

Package ‘fBasics’

September 28, 2009

Version 2100.78

Revision 4425

Date 2009-09-28

Title Rmetrics - Markets and Basic Statistics

Author Diethelm Wuertz and many others, see the SOURCE file

Depends R (>= 2.6.0), MASS, methods, timeDate, timeSeries (>= 2100.84)

Suggests akima, spatial, RUnit, tcltk

Maintainer Rmetrics Core Team <Rmetrics-core@r-project.org>

Description Environment for teaching “Financial Engineering and Computational Finance”

NOTE SEVERAL PARTS ARE STILL PRELIMINARY AND MAY BE CHANGED IN THE FUTURE. THIS TYPICALLY INCLUDES FUNCTION AND ARGUMENT NAMES, AS WELL AS DEFAULTS FOR ARGUMENTS AND RETURN VALUES.

LazyLoad yes

LazyData yes

License GPL (>= 2)

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

Repository CRAN

Date/Publication 2009-09-28 14:09:51

R topics documented:

fBasics-package	3
fUtilities-package	6
acfPlot	11
akimaInterp	13
baseMethods	15
BasicStatistics	16
BoxPlot	17
characterTable	18
colorLocator	18
colorPalette	19
colorTable	22
colVec	23
correlationTest	24
decor	25
distCheck	26
DistributionFits	27
fHTEST	29
getS4	30
gh	31
ghFit	33
ghMode	35
ghSlider	36
ght	36
ghtFit	37
gridVector	39
Heaviside	39
hilbert	41
HistogramPlot	42
hyp	43
hypFit	45
hypMode	47
hypSlider	48
Ids	48
interactivePlot	49
inv	50
krigeInterp	51
kron	52
ks2Test	53
lcg	54
linearInterp	56
listDescription	57
listFunctions	58
listIndex	58
locationTest	59
maxdd	61
nig	63

nigFit	64
nigShapeTriangle	66
nigSlider	67
norm	67
NormalityTests	68
pascal	72
pdl	73
positiveDefinite	74
print	75
QuantileQuantilePlots	75
ReturnSeriesGUI	77
rk	77
rowStats	78
scaleTest	80
ScalingLawPlot	81
sgh	83
sghFit	84
snig	85
snigFit	87
StableDistribution	88
symbolTable	90
TimeSeriesPlots	91
tr	92
triang	93
tslag	94
varianceTest	95
vec	96
Index	98

Description

The Rmetrics "fbasics" package is a collection of functions to explore and to investigate basic properties of financial returns and related quantities.

The covered fields include techniques of explorative data analysis and the investigation of distributional properties, including parameter estimation and hypothesis testing.

For *Explorative Data Analysis* several plot functions are available to explore the time series themselves, to show their distributional properties, and to encover correlations and dependencies.

Functions to compute *Distributional Properties* of financial returns and derivated and related series are enclosed. The in detail considered distributions include the normal, the student-t, the stable, the family of generalised hyperbolic, and max drawdown distributions. Moment and log-likelihood estimators allow to estimate the distributional parameters. Functions to compute moments and modes ar also avialable.

The functions from *Hypothesis Testing* one sample and two sample tests. Most of the one sample tests deal with testing normality. The two sample tests allow to compare two series to find out if the series are correlated or their distributions have the same distributional parameters.

Details

Package: fBasics
 Type: Package
 Version: 270.73
 Date: 2008
 License: GPL Version 2 or later
 Copyright: (c) 1999-2008 Diethelm Wuertz and Rmetrics Foundation
 URL: <http://www.rmetrics.org>

Overview:

The following chapters give a brief introduction how to optimize and analyze portfolios.

1. Explorative Data Analysis
2. Distributional Properties
3. Hypthesis Testing

1. Explorative Data Analysis

Explorative data analysis of financial return series and related series

Exploratory Data Analysis is an approach for data analysis that employs a variety of techniques most of graphical nature to maximize insight into a data set, to uncover underlying structures, and to detect outliers and anomalies. For this several functions are implemented to plot the series, to plot the distribution from different views, and to visualize correlations and dependencies.

Time Series Plots:

seriesPlot	Returns a tailored return series plot,
cumulatedPlot	returns a cumulated series given the returns,
returnPlot	returns returns given the cumulated series,
drawdownPlot	returns drawdowns given the return series,

Density Plots:

histPlot	Returns a tailored histogram plot,
densityPlot	returns a tailored kernel density estimate plot,
logDensityPlot	returns a tailored log kernel density estimate plot,
qqPlot	returns a quantile-quantile plot,
scalinglawPlot	returns a scaling law plot,

Box Plots:

boxPlot	Returns a side-by-side standard box plot,
boxPercentilePlot	Returns a side-by-side box-percentile plot,

CorrelationPlots:

acfPlot	autocorrelation function plot,
pacfPlot	partial autocorrelation function plot,
lacfPlot	lagged autocorrelation function plot,
teffectPlot	Taylor effect plot.

2. Distributional Properties

The distributional properties can be investigate for several types of distribution functions which are important in the investigation of financial returns and related time series.

Stable and Skew-Stable Distribution

[dpqr]stable	the stable and skew-stable distribution,
stableMode	the stable and skew-stable mode,
stableSlider	interactive stable distribution display.

*The family of Generalized Hyperbolic Distributions***GH:**

[dpqr]gh	GH, Generlized hyperbolic distribution,
ghFit	GH parameter estimation,
ghMode	mode of the GH distribution,
ghMoments	moments of the GH distribution,
ghSlider	HYP distribution Slider,

HYP:

[dpqr]hyp	HYP, hyperbolic distribution,
hypFit	HYP parameter estimation,
hypMode	mode of the hyperbolic distribution,
hypSlider	HYP distribution Slider,

NIG:

[dpqr]nig	NIG, hyperbolic distribution,
nigFit	NIG parameter estimation,
nigMode	mode of the NIG distribution,
nigSlider	NIG distribution Slider,

GHT:

[dpqr]ght	GHT, generalized hyperbolic Student-t,
ghtFit	GHT parameter estimation,
ghtMode	mode of the GHT distribution,
ghtSlider	GHT distribution Slider.

*The family of Standardized Distributions***SGH:**

[dpqr]sgh	Standardized GH distribution,
sghFit	Standardized GH parameter estimation,

SNIG:

[dpq]snig Standardized NIG distribution,
 snigFit Standardized NIG parameter estimation.

The max-Drawdown Distribution

The functions compute compute drawdown statistics. Included are density, distribution function, and random generation for the maximum drawdown distributions. In addition the expectation of drawdowns for Brownian motion can be computed.

[dpr]maxdd the max drawdown distribution,
 maxddStats the expectation of drawdowns.

3. Hypthesis Testing*One Sample Normality Tests*

normalityTests

Two Sample Tests

correlationTest

locationTest

varianceTest

scaleTest ?

Note

With Rmetrics version 2.7.0 ...

fUtilities-package *Utilities and Tools Package*

Description

Package of basic utilities and general tools for Rmetrics.

Details

Package: fUtilities
 Type: Package
 Version: 261.73.1
 Date: 2008
 License: GPL Version 2 or later
 Copyright: (c) 1999-2008 Diethelm Wuertz and Rmetrics Foundation
 URL: <http://www.rmetrics.org>

Overview of Topics:

1. Basic Function Extensions
2. Column and Row Statistics for Rectangular Objects
3. Skewness and Kurtosis Statistics
4. Bivariate Interpolation and Kriging
5. Code Tables, Color Selection and Palettes
6. Vector/Matrix Arithmetics and Linear Algebra Addons
7. Special Functions Addon

1. Basic Function Extensions

Several functions are added by Rmetrics which are missing in R's basic packages.

The first set of these functions are concerned with R functions which were made generic, so that they could be used to add additional methods:

align	adds align function,
as.POSIXlt	adds POSIXlt function,
atoms	adds atoms function,
attach	extends attach function,
colnames<-	adds colnames assignment,
cor	extends cor function,
cov	extends cov function,
log	extends log function,
outlier	adds outlier function,
rank	extends rank function,
rownames<-	adds rownames assignment,
sample	extends sample function,
stdev	adds stdev function,
termPlot	adds term plot function,
var	extends var function,
volatility	adds volatility function.

All these functions have now default methods. Furthermore the fUtilities package also provides additional methods. These include:

as.matrix.ts	adds as.matrix.ts method,
as.matrix.mts	adds as.matrix.mts method,
print.control	adds print.control method.

2. Statistical Function Extensions

Beside the basic function extensions also statistical function extensions are provided. This concerns the missing "skewness" and "kurtosis" functions in R

skewness	returns value of skewness,
kurtosis	returns value of kurtosis.

and functions for column and row statistics. The `colStats` and `rowStats` are quite general functions which allow to specify the function to compute the desired statistics by the user. The remaining column and row statistics functions are thought to compute often used time series statistics. This includes the `sum()`, mean, standard deviations, variance, skewness, kurtosis, maximum, minimum, product, and quantile value.

Column Statistics:

<code>colStats</code>	calculates column statistics,
<code>colSums</code>	calculates column sums,
<code>colMeans</code>	calculates column means,
<code>colSds</code>	calculates column standard deviations,
<code>colVars</code>	calculates column variances,
<code>colSkewness</code>	calculates column skewness,
<code>colKurtosis</code>	calculates column kurtosis,
<code>colMaxs</code>	calculates maximum values in each column,
<code>colMins</code>	calculates minimum values in each column,
<code>colProds</code>	computes product of all values in each column,
<code>colQuantiles</code>	computes quantiles of each column.

Row Statistics:

<code>rowStats</code>	calculates row statistics,
<code>rowSums</code>	calculates row sums,
<code>rowMeans</code>	calculates row means,
<code>rowSds</code>	calculates row standard deviations,
<code>rowVars</code>	calculates row variances,
<code>rowSkewness</code>	calculates row skewness,
<code>rowKurtosis</code>	calculates row kurtosis,
<code>rowMaxs</code>	calculates maximum values in each row,
<code>rowMins</code>	calculates minimum values in each row,
<code>rowProds</code>	computes product of all values in each row,
<code>rowQuantiles</code>	computes quantiles of each row.

For hypothesis testing `Rmetrics` offers a new S4 class and print method:

<code>fHTEST</code>	Representation for an S4 object of class "fHTEST",
<code>show</code>	S4 print method.

3. Graph and Plot Tools

Character, symbol and color tables are useful tools if one is concerned with graphs and charts:

<code>characterTable</code>	Table of Numerical Equivalentents to Latin Characters,
<code>symbolTable</code>	Table of plot characters, plot symbols,
<code>colorTable</code>	Table of Color Codes and Plot Colors itself,
<code>colorLocator</code>	Plots R's 657 named colors for selection,
<code>colorMatrix</code>	Returns matrix of R's color names.

Many wrapper functions to create color palettes are also added, all following the same naming conventions:

rainbowPalette	Contiguous rainbow color palette,
heatPalette	Contiguous heat color palette,
terrainPalette	Contiguous terrain color palette,
topoPalette	Contiguous topo color palette,
cmPalette	Contiguous cm color palette,
greyPalette	R's gamma-corrected gray palette,
timPalette	Tim's Matlab like color palette,
rampPalette	Color ramp palettes,
seqPalette	Sequential color brewer palettes,
divPalette	Diverging color brewer palettes,
qualiPalette	Qualified color brewer palettes,
focusPalette	Red, green blue focus palettes,
monoPalette	Red, green blue mono palettes.

An interactive plot function allows to create easily interactive plots:

`interactivePlot` a framework for interactive plot displays.

4. Bivariate Interpolation and Kriging

Functions which allow to interpolate and smooth bivariate irregular data sets including linear interpolation, Akima spline interpolation, and kriging:

<code>linearInterp</code>	performs linear spline interpolation,
<code>akimaInterp</code>	performs Akima spline interpolation,
<code>krigeInterp</code>	performs krige interpolation.

4. Vector/Matrix Arithmetics and Linear Algebra Addons

Functions for matrix arithmetics and linear algebra. These functions are often very useful for the manipulation of the data slot of multivariate financial time series.

General Matrix Functions:

<code>triang</code>	Extracts the lower tridiagonal part from a matrix,
<code>Triang</code>	Extracts the upper tridiagonal part from a matrix,
<code>pascal</code>	Creates a Pascal matrix,
<code>hilbert</code>	Creates a Hilbert matrix,
<code>colVec</code>	Creates a column vector from a vector,
<code>rowVec</code>	Creates a row vector from a vector,
<code>isPositiveDefinite</code>	Checks if a matrix is positive definite,
<code>makePositiveDefinite</code>	Forces a matrix to be positive definite,
<code>colIds</code>	Retrieves or sets the colnames of an object,
<code>rowIds</code>	Retrieves or sets the rowumn names.

Linear algebra functions in R's base package include the `%*%` product of two matrices, the `%x%`

Kronecker product, the `det` determinant of a matrix, and the `t` transposed matrix.

Rmetrics adds the following functions:

<code>inv</code>	Returns the inverse of a matrix,
<code>norm</code>	Returns the norm of a matrix,
<code>rk</code>	Returns the rank of a matrix,
<code>tr</code>	Returns trace of a matrix,
<code>vech</code>	Is the operator that stacks the lower triangle,
<code>vec</code>	Is the operator that stacks a matrix.

Note, additional linear algebra functionality is provided in R through the functions `chol` which returns the Cholesky factor matrix, `eigen` which computes eigenvalues and eigenvectors, `svd` which does singular value decomposition, `kappa` which determines the condition number of a matrix, `qr` which performs the QR decomposition of a matrix, `solve` which solves a system of linear equations, together with the functions `backsolve` used when the matrix is upper triangular, and `forwardsolve` used when the matrix is lower triangular.

6. Time Series Generation

For the computation of lagged or leading series the following two functions are provided by Rmetrics:

<code>tslag</code>	Lagged or leading vector/matrix of selected order(s),
<code>pdl</code>	Regressor matrix for polynomial distributed lags.

7. Special Functions Addon

Functions which compute special functions missing in R's base package include:

Heaviside and Related Functions:

<code>Heaviside</code>	Computes Heaviside unit step function,
<code>Sign</code>	Just another signum function,
<code>Delta</code>	Computes delta function,
<code>Boxcar</code>	Computes boxcar function,
<code>Ramp</code>	Computes ramp function.

Generator for Portable Random Innovations:

<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.

8. Some utility functions

Finally we like to mention some further utility functions:

`gridVector` creates from two vectors `x` and `y` all grid points.

Author(s)

The `fUtilities` package was originally written by Diethelm Wuertz and is maintained since 2007 and further developed by him and the Rmetrics core team.

acfPlot *Autocorrelation Function Plots*

Description

Returns plots of autocorrelations including the autocorrelation function ACF, the partial ACF, the lagged ACF, and the Taylor effect plot.

The functions to display stylized facts are:

<code>acfPlot</code>	autocorrelation function plot,
<code>pacfPlot</code>	partial autocorrelation function plot,
<code>lacfPlot</code>	lagged autocorrelation function plot,
<code>teffectPlot</code>	Taylor effect plot.

Usage

```
acfPlot(x, labels = TRUE, ...)
pacfPlot(x, labels = TRUE, ...)

lacfPlot(x, n = 12, lag.max = 20, type = c("returns", "values"),
         labels = TRUE, ...)

teffectPlot(x, deltas = seq(from = 0.2, to = 3, by = 0.2), lag.max = 10,
            ymax = NA, standardize = TRUE, labels = TRUE, ...)
```

Arguments

<code>deltas</code>	the exponents, a numeric vector, by default ranging from 0.2 to 3.0 in steps of 0.2.
<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 <code>TRUE</code> .
<code>lag.max</code>	maximum lag for which the autocorrelation should be calculated, an integer.
<code>n</code>	an integer value, the number of lags.
<code>standardize</code>	a logical value. Should the vector <code>x</code> be standardized?
<code>type</code>	[lacf] - a character string which specifies the type of the input series, either "returns" or series "values". In the case of a return series as input, the required value series is computed by cumulating the financial returns: <code>exp(colCumsums(x))</code>

<code>x</code>	an uni- or multivariate return series of class <code>timeSeries</code> or any other object which can be transformed by the function <code>as.timeSeries()</code> into an object of class <code>timeSeries</code> .
<code>ymax</code>	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

Autocorrelation Functions:

The functions `acfPlot` and `pacfPlot`, `plot` and `estimate` autocorrelation and partial autocorrelation function. The functions allow to get a first view on correlations within the time series. The functions are synonyme function calls for R's `acf` and `pacf` from the the `ts` package.

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 `delta`, $abs(x - \text{mean}(x))^{\text{delta}}$ 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 `delta`. In the case that the above formulated hypothesis is supported, all the curves should peak at the same value around `delta=1`.

Value

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

`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.

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

```
## data -
# require(MASS)
plot(SP500, type = "l", col = "steelblue", main = "SP500")
```

```

abline(h = 0, col = "grey")

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

```

akimaInterp

Bivariate Spline Interpolation

Description

Interpolates bivariate data sets using Akima spline interpolation.

Usage

```

akimaInterp(x, y = NULL, z = NULL, gridPoints = 21,
            xo = seq(min(x), max(x), length = gridPoints),
            yo = seq(min(y), max(y), length = gridPoints), extrap = FALSE)

akimaInterpp(x, y = NULL, z = NULL, xo, yo, extrap = FALSE)

```

Arguments

<code>x, y, z</code>	for <code>akimaInterp</code> the arguments <code>x</code> and <code>y</code> are two numeric vectors of grid points, and <code>z</code> is a numeric matrix or any other rectangular object which can be transformed by the function <code>as.matrix</code> into a matrix object. For <code>akimaInterpp</code> we consider either three numeric vectors of equal length or if <code>y</code> and <code>z</code> are <code>NULL</code> , a list with entries <code>x, y, z</code> , or named data frame with <code>x</code> in the first, <code>y</code> in the second, and <code>z</code> in the third column.
<code>gridPoints</code>	an integer value specifying the number of grid points in <code>x</code> and <code>y</code> direction.
<code>xo, yo</code>	for <code>akimaInterp</code> two numeric vectors of data points spanning the grid, and for <code>akimaInterpp</code> two numeric vectors of data points building pairs for point-wise interpolation.
<code>extrap</code>	a logical, if <code>TRUE</code> then the data points are extrapolated.

Details

Two options are available gridded and pointwise interpolation.

`akimaInterp` is a function wrapper to the `interp` function provided by the contributed R package `akima`. The Fortran code of the Akima spline interpolation routine was written by H. Akima.

Linear surface fitting and krige surface fitting are provided by the functions `linearInterp` and `krigeInterp`.

Value

akimaInterp

returns a list with at least three entries, x, y and z. Note, that the returned values, can be directly used by the persp and contour 3D plotting methods.

akimaInterpp

returns a data.frame with columns "x", "y", and "z".

Note

IMPORTANT: The contributed package `akima` is not in the dependence list of the DESCRIPTION file due to license conditions. The Rmetrics user has to load this package from the CRAN server on his own responsibility, please check the license conditions.

References

Akima H., 1978, *A Method of Bivariate Interpolation and Smooth Surface Fitting for Irregularly Distributed Data Points*, ACM Transactions on Mathematical Software 4, 149-164.

Akima H., 1996, *Algorithm 761: Scattered-Data Surface Fitting that has the Accuracy of a Cubic Polynomial*, ACM Transactions on Mathematical Software 22, 362-371.

See Also

[linearInterp](#), [krigeInterp](#).

Examples

```
## akimaInterp -  
# Akima Interpolation:  
if (require(akima)) {  
  set.seed(1953)  
  x = runif(999) - 0.5  
  y = runif(999) - 0.5  
  z = cos(2*pi*(x^2+y^2))  
  ans = akimaInterp(x, y, z, gridPoints = 41, extrap = FALSE)  
  persp(ans, theta = -40, phi = 30, col = "steelblue",  
        xlab = "x", ylab = "y", zlab = "z")  
  contour(ans)  
}
```

baseMethods

*Generic Functions Extensions***Description**

Basic extensions which which add and/or modify additional functionality which is not available in R's basic packages.

Added and/or modified functions:

attach	extends attach function,
rank	extends rank function,
stdev	adds stdev function,
termPlot	adds term plot function,
volatility	adds volatility function.

Usage

```
## Default S3 method:
stdev(x, na.rm = FALSE)
```

```
## Default S3 method:
termPlot(model, ...)
```

```
## Default S3 method:
volatility(object, ...)
```

Arguments

method	[align] - a character string which specifies the alignment method to be used. Choices are "linear" or "constant". [cov] - a character string indicating which correlation coefficient (or covariance) is to be computed. One of "pearson" (default), "kendall", or "spearman", can be abbreviated.
na.rm	an logical value - should the NA values be removed.

<code>model</code>	<code>[termPlot]</code> - a fitted model object.
<code>object</code>	<code>[volatility]</code> - an object from which to extract the volatility.
<code>x</code>	<code>[align]</code> - x-coordinates of the points to be aligned. <code>[log][sort][var]</code> - first argument. <code>[print.control]</code> - <code>cr</code> prints an unlisted object of class <code>control</code> . <code>[as.matrix.ts][as.matrix.mts]</code> - an univariate or multivariate time series object of class <code>"ts"</code> or <code>"mts"</code> which will be transformed into an one-column or multi-column rectangular object of class <code>"matrix"</code> . <code>[as.POSIXlt]</code> - an object to be converted.
<code>...</code>	arguments to be passed.

Details

For details we refer to the original help pages.

BasicStatistics *Basic Time Series Statistics*

Description

Computes basic financial time series statistics.

List of Functions:

`basicStats` Computes an overview of basic statistical values.

Usage

```
basicStats(x, ci = 0.95)
```

Arguments

<code>ci</code>	confidence interval, a numeric value, by default 0.95, i.e. 95 percent.
<code>x</code>	an object of class <code>"timeSeries"</code> or any other object which can be transformed by the function <code>as.timeSeries</code> into an object of class <code>timeSeries</code> . The latter case, other than <code>timeSeries</code> objects, is more or less untested.

Value

`basicsStats`

returns a 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.

Examples

```
## basicStats -
# Simulated Monthly Return Data:
tS = timeSeries(matrix(rnorm(12)), timeCalendar())
basicStats(tS)
```

BoxPlot

Time Series Box Plots

Description

Returns a box or a box percentile plot.

List of Functions:

<code>boxPlot</code>	Returns a side-by-side standard box plot,
<code>boxPercentilePlot</code>	Returns a side-by-side box-percentile plot.

Usage

```
boxPlot(x, col = "steelblue", title = TRUE, ...)
boxPercentilePlot(x, col = "steelblue", title = TRUE, ...)
```

Arguments

<code>col</code>	the color for the series. In the univariate case use just a color name like the default, <code>col="steelblue"</code> , in the multivariate case we recommend to select the colors from a color palette, e.g. <code>col=heat.colors(ncol(x))</code> .
<code>title</code>	a logical flag, by default TRUE. Should a default title added to the plot?
<code>x</code>	an object of class "timeSeries" or any other object which can be transformed by the function <code>as.timeSeries</code> into an object of class <code>timeSeries</code> . The latter case, other than <code>timeSeries</code> objects, is more or less untested.
<code>...</code>	optional arguments to be passed.

Value

displays a time series plot.

Examples

```
## boxplot -
```

characterTable *Table of Characters*

Description

Displays a table of numerical equivalents to Latin characters.

Usage

```
characterTable(font = 1, cex = 0.7)
```

Arguments

`cex` a numeric value, determines the character size, the default size is 0.7.
`font` an integer value, the number of the font, by default font number 1.

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.

See Also

`link{colorTable}`, `link{symbolTable}`.

Examples

```
## Character Table for Font 2:  
characterTable(font = 1)
```

colorLocator *Color Selection*

Description

Displays R's 657 named colors for selection and returns optionally R's color names.

Usage

```
colorLocator(locator = FALSE)  
colorMatrix(locator = TRUE)
```

Arguments

`locator` a logical flag, activates `locator` for interactive selection of color names, default is `FALSE`.

Details

Color Locator:

The `colorLocator` function plots R's 657 named colors. If `locator=TRUE` then you can interactively point and click to select the colors for which you want names. To end selection, right click on the mouse and select 'Stop', then R returns the selected color names.

The functions used here are wrappers to the functions provided by Tomas Aragon in the contributed R package. `epitools`.

Value

Color Locator:

`colorsLocator` generates a plot with R colors and, when `locator=TRUE`, returns matrix with graph coordinates and names of colors selected. `colorsMatrix` quietly returns matrix of names.

See Also

`link{colorPalette}`, `link{colorTable}`.

Examples

```
## colorLocator -
  colorLocator()
```

`colorPalette`

Color Palettes

Description

Functions to create color palettes.

The functions are:

<code>rainbowPalette</code>	Contiguous rainbow color palette,
<code>heatPalette</code>	Contiguous heat color palette,
<code>terrainPalette</code>	Contiguous terrain color palette,
<code>topoPalette</code>	Contiguous topo color palette,
<code>cmPalette</code>	Contiguous cm color palette,
<code>greyPalette</code>	R's gamma-corrected gray palette,
<code>timPalette</code>	Tim's Matlab like color palette,
<code>rampPalette</code>	Color ramp palettes,
<code>seqPalette</code>	Sequential color brewer palettes,

divPalette	Diverging color brewer palettes,
qualiPalette	Qualified color brewer palettes,
focusPalette	Red, green blue focus palettes,
monoPalette	Red, green blue mono palettes.

Usage

```
rainbowPalette(n = 64, ...)
heatPalette(n = 64, ...)
terrainPalette(n = 64, ...)
topoPalette(n = 64, ...)
cmPalette(n = 64, ...)

greyPalette(n = 64, ...)
timPalette(n = 64)

rampPalette(n, name = c("blue2red", "green2red", "blue2green",
  "purple2green", "blue2yellow", "cyan2magenta"))

seqPalette(n, name = c(
  "Blues", "BuGn", "BuPu", "GnBu", "Greens", "Greys", "Oranges",
  "OrRd", "PuBu", "PuBuGn", "PuRd", "Purples", "RdPu", "Reds",
  "YlGn", "YlGnBu", "YlOrBr", "YlOrRd"))

divPalette(n, name = c(
  "BrBG", "PiYG", "PRGn", "PuOr", "RdBu", "RdGy", "RdYlBu", "RdYlGn",
  "Spectral"))

qualiPalette(n, name = c(
  "Accent", "Dark2", "Paired", "Pastel1", "Pastel2", "Set1", "Set2",
  "Set3"))

focusPalette(n, name = c("redfocus", "greenfocus", "bluefocus"))
monoPalette(n, name = c("redmono", "greenmono", "bluemono"))
```

Arguments

n	an integer, giving the number of greys or colors to be constructed.
name	a character string, the name of the color set.
...	arguments to be passed, see the details section

Details

All Rmetrics' color sets are named as `fooPalette` where the prefix `foo` denotes the name of the underlying color set.

R's Contiguous Color Palettes:

Palettes for `n` contiguous colors are implemented in the `grDevices` package. To be conform with

Rmetrics' naming convention for color palettes we have build a wrapper around the underlying functions. These are the `rainbowPalette`, `heatPalette`, `terrainPalette`, `topoPalette`, and the `cmPalette`. Conceptually, all of these functions actually use (parts of) a line cut out of the 3-dimensional color space, parametrized by the function `hsv(h, s, v, gamma)`, where `gamma=1` for the `fooPalette` function, and hence, equispaced hues in RGB space tend to cluster at the red, green and blue primaries. Some applications such as contouring require a palette of colors which do not wrap around to give a final color close to the starting one. To pass additional arguments to the underlying functions we refer to consult `help(rainbow)`. With `rainbow`, the parameters `start` and `end` can be used to specify particular subranges of hues. Synonyme function calls are `rainbow`, `heat.colors`, `terrain.colors`, `topo.colors`, and the `cm.colors`.

R's Gamma-Corrected Gray Palette:

The function `grayPalette` chooses a series of `n` gamma-corrected gray levels. The range of the gray levels can be optionally monitored through the `...` arguments, for details `help(gray.colors)`, which is a synonyme function call in the `grDevices` package.

Tim's Matlab like Color Palette:

The function `timPalette` creates a color set ranging from blue to red, and passes through the colors cyan, yellow, and orange. It comes from the Matlab software, originally used in fluid dynamics simulations. The function here is a copy from R's contributed package `fields` doing a spline interpolation on `n=64` color points.

Color Ramp Palettes:

The function `rampPalette` creates several color ramps. The function is implemented from Tim Keitt's contributed R package `colorRamps`. Supported through the argument `name` are the following color ramps: "blue2red", "green2red", "blue2green", "purple2green", "blue2yellow", "cyan2magenta".

Color Brewer Palettes:

The functions `seqPalette`, `divPalette`, and `qualiPalette` create color sets according to R's contributed `RColorBrewer` package. The first letter in the function name denotes the type of the color set: "s" for sequential palettes, "d" for diverging palettes, and "q" for qualitative palettes. *Sequential palettes* are suited to ordered data that progress from low to high. Lightness steps dominate the look of these schemes, with light colors for low data values to dark colors for high data values. The sequential palettes names are: Blues, BuGn, BuPu, GnBu, Greens, Greys, Oranges, OrRd, PuBu, PuBuGn, PuRd, Purples, RdPu, Reds, YlGn, YlGnBu, YlOrBr, YlOrRd. *Diverging palettes* put equal emphasis on mid-range critical values and extremes at both ends of the data range. The critical class or break in the middle of the legend is emphasized with light colors and low and high extremes are emphasized with dark colors that have contrasting hues. The diverging palettes names are: BrBG, PiYG, PRGn, PuOr, RdBu, RdGy, RdYlBu, RdYlGn, Spectral. *Qualitative palettes* do not imply magnitude differences between legend classes, and hues are used to create the primary visual differences between classes. Qualitative schemes are best suited to representing nominal or categorical data. The qualitative palettes names are: Accent, Dark2, Paired,

Pastel1, Pastel2, Set1, Set2, Set3.

In contrast to the original color brewer palettes, the palettes here are created by spline interpolation from the color variation with the most different values, i.e for the sequential palettes these are 9 values, for the diverging palettes these are 11 values, and for the qualitative palettes these are between 8 and 12 values depending on the color set.

Graph Color Palettes:

The function `perfanPalette` creates color sets inspired by R's cotributed package `PerformanceAnalytics`. These color palettes have been designed to create readable, comparable line and bar graphs with specific objectives.

Focused Color Palettes: Color sets designed to provide focus to the data graphed as the first element. This palette is best used when there is clearly an important data set for the viewer to focus on, with the remaining data being secondary, tertiary, etc. Later elements graphed in diminishing values of gray.

Monochrome Color Palettes: These include color sets for monochrome color displays.

Value

returns a character string of color strings.

Note

The palettes are wrapper functions provided in several contributed R packages. These include:

Cynthia Brewer and Mark Harrower for the brewer palettes,
 Peter Carl and Brian G. Peterson for the "PerformanceAnalytics" package,
 Tim Keitt for the "colorRamps" package,
 Ross Ihaka for the "colorspace" package,
 Tomas Aragon for the "epitools" package,
 Doug Nychka for the "fields" package,
 Erich Neuwirth for the "RColorBrewer" package.

Additional undocumented hidden functions:

<code>.asRGB</code>	Converts any R color to RGB (red/green/blue),
<code>.chcode</code>	Changes from one to another number system,
<code>.hex.to.dec</code>	Converts heximal numbers do decimal numbers,
<code>.dec.to.hex</code>	Converts decimal numbers do heximal numbers.

Examples

```
## GreyPalette:
greyPalette()
```

Description

Displays a Table of color codes and plots the colors themselves.

Usage

```
colorTable(cex = 0.7)
```

Arguments

`cex` a numeric value, determines the character size in the color plot, the default size is 0.7.

Value

`colorTable`
returns a table of plot colors with the associated color numbers.

See Also

```
link{characterTable}, link{symbolTable}.
```

Examples

```
## Color Table:  
colorTable()
```

`colVec`*Column and Row Vectors*

Description

Creates a column or row vector from a numeric vector.

Usage

```
colVec(x)  
rowVec(x)
```

Arguments

`x` a numeric vector.

Details

The functions `colVec` and `rowVec` transform a vector into a column and row vector, respectively. A column vector is a matrix object with one column, and a row vector is a matrix object with one row.

Examples

```
## Create a numeric Vector:
x = rnorm(5)

## Column and Row Vectors:
colVec(x)
rowVec(x)
```

correlationTest *Correlation Tests*

Description

Tests if two series are correlated.

Usage

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

pearsonTest(x, y, title = NULL, description = NULL)
kendallTest(x, y, title = NULL, description = NULL)
spearmanTest(x, y, title = NULL, description = NULL)
```

Arguments

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

Details

The function `correlationTest` tests for association between paired samples allowing 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 hypothesis tests implemented from R's package `ctest`.

References

Conover, W. J. (1971); *Practical nonparametric statistics*, New York: John Wiley & Sons.
 Lehmann E.L. (1986); *Testing Statistical Hypotheses*, John Wiley and Sons, New York.

See Also

[locationTest](#), [scaleTest](#), [varianceTest](#).

Examples

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

## correlationTest -
correlationTest(x, y, "pearson")
correlationTest(x, y, "kendall")
spearmanTest(x, y)
```

decor

Decor Functions

Description

Functions for decorating plots.

The plot utility functions are:

decor	simple decoration function,
hgrid	creates horizontal grid lines,
vgrid	creates vertical grid lines,
boxL	creates a L-shaped box,
box_	creates a bogttom line box,
copyright	adds Rmetrics copyright to a plot.

Usage

```
decor()

hgrid(ny = NULL, ...)
vgrid(nx = NULL, ...)

boxL(col = "white")
box_(col = c("white", "black"))

copyright()
```

Arguments

col	the color of the background, "black" and foreground "white" lines of the box.
nx, ny	number of cells of the grid in x or y direction. When NULL, as per default, the grid aligns with the tick marks on the corresponding default axis (i.e., tickmarks as computed by axTicks).
...	additional arguments passed to the grid() function.

Examples

```
## Test Plot Function:
plot(x = rnorm(100), type = "l", col = "red",
     xlab = "", ylab = "Variates", las = 1)
title("Normal Deviates", adj = 0)
hgrid()
boxL()
copyright()
```

distCheck

Distribution Check

Description

Tests properties of a distribution.

Usage

```
distCheck(fun = "norm", n = 1000, robust = TRUE, subdivisions = 100, ...)
```

Arguments

fun a character string denoting the name of the distribution.
n an integer specifying the number of random variates to be generated.
robust a logical flag, should robust estimates be used? By default TRUE.
subdivisions an integer specifying the numbers of subdivisions in integration.
... the distributional parameters.

Examples

```
## distCheck:
  distCheck("norm", mean = 1, sd = 1)
```

DistributionFits *Parameter Fit of a Distribution*

Description

A collection and description of moment and maximum likelihood estimators to fit the parameters of a distribution.

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.

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, control = list(),
          trace = FALSE, title = NULL, description = NULL)

## S4 method for signature 'fDISTFIT':
show(object)
```

Arguments

control [stableFit] -
 a list of control parameters, see function nlminb.

<code>alpha, beta, gamma, delta</code>	<code>[stable]</code> - The parameters are <code>alpha</code> , <code>beta</code> , <code>gamma</code> , and <code>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>description</code>	a character string which allows for a brief description.
<code>df</code>	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>	<code>[show]</code> - an S4 class object as returned from the fitting functions.
<code>doplot</code>	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>	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: <code>"mle"</code> , the maximum log likelihood approach, or <code>"qm"</code> , McCulloch's quantile method.
<code>x</code>	a numeric vector.
<code>...</code>	parameters to be parsed.

Details

Stable Parameter Estimation:

Estimation techniques based on the quantiles of an empirical sample were first suggested by Fama and Roll [1971]. However their technique was limited to symmetric distributions and suffered from a small asymptotic bias. McCulloch [1986] developed a technique that uses five quantiles from a sample to estimate `alpha` and `beta` without asymptotic bias. Unfortunately, the estimators provided by McCulloch have restriction `alpha > 0.6`.

Value

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

<code>estimate</code>	the point at which the maximum value of the log likelihood function is obtained.
<code>minimum</code>	the value of the estimated maximum, i.e. the value of the log likelihood function.
<code>code</code>	an integer indicating why the optimization process terminated.
<code>gradient</code>	the gradient at the estimated maximum.

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

Examples

```
## nFit -
# Simulate random normal variates N(0.5, 2.0):
set.seed(1953)
s = rnorm(n = 1000, 0.5, 2)

## nigFit -
# Fit Parameters:
nFit(s, doplot = TRUE)
```

fHTEST

*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.

Usage

```
## S4 method for signature 'fHTEST':
show(object)
```

Arguments

object	[show] - an S4 object of class "fHTEST".
--------	---

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:

@call	the function call.
@data	the data as specified by the input argument(s).
@test	a list whose elements contain the results from the statistical test. The information provided is similar to a list object of class "htest".
@title	a character string with the name of the test. This can be overwritten specifying a user defined input argument.
@description	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.

Examples

```
## fHTEST -
  getClass("fHTEST")
  getSlots("fHTEST")
```

getS4

General S4 Class Extractor Functions

Description

A collection and description of functions to extract slots from S4 class objects.

The extractor functions are:

<code>getCall</code>	Extracts the call slot from a S4 object,
<code>getModel</code>	Extracts the model slot from a S4 object,
<code>getTitle</code>	Extracts the title slot from a S4 object,
<code>getDescription</code>	Extracts the description slot from a S4 object,
<code>getSlot</code>	Extracts a specified slot from a S4 object.

Usage

```
getCall(object)
getModel(object)
getTitle(object)
getDescription(object)

getSlot(object, slotName)
```

Arguments

<code>object</code>	an object of class S4.
<code>slotName</code>	a character string, the name of the slot to be extracted from the S4 object.

Value

```

getCall
getModel
getTitle
getDescription
getSlot
return the content of the slot.

```

Examples

```

## Example S4 Representation:
# Hypothesis Testing with Control Settings
setClass("hypTest",
  representation(
    call = "call",
    data = "numeric",
    test = "list",
    description = "character")
)

## Shapiro Wilk Normality Test
swTest = function(x, description = "") {
  ans = shapiro.test(x)
  class(ans) = "list"
  new("hypTest",
    call = match.call(),
    data = x,
    test = ans,
    description = description)
}
test = swTest(x = rnorm(500), description = "500 RVs")

## Extractor Functions:
isS4(test)
getCall(test)
getDescription(test)

```

gh

Generalized Hyperbolic Distribution

Description

Density, distribution function, quantile function and random generation for the generalized hyperbolic distribution.

Usage

```

dgh(x, alpha = 1, beta = 0, delta = 1, mu = 0, lambda = 1, log = FALSE)
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)
```

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; 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`.

`log` a logical flag by default `FALSE`. Should labels and a main title drawn to the plot?

`n` number of observations.

`p` a numeric vector of probabilities.

`x`, `q` a numeric vector of quantiles.

`...` arguments to be passed to the function `integrate`.

Details

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

Value

All values for the `*gh` 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.

Author(s)

David Scott for code implemented from R's contributed package `HyperbolicDist`.

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

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

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

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

## qgh -
# Compute Quantiles:
qgh(pgh(seq(-5, 5, 1), alpha = 1, beta = 0.3, delta = 1),
     alpha = 1, beta = 0.3, delta = 1)
```

ghFit

GH Distribution Fit

Description

Estimates the distributional parameters for a generalized hyperbolic distribution.

Usage

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

Arguments

x	a numeric vector.
alpha, beta, delta, mu, lambda	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.
scale	a logical flag, by default TRUE. Should the time series be scaled by its standard deviation to achieve a more stable optimization?
doplot	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>trace</code>	a logical flag. Should the parameter estimation process be traced?
<code>title</code>	a character string which allows for a project title.
<code>description</code>	a character string which allows for a brief description.
<code>...</code>	parameters to be parsed.

Details

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

Value

returns a list with the following components:

<code>estimate</code>	the point at which the maximum value of the log likelihood function is obtained.
<code>minimum</code>	the value of the estimated maximum, i.e. the value of the log likelihood function.
<code>code</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.
<code>gradient</code>	the gradient at the estimated maximum.
<code>steps</code>	number of function calls.

Examples

```
## ghFit -
# Simulate Random Variates:
set.seed(1953)
s = rgh(n = 1000, alpha = 1.5, beta = 0.3, delta = 0.5, mu = -1.0)

## ghFit -
# Fit Parameters:
ghFit(s, alpha = 1, beta = 0, delta = 1, mu = mean(s), doplot = TRUE)
```

ghMode

Generalized Hyperbolic Mode

Description

Computes the mode of the generalized hyperbolic function.

Usage

```
ghMode(alpha = 1, beta = 0, delta = 1, mu = 0, lambda = 1)
```

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; location parameter `mu`, by default 0. 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`.

Value

returns the mode for the generalized hyperbolic distribution. A numeric value.

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

```
## ghMode -
  ghMode ()
```

 ghSlider

Generalized Hyperbolic Distribution Slider

Description

Displays interactively the dependence of the generalized hyperbolic distribution on its parameters.

Usage

```
ghSlider()
```

Value

a tcl/tk based graphical user interface.

This is a nice display for educational purposes to investigate the densities and probabilities of the generalized hyperbolic distribution.

Examples

```
## ghSlider -
# ghSlider()
```

ght

Generalized Hyperbolic Student-t

Description

Density, distribution function, quantile function and random generation for the hyperbolic distribution.

Usage

```
dght(x, beta = 0.1, delta = 1, mu = 0, nu = 10)
pght(q, beta = 0.1, delta = 1, mu = 0, nu = 10)
qght(p, beta = 0.1, delta = 1, mu = 0, nu = 10)
rght(n, beta = 0.1, delta = 1, mu = 0, nu = 10)
```

Arguments

<code>x, q</code>	a numeric vector of quantiles.
<code>p</code>	a numeric vector of probabilities.
<code>n</code>	number of observations.
<code>beta, delta, mu</code>	skewness parameter <code>beta</code> , <code>abs(beta)</code> is in the range (0, alpha); scale parameter <code>delta</code> , <code>delta</code> must be zero or positive; location parameter <code>mu</code> , by default 0. These are the parameters in the first parameterization.
<code>nu</code>	a numeric value, the number of degrees of freedom.

Value

All values for the `*ght` 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.

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

```
## ght -
#
```

ghtFit	<i>GHT Distribution Fit</i>
--------	-----------------------------

Description

Estimates the distributional parameters for a generalized hyperbolic Student-t distribution.

Usage

```
ghtFit(x, beta = 0.1, delta = 1, mu = 0, nu = 10,
       scale = TRUE, doplot = TRUE, span = "auto", trace = TRUE,
       title = NULL, description = NULL, ...)
```

Arguments

`x` a numeric vector.
`beta, delta, mu`

The parameters are `beta`, `delta`, and `mu`: skewness parameter `beta` is in the range (0, alpha); scale parameter `delta` must be zero or positive; location parameter `mu`, by default 0; and lambda parameter `lambda`, by default 1.

`nu` defines the number of degrees of freedom. Note, `alpha` takes the limit of `abs(beta)`, and `lambda=-nu/2`.

`scale` a logical flag, by default TRUE. Should the time series be scaled by its standard deviation to achieve a more stable optimization?

doplot	a logical flag. Should a plot be displayed?
span	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.
trace	a logical flag. Should the parameter estimation process be traced?
title	a character string which allows for a project title.
description	a character string which allows for a brief description.
...	parameters to be parsed.

Details

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

Value

returns 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.

Examples

```
## ghtFit -
# Simulate Random Variates:
set.seed(1953)

## ghtFit -
# Fit Parameters:
```

gridVector *Grid Vector Coordinates*

Description

Creates from two vectors rectangular grid coordinates..

Usage

```
gridVector(x, y = NULL)
```

Arguments

`x`, `y` two numeric vectors of length `m` and `n` which span the rectangular grid of size `m` times `n`. If `y` takes the default value, `NULL`, then `y=x`.

Value

returns a list with two entries named `$X` and `$Y`, giving the coordinates which span the bivariate grid.

See Also

[expand.grid](#).

Examples

```
## gridVector -
gridVector((0:10)/10)
gridVector((0:10)/10, (0:10)/10)
```

Heaviside *Heaviside and Related Functions*

Description

Functions which compute the Heaviside and related functions. These include the sign function, the delta function, the boxcar function, and the ramp function.

The functions are:

Heaviside	Computes Heaviside unit step function,
Sign	Just another signum function,
Delta	Computes delta function,
Boxcar	Computes boxcar function,
Ramp	Computes ramp function.

Usage

```
Heaviside(x, a = 0)
Sign(x, a = 0)
Delta(x, a = 0)
Boxcar(x, a = 0.5)
Ramp(x, a = 0)
```

Arguments

a a numeric value, the location of the break.

x a numeric vector.

Details

The Heaviside step function `Heaviside` is 1 for $x > a$, $1/2$ for $x = a$, and 0 for $x < a$.

The Sign function `Sign` is 1 for $x > a$, 0 for $x = a$, and -1 for $x < a$.

The delta function `Delta` is defined as: $\text{Delta}(x) = d/dx H(x-a)$.

The boxcar function `Boxcar` is defined as: $\text{Boxcar}(x) = H(x+a) - H(x-a)$.

The ramp function is defined as: $\text{Ramp}(x) = (x-a) * H(x-a)$.

Value

returns the function values of the selected function.

Note

The Heaviside function is used in the implementation of the skew Normal, Student-t, and Generalized Error distributions, distributions functions which play an important role in modelling GARCH processes.

References

Weisstein W. (2004); <http://mathworld.wolfram.com/HeavisideStepFunction.html>, Mathworld.

See Also

`GarchDistribution`, `GarchDistributionFits`.

Examples

```
## Heaviside -
x = sort(round(c(-1, -0.5, 0, 0.5, 1, 5*rnorm(5)), 2))
h = Heaviside(x)

## Sign -
s = Sign(x)

## Delta -
d = Delta(x)

## Boxcar -
Pi = Boxcar(x)

## Ramp -
r = Ramp(x)
cbind(x = x, Step = h, Signum = s, Delta = d, Pi = Pi, R = r)
```

hilbert

Hilbert Matrix

Description

Creates a Hilbert matrix.

Usage

```
hilbert(n)
```

Arguments

n an integer value, the dimension of the square matrix.

Details

In linear algebra, a Hilbert matrix is a matrix with the unit fraction elements.

The Hilbert matrices are canonical examples of ill-conditioned matrices, making them notoriously difficult to use in numerical computation. For example, the 2-norm condition number of a 5x5 Hilbert matrix above is about 4.8e5.

The Hilbert matrix is symmetric and positive definite.

Value

hilbert generates a Hilbert matrix of order n.

References

- Hilbert D., *Collected papers*, vol. II, article 21.
- Beckermann B, (2000); *The condition number of real Vandermonde, Krylov and positive definite Hankel matrices*, Numerische Mathematik 85, 553–577, 2000.
- Choi, M.D., (1983); *Tricks or Treats with the Hilbert Matrix*, American Mathematical Monthly 90, 301–312, 1983.
- Todd, J., (1954); *The Condition Number of the Finite Segment of the Hilbert Matrix*, National Bureau of Standards, Applied Mathematics Series 39, 109–116.
- Wilf, H.S., (1970); *Finite Sections of Some Classical Inequalities*, Heidelberg, Springer.

Examples

```
## Create a Hilbert Matrix:
H = hilbert(5)
H
```

HistogramPlot *Histogram and Density Plots*

Description

Returns a histogram, a density, or a logarithmic density plot.

List of Functions:

histPlot	Returns a tailored histogram plot,
densityPlot	Returns a tailored kernel density estimate plot,
logDensityPlot	Returns a tailored log kernel density estimate plot.

Usage

```
histPlot(x, labels = TRUE, col = "steelblue", fit = TRUE,
         title = TRUE, grid = TRUE, rug = TRUE, skip = FALSE, ...)
densityPlot(x, labels = TRUE, col = "steelblue", fit = TRUE, hist = TRUE,
           title = TRUE, grid = TRUE, rug = TRUE, skip = FALSE, ...)
logDensityPlot(x, labels = TRUE, col = "steelblue", robust = TRUE,
              title = TRUE, grid = TRUE, rug = TRUE, skip = FALSE, ...)
```

Arguments

col	the color for the series. In the univariate case use just a color name like the default, col="steelblue", in the multivariate case we recommend to select the colors from a color palette, e.g. col=heat.colors(ncol(x)).
fit	a logical flag, should a fit added to the Plot?

grid	a logical flag, should a grid be added to the plot? By default TRUE. To plot a horizontal lines only use <code>grid="h"</code> and for vertical lines use <code>grid="v"</code> , respectively.
hist	a logical flag, by default TRUE. Should a histogram to be underlaid to the plot?
labels	a logical flag, should the plot be returned with default labels and decorated in an automated way? By default TRUE.
rug	a logical flag, by default TRUE. Should a rug representation of the data added to the plot?
skip	a logical flag, should zeros be skipped in the return Series?
robust	a logical flag, by default TRUE. Should a robust fit added to the plot?
title	a logical flag, by default TRUE. Should a default title added to the plot?
x	an object of class "timeSeries" or any other object which can be transformed by the function <code>as.timeSeries</code> into an object of class <code>timeSeries</code> . The latter case, other then <code>timeSeries</code> objects, is more or less untested.
...	optional arguments to be passed.

Value

displays a time series plot.

Examples

```
## histPlot
```

hyp *Hyperbolic Distribution*

Description

Density, distribution function, quantile function and random generation for the hyperbolic distribution.

Usage

```
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))
```

Arguments

<code>alpha</code> , <code>beta</code> , <code>delta</code> , <code>mu</code>	shape parameter <code>alpha</code> ; skewness parameter <code>beta</code> , $\text{abs}(\text{beta})$ is in the range $(0, \text{alpha})$; scale parameter <code>delta</code> , <code>delta</code> must be zero or positive; location parameter <code>mu</code> , by default 0. These is the meaning of the parameters in the first parameterization <code>pm=1</code> which is the default parameterization selection. In the second parameterization, <code>pm=2</code> <code>alpha</code> and <code>beta</code> take the meaning of the shape parameters (usually named) <code>zeta</code> and <code>rho</code> . In the third parameterization, <code>pm=3</code> <code>alpha</code> and <code>beta</code> take the meaning of the shape parameters (usually named) <code>xi</code> and <code>chi</code> . In the fourth parameterization, <code>pm=4</code> <code>alpha</code> and <code>beta</code> take the meaning of the shape parameters (usually named) <code>a.bar</code> and <code>b.bar</code> .
<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</code> , <code>q</code>	a numeric vector of quantiles.
<code>...</code>	arguments to be passed to the function <code>integrate</code> .

Details

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

Value

All values for the `*hyp` 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.

Author(s)

David Scott for code implemented from R's contributed package `HyperbolicDist`.

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

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

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

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

## hyp -
# Compute Quantiles:
qhyp(phyp(seq(-5, 5, 1), alpha = 1, beta = 0.3, delta = 1),
     alpha = 1, beta = 0.3, delta = 1)
```

hypFit

Fit of a Hyperbolic Distribution

Description

Estimates the parameters of a hyperbolic distribution.

Usage

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

Arguments

alpha, beta, delta, mu

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

description	a character string which allows for a brief description.
doplot	a logical flag. Should a plot be displayed?
scale	a logical flag, by default TRUE. Should the time series be scaled by its standard deviation to achieve a more stable optimization?
span	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.
title	a character string which allows for a project title.
trace	a logical flag. Should the parameter estimation process be traced?
x	a numeric vector.
...	parameters to be parsed.

Details

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.

Examples

```
## rhyp -
# Simulate Random Variates:
set.seed(1953)
s = rhyp(n = 1000, alpha = 1.5, beta = 0.3, delta = 0.5, mu = -1.0)

## hypFit -
# Fit Parameters:
hypFit(s, alpha = 1, beta = 0, delta = 1, mu = mean(s), doplot = TRUE)
```

hypMode

*Hyperbolic Mode***Description**

Computes the mode of the hyperbolic function.

Usage

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

Arguments

alpha, beta, delta, mu

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

pm

an integer value between 1 and 4 for the selection of the parameterization. The default takes the first parameterization.

Value

returns the mode in the appropriate parameterization for the hyperbolic distribution. A numeric value.

Author(s)

David Scott for code implemented from R's contributed package HyperbolicDist.

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

```
## hypMode -
hypMode()
```

hypSlider *Hyperbolic Distribution Slider*

Description

Displays interactively the dependence of the hyperbolic distribution on its parameters.

Usage

```
hypSlider()
```

Value

a tcl/tk based graphical user interface.

This is a nice display for educational purposes to investigate the densities and probabilities of the hyperbolic distribution.

Examples

```
## hypSlider -
#
```

Ids *Set and Retrieve Column/Row Names*

Description

Sets and retrieves column and row names. The functions are for compatibility with SPlus.

Usage

```
colIds(x, ...)
rowIds(x, ...)
```

Arguments

x a numeric matrix.
 ... arguments to be passed.

Details

Usual in R the functions `colnames`, and `rownames` are used to retrieve and set the names of matrices. The functions `rowIds` and `colIds`, are S-Plus like synonyms.

Examples

```
## pascal -  
# Create Pascal Matrix:  
P = pascal(3)  
P  
  
## rownames -  
rownames(P) <- letters[1:3]  
P  
  
## colIds<- -  
colIds(P) <- as.character(1:3)  
P
```

interactivePlot *Interactive Plot Utility*

Description

Plots with emphasis on interactive plots.

Usage

```
interactivePlot(x, choices = paste("Plot", 1:9),  
              plotFUN = paste("plot.", 1:9, sep = "."), which = "all", ...)
```

Arguments

<code>choices</code>	a vector of character strings for the choice menu. By Default "Plot 1" ... "Plot 9" allowing for 9 plots at maximum.
<code>plotFUN</code>	a vector of character strings naming the plot functions. By Default "plot.1" ... "plot.9" allowing for 9 plots at maximum.
<code>which</code>	plot selection, which graph should be displayed? If "which" is a character string named "ask" the user is interactively asked which to plot, if a logical vector of length N, those plots which are set TRUE are displayed, if a character string named "all" all plots are displayed.
<code>x</code>	an object to be plotted.
<code>...</code>	additional arguments passed to the FUN or plot function.

Examples

```
## Test Plot Function:
testPlot = function(x, which = "all", ...) {
  # Plot Function and Addons:
  plot.1 <- function(x, ...) plot(x, ...)
  plot.2 <- function(x, ...) acf(x, ...)
  plot.3 <- function(x, ...) hist(x, ...)
  plot.4 <- function(x, ...) qqnorm(x, ...)
  # Plot:
  interactivePlot(x,
    choices = c("Series Plot", "ACF", "Histogram", "QQ Plot"),
    plotFUN = c("plot.1", "plot.2", "plot.3", "plot.4"),
    which = which, ...)
  # Return Value:
  invisible()
}
# Plot:
par(mfrow = c(2, 2), cex = 0.7)
testPlot(rnorm(500))

# Try:
# par(mfrow = c(1,1))
# testPlot(rnorm(500), which = "ask")
```

 inv

The Inverse of a Matrix

Description

Returns the inverse of a matrix.

Usage

```
inv(x)
```

Arguments

x a numeric matrix.

Value

returns the inverse matrix.

Note

The function `inv` is a synonyme to the function `solve`.

References

Golub, van Loan, (1996); *Matrix Computations*, 3rd edition. Johns Hopkins University Press.

Examples

```
## Create Pascal Matrix:
P = pascal(5)
P

## Compute the Inverse Matrix:
inv(P)

## Check:
inv(P)

## Alternatives:
chol2inv(chol(P))
solve(P)
```

krigeInterp

Bivariate Krige Interpolation

Description

Bivariate Krige Interpolation.

Usage

```
krigeInterp(x, y = NULL, z = NULL, gridPoints = 21,
            xo = seq(min(x), max(x), length = gridPoints),
            yo = seq(min(y), max(y), length = gridPoints),
            extrap = FALSE, polDegree = 6)
```

Arguments

<code>x, y, z</code>	the arguments <code>x</code> and <code>y</code> are two numeric vectors of grid points, and <code>z</code> is a numeric matrix or any other rectangular object which can be transformed by the function <code>as.matrix</code> into a matrix object.
<code>gridPoints</code>	an integer value specifying the number of grid points in <code>x</code> and <code>y</code> direction.
<code>xo, yo</code>	two numeric vectors of data points spanning the grid.
<code>extrap</code>	a logical, if <code>TRUE</code> then the data points are extrapolated.
<code>polDegree</code>	the polynomial krige degree, an integer ranging between 1 and 6.

Value

`krigeInterp`

returns a list with at least three entries, `x`, `y` and `z`. Note, that the returned values, can be directly used by the `persp` and `contour` 3D plotting methods.

Note

The function `krigeInterp` requires loading of the R package `spatial`.

See Also

[akimaInterp](#), [linearInterp](#).

Examples

```
## krigInterp -  
# Kriging:  
set.seed(1953)  
x = runif(999) - 0.5  
y = runif(999) - 0.5  
z = cos(2*pi*(x^2+y^2))  
ans = krigInterp(x, y, z, extrap = FALSE)  
persp(ans, theta = -40, phi = 30, col = "steelblue",  
       xlab = "x", ylab = "y", zlab = "z")  
contour(ans)
```

kron

Kronecker Product

Description

Returns the Kronecker product.

Usage

```
kron(x, y)
```

Arguments

`x`, `y` two numeric matrixes.

Details

The *Kronecker product* can be computed using the operator `%x%` or alternatively using the function `kron` for SPlus compatibility.

Note

`kron` is a synonyme to `%x%`.

References

Golub, van Loan, (1996); *Matrix Computations*, 3rd edition. Johns Hopkins University Press.

Examples

```
## Create Pascal Matrix:
P = pascal(3)
P

## Return the Kronecker Product
kron(P, diag(3))
P
```

ks2Test

*Two Sample Kolmogorov–Smirnov Test***Description**

Tests if two series are distributional equivalent.

Usage

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

Arguments

`x`, `y` numeric vectors of data values.
`title` an optional title string, if not specified the inputs data name is deparsed.
`description` optional description string, or a vector of character strings.

Details

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.

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:

`@call` the function call.
`@data` the data as specified by the input argument(s).
`@test` a list whose elements contain the results from the statistical test. The information provided is similar to a list object of class `"htest"`.
`@title` a character string with the name of the test. This can be overwritten specifying a user defined input argument.

`@description` 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.

Author(s)

R-core team for hypothesis tests implemented from R's package `ctest`.

References

Conover, W. J. (1971); *Practical nonparametric statistics*, New York: John Wiley & Sons.

Lehmann E.L. (1986); *Testing Statistical Hypotheses*, John Wiley and Sons, New York.

Examples

```
## rnorm -
# Generate Series:
x = rnorm(50)
y = rnorm(50)

## ks2Test -
ks2Test(x, y)
```

lcg

Generator for Portable Random Innovations

Description

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

df	number of degrees of freedom, a positive integer, maybe non-integer.
mean, sd	means and standard deviation of the normal distributed innovations.
min, max	lower and upper limits of the uniform distributed innovations.
seed	an integer value, the random number seed.
n	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 congruential 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`.

References

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

Examples

```
## 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)
```

linearInterp *Bivariate Linear Interpolation*

Description

Bivariate Linear Interpolation. Two options are available gridded and pointwise interpolation.

Usage

```
linearInterp(x, y = NULL, z = NULL, gridPoints = 21,
             xo = seq(min(x), max(x), length = gridPoints),
             yo = seq(min(y), max(y), length = gridPoints))
```

```
linearInterpp(x, y = NULL, z = NULL, xo, yo)
```

Arguments

<code>x</code> , <code>y</code> , <code>z</code>	for <code>linearInterp</code> the arguments <code>x</code> and <code>y</code> are two numeric vectors of grid points, and <code>z</code> is a numeric matrix or any other rectangular object which can be transformed by the function <code>as.matrix</code> into a matrix object. For <code>linearInterpp</code> we consider either three numeric vectors of equal length or if <code>y</code> and <code>z</code> are <code>NULL</code> , a list with entries <code>x</code> , <code>y</code> , <code>z</code> , or named data frame with <code>x</code> in the first, <code>y</code> in the second, and <code>z</code> in the third column.
<code>gridPoints</code>	an integer value specifying the number of grid points in <code>x</code> and <code>y</code> direction.
<code>xo</code> , <code>yo</code>	for <code>linearInterp</code> two numeric vectors of data points spanning the grid, and for <code>linearInterpp</code> two numeric vectors of data points building pairs for pointwise interpolation.

Value

`linearInterp`

returns a list with at least three entries, `x`, `y` and `z`. Note, that the returned values, can be directly used by the `persp` and `contour` 3D plotting methods.

`linearInterpp`

returns a `data.frame` with columns "`x`", "`y`", and "`z`".

See Also

[akimaInterp](#), and [krigeInterp](#).

Examples

```
## linearInterp -
# Linear Interpolation:
if (require(akima)) {
  set.seed(1953)
```

```
x = runif(999) - 0.5
y = runif(999) - 0.5
z = cos(2*pi*(x^2+y^2))
ans = linearInterp(x, y, z, gridPoints = 41)
persp(ans, theta = -40, phi = 30, col = "steelblue",
      xlab = "x", ylab = "y", zlab = "z")
contour(ans)
}
```

listDescription *Description File Listing*

Description

Lists the content of a description file.

Usage

```
listDescription(package, character.only = FALSE)
```

Arguments

`package` a literal character or character string denoting the name of the package to be listed.

`character.only` a logical indicating whether 'package' can be assumed to be character strings.

Value

prints the description file.

See Also

[listFunctions](#), [listIndex](#).

Examples

```
## listDescription -
listDescription("fBasics")
```

listFunctions *Functions Listing*

Description

Lists and counts functions from packages.

Usage

```
listFunctions(package, character.only = FALSE)
countFunctions(package, character.only = FALSE)
```

Arguments

package a literal character or a character string denoting the name of the package to be listed.

character.only a logical indicating whether 'package' can be assumed to be character strings.

Value

prints a list and counts of functions.

See Also

[listFunctions](#), [listIndex](#).

Examples

```
## listFunctions -
listFunctions("fBasics")

## countFunctions -
countFunctions("fBasics")
```

listIndex *Index File Listing*

Description

Lists the content of an index file.

Usage

```
listIndex(package, character.only = FALSE)
```

Arguments

`package` a literal character string or a character string denoting the name of the package to be listed.

`character.only` a logical indicating whether 'package' can be assumed to be character strings.

Value

prints the index file.

See Also

[listDescription](#), [listIndex](#).

Examples

```
## listIndex -
listIndex("fBasics")
```

<code>locationTest</code>	<i>Two Sample Location Tests</i>
---------------------------	----------------------------------

Description

Tests if two series differ in their distributional location parameter.

Usage

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

Arguments

`x, y` numeric vectors of data values.

`method` a character string naming which test should be applied.

`title` an optional title string, if not specified the inputs data name is deparsed.

`description` optional description string, or a vector of character strings.

Details

The `method="t"` 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 `method="kw2"` 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.

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:

@call	the function call.
@data	the data as specified by the input argument(s).
@test	a list whose elements contain the results from the statistical test. The information provided is similar to a list object of class "htest".
@title	a character string with the name of the test. This can be overwritten specifying a user defined input argument.
@description	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.

Note

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

Author(s)

R-core team for hypothesis tests implemented from R's package `ctest`.

References

Conover, W. J. (1971); *Practical nonparametric statistics*, New York: John Wiley & Sons.
 Lehmann E.L. (1986); *Testing Statistical Hypotheses*, John Wiley and Sons, New York.

Examples

```
## rnorm -
# Generate Series:
x = rnorm(50)
y = rnorm(50)

## locationTest -
locationTest(x, y, "t")
locationTest(x, y, "kw2")
```

 maxdd *Drawdown Statistics*

Description

This is a collection and description of functions which compute drawdown statistics. Included are density, distribution function, and random generation for the maximum drawdown distribution. In addition the expectation of drawdowns for Brownian motion can be computed.

The functions are:

dmaxdd	the Density function,
pmaxdd	the Distribution function,
rmaxdd	the random number generator,
maxddStats	the expectation of drawdowns.

Usage

```
dmaxdd(x, sd = 1, horizon = 100, N = 1000)
pmaxdd(q, sd = 1, horizon = 100, N = 1000)
rmaxdd(n, mean = 0, sd = 1, horizon = 100)

maxddStats(mean = 0, sd = 1, horizon = 1000)
```

Arguments

x, q	a numeric vector of quantiles.
n	an integer value, the number of observations.
mean, sd	two numeric values, the mean and standard deviation.
horizon	an integer value, the (run time) horizon of the investor.
N	an integer value, the precession index for summations. Before you change this value please inspect Magdon-Ismail et. al. (2003).

Value

dmaxdd
returns for a trendless Brownian process mean=0 and standard deviation "sd" the density from the probability that the maximum drawdown "D" is larger or equal to "h" in the interval [0,T], where "T" denotes the time horizon of the investor.

pmaxdd
returns for a trendless Brownian process mean=0 and standard deviation "sd" the the probability that the maximum drawdown "D" is larger or equal to "h" in the interval [0,T], where "T" denotes the time horizon of the investor.

`rmaxdd`
returns for a Brownian Motion process with mean `mean` and standard deviation `sd` random variates of maximum drawdowns.

`maxddStats`
returns the expectation Value $E[D]$ of maximum drawdowns of Brownian Motion for a given drift mean, variance `sd`, and runtime horizon of the Brownian Motion process.

Note

Currently, for the functions `dmaxdd` and `pmaxdd` only the trend or driftless case is implemented.

References

Magdon-Ismail M., Atiya A.F., Pratap A., Abu-Mostafa Y.S. (2003); *On the Maximum Drawdown of a Brownian Motion*, Preprint, CalTech, Pasadena USA, p. 24.

Examples

```
## rmaxdd -
# Set a random seed
set.seed(1953)
# horizon of the investor, time T
horizon = 1000
# number of MC samples, N -> infinity
samples = 1000
# Range of expected Drawdowns
xlim = c(0, 5)*sqrt(horizon)
# Plot Histogram of Simulated Max Drawdowns:
r = rmaxdd(n = samples, mean = 0, sd = 1, horizon = horizon)
hist(x = r, n = 40, probability = TRUE, xlim = xlim,
     col = "steelblue4", border = "white", main = "Max. Drawdown Density")
points(r, rep(0, samples), pch = 20, col = "orange", cex = 0.7)

## dmaxdd -
x = seq(0, xlim[2], length = 200)
d = dmaxdd(x = x, sd = 1, horizon = horizon, N = 1000)
lines(x, d, lwd = 2)

## pmaxdd -
# Count Frequencies of Drawdowns Greater or Equal to "h":
n = 50
x = seq(0, xlim[2], length = n)
g = rep(0, times = n)
for (i in 1:n) g[i] = length(r[r > x[i]]) / samples
plot(x, g, type = "h", lwd = 3,
     xlab = "q", main = "Max. Drawdown Probability")
# Compare with True Probability "G_D(h)":
x = seq(0, xlim[2], length = 5*n)
p = pmaxdd(q = x, sd = 1, horizon = horizon, N = 5000)
lines(x, p, lwd = 2, col="steelblue4")

## maxddStats -
```

```
# Compute expectation Value E[D]:
maxddStats(mean = -0.5, sd = 1, horizon = 10^(1:4))
maxddStats(mean = 0.0, sd = 1, horizon = 10^(1:4))
maxddStats(mean = 0.5, sd = 1, horizon = 10^(1:4))
```

nig

Normal Inverse Gaussian Distribution

Description

Density, distribution function, quantile function and random generation for the normal inverse Gaussian distribution.

Usage

```
dnig(x, alpha = 1, beta = 0, delta = 1, mu = 0, log = FALSE)
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)
```

Arguments

alpha, beta, delta, mu	shape parameter alpha; skewness parameter beta, $\text{abs}(\text{beta})$ is in the range (0, alpha); scale parameter delta, delta must be zero or positive; location parameter mu, by default 0. These are the parameters in the first parameterization.
log	a logical flag by default FALSE. Should labels and a main title drawn to the plot?
n	number of observations.
p	a numeric vector of probabilities.
x, q	a numeric vector of quantiles.

Details

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

Value

All values for the `*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.

Author(s)

David Scott for code implemented from R's contributed package `HyperbolicDist`.

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

```
## 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 = "steelblue")
  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 = "steelblue")
  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)
```

nigFit

Fit of a Normal Inverse Gaussian Distribution

Description

Estimates the parameters of a normal inverse Gaussian distribution.

Usage

```
nigFit(x, alpha = 1, beta = 0, delta = 1, mu = 0,
       method = c("mle", "gmm", "mps"), scale = TRUE, doplot = TRUE,
       span = "auto", trace = TRUE, title = NULL, description = NULL, ...)
```

Arguments

<code>alpha</code> , <code>beta</code> , <code>delta</code> , <code>mu</code>	The parameters are <code>alpha</code> , <code>beta</code> , <code>delta</code> , and <code>mu</code> : shape parameter <code>alpha</code> ; skewness parameter <code>beta</code> , <code>abs(beta)</code> is in the range (0, <code>alpha</code>); scale parameter <code>delta</code> , <code>delta</code> must be zero or positive; location parameter <code>mu</code> , by default 0. These is the meaning of the parameters in the first parameterization <code>pm=1</code> which is the default parameterization selection. In the second parameterization, <code>pm=2</code> <code>alpha</code> and <code>beta</code> take the meaning of the shape parameters (usually named) <code>zeta</code> and <code>rho</code> . In the third parameterization, <code>pm=3</code> <code>alpha</code> and <code>beta</code> take the meaning of the shape parameters (usually named) <code>xi</code> and <code>chi</code> . In the fourth parameterization, <code>pm=4</code> <code>alpha</code> and <code>beta</code> take the meaning of the shape parameters (usually named) <code>a.bar</code> and <code>b.bar</code> .
<code>description</code>	a character string which allows for a brief description.
<code>doplot</code>	a logical flag. Should a plot be displayed?
<code>method</code>	a character string. Either "mle", Maximum Likelihood Estimation, the default, "gmm" Generalized Method of Moments Estimation, or "mps" Maximum Product Spacings Estimation.
<code>scale</code>	a logical flag, by default TRUE. Should the time series be scaled by its standard deviation to achieve a more stable optimization?
<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>	a logical flag. Should the parameter estimation process be traced?
<code>x</code>	a numeric vector.
<code>...</code>	parameters to be parsed.

Value

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

<code>estimate</code>	the point at which the maximum value of the log likelihood function is obtained.
<code>minimum</code>	the value of the estimated maximum, i.e. the value of the log likelihood function.
<code>code</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.
<code>gradient</code>	the gradient at the estimated maximum.
<code>steps</code>	number of function calls.

Examples

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

## nigFit -
# Fit Parameters:
nigFit(s, alpha = 1, beta = 0, delta = 1, mu = mean(s), doplot = TRUE)
```

nigShapeTriangle *NIG Shape Triangle*

Description

Plots the normal inverse Gaussian Shape Triangle.

Usage

```
nigShapeTriangle(object, add = FALSE, labels = TRUE, ...)
```

Arguments

object	an object of class "fDISTFIT" as returned by the function nigFit.
add	a logical value. Should another point added to the NIG shape triangle? By default FALSE, a new plot will be created.
labels	a logical flag by default TRUE. Should the logarithm of the density be returned?
...	arguments to be passed to the function integrate.

Value

displays the parameters of fitted distributions in the NIG shape triangle.

Author(s)

David Scott for code implemented from R's contributed package HyperbolicDist.

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

```
## nigShapeTriangle -
#
```

```
nigSlider          nigerbolic Distribution Slider
```

Description

Displays interactively the dependence of the nigerbolic distribution on its parameters.

Usage

```
nigSlider()
```

Value

a tcl/tk based graphical user interface.

This is a nice display for educational purposes to investigate the densities and probabilities of the invetrse Gaussian distribution.

Examples

```
## nigSlider -
# nigSlider()
```

```
norm          Matrix Norm
```

Description

Returns the norm of a matrix.

Usage

```
norm(x, p = 2)
```

Arguments

x	a numeric matrix.
p	an integer value, 1, 2 or Inf. p=1 - The maximum absolute column sum norm which is defined as the maximum of the sum of the absolute valued elements of columns of the matrix. p=2 - The spectral norm is "the norm" of a matrix X. This value is computed as the square root of the maximum eigenvalue of CX where C is the conjugate transpose. p=Inf - The maximum absolute row sum norm is defined as the maximum of the sum of the absolute valued elements of rows of the matrix.

Details

The function `norm` computes the norm of a matrix. Three choices are possible:

`p=1` - The maximum absolute column sum norm which is defined as the maximum of the sum of the absolute valued elements of columns of the matrix.

`p=2` - The spectral norm is "the norm" of a matrix X . This value is computed as the square root of the maximum eigenvalue of CX where C is the conjugate transpose.

`p=Inf` - The maximum absolute row sum norm is defined as the maximum of the sum of the absolute valued elements of rows of the matrix.

References

Golub, van Loan, (1996); *Matrix Computations*, 3rd edition. Johns Hopkins University Press.

Examples

```
## Create Pascal Matrix:
P = pascal(5)
P

## Return the Norm of the Matrix:
norm(P)
```

NormalityTests *Normality 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:

`jbTest` 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,

```

lillieTest  Lilliefors (Kolmogorov-Smirnov) normality test,
pchiTest   Pearson chi-square normality test,
sfTest     Shapiro-Francia normality test.

```

For SPlus/Finmetrics Compatibility:

```
normalTest  test suite for some normality tests.
```

Usage

```

ksnormTest(x, title = NULL, description = NULL)

jbTest(x, title = NULL, description = NULL)
shapiroTest(x, title = NULL, description = NULL)
normalTest(x, method = c("sw", "jb"), na.rm = FALSE)

jarqueberaTest(x, title = NULL, description = NULL)
dagoTest(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)

```

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".
na.rm        [normalTest] -
             a logical value. Should missing values removed before computing the tests? The
             default value is FALSE.
title        an optional title string, if not specified the inputs data name is deparsed.
x            a numeric vector of data values or a S4 object of class timeSeries.

```

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`.

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.
<code>data.name</code>	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

```
## Series:
x = rnorm(100)

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

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

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

pascal

Pascal Matrix

Description

Creates a Pascal matrix.

Usage

```
pascal(n)
```

Arguments

n an integer value, the dimension of the square matrix.

Details

The function `pascal` generates a Pascal matrix of order `n` which is a symmetric, positive, definite matrix with integer entries made up from Pascal's triangle. The determinant of a Pascal matrix is 1. The inverse of a Pascal matrix has integer entries. If `lambda` is an eigenvalue of a Pascal matrix, then `1/lambda` is also an eigenvalue of the matrix. Pascal matrices are ill-conditioned.

References

Call G.S., Velleman D.J., (1993); *Pascal's matrices*, American Mathematical Monthly 100, 372–376.

Edelman A., Strang G., (2004); *Pascal Matrices*, American Mathematical Monthly 111, 361–385.

Examples

```
## Create Pascal Matrix:
P = pascal(5)
P

## Determinant
det(pascal(5))
det(pascal(10))
det(pascal(15))
det(pascal(20))
```

pdl

Polynomial Distributed Lags

Description

Returns a regressor matrix for polynomial distributed lags.

Usage

```
pdl(x, d = 2, q = 3, trim = FALSE)
```

Arguments

<code>x</code>	a numeric vector.
<code>d</code>	an integer specifying the order of the polynomial.
<code>q</code>	an integer specifying the number of lags to use in creating polynomial distributed lags. This must be greater than <code>d</code> .
<code>trim</code>	a logical flag; if TRUE, the missing values at the beginning of the returned matrix will be trimmed.

See Also

[tslag](#).

Examples

```
## pd1 -  
#
```

positiveDefinite *Positive Definite Matrixes*

Description

Checks if a matrix is positive definite and/or forces a matrix to be positive definite.

Usage

```
isPositiveDefinite(x)  
makePositiveDefinite(x)
```

Arguments

x a square numeric matrix.

Details

The function `isPositiveDefinite` checks if a square matrix is positive definite.

The function `makePositiveDefinite` forces a matrix to be positive definite.

Author(s)

Korbinian Strimmer.

Examples

```
## isPositiveDefinite -  
# the 3x3 Pascal Matrix is positive definite  
isPositiveDefinite(pascal(3))
```

print *Print Control*

Description

Unlists and prints a control object.

Usage

```
## S3 method for class 'control':  
print(x, ...)
```

Arguments

x the object to be printed.
... arguments to be passed.

Value

```
print.control  
prints control.
```

Examples

```
## print -  
control = list(n = 211, seed = 54, name = "generator")  
print(control)  
class(control) = "control"  
print(control)
```

QuantileQuantilePlots
Quantile-Quantile Plots

Description

Returns quantile-quantile plots for the normal, the normal inverse Gaussian, and the generalized hyperbolic Student-t distribution.

List of Functions:

qqnormPlot	Returns a tailored Normal quantile-quantile plot,
qqnigPlot	Returns a tailored NIG quantile-quantile plot,
qqghtPlot	Returns a tailored GHT quantile-quantile plot.

Usage

```
qqnormPlot(x, labels = TRUE, col = "steelblue", pch = 19,
           title = TRUE, mtext = TRUE, grid = FALSE, rug = TRUE,
           scale = TRUE, ...)
qqnigPlot(x, labels = TRUE, col = "steelblue", pch = 19,
          title = TRUE, mtext = TRUE, grid = FALSE, rug = TRUE,
          scale = TRUE, ...)
qqghtPlot(x, labels = TRUE, col = "steelblue", pch = 19,
          title = TRUE, mtext = TRUE, grid = FALSE, rug = TRUE,
          scale = TRUE, ...)
```

Arguments

<code>x</code>	an object of class "timeSeries" or any other object which can be transformed by the function <code>as.timeSeries</code> into an object of class <code>timeSeries</code> . The latter case, other than <code>timeSeries</code> objects, is more or less untested.
<code>labels</code>	a logical flag, should the plot be returned with default labels and decorated in an automated way? By default <code>TRUE</code> .
<code>col</code>	the color for the series. In the univariate case use just a color name like the default, <code>col="steelblue"</code> , in the multivariate case we recommend to select the colors from a color palette, e.g. <code>col=heat.colors(ncol(x))</code> .
<code>pch</code>	an integer value, by default 19. Which plot character should be used in the plot?
<code>title</code>	a logical flag, by default <code>TRUE</code> . Should a default title added to the plot?
<code>mtext</code>	a logical flag, by default <code>TRUE</code> . Should a marginal text be printed on the third site of the graph?
<code>grid</code>	a logical flag, should a grid be added to the plot? By default <code>TRUE</code> . To plot a horizontal lines only use <code>grid="h"</code> and for vertical lines use <code>grid="v"</code> , respectively.
<code>rug</code>	a logical flag, by default <code>TRUE</code> . Should a rug representation of the data added to the plot?
<code>scale</code>	a logical flag, by default <code>TRUE</code> . Should the time series be scaled for the investigation?
<code>...</code>	optional arguments to be passed.

Value

displays a time series plot.

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

Examples

```
## qqPlot -
```

returnSeriesGUI *Return Series Plots*

Description

A graphical user interface to display financial time series plots.

List of Functions:

returnSeriesGUI Opens a GUI for return series plots.

Usage

```
returnSeriesGUI(x)
```

Arguments

x an object of class "timeSeries" or any other object which can be transformed by the function `as.timeSeries` into an object of class `timeSeries`. The latter case, other than `timeSeries` objects, is more or less untested.

Value

```
returnSeriesGUI
```

For the `returnSeriesGUI` function, beside the graphical user interface no values are returned.

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

Examples

```
##
```

rk *The Rank of a Matrix*

Description

Returns the rank of a matrix.

Usage

```
rk(x, method = c("qr", "chol"))
```

Arguments

`x` a numeric matrix.

`method` a character string. For `method = "qr"` the rank is computed as `qr(x)$rank`, or alternatively for `method="chol"` the rank is computed as `attr(chol(x, pivot=TRUE), "rank")`.

Details

The function `rk` computes the rank of a matrix which is the dimension of the range of the matrix corresponding to the number of linearly independent rows or columns of the matrix, or to the number of nonzero singular values.

The rank of a matrix is also named `inear map`.

References

Golub, van Loan, (1996); *Matrix Computations*, 3rd edition. Johns Hopkins University Press.

Examples

```
## Create Pascal Matrix:
P = pascal(5)
P

## Compute the Rank:
rk(P)
rk(P, "chol")
```

rowStats

Row Statistics

Description

Functions to compute row statistical properties of financial and economic time series data.

The functions are:

<code>rowStats</code>	calculates row statistics,
<code>rowSds</code>	calculates row standard deviations,
<code>rowVars</code>	calculates row variances,
<code>rowSkewness</code>	calculates row skewness,
<code>rowKurtosis</code>	calculates row kurtosis,
<code>rowMaxs</code>	calculates maximum values in each row,
<code>rowMins</code>	calculates minimum values in each row,
<code>rowProds</code>	computes product of all values in each row,
<code>rowQuantiles</code>	computes quantiles of each row.

Usage

```
rowStats(x, FUN, ...)  
  
rowSds(x, ...)  
rowVars(x, ...)  
rowSkewness(x, ...)  
rowKurtosis(x, ...)  
rowMaxs(x, ...)  
rowMins(x, ...)  
rowProds(x, ...)  
rowQuantiles(x, prob = 0.05, ...)  
  
rowStdevs(x, ...)  
rowAvg(x, ...)
```

Arguments

<code>FUN</code>	a function name. The statistical function to be applied.
<code>prob</code>	a numeric value, the probability with value in [0,1].
<code>x</code>	a rectangular object which can be transformed into a matrix by the function <code>as.matrix</code> .
<code>...</code>	arguments to be passed.

Value

the functions return a numeric vector of the statistics.

See Also

```
link{colStats}.
```

Examples

```
## Simulated Return Data in Matrix Form:  
x = matrix(rnorm(10*10), nrow = 10)  
  
## rowStats -  
rowStats(x, FUN = mean)  
  
## rowMaxs -  
rowMaxs(x)
```

 scaleTest

Two Sample Scale Tests

Description

Tests if two series differ in their distributional scale parameter.

Usage

```
scaleTest(x, y, method = c("ansari", "mood"),
          title = NULL, description = NULL)
```

Arguments

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

Details

The `method="ansari"` performs the Ansari–Bradley two–sample test for a difference in scale parameters. The test returns for any sizes of the series x and y the exact p value together with its asymptotic limit.

The `method="mood"`, 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 $s=l$.

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 "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.
<code>statistic</code>	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.

Note

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

Author(s)

R-core team for hypothesis tests implemented from R's package `ctest`.

References

- Conover, W. J. (1971); *Practical nonparametric statistics*, New York: John Wiley & Sons.
- 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

```
## rnorm -
# Generate Series:
x = rnorm(50)
y = rnorm(50)

## scaleTest -
scaleTest(x, y, "ansari")
scaleTest(x, y, "mood")
```

ScalingLawPlot *Scaling Law Behaviour*

Description

Evaluates the scaling exponent of a financial return series and plots the scaling law.

Usage

```
scalinglawPlot(x, span = ceiling(log(length(x)/252)/log(2)), doplot = TRUE,
  labels = TRUE, trace = TRUE, ...)
```

Arguments

<code>doplot</code>	a logical value. Should a plot be displayed?
<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>span</code>	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>trace</code>	a logical value. Should the computation be traced?
<code>x</code>	an uni- or multivariate return series of class <code>timeSeries</code> or any other object which can be transformed by the function <code>as.timeSeries()</code> into an object of class <code>timeSeries</code> .
<code>...</code>	arguments to be passed.

Details**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.

Value

returns a list with the following components: `Intercept`, `Exponent` the scaling exponent, and `InverseExponent` its inverse value.

Author(s)

Diethelm Wuertz for the Rmetrics R-port.

References

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

Examples

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

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

sgh

*Standardized Generalized Hyperbolic Distribution***Description**

Density, distribution function, quantile function and random generation for the standardized generalized hyperbolic distribution.

Usage

```
dsgh(x, zeta = 1, rho = 0, lambda = 1, log = FALSE)
psgh(q, zeta = 1, rho = 0, lambda = 1)
qsgh(p, zeta = 1, rho = 0, lambda = 1)
rsgh(n, zeta = 1, rho = 0, lambda = 1)
```

Arguments

<code>zeta</code> , <code>rho</code> , <code>lambda</code>	shape parameter <code>zeta</code> is positive, skewness parameter <code>rho</code> is in the range (-1, 1).
<code>log</code>	a logical flag by default <code>FALSE</code> . If <code>TRUE</code> , log values are returned.
<code>n</code>	number of observations.
<code>p</code>	a numeric vector of probabilities.
<code>x</code> , <code>q</code>	a numeric vector of quantiles.

Details

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

Value

All values for the `*sgh` 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.

Author(s)

Diethelm Wuertz.

Examples

```
## rsgh -
set.seed(1953)
r = rsgh(5000, zeta = 1, rho = 0.5, lambda = 1)
plot(r, type = "l", col = "steelblue",
     main = "gh: zeta=1 rho=0.5 lambda=1")
```

```
## dsgh -
# Plot empirical density and compare with true density:
hist(r, n = 50, probability = TRUE, border = "white", col = "steelblue",
     ylim = c(0, 0.6))
x = seq(-5, 5, length = 501)
lines(x, dsgh(x, zeta = 1, rho = 0.5, lambda = 1))

## psgh -
# Plot df and compare with true df:
plot(sort(r), (1:5000/5000), main = "Probability", col = "steelblue")
lines(x, psgh(x, zeta = 1, rho = 0.5, lambda = 1))

## qsgh -
# Compute Quantiles:
round(qsgh(psgl(seq(-5, 5, 1), zeta = 1, rho = 0.5), zeta = 1, rho = 0.5), 4)
```

sghFit

*Standardized GH Distribution Fit***Description**

Estimates the distributional parameters for a standardized generalized hyperbolic distribution.

Usage

```
sghFit(x, zeta = 1, rho = 0, lambda = 1, include.lambda = TRUE,
       scale = TRUE, doplot = TRUE, span = "auto", trace = TRUE,
       title = NULL, description = NULL, ...)
```

Arguments

<code>x</code>	a numeric vector.
<code>zeta, rho, lambda</code>	shape parameter <code>zeta</code> is positive, skewness parameter <code>rho</code> is in the range (-1, 1). and index parameter <code>lambda</code> , by default 1.
<code>include.lambda</code>	a logical flag, by default TRUE. Should the index parameter <code>lambda</code> included in the parameter estimate?
<code>scale</code>	a logical flag, by default TRUE. Should the time series be scaled by its standard deviation to achieve a more stable optimization?
<code>doplot</code>	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>trace</code>	a logical flag. Should the parameter estimation process be traced?

title a character string which allows for a project title.
 description a character string which allows for a brief description.
 ... parameters to be parsed.

Value

returns 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 `estimate`. Either `estimate`
 is an approximate local minimum of the function or `steptol` is too small;
 4: iteration limit exceeded;
 5: maximum step size `stepmax` exceeded five consecutive times. Either the
 function is unbounded below, becomes asymptotic to a finite value from above
 in some direction or `stepmax` is too small.
 gradient the gradient at the estimated maximum.
 steps number of function calls.

Examples

```
## sghFit -
# Simulate Random Variates:
set.seed(1953)
s = rsgn(n = 2000, zeta = 0.7, rho = 0.5, lambda = 0)

## sghFit -
# Fit Parameters:
sghFit(s, zeta = 1, rho = 0, lambda = 1, include.lambda = TRUE,
       doplot = TRUE)
```

snig

Standardized Normal Inverse Gaussian Distribution

Description

Density, distribution function, quantile function and random generation for the standardized normal inverse Gaussian distribution.

Usage

```
dsnig(x, zeta = 1, rho = 0, log = FALSE)
psnig(q, zeta = 1, rho = 0)
qsnig(p, zeta = 1, rho = 0)
rsnig(n, zeta = 1, rho = 0)
```

Arguments

<code>zeta, rho</code>	shape parameter <code>zeta</code> is positive, skewness parameter <code>rho</code> is in the range (-1, 1).
<code>log</code>	a logical flag by default <code>FALSE</code> . If <code>TRUE</code> , log values are returned.
<code>n</code>	number of observations.
<code>p</code>	a numeric vector of probabilities.
<code>x, q</code>	a numeric vector of quantiles.

Details

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

Value

All values for the `*snig` 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.

Author(s)

Diethelm Wuertz.

Examples

```
## snig -
set.seed(1953)
r = rsnig(5000, zeta = 1, rho = 0.5)
plot(r, type = "l", col = "steelblue",
     main = "snig: zeta=1 rho=0.5")

## snig -
# Plot empirical density and compare with true density:
hist(r, n = 50, probability = TRUE, border = "white", col = "steelblue")
x = seq(-5, 5, length = 501)
lines(x, dsnig(x, zeta = 1, rho = 0.5))

## snig -
# Plot df and compare with true df:
plot(sort(r), (1:5000/5000), main = "Probability", col = "steelblue")
lines(x, psnig(x, zeta = 1, rho = 0.5))

## snig -
# Compute Quantiles:
qsnig(psnig(seq(-5, 5, 1), zeta = 1, rho = 0.5), zeta = 1, rho = 0.5)
```

 snigFit

Fit of a Standardized NIG Distribution

Description

Estimates the parameters of a standardized normal inverse Gaussian distribution.

Usage

```
snigFit(x, zeta = 1, rho = 0, scale = TRUE, dplot = TRUE,
        span = "auto", trace = TRUE, title = NULL, description = NULL, ...)
```

Arguments

zeta, rho	shape parameter zeta is positive, skewness parameter rho is in the range (-1, 1).
description	a character string which allows for a brief description.
dplot	a logical flag. Should a plot be displayed?
scale	a logical flag, by default TRUE. Should the time series be scaled by its standard deviation to achieve a more stable optimization?
span	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 min and max are the left and right endpoints of the range, and n gives the number of the intermediate points.
title	a character string which allows for a project title.
trace	a logical flag. Should the parameter estimation process be traced?
x	a numeric vector.
...	parameters to be parsed.

Value

The function `snigFit` returns 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.

Examples

```
## snigFit -
# Simulate Random Variates:
set.seed(1953)
s = rsnig(n = 2000, zeta = 0.7, rho = 0.5)

## snigFit -
# Fit Parameters:
snigFit(s, zeta = 1, rho = 0, doplot = TRUE)
```

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.

The functions are:

[dpqr]stable	the skewed stable distribution,
stableMode	the stable mode,
stableSlider	interactive stable distribution display.

Usage

```
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)

stableSlider()
```

Arguments

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

n
 number of observations, an integer value.

p
 a numeric vector of probabilities.

`pm` parameterization, an integer value by default `pm=0`, the 'S0' parameterization.
`x, q` a numeric vector of quantiles.

Details

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 `*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 function `stableSlider` displays for educational purposes the densities and probabilities of the skew stable distribution.

Author(s)

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.

Samorodnitsky 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 University of Wroclaw, 13 pages.

Examples

```
## 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)
```

symbolTable

Table of Symbols

Description

Displays a Table of plot characters and symbols.

Usage

```
symbolTable(font = par('font'), cex = 0.7)
```

Arguments

cex a numeric value, determines the character size, the default size is 0.7.
font an integer value, the number of the font, by default font number 1.

Value

```
symbolTable
```

displays a table with the plot characters and symbols numbered from 0 to 255 and returns invisible the name of the font.

See Also

`link{characterTable}, link{colorTable}`.

Examples

```
## symbolTable -
# Default Symbol Table:
symbolTable()
```

TimeSeriesPlots *Financial Time Series Plots*

Description

Returns an index/price, a return, or a drawdown plot.

List of Functions:

<code>seriesPlot</code>	Returns a tailored return series plot,
<code>cumulatedPlot</code>	Displays a cumulated series given the returns,
<code>returnPlot</code>	Displays returns given the cumulated series,
<code>drawdownPlot</code>	Displays drawdowns given the return series.

Usage

```
seriesPlot(x, labels = TRUE, type = "l", col = "steelblue",
           title = TRUE, grid = TRUE, box = TRUE, rug = TRUE, ...)
cumulatedPlot(x, index = 100, labels = TRUE, type = "l", col = "steelblue",
              title = TRUE, grid = TRUE, box = TRUE, rug = TRUE, ...)
returnPlot(x, labels = TRUE, type = "l", col = "steelblue",
           title = TRUE, grid = TRUE, box = TRUE, rug = TRUE, ...)
drawdownPlot(x, labels = TRUE, type = "l", col = "steelblue",
             title = TRUE, grid = TRUE, box = TRUE, rug = TRUE, ...)
```

Arguments

<code>box</code>	a logical flag, should a box be added to the plot? By default TRUE.
<code>col</code>	the color for the series. In the univariate case use just a color name like the default, <code>col="steelblue"</code> , in the multivariate case we recommend to select the colors from a color palette, e.g. <code>col=heat.colors(ncol(x))</code> .
<code>grid</code>	a logical flag, should a grid be added to the plot? By default TRUE.
<code>index</code>	a numeric value, by default 100. The function cumulates column by column the returns and multiplies the result with the index value: <code>index*exp(colCumsums(x))</code> .
<code>labels</code>	a logical flag, should the plot be returned with default labels and decorated in an automated way? By default TRUE.

<code>rug</code>	a logical flag, by default TRUE. Should a rug representation of the data added to the plot?
<code>title</code>	a logical flag, by default TRUE. Should a default title added to the plot?
<code>type</code>	what type of plot should be drawn? By default we use a line plot, <code>type="l"</code> . An alternative plot style which produces nice figures is for example <code>type="h"</code> .
<code>x</code>	an object of class "timeSeries" or any other object which can be transformed by the function <code>as.timeSeries</code> into an object of class <code>timeSeries</code> . The latter case, other than <code>timeSeries</code> objects, is more or less untested.
<code>...</code>	optional arguments to be passed.

Details

The plot functions can be used to plot univariate and multivariate time series of class `timeSeries`.

The graphical parameters `type` and `col` can be set by the values specified through the argument list. In the case of multivariate time series `col` can be specified by the values returned by a color palette.

Automated titles including main title, x- and y-labels, grid lines, box style and rug representations can be selected by setting these arguments to TRUE which is the default. If the title flag is unset, then the main title, x-, and y-labels are empty strings. This allows to set user defined labels with the function `title` after the plot is drawn.

Beside `type`, `col`, `main`, `xlab` and `ylab`, all other `par` arguments can be passed to the plot function.

If the `labels` flag is unset to FALSE, then no decorations will be added to the plot, and the plot can be fully decorated by the user.

Value

displays a time series plot.

Examples

```
## seriesPlot -
  tS = as.timeSeries(data(LPP2005REC))
  seriesPlot(tS)
```

| `tr` | *Trace of a Matrix* |

Description

Returns trace of a matrix.

Usage

```
tr(x)
```

Arguments

`x` a numeric matrix.

Details

The function `tr` computes the trace of a square matrix which is the sum of the diagonal elements of the matrix under consideration.

References

Golub, van Loan, (1996); *Matrix Computations*, 3rd edition. Johns Hopkins University Press.

Examples

```
## Create Pascal Matrix:
P = pascal(3)
P

## Trace:
tr(P)
```

| `triang` | *Upper and Lower Triangular Matrixes* |

Description

Extracts the upper or lower tridiagonal part from a matrix.

Usage

```
triang(x)
Triang(x)
```

Arguments

`x` a numeric matrix.

Details

The functions `triang` and `Triang` allow to transform a square matrix to a lower or upper triangular form. A triangular matrix is either an upper triangular matrix or lower triangular matrix. For the first case all matrix elements $a[i, j]$ of matrix A are zero for $i > j$, whereas in the second case we have just the opposite situation. A lower triangular matrix is sometimes also called left triangular. In fact, triangular matrices are so useful that much computational linear algebra begins with factoring or decomposing a general matrix or matrices into triangular form. Some matrix factorization methods are the Cholesky factorization and the LU-factorization. Even including the factorization step, enough later operations are typically avoided to yield an overall time savings. Triangular matrices have the following properties: the inverse of a triangular matrix is a triangular matrix, the product of

two triangular matrices is a triangular matrix, the determinant of a triangular matrix is the product of the diagonal elements, the eigenvalues of a triangular matrix are the diagonal elements.

References

Higham, N.J., (2002); *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM.

Golub, van Loan, (1996); *Matrix Computations*, 3rd edition. Johns Hopkins University Press.

Examples

```
## Create Pascal Matrix:
P = pascal(3)
P

## Create lower triangle matrix
L = triang(P)
L
```

tslag

Lagged or Leading Vector/Matrix

Description

Creates a lagged or leading vector/matrix of selected order(s).

Usage

```
tslag(x, k = 1, trim = FALSE)
```

Arguments

k	an integer value, the number of positions the new series is to lag or to lead the input series.
x	a numeric vector or matrix, missing values are allowed.
trim	a logical flag, if TRUE, the missing values at the beginning and/or end of the returned series will be trimmed. The default value is FALSE.

See Also

[pdl](#).

Examples

```
## tslag -
#
```

varianceTest	<i>Two Sample Variance Tests</i>
--------------	----------------------------------

Description

Tests if two series differ in their distributional variance parameter.

Usage

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

Arguments

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

Details

The `method="varf"` 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 `method="bartlett"` performs the Bartlett 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 `method="fligner"` performs the Fligner-Killeen test of the null that the variances in each of the two samples are the same.

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.

`@description` 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.

Note

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

Author(s)

R-core team for hypothesis tests implemented from R's package `ctest`.

References

Conover, W. J. (1971); *Practical nonparametric statistics*, New York: John Wiley & Sons.
 Lehmann E.L. (1986); *Testing Statistical Hypotheses*, John Wiley and Sons, New York.

Examples

```
## rnorm -
# Generate Series:
x = rnorm(50)
y = rnorm(50)

## varianceTest -
varianceTest(x, y, "varf")
varianceTest(x, y, "bartlett")
varianceTest(x, y, "fligner")
```

 vec

Stacking Vectors and Matrixes

Description

Stacks either a lower triangle matrix or a matrix.

Usage

```
vec(x)
vech(x)
```

Arguments

x a numeric matrix.

Details

The function `vec` implements the operator that stacks a matrix as a column vector, to be more precise in a matrix with one column. $\text{vec}(X) = (X_{11}, X_{21}, \dots, X_{N1}, X_{12}, X_{22}, \dots, X_{NN})$.

The function `vech` implements the operator that stacks the lower triangle of a $N \times N$ matrix as an $N(N+1)/2 \times 1$ vector: $\text{vech}(X) = (X_{11}, X_{21}, X_{22}, X_{31}, \dots, X_{NN})$, to be more precise in a matrix with one row.

Examples

```
## Create Pascal Matrix:
P = pascal(3)

## Stack a matrix
vec(P)

## Stack the lower triangle
vech(P)
```

Index

*Topic **distribution**

- DistributionFits, 25
- gh, 30
- ghFit, 32
- ghMode, 33
- ghSlider, 34
- ght, 35
- ghtFit, 36
- hyp, 42
- hypFit, 44
- hypMode, 45
- hypSlider, 46
- maxdd, 59
- nig, 61
- nigFit, 63
- nigShapeTriangle, 65
- nigSlider, 66
- sgh, 82
- sghFit, 83
- snig, 84
- snigFit, 86
- StableDistribution, 87

*Topic **hplot**

- acfPlot, 9
- decor, 24
- gridVector, 37
- interactivePlot, 48
- ScalingLawPlot, 80

*Topic **htest**

- correlationTest, 22
- fHTEST, 27
- ks2Test, 52
- locationTest, 58
- NormalityTests, 67
- scaleTest, 79
- varianceTest, 94

*Topic **math**

- colVec, 22
- hilbert, 40

- Ids, 47
- inv, 49
- kron, 51
- norm, 66
- pascal, 71
- pdl, 72
- positiveDefinite, 73
- rk, 76
- tr, 91
- triang, 92
- tslag, 93
- vec, 95

*Topic **models**

- fBasics-package, 2

*Topic **package**

- fUtilities-package, 5

*Topic **programming**

- akimaInterp, 11
- baseMethods, 13
- BasicStatistics, 14
- BoxPlot, 15
- characterTable, 16
- colorLocator, 17
- colorPalette, 18
- colorTable, 21
- distCheck, 25
- getS4, 28
- Heaviside, 38
- HistogramPlot, 41
- krigeInterp, 50
- lcg, 53
- linearInterp, 54
- listDescription, 56
- listFunctions, 56
- listIndex, 57
- print, 74
- QuantileQuantilePlots, 74
- ReturnSeriesGUI, 76
- symbolTable, 89

- TimeSeriesPlots, 90
- *Topic **univar**
 - rowStats, 77
- acf, 11
- acfPlot, 9
- adTest (*NormalityTests*), 67
- akimaInterp, 11, 50, 55
- akimaInterpp (*akimaInterp*), 11
- baseMethods, 13
- BasicStatistics, 14
- basicStats (*BasicStatistics*), 14
- box_ (*decor*), 24
- Boxcar (*Heaviside*), 38
- boxL (*decor*), 24
- boxPercentilePlot (*BoxPlot*), 15
- BoxPlot, 15
- boxPlot (*BoxPlot*), 15
- characterTable, 16
- cmPalette (*colorPalette*), 18
- colIds (*Ids*), 47
- colIds<- (*Ids*), 47
- colorLocator, 17
- colorMatrix (*colorLocator*), 17
- colorPalette, 18
- colorTable, 21
- colVec, 22
- copyright (*decor*), 24
- correlationTest, 22
- countFunctions (*listFunctions*), 56
- cumulatedPlot (*TimeSeriesPlots*), 90
- cvmTest (*NormalityTests*), 67
- dagoTest (*NormalityTests*), 67
- decor, 24
- Delta (*Heaviside*), 38
- densityPlot (*HistogramPlot*), 41
- dgh (*gh*), 30
- dght (*ght*), 35
- dhyp (*hyp*), 42
- distCheck, 25
- DistributionFits, 25
- divPalette (*colorPalette*), 18
- dmaxdd (*maxdd*), 59
- dnig (*nig*), 61
- drawdownPlot (*TimeSeriesPlots*), 90
- dsgh (*sg*), 82
- dsnig (*snig*), 84
- dstable (*StableDistribution*), 87
- expand.grid, 38
- fBasics (*fBasics-package*), 2
- fBasics-package, 2
- fDISTFIT (*DistributionFits*), 25
- fDISTFIT-class (*DistributionFits*), 25
- fHTEST, 27
- fHTEST-class (*fHTEST*), 27
- focusPalette (*colorPalette*), 18
- fUtilities (*fUtilities-package*), 5
- fUtilities-package, 5
- get.lcgseed (*lcg*), 53
- getCall (*gets4*), 28
- getDescription (*gets4*), 28
- getModel (*gets4*), 28
- gets4, 28
- getSlot (*gets4*), 28
- getTitle (*gets4*), 28
- gh, 30
- ghFit, 32
- ghMode, 33
- ghSlider, 34
- ght, 35
- ghtFit, 36
- greyPalette (*colorPalette*), 18
- gridVector, 37
- heatPalette (*colorPalette*), 18
- Heaviside, 38
- hgrid (*decor*), 24
- hilbert, 40
- HistogramPlot, 41
- histPlot (*HistogramPlot*), 41
- hyp, 42
- hypFit, 44
- hypMode, 45
- hypSlider, 46
- Ids, 47
- interactivePlot, 48
- inv, 49
- isPositiveDefinite (*positiveDefinite*), 73

- jarqueberaTest (*NormalityTests*), 67
- jbtTest (*NormalityTests*), 67
- kendallTest (*correlationTest*), 22
- krigeInterp, 12, 50, 55
- kron, 51
- ks2Test, 52
- ksnormTest (*NormalityTests*), 67
- lacfPlot (*acfPlot*), 9
- lcg, 53
- lillieTest (*NormalityTests*), 67
- linearInterp, 12, 50, 54
- linearInterpp (*linearInterp*), 54
- listDescription, 56, 57
- listFunctions, 56, 56, 57
- listIndex, 56, 57, 57
- locationTest, 24, 58
- logDensityPlot (*HistogramPlot*), 41
- makePositiveDefinite (*positiveDefinite*), 73
- maxdd, 59
- maxddStats (*maxdd*), 59
- monoPalette (*colorPalette*), 18
- nFit (*DistributionFits*), 25
- nig, 61
- nigFit, 63
- nigShapeTriangle, 65
- nigSlider, 66
- nlm, 32, 36, 44
- norm, 66
- NormalityTests, 67
- normalTest (*NormalityTests*), 67
- pacfPlot (*acfPlot*), 9
- pascal, 71
- pchiTest (*NormalityTests*), 67
- pdl, 72, 93
- pearsonTest (*correlationTest*), 22
- pgh (*gh*), 30
- pght (*ght*), 35
- phyp (*hyp*), 42
- pmaxdd (*maxdd*), 59
- pnig (*nig*), 61
- positiveDefinite, 73
- print, 74
- print.fDISTFIT (*DistributionFits*), 25
- psgh (*sg*), 82
- psnig (*snig*), 84
- pstable (*StableDistribution*), 87
- qgh (*gh*), 30
- qght (*ght*), 35
- qhyp (*hyp*), 42
- qnig (*nig*), 61
- qqghtPlot (*QuantileQuantilePlots*), 74
- qqnigPlot (*QuantileQuantilePlots*), 74
- qqnormPlot (*QuantileQuantilePlots*), 74
- qsg (*sg*), 82
- qsnig (*snig*), 84
- qstable (*StableDistribution*), 87
- qualiPalette (*colorPalette*), 18
- QuantileQuantilePlots, 74
- rainbowPalette (*colorPalette*), 18
- Ramp (*Heaviside*), 38
- rampPalette (*colorPalette*), 18
- returnPlot (*TimeSeriesPlots*), 90
- ReturnSeriesGUI, 76
- returnSeriesGUI (*ReturnSeriesGUI*), 76
- rgh (*gh*), 30
- rght (*ght*), 35
- rhyp (*hyp*), 42
- rk, 76
- rmaxdd (*maxdd*), 59
- rnig (*nig*), 61
- rnorm.lcg (*lcg*), 53
- rowAverages (*rowStats*), 77
- rowIds (*Ids*), 47
- rowIds<- (*Ids*), 47
- rowKurtosis (*rowStats*), 77
- rowMaxs (*rowStats*), 77
- rowMins (*rowStats*), 77
- rowProds (*rowStats*), 77
- rowQuantiles (*rowStats*), 77
- rowSds (*rowStats*), 77
- rowSkewness (*rowStats*), 77
- rowStats, 77
- rowStdevs (*rowStats*), 77
- rowVars (*rowStats*), 77

- rowVec (*colVec*), 22
- rsgh (*sgh*), 82
- rsnig (*snig*), 84
- rstable (*StableDistribution*), 87
- rt.lcg (*lcg*), 53
- runif.lcg (*lcg*), 53

- scaleTest, 24, 79
- ScalingLawPlot, 80
- scalinglawPlot (*ScalingLawPlot*), 80
- seqPalette (*colorPalette*), 18
- seriesPlot (*TimeSeriesPlots*), 90
- set.lcgseed (*lcg*), 53
- sfTest (*NormalityTests*), 67
- sgh, 82
- sghFit, 83
- shapiroTest (*NormalityTests*), 67
- show, fDISTFIT-method (*DistributionFits*), 25
- show, fHTEST-method (*fHTEST*), 27
- Sign (*Heaviside*), 38
- snig, 84
- snigFit, 86
- spearmanTest (*correlationTest*), 22
- StableDistribution, 87
- stableFit (*DistributionFits*), 25
- stableMode (*StableDistribution*), 87
- stableSlider (*StableDistribution*), 87
- stdev (*baseMethods*), 13
- symbolTable, 89

- teffectPlot (*acfPlot*), 9
- termPlot (*baseMethods*), 13
- terrainPalette (*colorPalette*), 18
- tFit (*DistributionFits*), 25
- TimeSeriesPlots, 90
- timPalette (*colorPalette*), 18
- topoPalette (*colorPalette*), 18
- tr, 91
- Triang (*triang*), 92
- triang, 92
- tslag, 72, 93

- varianceTest, 24, 94
- vec, 95
- vech (*vec*), 95

- vgrid (*decor*), 24
- volatility (*baseMethods*), 13