Package ‘betafunctions’

April 12, 2022

Type Package

Title Functions for Working with Two- And Four-Parameter Beta Probability Distributions

Version 1.7.0

Author Haakon Eidem Haakstad

Maintainer Haakon Eidem Haakstad <h.e.haakstad@gmail.com>

Description Package providing a number of functions for working with Two- and Four-parameter Beta and closely related distributions (i.e., the Gamma-Binomial-, and Beta-Binomial distributions), including parameterization in terms of moments, and fitting of Beta distributions to vectors of values. Includes d/p/r- and a function and for calculating moments of Beta-Binomial distributions. Also includes functions for estimating classification accuracy, diagnostic performance, and consistency, making use of what is generally known as the 'Livingston and Lewis approach' in the psychometric literature which models observed-score distributions in terms of the Beta-Binomial distribution. A shiny app is available, providing a GUI for the Livingston and Lewis approach when used for binary classifications. For url to the app, see documentation for the LL.CA() function.
Livingston and Lewis (1995) <doi:10.1111/j.1745-3984.1995.tb00462.x>.
Lord (1965) <doi:10.1007/BF02289490>.
Hanson (1991) <https://files.eric.ed.gov/fulltext/ED344945.pdf>.
Tharwat (2020) <doi:10.1016/j.aci.2018.08.003>.

Encoding UTF-8

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2022-04-11 23:22:28 UTC

R topics documented:

afac 3

AMS . . . 4
AUC . . . 5
Beta.2p.fit . . . 6
Beta.4p.fit . . . 7
Beta.gfx.poly.cdf . . . 8
Beta.gfx.poly.pdf . . . 9
Beta.gfx.poly.qdf . . . 10
Beta.tp.fit . . . 11
betabinomialmoments . . . 13
betamedian . . . 14
betamode . . . 15
betamoments . . . 15
binomialmoments . . . 17
BMS . . . 18
caStats . . . 19
cba . . . 20
ccStats . . . 21
confmat . . . 22
dBeta.4P . . . 23
dBeta.pBeta . . . 24
dBeta.pBinom . . . 25
dBeta.pGammaBinom . . . 26
dBetaBinom . . . 28
dBetaMS . . . 28
dfac . . . 29
dGammaBinom . . . 30
ETL . . . 31
gchoose . . . 32
LABMSU . . . 32
LL.CA . . . 34
LL.CA.MC . . . 37
LL.ROC . . . 40
MC.out.tabular . . . 42
mdo . . . 43
MLA . . . 44
MLB . . . 45
MLM . . . 45
observedmoments . . . 46
pBeta.4P . . . 47
pBetaBinom . . . 48
pBetaMS . . . 49
pGammaBinom . . . 50
qBeta.4P . . . 50
qBetaMS . . . 51
qGammaBinom . . . 52
R.ETL . . . 53
rBeta.4P . . . 54
rBetaBinom . . . 55

rBetaMS 55
rGammaBinom 56
tsm 57
UABMSL 58

Index **60**

`afac` *Ascending (rising) factorial.*

Description

Calculate the ascending (or rising) factorial of a value `x` of order `r`.

Usage

```
afac(x, r, method = "product")
```

Arguments

`x`	A value for which the ascending factorial is to be calculated.
`r`	The power `x` is to be raised to.
`method`	The method by which the descending factorials are to be calculated. Default is `"product"` which uses direct arithmetic. Alternative is `"gamma"` which calculates the descending factorial using the Gamma function. The alternative method might be faster but might fail because the Gamma function is not defined for negative integers (returning Inf).

Value

The ascending factorial of value `x` raised to the `r`'th power.

Examples

```
# To calculate the 4th ascending factorial for a value (e.g., 3.14):
afac(x = 3.14, r = 4)

# To calculate the 5th ascending factorial for values 3.14, 2.72, and 0.58:
afac(x = c(3.14, 2.72, 0.58), r = 5)
```

AMS *Alpha Shape-Parameter Given Location-Parameters, Mean, Variance, Skewness, Kurtosis and Beta Shape-Parameter of a Four-Parameter Beta PDD.*

Description

Calculates the Beta value required to produce a Beta probability density distribution with defined moments and parameters. Be advised that not all combinations of moments and parameters can be satisfied (e.g., specifying mean, variance, skewness and kurtosis uniquely determines both location-parameters, meaning that the value of the lower-location parameter will take on which ever value it must, and cannot be specified).

Usage

```
AMS(
  mean = NULL,
  variance = NULL,
  skewness = NULL,
  kurtosis = NULL,
  l = 0,
  u = 1,
  beta = NULL,
  sd = NULL
)
```

Arguments

`mean`	The mean (first raw moment) of the target Standard Beta probability density distribution.
`variance`	The variance (second central moment) of the target Standard Beta probability density distribution.
`skewness`	The skewness (third standardized moment) of the target Beta probability density distribution.
`kurtosis`	The kurtosis (fourth standardized moment) of the target Beta probability density distribution.
`l`	The lower-bound of the Beta distribution. Default is 0 (i.e., the lower-bound of the Standard, two-parameter Beta distribution).
`u`	The upper-bound of the Beta distribution. Default is 1 (i.e., the upper-bound of the Standard, two-parameter Beta distribution).
`beta`	Optional specification of the Beta shape-parameter of the target Beta distribution. Finds then the Alpha parameter necessary to produce a distribution with the specified mean, given specified Beta, l, and u parameters.
`sd`	Optional alternative to specifying `var`. The standard deviation of the target Standard Beta probability density distribution.

Value

A numeric value representing the required value for the Alpha shape-parameter in order to produce a Beta probability density distribution with the target mean and variance, given specified lower- and upper bounds of the Beta distribution.

Examples

```
# Generate some fictional data. Say, 100 individuals take a test with a
# maximum score of 100 and a minimum score of 0, rescaled to proportion
# of maximum.
set.seed(1234)
testdata <- rbinom(100, 100, rBeta.4P(100, 0.25, 0.75, 5, 3)) / 100
hist(testdata, xlim = c(0, 1))

# To find the alpha shape-parameter of a Standard (two-parameter) Beta
# distribution with the same mean and variance as the observed-score
# distribution using AMS():
AMS(mean(testdata), var(testdata))
```

AUC — *Area Under the ROC Curve.*

Description

Given a vector of false-positive rates and a vector of true-positive rates, calculate the area under the Receiver Operator Characteristic (ROC) curve.

Usage

```
AUC(FPR, TPR)
```

Arguments

FPR	Vector of False-Positive Rates.
TPR	Vector of True-Positive Rates.

Value

A value representing the area under the ROC curve.

Note

Script originally retrieved and modified from https://blog.revolutionanalytics.com/2016/11/calculating-auc.html.

Examples

```
# Generate some fictional data. Say, 100 individuals take a test with a
# maximum score of 100 and a minimum score of 0.
set.seed(1234)
testdata <- rbinom(100, 100, rBeta.4P(100, 0.25, 0.75, 5, 3))
hist(testdata, xlim = c(0, 100))

# Suppose the cutoff value for attaining a pass is 50 items correct, and
# that the reliability of this test was estimated to 0.7. To calculate the
# necessary (x, y) coordinates to compute the area under the curve statistic
# one can use the LL.ROC() function with the argument
# raw.out = TRUE.
coords <- LL.ROC(x = testdata, reliability = .7, truecut = 50, min = 0,
max = 100, raw.out = TRUE)

# To calculate and retrieve the Area Under the Curve (AUC) with the AUC()
# function, feed it the raw coordinates calculated above.
AUC(coords[, "FPR"], coords[, "TPR"])
```

Beta.2p.fit — *Method of Moment Estimates of Shape-Parameters of the Two-Parameter (Standard) Beta Distribution.*

Description

An implementation of the method of moments estimation of two-parameter Beta distribution parameters. Given a vector of values, calculates the shape parameters required to produce a two-parameter Beta distribution with the same mean and variance (i.e., the first two moments) as the observed-score distribution.

Usage

```
Beta.2p.fit(scores, mean = NULL, variance = NULL, l = 0, u = 1)
```

Arguments

`scores`	A vector of values to which the two-parameter Beta distribution is to be fitted. The values ought to fall within the [0, 1] interval.
`mean`	The mean of the target Beta distribution. Alternative to feeding the function raw scores.
`variance`	The variance of the target Beta distribution. Alternative to feeding the function raw scores.
`l`	Optional specification of a lower-bound parameter of the Beta distribution. Default is 0 (i.e., the lower-bound of the Standard two-parameter Beta distribution).
`u`	Optional specification of an upper-bound parameter of the Beta distribution. Default is 1 (i.e., the lower-bound of the Standard two-parameter Beta distribution).

Value

A list of parameter-values required to produce a Standard two-parameter Beta distribution with the same first two moments as the observed distribution.

Examples

```
# Generate some fictional data. Say, 100 individuals take a test with a
# maximum score of 100 and a minimum score of 0.
set.seed(1234)
testdata <- rbinom(100, 100, rBeta.4P(100, 0.25, 0.75, 5, 3)) / 100
hist(testdata, xlim = c(0, 1), freq = FALSE)

# To fit and retrieve the parameters for a two-parameter Beta distribution
# to the observed-score distribution using Beta.2p.fit():
(params.2p <- Beta.2p.fit(testdata))
curve(dbeta(x, params.2p$alpha, params.2p$beta), add = TRUE)
```

Beta.4p.fit — *Method of Moment Estimates of Shape- and Location Parameters of the Four-Parameter Beta Distribution.*

Description

An implementation of the method of moments estimation of four-parameter Beta distribution parameters presented by Hanson (1991). Given a vector of values, calculates the shape- and location parameters required to produce a four-parameter Beta distribution with the same mean, variance, skewness and kurtosis (i.e., the first four moments) as the observed-score distribution.

Usage

```
Beta.4p.fit(
  scores,
  mean = NULL,
  variance = NULL,
  skewness = NULL,
  kurtosis = NULL
)
```

Arguments

`scores`	A vector of values to which the four-parameter Beta distribution is to be fitted.
`mean`	If scores are not supplied: specification of the mean for the target four-parameter Beta distribution.
`variance`	If scores are not supplied: specification of the variance for the target four-parameter Beta distribution.
`skewness`	If scores are not supplied: specification of the skewness for the target four-parameter Beta distribution.
`kurtosis`	If scores are not supplied: specification of the kurtosis for the target four-parameter Beta distribution.

Value

A list of parameter-values required to produce a four-parameter Beta distribution with the same first four moments as the observed distribution.

References

Hanson, Bradley A. (1991). Method of Moments Estimates for the Four-Parameter Beta Compound Binomial Model and the Calculation of Classification Consistency Indexes.American College Testing Research Report Series.

Lord, Frederic M. (1965). A Strong True-Score Theory, With Applications. Psychometrika, 30(3).

Examples

```
# Generate some fictional data. Say, 100 individuals take a test with a
# maximum score of 100 and a minimum score of 0.
set.seed(1234)
testdata <- rbinom(100, 100, rBeta.4P(100, 0.25, 0.75, 5, 3))
hist(testdata, xlim = c(0, 100), freq = FALSE)

# To fit and retrieve the parameters for a four-parameter Beta distribution
# to the observed-score distribution using Beta.4p.fit():
(params.4p <- Beta.4p.fit(testdata))
curve(dBeta.4P(x, params.4p$l, params.4p$u, params.4p$alpha, params.4p$beta), add = TRUE)
```

Beta.gfx.poly.cdf — *Coordinate Generation for Marking an Area Under the Curve for the Beta Cumulative Probability Density Distribution.*

Description

Plotting tool, producing a two-column matrix with values of y corresponding to locations on x. Useful for shading areas under the curve when tracing the line for the Standard Beta cumulative probability function.

Usage

```
Beta.gfx.poly.cdf(from, to, by, alpha, beta, l = 0, u = 1)
```

Arguments

from	The point of the x-axis from where to start producing y-density values.
to	The point of the x-axis to where y-density values are to be produced.
by	The resolution (or spacing) at which to produce y-density values.
alpha	The alpha shape-parameter value for the Standard Beta cumulative probability distribution.
beta	The beta shape-parameter for the Standard Beta cumulative probability distribution.

`l`	The lower-bound location parameter of the Beta distribution.
`u`	The upper-bound location parameter of the Beta distribution.

Value

A two-column matrix with cumulative probability-values of y to plot against corresponding location values of x.

Examples

```
# To box in an area under a four-parameter Beta cumulative distribution with
# location parameters l = 0.25 and u = 0.75, and shape parameters
# alpha = 5 and beta = 3, from 0.4 to 0.6:
plot(NULL, xlim = c(0, 1), ylim = c(0, 1))
coords <- Beta.gfx.poly.cdf(from = 0.4, to = 0.6, by = 0.001, alpha = 5,
beta = 3, l = 0.25, u = 0.75)
polygon(coords)
```

Beta.gfx.poly.pdf — *Coordinate Generation for Marking an Area Under the Curve for the Beta Probability Density Distribution.*

Description

Plotting tool, producing a two-column matrix with values of y corresponding to locations on x. Useful for shading areas under the curve when tracing the line for the Standard Beta probability density function.

Usage

```
Beta.gfx.poly.pdf(from, to, by, alpha, beta, l = 0, u = 1)
```

Arguments

`from`	The point of the x-axis from where to start producing y-density values.
`to`	The point of the x-axis to where y-density values are to be produced.
`by`	The resolution (or spacing) at which to produce y-density values.
`alpha`	The alpha (first) shape-parameter value for the Standard Beta probability density distribution.
`beta`	The beta (second) shape-parameter for the Standard Beta probability density distribution.
`l`	The lower-bound location parameter of the Beta distribution.
`u`	The upper-bound location parameter of the Beta distribution.

Value

A two-column matrix with density-values of y to plot against corresponding location values of x.

Examples

```
# To box in an area under a four-parameter Beta distribution with location
# parameters l = .25 and u = .75, and shape parameters alpha = 5 and
# rbeta = 3, from 0.4 to 0.6:
plot(NULL, xlim = c(0, 1), ylim = c(0, 7))
coords <- Beta.gfx.poly.pdf(from = 0.4, to = 0.6, by = 0.001, alpha = 5,
beta = 3, l = 0.25, u = 0.75)
polygon(coords)
```

Beta.gfx.poly.qdf — *Coordinate Generation for Marking an Area Under the Curve for the Beta Quantile Density Distribution.*

Description

Plotting tool, producing a two-column matrix with values of y corresponding to locations on x. Useful for shading areas under the curve when tracing the line for the Standard Beta probability quantile function.

Usage

```
Beta.gfx.poly.qdf(from, to, by, alpha, beta, l = 0, u = 1)
```

Arguments

from	The point of the x-axis from where to start producing y-quantile values.
to	The point of the x-axis to where y-quantile values are to be produced.
by	The resolution (or spacing) at which to produce y-density values.
alpha	The alpha shape-parameter value for the Standard Beta probability distribution.
beta	The beta shape-parameter for the Standard Beta probability distribution.
l	The lower-bound location parameter of the Beta distribution.
u	The upper-bound location parameter of the Beta distribution.

Value

A two-column matrix with quantile-values of y to plot against corresponding location values of x.

Examples

```
# To box in an area under a four-parameter Beta quantile distribution with
# location parameters l = .25 and u = 75, and shape parameters alpha = 5 and
# beta = 3, from .4 to .6:
plot(NULL, xlim = c(0, 1), ylim = c(0, 1))
coords <- Beta.gfx.poly.qdf(from = 0.4, to = 0.6, by = 0.001, alpha = 5,
beta = 3, l = 0.25, u = 0.75)
polygon(coords)
```

`Beta.tp.fit` *Estimate Beta true-score distribution based on observed-score raw-moments and the effective test length.*

Description

Estimator for the Beta true-score distribution shape-parameters from the observed-score distribution and Livingston and Lewis' effective test length. Returns a list with entries representing the lower- and upper shape parameters (l and u), and the shape parameters (alpha and beta) of the four-parameters beta distribution.

Usage

```
Beta.tp.fit(
  x,
  min,
  max,
  etl,
  reliability = NULL,
  true.model = "4P",
  failsafe = FALSE,
  l = 0,
  u = 1,
  alpha = NA,
  beta = NA,
  output = "parameters"
)
```

Arguments

`x`	Vector of observed-scores.
`min`	The minimum possible score to attain on the test.
`max`	The maximum possible score to attain on the test.
`etl`	The value of Livingston and Lewis' effective test length. See `?ETL()`. Not necessary to specify if reliability is supplied to the `reliability` argument.
`reliability`	Optional specification of the test-score reliability coefficient. If specified, overrides the input of the `etl` argument.
`true.model`	The type of Beta distribution which is to be fit to the moments of the true-score distribution. Options are `"4P"` and `"2P"`, where `"4P"` refers to the four-parameter (with the same mean, variance, skewness, and kurtosis), and `"2P"` the two-parameter solution where both location-parameters are specified (with the same mean and variance).
`failsafe`	Logical. Whether to revert to a fail-safe two-parameter solution should the four-parameter solution contain invalid parameter estimates.

l	If failsafe = TRUE or true.model = "2P": The lower-bound of the Beta distribution. Default is 0 (i.e., the lower-bound of the Standard, two-parameter Beta distribution).
u	If failsafe = TRUE or true.model = "2P": The upper-bound of the Beta distribution. Default is 1 (i.e., the upper-bound of the Standard, two-parameter Beta distribution).
alpha	If failsafe = TRUE or true.model = "2P": The alpha shape-parameter of the Beta distribution. Default is NA (i.e., estimate the parameter).
beta	If failsafe = TRUE or true.model = "2P": The beta shape-parameter of the Beta distribution. Default is NA (i.e., estimate the parameter).
output	Option to specify true-score distribution moments as output if the value of the output argument does not equal "parameters".

Value

A list with the parameter values of a four-parameter Beta distribution. "l" is the lower location-parameter, "u" the upper location-parameter, "alpha" the first shape-parameter, and "beta" the second shape-parameter.

References

Hanson, B. A. (1991). Method of Moments Estimates for the Four-Parameter Beta Compound Binomial Model and the Calculation of Classification Consistency Indexes. American College Testing Research Report Series. Retrieved from https://files.eric.ed.gov/fulltext/ED344945.pdf

Lord, F. M. (1965). A strong true-score theory, with applications. Psychometrika. 30(3). pp. 239–270. doi: 10.1007/BF02289490

Rogosa, D. & Finkelman, M. (2004). How Accurate Are the STAR Scores for Individual Students? – An Interpretive Guide. Retrieved from http://statweb.stanford.edu/~rag/accguide/guide04.pdf

Examples

```
# Generate some fictional data. Say 1000 individuals take a 100-item test
# where all items are equally difficult, and the true-score distribution
# is a four-parameter Beta distribution with location parameters l = 0.25,
# u = 0.75, alpha = 5, and beta = 3:
set.seed(12)
testdata <- rbinom(1000, 100, rBeta.4P(1000, 0.25, 0.75, 5, 3))

# Since this test contains items which are all equally difficult, the true
# effective test length (etl) is the actual test length. I.e., etl = 100.
# To estimate the four-parameter Beta distribution parameters underlying
# the draws from the binomial distribution:
Beta.tp.fit(testdata, 0, 100, 100)

# Imagine a case where the fitting procedure produces an impermissible
# estimate (e.g., l < 0 or u > 1).
set.seed(1234)
testdata <- rbinom(1000, 50, rBeta.4P(1000, 0.25, 0.75, 5, 3))
Beta.tp.fit(testdata, 0, 50, 50)
```

```
# This example produced an l-value estimate less than 0. One way of
# dealing with such an occurrence is to revert to a two-parameter
# model, specifying the l and u parameters and estimating the
# alpha and beta parameters necessary to produce a Beta distribution
# with the same mean and variance as the estimated true-score distribution.

# Suppose you have good theoretical reasons to fix the l parameter at a
# value of 0.25 (e.g., the test is composed of multiple-choice questions
# with four response-options, resulting in a 25% chance of guessing the
# correct answer). The l-parameter could be specified to this theoretically
# justified value, and the u-parameter could be specified to be equal to the
# estimate above (u = 0.7256552) as such:
Beta.tp.fit(testdata, 0, 50, 50, true.model = "2P", l = 0.25, u = 0.7256552)
```

---

betabinomialmoments *Compute Moments of Beta-Binomial Probability Mass Functions.*

Description

Computes Raw, Central, or Standardized moment properties of defined Beta-Binomial probability mass functions.

Usage

```
betabinomialmoments(
  N,
  l,
  u,
  alpha,
  beta,
  types = c("raw", "central", "standardized"),
  orders = 4
)
```

Arguments

N	Number of trials.
l	The first (lower) location-parameter of the Beta distribution.
u	The second (upper) location-parameter of the Beta distribution.
alpha	The alpha (first) shape-parameter of the Beta distribution.
beta	The beta (second) shape-parameter of the Beta-distribution.
types	A character vector determining which moment-types are to be calculated. Permissible values are "raw", "central", and "standardized".
orders	The number of moment-orders to be calculated for each of the moment-types.

Value

A list of moment types, each a list of moment orders.

References

Hanson, B. A (1991). Method of Moments Estimates for the Four-Parameter Beta Compound Binomial Model and the Calculation of Classification Consistency Indexes. American College Testing Research Report Series.

Examples

```
# Assume 100 observations of a discrete variable with probabilities of
# positive outcomes adhering to a four-parameter Beta distribution with
# location parameters l = 0.25 and u = .95, and shape parameters a = 5 and
# b = 3. To compute the first four raw, central, and standardized moments of
# this distrubution using betabinomialmoments():
betabinomialmoments(N = 100, l = .25, u = .95, alpha = 5, beta = 3,
types = c("raw", "central", "standardized"), orders = 4)
```

---

betamedian — *Compute Median of Two- and Four-Parameter Beta Probability Density distribution.*

Description

Computes the median of a Beta distribution with specified shape- and location parameters.

Usage

```
betamedian(alpha, beta, l = 0, u = 1)
```

Arguments

`alpha`	The alpha shape parameter.
`beta`	The beta shape parameter.
`l`	The first (lower) location parameter. Default set to `0`.
`u`	The second (upper) location parameter. Default set to `1`.

Examples

```
# To calculate the median of a two-parameter (standard) Beta distribution with
# shape parameters alpha = 5 and beta = 3:
betamedian(alpha = 5, beta = 3)

# To calculate the median of a four-parameter Beta distribution with shape
# parameters alpha = 5 and beta = 3, and location parameters l = 25 and
# u = 150:
betamedian(alpha = 5, beta = 3, l = 25, u = 150)
```

betamode — *Compute Mode of Two- and Four-Parameter Beta Probability Density distribution.*

Description

Computes the mode of a Beta distribution with specified shape- and location parameters.

Usage

```
betamode(alpha, beta, l = 0, u = 1)
```

Arguments

alpha	The alpha shape parameter of the PDD.
beta	The beta shape parameter of the PDD.
l	The first (lower) location parameter of a four-parameter distribution. Default set to `0`.
u	The second (upper) location parameter of a four-parameter distribution. Default set to 1.

Examples

```
# To calculate the mode of a two-parameter (standard) Beta distribution with
# shape parameters alpha = 5 and beta = 3:
betamode(alpha = 5, beta = 3)

# To calculate the mode of a four-parameter Beta distribution with shape
# parameters alpha = 5 and beta = 3, and location parameters l = 25 and
# u = 150:
betamode(alpha = 5, beta = 3, l = 25, u = 150)
```

betamoments — *Compute Moments of Two-to-Four Parameter Beta Probability Density Distributions.*

Description

Computes Raw, Central, or Standardized moment properties of defined Standard Beta probability density distributions.

Usage

```
betamoments(
  alpha,
  beta,
  l = 0,
  u = 1,
  types = c("raw", "central", "standardized"),
  orders = 4
)
```

Arguments

`alpha`	The alpha shape parameter.
`beta`	The beta shape parameter.
`l`	The first (lower) location parameter.
`u`	The second (upper) location parameter.
`types`	A character vector determining which moment-types are to be calculated. Permissible values are "raw", "central", and "standardized".
`orders`	The number of moment-orders to be calculated for each of the moment-types.

Value

A list of moment types, each a list of moment orders.

References

Hanson, B. A (1991). Method of Moments Estimates for the Four-Parameter Beta Compound Binomial Model and the Calculation of Classification Consistency Indexes. American College Testing Research Report Series.

Examples

```
# Assume some variable follows a four-parameter Beta distribution with
# location parameters l = 0.25 and u = 0.75, and shape parameters alpha = 5
# and beta = 3. To compute the first four raw, central, and standardized
# moments of this distribution using betamoments():
betamoments(alpha = 5, beta = 3, l = 0.25, u = 0.75,
types = c("raw", "central", "standardized"), orders = 4)
```

binomialmoments *Compute Moments of Binomial Probability Mass Functions.*

Description

Computes Raw, Central, or Standardized moment properties of defined Binomial probability mass functions.

Usage

```
binomialmoments(n, p, types = c("raw", "central", "standardized"), orders = 4)
```

Arguments

`n`	Number of Binomial trials
`p`	Probability of success per trial.
`types`	A character vector determining which moment-types are to be calculated. Permissible values are "raw", "central", and "standardized".
`orders`	The number of moment-orders to be calculated for each of the moment-types.

Value

A list of moment types, each a list of moment orders.

References

Hanson, B. A (1991). Method of Moments Estimates for the Four-Parameter Beta Compound Binomial Model and the Calculation of Classification Consistency Indexes. American College Testing Research Report Series.

Examples

```
# Assume some variable follows a four-parameter Beta distribution with
# location parameters l = 0.25 and u = .75, and shape parameters a = 5
# and b = 3. To compute the first four raw, central, and standardized
# moments of this distrubution using betamoments():
betamoments(a = 5, b = 3, l = .25, u = .75,
types = c("raw", "central", "standardized"), orders = 4)
```

BMS *Beta Shape-Parameter Given Location-Parameters, Mean, Variance, Skewness, Kurtosis and Alpha Shape-Parameter of a Four-Parameter Beta PDD.*

Description

Calculates the Beta value required to produce a Beta probability density distribution with defined moments and parameters. Be advised that not all combinations of moments and parameters can be satisfied (e.g., specifying mean, variance, skewness and kurtosis uniquely determines both location-parameters, meaning that the value of the lower-location parameter will take on which ever value it must, and cannot be specified).

Usage

```
BMS(
  mean = NULL,
  variance = NULL,
  skewness = NULL,
  kurtosis = NULL,
  l = 0,
  u = 1,
  alpha = NULL,
  sd = NULL
)
```

Arguments

`mean`	The mean (first raw moment) of the target Standard Beta probability density distribution.
`variance`	The variance (second central moment) of the target Standard Beta probability density distribution.
`skewness`	The skewness (third standardized moment) of the target Beta probability density distribution.
`kurtosis`	The kurtosis (fourth standardized moment) of the target Beta probability density distribution.
`l`	The lower-bound of the Beta distribution. Default is 0 (i.e., the lower-bound of the Standard, two-parameter Beta distribution).
`u`	The upper-bound of the Beta distribution. Default is 1 (i.e., the upper-bound of the Standard, two-parameter Beta distribution).
`alpha`	Optional specification of the Alpha shape-parameter of the target Beta distribution. Finds then the Beta parameter necessary to produce a distribution with the specified mean, given specified Alpha, l, and u parameters.
`sd`	Optional alternative to specifying `var`. The standard deviation of the target Standard Beta probability density distribution.

Value

A numeric value representing the required value for the Beta shape-parameter in order to produce a Standard Beta probability density distribution with the target mean and variance, given specified lower- and upper bounds of the Beta distribution.

Examples

```
# Generate some fictional data. Say, 100 individuals take a test with a
# maximum score of 100 and a minimum score of 0, rescaled to proportion
# of maximum.
set.seed(1234)
testdata <- rbinom(100, 100, rBeta.4P(100, 0.25, 0.75, 5, 3)) / 100
hist(testdata, xlim = c(0, 1))

# To find the beta shape-parameter of a Standard (two-parameter) Beta
# distribution with the same mean and variance as the observed-score
# distribution using BMS():
BMS(mean(testdata), var(testdata))

# To find the beta shape-parameter of a four-parameter Beta
# distribution with specified lower- and upper-bounds of l = 0.25 and
# u = 0.75 using BMS:
BMS(mean(testdata), var(testdata), 0.25, 0.75)
```

---

caStats *Classification Accuracy Statistics.*

Description

Provides a set of statistics often used for conveying information regarding the certainty of classifications based on tests.

Usage

```
caStats(tp, tn, fp, fn)
```

Arguments

tp	The frequency or rate of true-positive classifications.
tn	The frequency or rate of true-negative classifications.
fp	The frequency or rate of false-positive classifications.
fn	The frequency or rate of false-negative classifications.

Value

A list of diagnostic performance statistics based on true/false positive/negative statistics. Specifically, the sensitivity, specificity, positive likelihood ratio (LR.pos), negative likelihood ratio (LR.neg), positive predictive value (PPV), negative predictive value (NPV), Youden's J. (Youden.J), and Accuracy.

References

Glas et al. (2003). The Diagnostic Odds Ratio: A Single Indicator of Test Performance, Journal of Clinical Epidemiology, 1129-1135, 56(11). doi: 10.1016/S0895-4356(03)00177-X

Examples

```
# Generate some fictional data. Say, 100 individuals take a test with a
# maximum score of 100 and a minimum score of 0.
set.seed(1234)
testdata <- rbinom(100, 100, rBeta.4P(100, 0.25, 0.75, 5, 3))
hist(testdata, xlim = c(0, 100))

# Suppose the cutoff value for attaining a pass is 50 items correct, and
# that the reliability of this test was estimated to 0.7. First, compute the
# estimated confusion matrix using LL.CA():
cmat <- LL.CA(x = testdata, reliability = 0.7, cut = 50, min = 0,
max = 100)$confusionmatrix

# To estimate and retrieve diagnostic performance statistics using caStats(),
# feed it the appropriate entries of the confusion matrix.
caStats(tp = cmat["True", "Positive"], tn = cmat["True", "Negative"],
fp = cmat["False", "Positive"], fn = cmat["False", "Negative"])
```

---

cba — *Calculate Cronbach's Alpha reliability-coefficient from supplied variables.*

Description

Calculates Cronbach's Alpha reliability coefficient of the sum-score.

Usage

```
cba(x)
```

Arguments

`x`	A data-frame or matrix of numerical values where rows represent respondents, and columns represent items.

Value

Cronbach's Alpha for the sum-score of supplied variables.

Note

Missing values are treated by passing `na.rm = TRUE` to the `var` function call.

Be aware that this function does not issue a warning if there are negative correlations between variables in the supplied data-set.

References

Cronbach, L.J. (1951). Coefficient alpha and the internal structure of tests. Psychometrika 16, 297–334. doi: 10.1007/BF02310555

Examples

```
# Generate some fictional data. Say 100 students take a 50-item long test
# where all items are equally difficult.
set.seed(1234)
p.success <- rBeta.4P(100, 0.25, 0.75, 5, 3)
for (i in 1:50) {
  if (i == 1) {
    rawdata <- matrix(nrow = 100, ncol = 50)
  }
  rawdata[, i] <- rbinom(100, 1, p.success)
}
# To calculate Cronbach's Alpha for this test:
cba(rawdata)
```

`ccStats` *Classification Consistency Statistics.*

Description

Provides a set of statistics often used for conveying information regarding the consistency of classifications based on tests.

Usage

```
ccStats(ii, ij, ji, jj)
```

Arguments

`ii`	The frequency or rate of consistent classifications into category "i".
`ij`	The frequency or rate of inconsistent classifications into categories "i" and "j".
`ji`	The frequency or rate of inconsistent classifications into categories "j" and "i".
`jj`	The frequency or rate of consistent classifications into category "j".

Value

A list of classification consistency statistics. Specifically, the coefficient of consistent classification (p), the coefficient of consistent classification by chance (p_c), the proportion of positive classifications due to chance (p_c_pos), the proportion of negative classifications due to chance (p_c_neg), and Cohen's Kappa coefficient.

References

Hanson, Bradley A. (1991). Method of Moments Estimates for the Four-Parameter Beta Compound Binomial Model and the Calculation of Classification Consistency Indexes. American College Testing.

Examples

```
# Generate some fictional data. Say, 100 individuals take a test with a
# maximum score of 100 and a minimum score of 0.
set.seed(1234)
testdata <- rbinom(100, 100, rBeta.4P(100, .25, .75, 5, 3))
hist(testdata, xlim = c(0, 100))

# Suppose the cutoff value for attaining a pass is 50 items correct, and
# that the reliability of this test was estimated to 0.7. First, compute the
# estimated consistency matrix using LL.CA():
cmat <- LL.CA(x = testdata, reliability = .7, cut = 50, min = 0,
max = 100)$consistencymatrix

# To estimate and retrieve consistency statistics using ccStats(),
# feed it the appropriate entries of the consistency matrix.
ccStats(ii = cmat["i", "i"], ij = cmat["i", "j"],
ji = cmat["j", "i"], jj = cmat["j", "j"])
```

`confmat` *Confusion matrix*

Description

Organizes supplied values of true and false positives and negatives into a confusion matrix.

Usage

```
confmat(tp, tn, fp, fn, output = "freq")
```

Arguments

`tp`	The frequency or rate of true-positive classifications.
`tn`	The frequency or rate of true-negative classifications.
`fp`	The frequency or rate of false-positive classifications.
`fn`	The frequency or rate of false-negative classifications.
`output`	Whether the returned output reflects frequencies or proportions. Defaults to returning frequencies.

Value

A confusion matrix organizing the input values of true and false positive and negatives.

Examples

```
# Generate some true and observed conditions.
set.seed(1234)
true.ability <- rbeta(50, 4, 4)
true.category <- ifelse(true.ability < 0.5, 0, 1)
observed.score <- rbinom(50, 50, true.ability)
observed.category <- ifelse(observed.score < 25, 0, 1)
# Calculate the frequencies of true and false positives and negatives based on the true and
# observed conditions.
TP <- sum(ifelse(observed.category == 0 & true.category == 0, 1, 0))
FP <- sum(ifelse(observed.category == 0 & true.category != 0, 1, 0))
TN <- sum(ifelse(observed.category == 1 & true.category == 1, 1, 0))
FN <- sum(ifelse(observed.category == 1 & true.category != 1, 1, 0))
# Organize the above values in a confusion matrix using the confmat function:
confmat(tp = TP, fp = FP, tn = TN, fn = FN)
```

---

dBeta.4P *Probability Density under the Four-Parameter Beta PDD.*

Description

Gives the density at desired values of x under the Four-Parameter Beta PDD.

Usage

```
dBeta.4P(x, l, u, alpha, beta)
```

Arguments

x	Value of x.
l	The first (lower) location parameter.
u	The second (upper) location parameter.
alpha	The first shape parameter.
beta	The second shape parameter.

Value

The value for the probability density at specified values of x.

Examples

```
# Assume some variable follows a four-parameter Beta distribution with
# location parameters l = 0.25 and u = 0.75, and shape parameters alpha = 5
# and beta = 3. To compute the probability density at a specific point of
# the distribution (e.g., 0.5) using dBeta.4P():
dBeta.4P(x = 0.5, l = 0.25, u = 0.75, alpha = 5, beta = 3)
```

`dBeta.pBeta` *An implementation of a Beta-density Compound Cumulative-Beta Distribution.*

Description

The Beta Compound Beta distribution: The product of the four-parameter Beta probability density function and the Beta cumulative probability function. Used in the Livingston and Lewis approach to classification accuracy and consistency, the output can be interpreted as the population density of passing scores produced at "x" (a value of true-score).

Usage

```
dBeta.pBeta(x, l, u, alpha, beta, n, c, lower.tail = FALSE)
```

Arguments

`x`	x-axis input for which `p` (proportion or probability) is to be computed.
`l`	The lower-bound of the four-parameter Beta distribution.
`u`	The upper-bound of the four-parameter Beta distribution.
`alpha`	The alpha shape-parameter of the Beta density distribution.
`beta`	The beta shape-parameter of the Beta density distribution.
`n`	The number of trials for the Beta cumulative probability distribution.
`c`	The "true-cut" (proportion) of on the Beta cumulative probability distribution.
`lower.tail`	Logical. Whether to compute the lower or upper tail of the Beta cumulative probability distribution. Default is `FALSE` (i.e., upper tail).

References

Hanson, Bradley A. (1991). Method of Moments Estimates for the Four-Parameter Beta Compound Binomial Model and the Calculation of Classification Consistency Indexes.American College Testing Research Report Series.

Livingston, Samuel A. and Lewis, Charles. (1995). Estimating the Consistency and Accuracy of Classifications Based on Test Scores. Journal of Educational Measurement, 32(2).

Lord, Frederic M. (1965). A Strong True-Score Theory, With Applications. Psychometrika, 30(3).

Examples

```
# Given a four-parameter Beta distribution with parameters l = 0.25, u = 0.75,
# alpha = 5, and beta = 3, and a Beta error distribution with number of
# trials (n) = 10 and a cutoff-point (c) at 50% correct (i.e., proportion correct
# of 0.5), the population density of passing scores produced at true-score
# (x) = 0.5 can be calculated as:
dBeta.pBeta(x = 0.5, l = 0.25, u = 0.75, a = 5, b = 3, n = 10, c = 0.5)
```

```
# Conversely, the density of failing scores produced at x can be calculated
# by passing the additional argument "lower.tail = TRUE" to the function.
# That is:
dBeta.pBeta(x = 0.5, l = 0.25, u = 0.75, a = 5, b = 3, n = 10, c = 0.5,
lower.tail = TRUE)

# By integration, the population proportion of (e.g.) passing scores in some
# region of the true-score distribution (e.g. between 0.25 and 0.5) can be
# calculated as:
integrate(function(x) { dBeta.pBeta(x, 0.25, 0.75, 5, 3, 10, 0.5) },
lower = 0.25, upper = 0.5)
```

`dBeta.pBinom` *An implementation of the Beta-density Compound Cumulative Binomial Distribution.*

Description

The Beta Compound Binomial distribution: The product of the four-parameter Beta probability density function and the binomial cumulative probability mass function. Used in the Livingston and Lewis approach to classification accuracy and consistency, the output can be interpreted as the population density of passing scores produced at "x" (a value of true-score).

Usage

```
dBeta.pBinom(x, l, u, alpha, beta, n, c, lower.tail = FALSE)
```

Arguments

`x`	x-axis input for which `p` (proportion or probability) is to be computed.
`l`	The lower-bound of the four-parameter Beta distribution.
`u`	The upper-bound of the four-parameter Beta distribution.
`alpha`	The alpha shape-parameter of the Beta distribution.
`beta`	The beta shape-parameter of the Beta distribution.
`n`	The number of trials for the Binomial distribution.
`c`	The "true-cut" (proportion) of the Binomial distribution.
`lower.tail`	Logical. Whether to compute the lower or upper tail of the Binomial distribution. Default is `FALSE` (i.e., upper tail).

Note

The Binomial distribution cut-point is up-to but not including, unlike the standard behaviour of base-R pbinom() function.

References

Hanson, Bradley A. (1991). Method of Moments Estimates for the Four-Parameter Beta Compound Binomial Model and the Calculation of Classification Consistency Indexes.American College Testing Research Report Series.

Livingston, Samuel A. and Lewis, Charles. (1995). Estimating the Consistency and Accuracy of Classifications Based on Test Scores. Journal of Educational Measurement, 32(2).

Lord, Frederic M. (1965). A Strong True-Score Theory, With Applications. Psychometrika, 30(3).

Examples

```
# Given a four-parameter Beta distribution with parameters l = 0.25, u = 0.75,
# alpha = 5, and beta = 3, and a Binomial error distribution with number of
# trials (n) = 10 and a cutoff-point (c) at 50% correct (i.e., proportion correct
# of 0.5), the population density of passing scores produced at true-score
# (x) = 0 can be calculated as:
dBeta.pBinom(x = 0.5, l = 0.25, u = 0.75, a = 5, b = 3, n = 10, c = 0.5)

# Conversely, the density of failing scores produced at x can be calculated
# by passing the additional argument "lower.tail = TRUE" to the function.
# That is:
dBeta.pBinom(x = 0.5, l = 0.25, u = 0.75, a = 5, b = 3, n = 10, c = 0.5,
lower.tail = TRUE)

#By integration, the population proportion of (e.g.) passing scores in some
#region of the true-score distribution (e.g. between 0.25 and 0.5) can be
#calculated as:
integrate(function(x) { dBeta.pBinom(x, 0.25, .75, 5, 3, 10, 0.5) },
lower = 0.25, upper = 0.5)
```

`dBeta.pGammaBinom` *An implementation of a Beta-density Compound Cumulative Gamma-Binomial Distribution.*

Description

The Beta Compound Binomial distribution: The product of the four-parameter Beta probability density function and the binomial cumulative probability mass function. Used in the Livingston and Lewis approach to classification accuracy and consistency, the output can be interpreted as the population density of passing scores produced at "x" (a value of true-score).

Usage

```
dBeta.pGammaBinom(x, l, u, alpha, beta, n, c, lower.tail = FALSE)
```

Arguments

`x`	x-axis input for which `p` (proportion or probability) is to be computed.
`l`	The lower-bound of the four-parameter Beta distribution.
`u`	The upper-bound of the four-parameter Beta distribution.
`alpha`	The alpha shape-parameter of the four-parameter Beta distribution.
`beta`	The beta shape-parameter of the four-parameter Beta distribution.
`n`	The number of "trials" for the Gamma-Binomial distribution.
`c`	The "true-cut" (proportion) on the Gamma-Binomial distribution. Need not be an integer (unlike Binomial distribution).
`lower.tail`	Logical. Whether to compute the lower or upper tail of the Binomial distribution. Default is `FALSE` (i.e., upper tail).

References

Hanson, Bradley A. (1991). Method of Moments Estimates for the Four-Parameter Beta Compound Binomial Model and the Calculation of Classification Consistency Indexes.American College Testing Research Report Series.

Livingston, Samuel A. and Lewis, Charles. (1995). Estimating the Consistency and Accuracy of Classifications Based on Test Scores. Journal of Educational Measurement, 32(2).

Lord, Frederic M. (1965). A Strong True-Score Theory, With Applications. Psychometrika, 30(3).

Loeb, D. E. (1992). A generalization of the binomial coefficients. Discrete Mathematics, 105(1-3).

Examples

```
# Given a four-parameter Beta distribution with parameters l = 0.25, u = 0.75,
# alpha = 5, and beta = 3, and a Binomial error distribution with number of
# trials (n) = 10 and a cutoff-point (c) at 50% correct (i.e., proportion correct
# of 0.5), the population density of passing scores produced at true-score
# (x) = 0 can be calculated as:
dBeta.pGammaBinom(x = 0.5, l = 0.25, u = 0.75, a = 5, b = 3, n = 10, c = 0.5)

# Conversely, the density of failing scores produced at x can be calculated
# by passing the additional argument "lower.tail = TRUE" to the function.
# That is:
dBeta.pGammaBinom(x = 0.5, l = 0.25, u = 0.75, a = 5, b = 3, n = 10.1, c = 0.5,
lower.tail = TRUE)

#By integration, the population proportion of (e.g.) passing scores in some
#region of the true-score distribution (e.g. between 0.25 and 0.5) can be
#calculated as:
integrate(function(x) { dBeta.pGammaBinom(x, 0.25, 0.75, 5, 3, 10, 0.5) },
lower = 0.25, upper = 0.5)
```

dBetaBinom *Probability Mass under the Beta-Binomial Probability-Mass Distribution.*

Description

Gives the density at x under the Beta-Binomial PMF.

Usage

```
dBetaBinom(x, N, l, u, alpha, beta)
```

Arguments

x	Value of x (a specific number of successes).
N	The total number of trials.
l	The first (lower) location parameter.
u	The second (upper) location parameter.
alpha	The first shape parameter.
beta	The second shape parameter.

Value

The value for the probability mass at x given the specified Beta-Binomial distribution.

Examples

```
# Assume some variable follows a Beta-Binomial distribution with 100 number
# of trials, and with probabilities of successful trials drawn from a four-
# parameter Beta distribution with location parameters l = 0.25 and u = 0.75
# and shape parameters alpha = 5 and beta = 3. To compute the probability
# density at a specific point of the distribution (e.g., 50):
dBetaBinom(x = 50, N = 100, l = 0.25, u = 0.75, alpha = 5, beta = 3)
```

dBetaMS *Density Under a Specific Point of the Standard Beta PDD with Specific Mean and Variance or Standard Deviation.*

Description

Calculates the density under specific points of the Standard Beta probability density distribution with defined mean and variance or standard deviation.

Usage

```
dBetaMS(x, mean, variance = NULL, sd = NULL)
```

Arguments

`x`	A specific point on the x-axis of the Standard Beta PDD.
`mean`	The mean of the target Standard Beta probability density distribution.
`variance`	The variance of the target Standard Beta probability density distribution.
`sd`	The standard deviation of the target Standard Beta probability density distribution.

Value

A numeric value representing the required value for the beta Shape-parameter in order to produce a Standard Beta probability density distribution with the target mean and variance.

Examples

```
# To compute the density at a specific point (e.g., 0.5) along the Standard
# (two-parameter) PDD with mean of 0.6 and variance of 0.04:
dBetaMS(x = 0.5, mean = 0.6, variance = 0.04)
```

`dfac` *Descending (falling) factorial.*

Description

Calculate the descending (or falling) factorial of a value `x` of order `r`.

Usage

```
dfac(x, r, method = "product")
```

Arguments

`x`	A value for which the descending factorial is to be calculated.
`r`	The power `x` is to be raised to.
`method`	The method by which the descending factorials are to be calculated. Default is `"product"` which uses direct arithmetic. Alternative is `"gamma"` which calculates the ascending factorial using the Gamma function. The alternative method might be faster but might fail because the Gamma function is not defined for negative integers (returning Inf).

Value

The descending factorial of value `x` raised to the `r`'th power.

Examples

```
# To calculate the 4th descending factorial for a value (e.g., 3.14):
dfac(x = 3.14, r = 4)

# To calculate the 5th descending factorial for values 3.14, 2.72, and 0.58:
dfac(x = c(3.14, 2.72, 0.58), r = 5)
```

---

dGammaBinom — *Probability density function under the Gamma-extended Binomial distribution.*

Description

Probability density function under the Gamma-extended Binomial distribution.

Usage

```
dGammaBinom(x, size, prob, nc = FALSE)
```

Arguments

x	Vector of quantiles.
size	Number of "trials" (zero or more). Need not be integer.
prob	Probability of "success" on each "trial". Need not be integer.
nc	Whether to include a normalizing constant making sure that the sum of the distribution's density is 1.

References

Loeb, D. E. (1992). A generalization of the binomial coefficients. Discrete Mathematics, 105(1-3).

Examples

```
#' # Assume some variable follows a Gamma-Binomial distribution with
# "number of trials" = 10.5 and probability of "success" for each "trial"
# = 0.75, to compute the probability density to attain a "number of success"
# at a specific point (e.g., 7.5 "successes"):
dGammaBinom(x = 7.5, size = 10.5, prob = 0.75)

# Including a normalizing constant (then diverges from binomial dist.):
dGammaBinom(x = 7.5, size = 10.5, prob = 0.75, nc = TRUE)
dGammaBinom(x = 7, size = 10, prob = 0.75) == dbinom(7, 10, 0.75)
dGammaBinom(x = 7, size = 10, prob = 0.75, nc = TRUE) == dbinom(7, 10, 0.75)
```

ETL *Livingston and Lewis' "Effective Test Length".*

Description

According to Livingston and Lewis (1995), "The effective test length corresponding to a test score is the number of discrete, dichotomously scored, locally independent, equally difficult items required to produce a total score of the same reliability."

Usage

```
ETL(mean, variance, min = 0, max = 1, reliability)
```

Arguments

`mean`	The mean of the observed-score distribution.
`variance`	The variance of the observed-score distribution.
`min`	The lower-bound (minimum possible value) of the observed-score distribution. Default is 0 (assuming observed scores represent proportions).
`max`	The upper-bound (maximum possible value) of the observed-score distribution. Default is 1 (assuming observed scores represent proportions).
`reliability`	The reliability of the observed scores (proportion of observed-score distribution variance shared with true-score distribution).

Value

An estimate of the effective length of a test, given the stability of the observations it produces.

References

Livingston, Samuel A. and Lewis, Charles. (1995). Estimating the Consistency and Accuracy of Classifications Based on Test Scores. Journal of Educational Measurement, 32(2).

Examples

```
# Generate some fictional data. Say, 100 individuals take a test with a
# maximum score of 100 and a minimum score of 0.
set.seed(1234)
testdata <- rbinom(100, 100, rBeta.4P(100, .25, .75, 5, 3))
hist(testdata, xlim = c(0, 100))

# Suppose the reliability of this test was estimated to 0.7. To estimate and
# retrieve the effective test length using ETL():
ETL(mean = mean(testdata), variance = var(testdata), min = 0, max = 100,
reliability = .7)
```

gchoose — *Gamma-extended Binomial coefficient (choose function).*

Description

Extends the Binomial coefficient for positive non-integers (including 0) by employing the Gamma rather than the factorial function.

Usage

```
gchoose(n, k)
```

Arguments

`n`	In Binomial terms, the number of Binomial "trials". Need not be an integer.
`k`	In Binomial terms, the number of successful "trials". Need not be an integer.

Note

Not defined for negative integers.

References

Loeb, D. E. (1992). A generalization of the binomial coefficients. Discrete Mathematics, 105(1-3).

Examples

```
# Compare choose function with gchoose function for integers:
gchoose(c(8, 9, 10), c(3, 4, 5)) == choose(c(8, 9, 10), c(3, 4, 5))

# The gchoose function also works for non-integers:
gchoose(10.5, 7.5)
```

LABMSU — *Lower Location Parameter Given Shape Parameters, Mean, Variance, and Upper Location Parameter of a Four-Parameter Beta PDD.*

Description

Calculates the lower-bound value required to produce a Beta probability density distribution with defined moments and parameters. Be advised that not all combinations of moments and parameters can be satisfied (e.g., specifying mean, variance, skewness and kurtosis uniquely determines both location-parameters, meaning that the value of the lower-location parameter will take on which ever value it must, and cannot be specified).

Usage

```
LABMSU(
  alpha = NULL,
  beta = NULL,
  u = NULL,
  mean = NULL,
  variance = NULL,
  skewness = NULL,
  kurtosis = NULL,
  sd = NULL
)
```

Arguments

alpha	The alpha (first) shape-parameter of the target Beta probability density distribution.
beta	The beta (second) shape-parameter of the target Beta probability density distribution.
u	The upper-bound of the Beta distribution. Default is NULL (i.e., does not take a specified u-parameter into account).
mean	The mean (first raw moment) of the target Standard Beta probability density distribution.
variance	The variance (second central moment) of the target Standard Beta probability density distribution.
skewness	The skewness (third standardized moment) of the target Beta probability density distribution.
kurtosis	The kurtosis (fourth standardized moment) of the target Beta probability density distribution.
sd	Optional alternative to specifying `var`. The standard deviation of the target Standard Beta probability density distribution.

Value

A numeric value representing the required value for the Beta lower location-parameter (`l`) in order to produce a Beta probability density distribution with the target moments and parameters.

Examples

```
# Generate some fictional data.
set.seed(1234)
testdata <- rBeta.4P(100000, 0.25, 0.75, 5, 3)
hist(testdata, xlim = c(0, 1), freq = FALSE)

# Suppose you know three of the four necessary parameters to fit a four-
# parameter Beta distribution (i. e., u = 0.75, alpha = 5, beta = 3) to this
# data. To find the value for the necessary l parameter, estimate the mean
# and variance of the distribution:
```

```
M <- mean(testdata)
S2 <- var(testdata)

# To find the l parameter necessary to produce a four-parameter Beta
# distribution with the target mean, variance, and u, alpha, and beta
# parameters using the LMSBAU() function:
(l <- LABMSU(alpha = 5, beta = 3, mean = M, variance = S2, u = 0.75))
curve(dBeta.4P(x, l, .75, 5, 3), add = TRUE, lwd = 2)
```

LL.CA *An Implementation of the Livingston and Lewis (1995) Approach to Estimate Classification Consistency and Accuracy based on Observed Test Scores and Test Reliability.*

Description

An implementation of what has been come to be known as the "Livingston and Lewis approach" to classification consistency and accuracy, which by employing a compound beta-binomial distribution assumes that true-scores conform to the four-parameter beta distribution, and errors of measurement to the binomial distribution. Under these assumptions, the expected classification consistency and accuracy of tests can be estimated from observed outcomes and test reliability.

Usage

```
LL.CA(
  x = NULL,
  reliability,
  cut,
  min = 0,
  max = 1,
  true.model = "4P",
  truecut = NULL,
  output = c("accuracy", "consistency"),
  failsafe = TRUE,
  l = 0,
  u = 1,
  modelfit = c(nbins = 100, minbin = 10)
)
```

Arguments

x	A vector of observed scores for which a Beta true-score distribution is to be estimated, or a list of pre-defined true-score distribution parameter values. If a list is provided, the list entries must be named after the parameters: `l` and `u` for the location parameters, `alpha` and `beta` for the shape parameters, and `etl` for the effective test length (see documentation for the `ETL` function).
reliability	The observed-score squared correlation (i.e., proportion of shared variance) with the true-score.

cut	The cutoff value for classifying observations into pass or fail categories.
min	The minimum value possible to attain on the test. Default is 0 (assuming x represent proportions).
max	The maximum value possible to attain on the test. Default is 1 (assuming x represent proportions).
true.model	The probability distribution to be fitted to the moments of the true-score distribution. Options are "4P" (default) and "2P", referring to four- and two-parameter Beta distributions. The "4P" method produces a four-parameter Beta distribution with the same first four moments (mean, variance, skewness, and kurtosis) as the estimated true-score distribution, while the "2P" method produces a two-parameter Beta distribution with the first two moments (mean and variance) as the estimated true-score distribution.
truecut	Optional specification of a "true" cutoff. Useful for producing ROC curves (see documentation for the LL.ROC() function).
output	Character vector indicating which types of statistics (i.e, accuracy and/or consistency) are to be computed and included in the output. Permissible values are "accuracy" and "consistency".
failsafe	Logical value indicating whether to engage the automatic fail-safe defaulting to the two-parameter Beta true-score distribution if the four-parameter fitting procedure produces impermissible parameter estimates. Default is TRUE (i.e., the function will engage failsafe if the four-parameter Beta-distribution fitting-procedure produced impermissible estimates).
l	If true.model = "2P" or failsafe = TRUE, the lower-bound location parameter to be used in the two-parameter fitting procedure. Default is 0 (i.e., the lower-bound of the Standard Beta distribution).
u	If true.model = "2P" or failsafe = TRUE, the upper-bound location parameter to be used in the two-parameter fitting procedure. Default is 1 (i.e., the upper-bound of the Standard Beta distribution).
modelfit	Allows for controlling the chi-square test for model fit. The argument takes a vector of two values. The first value is to represent the initial number of bins the distribution of scores is to be divided in to. This value is set to a default of 100. If this default results in too few bins to conduct the chi-square test, this value can be made larger. The second value represents the minimum expected number of observations that the bins should consist of. In accordance with standard recommendations for chi-square tests, the default value is set to 10.

Value

A list containing the estimated parameters necessary for the approach (i.e., the effective test-length and the beta distribution parameters), a chi-square test of model-fit, the confusion matrix containing estimated proportions of true/false pass/fail categorizations for a test, diagnostic performance statistics, and / or a classification consistency matrix and indices. Accuracy output includes a confusion matrix and diagnostic performance indices, and consistency output includes a consistency matrix and consistency indices p (expected proportion of agreement between two independent test administrations), p_c (proportion of agreement on two independent administrations expected by chance alone), and Kappa (Cohen's Kappa).

Note

It should be noted that this implementation differs from the original articulation of Livingston and Lewis (1995) in some respects. First, the procedure includes a number of diagnostic performance (accuracy) indices which the original procedure enables but that were not included. Second, the way consistency is calculated differs substantially from the original articulation of the procedure, which made use of a split-half approach. Rather, this implementation uses the approach to estimating classification consistency outlined by Hanson (1991).

A shiny application providing a GUI for this method is available at https://hthaa.shinyapps.io/shinybeta/ .

References

Livingston, Samuel A. and Lewis, Charles. (1995). Estimating the Consistency and Accuracy of Classifications Based on Test Scores. Journal of Educational Measurement, 32(2).

Hanson, Bradley A. (1991). Method of Moments Estimates for the Four-Parameter Beta Compound Binomial Model and the Calculation of Classification Consistency Indexes. American College Testing.

Lord. Frederic M. (1965). A Strong True-Score Theory, With Applications. Psychometrika, 30(3).

Lewis, Don and Burke, C. J. (1949). The Use and Misuse of the Chi-Square Test. Psychological Bulletin, 46(6).

Examples

```
# Generate some fictional data. Say, 1000 individuals take a test with a
# maximum score of 100 and a minimum score of 0.
set.seed(1234)
testdata <- rbinom(1000, 100, rBeta.4P(1000, 0.25, 0.75, 5, 3))
hist(testdata, xlim = c(0, 100))

# Suppose the cutoff value for attaining a pass is 50 items correct, and
# that the reliability of this test was estimated to 0.7. To estimate and
# retrieve the estimated parameters, confusion matrix, consistency and
# accuracy statistics using LL.CA():
LL.CA(x = testdata, reliability = .7, cut = 50, min = 0, max = 100)

# Suppose the true-score parameter estimation procedure arrived at
# impermissible parameter estimates (i.e., l < 0, u > 1, alpha < 0, or
# beta < 0). For example:
set.seed(9)
testdata <- rbinom(100, 25, rBeta.4P(100, 0.25, 1, 5, 3))
Beta.tp.fit(testdata, 0, 25, 25, failsafe = TRUE)

# Suppose further that you have good grounds for assuming that the lower-
# bound parameter is equal to 0.25 (e.g., the test consists of multiple-
# choice questions with four response options, leading to a 25% probability
# of guessing the correct answer per question), and good reason to believe
# that the upper-bound parameter is equal to 1 (i.e., there is no reason to
# believe that there are no members of the population who will attain a
# perfect score across all possible test-forms.) To set these lower and
```

```
# upper bounds for the fitting procedure in the LL.CA() function, set
# the argument true.model = "2p", and specify the location parameters
# l = 0.25 and u = 1:
LL.CA(testdata, 0.6287713, 12, 0, 25, true.model = "2p", l = 0.25, u = 1)

# Alternatively to supplying scores to which a true-score distribution is
# to be fit, a list with true-score distribution parameter values can be
# supplied manually along with the effective test length (see documentation
# for the ETL() function), foregoing the need for actual data. The list
# entries must be named. "l" is the lower-bound and "u" the upper-bound
# location parameters of the true-score distribution, "alpha" and "beta" for
# the shape parameters, and "etl" for the effective test-length..
trueparams <- list("l" = 0.25, "u" = 0.75, "alpha" = 5, "beta" = 3, "etl" = 50)
LL.CA(x = trueparams, cut = 50, min = 0, max = 100)
```

LL.CA.MC — *An Extension of the Livingston and Lewis (1995) Approach to Estimate Classification Consistency and Accuracy for Multiple Classifications based on Observed Test Scores and Test Reliability.*

Description

An implementation of what has been come to be known as the "Livingston and Lewis approach" to classification consistency and accuracy, which by employing a compound beta-binomial distribution assumes that true-scores conform to the four-parameter beta distribution, and errors of measurement to the binomial distribution. Under these assumptions, the expected classification consistency and accuracy of tests can be estimated from observed outcomes and test reliability.

Usage

```
LL.CA.MC(
  x = NULL,
  reliability,
  cut,
  min = 0,
  max = 1,
  true.model = "4P",
  failsafe = TRUE,
  l = 0,
  u = 1,
  modelfit = c(nbins = 100, minbin = 10)
)
```

Arguments

x	A vector of observed scores for which a Beta true-score distribution is to be estimated, or a list of pre-defined true-score distribution parameter values. If a list is provided, the list entries must be named after the parameters: `l` and `u` for the location parameters, alpha and beta for the shape parameters, and etl for the effective test length (see documentation for the ETL function).

reliability
: The observed-score squared correlation (i.e., proportion of shared variance) with the true-score.

cut
: A vector of cut-off values for classifying observations into two or more categories.

min
: The minimum value possible to attain on the test. Default is 0 (assuming x represent proportions).

max
: The maximum value possible to attain on the test. Default is 1 (assuming x represent proportions).

true.model
: The probability distribution to be fitted to the moments of the true-score distribution. Options are "4P" (default) and "2P", referring to four- and two-parameter Beta distributions. The "4P" method produces a four-parameter Beta distribution with the same first four moments (mean, variance, skewness, and kurtosis) as the estimated true-score distribution, while the "2P" method produces a two-parameter Beta distribution with the first two moments (mean and variance) as the estimated true-score distribution.

failsafe
: Logical value indicating whether to engage the automatic fail-safe defaulting to the two-parameter Beta true-score distribution if the four-parameter fitting procedure produces impermissible parameter estimates. Default is TRUE (i.e., the function will engage failsafe if the four-parameter Beta-distribution fitting-procedure produced impermissible estimates).

l
: If true.model = "2P" or failsafe = TRUE, the lower-bound location parameter to be used in the two-parameter fitting procedure. Default is 0 (i.e., the lower-bound of the Standard Beta distribution).

u
: If true.model = "2P" or failsafe = TRUE, the upper-bound location parameter to be used in the two-parameter fitting procedure. Default is 1 (i.e., the upper-bound of the Standard Beta distribution).

modelfit
: Allows for controlling the chi-square test for model fit. The argument takes a vector of two values. The first value is to represent the initial number of bins the distribution of scores is to be divided in to. This value is set to a default of 100. If this default results in too few bins to conduct the chi-square test, this value can be made larger. The second value represents the minimum expected number of observations that the bins should consist of. In accordance with standard recommendations for chi-square tests, the default value is set to 10.

Value

A list containing the estimated parameters necessary for the approach (i.e., the effective test-length and the beta distribution parameters), a chi-square test of model-fit, the confusion matrix containing estimated proportions of true/false positive/negative categorizations for a test, diagnostic performance statistics, and/or a classification consistency matrix and indices. Accuracy output includes a confusion matrix and diagnostic performance indices, and consistency output includes a consistency matrix and consistency indices p (expected proportion of agreement between two independent test administrations), p_c (proportion of agreement on two independent administrations expected by chance alone), and Kappa (Cohen's Kappa).

Note

It should be noted that this implementation differs from the original articulation of Livingston and Lewis (1995) in some respects. First, the procedure includes a number of diagnostic performance (accuracy) indices which the original procedure enables but that were not included. Second, the way consistency is calculated differs substantially from the original articulation of the procedure, which made use of a split-half approach. Rather, this implementation uses the approach to estimating classification consistency outlined by Hanson (1991).

References

Livingston, Samuel A. and Lewis, Charles. (1995). Estimating the Consistency and Accuracy of Classifications Based on Test Scores. Journal of Educational Measurement, 32(2).

Hanson, Bradley A. (1991). Method of Moments Estimates for the Four-Parameter Beta Compound Binomial Model and the Calculation of Classification Consistency Indexes. American College Testing.

Lord. Frederic M. (1965). A Strong True-Score Theory, With Applications. Psychometrika, 30(3).

Lewis, Don and Burke, C. J. (1949). The Use and Misuse of the Chi-Square Test. Psychological Bulletin, 46(6).

Examples

```
# Generate some fictional data. Say, 1000 individuals take a test with a
# maximum score of 100 and a minimum score of 0.
set.seed(1234)
p.success <- rBeta.4P(1000, 0.1, 0.95, 5, 3)
for (i in 1:100) {
  if (i == 1) {
    rawdata <- matrix(nrow = 1000, ncol = 100)
  }
  rawdata[, i] <- rbinom(1000, 1, p.success)
}

# Suppose the cutoff value for being placed in the lower category is a score
# below 50, second lowest 60, then 70, 80, and 90. Using the cba() function
# to estimate the reliability of this test, to use the LL.CA.MC() function
# or estimating diagnostic performance and consistency indices of
# classifications when using several cut-points:
LL.CA.MC(rowSums(rawdata), cba(rawdata), c(50, 60, 70, 80, 90), min = 0, max = 100)

# The output from this function can get quite verbose when operating with
# several cut-points. In order to retrieve only model parameter estimates:
LL.CA.MC(rowSums(rawdata), cba(rawdata), c(50, 60, 70, 80, 90), min = 0, max = 100)$parameters

# To retrieve only the model-fit estimate:
LL.CA.MC(rowSums(rawdata), cba(rawdata), c(50, 60, 70, 80, 90), min = 0, max = 100)$modelfit

# To retrieve only the diagnostic performance estimates:
LL.CA.MC(rowSums(rawdata), cba(rawdata), c(50, 60, 70, 80, 90), min = 0, max = 100)$accuracy

# To retrieve only the classification consistency indices:
```

```
LL.CA.MC(rowSums(rawdata), cba(rawdata), c(50, 60, 70, 80, 90), min = 0, max = 100)$consistency

# Alternatively, the MC.out.tabular() function can be used to organize the
# category-specific indices in a tabular format:
MC.out.tabular(LL.CA.MC(rowSums(rawdata), cba(rawdata), c(50, 60, 70, 80, 90), min = 0, max = 100))
```

LL.ROC *ROC curves for the Livingston and Lewis approach.*

Description

Generate a ROC curve plotting the false-positive rate against the true-positive rate at different cut-off values across the observed proportion-score scale.

Usage

```
LL.ROC(
  x = NULL,
  reliability,
  min = 0,
  max = 1,
  truecut,
  true.model = "4P",
  failsafe = TRUE,
  l = 0,
  u = 1,
  AUC = FALSE,
  maxJ = FALSE,
  maxAcc = FALSE,
  locate = NULL,
  raw.out = FALSE,
  grainsize = 100
)
```

Arguments

`x`	A vector of observed results.
`reliability`	The reliability coefficient of the test.
`min`	The minimum possible value to attain on the observed-score scale. Default is 0 (assuming `x` represent proportions).
`max`	The maximum possible value to attain on the observed-score scale. Default is 1 (assuming `x` represent proportions).
`truecut`	The true point along the x-scale that marks the categorization-threshold.

`true.model` The probability distribution to be fitted to the moments of the true-score distribution. Options are "4P" (default) and "2P", referring to four- and two-parameter Beta distributions. The "4P" method produces a four-parameter Beta distribution with the same first four moments (mean, variance, skewness, and kurtosis) as the estimated true-score distribution, while the "2P" method produces a two-parameter Beta distribution with the first two moments (mean and variance) as the estimated true-score distribution.

`failsafe` If true-model == "4P": Whether to engage a fail-safe reverting to a two-parameter true-score distribution solution should the four-parameter fitting procedure produce impermissible results. Default is TRUE (engage fail-safe in the event of impermissible estimates).

`l` If `true.model` == "2P" or `failsafe` == TRUE: The lower-bound location parameter of the two-parameter true-score distribution solution.

`u` If `true.model` == "2P" or `failsafe` == TRUE: The upper-bound location parameter of the two-parameter true-score distribution solution.

`AUC` Calculate and include the area under the curve? Default is FALSE.

`maxJ` Logical. Mark the point along the curve where Youden's J statistic is maximized? Default is FALSE.

`maxAcc` Logical. Mark the point along the curve where the Accuracy statistic is maximized? Default is FALSE.

`locate` Ask the function to locate the cut-point at which sensitivity or NPV is greater than or equal to some value, or specificity or PPV is lesser than or equal to some value. Take as input a character-vector of length 2, with the first argument being which index is to be found (e.g., "sensitivity"), and the second argument the value to locate (e.g., "0.75"). For example: c("sensitivity", "0.75").

`raw.out` Give raw coordinates as output rather than plot? Default is FALSE

`grainsize` Specify the number of cutoff-points for which the ROC curve is to be calculated. The greater this number the greater the accuracy. Default is 100 points.

Value

A plot tracing the ROC curve for the test, or matrix of coordinates if raw.out is TRUE.

Examples

```
# Generate some fictional data. Say, 1000 individuals take a test with a
# maximum score of 100 and a minimum score of 0.
set.seed(1234)
testdata <- rbinom(1000, 100, rBeta.4P(1000, 0.25, 0.75, 5, 3))
hist(testdata / 100, xlim = c(0, 1), freq = FALSE)

# Suppose the cutoff value for attaining a pass is 50 items correct.
# Suppose further that the reliability of the test-scores were estimated to
# 0.75. To produce a plot with an ROC curve using LL.ROC(), along with the
# AUC statistics and the points at which Youden's J. is maximized:
LL.ROC(x = testdata, reliability = 0.7, truecut = 50, min = 0, max = 100,
AUC = TRUE, maxJ = TRUE)
```

```
# Or to locate the point at which accuracy is maximized:
LL.ROC(x = testdata, reliability = 0.7, truecut = 50, min = 0, max = 100,
maxAcc = TRUE)

# Using the example data above, the function can be instructed to locate an
# operational cut-point at which sensitivity or specificity is equal to or
# greater than some specified value by specifying the "locate" argument with
# c("statistic", value). For example, to locate the operational cut-point at
# which sensitivity is first equal to or greater than 0.9:
LL.ROC(testdata, reliability = 0.7, min = 0, max = 100, truecut = 50,
locate = c("sensitivity", 0.9))
# For Negative Predictive value, the point at which it is equal or greater:
LL.ROC(testdata, reliability = 0.7, min = 0, max = 100, truecut = 50,
locate = c("NPV", 0.9))
# And so on for other statistics such as Specificity and Positive Predictive
# Value.
```

MC.out.tabular — *Tabular organization of accuracy and consistency output from the* `LL.CA.MC()` *function.*

Description

Function that takes the output from the `LL.CA.MC()` function and organizes it in a table with accuracy and consistency indices represented by columns and categories as rows.

Usage

```
MC.out.tabular(x)
```

Arguments

x	The list-output from the `LL.CA.MC()` function.

Examples

```
# Generate some fictional data. Say, 1000 individuals take a test with a
# maximum score of 100 and a minimum score of 0.
set.seed(1234)
p.success <- rBeta.4P(1000, 0.1, 0.95, 5, 3)
for (i in 1:100) {
  if (i == 1) {
    rawdata <- matrix(nrow = 1000, ncol = 100)
  }
  rawdata[, i] <- rbinom(1000, 1, p.success)
}

# Estimate accuracy and consistency where the lowest category are scores
# below 50, second lowest 60, then 70, 80, and 90. Using the cba() function
# to estimate the reliability of this test, to use the LL.CA.MC() function
```

```
# or estimating diagnostic performance and consistency indices of
# classifications when using several cut-points:
output <- LL.CA.MC(rowSums(rawdata), cba(rawdata), seq(50, 90, 10), 0, 100)

# As this output can get quite verbose as the number of categories increase,
# the MC.out.tabular() function can be used to organize the output more
# concisely in a tabular format.
MC.out.tabular(output)
```

---

mdo *Calculate McDonald's Omega reliability-coefficient from supplied variables.*

Description

Calculates McDonalds's Omega reliability-coefficient of the sum-score from the Spearman one-factor model using the procedure outlined in McDonald (1999).

Usage

```
mdo(x, fit = FALSE)
```

Arguments

x	A data-frame or matrix of numerical values where rows represent respondents, and columns represent items.
fit	Logical. Default is `FALSE`. If `TRUE`, the output changes from a vector containing the Omega reliability-estimate to a list containing additional detailed information concerning the fitted factor model.

Value

If `fit = FALSE`, A vector of length 1 containing the estimated McDonalds's Omega reliability-coefficient for the sum-score of the supplied variables. If `fit = TRUE`, a list containing the Omega-coefficient reliability-estimate as the first entry, followed by the goodness-of-fit index (GFI), a two-row matrix containing the estimated factor-loadings and error-variances, and the observed and fitted covariance-matrices and the discrepancy matrix.

Note

Missing values are treated by passing `na.rm = TRUE` to the `var` function call and `use = "pairwise.complete.obs"` to the cov function call.

The function terminates with an error if there are negative covariance-matrix entries.

References

McDonald, R. P. (1999). Test Theory: A Unified Treatment. Routledge.

Examples

```
# Generate some fictional data.
set.seed(1234)
rawdata <- matrix(rnorm(500), ncol = 5)
common <- rnorm(100)
rawdata <- apply(rawdata, 2, function(x) {x + common})

# To estimate McDonald's Omega from this data:
mdo(rawdata)

# To retrieve additional information such as the GFI fit-index and model-
# parameter estimates:
mdo(rawdata, fit = TRUE)
```

MLA *Most Likely True Alpha Value Given Observed Outcome.*

Description

Given a fitted Standard (two-parameter) Beta Distribution, return the alpha shape-parameter value where the observed mean becomes the mode.

Usage

```
MLA(alpha, beta, x = NULL, n = NULL)
```

Arguments

alpha	Observed alpha-parameter value for fitted Standard Beta PDD.
beta	Observed beta-parameter value for fitted Standard Beta PDD.
x	Observed proportion-correct outcome.
n	Test-length.

Value

The Alpha shape-parameter value for the Standard Beta probability density distribution where the observed mean is the expected mode.

Examples

```
# Assuming a prior Standard (two-parameter) Beta distribution is fit, which
# yield an alpha parameter of 10 and a beta parameter of 8, calculate the
# true-alpha parameter most likely to have produced the observations:
MLA(a = 10, b = 8)
```

MLB — *Most Likely True Beta Value Given Observed Outcome.*

Description

Assuming a prior standard (two-parameter) Beta Distribution, return the beta shape-parameter value where the observed mean becomes the mode.

Usage

```
MLB(alpha, beta, x = NULL, n = NULL)
```

Arguments

alpha	Observed alpha-parameter value for fitted Standard Beta PDD.
beta	Observed beta-parameter value for fitted Standard Beta PDD.
x	Observed proportion-correct outcome.
n	Test-length.

Value

The Beta shape-parameter value for the Standard Beta probability density distribution where the observed mean is the expected mode.

Examples

```
# Assuming a prior Standard (two-parameter) Beta distribution is fit, which
# yield an alpha parameter of 10 and a beta parameter of 8, calculate the
# true-beta parameter most likely to have produced the observations:
MLB(a = 10, b = 8)
```

MLM — *Most Likely Mean of the Standard Beta PDD, Given that the Observation is Considered the Most Likely Observation of the Standard Beta PDD (i.e., the mode).*

Description

Assuming a prior Standard (two-parameter) Beta Distribution, returns the expected mean of the distribution under the assumption that the observed value is the most likely value of the distribution.

Usage

```
MLM(alpha, beta, x = NULL, n = NULL)
```

Arguments

alpha	Observed alpha value for fitted Standard Beta PDD.
beta	Observed beta value for fitted Standard Beta PDD.
x	Observed proportion-correct outcome.
n	Test-length.

Value

The expected mean of the Standard Beta probability density distribution, for which the observed mean is the most likely value.

Examples

```
# Assuming a prior Standard (two-parameter) Beta distribution is fit, which
# yield an alpha parameter of 10 and a beta parameter of 8, calculate the
# true-mean most likely to have produced the observations:
MLM(a = 10, b = 8)
```

---

observedmoments *Compute Moments of Observed Value Distribution.*

Description

Computes Raw, Central, or Standardized moment properties of a vector of observed scores.

Usage

```
observedmoments(
  x,
  type = c("raw", "central", "standardized"),
  orders = 4,
  correct = TRUE
)
```

Arguments

x	A vector of values, the distribution of which moments are to be calculated.
type	A character vector determining which moment-types are to be calculated. Permissible values are "raw", "central", and "standardized".
orders	The number of moment-orders to be calculated for each of the moment-types.
correct	Logical. Whether to include bias correction in estimation of orders. Default is TRUE.

Value

A list of moment types, each a list of moment orders.

Examples

```
# Generate some fictional data. Say, 100 individuals take a test with a
# maximum score of 100 and a minimum score of 0.
set.seed(1234)
testdata <- rbinom(100, 100, rBeta.4P(100, 0.25, 0.75, 5, 3))
hist(testdata, xlim = c(0, 100))

# To compute the first four raw, central, and standardized moments for this
# distribution of observed scores using observedmoments():
observedmoments(x = testdata, type = c("raw", "central", "standardized"),
orders = 4, correct = TRUE)
```

---

pBeta.4P *Cumulative Probability Function under the Four-Parameter Beta Probability Density Distribution.*

Description

Function for calculating the proportion of observations up to a specifiable quantile under the Four-Parameter Beta Distribution.

Usage

```
pBeta.4P(q, l, u, alpha, beta, lower.tail = TRUE)
```

Arguments

`q`	The quantile or a vector of quantiles for which the proportion is to be calculated.
`l`	The first (lower) location parameter.
`u`	The second (upper) location parameter.
`alpha`	The first shape parameter.
`beta`	The second shape parameter.
`lower.tail`	Whether the proportion to be calculated is to be under the lower or upper tail. Default is `TRUE` (lower tail).

Value

A vector of proportions of observations falling under specified quantiles under the four-parameter Beta distribution.

Examples

```
# Assume some variable follows a four-parameter Beta distribution with
# location parameters l = 0.25 and u = 0.75, and shape parameters alpha = 5
# and beta = 3. To compute the cumulative probability at a specific point of
# the distribution (e.g., 0.5)
# using pBeta.4P():
pBeta.4P(q = 0.5, l = 0.25, u = 0.75, alpha = 5, beta = 3)
```

pBetaBinom *Cumulative Probability Function under the Beta-Binomial Probability Distribution.*

Description

Function for calculating the proportion of observations up to a specifiable quantile under the Beta-Binomial Probability Distribution.

Usage

```
pBetaBinom(q, N, l, u, alpha, beta, lower.tail = TRUE)
```

Arguments

q	The quantile or a vector of quantiles for which the proportion is to be calculated.
N	The total number of trials.
l	The first (lower) location parameter.
u	The second (upper) location parameter.
alpha	The first shape parameter.
beta	The second shape parameter.
lower.tail	Whether the proportion to be calculated is to be under the lower or upper tail. Default is TRUE (lower tail).

Value

A vector of proportions of observations falling under specified quantiles under the four-parameter Beta distribution.

Examples

```
# Assume some variable follows a Beta-Binomial distribution with number of
# trials = 50, and probabilities of successful trials are drawn from a four-
# parameter Beta distribution with location parameters l = 0.25 and u =
# 0.75, and shape parameters alpha = 5 and beta = 3. To compute the
# cumulative probability at a specific point of the distribution (e.g., 25):
pBetaBinom(q = 25, N = 50, l = .25, u = .75, alpha = 5, beta = 3)
```

`pBetaMS` *Probability of Some Specific Observation under the Standard Beta PDD with Specific Mean and Variance.*

Description

Calculates the probability of some specific observation falling under a specified interval ([0, x] or [x, 1]) under the Standard Beta probability density distribution with defined mean and variance or standard deviation.

Usage

```
pBetaMS(q, mean, variance = NULL, sd = NULL, lower.tail = TRUE)
```

Arguments

`q`	A specific point on the x-axis of the Standard Beta probability density distribution with a defined mean and variance.
`mean`	The mean of the target Standard Beta probability density distribution.
`variance`	The variance of the target Standard Beta probability density distribution.
`sd`	The standard deviation of the target Standard Beta probability density distribution.
`lower.tail`	Whether the density that should be considered is between the lower-end (i.e., [0 -> x]) or the higher-end of the distribution (i.e., [x -> 1]).

Value

A value representing the probability of a random draw from the Standard Beta probability density distribution with a defined mean and variance being from one of two defined intervals (i.e., [0 -> x] or [x -> 1]).

Examples

```
# To compute the proportion of the density under the lower-end tail of a
# point along the Standard (two-parameter) PDD (e.g., 0.5) with mean of 0.6
# and variance of 0.04:
pBetaMS(q = 0.5, mean = 0.6, variance = 0.04)
```

pGammaBinom — *Cumulative probability density function under the Gamma-extended Binomial distribution.*

Description

Extends the cumulative Binomial probability mass function to positive non-integers, effectively turning the mass-function into a density-function.

Usage

```
pGammaBinom(q, size, prob, lower.tail = TRUE)
```

Arguments

q	Vector of quantiles.
size	Number of "trials" (zero or more). Need not be integer.
prob	Probability of "success" on each "trial". Need not be integer.
lower.tail	Logical. If TRUE (default), probabilities are P[X<x], otherwise, P[X >= x]. Note that this differs from base-R `binom()` functions.

References

Loeb, D. E. (1992). A generalization of the binomial coefficients. Discrete Mathematics, 105(1-3).

Examples

```
# Assume some variable follows a Gamma-Binomial  distribution with
# "number of trials" = 10.5 and probability of "success" for each "trial"
# = 0.75, to compute the cumulative probability to attain a "number of
# success" below a specific point (e.g., less than 7.5 "successes":
pGammaBinom(q = 7.5, size = 10.5, prob = 0.75)

# Conversely, to attain a value at or above 7.5:
pGammaBinom(q = 7.5, size = 10.5, prob = 0.75, lower.tail = FALSE)
```

qBeta.4P — *Quantile Given Probability Under the Four-Parameter Beta Distribution.*

Description

Function for calculating the quantile (i.e., value of `x`) for a given proportion (i.e., the value of `y`) under the Four-Parameter Beta Distribution.

Usage

```
qBeta.4P(p, l, u, alpha, beta, lower.tail = TRUE)
```

Arguments

p	A vector (or single value) of proportions or probabilities for which the corresponding value of x (i.e., the quantiles) are to be calculated.
l	The first (lower) location parameter.
u	The second (upper) location parameter.
alpha	The first shape parameter.
beta	The second shape parameter.
lower.tail	Logical. Whether the quantile(s) to be calculated is to be under the lower or upper tail. Default is TRUE (lower tail).

Value

A vector of quantiles for specified probabilities or proportions of observations under the four-parameter Beta distribution.

Examples

```
# Assume some variable follows a four-parameter Beta distribution with
# location parameters l = 0.25 and u = 0.75, and shape parameters alpha = 5
# and beta = 3. To compute the quantile at a specific point of the
# distribution (e.g., 0.5) using qBeta.4P():
qBeta.4P(p = 0.5, l = 0.25, u = 0.75, alpha = 5, beta = 3)
```

---

qBetaMS *Quantile Containing Specific Proportion of the Distribution, Given a Specific Probability of the Standard Beta PDD with Specific Mean and Variance or Standard Deviation.*

Description

Calculates the quantile corresponding to a specific probability of some observation falling within the [0, x] (lt = TRUE) or [x, 1] (lt = FALSE) interval under the Standard Beta probability density distribution with defined mean and variance or standard deviation.

Usage

```
qBetaMS(p, mean, variance = NULL, sd = NULL, lower.tail = TRUE)
```

Arguments

`p`	A value of probability marking the point of the Y-axis to correspond to the X-axis.
`mean`	The mean of the target Standard Beta probability density distribution.
`variance`	The variance of the target Standard Beta probability density distribution.
`sd`	The standard deviation of the target Standard Beta probability density distribution.
`lower.tail`	Logical. Specifies which end of the tail for which to calculate quantile. Default is `TRUE` (meaning, find q for lower tail.)

Value

A numeric value representing the quantile for which the specified proportion of observations fall within.

Examples

```
# To compute the quantile at a specific point (e.g., 0.5) along the Standard
# (two-parameter) PDD with mean of 0.6 and variance of 0.04:
qBetaMS(p = 0.5, mean = 0.6, variance = 0.04)
```

`qGammaBinom` *Quantile function for the Gamma-extended Binomial distribution.*

Description

Quantile function for the Gamma-extended Binomial distribution.

Usage

```
qGammaBinom(p, size, prob, lower.tail = TRUE, precision = 1e-07)
```

Arguments

`p`	Vector of probabilities.
`size`	Number of "trials" (zero or more, including positive non-integers).
`prob`	Probability of success on each "trial".
`lower.tail`	Logical. If TRUE (default), probabilities are P[X < x], otherwise P[X > x].
`precision`	The precision with which the quantile is to be calculated. Default is 1e-7 (i.e., search terminates when there is no registered change in estimate at the seventh decimal). Tuning this value will impact the time it takes for the search algorithm to arrive at an estimate.

Note

This function uses a bisection search-algorithm to find the number of successes corresponding to the specified quantile(s). This algorithm is inefficient with respect to the number of iterations required to converge on the solution. More efficient algorithms might be added in later versions.

References

Loeb, D. E. (1992). A generalization of the binomial coefficients. Discrete Mathematics, 105(1-3).

Examples

```
# For a Gamma-extended Binomial distribution with number of trials = 10 and
# probability of success per trial of 0.75, calculate the number of success-
# ful trials at or below the 25% quantile:
qGammaBinom(p = 0.25, size = 10, prob = 0.75)

# Conversely, for a Gamma-extended Binomial distribution with number of
# trials = 10 and probability of success per trial of 0.75, calculate the
# number of successful trials at or above the 25% quantile:
qGammaBinom(p = 0.25, size = 10, prob = 0.75, lower.tail = FALSE)
```

R.ETL — *Model implied reliability from Livingston and Lewis' "Effective Test Length".*

Description

Calculate model-implied reliability given mean, variance, the minimum and maximum possible scores, and the effective test length.

Usage

```
R.ETL(mean, variance, min = 0, max = 1, ETL)
```

Arguments

`mean`	The mean of the observed-score distribution.
`variance`	The variance of the observed-score distribution.
`min`	The lower-bound (minimum possible value) of the observed-score distribution. Default is 0 (assuming observed scores represent proportions).
`max`	The upper-bound (maximum possible value) of the observed-score distribution. Default is 1 (assuming observed scores represent proportions).
`ETL`	The effective test length as defined by Livingston and Lewis (1995).

Value

An estimate of the reliability of a test, given the effective test length, mean, variance, and minimum and maximum possible scores of the observed-score distribution..

References

Livingston, Samuel A. and Lewis, Charles. (1995). Estimating the Consistency and Accuracy of Classifications Based on Test Scores. Journal of Educational Measurement, 32(2).

Examples

```
# Generate some fictional data. Say, 100 individuals take a test with a
# maximum score of 100 and a minimum score of 0.
set.seed(1234)
testdata <- rbinom(100, 100, rBeta.4P(100, .25, .75, 5, 3))
hist(testdata, xlim = c(0, 100))

# From the data-generating script above, the effective test length is 100.
# To estimate and retrieve the model-implied reliability using R.ETL():
R.ETL(mean = mean(testdata), variance = var(testdata), min = 0, max = 100,
ETL = 100)
```

---

rBeta.4P — *Random Number Generation under the Four-Parameter Beta Probability Density Distribution.*

Description

Function for generating random numbers from a specified Four-Parameter Beta Distribution.

Usage

```
rBeta.4P(n, l, u, alpha, beta)
```

Arguments

n	Number of draws.
l	The first (lower) location parameter.
u	The second (upper) location parameter.
alpha	The alpha (first) shape parameter.
beta	The beta (second) shape parameter.

Value

A vector with length n of random values drawn from the Four-Parameter Beta Distribution.

Examples

```
# Assume some variable follows a four-parameter Beta distribution with
# location parameters l = 0.25 and u = 0.75, and shape parameters alpha = 5
# and beta = 3. To draw a random value from this distribution using
# rBeta.4P():
rBeta.4P(n = 1, l = 0.25, u = 0.75, alpha = 5, beta = 3)
```

`rBetaBinom` *Random Number Generation under the Beta-Binomial Probability Mass Distribution.*

Description

Random Number Generation under the Beta-Binomial Probability Mass Distribution.

Usage

```
rBetaBinom(n, N, l, u, alpha, beta)
```

Arguments

`n`	Number of draws.
`N`	Number of trials.
`l`	The first (lower) location parameter.
`u`	The second (upper) location parameter.
`alpha`	The alpha (first) shape parameter.
`beta`	The beta (second) shape parameter.

Value

A vector with length `n` of random values drawn from the Beta-Binomial Distribution.

Examples

```
# To draw a sample of 50 values from a Beta-Binomial distribution with
# number of trials = 100, and with success-probabilities drawn from a
# Four-Parameter Beta distribution with location parameters l = 0.25 and
# u = 0.95, and shape-parameters alpha = 5 and beta = 3:
rBetaBinom(n = 50, N = 100, l = 0.25, u = 0.95, alpha = 5, beta = 3)
```

`rBetaMS` *Random Draw from the Standard Beta PDD With Specific Mean and Variance.*

Description

Draws random samples of observations from the Standard Beta probability density distribution with defined mean and variance.

Usage

```
rBetaMS(n, mean, variance = NULL, sd = NULL)
```

Arguments

n	Number of observations to be drawn from under the Standard Beta PDD.
mean	The mean of the target Standard Beta probability density distribution.
variance	The variance of the target Standard Beta probability density distribution.
sd	The standard deviation of the target Standard probability density distribution.

Value

A vector of length n, each value representing a random draw from the Standard Beta probability density distribution with defined mean and variance.

---

rGammaBinom *Random number generation under the Gamma-extended Binomial distribution.*

Description

Random number generation under the Gamma-extended Binomial distribution.

Usage

```
rGammaBinom(n, size, prob, precision = 1e-04)
```

Arguments

n	Number of observations.
size	Number of "trials" (zero or more). Need not be integer.
prob	Probability of "success" on each "trial". Need not be integer.
precision	The precision with which the quantile is to be calculated. Default is 1e-4 (i.e., search terminates when there is no registered change in estimate at the fourth decimal). Tuning this value will impact the time it takes for the search algorithm to arrive at an estimate.

Note

Calls qGammaBinom(), which makes the random draw slower than what one might be used to (since qGammaBinom() calls pGammaBinom() and employs a search-algorithm to find the appropriate value down to a specifiable level of precision).

Examples

```
# Assume some variable follows a Gamma-Binomial distribution with
# "number of trials" = 10.5 and probability of "success" for each "trial"
# = 0.75 To draw a random value from this distribution:
rGammaBinom(n = 1, size = 10, prob = 0.75)
```

tsm — *Proportional true-score distribution raw moments from Livingston and Lewis' effective test-score and effective test-length.*

Description

An implementation of Lords (1965, p. 265) equation 37 for estimating the raw moments of the true-score distribution, modified to function for the Livingston and Lewis approach.

Usage

```
tsm(x, r, n, method = "product")
```

Arguments

`x`	The effective test-score of test-takers.
`r`	The moment-order that is to be calculated (where 1 is the mean, 2 is the raw variance, 3 is the raw skewness, etc.).
`n`	The effective test-length.
`method`	The method by which the descending factorials are to be calculated. Default is `"product"` which uses direct arithmetic. Alternative is "gamma" which calculates the descending factorial using the Gamma function. The alternative method might be faster but might fail because the Gamma function is not defined for negative integers (returning Inf).

References

Lord, F. M. (1965). A strong true-score theory, with applications. Psychometrika. 30(3). pp. 239–270. doi: 10.1007/BF02289490

Livingston, Samuel A. and Lewis, Charles. (1995). Estimating the Consistency and Accuracy of Classifications Based on Test Scores. Journal of Educational Measurement, 32(2).

Examples

```
# Examine the raw moments of the underlying Beta distribution that is to provide the basis for
# observed-scores:
betamoments(alpha = 5, beta = 3, l = 0.25, u = 0.75, types = "raw")

# Generate observed-scores from true-scores by passing the true-scores as binomial probabilities
# for the rbinom function.
set.seed(1234)
obs.scores <- rbinom(1000, 100, rBeta.4P(1000, 0.25, 0.75, 5, 3))
# Examine the raw moments of the observed-score distribution.
observedmoments(obs.scores, type = "raw")

# First four estimated raw moment of the proportional true-score distribution from the observed-
# score distribution. As all items are equally difficult, the effective test-length is equal to
```

```
# the actual test-length.
tsm(x = obs.scores, r = 1, n = 100)
tsm(x = obs.scores, r = 2, n = 100)
tsm(x = obs.scores, r = 3, n = 100)
tsm(x = obs.scores, r = 4, n = 100)
# Which is fairly close to the true raw moments of the proportional true-score distribution
# calculated above.
```

UABMSL *Upper Location Parameter Given Shape Parameters, Mean, Variance, and Lower Location Parameter of a Four-Parameter Beta PDD.*

Description

Calculates the upper-bound value required to produce a Beta probability density distribution with defined moments and parameters. Be advised that not all combinations of moments and parameters can be satisfied (e.g., specifying mean, variance, skewness and kurtosis uniquely determines both location-parameters, meaning that the value of the upper-location parameter will take on which ever value it must, and cannot be specified).

Usage

```
UABMSL(
  alpha = NULL,
  beta = NULL,
  mean = NULL,
  variance = NULL,
  skewness = NULL,
  kurtosis = NULL,
  l = NULL,
  sd = NULL
)
```

Arguments

`alpha`	The alpha shape-parameter of the target Beta probability density distribution.
`beta`	The beta shape-parameter of the target Beta probability density distribution.
`mean`	The mean (first raw moment) of the target Standard Beta probability density distribution.
`variance`	The variance (second central moment) of the target Standard Beta probability density distribution.
`skewness`	The skewness (third standardized moment) of the target Beta probability density distribution.
`kurtosis`	The kurtosis (fourth standardized moment) of the target Beta probability density distribution.

l	The lower-bound of the Beta distribution. Default is NULL (i.e., does not take a specified l-parameter into account).
sd	Optional alternative to specifying var. The standard deviation of the target Standard Beta probability density distribution.

Value

A numeric value representing the required value for the Beta upper location-parameter (u) in order to produce a Beta probability density distribution with the target moments and parameters.

Examples

```
# Generate some fictional data.
set.seed(1234)
testdata <- rBeta.4P(100000, 0.25, 0.75, 5, 3)
hist(testdata, xlim = c(0, 1), freq = FALSE)

# Suppose you know three of the four necessary parameters to fit a four-
# parameter Beta distribution (i. e., l = 0.25, alpha = 5, beta = 3) to this
# data. To find the value for the necessary u parameter, estimate the mean
# and variance of the distribution:
M <- mean(testdata)
S2 <- var(testdata)

# To find the l parameter necessary to produce a four-parameter Beta
# distribution with the target mean, variance, and u, alpha, and beta
# parameters using the LMSBAU() function:
(u <- UABMSL(alpha = 5, beta = 3, mean = M, variance = S2, l = 0.25))
curve(dBeta.4P(x, 0.25, u, 5, 3), add = TRUE, lwd = 2)
```

Index

afac, 3
AMS, 4
AUC, 5

Beta.2p.fit, 6
Beta.4p.fit, 7
Beta.gfx.poly.cdf, 8
Beta.gfx.poly.pdf, 9
Beta.gfx.poly.qdf, 10
Beta.tp.fit, 11
betabinomialmoments, 13
betamedian, 14
betamode, 15
betamoments, 15
binomialmoments, 17
BMS, 18

caStats, 19
cba, 20
ccStats, 21
confmat, 22

dBeta.4P, 23
dBeta.pBeta, 24
dBeta.pBinom, 25
dBeta.pGammaBinom, 26
dBetaBinom, 28
dBetaMS, 28
dfac, 29
dGammaBinom, 30

ETL, 31

gchoose, 32

LABMSU, 32
LL.CA, 34
LL.CA.MC, 37
LL.ROC, 40

MC.out.tabular, 42
mdo, 43
MLA, 44
MLB, 45
MLM, 45

observedmoments, 46

pBeta.4P, 47
pBetaBinom, 48
pBetaMS, 49
pGammaBinom, 50

qBeta.4P, 50
qBetaMS, 51
qGammaBinom, 52

R.ETL, 53
rBeta.4P, 54
rBetaBinom, 55
rBetaMS, 55
rGammaBinom, 56

tsm, 57

UABMSL, 58