Package ‘DiscreteDists’

September 13, 2024

Title Discrete Statistical Distributions

Version 1.0.0

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

RdMacros Rdpack

Imports gamlss, gamlss.dist, pracma, Rdpack, Rcpp

LinkingTo Rcpp

Encoding UTF-8

RoxygenNote 7.3.1

Suggests knitr, rmarkdown

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

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

NeedsCompilation yes

Author Freddy Hernandez-Barajas [aut, cre]
(<https://orcid.org/0000-0001-7459-3329>),
Fernando Marmolejo-Ramos [aut]
(<https://orcid.org/0000-0003-4680-1287>),
Jamiu Olumoh [aut] (<https://orcid.org/0000-0002-7371-3920>),
Osho Ajayi [aut] (<https://orcid.org/0000-0002-2436-3338>),
Olga Usuga-Manco [aut] (<https://orcid.org/0000-0003-3062-1820>)

Maintainer Freddy Hernandez-Barajas <fhernanb@unal.edu.co>

Repository CRAN

Date/Publication 2024-09-13 18:10:06 UTC

Contents

`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
```

DBH *The Discrete Burr Hatke family*

Description

The function DBH() defines the Discrete Burr Hatke distribution, one-parameter discrete 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 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

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 MS, 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"))
```

---

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 MS, 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])
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 <- 0.97
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 MH, 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])
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 <- 0.5
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 AA, Hegazy MA, AL-Dayian GR, Abd EL-Kader RE (2022). “A Discrete Analog of the Inverted Kumaraswamy Distribution: Properties and Estimation with Application to COVID-19 Data.” *Pakistan Journal of Statistics & Operation Research*, **18**(1).

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

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

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 HS, Jazi MA, 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])
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 <- 0.9
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 HM, Ahsan-ul-Haq M, Zafar J, Almetwally EM, Alghamdi AS, Hussam E, Muse AH (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 <- 15
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])
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 <- 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)")
```

DGEII *The DGEII distribution*

Description

The function `DGEII()` defines the Discrete generalized exponential distribution, 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 MH, 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
set.seed(189)
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=FALSE))

# 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
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)
}

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

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

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=FALSE))

# 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])
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 <- 0.5
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 AJ, 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 AJ, 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)")
```

`dHYPERPO_single` *Function to obtain the dHYPERPO for a single value x*

Description

Function to obtain the dHYPERPO for a single value x

Usage

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

Arguments

`x`	numeric value for x.
`mu`	numeric value for nu.
`sigma`	numeric value for sigma.
`log`	logical value for log.

Value

returns the pmf for a single value x.

`dHYPERPO_vec` *Function to obtain the dHYPERPO for a vector x*

Description

Function to obtain the dHYPERPO for a vector x

Usage

```
dHYPERPO_vec(x, mu, sigma, log)
```

Arguments

`x`	numeric value for x.
`mu`	numeric value for nu.
`sigma`	numeric value for sigma.
`log`	logical value for log.

Value

returns the pmf for a vector.

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 AA, Hegazy MA, AL-Dayian GR, Abd EL-Kader RE (2022). "A Discrete Analog of the Inverted Kumaraswamy Distribution: Properties and Estimation with Application to COVID-19 Data." *Pakistan Journal of Statistics & Operation Research*, **18**(1).

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=150)

# 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, one-parameter discrete 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 HS, Jazi MA, 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)
```

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 HM, Ahsan-ul-Haq M, Zafar J, Almetwally EM, Alghamdi AS, Hussam E, Muse AH (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=100, mu=10, sigma=7)

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

# A function to simulate a data set with Y ~ DMOLBE
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 <- rDMOLBE(n=n, mu=mu, sigma=sigma)
  data.frame(y=y, x1=x1,x2=x2)
}

set.seed(123)
dat <- gendat(n=350)
```

```
# 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)
```

`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 MS, others (2022). "Poisson XLindley distribution for count data: statistical and reliability properties with estimation techniques and inference." *Computational Intelligence and Neuroscience*, **2022**.

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])
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 <- 0.3
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")
```

`f11_cpp` *Function to obtain F11 with C++.*

Description

Function to obtain F11 with C++.

Usage

```
f11_cpp(gamma, lambda, maxiter_series = 10000L, tol = 1e-10)
```

Arguments

`gamma`	numeric value for gamma.
`lambda`	numeric value for lambda.
`maxiter_series`	numeric value.
`tol`	numeric value.

Value

returns the F11 value.

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=FALSE))

# 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
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=FALSE))

summary(mod2)

# 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"))
```

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 AJ, 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=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

# 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(1235)
datos <- gendat(n=150)

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

summary(mod2)
```

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

Description

The function `HYPERPO2()` 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

```
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 AJ, 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=200, mu=3, sigma=0.5)

# Fitting the model
library(gamlss)
mod1 <- gamlss(y~1, sigma.fo=~1, family=HYPERPO2,
               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


# 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(1234)
datos <- gendat(n=500)

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

summary(mod2)
```

`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 AJ, 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)
```

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, one-parameter discrete 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 MS, others (2022). “Poisson XLindley distribution for count data: statistical and reliability properties with estimation techniques and inference.” *Computational Intelligence and Neuroscience*, **2022**.

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)
```

Index