Package ‘usefr’

July 2, 2019

Type Package

Title Utility Functions for Statistical Analyses

Version 0.1.0

Author Robersy Sanchez

Authors@genomaths c(person(``Robersy", ``Sanchez", email = ``robersy@gmail.com",
role = c(``aut", ``cre", ``cph"))

Maintainer Robersy Sanchez <robersy@gmail.com>

Description It provides some useful functions frequently needed in the
downstream statistical analyses.

License file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Suggests testthat,
knitr,
rmarkdown

Imports BiocParallel,
minpack.lm,
numDeriv,
utils,
stats,
graphics

VignetteBuilder knitr

R topics documented:

AICmodel 2
BICmodel 3
bicopulaGOF 4
fitCDF 7
fitMixDist 10

ggamma 12
hnorm 14
mcgoftest 16
mixtdistr 18
ppCplot 20

Index **23**

`AICmodel` *Akaike's Information Criterion (AIC)*

Description

this function permits the estimation of the AIC for models for which the function 'AIC' from the 'stats' package does not work.

Usage

```
AICmodel(model = NULL, residuals = NULL, np = NULL)
```

Arguments

`model`	if provided, it is an R object from where the residuals and model parameters can be retrieved using resid(model) and coef(model), respectively.
`residuals`	if provided, it is numerical vector with the residuals: residuals = observed.values - predicted.values, where predicted values are estimated from the model. If the parameter 'model' is not provided, then this parameter must be provided.
`np`	number of model parameters. If the parameter 'model' is not provided, then 'np' and 'residuals' must be provided.

Details

if for a given model 'm' AIC(m) works, then AICmodel(m) = AIC(m).

Value

AIC numerical value

Author(s)

Robersy Sanchez (`https://genomaths.com`).

Examples

```
set.seed(77)
x = runif(100, 1, 5)
y = 2 * exp(-0.5 * x) + runif(100, 0, 0.1)
plot(x, y)

nlm <- nls(Y ~ a * exp( b * X), data = data.frame(X=x, Y=y),
            start=list(a=1.5, b=-0.7),
            control=nls.control(maxiter=10^4, tol=1e-05),
            algorithm="port")
## The estimations of Akaike information criteria given by 'AIC' function
## from stats' R package and from 'AICmodel' function are equal.
AICmodel(nlm) == AIC(nlm)

## Now, using residuals from the fitted model:
res = y - coef(nlm)[1] * exp(coef(nlm)[2] * x)

AICmodel(residuals=res, np=2) == AIC(nlm)
```

`BICmodel` *Bayesian Information Criterion (BIC)*

Description

this function permits the estimation of the BIC for models for which the function 'BIC' from 'stats' packages does not work.

Usage

```
BICmodel(model = NULL, residuals = NULL, np = NULL)
```

Arguments

`model`	if provided, it is an R object from where the residuals and model parameters can be retrieved using resid(model) and coef(model), respectively.
`residuals`	if provided, it is numerical vector with the residuals: residuals = observe.values - predicted.values, where predicted values are estimated from the model. If the parameter 'model' is not provided, then this parameter must be provided.
`np`	number of model parameters. If the parameter 'model' is not provided, then 'np' and 'residuals' must be provided.

Details

if for a given model 'm' BIC(m) works, then BICmodel(m) = BIC(m).

Value

BIC numerical value

Author(s)

Robersy Sanchez (https://genomaths.com).

Examples

```
set.seed(77)
x = runif(100, 1, 5)
y = 2 * exp(-0.5 * x) + runif(100, 0, 0.1)
plot(x, y)

nlm <- nls(Y ~ a * exp( b * X), data = data.frame(X = x, Y = y),
            start = list( a = 1.5, b = -0.7),
            control = nls.control(maxiter = 10^4, tol = 1e-05),
            algorithm = "port")
## The estimations of Akaike information criteria given by BIC' function
## from stats' R package and from 'AICmodel' function are equals.
BICmodel(nlm) == BIC(nlm)

## Now, using residuals from the fitted model:
res = y - coef(nlm)[1] * exp(coef(nlm)[2] * x)

BICmodel(residuals = res, np = 2) == BIC(nlm)
```

---

bicopulaGOF *Goodness of fit for Bidimensional Copula with Known Margins*

Description

Goodness-of-fit (GOF) tests for a two-dimensional copula based, by default, on the knowledge of the marginal probability distributions. Several functionalities/tools from copula-package are integrated to perform the GOF of copulas that includes specific margin parameter settings. In terms of copula-package vocabulary, these are GOF for copula objects from class Mvdc (also called non-free copulas).

Usage

```
bicopulaGOF(x, y, copula = NULL, margins = NULL, paramMargins = NULL,
  sample.size = NULL, nboots = 10, approach = c("adchisq", "adgamma",
  "chisq", "rmse", "Sn", "SnB", "SnC"), Rosenblatt = FALSE,
  breaks = 12, method = "ml", num.cores = 1L, tasks = 0,
  seed = 123, verbose = TRUE, ...)
```

Arguments

x	Numerical vector with the observations from the first margin distribution.
y	Numerical vector with the observations from the second margin distribution.

`copula`	A copula object from class `Mvdc` or string specifying all the name for a copula from package `copula-package`.
`margins`	A character vector specifying all the parametric marginal distributions. See details below.
`paramMargins`	A list whose each component is a list (or numeric vectors) of named components, giving the parameter values of the marginal distributions. See details below.
`sample.size`	The size of the samples used for each sampling. It is not required for the approaches: "Sn", "SnB", and "SnC"; see below.
`nboots`	The number of booststrap resampling to perform.
`approach`	a character string specifying the goodness-of-fit test statistic to be used, which has to be one (or a unique abbreviation) of following: "adchisq", "adgamma", "Sn", "SnB", "SnC", "chisq", and "rmse". With the exception of *chisq* and *rmse*, all the other statistics are the same as in functions `gofTstat` and `gofCopula`. The test using *chisq* implement the approach described in reference [1].
`Rosenblatt`	The Anderson–Darling statistic approach using Rosenblatt transformation is normally used for the GOF in function `gofCopula` from `copula-package` package. since, the current function applies a parametric bootstrap approach generating random variates from the analytical expression for the margin CDFs, the test does not depend on the theoretical distribution of the Anderson–Darling statistic. Simulations suggest, so far, that the application of Rosenblatt transformation may not be needed in this case. SO, the desicion on whether to apply the Rosenblatt transformation (computational expensive for big datasets) is left to the users.
`breaks`	A single number giving the number of bins for the computation of the Pearson's Chi-squared statistic as suggested in reference [1]. Bascally, it is used to split the unit square [0, 1]^2 into bins/regions.
`method`	A character string specifying the estimation method to be used to estimate the dependence parameter(s); see `fitCopula`.
`num.cores, tasks`	Paramaters for parallele computation using package `BiocParallel-package`: the number of cores to use, i.e. at most how many child processes will be run simultaneously (see `bplapply` and the number of tasks per job (only for Linux OS).
`seed`	An integer used to set a 'seed' for random number generation.
`verbose`	if verbose, comments and progress bar will be printed.

Details

Notice that `copula-package` already have function `gofCopula` to perform GOF. However, its use can be computational expensive for big datasets.

Value

The statistic value estimated for the observations, and the estimated bootstrap p.value.

Author(s)

Robersy Sanchez (https://genomaths.com).

References

1. Jaworski, P. Copulae in Mathematical and Quantitative Finance. 213, d (2013).
2. Wang, Y. et al. Multivariate analysis of joint probability of different rainfall frequencies based on copulas. Water (Switzerland) 9, (2017).

See Also

ppCplot, gofCopula, fitCDF, fitdistr, and fitMixDist

Examples

```
require(stats)

set.seed(12)
margins = c("norm", "norm")
## Random variates from normal distributions
X <- rnorm(2*1e3, mean = 0, sd = 10)
Y <- rnorm(2*1e3, mean = 0, sd = 10)

parMargins = list( list(mean = 0, sd = 10),
                   list(mean = 0, sd = 10))

bicopulaGOF(x = X, y = Y, copula = "normalCopula", sample.size = 1e2,
            margins = margins, paramMargins = parMargins, nboots = 999,
            Rosenblatt = TRUE, approach = "adgamma", num.cores = 1L)

bicopulaGOF(x = X, y = Y, copula = "normalCopula", sample.size = 1e2,
            margins = margins, paramMargins = parMargins, nboots = 999,
            Rosenblatt = FALSE, approach = "adgamma", num.cores = 1L)

## --- Non-parallel expensive computation ---- -
# require(copula)
#
# U <- pobs(cbind(X, Y)) #' # Compute the pseudo-observations
# fit <- fitCopula(normalCopula(), U, method = 'ml')
# U <- cCopula(u = U, copula = fit@copula) #' # Rosenblatt transformation
#
# set.seed(123)
# system.time(
#   gof <- gofCopula(copula = fit@copula, x = U, N = 99, method = "Sn",
#             simulation = "pb")
# )
# gof
## About
##    user  system elapsed
## 103.370   0.613 105.022
#
```

```
## --- Parallel computation with 2 cores ---- -
## Same algorithm as in 'gofCopula' adapted for parallel computation
# system.time(
#   gof <- bicopulaGOF(x = X, y = Y, copula = "normalCopula",
#               margins = margins, paramMargins = parMargins, nboots = 99,
#               Rosenblatt = TRUE, approach = "Sn", seed = 123,
#               num.cores = 2L)
# )
# gof
## About
##  user  system elapsed
## 2.491   0.100  51.185
##
## Same algorithm as in 'gofCopula' adapted for parallel computation and
## Rosenblatt = FALSE
# system.time(
#   gof <- bicopulaGOF(x = X, y = Y, copula = "normalCopula",
#               margins = margins, paramMargins = parMargins, nboots = 99,
#               Rosenblatt = FALSE, approach = "Sn", seed = 123,
#               num.cores = 2L)
# )
# gof
```

---

fitCDF *Nonlinear fit of a commulative distribution function*

Description

Usually the parameter estimation of a cummulative distribution function (*CDF*) are accomplished using the corresponding probability density function (*PDF*). DIfferent optimization algorithms can be used to accomplished this task and different algorithms can yield different esitmated parameters. Hence, why not try to fit the CDF directly?

Usage

```
fitCDF(varobj, distNames, plot = TRUE, plot.num = 1,
  only.info = FALSE, maxiter = 10^4, maxfev = 1e+05, ptol = 1e-12,
  verbose = TRUE)
```

Arguments

`varobj`	A a vector containing observations, the variable for which the CDF parameters will be estimated.
`distNames`	a vector of distribution numbers to select from the listed below in details section, e.g. c(1:10, 15)
`plot`	Logic. Default TRUE. Whether to produce the plots for the best fitted CDF.
`plot.num`	The number of distributions to be plotted.

`only.info`	Logic. Default TRUE. If true, only information about the parameter estimation is returned.
`maxiter, maxfev, ptol`	Parameters to ontrol of various aspects of the Levenberg-Marquardt algorithm through function `nls.lm.control` from *minpack.lm* package.
`verbose`	Logic. If TRUE, prints the function log to stdout

Details

The nonlinear fit (NLF) problem for CDFs is addressed with Levenberg-Marquardt algorithm implemented in function `nls.lm` from package *minpack.lm*. This function is inspired in a script for the function `fitDistr` from the package propagate [1]. Some parts or script ideas from function `fitDistr` are used, but here we to estimate CDF and not the PDF as in the case of "`fitDistr`. A more informative are also incorporated. The studentized residuals are provided as well. The list (so far) of possible CDFs is:

1. Normal (Wikipedia)
2. Log-normal (Wikipedia)
3. Half-normal (Wikipedia). An Alternatively using a scaled precision (inverse of the variance) parametrization (to avoid issues if σ is near zero), obtained by setting $\theta = sqrt(\pi)/\sigma * sqrt(2)$.
4. Generalized Normal (Wikipedia)
5. T-Generalized Normal [2].
6. Laplace (Wikipedia)
7. Gamma (Wikipedia)
8. 3P Gamma [3].
9. Generalized 4P Gamma [3] (Wikipedia)
10. Generalized 3P Gamma [3].
11. Weibull (Wikipedia)
12. 3P Weibull (Wikipedia)
13. Beta (Wikipedia)
14. 3P Beta (Wikipedia)
15. 4P Beta (Wikipedia)
16. Beta-Weibull ReliaWiki
17. Generalized Beta (Wikipedia)
18. Rayleigh (Wikipedia)
19. Exponential (Wikipedia)
20. 2P Exponential (Wikipedia)

Value

After return the plots, a list with following values is provided:

- aic: Akaike information creterion
- fit: list of results of fitted distribution, with parameter values
- bestfit: the best fitted distribution according to AIC
- fitted: fitted values from the best fit
- rstudent: studentized residuals
- residuals: residuals

After x = fitCDF(varobj, ...), attributes(x$bestfit) yields: $names [1] "par" "hessian" "fvec" "info" "message" "diag" "niter" "rsstrace" "deviance" $class [1] "nls.lm" And fitting details can be retrived with summary(x$bestfit)

Author(s)

Robersy Sanchez (`https://genomaths.com`).

References

1. Andrej-Nikolai Spiess (2014). propagate: Propagation of Uncertainty. R package version 1.0-4. http://CRAN.R-project.org/package=propagate
2. Abramowitz, M. and Stegun, I. A. (1972) Handbook of Mathematical Functions. New York: Dover. Chapter 6: Gamma and Related Functions.
3. Hand-book on STATISTICAL DISTRIBUTIONS for experimentalists (pag 73) by Christian Walck. Particle Physics Group Fysikum. University of Stockholm (e-mail: walck@physto.se).

See Also

`fitdistr` and `fitMixDist` and for goodness-of-fit: `mcgoftest`.

Examples

```
set.seed(1230)
x1 <- rnorm(10000, mean = 0.5, sd = 1)
cdfp <- fitCDF(x1, distNames = "Normal", plot = FALSE)
summary(cdfp$bestfit)
```

`fitMixDist` *Nonlinear fit of Mixture distributions*

Description

This function performs the nonlinear fit of mixture distributions exploiting a firth approach on parameterized finite Gaussian mixture models obtained through the function `Mclust` from package *mclust*.

Usage

```
fitMixDist(X, args = list(norm = c(mean = NA, sd = NA), weibull = c(shape
  = NA, scale = NA)), npoints = 100, maxiter = 1024,
  prior = priorControl(), ftol = 1e-14, ptol = 1e-14,
  maxfev = 1e+05, equalPro = FALSE, eps, tol, usepoints, seed = 123,
  dens = TRUE, kmean = FALSE, verbose = TRUE, ...)
```

Arguments

`X`	numerical vector
`npoints`	number of points used in the fit of the density function or NULL. These are used as histogram break points to estimate the empirical density values. If *npoints* = NULL and *dens* = TRUE, then. Kernel Density Estimation function `density` from *stats* package is used to estimate the empirical density. Default value is 100.
`maxiter`	positive integer. Termination occurs when the number of iterations reaches maxiter. Default value: 1024.
`prior`	Same as in `Mclust` function.
`ftol`	non-negative numeric. Termination occurs when both the actual and predicted relative reductions in the sum of squares are at most ftol. Therefore, ftol measures the relative error desired in the sum of squares. Default value: 1e-12
`ptol`	non-negative numeric. Termination occurs when the relative error between two consecutive iterates is at most ptol. Therefore, ptol measures the relative error desired in the approximate solution. Default value: 1e-12.
`maxfev`	Integer; termination occurs when the number of calls to fn has reached maxfev. Note that nls.lm sets the value of maxfev to 100*(length(par) + 1) if maxfev = integer(), where par is the list or vector of parameters to be optimized.
`equalPro`	An argument to pass to `emControl` function. Logical variable indicating whether or not the mixing proportions are equal in the model. Default: equalPro = FALSE.
`eps, tol`	Arguments to pass to `emControl` function.
`usepoints`	Integer. Computation by function `Mclust` could take long time when the sample size is about >= 10000. This number can be used to extract a random sample of size 'usepoints' and to do the estimation with it.

`seed`	Seed for random number generation.
`dens`	Logic. Whether to use fit the 'PDF' or 'CDF'. Default is TRUE.
`kmean`	Logic. Whether to use `kmeans` algorithm to perform the estimation in place of `Mclust`. Deafult is FALSE.
`verbose`	if TRUE, prints the function log to stdout and a progress bar
`...`	Further arguments to pass to other functions like `Mclust` and `density`.
`arg`	A list of named vectors with the corresponding named distribution parameters values. The names of the vector of parameters and the parameter names must correspond to defined functions. For example, if one of the involved distributions is the gamma density (see `GammaDist`), then the corresponding vector of parameters must be gamma = c(shape = 'some value', scale = 'some value'). For the following distributions, the arguments can be provided with NULL values: • "norm" (Wikipedia) • "halfnorm" (Wikipedia). • "gnorm" (Wikipedia) • "gamma" (Wikipedia) • "beta" (Wikipedia) • "laplace" (Wikipedia) • "weibull" (Wikipedia) • "rayleigh" (Wikipedia) • "exp" (Wikipedia) Notice that the distribution given names correspond to the root-names as given for R functions. For example, 'gamma' is the root-name for functions `GammaDist`. See example, for more details.

Details

The approch tries to fit the proposed mixture distributions using a modification of Levenberg-Marquardt algorithm implemented in function `nls.lm` from *minpack.lm* package that is used to perform the nonlinear fit. Cross-validations for the nonlinear regressions (R.Cross.val) are performed as described in reference [1]. In addition, Stein's formula for adjusted R squared (rho) was used as an estimator of the average cross-validation predictive power [1]. Notice that the parameter values must be given in way *understandable* by the set of functions `mixtdistr` (see the example below)

Value

A list with the model table with coefficients and goodness-of-fit results, the fitted model returned by function `nls.lm`, and a named list of fitted arguments.

Author(s)

Robersy Sanchez (`https://genomaths.com`).

References

1. Stevens JP. Applied Multivariate Statistics for the Social Sciences. Fifth Edit. Routledge Academic; 2009.

See Also

`fitdistr`, `fitCDF`, `mixtdistr`, and `mcgoftest`.

Examples

```
set.seed(123) # set a seed for random generation
## ========= A mixture of three distributions =========
phi = c(3/10, 7/10) #' Mixture proportions
## ---------------------------------------------------------

## === Named vector of the corresponding distribution function parameters
## must be provided
args <- list(gamma = c(shape = 2, scale = 0.1), weibull = c(shape = 3, scale = 0.5))
## ------------------------------------------------------------

## ===== Sampling from the specified mixture distribution ====
X <- rmixtdistr(n = 1e5, phi = phi , arg = args)
## ------------------------------------------------------------

## ===== Nonlinear fit of the specified mixture distribution ====
FIT <- fitMixDist(X, args = list(gamma = c(shape = NA, scale = NA),
                                 weibull = c(shape = NA, scale = NA)),
                  npoints = 200, usepoints = 1000)

## === The graphics for the simulated dataset and the corresponding theoretical
## mixture distribution
par(bg = "gray98",  mar = c(3, 4, 2, 1) )
hist(X, 90, freq = FALSE, las = 1, family = "serif", col = rgb(0, 0, 1, 0.),
     border = "deepskyblue", main = "Histogram of Mixture Distribution")
x1 <- seq(-4, 10, by = 0.001)
lines(x1, dmixtdistr(x1, phi = phi, arg = args), col = "red")
lines(x1, dmixtdistr(x1, phi = FIT$phi, arg = FIT$args), col = "blue")
legend(1, 1.5, legend=c("Theoretical Mixture PDF", "Estimated Mixture PDF"),
       col=c("red", "blue"), lty=1, cex=0.8)
```

`ggamma` *Generalized Gamma distribution*

Description

Probability density function (PDF), cummulative density function (CDF), quantile function and random generation for the Generalized Gamma (GG) distribution with 3 or 4 parameters: alpha, scale, mu, and psi. The function is reduced to GGamma distribution with 3 parameters by setting mu = 0.

Usage

```
dggamma(q, alpha = 1, scale = 1, mu = 0, psi = 1, log.p = FALSE)

pggamma(q, alpha = 1, scale = 1, mu = 0, psi = 1,
  lower.tail = TRUE, log.p = FALSE)

qggamma(p, alpha = 1, scale = 1, mu = 0, psi = 1,
  lower.tail = TRUE, log.p = FALSE)

rggamma(n, alpha = 1, scale = 1, mu = 0, psi = 1)
```

Arguments

`q`	numeric vector
`alpha`	numerical parameter, strictly positive (default 1). The generalized gamma becomes the gamma distribution for alpha = 1.
`scale, psi`	the same real positive parameters as is used for the Gamma distribution. These are numerical and strictly positives; default 1. (see ?pgamma).
`mu`	location parameter (numerical, default 0).
`log.p`	logical; if TRUE, probabilities/densities p are returned as log(p).
`lower.tail`	logical; if TRUE (default), probabilities are P[X<=x], otherwise, P[X > x]
`n`	number of observations

Details

Details about these function can be found in references 1 to 3. You may also see section Note at ?pgamma or ?rgamma. Herein, we are using Stacy' s formula (references 2 to 3) with the parametrization given in reference 4 (equation 6, page 12). As in the case of gamma distribution function, the cumulative distribution function (as given in equation 12, page 13 from reference 4) is expressed in terms of the lower incomplete gamma function (see ?pgamma).

The GG distribution with parameters α, β (scale), ψ, and μ has density:

$$f(x|\alpha,\beta,\mu,\psi) = \alpha exp(-((x-\mu)/\beta)^{\alpha})((x-\mu)/\beta)^(\alpha*\psi-1)/(\beta Gamma(\psi))$$

Value

GG PDF values (3-parameters or 4-parameters) for dggamma, GG probability for pggamma, quantiles or GG random generated values for rggamma.

Author(s)

Robersy Sanchez (`https://genomaths.com`).

References

1. Handbook on STATISTICAL DISTRIBUTIONS for experimentalists (p. 73) by Christian Walck. Particle Physics Group Fysikum. University of Stockholm (e-mail: walck@physto.se)

2. Stacy, E. W. A Generalization of the Gamma Distribution. Ann. Math. Stat. 33, 1187–1192 (1962).

3. Stacy E, Mihram G (1965) Parameter estimation for a generalized gamma distribution. Technometrics 7: 349-358.

4. Sanchez, R. & Mackenzie, S. A. Information Thermodynamics of Cytosine DNA Methylation. PLoS One 11, e0150427 (2016).

Examples

```
q <- (1:9)/10
pggamma(q, alpha = 1, scale = 1, mu = 0,
       psi = 1, lower.tail = TRUE, log.p = FALSE)

## To fit random generated numbers
set.seed(126)
x <- rggamma(1000, alpha = 1.03, psi = 0.75, scale = 2.1)
fitGGammaDist(x)
```

---

hnorm *Half-Normal distribution*

Description

Probability density function (PDF), cummulative density function (CDF), quantile function and random generation for the Half-normal (hnorm) distribution.

Usage

```
dhnorm(x, theta = 1, log = FALSE)

phnorm(q, theta = 1, lower.tail = TRUE, log.p = FALSE)

qhnorm(p, theta = 1, sigma = NULL, lower.tail = TRUE,
  log.p = FALSE)

rhnorm(n, theta = 1)

theta2sigma(theta)

sigma2theta(sigma)
```

Arguments

`theta`	numerical parameter, strictly positive (default 1).
`q`	numeric vector
`lower.tail`	logical; if TRUE (default), probabilities are P[X<=x], otherwise, P[X > x]
`log.p`	logical; if TRUE, probabilities/densities p are returned as log(p).
`sigma`	Standart deviation of the normal distribution. Here, $\sigma = \sqrt{\pi}/(\theta\sqrt{2})$
`n`	number of observations

Details

An alternative parametrization to avoid issues when sigma is near zero is applied by using a scaled precision (inverse of the variance) obtained by setting $\theta = sqrt(\pi)/\sigma * sqrt(2)$. Details about these functions can be found in Wikipedia and in MathWorld. Notice that $\theta = 1$ means $\sigma = \sqrt{\pi}/\sqrt{2}$.

Value

Half-normal PDF values (theta parameter) for dhnorm, Half-normal probability for phnorm, quantiles or Half-normal random generated values for rhnorm.

Author(s)

Robersy Sanchez (`https://genomaths.com`).

Examples

```
set.seed(123) # set a seed
sigma = 1.2
theta = sigma2theta(sigma)
x <- rhnorm(n = 1e5, theta = theta)
hist(x, 100, freq = FALSE)
curve(dhnorm(x, theta = theta), col = "red", add = TRUE)

#' # Checking the function outputs for the logarithms of probabilities
x <- rhalfnorm(n = 10, theta = sigma2theta(2))
x1 <- phnorm(x, theta = sigma2theta(2), log = TRUE)
x2 <- phnorm(x, theta = sigma2theta(2), log = FALSE)
all(round(x1, 8) == round(log(x2), 8))

x3 <- dhnorm(x, theta = sigma2theta(2), log = TRUE)
x4 <- dhnorm(x, theta = sigma2theta(2), log = FALSE)
all(round(x3, 8) == round(log(x4), 8))
```

mcgoftest	*Bootstrap test for Goodness of fit (GoF)*

Description

To accomplish the nonlinear fit of a probability distribution function (*PDF*), dIfferent optimization algorithms can be used. Each algorithm will return a different set of estimated parameter values. AIC and BIC are not useful (in this case) to decide which parameter set of values is the best. The goodness-of-fit tests (GOF) can help in this case.

Usage

```
mcgoftest(varobj, distr, pars, num.sampl = 999, sample.size,
  stat = c("ks", "ad", "rmst", "chisq"), breaks = NULL,
  parametric = TRUE, seed = 1, num.cores = 1, tasks = 0)
```

Arguments

`varobj`	A a vector containing observations, the variable for which the CDF parameters was estimated.
`distr`	The name of the cummulative distribution function (CDF) or a concrete CDF from where estimate the cummulative probabilities. Distribution *distr* must be defined in environment-namespace from any package or environment defined by user.
`pars`	CDF model parameters. A list of parameters to evaluate the CDF.
`num.sampl`	Number of resamplings.
`sample.size`	Size of the samples used for each sampling.
`stat`	One string denoting the statistic to used in the testing: "ks": Kolmogorov–Smirnov, "ad": Anderson–Darling statistic, "chisq: Pearson's Chi-squared, and "rmst": Root Mean Square statistic.
`breaks`	Default is NULL. Basically, the it is same as in function `hist`. If *breaks* = NULL, then function 'nclass.FD' (see `nclass` is applied to estimate the breaks.
`parametric`	Logical object. If TRUE, then samples are drawn from the theoretical population described by *distr*. Default: TRUE.
`seed`	An integer used to set a 'seed' for random number generation.
`num.cores, tasks`	Paramaters for parallele computation using package `BiocParallel-package`: the number of cores to use, i.e. at most how many child processes will be run simultaneously (see `bplapply` and the number of tasks per job (only for Linux OS).

Details

The test is intended for continuos distributions. If sampling size is lesser the size of the sample, then the test becomes a Monte Carlo test. The thes is based on the use of measures of goodness of fit, statistics. The following statistics are availible:

- Kolmogorov- Smirnov statistic (ks). Limitations: sensitive to ties [1]. Only the parametric Monte Carlo resampling (provided that there is not ties in the data) can be used.
- Anderson–Darling statistic (ad) [2]. Limitation: by construction, it depends on the sample size. So, the size of the sampling must be close to the sample size if Monte Carlo resampling is used, which could be a limitation if the sample size is too large [2]. In particular, could be an issue in some genomic applications. It is worth highlighting that, for the current application, the Anderson–Darling statistic is not standardized as typically done in testing GoF for normal distribution with Anderson–Darling test. It is not required since, the statistic is not compared with a corresponding theoretical value. In addition, since the computation of this statistic requires for the data to be put in order [2], it does not make sense to perform a permutation test. That is, the maximum sampling size is the sample size less 1.
- Pearson's Chi-squared statistic (chisq). Limitation: the sample must be discretized (partitioned into bins), which is could be a source of bias that leads to the rejection of the null hypothesis. Here, the discretization is done using function the resources from function `hist`.
- Root Mean Square statistic (rmst). Limitation: the same as 'chisq'.

Value

A numeric vector with the following data:

1. Statistic value.
2. mc_p.value: the probability of finding the observed, or more extreme, results when the null hypothesis H_0 of a study question is true obtained Monte Carlo resampling approach.

Author(s)

Robersy Sanchez (`https://genomaths.com`).

References

1. Feller, W. On the Kolmogorov-Smirnov Limit Theorems for Empirical Distributions. Ann. Math. Stat. 19, 177–189 (1948).
2. Anderson, T. . & Darling, D. A. A Test Of Goodness Of Fit. J. Am. Stat. Assoc. 49, 765–769 (1954).
3. Watson, G. S. On Chi-Square Goodness-Of-Fit Tests for Continuous Distributions. J. R. Stat. Soc. Ser. B Stat. Methodol. 20, 44–72 (1958).

See Also

Distribution fitting: `fitMixDist`, `fitdistr`, `fitCDF`.

Examples

```
# Example 1
# Let us generate a random sample a from a specified Weibull distribution:
# Set a seed
set.seed( 1 )
# Random sample from Weibull( x | shape = 0.5, scale = 1.2 )
x = rweibull(10000, shape = 0.5, scale = 1.2)

# MC KS test accept the null hypothesis that variable x comes
# from Weibull(x | shape = 0.5, scale = 1.2), while the standard
# Kolmogorov-Smirnov test reject the Null Hypothesis.
mcgoftest(x, distr = pweibull, pars = c( 0.5, 1.2 ), num.sampl = 500,
        sample.size = 1000, num.cores = 4)

# Example 2
# Let us generate a random sample a random sample from a specified Normal
# distribution:
# Set a seed
set.seed( 1 )
x = rnorm(10000, mean = 1.5, sd = 2)

# MC KS test accept the null hypothesis that variable x comes
# from N(x | mean = 0.5, sd = 1.2), while the standard
# Kolmogorov-Smirnov test reject the Null Hypothesis.
mcgoftest(x, distr = pnorm, pars = c(1.5, 2), num.sampl = 500,
          sample.size = 1000, num.cores = 1)
```

---

mixtdistr *Mixture of Distribution Functions*

Description

Density, distribution function, quantile function and random generation for mixture of distributions

Usage

```
dmixtdistr(x, phi, arg, log = FALSE, lower.tail = TRUE,
  log.p = FALSE)

pmixtdistr(q, phi, arg, lower.tail = TRUE, log.p = FALSE)

qmixtdistr(p, interval = c(0, 1000), phi, arg, lower.tail = TRUE,
  log.p = FALSE)

rmixtdistr(n, phi, arg)
```

Arguments

`x,`	q vector of quantiles.
`phi`	Numerical vector with mixture proportions, where $sum(phi) = 1$.
`arg`	A list of named vectors with the corresponding named distribution parameters values. The names of the vector of parameters and the parameter names must correspond to defined functions. For example, if one of the involved distributions is the gamma density (see `GammaDist`), then the corresponding vector of parameters must be gamma = c(shape = 'some value', scale = 'some value'). See examples for more details.
`log, log.p`	logical; if TRUE, probabilities p are given as log(p).
`lower.tail`	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.
`p`	vector of probabilities.
`n`	number of observations. If length(n) > 1, the length is taken to be the number required.

Author(s)

Robersy Sanchez (`https://genomaths.com`).

See Also

`fitMixDist`, `mcgoftest` (for goodness-of fit), and for additional examples: `https://genomaths.com/stats/sampling-from-a-mixture-of-distributions/`

Examples

```
set.seed(123) # set a seed for random generation
# A mixture of three distributions
phi = c(5/10, 3/10, 2/10) # Mixture proportions

# Named vector of the corresponding distribution function parameters
# must be provided
args <- list(gamma = c(shape = 20, scale = 1/10),
            weibull = c(shape =  4, scale = 0.8),
            lnorm = c(meanlog = 1.2, sdlog = 0.08))

#  Sampling from the specified mixture distribution
x <- rmixtdistr(n = 1e5, phi = phi , arg = args)

# The graphics for the simulated dataset and the corresponding theoretical
# mixture distribution
hist(x, 100, freq = FALSE)
x1 <- seq(0, 10, by = 0.001)
lines(x1, dmixtdistr(x1, phi = phi, arg = args), col = "red")
```

ppCplot *P-P plot of Two-dimensional Copulas*

Description

The function build the P-P plot of Two-dimensional Copulas upon the knowledge of the margin distribution provided by the user. The empirical probabilities are computed using function `empCopula` from package `[copula-package]{copula}`.

Usage

```
ppCplot(X, Y, copula = NULL, margins = NULL, paramMargins = NULL,
  npoints = 100, method = "ml", smoothing = c("none", "beta",
  "checkerboard"), ties.method = "max",
  xlab = "Empirical probabilities", ylab = "Theoretical probabilities",
  glwd = 1.2, bgcol = "grey94", gcol = "white", dcol = "red",
  dlwd = 0.8, tck = NA, tcl = -0.3, xlwd = 0.8, ylwd = 0.8,
  xcol = "black", ycol = "black", cex.xtitle = 1.3,
  cex.ytitle = 1.3, padj = -1, hadj = 0.7, xcex = 1.3,
  ycex = 1.3, xline = 1.6, yline = 2.1, xfont = 3, yfont = 3,
  family = "serif", lty = 1, bty = "n", col = "black",
  xlim = c(0, 1), ylim = c(0, 1), pch = 20, las = 1, mar = c(4,
  4, 2, 1), font = 3, cex = 1, seed = 132, ...)
```

Arguments

`X`	Numerical vector with the observations from the first margin distribution.
`Y`	Numerical vector with the observations from the second margin distribution.
`copula`	A copula object from class `Mvdc` or string specifying all the name for a copula from package `copula-package`.
`margins`	A character vector specifying all the parametric marginal distributions. See details below.
`paramMargins`	A list whose each component is a list (or numeric vectors) of named components, giving the parameter values of the marginal distributions. See details below.
`npoints`	Number of points used to build the P-P plot. The
`method`	A character string specifying the estimation method to be used to estimate the dependence parameter(s); see `fitCopula`.
`smoothing`	character string specifying whether the empirical distribution function (for F.n()) or copula (for C.n()) is computed (if smoothing = "none"), or whether the empirical beta copula (smoothing = "beta") or the empirical checkerboard copula (smoothing = "checkerboard") is computed (see `empCopula`.
`ties.method`	character string specifying how ranks should be computed if there are ties in any of the coordinate samples of x; passed to `pobs` (see `empCopula`.

`xlab`	A label for the x axis, defaults to a description of x.
`ylab`	A label for the y axis, defaults to a description of y.
`glwd`	Grid line width.
`bgcol`	Grid background color.
`gcol`	Grid line color
`dcol`	Diagonal line color.
`dlwd`	Diagonal line color.
`tck`	The length of tick marks as a fraction of the smaller of the width or height of the plotting region. If tck >= 0.5 it is interpreted as a fraction of the relevant side, so if tck = 1 grid lines are drawn. The default setting (tck = NA) is to use tcl = -0.5.
`tcl`	The length of tick marks as a fraction of the height of a line of text. The default value is -0.5; setting tcl = NA sets tck = -0.01 which is S' default.
`xlwd`	X-axis line width.
`ylwd`	Y-axis line width.
`xcol`	X-axis line color.
`ycol`	Y-axis line color.
`cex.xtitle`	Cex for x-axis title.
`cex.ytitle`	Cex for y-axis title.
`padj`	adjustment for each tick label perpendicular to the reading direction. For labels parallel to the axes, padj = 0 means right or top alignment, and padj = 1 means left or bottom alignment. This can be a vector given a value for each string, and will be recycled as necessary.
`hadj`	adjustment (see par("adj")) for all labels parallel ('horizontal') to the reading direction. If this is not a finite value, the default is used (centring for strings parallel to the axis, justification of the end nearest the axis otherwise).
`xcex, ycex`	A numerical value giving the amount by which axis labels should be magnified relative to the default.
`xline, yline`	On which margin line of the plot the x & y labels must be placed, starting at 0 counting outwards (see `mtext`).
`xfont, yfont`	An integer which specifies which font to use for x & y axes titles (see `par`).
`family, lty, bty, col, xlim, ylim, pch, las, mar, font`	Graphical parameters (see `par`).
`cex`	A numerical value giving the amount by which plotting text and symbols should be magnified relative to the default. This starts as 1 when a device is opened, and is reset when the layout is changed, e.g. by setting mfrow.
`seed`	An integer used to set a 'seed' for random number generation.
`...`	Other graphical parameters to pass to functions: `abline`, `mtext` and `axis`.

Details

Empirical and theoretical probabilities are estimated using the quantiles generated with the margin quantile functions. Nonlinear fit of margin distributions can be previously accomplished using any of the functions `fitCDF`, `fitdistr`, or function `fitMixDist` for the case where the margins are mixture of distributions. *npoints* random uniform and iid numbers from the interval [0, 1] are generated and used to evaluate the quantile margin distribution functions. Next, the quantiles are used to compute the empirical and theoretical copulas, which will be used to estimate the corresponding probabilities.

Value

The P-P plot and invisible temporary object with the information to build the graphic which can be assigned to a variable to use in further plots or analyses.

Author(s)

Robersy Sanchez (`https://genomaths.com`).

See Also

`fitCDF`, `fitdistr`, `fitMixDist`, and `bicopulaGOF`.

Examples

```
set.seed(12)
margins = c("norm", "norm")
## Random variates from normal distributions
X <- rlnorm(200, meanlog =- 0.5, sdlog = 3.1)
Y <- rnorm(200, mean = 0, sd = 6)
cor(X,Y) ## Correlation between X and Y

parMargins = list( list(meanlog = 0.5, sdlog = 3.1),
                   list(mean = 0, sd = 10))

copula = "normalCopula"
npoints = 100

## The information to build the graphic is stored in object 'g'.
g <- ppCplot(X = X, Y = Y, copula = "normalCopula", margins = margins,
             paramMargins = parMargins, npoints = 20)
```

Index

abline, *21*
AICmodel, 2
axis, *21*

BICmodel, 3
bicopulaGOF, 4, *22*
bplapply, *5*, *16*

density, *10*, *11*
dggamma *(*ggamma*)*, 12
dhnorm *(*hnorm*)*, 14
dmixtdistr *(*mixtdistr*)*, 18

emControl, *10*
empCopula, *20*

fitCDF, *6*, 7, *12*, *17*, *22*
fitCopula, *5*, *20*
fitDistr, *8*
fitdistr, *6*, *9*, *12*, *17*, *22*
fitMixDist, *6*, *9*, 10, *17*, *19*, *22*

GammaDist, *11*, *19*
ggamma, 12
gofCopula, *5*, *6*
gofTstat, *5*

hist, *16*, *17*
hnorm, 14

kmeans, *11*

mcgoftest, *9*, *12*, 16, *19*
Mclust, *10*, *11*
mixtdistr, *11*, *12*, 18
mtext, *21*
Mvdc, *4*, *5*, *20*

nclass, *16*
nls.lm, *8*, *11*
nls.lm.control, *8*

par, *21*
pggamma *(*ggamma*)*, 12
phnorm *(*hnorm*)*, 14
pmixtdistr *(*mixtdistr*)*, 18
pobs, *20*
ppCplot, *6*, 20

qggamma *(*ggamma*)*, 12
qhnorm *(*hnorm*)*, 14
qmixtdistr *(*mixtdistr*)*, 18

rggamma *(*ggamma*)*, 12
rhnorm *(*hnorm*)*, 14
rmixtdistr *(*mixtdistr*)*, 18

sigma2theta *(*hnorm*)*, 14

theta2sigma *(*hnorm*)*, 14