Package 'RMKdiscrete'

January 20, 2025

Version 0.2

Date 2022-05-07

Title Sundry Discrete Probability Distributions

Author Robert M. Kirkpatrick <rkirkpatrick2@vcu.edu>

Maintainer Robert M. Kirkpatrick <rkirkpatrick2@vcu.edu>

Description Sundry discrete probability distributions and helper functions.

Depends stats, R (>= 2.15.0)

NeedsCompilation yes

Repository CRAN

License GPL (>= 2)

Date/Publication 2022-05-07 20:30:02 UTC

Contents

`RMKdiscrete-package` *Sundry discrete probability distributions.*

Description

RMKdiscrete implements several univariate and bivariate discrete probability distributions:

- The univariate Lagrangian Poisson distribution has five functions: `dLGP()`, `pLGP()`, `qLGP()`, `rLGP()`, `sLGP()`, `LGP.findmax()`, `LGP.get.nc()`, and `LGPMVP()`.
- The bivariate Lagrangian Poisson distribution has three functions: `dbiLGP()`, `rbiLGP()`, and `biLGP.logMV()`.
- The bivariate negative binomial distribution has three functions: `dbinegbin()`, `rbinegbin()`, and `binegbin.logMV()`.
- Although the univariate negative binomial distribution is implemented in base R, *RMKdiscrete* provides two helper functions for it: `dnegbin()` and `negbinMVP()`.

Finally, the `ManaClash` distributions are provided just for fun.

The package is presently in an unoptimized but functional "beta" state. Additional helper functions and distributions, including multivariate distributions in more than two dimensions, are planned for subsequent versions of the package. Contact the maintainer, <rkirkpatrick2@vcu.edu>, with suggestions, bug reports, and feature requests.

Details

Package:	RMKdiscrete
Version:	0.1
Date:	2014/10/17
Depends:	R (>= 2.15.0), stats
License:	GPL (>= 2)

Author(s)

Robert M. Kirkpatrick <rkirkpatrick2@vcu.edu> Maintainer: Robert M. Kirkpatrick

---

`biLGP` *The bivariate Lagrangian Poisson (LGP) distribution*

Description

Density, random-number generation, and moments of the log-transformed distribution.

Usage

```
dbiLGP(y, theta, lambda, nc=NULL, log=FALSE, add.carefully=FALSE)
biLGP.logMV(theta,lambda,nc=NULL,const.add=1,tol=1e-14,add.carefully=FALSE)
rbiLGP(n, theta, lambda)
```

Arguments

y	Numeric vector or two-column matrix of bivariate data. If matrix, each row corresponds to an observation.
theta	Numeric vector or three-column matrix of non-negative values for index parameters θ_0, θ_1, and θ_2, in that order. If matrix, is read by row.
lambda	Numeric vector or three-column matrix of values for multiplicative parameters λ_0, λ_1, and λ_2, in that order. If matrix, is read by row. Values must be on the interval [-1,1].
nc	Numeric vector or three-column matrix of (reciprocals of) the normalizing constants. These constants differ from 1 only if the corresponding lambda parameter is negative; see dLGP() for details. If matrix, is read by row. Defaults to NULL, in which case the normalizing constants are computed automatically.
log	Logical; should the natural log of the probability be returned? Defaults to FALSE.
add.carefully	Logical. If TRUE, the program takes extra steps to try to prevent round-off error during the addition of probabilities. Defaults to FALSE, which is recommended, since using TRUE is slower and rarely makes a noticeable difference in practice.
const.add	Numeric vector of positive constants to add to the non-negative integers before taking their natural logarithm. Defaults to 1, for the typical $\log(y+1)$ transformation.
tol	Numeric; must be positive. When biLGP.logMV() is calculating the second moment of the log-transformed distribution, it stops when the next term in the series is smaller than tol.
n	Integer; number of observations to be randomly generated.

Details

The bivariate LGP is constructed from three independent latent random variables, X_0, X_1, and X_2, where

$$X_0 \sim LGP(\theta_0, \lambda_0)$$

$$X_1 \sim LGP(\theta_1, \lambda_1)$$

$$X_2 \sim LGP(\theta_2, \lambda_2)$$

The observable variables, Y_1 and Y_2, are defined as $Y_1 = X_0 + X_1$ and $Y_2 = X_0 + X_2$, and thus the dependence between Y_1 and Y_2 arises because of the common term X_0. The joint PMF of Y_1 and Y_2 is derived from the joint PMF of the three independent latent variables, with X_1 and X_2 re-expressed as $Y_1 - X_0$ and $Y_2 - X_0$, and after X_0 is marginalized out.

Function dbiLGP() is the bivariate LGP density (PMF). Function rbiLGP() generates random draws from the bivariate LGP distribution, via calls to rLGP(). Function biLGP.logMV() numerically computes the means, variances, and covariance of a bivariate LGP distribution, after it has been log transformed following addition of a positive constant.

Vectors of numeric arguments other than tol are cycled, whereas only the first element of logical and integer arguments is used.

Value

`dbiLGP()` returns a numeric vector of probabilities. `rbiLGP()` returns a matrix of random draws, which is of type 'numeric' (rather than 'integer', even though the bivariate LGP only has support on the non-negative integers). `biLGP.logMV()` returns a numeric matrix with the following five named columns:

1. `EY1`: Post-tranformation expectation of Y_1.
2. `EY2`: Post-tranformation expectation of Y_2.
3. `VY1`: Post-tranformation variance of Y_1.
4. `VY2`: Post-tranformation variance of Y_2.
5. `COV`: Post-tranformation covariance of Y_1 and Y_2.

Author(s)

Robert M. Kirkpatrick <rkirkpatrick2@vcu.edu>

References

Famoye, F., & Consul, P. C. (1995). Bivariate generalized Poisson distribution with some applications. *Metrika*, 42, 127-138.

Consul, P. C., & Famoye, F. (2006). *Lagrangian Probability Distributions*. Boston: Birkhauser.

See Also

LGP, dpois()

Examples

```
## The following two lines do the same thing:
dbiLGP(y=1,theta=1,lambda=0.1)
dbiLGP(y=c(1,1),theta=c(1,1,1),lambda=c(0.1,0.1,0.1))

dbiLGP(y=c(1,1,2,2,3,5),theta=c(1,1,1,2,2,2),lambda=0.1)
## Due to argument cycling, the above line is doing the following three steps:
dbiLGP(y=c(1,1),theta=c(1,1,1),lambda=c(0.1,0.1,0.1))
dbiLGP(y=c(2,2),theta=c(2,2,2),lambda=c(0.1,0.1,0.1))
dbiLGP(y=c(3,5),theta=c(1,1,1),lambda=c(0.1,0.1,0.1))

## Inputs to dbiLGP() can be matrices, too:
dbiLGP(y=matrix(c(1,1,2,2,3,5),ncol=2,byrow=TRUE),
  theta=matrix(c(1,1,1,2,2,2,1,1,1),ncol=3,byrow=TRUE),
  lambda=0.1)

## theta0 = 0 implies independence:
a <- dbiLGP(y=c(1,3),theta=c(0,1,2),lambda=c(0.1,-0.1,0.5))
b <- dLGP(x=1,theta=1,lambda=-0.1) * dLGP(x=3,theta=2,lambda=0.5)
a-b #<--near zero.
## lambdas of zero yield the ordinary Poisson:
a <- dbiLGP(y=c(1,3), theta=c(0,1,2),lambda=0)
```

```
b <- dpois(x=1,lambda=1) * dpois(x=3,lambda=2) #<--LGP theta is pois lambda
a-b #<--near zero

( y <- rbiLGP(10,theta=c(1.1,0.87,5.5),lambda=c(0.87,0.89,0.90)) )
dbiLGP(y=y,theta=c(1.1,0.87,5.5),lambda=c(0.87,0.89,0.90))

biLGP.logMV(theta=c(0.65,0.35,0.35),lambda=0.7,tol=1e-8)
```

---

binegbin *The bivariate negative binomial distribution*

Description

Functions for the bivariate negative binomial distribution, as generated via trivariate reduction: density, random-number generation, and moments of the log-transformed distribution.

Usage

```
dbinegbin(y, nu, p, log=FALSE, add.carefully=FALSE)
binegbin.logMV(nu,p,const.add=1,tol=1e-14,add.carefully=FALSE)
rbinegbin(n, nu, p)
```

Arguments

y	Numeric vector or two-column matrix of bivariate data. If matrix, each row corresponds to an observation.
nu	Numeric vector or three-column matrix of non-negative values for index parameters ν_0, ν_1, and ν_2, in that order. If matrix, is read by row.
p	Numeric vector or three-column matrix of values for Bernoulli parameters p_0, p_1, and p_2, in that order. If matrix, is read by row. Values must be on the interval (0,1].
log	Logical; should the natural log of the probability be returned? Defaults to FALSE.
add.carefully	Logical. If TRUE, the program takes extra steps to try to prevent round-off error during the addition of probabilities. Defaults to FALSE, which is recommended, since using TRUE is slower and rarely makes a noticeable difference in practice.
const.add	Numeric vector of positive constants to add to the non-negative integers before taking their natural logarithm. Defaults to 1, for the typical $\log(y+1)$ transformation.
tol	Numeric; must be positive. When binegbin.logMV() is calculating the second moment of the log-transformed distribution, it stops when the next term in the series is smaller than tol.
n	Integer; number of observations to be randomly generated.

Details

This bivariate negative binomial distribution is constructed from three independent latent variables, in the same manner as the bivariate Lagrangian Poisson distribution.

Function `dbinegbin()` is the bivariate negative binomial density (PMF). Function `rbinegbin()` generates random draws from the bivariate negative binomial distribution, via calls to `rnbinom()`. Function `binegbin.logMV()` numerically computes the means, variances, and covariance of a bivariate LGP distribution, after it has been log transformed following addition of a positive constant.

Vectors of numeric arguments other than `tol` are cycled, whereas only the first element of logical and integer arguments is used.

Value

`dbinegbin()` returns a numeric vector of probabilities. `rbinegbin()` returns a matrix of random draws, which is of type 'numeric' (rather than 'integer', even though the negative binomial only has support on the non-negative integers). `binegbin.logMV()` returns a numeric matrix with the following five named columns:

1. `EY1`: Post-tranformation expectation of Y_1.
2. `EY2`: Post-tranformation expectation of Y_2.
3. `VY1`: Post-tranformation variance of Y_1.
4. `VY2`: Post-tranformation variance of Y_2.
5. `COV`: Post-tranformation covariance of Y_1 and Y_2.

Author(s)

Robert M. Kirkpatrick <rkirkpatrick2@vcu.edu>

See Also

`dbiLGP`, `dnbinom()`, `rnbinom()`

Examples

```
## The following two lines do the same thing:
dbinegbin(y=1,nu=1,p=0.9)
dbinegbin(y=c(1,1),nu=c(1,1,1),p=c(0.9,0.9,0.9))

dbinegbin(y=c(1,1,2,2,3,5),nu=c(1,1,1,2,2,2),p=0.9)
## Due to argument cycling, the above line is doing the following three steps:
dbinegbin(y=c(1,1),nu=c(1,1,1),p=c(0.9,0.9,0.9))
dbinegbin(y=c(2,2),nu=c(2,2,2),p=c(0.9,0.9,0.9))
dbinegbin(y=c(3,5),nu=c(1,1,1),p=c(0.9,0.9,0.9))

## Inputs to dbinegbin() can be matrices, too:
dbinegbin(y=matrix(c(1,1,2,2,3,5),ncol=2,byrow=TRUE),
  nu=matrix(c(1,1,1,2,2,2,1,1,1),ncol=3,byrow=TRUE),
  p=0.9)

## nu0 = 0 implies independence:
```

```
a <- dbinegbin(y=c(1,3),nu=c(0,1,2),p=c(0.1,0.5,0.9))
b <- dnegbin(x=1,nu=1,p=0.5) * dnegbin(x=3,nu=2,p=0.9)
a-b #<--near zero.

( y <- rbinegbin(10,nu=c(1.1,0.87,5.5),p=c(0.87,0.89,0.90)) )
dbinegbin(y=y,nu=c(1.1,0.87,5.5),p=c(0.87,0.89,0.90))
( mv <- negbinMVP(nu=c(1.1,0.87,5.5),p=c(0.87,0.89,0.90)) )
mv[1,2] #<--Covariance of this distribution
mv[1,2]+mv[2,2] #<--Marginal variance of Y1
mv[1,2]+mv[3,2] #<--Marginal variance of Y2
mv[1,2]/(sqrt(mv[1,2]+mv[2,2])*sqrt(mv[1,2]+mv[3,2])) #<--Correlation
logmv <- binegbin.logMV(nu=c(1.1,0.87,5.5),p=c(0.87,0.89,0.90))
## Log transformation nearly cuts the correlation in half:
logmv[1,5]/sqrt(logmv[1,3]*logmv[1,4])
```

LGP *The (univariate) Lagrangian Poisson (LGP) Distribution*

Description

Density, distribution function, quantile function, summary, random number generation, and utility functions for the (univariate) Lagrangian Poisson distribution.

Usage

```
dLGP(x,theta,lambda,nc=NULL,log=FALSE)
pLGP(q,theta,lambda,nc=NULL,lower.tail=TRUE,log.p=FALSE,add.carefully=FALSE)
qLGP(p,theta,lambda,nc=NULL,lower.tail=TRUE,log.p=FALSE,add.carefully=FALSE)
rLGP(n,theta,lambda)
sLGP(theta,lambda,nc=NULL,do.numerically=FALSE,add.carefully=FALSE)
LGP.findmax(theta,lambda)
LGP.get.nc(theta,lambda,nctol=1e-14,add.carefully=FALSE)
LGPMVP(mu,sigma2,theta,lambda)
```

Arguments

x, q	Numeric vector of quantiles.
p	Numeric vector of probabilities.
n	Integer; number of observations to be randomly generated.
theta	Numeric; the index (or "additive") parameter of the LGP distribution. Must be non-negative.
lambda	Numeric; the dispersion (or "Lagrangian" or "multiplicative") parameter of the LGP distribution. Must not exceed 1 in absolute value. When equal to zero, the LGP reduces to the ordinary Poisson distribution, with mean equal to theta. When negative, then the distribution has an upper limit to its support, which may be found with LGP.findmax()

.

nc	Numeric; the reciprocal of the normalizing constant of the distribution, by which the raw PMF must be multiplied so that it is a proper PMF, with values that sum to 1 across the support, when lambda is negative. Defaults to NULL, in which case it is computed numerically by a call to LGP.get.nc().
log, log.p	Logical; if TRUE, then probabilities p are given as log(p).
lower.tail	Logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
nctol	Numeric; while numerically computing the normalizing constant, how close to 1 should it be before stopping? Ignored unless lambda is negative, and the upper support limit exceeds 200,000.
add.carefully	Logical. If TRUE, the program takes extra steps to try to prevent round-off error during the addition of probabilities. Defaults to FALSE, which is recommended, since using TRUE is slower and rarely makes a noticeable difference in practice.
do.numerically	Logical; should moments be computed numerically when lambda<0? Defaults to FALSE, which is recommended unless the upper support limit is fairly small (say, less than 10).
mu	Numeric vector of mean parameters.
sigma2	"Sigma squared"–numeric vector of variance parameters.

Details

The Lagrangian Poisson (LGP) distribution has density

$$p(x) = \frac{\theta(\theta + \lambda x)^{x-1} \exp(-\theta - \lambda x)}{x!}$$

for $0, 1, 2, \ldots$,

$$p(x) = 0$$

for $x > m$ if $\lambda < 0$, and zero otherwise, where $\theta > 0$, $m = \lfloor -\theta/\lambda \rfloor$ if $\lambda < 0$, and $\max(-1, -\theta/m) \leq \lambda \leq 1$. So, when λ is negative, there is an upper limit to the distribution's support, m, equal to $-\theta/\lambda$, rounded down to the next-smallest integer. When λ is negative, the PMF must also be normalized numerically if it is to describe a proper probability distribution. When $\lambda = 0$, the Lagrangian Poisson reduces to the ordinary Poisson, with mean equal to θ. When $\theta = 0$, we define the distribution as having unit mass on the event $X = 0$.

Function LGP.findmax() calculates the value of upper support limit m; LGP.get.nc() calculates the (reciprocal of) the normalizing constant.

Function LGPMVP() accepts exactly two of its four arguments, and returns the corresponding values of the other two arguments. For example, if given values for theta and lambda, it will return the corresponding means (mu) and variances (sigma2) of an LGP distribution with the given values of θ and λ. LGPMVP() does not enforce the parameter space as strictly as other functions, but will throw a warning for bad parameter values.

When the upper support limit is 5 or smaller, rLGP() uses simple inversion (i.e., random unit-uniform draws passed to qLGP()). Otherwise, it uses random-number generation algorithms from Consul & Famoye (2006); exactly which algorithm is used depends upon the values of theta and lambda. All four of rLGP(), dLGP(), pLGP(), and qLGP() make calls to the corresponding functions for the ordinary Poisson distribution (dpois(),etc.) when lambda=0.

Vectors of numeric arguments are cycled, whereas only the first element of logical and integer arguments is used.

Value

dLGP() and pLGP() return numeric vectors of probabilities. qLGP(), rLGP(), and LGP.findmax() return vectors of quantiles, which are of class 'numeric' rather than 'integer' for the sake of compatibility with very large values. LGP.get.nc() returns a numeric vector of reciprocal normalizing constants. LGPMVP() returns a numeric matrix with two columns, named for the missing arguments in the function call.

sLGP() returns a numeric matrix with 10 columns, with the mostly self-explanatory names "Mean", "Median", "Mode", "Variance", "SD", "ThirdCentralMoment", "FourthCentralMoment", "PearsonsSkewness", "Skewness", and "Kurtosis". Here, "Kurtosis" refers to excess kurtosis (greater than 3), and "PearsonsSkewness" equals $\frac{(mean-mode)}{SD}$. A "Mode" of 0.5 indicates that the point probabilities at $x = 0$ and $x = 1$ are tied for highest density; other than this possibility, the LGP is strictly unimodal.

Warning

There is a known issue with sLGP(): when lambda is negative and theta is large, the third and fourth moments returned by sLGP(), with do.numerically=TRUE, can be quite incorrect due to numerical imprecision.

Author(s)

Robert M. Kirkpatrick <rkirkpatrick2@vcu.edu>

References

Consul, P. C. (1989). *Generalized Poisson Distributions: Properties and Applications*. New York: Marcel Dekker, Inc.

Consul, P. C., & Famoye, F. (2006). *Lagrangian Probability Distributions*. Boston: Birkhauser.

Johnson, N. L., Kemp, A. W., & Kotz, S. (2005). *Univariate Discrete Distributions* (3rd. ed.). Hoboken, NJ: John Wiley & Sons, Inc.

Examples

```
LGP.findmax(theta=2, lambda=0.2) #<--No upper support limit
LGP.findmax(theta=2, lambda=-0.2) #<--Upper support limit of 9
LGP.get.nc(theta=2, lambda=0.2)-1==0 #<--TRUE
LGP.get.nc(theta=2, lambda=-0.2)-1 #<--nc differs appreciably from 1
LGP.get.nc(theta=2, lambda=-0.1)-1 #<--nc doesn't differ appreciably from 1
LGPMVP(theta=2, lambda=0.9)
LGPMVP(mu=20, sigma2=2000)
sLGP(theta=2, lambda=0.9)
dLGP(x=0:10,theta=1,lambda=0.1)
dLGP(x=0:10,theta=1,lambda=0)
dLGP(x=0:10,theta=1,lambda=-0.1) #<--Upper support limit of 9
pLGP(q=0:10,theta=1,lambda=0.1)
pLGP(q=0:10,theta=1,lambda=0)
pLGP(q=0:10,theta=1,lambda=-0.1)
qLGP(p=(0:9)/10,theta=1,lambda=0.1)
qLGP(p=(0:9)/10,theta=1,lambda=0)
```

```
qLGP(p=(0:9)/10,theta=1,lambda=-0.1)
rLGP(n=5,theta=1e12,lambda=-0.0001)
```

ManaClash *The Mana Clash distributions (just for fun!)*

Description

Density and random-number functions for distributions pertinent to "Mana Clash", a card from the *Magic: The Gathering* trading-card game. As of 08/29/2014, the official card text read: "You and target opponent each flip a coin. Mana Clash deals 1 damage to each player whose coin comes up tails. Repeat this process until both players' coins come up heads on the same flip."

Usage

```
dmanaclash.dmg(x,y,N=NULL,pA=0.25,pB=0.25,pC=0.25,pD=0.25,log=FALSE)
dmanaclash.xyN(x,y,N,pA=0.25,pB=0.25,pC=0.25,pD=0.25,log=FALSE)
dmanaclash.net(z,pA=0.25,pB=0.25,pC=0.25,pD=0.25,rel.eps=1e-8,log=FALSE)
rmanaclash(n,pA=0.25,pB=0.25,pC=0.25,pD=0.25,N=NULL)
```

Arguments

x	Numeric amount of damage dealt to opponent.
y	Numeric amount of damage dealt to Mana Clash's controller (hereinafter, "you").
N	Numeric. Number of rounds of coin-tossing that *precede* the last round, when double heads occurs. That is, N is the number of rounds of coin-tossing in which at least one player takes damage.
z	Numeric; *net* damage dealt to opponent; negative values are allowed.
n	Integer number of random vectors to generate.
pA	Numeric; probability that both players take damage in a round of coin-tossing (i.e., double tails).
pB	Numeric; probability that you take damage but opponent does not in a round of coin-tossing.
pC	Numeric; probability that opponent takes damage but you do not in a round of coin-tossing.
pD	Numeric; probability that neither player takes damage in a round of coin-tossing (i.e., double heads).
log	Logical; should the natural log of the probability be returned? Defaults to FALSE.
rel.eps	Numeric; when computing the sum of an infinite series, how small should the relative change in the sum get before stopping?

Details

The probability arguments–pA, pB, pC, and pD–are named as in a two-way contingency table. They cannot be negative, although rmanaclash accepts values of zero for pA, pB, and pC. If they do not sum to 1, they are automatically normalized. They default to the scenario of two independent fair coins.

Vectors of numeric arguments other than rel.eps are cycled, whereas only the first element of logical and integer arguments is used.

Function dmanaclash.dmg() is the bivariate PMF of the amount of damage dealt to opponent and you. If N = NULL (default), the probabilities are marginal with respect to the number of rounds of damage-dealing. Otherwise, the probabilities are conditioned upon the given value of N.

Function dmanaclash.xyN() is the trivariate joint PMF of the amount of damage dealt to opponent, the amount dealt to you, and the number of rounds of damage-dealing.

Function dmanaclash.net is the univariate PMF of the *net* amount of damage dealt to opponent, i.e. damage dealt to opponent minus damage dealt to you. This distribution has support on the set of integers–including negative values.

Function rmanaclash() generates random draws from the trivariate joint distribution of x, y, and N; if a non-NULL value for N is supplied, the random draws are generated conditionally on that number of damage-dealing rounds.

Value

dmanaclash.dmg(), dmanaclash.xyN(), and dmanaclash.net() all return numeric vectors of probabilities. rmanaclash() returns a numeric matrix, with n rows, and three columns, named "x", "y", and "N". Each row is a random draw.

Derivation

Note: This section is only displayed in the PDF of the package documentation.

Let random variables X, Y, and N respectively denote the amount of damage dealt to opponent, the amount of damage dealt to you, and the number of rounds of coin-tossing in which any damage is dealt. In a given round of coin-tossing, let a denote the probability of damage to both players (i.e., two tails), b, the probability of damage to you but not opponent, c, the probability of damage to opponent but not you, and d the probability of no damage (i.e., two heads, and no more coin-tossing). Define $r = a + b + c = 1 - d$.

It is obvious that the marginal distribution of N is geometric, with PMF

$$p(n) = dr^n$$

Further, given that $N = n$, the conditional distribution of X is binomial, with PMF

$$p(x|n) = \left(\frac{n!}{(n-x)!x!}\right)\left(\frac{a+c}{r}\right)^x\left(\frac{b}{r}\right)^{n-x}$$

Now, consider Y, given $N = n$ and $X = x$. Of course, Y cannot exceed n. On any of the n rounds in which the opponent took no damage, you must have taken damage, and you may or may not have taken damage on those rounds in which opponent did. Thus, Y cannot be less than $n - x$, and will equal $n - x$ only if exactly one player took damage on all n rounds (i.e., double tails

never occurred). On the x undetermined rounds, you take damage with probability $\frac{a}{a+c}$, and do not take damage with probability $\frac{c}{a+c}$. Thus, the random variable $Y-(n-x)$ follows a binomial distribution, with PMF

$$p(y|x,n)=\left(\frac{x!}{(n-y)!(y-n+x)!}\right)\left(\frac{a}{a+c}\right)^{x+y-n}\left(\frac{c}{a+c}\right)^{n-y}$$

The joint distribution of X and Y, conditional on $N=n$, has PMF

$$p(x,y|n)=p(x|n)p(y|x,n)$$

which is the product of two binomial PMFs. The joint distribution of X, Y, and N has PMF

$$p(x,y,n)=p(n)p(x|n)p(y|x,n)$$

which after some algebraic manipulation can be written as

$$p(x,y,n)=\frac{n!}{(n-x)!(n-y)!(x+y-n)!}b^{n-x}c^{n-y}a^{x+y-n}d$$

The joint distribution of X and Y, marginal with respect to N, then has PMF

$$p(x,y)=\frac{da^{x+y}}{b^xc^y}\cdot\sum_{n=\max(x,y)}^{x+y}\left[\left(\frac{bc}{a}\right)^n\left(\frac{n!}{(n-x)!(n-y)!(x+y-n)!}\right)\right]$$

Now, define random variable $Z=X-Y$, i.e. the *net* amount of damage dealt to opponent. The PMF of Z may be obtained from $p(x,y)$, the joint PMF of X and Y:

$$p(z)=\sum_{\{(x,y):x-y=z\}}p(x,y)$$

That is, the probability that $Z=z$ is the sum of the probabilities of all pairs (x,y) such that $x-y=z$. For example, the probability that $Z=-2$ is the sum $p(0,2)+p(1,3)+p(2,4)+...$

Author(s)

Robert M. Kirkpatrick <rkirkpatrick2@vcu.edu>

References

Magic: The Gathering is a trademark of Wizards of the Coast, LLC, a subsidiary of Hasbro, Inc.

Examples

```
## Same outcome, with and without conditioning on N:
dmanaclash.dmg(x=1,y=1,N=1)
dmanaclash.dmg(x=1,y=1)

## Same damage amounts, with N fixed versus random:
dmanaclash.dmg(x=1,y=1,N=2)
dmanaclash.xyN(x=1,y=1,N=2)
```

```
## Net damage distribution is symmetric with defaults:
dmanaclash.net(z=c(-3,-2,-1,0,1,2,3))
## But if coins are biased against opponent...:
dmanaclash.net(z=c(-3,-2,-1,0,1,2,3),pA=0.1,pB=0.1,pC=0.7,pD=0.1)

## Random draws:
rmanaclash(n=10)
rmanaclash(n=10,pA=0.1,pB=0.1,pC=0.7,pD=0.1)
rmanaclash(n=10,N=5)
```

`negbin` *Helper functions for the (unvariate) negative binomial distribution.*

Description

Helper functions for the (unvariate) negative binomial distribution: change-of-parameter, wrapper function for density.

Usage

```
dnegbin(x,nu,p,mu,log=FALSE)
negbinMVP(nu, p, mu, sigma2)
```

Arguments

`x`	Numeric vector of quantiles.
`nu`	Numeric; equivalent to argument `size` in `dnbinom()`, etc.
`p`	Numeric; equivalent to argument `prob` in `dnbinom()`, etc.
`mu`	Numeric vector of mean parameters.
`sigma2`	"Sigma squared"–numeric vector of variance parameters.
`log`	Logical; should the natural log of the probability be returned? Defaults to `FALSE`.

Details

Function `dnegbin()` is a wrapper for `dnbinom()`. Two of the three arguments `nu`, `p`, and `mu` must be provided. Unlike `dnbinom()`, `dnegbin()` will accept `mu` and `p`(`prob`) with `nu`(`size`) missing. In that case, it calculates `nu` as `mu * p/(1-p)`, and passes `nu` and `p` to `dnbinom()`.

Function `negbinMVP()` accepts exactly two of its four arguments, and returns the corresponding values of the other two arguments. For example, if given values for `nu` and `p`, it will return the corresponding means (`mu`) and variances (`sigma2`) of a negative binomial distribution with the given values of `nu` and `p`. `negbinMVP()` does not strictly enforce the parameter space, but will throw a warning for bad input values.

Value

`dnegbin()` returns a numeric vector of probabilities. `negbinMVP()` returns a numeric matrix with two columns, named for the missing arguments in the function call.

Author(s)

Robert M. Kirkpatrick <rkirkpatrick2@vcu.edu>

See Also

dnbinom()

Examples

```
## These two lines do the same thing:
dnegbin(x=1,nu=2,p=0.5)
dnbinom(x=1,size=2,prob=0.5)

## What is the mean of this distribution?
negbinMVP(nu=2,p=0.5) #<--mu=2

## These two lines also do the same thing:
dnegbin(x=1,nu=2,mu=2)
dnbinom(x=1,size=2,mu=2)

## Parametrize with mu & p:
dnegbin(x=1,mu=2,p=0.5)
## Not run (will throw an error):
## dnbinom(x=1,prob=0.5,mu=2)
```

Index