

The fBasics Package

December 14, 2006

Version 240.10068.1

Date 1996 - 2006

Title Rmetrics - Markets and Basic Statistics

Author Diethelm Wuertz and many others, see the SOURCE file

Depends R (>= 1.9.0), methods, MASS, fEcofin, fCalendar

Maintainer Diethelm Wuertz and Rmetrics Core Team <wuertz@itp.phys.ethz.ch>

Description Environment for teaching “Financial Engineering and Computational Finance”

License GPL Version 2 or later

URL <http://www.rmetrics.org>

R topics documented:

fBasicsUtilities	2
TailoredReturnPlots	4
StableDistribution	5
HyperbolicDistribution	7
DistributionFits	10
BasicStatistics	14
StylizedFacts	17
PortableInnovations	21
HypothesisTesting	24
OneSampleTests	26
TwoSampleTests	31

Index	36
--------------	-----------

Description

A collection and description of functions which are useful utilities and tools used by Rmetrics.

The plot utility functions are:

<code>characterTable</code>	Table of Numerical Equivalents to Latin Characters,
<code>symbolTable</code>	Table of plot characters, plot symbols,
<code>colorTable</code>	Table of Color Codes and Plot Colors itself,
<code>greyPalette</code>	Creates a grey palette like rainbow does for colors.

Usage

```
characterTable(font = 1, cex = 0.7)
symbolTable(font = par('font'), cex = 0.7)
colorTable(cex = 0.7)
greyPalette(n = 64, start = 255-n, end = 255)
```

Arguments

`font, cex` [**Table*] -
an integer value, the number of the font, by default font number 1, the standard font for the `characterTable` or the current plot character font for the `plotcharacterTable`. The character size is determined by the numeric value `cex`, the default size is 0.7.

`n, start, end`
[*greyPal*] - `cr n` gives the number of greys to be constructed, `start` and `end` span the range of the color palette. By default 64 grey tones equidistant chosen from the color range (191, 191, 191) to (255, 255, 255).

Value

`characterTable`
displays a table with the characters of the requested font. The character on line "xy" and column "z" of the table has code "xyz", e.g `cat("\126")` prints: V for font number 1. These codes can be used as any other characters.

`symbolTable`
displays a table with the plot characters numbered from 0 to 255.

`colorTable`
displays a table with the plot colors with the associated color number.

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

Examples

```
## SOURCE("fBasics.1A-fBasicsUtilities")
```

```
TailoredReturnPlots
      Tailored Return Plots
```

Description

A collection and description of functions which allow to create easily financial return plots.

The tailored plot functions are:

seriesPlot	Returns a tailored time series plot,
histPlot	Returns a tailored histogram plot,
densityPlot	Returns a tailored kernel density estimate plot,
quantilePlot	Returns a tailored quantile-quantile plot.

Usage

```
seriesPlot(x, col = "steelblue", main = x@units, ...)
histPlot(x, col = "steelblue", main = x@units, add.fit = TRUE, ...)
densityPlot(x, col = "steelblue", main = x@units, add.fit = TRUE, ...)
quantilePlot(x, col = "steelblue", main = x@units, labels = TRUE, ...)
```

Arguments

add.fit	[*Plot] - a logical, should a fit added to the Plot?
col, main	[*Plot] - plot parameters, color and main title.
labels	a logical, should labels be added to the plot?
x	an object of class "timeSeries".
...	optional arguments to be passed.

Value

Beside the plot, no other values are returned.

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

Examples

```
## SOURCE("fBasics.1B-TailoredReturnPlots")
```

StableDistribution *Stable Distribution Function*

Description

A collection and description of functions to compute density, distribution function, quantile function and to generate random variates, the stable distribution, and the stable mode. Two different cases are considered, the first for the symmetric and the second for the skewed distribution.

The functions are:

<code>[dpqr] symstb</code>	The symmetric stable distribution,
<code>[dpqr] stable</code>	the skewed stable distribution,
<code>stableMode</code>	the stable mode,
<code>symstbSlider</code>	interactive symmetric distribution display,
<code>stableSlider</code>	interactive stable distribution display.

Usage

```
dsymstb(x, alpha)
psymstb(q, alpha)
qsymstb(p, alpha)
rsymstb(n, alpha)
```

```
dstable(x, alpha, beta, gamma = 1, delta = 0, pm = c(0, 1, 2))
pstable(q, alpha, beta, gamma = 1, delta = 0, pm = c(0, 1, 2))
qstable(p, alpha, beta, gamma = 1, delta = 0, pm = c(0, 1, 2))
rstable(n, alpha, beta, gamma = 1, delta = 0, pm = c(0, 1, 2))
```

```
stableMode(alpha, beta)
```

```
symstbSlider()
stableSlider()
```

Arguments

<code>alpha, beta, gamma, delta</code>	value of the index parameter <code>alpha</code> with <code>alpha = (0, 2]</code> ; skewness parameter <code>beta</code> , in the range <code>[-1, 1]</code> ; scale parameter <code>gamma</code> ; and shift parameter <code>delta</code> .
<code>n</code>	number of observations, an integer value.
<code>p</code>	a numeric vector of probabilities.
<code>pm</code>	parameterization, an integer value by default <code>pm=0</code> , the 'S0' parameterization.

x , q a numeric vector of quantiles.

Details

Symmetric Stable Distribution:

For the density and probability the approach of McCulloch is implemented. Note, that McCulloch's approach has a density precision of 0.000066 and a distribution precision of 0.000022 for `alpha` in the range [0.84, 2.00]. Quantiles are evaluated from a root finding process via the probability function. Thus, this leads to nonnegligible errors for small quantiles, since the quantile evaluation depends on the quality of the probability function. To achieve higher precisions use the function `stable` with argument `beta=0`.

For generation of random deviates the results of Chambers, Mallows, and Stuck are used.

Skew Stable Distribution:

The function uses the approach of J.P. Nolan for general stable distributions. Nolan derived expressions in form of integrals based on the characteristic function for standardized stable random variables. These integrals are numerically evaluated using R's function `integrate`.

"S0" parameterization [`pm=0`]: based on the (M) representation of Zolotarev for an alpha stable distribution with skewness `beta`. Unlike the Zolotarev (M) parameterization, `gamma` and `delta` are straightforward scale and shift parameters. This representation is continuous in all 4 parameters, and gives an intuitive meaning to `gamma` and `delta` that is lacking in other parameterizations.

"S" or "S1" parameterization [`pm=1`]: the parameterization used by Samorodnitsky and Taqqu in the book *Stable Non-Gaussian Random Processes*. It is a slight modification of Zolotarev's (A) parameterization.

"S*" or "S2" parameterization [`pm=2`]: a modification of the S0 parameterization which is defined so that (i) the scale `gamma` agrees with the Gaussian scale (standard dev.) when `alpha=2` and the Cauchy scale when `alpha=1`, (ii) the mode is exactly at `delta`.

"S3" parameterization [`pm=3`]: an internal parameterization. The scale is the same as the S2 parameterization, the shift is $-\beta * g(\alpha)$, where $g(\alpha)$ is defined in Nolan [1999].

Value

All values for the `*symstb` and `*stable` functions are numeric vectors: `d*` returns the density, `p*` returns the distribution function, `q*` returns the quantile function, and `r*` generates random deviates.

The function `stableMode` returns a numeric value, the location of the stable mode.

The functions `symstbSlider` and `stableSlider` display for educational purposes the densities and probabilities of the symmetric and skew stable distributions.

Author(s)

McCulloch for the 'symstb' Fortran program, and Diethelm Wuertz for the Rmetrics R-port.

References

Chambers J.M., Mallows, C.L. and Stuck, B.W. (1976); *A Method for Simulating Stable Random Variables*, J. Amer. Statist. Assoc. 71, 340–344.

Nolan J.P. (1999); *Stable Distributions*, Preprint, University Washington DC, 30 pages.

Nolan J.P. (1999); *Numerical Calculation of Stable Densities and Distribution Functions*, Preprint, University Washington DC, 16 pages.

Samoridnitsky G., Taqqu M.S. (1994); *Stable Non-Gaussian Random Processes, Stochastic Models with Infinite Variance*, Chapman and Hall, New York, 632 pages.

Weron, A., Weron R. (1999); *Computer Simulation of Levy alpha-Stable Variables and Processes*, Preprint Technical Univeristy of Wroclaw, 13 pages.

Examples

```
## SOURCE("fBasics.2A-StableDistribution")

## Plot:
  par(ask = FALSE)

## stable -
  # Plot rvs Series
  set.seed(1953)
  r = rstable(n = 1000, alpha = 1.9, beta = 0.3)
  plot(r, type = "l", main = "stable: alpha=1.9 beta=0.3",
       col = "steelblue")
  grid()

## stable -
  # Plot empirical density and compare with true density:
  hist(r, n = 25, probability = TRUE, border = "white",
       col = "steelblue")
  x = seq(-5, 5, 0.4)
  lines(x, dstable(x = x, alpha = 1.9, beta = 0.3))

## stable -
  # Plot df and compare with true df:
  plot(sort(r), (1:1000/1000), main = "Probability", pch = 19,
       col = "steelblue")
  lines(x, pstable(q = x, alpha = 1.9, beta = 0.3))
  grid()

## stable -
  # Compute quantiles:
  qstable(pstable(seq(-4, 4, 1), alpha = 1.9, beta = 0.3),
         alpha = 1.9, beta = 0.3)
```

HyperbolicDistribution
Generalized Hyperbolic Distribution

Description

A collection and description of functions to compute density, distribution function, quantile function and to generate random variates for three cases of the generalized hyperbolic distribution: the generalized hyperbolic distribution itself, the hyperbolic distribution and the normal inverse Gaussian distribution.

The functions are:

<code>[dpqr]gh</code>	The generalized hyperbolic distribution,
<code>[dpqr]hyp</code>	The hyperbolic distribution,
<code>hypMode</code>	the hyperbolic mode,
<code>[dpqr]nig</code>	The normal inverse Gaussian distribution,
<code>hypSlider</code>	interactive hyperbolic distribution display,
<code>nigSlider</code>	interactive NIG distribution display.

Usage

```
dgh(x, alpha = 1, beta = 0, delta = 1, mu = 0, lambda = 1)
pgh(q, alpha = 1, beta = 0, delta = 1, mu = 0, lambda = 1)
qgh(p, alpha = 1, beta = 0, delta = 1, mu = 0, lambda = 1)
rgh(n, alpha = 1, beta = 0, delta = 1, mu = 0, lambda = 1)

dhyp(x, alpha = 1, beta = 0, delta = 1, mu = 0, pm = c(1, 2, 3, 4))
phyp(q, alpha = 1, beta = 0, delta = 1, mu = 0, pm = c(1, 2, 3, 4), ...)
qhyp(p, alpha = 1, beta = 0, delta = 1, mu = 0, pm = c(1, 2, 3, 4), ...)
rhyp(n, alpha = 1, beta = 0, delta = 1, mu = 0, pm = c(1, 2, 3, 4))

hypMode(alpha = 1, beta = 0, delta = 1, mu = 0, pm = c(1, 2, 3, 4))

dnig(x, alpha = 1, beta = 0, delta = 1, mu = 0)
pnig(q, alpha = 1, beta = 0, delta = 1, mu = 0)
qnig(p, alpha = 1, beta = 0, delta = 1, mu = 0)
rnig(n, alpha = 1, beta = 0, delta = 1, mu = 0)

hypSlider()
nigSlider()
```

Arguments

`alpha`, `beta`, `delta`, `mu`, `lambda`
 shape parameter `alpha`; skewness parameter `beta`, `abs(beta)` is in the range $(0, \alpha)$; scale parameter `delta`, `delta` must be zero or positive; lo-

cation parameter `mu`, by default 0; and lambda parameter `lambda`, by default 1. These is the meaning of the parameters in the first parameterization `pm=1` which is the default parameterization selection. In the second parameterization, `pm=2` `alpha` and `beta` take the meaning of the shape parameters (usually named) `zeta` and `rho`. In the third parameterization, `pm=3` `alpha` and `beta` take the meaning of the shape parameters (usually named) `xi` and `chi`. In the fourth parameterization, `pm=4` `alpha` and `beta` take the meaning of the shape parameters (usually named) `a.bar` and `b.bar`.

<code>n</code>	number of observations.
<code>p</code>	a numeric vector of probabilities.
<code>pm</code>	an integer value between 1 and 4 for the selection of the parameterization. The default takes the first parameterization.
<code>x, q</code>	a numeric vector of quantiles.
<code>...</code>	arguments to be passed to the function <code>integrate</code> .

Details

Generalized Hyperbolic Distribution:

The generator `rgh` is based on the GH algorithm given by Scott (2004).

Hyperbolic Distribution:

The generator `rhyp` is based on the HYP algorithm given by Atkinson (1982).

Normal Inverse Gaussian Distribution:

The random deviates are calculated with the method described by Raible (2000).

Value

All values for the `*gh`, `*hyp`, and `*nig` functions are numeric vectors: `d*` returns the density, `p*` returns the distribution function, `q*` returns the quantile function, and `r*` generates random deviates.

All values have attributes named "param" listing the values of the distributional parameters.

The function `hyp*Mode` returns the mode in the appropriate parameterization. A numeric value.

The functions `hypSlider` and `nigSlider` display for educational purposes the densities and probabilities of the hyperbolic and normal inverse Gaussian distributions.

Note

An undocumented R function for the modified Bessel function K_1 named `.BesselK1(x)` is available, which is called by the S-Plus version of the program.

Author(s)

David Scott for the HYP Generator from R's "HyperbolicDist" package,
Diethelm Wuertz for the Rmetrics R-port.

References

Atkinson, A.C. (1982); *The simulation of generalized inverse Gaussian and hyperbolic random variables*, SIAM J. Sci. Stat. Comput. 3, 502–515.

Barndorff-Nielsen O. (1977); *Exponentially decreasing distributions for the logarithm of particle size*, Proc. Roy. Soc. Lond., A353, 401–419.

Barndorff-Nielsen O., Blaesild, P. (1983); *Hyperbolic distributions*. In *Encyclopedia of Statistical Sciences*, Eds., Johnson N.L., Kotz S. and Read C.B., Vol. 3, pp. 700–707. New York: Wiley.

Raible S. (2000); *Levy Processes in Finance: Theory, Numerics and Empirical Facts*, PhD Thesis, University of Freiburg, Germany, 161 pages.

Examples

```
## SOURCE("fBasics.2B-HyperbolicDistribution")

## Plot:
par(ask = FALSE)

## nig -
set.seed(1953)
r = rnig(5000, alpha = 1, beta = 0.3, delta = 1)
plot(r, type = "l", col = "steelblue",
     main = "nig: alpha=1 beta=0.3 delta=1")

## nig -
# Plot empirical density and compare with true density:
hist(r, n = 25, probability = TRUE, border = "white", col = "steelblue4")
x = seq(-5, 5, 0.25)
lines(x, dnig(x, alpha = 1, beta = 0.3, delta = 1))

## nig -
# Plot df and compare with true df:
plot(sort(r), (1:5000/5000), main = "Probability", col = "steelblue4")
lines(x, pnig(x, alpha = 1, beta = 0.3, delta = 1))

## nig -
# Compute Quantiles:
qnig(pnig(seq(-5, 5, 1), alpha = 1, beta = 0.3, delta = 1),
     alpha = 1, beta = 0.3, delta = 1)
```

Description

A collection and description of moment and maximum likelihood estimators to fit the parameters of a distribution. Included are estimators for the Student-t, for the stable, for the generalized hyperbolic hyperbolic, for the normal inverse Gaussian, and for empirical distributions.

The functions are:

nFit	MLE parameter fit for a normal distribution,
tFit	MLE parameter fit for a Student t-distribution,
stableFit	MLE and Quantile Method stable parameter fit,
ghFit	MLE parameter fit for a generalized hyperbolic distribution,
hypFit	MLE parameter fit for a hyperbolic distribution,
nigFit	MLE parameter fit for a normal inverse Gaussian distribution.

Usage

```
nFit(x, doplot = TRUE, span = "auto", title = NULL, description = NULL, ...)
tFit(x, df = 4, doplot = TRUE, span = "auto", trace = FALSE, title = NULL,
     description = NULL, ...)
```

```
stableFit(x, alpha = 1.75, beta = 0, gamma = 1, delta = 0,
          type = c("q", "mle"), doplot = TRUE, trace = FALSE, title = NULL,
          description = NULL)
```

```
ghFit(x, alpha = 1, beta = 0, delta = 1, mu = 0, lambda = 1, doplot = TRUE,
      span = "auto", trace = FALSE, title = NULL, description = NULL, ...)
```

```
hypFit(x, alpha = 1, beta = 0, delta = 1, mu = 0, doplot = TRUE,
       span = "auto", trace = FALSE, title = NULL, description = NULL, ...)
```

```
nigFit(x, alpha = 1, beta = 0, delta = 1, mu = 0, doplot = TRUE,
       span = "auto", trace = FALSE, title = NULL, description = NULL, ...)
```

```
show.fDISTFIT(object)
```

Arguments

alpha, beta, gamma, delta, mu, lambda

[stable] -

The parameters are alpha, beta, gamma, and delta:
value of the index parameter alpha with $\alpha = (0, 2]$; skewness parameter beta, in the range $[-1, 1]$; scale parameter gamma; and shift parameter delta.

[hyp] -

The parameters are alpha, beta, delta, mu, and lambda:

shape parameter `alpha`; skewness parameter `beta`, `abs(beta)` is in the range $(0, \alpha)$; scale parameter `delta`, `delta` must be zero or positive; location parameter `mu`, by default 0; and lambda parameter `lambda`, by default 1. These is the meaning of the parameters in the first parameterization `pm=1` which is the default parameterization selection. In the second parameterization, `pm=2` `alpha` and `beta` take the meaning of the shape parameters (usually named) `zeta` and `rho`. In the third parameterization, `pm=3` `alpha` and `beta` take the meaning of the shape parameters (usually named) `xi` and `chi`. In the fourth parameterization, `pm=4` `alpha` and `beta` take the meaning of the shape parameters (usually named) `a.bar` and `b.bar`.

<code>description</code>	a character string which allows for a brief description.
<code>df</code>	[tFit] - the number of degrees of freedom for the Student distribution, <code>df > 2</code> , maybe non-integer. By default a value of 4 is assumed.
<code>object</code>	[show] - an S4 class object as returned from the fitting functions.
<code>doplot</code>	[tFit][hypFit][nigFit] - a logical flag. Should a plot be displayed?
<code>span</code>	x-coordinates for the plot, by default 100 values automatically selected and ranging between the 0.001, and 0.999 quantiles. Alternatively, you can specify the range by an expression like <code>span=seq(min, max, times = n)</code> , where, <code>min</code> and <code>max</code> are the left and right endpoints of the range, and <code>n</code> gives the number of the intermediate points.
<code>title</code>	a character string which allows for a project title.
<code>trace</code>	[*Fit] - a logical flag. Should the parameter estimation process be traced?
<code>type</code>	a character string which allows to select the method for parameter estimation: "mle", the maximum log likelihood approach, or "qm", McCulloch's quantile method.
<code>x</code>	[*Fit] - a numeric vector.
<code>...</code>	parameters to be parsed.

Details

Maximum Likelihood Estimation:

The function `nlm` is used to minimize the "negative" maximum log-likelihood function. `nlm` carries out a minimization using a Newton-type algorithm.

Value

The functions `tFit`, `hypFit` and `nigFit` return a list with the following components:

`estimate` the point at which the maximum value of the log likelihood function is obtained.

minimum	the value of the estimated maximum, i.e. the value of the log likelihood function.
code	an integer indicating why the optimization process terminated. 1: relative gradient is close to zero, current iterate is probably solution; 2: successive iterates within tolerance, current iterate is probably solution; 3: last global step failed to locate a point lower than <code>estimate</code> . Either <code>estimate</code> is an approximate local minimum of the function or <code>steptol</code> is too small; 4: iteration limit exceeded; 5: maximum step size <code>stepmax</code> exceeded five consecutive times. Either the function is unbounded below, becomes asymptotic to a finite value from above in some direction or <code>stepmax</code> is too small.
gradient	the gradient at the estimated maximum.
steps	number of function calls.

Remark: The parameter estimation for the stable distribution via the maximum Log-Likelihood approach may take a quite long time.

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

Examples

```
## SOURCE("fBasics.2D-DistributionFits")

## Plot:
par(ask = FALSE)

## nigFit -
# Simulate random variates HYP(1.5, 0.3, 0.5, -1.0):
set.seed(1953)
s = rnig(n = 1000, alpha = 1.5, beta = 0.3, delta = 0.5, mu = -1.0)

## nigFit -
# Fit Parameters:
# Note, this may take some time.
# Starting vector (1, 0, 1, mean(s)):
nigFit(s, alpha = 1, beta = 0, delta = 1, mu = mean(s), doplot = TRUE)
```

BasicStatistics *Basic Statistics Summary*

Description

A collection and description of functions to compute basic statistical properties of financial and economic time series data. Missing functions in R to calculate skewness and kurtosis are added, a function which creates a summary statistics, and functions to calculate column and row statistics.

The functions are:

skewness	returns value of skewness,
kurtosis	returns value of kurtosis,
basicStats	computes an overview of basic statistical values,
rowStats	calculates row statistics,
colStats	calculates column statistics,
rowAvg	calculates row means,
colAvg	calculates column means,
rowVars	calculates row variances,
colVars	calculates column variances,
rowStdevs	calculates row standard deviations,
colStdevs	calculates column standard deviations,
rowSkewness	calculates row skewness,
colSkewness	calculates column skewness,
rowKurtosis	calculates row kurtosis,
colKurtosis	calculates column kurtosis,
rowCumsums	calculates row cumulated Sums,
colCumsums	calculates column cumulated Sums.

For SPLUS Compatibility:

stdev Returns the standard deviation of a vector or matrix.

Usage

```

stdev(x, na.rm = FALSE)

skewness(x, ...)
## Default S3 method:
skewness(x, na.rm = FALSE, method = c("moment", "fisher"), ...)
## S3 method for class 'data.frame':
skewness(x, ...)
## S3 method for class 'POSIXct':
skewness(x, ...)
## S3 method for class 'POSIXlt':
skewness(x, ...)

kurtosis(x, ...)
## Default S3 method:
kurtosis(x, na.rm = FALSE, method = c("excess", "moment", "fisher"), ...)
## S3 method for class 'data.frame':
kurtosis(x, ...)
## S3 method for class 'POSIXct':
kurtosis(x, ...)
## S3 method for class 'POSIXlt':
kurtosis(x, ...)

basicStats(x, ci = 0.95)

```

```

rowStats(x, FUN, na.rm = FALSE, ...)
rowAves(x, na.rm = FALSE, ...)
rowVars(x, na.rm = FALSE, ...)
rowStdevs(x, na.rm = FALSE, ...)
rowSkewness(x, na.rm = FALSE, ...)
rowKurtosis(x, na.rm = FALSE, ...)
rowCumsums(x, na.rm = FALSE, ...)

```

```

colStats(x, FUN, na.rm = FALSE, ...)
colAves(x, na.rm = FALSE, ...)
colVars(x, na.rm = FALSE, ...)
colStdevs(x, na.rm = FALSE, ...)
colSkewness(x, na.rm = FALSE, ...)
colKurtosis(x, na.rm = FALSE, ...)
colCumsums(x, na.rm = FALSE, ...)

```

Arguments

<code>ci</code>	confidence interval, a numeric value, by default 0.95, i.e. 95 percent.
<code>FUN</code>	[colStats][rowStats] - the statistical function to be applied.
<code>na.rm</code>	a logical. Should missing values be removed?
<code>method</code>	[kurtosis][skewness] - a character string which specifies the method of computation. These are either "moment" or "fisher", kurtosis allows in addition for "excess". If "excess" is selected, then the value of the kurtosis is computed by the "moment" method and a value of 3 will be subtracted. The "moment" method is based on the definitions of skewness and kurtosis for distributions; these forms should be used when resampling (bootstrap or jackknife). The "fisher" method correspond to the usual "unbiased" definition of sample variance, although in the case of skewness and kurtosis exact unbiasedness is not possible.
<code>x</code>	a numeric vector, or a matrix for column statistics. [basicStats] - allows also a matrix, data.frame or timeSeries as input. In this case only the first column of data will be considered and a warning will be printed.
<code>...</code>	arguments to be passed.

Value

skewness
kurtosis
return the value of the statistics, a numeric value. An attribute which reports the used method is added.

basicsStats
returns data frame with the following entries and row names: nobs, NAs, Minimum, Maximum, 1. Quartile, 3. Quartile, Mean, Median, Sum, SE Mean, LCL Mean, UCL Mean, Variance, Stdev,

Skewness, Kurtosis.

```
rowStats
rowAvs
rowVars
rowStdevs
rowSkewness
rowKurtosis
rowCumsum
compute sample statistics by column. Missing values can be handled.
```

```
colStats
colAvs
colVars
colStdevs,
colSkewness
colKurtosis
colCumsum
compute sample statistics by column. Missing values can be handled.
```

Note

R's-base package contains a function `colMeans` with an additional argument `dim=1`. Therefore, the function used here to compute column means (averages) is named `colAvs`.

The function `stdev` computes the standard deviation for a vector or matrix and was introduced for SPlus compatibility. Under R use the function `sd`.

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

Examples

```
## SOURCE("fBasics.3A-BasicStatistics")

## basicStats -
# Simulated Monthly Return Data:
tS = timeSeries(matrix(rnorm(12)), timeCalendar())
# ... must be univariate:
basicStats(tS)

## mean -
## var -
## skewness -
## kurtosis -
# Mean, Variance:
mean(tS)
var(tS)
```

```
# Skewness, Kurtosis:
class(tS)
skewness(tS)
kurtosis(tS)
```

StylizedFacts

Stylized Facts

Description

A collection and description of functions to investigate and to plot several stylized facts of economic and financial time series. This includes fat tails, autocorrelations, crosscorrelations, long memory behavior, and the Taylor effect.

The functions to display stylized facts are:

acfPlot	autocorrelation function plot,
pacfPlot	partial autocorrelation function plot,
ccfPlot	cross correlation function plot,
lacfPlot	lagged autocorrelation function plot,
lmacfPlot	long memory autocorrelation function plot,
logpdfPlot	logarithmic density plots,
qqgaussPlot	Gaussian quantile quantile plot,
scalinglawPlot	scaling behavior plot,
teffectPlot	Taylor effect plot.

Usage

```
acfPlot(x, labels = TRUE, ...)
pacfPlot(x, labels = TRUE, ...)
ccfPlot(x, y, lag.max = max(2, floor(10*log10(length(x)))), type =
  c("correlation", "covariance", "partial"), labels = TRUE, ...)
lacfPlot(x, n = 12, lag.max = 20, labels = TRUE, ...)
lmacfPlot(x, lag.max = max(2, floor(10*log10(length(x)))), ci = 0.95,
  type = c("both", "acf", "hurst"), labels = TRUE, details = TRUE, ...)
logpdfPlot(x, n = 50, type = c("lin-log", "log-log"), doplot = TRUE,
  labels = TRUE, ...)
qqgaussPlot(x, span = 5, col = "steelblue", labels = TRUE, ...)
scalinglawPlot(x, span = ceiling(log(length(x)/252)/log(2)), doplot = TRUE,
  labels = TRUE, details = TRUE, ...)
teffectPlot(x, deltas = seq(from = 0.2, to = 3, by = 0.2), lag.max = 10,
  ymax = NA, standardize = TRUE, labels = TRUE, ...)
```

Arguments

ci [lmacfPlot] -
the confidence interval, by default 95 percent, i.e. 0.95.

<code>col</code>	a character string denoting the plot color, by default "steelblue".
<code>deltas</code>	[teffectPlot] - the exponents, a numeric vector, by default ranging from 0.2 to 3.0 in steps of 0.2.
<code>doplot</code>	a logical value. Should a plot be displayed?
<code>details</code>	a logical value. Should the results be printed?
<code>labels</code>	a logical value. Whether or not x- and y-axes should be automatically labeled and a default main title should be added to the plot. By default TRUE.
<code>lag.max</code>	maximum lag for which the autocorrelation should be calculated, an integer.
<code>n</code>	[lacfPlot] - an integer value, the number of lags. [logpdfPlot] - an integer value, the number of break and count points.
<code>span</code>	[scalinglawPlot] - an integer value, determines for the <code>qqgaussPlot</code> the plot range, by default 5, and for the <code>scalingPlot</code> a reasonable number of of points for the scaling range, by default daily data with 252 business days per year are assumed.
<code>standardize</code>	[teffectPlot] - a logical value. Should the vector x be standardized?
<code>type</code>	[ccfPlot] - a character string, either "correlation", "covariance", or "partial" denoting which type of correlation should be plotted, [lmacf] - a character string, either "both", "acf", or "hurst" denoting which type of plot should be generated: an autocorrelation plot, a hurst plot, or both, [logpdf] - a character string, either "lin-log", or "log-log" denoting on which scale the plot will produced: on a linear vs. log scale, or on a double logarithmic scale.
<code>x, y</code>	numeric vectors, univariate time series objects, or any other object which can be transformed by the function <code>as.vector</code> into a numeric vector.
<code>ymax</code>	[teffectPlot] - maximum y-axis value on plot, <code>is.na(ymax)</code> TRUE, then the value is selected automatically.
<code>...</code>	arguments to be passed.

Details

Tail Behavior:

`logpdfPlot` and `qqgaussPlot` are two simple functions which allow a quick view on the tails of a distribution. The first creates a logarithmic or double-logarithmic density plot and returns breaks and counts. For the double logarithmic plot, the negative side of the distribution is reflected onto the positive axis.

The second creates a Gaussian Quantile-Quantile plot.

Scaling Behavior:

The function `scalingPlot` plots the scaling law of financial time series under aggregation and returns an estimate for the scaling exponent. The scaling behavior is a very striking effect of the foreign exchange market and also other markets expressing a regular structure for the volatility. Considering the average absolute return over individual data periods one finds a scaling power law which relates the mean volatility over given time intervals to the size of these intervals. The power law is in many cases valid over several orders of magnitude in time. Its exponent usually deviates significantly from a Gaussian random walk model which implies $1/2$.

Autocorrelation Functions:

The functions `acfPlot`, `pacfPlot`, and `ccfPlot` plots and estimate autocorrelation, ACF, partial autocorrelation, PACF, and cross-covariance and cross-correlation functions, CCF. The functions allow to get a first view on correlations in and between time series. The functions are synonme function calls for R's `acf`, `pacf`, and `ccf` from the the `ts` package.

Long Memory Autocorrelation Function:

The function `lmacfPlot` plots and estimates the long memory autocorrelation function and computes from the plot the Hurst exponent of a time series. The volatility of financial time series exhibits (in contrast to the logarithmic returns) in almost every financial market a slow ecaying autocorrelation function, ACF. We talk of a long memory if the decay in the ACF is slower than exponential, i.e. the correlation function decreases algebraically with increasing (integer) lag. Thus it makes sense to investigate the decay on a double-logarithmic scale and to estimate the decay exponent. The function `lmacf` calculates and plots the autocorrelation function of the vector x . If the time series exhibits long memory behaviour, it can easily be seen as a stright line in the plot. This double-logarithmic plot is displayed and a linear regression fit is done from which the intercept and slope ar calculated. From the slope the Hurst exponent is derived.

Taylor Effect:

The "Taylor Effect" describes the fact that absolute returns of speculative assets have significant serial correlation over long lags. Even more, autocorrelations of absolute returns are typically greater than those of squared returns. From these observations the Taylor effect states, that that the autocorrelations of absolute returns to the the power of δ , $abs(x - \text{mean}(x))^{\delta}$ reach their maximum at $\delta=1$. The function `teffect` explores this behaviour. A plot is created which shows for each lag (from 1 to `max.lag`) the autocorrelations as a function of the exponent δ . In the case that the above formulated hypothesis is supported, all the curves should peak at the same value around $\delta=1$.

Value

`logpdfPlot`

returns a list with the following components: `breaks`, histogram mid-point breaks; `counts`, histogram counts; `fbreaks`, fitted Gaussian breaks; `fcounts`, fitted Gaussian counts.

qqgaussPlot
returns a Gaussian Quantile-Quantile Plot.

scalingPlot
returns a list with the following components: `exponent`, the scaling exponent, a numeric value;
`fit`, a list with the coefficients returned by `lsfit`, i.e. `intercept` and `X`.

`acfPlot`, `pacfplot`, `ccfPlot`
return an object of class "acf", see [acf](#).

`lmacfPlot`
returns a list with the following elements: `fit`, a list by itself with elements `Intercept` and `slope X`, `hurst`, the Hurst exponent, both are numeric values.

`lacfPlot` returns a list with the following two elements: `Rho`, the autocorrelation function,
`lagged`, the lagged correlations.

`teffectPlot`
returns a numeric matrix of order `deltas` by `max.lag` with the values of the autocorrelations.

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

References

Taylor S.J. (1986); *Modeling Financial Time Series*, John Wiley and Sons, Chichester.

Ding Z., Granger C.W.J., Engle R.F. (1993); *A long memory property of stock market returns and a new model*, Journal of Empirical Finance 1, 83.

Examples

```
## SOURCE("fBasics.3B-StylizedFacts")
par(ask = FALSE)

## data -
# require(MASS)
plot(SP500, type = "l", col = "steelblue", main = "SP500")
abline(h = 0, col = "grey")

## qqgaussPlot -
# Lagged Correlations:
qqgaussPlot(SP500)

## teffectPlot -
# Taylor Effect:
teffectPlot(SP500)
```

 PortableInnovations

Generator for Portable Random Innovations

Description

A collection and description of functions to generate portable random innovations. The functions run under R and S-Plus and generate the same sequence of random numbers. Supported are uniform, normal and Student-t distributed random numbers.

The functions are:

<code>set.lcgseed</code>	Set initial random seed,
<code>get.lcgseed</code>	Get the current value of the random seed,
<code>runif.lcg</code>	Uniform linear congruational generator,
<code>rnorm.lcg</code>	Normal linear congruational generator,
<code>rt.lcg</code>	Student-t linear congruational generator.

Usage

```
set.lcgseed(seed = 4711)
get.lcgseed()
runif.lcg(n, min = 0, max = 1)
rnorm.lcg(n, mean = 0, sd = 1)
rt.lcg(n, df)
```

Arguments

<code>df</code>	number of degrees of freedom, a positive integer, maybe non-integer.
<code>mean, sd</code>	means and standard deviation of the normal distributed innovations.
<code>min, max</code>	lower and upper limits of the uniform distributed innovations.
<code>seed</code>	an integer value, the random number seed.
<code>n</code>	an integer, the number of random innovations to be generated.

Details

A simple portable random number generator for use in R and SPlus. We recommend to use this generator only for comparisons of calculations in R and Splus.

The generator is a linear congruational generator with parameters LCG ($a=13445$, $c=0$, $m=2^{31}-1$, $X=0$). It is a simple random number generator which passes the bitwise randomness test.

Value

A vector of generated random innovations. The value of the current seed is stored in the variable `lcg.seed`.

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

References

Altman, N.S. (1988); *Bitwise Behavior of Random Number Generators*, SIAM J. Sci. Stat. Comput., 9(5), September, 941–949.

Examples

```
## SOURCE("fSeries.4A-PortableRandomInnovations")

## set.lcgseed -
  set.lcgseed(seed = 65890)

## runif.lcg - rnorm.lcg - rt.lcg -
  cbind(runif.lcg(10), rnorm.lcg(10), rt.lcg(10, df = 4))

## get.lcgseed -
  get.lcgseed()

## Note, to overwrite rnorm, use
  # rnorm = rnorm.lcg
  # Going back to rnorm
  # rm(rnorm)
```

HypothesisTesting *Tests Class Representation and Utilities*

Description

Class representation, methods and utility functions for objects of class 'fHTEST'.

The class representation and methods are:

fHTEST	Representation for an S4 object of class "fHTEST",
show	S4 print method.

Utility Functions:

pPlot	General finite sample probability plot,
pTable	interpolated probabilities from finite sample table,
qTable	interpolated quantiles from finite sample table.

Usage

```
show.fHTEST(object)
```

```
pPlot(X, nN = 100, nStat = 100, logN = TRUE, logStat = FALSE,
      fill = FALSE, linear = TRUE, digits = 8, doplot = TRUE, ...)
pTable(X, Stat, N, digits = 4)
qTable(X, p, N, digits = 4)
```

Arguments

<code>digits</code>	an integer value with the number of rounding digits.
<code>doplot</code>	a logical flag. Should a plot be displayed?
<code>fill</code>	[pPlot] - a logical flag deciding if missing data should be filled with asymptotic values zero and one.
<code>linear</code>	[pPlot] - a logical flag indicating the type of interpolation.
<code>logN, logStat</code>	[pPlot] - two logical flags deciding if the data should be on a logarithmic scale or not.
<code>N</code>	an integer value or vector of sample sizes.
<code>nN, nStat</code>	[pPlot] - two integer values with the size of the table.
<code>object</code>	[show] - an S4 object of class "fHTEST".
<code>p</code>	a numeric value or vector of probabilities.
<code>Stat</code>	a numeric value or vector of quantiles or statistic values.
<code>X</code>	[pPlot][*Table] - a data frame or matrix of a finite sample test table.
<code>...</code>	[pPlot] - additional arguments to be passed.

Value

In contrast to R's output report from S3 objects of class "htest" a different output report is produced. The tests return an S4 object of class "fHTEST". The object contains the following slots:

<code>@call</code>	the function call.
<code>@data</code>	the data as specified by the input argument(s).
<code>@test</code>	a list whose elements contain the results from the statistical test. The information provided is similar to a list object of class "htest".
<code>@title</code>	a character string with the name of the test. This can be overwritten specifying a user defined input argument.
<code>@description</code>	a character string with an optional user defined description. By default just the current date when the test was applied will be returned.

statistic	the value(s) of the test statistic.
p.value	the p-value(s) of the test.
parameters	a numeric value or vector of parameters.
estimate	a numeric value or vector of sample estimates.
conf.int	a numeric two row vector or matrix of 95
method	a character string indicating what type of test was performed.
data.name	a character string giving the name(s) of the data.

The functions `pPlot`, `pTable`, and `qTable` plot and interpolate finite sample test statistic data from a table. The table is a data frame or a matrix where columns denote the size and rows the probabilities. The column and row names must hold the sizes and probabilities as character strings. The values of the matrix hold the statistic values.

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

Examples

```
## SOURCE("fBasics.5A-HypothesisTesting")

## fHTEST -
  getClass("fHTEST")

## pPlot -
## jbTable -
# Interpolated plot of Small Jarque Bera Table:
X = jbTable(type = "LM", size = "small")
par(ask = FALSE)
pPlot(X, linear = TRUE, logStat = TRUE)
pPlot(X, linear = TRUE, logStat = TRUE, fill = TRUE, main = "JB LM")
pPlot(X, linear = FALSE, logStat = TRUE)
pPlot(X, linear = FALSE, logStat = TRUE, fill = TRUE)

## [pq]Table -
## jbTable -
# Jarque Bera B q and p Table:
X = jbTable(type = "LM", size = "small")
p = (1:99)/100
plot(qTable(X, p, N = 100), p, type = "b")
Stat = seq(0.01, 15, length = 100)
plot(Stat, pTable(X, Stat, N = 100), type = "b")
```

OneSampleTests *One Sample Tests*

Description

A collection and description of functions of one sample tests for testing normality of financial return series.

The functions for testing normality are:

<code>ksnormTest</code>	Kolmogorov-Smirnov normality test,
<code>shapiroTest</code>	Shapiro-Wilk's test for normality,
<code>jarqueberaTest</code>	Jarque-Bera test for normality,
<code>dagoTest</code>	D'Agostino normality test.

Functions for high precision Jarque Bera LM and ALM tests:

<code>jbTable</code>	Table of finite sample p values for the JB test,
<code>pjb</code>	Computes probabilities for the Jarque Bera Test,
<code>qjb</code>	Computes quantiles for the Jarque Bera Test,
<code>jbTest</code>	Performs finite sample adjusted JB LM and ALM test.

Additional functions for testing normality from the 'nortest' package:

<code>adTest</code>	Anderson-Darling normality test,
<code>cvmTest</code>	Cramer-von Mises normality test,
<code>lillieTest</code>	Lilliefors (Kolmogorov-Smirnov) normality test,
<code>pchiTest</code>	Pearson chi-square normality test,
<code>sfTest</code>	Shapiro-Francia normality test.

For SPlus/Finmetrics Compatibility:

<code>normalTest</code>	test suite for some normality tests.
-------------------------	--------------------------------------

More tests ...

<code>runsTest</code>	Runs test for detecting non-randomness.
-----------------------	---

Usage

```
ksnormTest(x, title = NULL, description = NULL)
shapiroTest(x, title = NULL, description = NULL)
jarqueberaTest(x, title = NULL, description = NULL)
dagoTest(x, title = NULL, description = NULL)
```

```

jbtTable(type = c("LM", "ALM"), size = c("all", "small"))
pjb(q, N = Inf, type = c("LM", "ALM"))
qjb(p, N = Inf, type = c("LM", "ALM"))
jbtTest(x, title = NULL, description = NULL)

adTest(x, title = NULL, description = NULL)
cvmTest(x, title = NULL, description = NULL)
lillieTest(x, title = NULL, description = NULL)
pchiTest(x, title = NULL, description = NULL)
sfTest(x, title = NULL, description = NULL)

normalTest(x, method = c("sw", "jb"), na.rm = FALSE)

runsTest(x)

```

Arguments

description	optional description string, or a vector of character strings.
method	[normalTest] - indicates four different methods for the normality test, "ks" for the Kolmogorov-Smirnov one-sample test, "sw" for the Shapiro-Wilk test, "jb" for the Jarque-Bera Test, and "da" for the D'Agostino Test. The default value is "ks".
N	an integer value specifying the sample size.
na.rm	[normalTest] - a logical value. Should missing values removed before computing the tests? The default value is FALSE.
p	a numeric vector of probabilities. Missing values are not allowed.
q	vector of quantiles or test statistics. Missing values are not allowed.
size	[jbtTable] - a character string denoting the size of the table. If set to "all" then all data are used from the table, if set to "small" then only a small part of the data will be returned.
title	an optional title string, if not specified the inputs data name is deparshed.
type	[jbtTest][pjb][qjb] - the same for the Jarque Bera test statistic. "LM" denotes the Lagrange multiplier test, and "ALM" the adjusted Lagrange multiplier test.
x	a numeric vector of data values or a S4 object of class <code>timeSeries</code> .

Details

The hypothesis tests may be of interest for many financial and economic applications, especially for the investigation of univariate time series returns.

Normal Tests:

Several tests for testing if the records from a data set are normally distributed are available. The

input to all these functions may be just a vector `x` or a univariate time series object `x` of class `timeSeries`.

First there exists a wrapper function which allows to call one from two normal tests either the Shapiro–Wilks test or the Jarque–Bera test. This wrapper was introduced for compatibility with S-Plus' FinMetrics package.

Also available are the Kolmogorov–Smirnov one sample test and the D'Agostino normality test.

The remaining five normal tests are the Anderson–Darling test, the Cramer–von Mises test, the Lilliefors (Kolmogorov–Smirnov) test, the Pearson chi–square test, and the Shapiro–Francia test. They are calling functions from R's contributed package `nortest`. The difference to the original test functions implemented in R and from contributed R packages is that the Rmetrics functions accept time series objects as input and give a more detailed output report.

The Anderson-Darling test is used to test if a sample of data came from a population with a specific distribution, here the normal distribution. The `adTest` goodness-of-fit test can be considered as a modification of the Kolmogorov–Smirnov test which gives more weight to the tails than does the `ksnormTest`.

Runs Test:

The runs test can be used to decide if a data set is from a random process. A run is defined as a series of increasing values or a series of decreasing values. The number of increasing, or decreasing, values is the length of the run. In a random data set, the probability that the $(i+1)$ -th value is larger or smaller than the i -th value follows a binomial distribution, which forms the basis of the runs test.

Value

In contrast to R's output report from S3 objects of class `"htest"` a different output report is produced. The tests here return an S4 object of class `"fHTEST"`. The object contains the following slots:

<code>@call</code>	the function call.
<code>@data</code>	the data as specified by the input argument(s).
<code>@test</code>	a list whose elements contain the results from the statistical test. The information provided is similar to a list object of class <code>"htest"</code> .
<code>@title</code>	a character string with the name of the test. This can be overwritten specifying a user defined input argument.
<code>@description</code>	a character string with an optional user defined description. By default just the current date when the test was applied will be returned.
<code>statistic</code>	the value(s) of the test statistic.
<code>p.value</code>	the p-value(s) of the test.
<code>parameters</code>	a numeric value or vector of parameters.
<code>estimate</code>	a numeric value or vector of sample estimates.
<code>conf.int</code>	a numeric two row vector or matrix of 95
<code>method</code>	a character string indicating what type of test was performed.

`data.name` a character string giving the name(s) of the data.

The meaning of the elements of the `@test` slot is the following:

`ksnormTest`

returns the values for the 'D' statistic and p-values for the three alternatives 'two-sided', 'less' and 'greater'.

`shapiroTest`

returns the values for the 'W' statistic and the p-value.

`jarqueberaTest`

`jbTest`

returns the values for the 'Chi-squared' statistic with 2 degrees of freedom, and the asymptotic p-value. `jbTest` is the finite sample version of the Jarque Bera Lagrange multiplier, LM, and adjusted Lagrange multiplier test, ALM.

`dagoTest`

returns the values for the 'Chi-squared', the 'Z3' (Skewness) and 'Z4' (Kurtosis) statistic together with the corresponding p values.

`adTest`

returns the value for the 'A' statistic and the p-value.

`cvmTest`

returns the value for the 'W' statistic and the p-value.

`lillieTest`

returns the value for the 'D' statistic and the p-value.

`pchiTest`

returns the value for the 'P' statistic and the p-values for the adjusted and not adjusted test cases. In addition the number of classes is printed, taking the default value due to Moore (1986) computed from the expression `n.classes = ceiling(2 * (n^(2/5)))`, where `n` is the number of observations.

`sfTest`

returns the value for the 'W' statistic and the p-value.

Note

Some of the test implementations are selected from R's `ctest` and `nortest` packages.

Author(s)

R-core team for the tests from R's `ctest` package,
 Adrian Trapletti for the runs test from R's `tseries` package,
 Juergen Gross for the normal tests from R's `nortest` package,
 James Filliben for the Fortran program producing the runs report,
 Diethelm Wuertz and Helmut Katzgraber for the finite sample JB tests,
 Diethelm Wuertz for the Rmetrics R-port.

References

- Anderson T.W., Darling D.A. (1954); *A Test of Goodness of Fit*, JASA 49:765–69.
- Conover, W. J. (1971); *Practical nonparametric statistics*, New York: John Wiley & Sons.
- D’Agostino R.B., Pearson E.S. (1973); *Tests for Departure from Normality*, Biometrika 60, 613–22.
- D’Agostino R.B., Rosman B. (1974); *The Power of Geary’s Test of Normality*, Biometrika 61, 181–84.
- Durbin J. (1961); *Some Methods of Constructing Exact Tests*, Biometrika 48, 41–55.
- Durbin, J. (1973); *Distribution Theory Based on the Sample Distribution Function*, SIAM, Philadelphia.
- Geary R.C. (1947); *Testing for Normality*; Biometrika 36, 68–97.
- Lehmann E.L. (1986); *Testing Statistical Hypotheses*, John Wiley and Sons, New York.
- Linnet K. (1988); *Testing Normality of Transformed Data*, Applied Statistics 32, 180–186.
- Moore, D.S. (1986); *Tests of the chi-squared type*, In: D’Agostino, R.B. and Stephens, M.A., eds., *Goodness-of-Fit Techniques*, Marcel Dekker, New York.
- Shapiro S.S., Francia R.S. (1972); *An Approximate Analysis of Variance Test for Normality*, JASA 67, 215–216.
- Shapiro S.S., Wilk M.B., Chen V. (1968); *A Comparative Study of Various Tests for Normality*, JASA 63, 1343–72.
- Thode H.C. (2002); *Testing for Normality*, Marcel Dekker, New York.
- Weiss M.S. (1978); *Modification of the Kolmogorov-Smirnov Statistic for Use with Correlated Data*, JASA 73, 872–75.
- Wuertz D., Katzgraber H.G. (2005); *Precise finite-sample quantiles of the Jarque-Bera adjusted Lagrange multiplier test*, ETHZ Preprint.

Examples

```
## SOURCE ("fBasics.5B-OneSampleTests")

## Series:
x = rnorm(100)

## ksnormTests -
# Kolmogorov - Smirnov One-Sampel Test
ksnormTest(x)

## shapiroTest - Shapiro-Wilk Test
shapiroTest(x)

## jarqueberaTest -
## jbTest - Jarque-Bera Test
jarqueberaTest(x)
jbTest(x)

## runsTest -
runsTest(x)
```

TwoSampleTests *Two Sample Tests*

Description

A collection and description of functions for two sample statistical tests. The functions allow an easy use to test financial return series for distributional equivalence, for difference in location, variance and scale, and for correlations and association.

Distributional Equivalence:

`ks2Test` Two sample Kolmogorov–Smirnov test.

Test Difference in Locations:

`locationTest` The location test suite,
`method="t"` the t test,
`method="kw"` the Kruskal–Wallis test.

Test Difference in Variance:

`varianceTest` The variance test suite,
`method="varf"` the variance F test,
`method="bartlett"` the Bartlett test,
`method="fligner"` the Fligner–Killeen test.

Test Difference in Scale:

`scaleTest` The scale test suite,
`method=ansari` the Ansari–Bradley test,
`method=mood` the Mood test.

Test for Correlations:

`correlationTest` The correlation test suite,
`method=pearson` Pearson's coefficient,
`method=kendall` Kendall's tau,
`method=spearman` Spearman's rho.

Usage

```
ks2Test(x, y, title = NULL, description = NULL)
```

```
locationTest(x, y, method = c("t", "kw2"),  
            title = NULL, description = NULL)
```

```

varianceTest(x, y, method = c("varf", "bartlett", "fligner"),
  title = NULL, description = NULL)
scaleTest(x, y, method = c("ansari", "mood"),
  title = NULL, description = NULL)

correlationTest(x, y, method = c("pearson", "kendall", "spearman"),
  title = NULL, description = NULL)

```

Arguments

description optional description string, or a vector of character strings.
method a character string naming which test should be applied.
title an optional title string, if not specified the inputs data name is deparsed.
x, y numeric vectors of data values.

Details

The tests may be of interest for many financial and economic applications, especially for the comparison of two time series. The tests are grouped according to their functionalities.

Distributional Equivalence:

The test `ks2Test` performs a Kolmogorov–Smirnov two sample test that the two data samples x and y come from the same distribution, not necessarily a normal distribution. That means that it is not specified what that common distribution is.

Differences in Location:

The `tTest` can be used to determine if the two sample means are equal for unpaired data sets. Two variants are used, assuming equal or unequal variances.

The `kw2Test` performs a Kruskal-Wallis rank sum test of the null hypothesis that the central tendencies or medians of two samples are the same. The alternative is that they differ. Note, that it is not assumed that the two samples are drawn from the same distribution. It is also worth to know that the test assumes that the variables under consideration have underlying continuous distributions.

Differences in Variances:

The `varfTest` can be used to compare variances of two normal samples performing an F test. The null hypothesis is that the ratio of the variances of the populations from which they were drawn is equal to one.

The `bartlett2Test` performs the Bartlett's test of the null hypothesis that the variances in each of the samples are the same. This fact of equal variances across samples is also called *homogeneity of variances*. Note, that Bartlett's test is sensitive to departures from normality. That is, if the samples come from non-normal distributions, then Bartlett's test may simply be testing for non-normality. The Levene test (not yet implemented) is an alternative to the Bartlett test that is less sensitive to departures from normality.

The `fligner2Test` performs the Fligner-Killeen test of the null that the variances in each of the two samples are the same.

Differences in Scale:

The `ansariTest` performs the Ansari-Bradley two-sample test for a difference in scale parameters. Note, that we have completely reimplemented this test based on the statistics and p-values computed from algorithm AS 93. The test returns for any sizes of the series x and y the exact p value together with its asymptotic limit. The test procedure is not limited to sizes shorter of length 50 as this is the case for the function `ansari.Test` implemented in R's `stats` package.

The `codemoodTest`, is another test which performs a two-sample test for a difference in scale parameters. The underlying model is that the two samples are drawn from $f(x-l)$ and $f((x-l)/s)/s$, respectively, where l is a common location parameter and s is a scale parameter. The null hypothesis is $s=l$.

Correlations:

The `correlationTest` for association between paired samples, allows to compute Pearson's product moment correlation coefficient, Kendall's tau, or Spearman's rho.

Value

In contrast to R's output report from S3 objects of class `"htest"` a different output report is produced. The classical tests presented here return an S4 object of class `"fHTEST"`. The object contains the following slots:

<code>@call</code>	the function call.
<code>@data</code>	the data as specified by the input argument(s).
<code>@test</code>	a list whose elements contain the results from the statistical test. The information provided is similar to a list object of class <code>"htest"</code> .
<code>@title</code>	a character string with the name of the test. This can be overwritten specifying a user defined input argument.
<code>@description</code>	a character string with an optional user defined description. By default just the current date when the test was applied will be returned.
<code>statistic</code>	the value(s) of the test statistic.
<code>p.value</code>	the p-value(s) of the test.
<code>parameters</code>	a numeric value or vector of parameters.
<code>estimate</code>	a numeric value or vector of sample estimates.
<code>conf.int</code>	a numeric two row vector or matrix of 95
<code>method</code>	a character string indicating what type of test was performed.
<code>data.name</code>	a character string giving the name(s) of the data.

Note

Some of the test implementations are selected from R's `ctest` package.

Author(s)

R-core team for the tests from R's ctest package,
Diethelm Wuertz for the Rmetrics R-port.

References

- Conover, W. J. (1971); *Practical nonparametric statistics*, New York: John Wiley & Sons.
 Durbin J. (1961); *Some Methods of Constructing Exact Tests*, *Biometrika* 48, 41–55.
 Durbin, J. (1973); *Distribution Theory Based on the Sample Distribution Function*, SIAM, Philadelphia.
 Lehmann E.L. (1986); *Testing Statistical Hypotheses*, John Wiley and Sons, New York.
 Moore, D.S. (1986); *Tests of the chi-squared type*, In: D'Agostino, R.B. and Stephens, M.A., eds., *Goodness-of-Fit Techniques*, Marcel Dekker, New York.

Examples

```
## SOURCE("fBasics.5C-TwoSampleTests")

## x, y -
  x = rnorm(50)
  y = rnorm(50)

## ks2Test -
  ks2Test(x, y)

## locationTest | .tTest | .kw2Test -
  locationTest(x, y)

## varianceTest | .varfTest, .bartlett2Test | .fligner2Test -
  varianceTest(x, y)

## scaleTest | .ansariTest | .moodTest -
  scaleTest(x, y)

## correlationTest | .pearsonTest | .kendallTest | .spearmanTest -
  correlationTest(x, y)
```

Index

- *Topic **distribution**
 - DistributionFits, 9
 - HyperbolicDistribution, 6
 - StableDistribution, 4
- *Topic **hplot**
 - StylizedFacts, 16
- *Topic **htest**
 - HypothesisTesting, 23
 - OneSampleTests, 25
 - TwoSampleTests, 30
- *Topic **programming**
 - fBasicsUtilities, 1
 - PortableInnovations, 20
 - TailoredReturnPlots, 3
- *Topic **univar**
 - BasicStatistics, 13
- acf, 19
- acfPlot (*StylizedFacts*), 16
- adTest (*OneSampleTests*), 25
- BasicStatistics, 13
- basicStats (*BasicStatistics*), 13
- ccfPlot (*StylizedFacts*), 16
- characterTable
 - (*fBasicsUtilities*), 1
- colAvg (*BasicStatistics*), 13
- colCumsums (*BasicStatistics*), 13
- colKurtosis (*BasicStatistics*), 13
- colorTable (*fBasicsUtilities*), 1
- colSkewness (*BasicStatistics*), 13
- colStats (*BasicStatistics*), 13
- colStdevs (*BasicStatistics*), 13
- colVars (*BasicStatistics*), 13
- correlationTest (*TwoSampleTests*), 30
- cvmTest (*OneSampleTests*), 25
- dagoTest (*OneSampleTests*), 25
- densityPlot
 - (*TailoredReturnPlots*), 3
- dgh (*HyperbolicDistribution*), 6
- dhyp (*HyperbolicDistribution*), 6
- DistributionFits, 9
- dnig (*HyperbolicDistribution*), 6
- dstable (*StableDistribution*), 4
- dsymstb (*StableDistribution*), 4
- fBasicsUtilities, 1
- fDISTFIT (*DistributionFits*), 9
- fDISTFIT-class
 - (*DistributionFits*), 9
- fHTEST (*HypothesisTesting*), 23
- fHTEST-class (*HypothesisTesting*), 23
- get.lcgseed
 - (*PortableInnovations*), 20
- ghFit (*DistributionFits*), 9
- greyPalette (*fBasicsUtilities*), 1
- histPlot (*TailoredReturnPlots*), 3
- HyperbolicDistribution, 6
- hypFit (*DistributionFits*), 9
- hypMode (*HyperbolicDistribution*), 6
- HypothesisTesting, 23
- hypSlider
 - (*HyperbolicDistribution*), 6
- jarqueberaTest (*OneSampleTests*), 25
- jbTable (*OneSampleTests*), 25
- jbTest (*OneSampleTests*), 25
- ks2Test (*TwoSampleTests*), 30
- ksnormTest (*OneSampleTests*), 25
- kurtosis (*BasicStatistics*), 13
- lacfPlot (*StylizedFacts*), 16

- lillieTest (*OneSampleTests*), 25
- lmacfPlot (*StylizedFacts*), 16
- locationTest (*TwoSampleTests*), 30
- logpdfPlot (*StylizedFacts*), 16
- nFit (*DistributionFits*), 9
- nigFit (*DistributionFits*), 9
- nigSlider
 - (*HyperbolicDistribution*), 6
- nlm, 12
- normalTest (*OneSampleTests*), 25
- OneSampleTests, 25
- pacfPlot (*StylizedFacts*), 16
- pchiTest (*OneSampleTests*), 25
- pgh (*HyperbolicDistribution*), 6
- phyp (*HyperbolicDistribution*), 6
- pjb (*OneSampleTests*), 25
- pnig (*HyperbolicDistribution*), 6
- PortableInnovations, 20
- pPlot (*HypothesisTesting*), 23
- print.fDISTFIT
 - (*DistributionFits*), 9
- pstable (*StableDistribution*), 4
- psymstb (*StableDistribution*), 4
- pTable (*HypothesisTesting*), 23
- qgh (*HyperbolicDistribution*), 6
- qhyp (*HyperbolicDistribution*), 6
- qjb (*OneSampleTests*), 25
- qnig (*HyperbolicDistribution*), 6
- qqgaussPlot (*StylizedFacts*), 16
- qstable (*StableDistribution*), 4
- qsymstb (*StableDistribution*), 4
- qTable (*HypothesisTesting*), 23
- quantilePlot
 - (*TailoredReturnPlots*), 3
- rgH (*HyperbolicDistribution*), 6
- rhyp (*HyperbolicDistribution*), 6
- rnig (*HyperbolicDistribution*), 6
- rnorm.lcg (*PortableInnovations*), 20
- rowAverages (*BasicStatistics*), 13
- rowCumsums (*BasicStatistics*), 13
- rowKurtosis (*BasicStatistics*), 13
- rowSkewness (*BasicStatistics*), 13
- rowStats (*BasicStatistics*), 13
- rowStdevs (*BasicStatistics*), 13
- rowVars (*BasicStatistics*), 13
- rstable (*StableDistribution*), 4
- rsymstb (*StableDistribution*), 4
- rt.lcg (*PortableInnovations*), 20
- runif.lcg (*PortableInnovations*), 20
- runsTest (*OneSampleTests*), 25
- scaleTest (*TwoSampleTests*), 30
- scalinglawPlot (*StylizedFacts*), 16
- seriesPlot (*TailoredReturnPlots*), 3
- set.lcgseed
 - (*PortableInnovations*), 20
- sfTest (*OneSampleTests*), 25
- shapiroTest (*OneSampleTests*), 25
- show, fDISTFIT-method
 - (*DistributionFits*), 9
- show, fHTEST-method
 - (*HypothesisTesting*), 23
- show.fDISTFIT (*DistributionFits*), 9
- show.fHTEST (*HypothesisTesting*), 23
- skewness (*BasicStatistics*), 13
- StableDistribution, 4
- stableFit (*DistributionFits*), 9
- stableMode (*StableDistribution*), 4
- stableSlider
 - (*StableDistribution*), 4
- stdev (*BasicStatistics*), 13
- StylizedFacts, 16
- symbolTable (*fBasicsUtilities*), 1
- symstbSlider
 - (*StableDistribution*), 4
- TailoredReturnPlots, 3
- teffectPlot (*StylizedFacts*), 16
- tFit (*DistributionFits*), 9
- TwoSampleTests, 30
- varianceTest (*TwoSampleTests*), 30