generating some random values with
# known mu and sigma

set.seed(123)
y <- rNPGL(n=100, mu=20, sigma=2)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y~1, family=NPGL,
               control=gamlss.control(n.cyc=500, trace=FALSE))

# Extracting the fitted values for mu and sigma
exp(coef(mod1, what="mu"))
exp(coef(mod1, what="sigma"))

# Example 2
# Generating random values under some model
```

```
# A function to simulate a data set with Y ~ NPGL
gendat <- function(n) {
  x1 <- runif(n)
  x2 <- runif(n)
  mu    <- exp(1.7 - 2.8 * x1) # Approx 1.35
  sigma <- exp(0.73 + 1 * x2)  # Approx 3.42
  y <- rNPGL(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1, x2=x2)
}

set.seed(1234)
datos <- gendat(n=200)

mod2 <- gamlss(y~x1, sigma.fo=~x2, family=NPGL, data=datos,
               control=gamlss.control(n.cyc=800, trace=FALSE))

summary(mod2)
```

NPGL2 *Mean-parametrized New Poisson-generalised Lindley distribution (NPGL2)*

Description

The function NPGL2() defines the mean-parametrized version of the Poisson-generalised Lindley distribution, a two-parameter distribution, for a gamlss.family object to be used in GAMLSS fitting using the function gamlss().

Usage

```
NPGL2(mu.link = "log", sigma.link = "log")
```

Arguments

mu.link	defines the mu.link, with "log" link as the default for the mu parameter (the mean of the distribution).
sigma.link	defines the sigma.link, with "log" link as the default for the sigma parameter (parameter alpha of the original NPGL).

Details

This family uses the mean-parametrized version of the NPGL distribution proposed in Section 6 of Altun (2021). The new parameters are:

- $\mu > 0$: the mean of the distribution, i.e. $E[X]$.
- $\sigma > 0$

The reparametrization links the mean and σ to the original parameter μ via the mean equation $E[X] = (\sigma + \mu)/(\mu(1 + \mu))$

Value

Returns a `gamlss.family` object which can be used to fit a mean-parametrized NPGL distribution in the `gamlss()` function.

Author(s)

Tomas Mesa, <tomas.mesaz@udea.edu.co>

References

Altun, E. A new two-parameter discrete poisson-generalized Lindley distribution with properties and applications to healthcare data sets. Comput Stat 36, 2841-2861 (2021). https://doi.org/10.1007/s00180-021-01097-0

See Also

dNPGL2, NPGL.

Examples

```
# Example 1
# Generating some random values with
# known mu and sigma

set.seed(123)
y <- rNPGL2(n=5000, mu=4, sigma=2)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y~1, sigma.fo=~1, family=NPGL2,
               control=gamlss.control(n.cyc=500, trace=TRUE))

# Extracting the fitted values for mu and sigma
# using the inverse link function
exp(coef(mod1, what="mu"))
exp(coef(mod1, what="sigma"))

# Example 2
# Generating random values under some model

# A function to simulate a data set with Y ~ NPGL2
gendat <- function(n) {
  x1 <- runif(n)
  x2 <- runif(n)
  mu    <- exp(1.5 - 1.8 * x1)  # mean of the distribution
  sigma <- exp(0.5 + 0.8 * x2)  # shape parameter alpha
  y <- rNPGL2(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1, x2=x2)
}

set.seed(123)
```

```
dat <- gendat(n=1000)

# Fitting the model
mod2 <- gamlss(y~x1, sigma.fo=~x2, family=NPGL2, data=dat,
               control=gamlss.control(n.cyc=800, trace=TRUE))

summary(mod2)

# Example 3
# Using the data from Altun (2021), Section 8.1.
# The dataset is about the 1991 Arizona cardiovascular patient.
# We model the length of hospital stay (los) with cardiovascular
# procedure, sex, type of admission and age as covariates.
# The response variable has a dispersion index of 5.43,
# confirming over-dispersion.
# Data available in the azpro dataset of the COUNT package.

mod3 <- gamlss(
  los ~ procedure + sex + admit + age75,
  sigma.fo = ~1,
  family = NPGL2(mu.link = "log"),
  data = azpro,
  control = gamlss.control(n.cyc = 500, trace = TRUE)
  )

# Extracting the fitted values for mu and sigma
# using the inverse link function
coef(mod3, what="mu")
coef(mod3, what="sigma")

# Note: coef(mod3, what="mu") is the estimated mean length
# of hospital stay for the reference group (PTCA procedure,
# female, elective admission, age <= 75), directly interpretable
# because mu is the mean of the NPGL2 distribution.

# Example 4
# Using the data from Altun (2021), Section 8.2.
# The dataset originates from the US National Medical Expenditure
# Survey (NMES) conducted in 1987 and 1988. We model the number
# of physician office visits with hospital stays, chronic conditions,
# activity limitations, age, gender, marital status, income,
# employment status, Medicaid, private insurance and self-perceived
# health status as covariates.
# The response variable has a dispersion index of 7.91,
# confirming over-dispersion.
# Data available in the NMES1988 dataset of the AER package.
# Set "average" as the reference category for health status,
# so that healthexcellent and healthpoor are the two dummies,
# matching x11 and x12 in Altun (2021).

NMES1988$health <- relevel(factor(NMES1988$health), ref = "average")

mod4 <- gamlss(
```

```
  visits ~ hospital + chronic + adl + age + gender + married +
    income + employed + medicaid + insurance + health,
  sigma.fo = ~1,
  family = NPGL2(mu.link = "log"),
  data = NMES1988,
  control = gamlss.control(n.cyc = 500, trace = TRUE)
)

coef(mod4, what="mu")
coef(mod4, what="sigma")
```

`plot_discrete_cdf` *Draw the CDF for a discrete random variable*

Description

Draw the CDF for a discrete random variable

Usage

```
plot_discrete_cdf(x, fx, col = "blue", lwd = 3, ...)
```

Arguments

`x`	vector with the values of the random variable X.
`fx`	vector with the probabilities of X.
`col`	color for the line.
`lwd`	line width.
`...`	further arguments and graphical parameters.

Value

A plot with the cumulative distribution function.

Author(s)

Freddy Hernandez, <fhernanb@unal.edu.co>

Examples

```
# Example 1
# for a particular distribution

x <- 1:6
fx <- c(0.19, 0.21, 0.4, 0.12, 0.05, 0.03)
plot_discrete_cdf(x, fx, las=1, main="")

# Example 2
```

```
# for a Poisson distribution
x <- 0:10
fx <- dpois(x, lambda=3)
plot_discrete_cdf(x, fx, las=1,
                  main="CDF for Poisson")
```

POISXL *The Discrete Poisson XLindley*

Description

The function POISXL() defines the Discrete Poisson XLindley distribution a single parameter distribution, for a gamlss.family object to be used in GAMLSS fitting using the function gamlss().

Usage

```
POISXL(mu.link = "log")
```

Arguments

mu.link	defines the mu.link, with "log" link as the default for the mu parameter.

Details

The Discrete Poisson XLindley distribution with parameters μ has a support 0, 1, 2, ... and mass function given by

$f(x|\mu) = \frac{\mu^2(x+\mu^2+3(1+\mu))}{(1+\mu)^{4+x}}$; with $\mu > 0$.

Note: in this implementation we changed the original parameters α for μ, we did it to implement this distribution within gamlss framework.

Value

Returns a gamlss.family object which can be used to fit a Discrete Poisson XLindley distribution in the gamlss() function.

Author(s)

Mariana Blandon Mejia, <mblandonm@unal.edu.co>

References

Ahsan-ul-Haq, M., Al-Bossly, A., El-Morshedy, M., & Eliwa, M. S. (2022). Poisson XLindley distribution for count data: statistical and reliability properties with estimation techniques and inference. Computational Intelligence and neuroscience, 2022(1), 6503670.

See Also

dPOISXL.

Examples

```
# Example 1
# Generating some random values with
# known mu
y <- rPOISXL(n=1000, mu=1)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y~1, family=POISXL,
               control=gamlss.control(n.cyc=500, trace=FALSE))

# Extracting the fitted values for mu
# using the inverse link function
exp(coef(mod1, what="mu"))

# Example 2
# Generating random values under some model

# A function to simulate a data set with Y ~ POISXL
gendat <- function(n) {
  x1 <- runif(n, min=0.4, max=0.6)
  mu <- exp(1.21 - 3 * x1) # 0.75 approximately
  y <- rPOISXL(n=n, mu=mu)
  data.frame(y=y, x1=x1)
}

dat <- gendat(n=1500)

# Fitting the model
mod2 <- NULL
mod2 <- gamlss(y~x1, family=POISXL, data=dat,
               control=gamlss.control(n.cyc=500, trace=FALSE))

summary(mod2)

# Example 3
# The counts the number of borers per hill of corn in an
# experiment conducted randomly on 8 hills in 15 replications.
# Taken from Ahsan-ul-Haq et al (2022) page 10.

y <- rep(x=0:8, times=c(43, 35, 17, 11, 5, 4, 1, 2, 2))

mod3 <- gamlss(y~1, family=POISXL,
               control=gamlss.control(n.cyc=500, trace=FALSE))

# Extracting the fitted values for mu
exp(coef(mod3, what="mu"))

# Example 4
# The number of mammalian cytogenetic dosimetry lesions produced
# by streptogramin exposure in rabbit.
# Taken from Ahsan-ul-Haq et al (2022) page 10.
```

```
y <- rep(x=0:4, times=c(200, 57, 30, 7, 6))

mod4 <- gamlss(y~1, family=POISXL,
               control=gamlss.control(n.cyc=500, trace=FALSE))

# Extracting the fitted values for mu
exp(coef(mod4, what="mu"))
```

PTRTE *Poisson-transmuted record type exponential distribution*

Description

The function `PTRTE()` defines the Poisson-transmuted record type exponential distribution, a two-parameter discrete distribution, for a `gamlss.family` object to be used in GAMLSS fitting using the function `gamlss()`.

Usage

```
PTRTE(mu.link = "log", sigma.link = "logit")
```

Arguments

`mu.link`	defines the link function for the mu parameter, with "log" as the default.
`sigma.link`	defines the link function for the sigma parameter, with "logit" as the default.

Details

The Poisson-transmuted record type exponential distribution with parameters μ and σ has support $x = 0, 1, 2, \ldots$ and probability mass function given by

$$f(x|\mu,\sigma) = \frac{\mu}{(1+\mu)^{x+1}}\left(\frac{\sigma\mu(1+x)}{1+\mu} - (\sigma-1)\right)$$

Parameter restrictions: $\mu > 0$ and $0 < \sigma < 1$.

Note: we renamed the original parameters θ and p to μ and σ respectively to implement this distribution within the `gamlss` framework.

Value

Returns a `gamlss.family` object that can be used to fit the Poisson-transmuted record type exponential distribution using the `gamlss()` function.

Author(s)

Rebeca Isabel Rodriguez Gonzalez, <rebeca.rodriguez@udea.edu.co>

References

Erbayram, T., & Akdogan, Y. (2025). A new discrete model generated from mixed Poisson transmuted record type exponential distribution. Ricerca di Matematica, 74, 1225–1247.

See Also

dPTRTE.

Examples

```
# Example 1
# Generating some random values with known mu and sigma
# logit_inv function
logit_inv <- function(x) exp(x) / (exp(x)+1)

set.seed(12345)
y <- rPTRTE(n=200, mu=0.2, sigma=0.5)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y~1, family=PTRTE)

# Extracting the fitted values for mu and sigma
exp(coef(mod1, what="mu"))
logit_inv(coef(mod1, what="sigma"))

# Example 2
# Generating random values under some model
# A function to simulate a data set with Y ~ PTRTE

gendat <- function(n) {
  x1 <- runif(n)
  x2 <- runif(n)
  mu    <- exp(2 + 1 * x1) # 12 approximately
  sigma <- logit_inv(2 - 2 * x2) # 0.73 approximately
  y <- rPTRTE(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1, x2=x2)
}

set.seed(12345)
dat <- gendat(n=2000)

# Fitting the model
mod2 <- NULL
mod2 <- gamlss(y~x1, sigma.fo=~x2, family=PTRTE, data=dat,
               control=gamlss.control(n.cyc=500, trace=FALSE))

summary(mod2)

# Example 3 (Second data set of the article)
# European corn-borer count data reported by McGuire et al. (1957).
# The observed and fitted frequencies are given in Table 11 of
```

```
# Erbayram and Akdogan (2025), where the P-TRTE distribution is
# illustrated using this data set.

values <- 0:5
freq <- c(188, 83, 36, 14, 2, 1)

y <- rep(x=values, times=freq)

mod3 <- gamlss(y~1, sigma.fo=~1, family=PTRTE(),
               control=gamlss.control(n.cyc=500, trace=TRUE))

exp(coef(mod3, what="mu"))
logit_inv(coef(mod3, what="sigma"))
```

Index

∗ **datasets**
  azpro, 4
  grazing, 79
  NMES1988, 87

add, 3
azpro, 4

BerG, 5, *13*

checklink, 7
COMPO, 7, *18*
COMPO2, 10, *21*

dBerG, *5*, 12
DBH, 15, *23*
dCOMPO, *8*, 17
dCOMPO2, *11*, 19
dDBH, *16*, 22
dDGEII, 25, *41*
dDIKUM, 27, *51*
dDLD, 30, *53*
dDMOLBE, 33, *56*
dDPERKS, 35, *65*
dDsPA, 38, *72*
DGEII, *26*, 40
dGGEO, 42, *76*
dHYPERPO, 45, *80*
dHYPERPO2, 48, *82*
DIKUM, *29*, 50
DLD, *31*, 52
DMOLBE, *34*, 55
dNPGL, 58, *89*
dNPGL2, 61, *91*
DPERKS, *36*, 64
dPOISXL, 66, *94*
dPTRTE, 69, *97*
DsPA, *39*, 71

estim_mu_sigma_NPGL, 74
estim_mu_sigma_NPGL2, 75
estim_mu_sigma_NPGL_MM, 75

GGEO, *43*, 76
grazing, 79

HYPERPO, *46*, *49*, 80, *82*, *86*
HYPERPO2, *49*, 81

logLik_NPGL, 83
logLik_NPGL2, 84

make.link.gamlss, 84
mean_var_hp, 86
mean_var_hp2 *(*mean_var_hp*)*, 86

NMES1988, 87
NPGL, *59*, *62*, 88, *91*
NPGL2, *62*, 90

pBerG *(*dBerG*)*, 12
pCOMPO *(*dCOMPO*)*, 17
pCOMPO2 *(*dCOMPO2*)*, 20
pDBH *(*dDBH*)*, 22
pDGEII *(*dDGEII*)*, 25
pDIKUM *(*dDIKUM*)*, 27
pDLD *(*dDLD*)*, 30
pDMOLBE *(*dDMOLBE*)*, 33
pDPERKS *(*dDPERKS*)*, 35
pDsPA *(*dDsPA*)*, 38
pGGEO *(*dGGEO*)*, 42
pHYPERPO *(*dHYPERPO*)*, 45
pHYPERPO2 *(*dHYPERPO2*)*, 48
plot_discrete_cdf, 93
pNPGL *(*dNPGL*)*, 58
pNPGL2 *(*dNPGL2*)*, 61
POISXL, *68*, 94
pPOISXL *(*dPOISXL*)*, 66
pPTRTE *(*dPTRTE*)*, 69
PTRTE, *70*, 96

qBerG *(*dBerG*)*, 12

qCOMPO (dCOMPO), 17
qCOMPO2 (dCOMPO2), 20
qDBH (dDBH), 22
qDGEII (dDGEII), 25
qDIKUM (dDIKUM), 27
qDLD (dDLD), 30
qDMOLBE (dDMOLBE), 33
qDPERKS (dDPERKS), 35
qDsPA (dDsPA), 38
qGGEO (dGGEO), 42
qHYPERPO (dHYPERPO), 45
qHYPERPO2 (dHYPERPO2), 48
qNPGL (dNPGL), 58
qNPGL2 (dNPGL2), 61
qPOISXL (dPOISXL), 66
qPTRTE (dPTRTE), 69

rBerG (dBerG), 12
rCOMPO (dCOMPO), 17
rCOMPO2 (dCOMPO2), 20
rDBH (dDBH), 22
rDGEII (dDGEII), 25
rDIKUM (dDIKUM), 27
rDLD (dDLD), 30
rDMOLBE (dDMOLBE), 33
rDPERKS (dDPERKS), 35
rDsPA (dDsPA), 38
rGGEO (dGGEO), 42
rHYPERPO (dHYPERPO), 45
rHYPERPO2 (dHYPERPO2), 48
rNPGL (dNPGL), 58
rNPGL2 (dNPGL2), 61
rPOISXL (dPOISXL), 66
rPTRTE (dPTRTE), 69