Package 'FatTailsR'

March 12, 2021

Title Kiener Distributions and Fat Tails in Finance

Description Kiener distributions K1, K2, K3, K4 and K7 to characterize distributions with left and right, symmetric or asymmetric fat tails in market finance, neuroscience and other disciplines. Two algorithms to estimate with a high accuracy distribution parameters, quantiles, value-at-risk and expected shortfall. Include power hyperbolas and power hyperbolic functions.

Version 1.8-0

Date 2021-03-12

Depends R (>= 3.1.0)

Imports minpack.lm, timeSeries, parallel, methods, stats

Suggests zoo, xts

Author Patrice Kiener [aut, cre] (<https://orcid.org/0000-0002-0505-9920>)

Maintainer Patrice Kiener <fattailsr@inmodelia.com>

URL https://www.inmodelia.com/fattailsr-en.html

License GPL-2

Encoding UTF-8

LazyData true

NeedsCompilation no

RoxygenNote 7.1.1

Repository CRAN

Date/Publication 2021-03-12 09:00:02 UTC

R topics documented:

`FatTailsR-package` *Package FatTailsR*

Description

This package includes Kiener distributions K1, K2, K3, K4 and K7 and two estimation algorithms to characterize with a high precision symmetric or asymmetric distributions with left and right fat tails that appear in market finance, neuroscience and many other disciplines. The estimation of the distribution parameters, quantiles, value-at-risk and expected shortfall is usually very accurate. Two datasets are provided, as well as power hyperbolas and power hyperbolic functions which are simplified versions of symmetric distribution K1.

Download the pdf cited in the references to get an overview of the theoretical part and several examples on stocks and indices.

A commercial package, `FatTailsRplot`, with advanced plotting functions and calculation of matrix of stocks over rolling windows is also developped by the author.

Details

With so many functions, this package could look fat. But it's not! It's rather agile and easy to use! The various functions included in this package can be assigned to the following groups:

1. Two datasets presented in different formats: list, data.frame, matrix, timeSeries, xts, zoo:
   - `getDSdata`.
   - `extractData`, dfData, mData, tData, xData, zData.
2. Functions to check the dimensions of vector, matrix, array, list:
   - `dimdim`, dimdim1, dimdimc.
3. Functions to calculate (positive, negative) prices to returns on vector, matrix, array, list, data.frame, timeSeries, xts, zoo:
   - `elevate`.
   - `fatreturns`, logreturns.
4. Several predefined vectors of probability. One function to check them. A conversion function from probabilities to characters
   - `pprobs0`, pprobs1, pprobs2, ..., pprobs9.
   - `checkquantiles`.
   - `getnamesk`.
5. Miscellaneous functions related to the logistic function:
   - `logit`, invlogit, ltmlogis, rtmlogis, eslogis.
6. Power hyperbolas, power hyperbolic functions and their reciprocal functions:
   - `exphp`, coshp, sinhp, tanhp, sechp, cosechp, cotanhp.
   - `loghp`, acoshp, asinhp, atanhp, asechp, acosechp, acotanhp.
   - `kashp`, dkashp_dx, ashp.
7. Logishp function, kogit and invkogit = logistic function + power hyperbolas:
   - d, p, q, r, dp, dq, l, dl, ql `logishp`.
   - `kogit`, invkogit.
8. Conversion functions between parameters related to Kiener distributions K1, K2, K3, K4:
   - `aw2k`, aw2d, aw2e, ad2e, ad2k, ad2w, ae2d, ae2k, ae2w, ak2e, ak2w, de2a, de2k, de2w, dk2a, dk2e, dw2a, dw2e, dw2k, ek2a, ak2d, ek2w, aw2a, aw2d, ew2a, aw2d, ew2k, kd2a, kd2e, kd2w, ke2a, ke2d, ke2w, kw2a, kw2d, kw2e.
   - `pk2pk`.
9. Kiener distributions K1, K2, K3, K4 and the new K7 (introduced in v1.7-0):
   - d, p, q, r, dp, dq, l, dl, ql, var, ltm, rtm, dtmq, es `kiener1`,
   - d, p, q, r, dp, dq, l, dl, ql, var, ltm, rtm, dtmq, es `kiener2`,
   - d, p, q, r, dp, dq, l, dl, ql, var, ltm, rtm, dtmq, es `kiener3`,
   - d, p, q, r, dp, dq, l, dl, ql, var, ltm, rtm, dtmq, es `kiener4`,
   - d, p, q, r, dp, dq, l, dl, ql, var, ltm, rtm, dtmq, es `kiener7`.
10. Quantile (VaR) corrective function (as a multiplier of the logistic function). Expected shortfall corrective function (as a multiplier of the expected shortfall of the logistic distribution):
    - `ckiener1`, ckiener2, ckiener3, ckiener4, ckiener7.

- hkiener1, hkiener2, hkiener3, hkiener4, hkiener7.

11. Moments of the distribution estimated from the dataset and from the regression parameters:
    - xmoments.
    - kmoments, kmoment, kcmoment, kmean, kstandev, kvariance, kskewness, kkurtosis, kekurtosis.
12. Regression and estimation functions to estimate Kiener distribution parameters on a given dataset. *fit* and *param* are wrappers of algorithms reg and estim. reg uses an unweighted nonlinear regression function. estim uses a fast estimation based on quantiles:
    - regkienerLX, laplacegaussnorm.
    - fitkienerX.
    - paramkienerX,paramkienerX5,paramkienerX7.
13. Functions related to paramkienerX:
    - elevenprobs, sevenprobs, fiveprobs.
    - estimkiener11, estimkiener7, estimkiener5.
    - roundcoefk.
    - checkcoefk.
14. Predefined subsets of parameters to extract them from the long vector fitk obtained after regression/estimation regkienerLX, fitkienerX :
    - exfit0, ..., exfit7.

For a quick start, jump to the functions regkienerLX, fitkienerX and run the examples. Then, download and read the documents in pdf format cited in the references to get an overview on the major functions. Finally, explore the other examples.

References

P. Kiener, Explicit models for bilateral fat-tailed distributions and applications in finance with the package FatTailsR, 8th R/Rmetrics Workshop and Summer School, Paris, 27 June 2014. Download it from: https://www.inmodelia.com/exemples/2014-0627-Rmetrics-Kiener-en.pdf

P. Kiener, Fat tail analysis and package FatTailsR, 9th R/Rmetrics Workshop and Summer School, Zurich, 27 June 2015. Download it from: https://www.inmodelia.com/exemples/2015-0627-Rmetrics-Kiener-en.pdf

Examples

```
require(graphics)
require(minpack.lm)
require(timeSeries)

### Load the datasets and select one number (1-16)
DS     <- getDSdata()
j      <- 5

### and run this block
X      <- DS[[j]]
```

```
nameX  <- names(DS)[j]
reg    <- regkienerLX(X)
lgn    <- laplacegaussnorm(X)
lleg   <- c("logit(0.999) = 6.9", "logit(0.99)   = 4.6",
            "logit(0.95)   = 2.9", "logit(0.50)   = 0",
            "logit(0.05)   = -2.9", "logit(0.01)   = -4.6",
            "logit(0.001) = -6.9  ")
pleg   <- c( paste("m =",  reg$coefr4[1]), paste("g  =", reg$coefr4[2]),
              paste("k  =", reg$coefr4[3]), paste("e  =", reg$coefr4[4]) )

## Main plot
op     <- par(mfrow = c(1,1), mgp = c(1.5,0.8,0), mar = c(3,3,2,1))
plot(reg$dfrXP, main = nameX)
legend("top", legend = pleg, cex = 0.9, inset = 0.02 )
lines(reg$dfrEP, col = 2, lwd = 2)
points(reg$dfrQkPk, pch = 3, col = 2, lwd = 2, cex = 1.5)
lines(lgn$dfrXPn, col = 7, lwd = 2)

## Plot F(X) > 0,97
front = c(0.06, 0.39, 0.50, 0.95)
par(fig = front, new = TRUE, mgp = c(1.5, 0.6, 0), las = 0)
plot( reg$dfrXP[which(reg$dfrXP$P > 0.97),] , pch = 1, xlab = "", ylab = "", main = "F(X) > 0,97" )
lines(reg$dfrEP[which(reg$dfrEP$P > 0.97),], type="l", col = 2, lwd = 3 )
lines(lgn$dfrXPn[which(lgn$dfrXPn$Pn > 0.97),], type = "l", col = 7, lwd= 2 )
points(reg$dfrQkPk, pch = 3, col = 2, lwd = 2, cex = 1.5)
points(lgn$dfrQnPn, pch = 3, col = 7, lwd = 2, cex = 1)

## Plot F(X) < 0,03
front = c(0.58, 0.98, 0.06, 0.61)
par(fig = front, new = TRUE, mgp = c(0.5, 0.6, 0), las = 0 )
plot( reg$dfrXP[which(reg$dfrXP$P < 0.03),] , pch = 1, xlab = "", ylab = "", main = "F(X) < 0,03")
lines(reg$dfrEP[which(reg$dfrEP$P < 0.03),], type = "l", col = 2, lwd = 3 )
lines(lgn$dfrXPn[which(lgn$dfrXPn$Pn < 0.03),], type = "l", col= 7, lwd= 2 )
points(reg$dfrQkPk, pch = 3, col = 2, lwd = 2, cex = 1.5)
points(lgn$dfrQnPn, pch = 3, col = 7, lwd = 2, cex = 1)

## Moments from the parameters (k) and from the Dataset (X)
round(cbind("k" = kmoments(reg$coefk, lengthx = nrow(reg$dfrXL)), "X" = xmoments(X)), 2)
attributes(reg)
### End block
```

---

aw2k *Local Conversion Functions Between Kiener Distribution Parameters*

Description

Conversion functions between parameters a, k, w, d, e used in Kiener distributions K2, K3 and K4.

Usage

```
aw2k(a, w)

aw2d(a, w)

aw2e(a, w)

ad2e(a, d)

ad2k(a, d)

ad2w(a, d)

ae2d(a, e)

ae2k(a, e)

ae2w(a, e)

ak2d(a, k)

ak2e(a, k)

ak2w(a, k)

de2a(d, e)

de2k(d, e)

de2w(d, e)

dk2a(d, k)

dk2e(d, k)

dk2w(d, k)

dw2a(d, w)

dw2e(d, w)

dw2k(d, w)

ek2a(e, k)

ek2d(e, k)

ek2w(e, k)
```

```
ew2a(e, w)

ew2d(e, w)

ew2k(e, w)

kd2a(k, d)

kd2e(k, d)

kd2w(k, d)

ke2a(k, e)

ke2d(k, e)

ke2w(k, e)

kw2a(k, w)

kw2d(k, w)

kw2e(k, w)
```

Arguments

a	a numeric value.
w	a numeric value.
d	a numeric value.
e	a numeric value.
k	a numeric value.

Details

a (alpha) is the left tail parameter, w (omega) is the right tail parameter, d (delta) is the distortion parameter, e (epsilon) is the eccentricity parameter. k (kappa) is the harmonic mean of a and w and describes a global tail parameter. They are defined by:

$$aw2k(a,w) = k = 2/(1/a + 1/w) = \frac{2}{\frac{1}{a} + \frac{1}{w}}$$

$$aw2d(a,w) = d = (-1/a + 1/w)/2 = \frac{-\frac{1}{a} + \frac{1}{w}}{2}$$

$$aw2e(a,w) = e = (a-w)/(a+w) = \frac{a-w}{a+w}$$

$$kd2a(k,d) = a = 1/(1/k - d) = \frac{1}{\frac{1}{k} - d}$$

$$kd2w(k,d) = w = 1/(1/k + d) = \frac{1}{\frac{1}{k} + d}$$

$$ke2a(k,e) = a = k/(1-e) = \frac{k}{1-e}$$

$$ke2w(k,e) = w = k/(1+e) = \frac{k}{1+e}$$

$$ke2d(k,e) = d = e/k = \frac{e}{k}$$

$$kd2e(k,d) = e = k * d$$

$$de2k(k,e) = k = e/d = \frac{e}{d}$$

See Also

The asymmetric Kiener distributions K2, K3, K4: kiener2, kiener3, kiener4

Examples

```
aw2k(4, 6); aw2d(4, 6); aw2e(4, 6)
outer(1:6, 1:6, aw2k)
```

---

checkcoefk *Check Coefk*

Description

Check that coefk is either a vector of length 7 or a matrix with 7 columns or an array with length of last dimension equal to 7.

Usage

```
checkcoefk(coefk, dim = c(1, 2), STOP = TRUE)
```

Arguments

coefk	numeric, matrix or data.frame representing parameters c(m,g,a,k,w,d,e).
dim	numeric. Accepted dimension(s) for coefk: 1 for vector, 2 for matrix, 3 for array. List is not accepted. Default is c(1, 2).
STOP	boolean. If an error is encountered, TRUE stops the function and returns an error message. FALSE just returns FALSE.

Examples

```
(coefk <- paramkienerX(getDSdata()))
checkcoefk(coefk)
checkcoefk(t(coefk), STOP = FALSE)
```

---

checkquantiles *Check Quantiles and Probabilities*

Description

Check that quantiles (or probabilities) are all different from each other and correctly ordered. If proba = TRUE, check that values are in range (0, 1).

Usage

```
checkquantiles(x, proba = FALSE, acceptNA = FALSE, STOP = TRUE)
```

Arguments

x	vector of quantiles.
proba	boolean. If TRUE, check range (0,1).
acceptNA	boolean. If FALSE, NA value are not accepted.
STOP	boolean. If an error is encountered, TRUE stops the function and returns an error message. FALSE just returns FALSE.

Examples

```
lst <- list(
  0.8,
  c(0.1, 0.5, 0.8),
  c(0.1, 0.5, 0.8, 0.2),
  c(2, 3, 1),
  c(2, 3),
  -0.01,
  NA,
  c(NA, NA),
  c(0.1, NA),
  c(0.1, NA, 0.5, 0.8),
  c(0.1, NA, 0.8, NA, 0.5),
  c(12, NA)
)

## Evaluate
```

```
for (i in seq_along(lst)) {
  cat(i, lst[[i]], " : ",
      checkquantiles(lst[[i]], proba = FALSE, STOP = FALSE),
      checkquantiles(lst[[i]], proba = TRUE, STOP = FALSE),
      checkquantiles(lst[[i]], proba = FALSE, acceptNA = TRUE, STOP = FALSE),
      checkquantiles(lst[[i]], proba = TRUE,  acceptNA = TRUE, STOP = FALSE),
     "\n")
}

sapply(lst, checkquantiles, proba = TRUE, acceptNA = TRUE, STOP = FALSE)

## Not run:
checkquantiles(matrix((1:12)/16, ncol=3), proba = TRUE, STOP = FALSE)
## End(Not run)
```

---

ckiener1234 *Quantile (VaR) and Expected Shortfall Corrective Functions*

Description

Quantile functions (or VaR) and Expected Shortfall of Kiener distributions K1, K2, K3 and K4, usually calculated at pprobs2 = c(0.01, 0.025, 0.05, 0.95, 0.975, 0.99), can be expressed as:

1. Quantile of the logit function multiplied by a fat tail (c)orrective function ckiener1234;
2. Expected s(h)ortfall of the logistic function multiplied by a corrective function hkiener1234.

Both functions ckiener1234 and hkiener1234 are independant from the scale parameter g and are indirect measures of the tail curvature. A value close to 1 indicates a model similar to the logistic function with almost no curvature and probably parameter k > 8. When k (or a,w) decreases, the values of c and h increase and indicate some more pronounced symmetric or asymmetric curvature, depending on values of d,e. Note that if (min(a,k,w) <= 1), ckiener1234 still exists but the expected shortfall and hkiener1234 become undefined (NA).

Some financial applications use threshold values on ckiener1234 or hkiener1234 to select or discard stocks over time as they become less or more risky.

Usage

```
hkiener1(p, m = 0, g = 1, k = 3.2, lower.tail = TRUE, log.p = FALSE)

hkiener2(p, m = 0, g = 1, a = 3.2, w = 3.2, lower.tail = TRUE, log.p = FALSE)

hkiener3(p, m = 0, g = 1, k = 3.2, d = 0, lower.tail = TRUE, log.p = FALSE)

hkiener4(p, m = 0, g = 1, k = 3.2, e = 0, lower.tail = TRUE, log.p = FALSE)

hkiener7(p, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0), lower.tail = TRUE,
  log.p = FALSE)
```

```
ckiener1(p, k = 3.2, lower.tail = TRUE, log.p = FALSE)

ckiener2(p, a = 3.2, w = 3.2, lower.tail = TRUE, log.p = FALSE)

ckiener3(p, k = 3.2, d = 0, lower.tail = TRUE, log.p = FALSE)

ckiener4(p, k = 3.2, e = 0, lower.tail = TRUE, log.p = FALSE)

ckiener7(p, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0), lower.tail = TRUE,
  log.p = FALSE)
```

Arguments

p	numeric or vector of probabilities.
m	numeric. parameter m used in model K1, K2, K3 and K4.
g	numeric. parameter g used in model K1, K2, K3 and K4.
k	numeric. parameter k used in model K1, K3 and K4.
lower.tail	logical. If TRUE, use p. If FALSE, use 1-p.
log.p	logical. If TRUE, probabilities p are given as log(p).
a	numeric. parameter a used in model K2.
w	numeric. parameter w used in model K2.
d	numeric. parameter d used in model K3.
e	numeric. parameter e used in model K4.
coefk	vector or 7 columns-matrix representing parameters c(m,g,a,k,w,d,e) obtained from paramkienerX.

See Also

logit, qkiener1, qkiener2, qkiener3, qkiener4, fitkienerX.

---

dfData *Datasets dfData, mData, tData, xData, zData, extractData : dfData*

Description

A list of datasets in data.frame, matrix, timeSeries, xts and zoo formats. This is the data.frame format. Visit extractData for more information.

dimdim *Length and Dimensions of Vector, Matrix, Array, Data.Frame, List*

Description

Dimensions and length of vector, matrix, array, data.frame and list. A friendly version of `dim` that returns the true dimension rather than the sometimes unexpected `NULL` value. The number of dimensions appears first, then the length in each dimension. A special case is list: the list's length (number of items) is turned into a negative integer and the dimension/length of each item is either positive if the item is a vector, matrix, array or data.frame or negative if the item is itself a list. Only the first level of the list is explored.

`dimdim1` and `dimdimc` return the first item of `dimdim`, thus the true dimension, either as an integer or as a character and, in this latest case, always `"-1"` for lists.

Notes: From version 1.6.2 (April 2016), `dimdim(NULL) = c(0,0)`. (before `c(1,0)`). Hence, `dimdim1(NULL) = 0` and `dimdimc(NULL) = "0"`. Some problems may occur with S4 objects like `dimdim(qualityTools::fracDesign(k = 3,gen = "C = AB"))`.

Usage

```
dimdim(x)

dimdim1(x)

dimdimc(x)
```

Arguments

x	vector, matrix, array, data.frame, list.

Examples

```
require(timeSeries)

dimdim(NULL)
dimdim(NA); dimdim(NaN); dimdim(Inf); dimdim(TRUE); dimdim(FALSE)
dimdim(11:39)
dimdim(LETTERS[1:8])
dimdim(matrix(1:60, ncol=5))
dimdim(extractData())
dimdim(as.data.frame(extractData()))
dimdim(data.frame(X=1:2, Y=1:4, Z=LETTERS[1:8]))
dimdim(array(1:240, c(8,6,5)))
dimdim(array(1:240, c(4,2,6,5)))
dimdim(getDSdata())
dimdim(zData)
dimdim(xData)
dimdim(tData)
```

```
dimdim1(matrix(1:60, ncol=5))
dimdimc(matrix(1:60, ncol=5))
dimdim1(tData)
dimdimc(tData)
```

`elevate` *Elevate*

Description

A transformation to turn negative prices into positive prices and maintain at the same time the hierachy between all prices.

Usage

```
elevate(X, e = NULL)
```

Arguments

`X`	The prices.
`e`	numeric. The focal point of the hyperbola.

Details

Negative prices in financial markets, like interest rates in Europe, are a nightmare as the rough calculation of the returns generates non-sense values. `elevate` uses an hyperbola and implements the following formula:

$$elevate(x, e) = (x + sqrt(x * x + e * e))/2$$

There is currently no rule of thumb to calculate `e`. When $e = NULL$, there is no change and the output is identical to the input. When $e = 0$, all negative values are turned to 0.

Examples

```
require(graphics)

X <- (-50:100)/5
plot( X, elevate(X, e = 5), type = "l", ylim = c(0, 20) )
lines(X, elevate(X, e = 2),   col = 2)
lines(X, elevate(X, e = 1),   col = 3)
lines(X, elevate(X, e = 0.5), col = 4)
lines(X, elevate(X, e = 0),   col = 1)
```

elevenprobs *Eleven, Seven, Five Probabilities*

Description

Extract from a dataset X a vector of 11, 7 or 5 probabilities:

- c(p1,p2,p3,0.25,0.35,0.50,0.65,0.75,1-p3,1-p2,1-p1)
- c(p1,p2,0.25,0.50,0.75,1-p2,1-p1)
- c(p1,0.25,0.50,0.75,1-p1)

where p1, p2 and p3 are the most extreme probabilities with values finishing by ..01, ..025 or ..05 that can be extracted from the dataset X. Parameters names are displayed if parnames = TRUE.

From version 1.8-0, p1 and 1-p1 can be associated to the i-th and (N-i)-th element.

Usage

```
elevenprobs(X, parnames = FALSE)

sevenprobs(X, parnames = FALSE)

fiveprobs(X, i = 4, parnames = FALSE)
```

Arguments

X	numeric. Vector of quantiles.
parnames	boolean. Output parameter vector with or without names.
i	integer. The i-th and (N-i)-th elements for which the probabilities p1 and 1-p1 are calculated. If (i == 0), the method used before version 1.8-0 : the extreme finishing by ..01, ..025 or ..05.

See Also

fitkienerX, estimkiener11.

Examples

```
require(timeSeries)

## DS
DS  <- getDSdata()
for (j in 1:16) { print(round(elevenprobs(DS[[j]]), 6)) }
z   <- cbind(t(sapply(DS, elevenprobs)), sapply(DS, length))
colnames(z) <- c("p1","p2","p3","p.25","p.35","p.50","p.65","p.75","1-p3","1-p2","1-p1","length")
z
```

```
## Choose j in 1:16
j   <- 1
X   <- sort(DS[[j]])
leX <- logit(eX <- elevenprobs(X))
lpX <- logit(ppoints(length(X), a = 0))
plot(X, lpX)
abline(h = leX, lty = 3)
mtext(eX, side = 4, at = leX, las = 1, line = -3.3)
```

`estimkiener11` *Estimation Functions with 5, 7 or 11 Quantiles*

Description

Several functions to estimate the parameters of asymmetric Kiener distributions with just 5, 7 or 11 quantiles.

Usage

```
estimkiener11(x11, p11, ord = 7, maxk = 10)

estimkiener7(x7, p7, maxk = 10)

estimkiener5(x5, p5, maxk = 20, maxe = 0.9)
```

Arguments

`ord`	integer. Option for probability selection and treatment.
`maxk`	numeric. Maximum value for k (kappa).
`x5, x7, x11`	vector of 5, 7 or 11 quantiles.
`p5, p7, p11`	vector of 5, 7 or 11 probabilities.
`maxe`	numeric. Maximum value for abs(e) (epsilon). Maximum is `maxe = 1`.

Details

These functions, called by `paramkienerX5`, `paramkienerX7`, `paramkienerX`, use 5, 7 or 11 probabilites and quantiles to estimate the parameters of Kiener distributions.

`p5,x5` are obtained with functions `fiveprobs(X)` and `quantile(p5)`.

`p7,x7` are obtained with functions `sevenprobs(X)` and `quantile(p7)`.

`p11,x11` are obtained with functions `elevenprobs(X)` and `quantile(p11)`.

The extraction of the 11 probabilities is controlled with the option `ord` which can take 12 integer values, `ord = 7` being the default. Small dataset should consider `ord = 5` and large dataset can consider `ord = 12`:

1. c(p1,0.35,0.50,0.65,1-p1)
2. c(p2,0.35,0.50,0.65,1-p2)
3. c(p1,p2,0.35,0.50,0.65,1-p2,1-p1)
4. c(p1,p2,p3,0.35,0.50,0.65,1-p3,1-p2,1-p1)
5. c(p1,0.25,0.50,0.75,1-p1)
6. c(p2,0.25,0.50,0.75,1-p2)
7. c(p1,p2,0.25,0.50,0.75,1-p2,1-p1)
8. c(p1,p2,p3,0.25,0.50,0.75,1-p3,1-p2,1-p1)
9. c(p1,0.25,0.35,0.50,0.65,0.75,1-p1)
10. c(p2,0.25,0.35,0.50,0.65,0.75,1-p2)
11. c(p1,p2,0.25,0.35,0.50,0.65,0.75,1-p2,1-p1)
12. c(p1,p2,p3,0.25,0.35,0.50,0.65,0.75,1-p3,1-p2,1-p1)

p5 = fiveprobs(X) corresponds to c(p1,0.25,0.50,0.75,1-p1).

p7 = sevenprobs(X) corresponds to c(p1,p2,0.25,0.50,0.75,1-p2,1-p1).

The above probabilities are then transfered to the quantile function whose parameter type can change significantly the extracted quantiles. Our experience is that type = 6 is appropriate when k > 1.9 and type = 5 is appropriate when k < 1.9. Other types type = 8 and type = 9 can be considered as well. The other types should be ignored. (Note: when k < 1.5, algorithm algo = "reg" returns better results).

Parameter maxk controls the maximum allowed value for estimated parameter k. Reasonnable values are maxk = 10,15,20. Default is maxk = 10 to be consistent with regkienerLX.

See Also

elevenprobs, paramkienerX, quantile, roundcoefk.

Examples

```
require(timeSeries)

## Choose j in 1:16. Choose ord in 1:12 (7 is default)
j    <- 5
ord  <- 5
DS   <- getDSdata()
p11  <- elevenprobs(DS[[j]])
x11  <- quantile(DS[[j]], probs = p11, na.rm = TRUE, names = TRUE, type = 6)
round(estimkiener11(x11, p11, ord), 3)

## Compare the results obtained with the 12 different values of ord on stock j
compare <- function(ord, x11, p11) {estimkiener11(x11, p11, ord)}
coefk   <- t(sapply(1:12, compare, x11, p11))
rownames(coefk) <- 1:12
mcoefk  <- apply(coefk, 2, mean) # the mean of the 12 results above
```

```
roundcoefk(rbind(coefk, mcoefk), 13)
```

---

exfit0 *Parameter Subsets*

Description

Some vectors of parameter names to be used with parameter `exfitk` in functions regkienerLX(.., exfitk = ...) and `fitkienerX(..,exfitk = ...)` or to subset the vector (or matrix) `fitk` obtained after regression `fitk <-regkienerLX(..)$fitk` or estimation `fitk <-fitkienerX(..)`. Visit `fitkienerX` for details on each parameter.

```
exfit0 <-c("lh","ret")

exfit1 <-c("m","g","a","k","w","d","e")

exfit2 <-c("m1","sd","sk","ke","m1x","sdx","skx","kex")

exfit3 <-c("q.01","q.05","q.95","q.99","ltm.025","rtm.975")

exfit4 <-c("VaR.01","VaR.05","VaR.95","VaR.99","ES.025","ES.975")

exfit5 <-c("c.01","c.05","c.95","c.99","h.025","h.975")

exfit6 <-c(exfit1,exfit2,exfit3,exfit4,exfit5)

exfit7 <-c(exfit0,exfit1,exfit2,exfit3,exfit4,exfit5)
```

Usage

```
exfit0

exfit1

exfit2

exfit3

exfit4

exfit5

exfit6

exfit7
```

Format

An object of class character of length 2.

An object of class character of length 7.

An object of class character of length 8.

An object of class character of length 6.

An object of class character of length 6.

An object of class character of length 6.

An object of class character of length 33.

An object of class character of length 35.

Examples

```
require(minpack.lm)
require(timeSeries)

### Load the datasets and select one number j in 1:16
j      <- 5
DS     <- getDSdata()
(fitk  <- regkienerLX(DS[[j]])$fitk)
fitk[exfit3]
fitkienerX(DS[[j]], exfitk = exfit3)
```

---

exphp *Power Hyperbolas and Power Hyperbolic Functions*

Description

These functions define the power hyperbola exphp and the associated power hyperbolic cosine, sine, tangent, secant, cosecant, cotangent. They are similar to the traditional hyperbolic functions with term x receiving a nonlinear transformation via the function kashp.

Usage

```
exphp(x, k = 1)

coshp(x, k = 1)

sinhp(x, k = 1)

tanhp(x, k = 1)
```

```
sechp(x, k = 1)

cosechp(x, k = 1)

cotanhp(x, k = 1)
```

Arguments

x	a numeric value, vector or matrix.
k	a numeric value, preferably strictly positive.

Details

exphp function is defined for x in (-Inf, +Inf) by:

$$exphp(x, k) = exp(kashp(x, k)) = exp(k * asinh(x/2/k))$$

coshp function is defined for x in (-Inf, +Inf) by:

$$coshp(x, k) = cosh(kashp(x, k))$$

sinhp function is defined for x in (-Inf, +Inf) by:

$$sinhp(x, k) = sinh(kashp(x, k))$$

tanhp function is defined for x in (-Inf, +Inf) by:

$$tanhp(x, k) = tanh(kashp(x, k))$$

sechp function is defined for x in (-Inf, +Inf) by:

$$sechp(x, k) = 1/coshp(x, k)$$

cosechp function is defined for x in (-Inf, 0) U (0, +Inf) by:

$$cosechp(x, k) = 1/sinhp(x, k)$$

cotanhp function is defined for x in (-Inf, 0) U (0, +Inf) by:

$$cotanhp(x, k) = 1/tanhp(x, k)$$

The undesired case k = 0 returns 0 for sinhp and tanhp, 1 for exphp, coshp and sechp, Inf for cosechp and cotanhp.

If k is a vector of length > 1, then the use of the function `outer` is recommanded.

See Also

The nonlinear transformation `kashp`, the inverse power hyperbolas and the inverse power hyperbolic functions `loghp`.

Examples

```
### Example 1
x  <- (-3:3)*3
exphp(x, k = 4)
coshp(x, k = 4)
sinhp(x, k = 4)
tanhp(x, k = 4)

### Example 2 outer + plot(exphp, coshp, sinhp, tanhp)
xmin  <- -10
xd    <- 0.5
x     <- seq(xmin, -xmin, xd) ; names(x) <- x
k     <- c(0.6, 1, 1.5, 2, 3.2, 10) ; names(k) <- k
olty  <- c(2, 1, 2, 1, 2, 1, 1)
olwd  <- c(1, 1, 2, 2, 3, 4, 2)
ocol  <- c(2, 2, 4, 4, 3, 3, 1)
op    <- par(mfrow = c(2,2), mgp = c(1.5,0.8,0), mar = c(3,3,2,1))

## exphp(x, k)
Texphp <- ts(cbind(outer(-x, k, exphp), "exp(-x/2)" = exp(-x/2)),
             start = xmin, deltat = xd)
plot(Texphp, plot.type = "single", ylim = c(0,20),
       lty = olty, lwd = olwd, col = ocol, xaxs = "i", yaxs = "i", xlab = "",
       ylab = "", main = "exphp(-x, k)" )
legend("topright", title = expression(kappa), legend = colnames(Texphp),
       inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

## coshp(x, k)
Tcoshp <- ts(cbind(outer(x, k, coshp), "cosh(x/2)" = cosh(x/2)),
             start = xmin, deltat = xd)
plot(Tcoshp, plot.type = "single", ylim = c(0,20),
       lty = olty, lwd = olwd, col = ocol, xaxs = "i", yaxs = "i",
       xlab = "", ylab = "", main = "coshp(x, k)" )
legend("top", title = expression(kappa), legend = colnames(Tcoshp),
       inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

## sinhp(x, k)
Tsinhp <- ts(cbind(outer(x, k, sinhp), "sinh(x/2)" = sinh(x/2)),
             start = xmin, deltat=xd)
plot(Tsinhp, plot.type = "single", ylim = c(-10,10),
       lty = olty, lwd = olwd, col = ocol, xaxs = "i", yaxs = "i",
       xlab = "", ylab = "", main = "sinhp(x, k)" )
legend("topleft", title = expression(kappa), legend = colnames(Tsinhp),
       inset = 0.02, lty = olty, lwd=  olwd, col = ocol, cex = 0.7 )

## tanhp(x, k)
Ttanhp <- ts(cbind(outer(x, k, tanhp), "tanh(x/2)" = tanh(x/2)),
             start = xmin, deltat = xd)
plot(Ttanhp, plot.type = "single", ylim = c(-1,1),
       lty = olty, lwd = olwd, col = ocol, xaxs = "i", yaxs = "i", xlab = "",
       ylab = "", main = "tanhp(x, k)" )
```

```
legend("topleft", title = expression(kappa), legend = colnames(Ttanhp),
       inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )
### End Example 3
```

`extractData` *Datasets dfData, mData, tData, xData, zData, extractData : extractData*

Description

dfData, mData, tData, xData, zData are datasets made of lists of data.frame, matrix, timeSeries, xts and zoo components. They describe prices and returns of 10 financial series used in the documents and demos presented at 8th and 9th R/Rmetrics conferences (2014, 2015). See the references. The last serie (CHF, interest rates in Switzerland) exhibits negative prices. All distributions of logreturns exhibit fat tails. Function `extractData` converts subsets of mData, tData, xData, zData.

Usage

```
extractData(pr = "p", ft = "tss", start = "2007-01-01", end = "2013-12-31")
```

Arguments

`pr`	character. Extract prices or returns: `c("p","r","prices","returns")`.
`ft`	character. Output format among `c("tss","xts","zoo","dfr","bfr","mat")`.
`start`	character. Start date.
`end`	character. End date.

Details

10 financial series presented in four different formats for convenient use: dfData is the complete dataset in data.frame format with dates as row.names and one or several columns. tData, xData and zData are lists of timeSeries, xts and zoo formats and display only one column per stock.

1. "GOLD" from 1999-01-04 to 2013-12-31, dim 3694x1 (df, m, t, x, z).
2. "DEXIA" from 2008-10-27 to 20009-10-26, dim 255x1 (t, x, z), 255x5 (df).
3. "SOCGEN" from 1992-07-20 to 2013-12-31, dim 5445x1 (m, t, x, z), 5445x5 (df).
4. "VIVENDI" from 1992-07-20 to 2013-12-31, dim 5444x1 (m, t, x, z) 5444x5 (df).
5. "EURUSD" from 1999-01-03 to 2013-12-31, dim 3843x1 (m, t, x, z), 3843x4 (df).
6. "VIX" from 2004-01-02 to 2013-12-31, dim 2517x1 (m, t, x, z), 2517x4 (df).
7. "CAC40" from 1988-01-04 to 2013-12-31, dim 6574x1 (m, t, x, z), 6574x4 (df).
8. "DJIA" from 1896-05-26 to 2013-12-31, dim 32064x1 (m, df, t, x, z).
9. "SP500" from 1957-01-02 to 2013-12-31, dim 14350x1 (m, df, t, x, z).

10. "CHF" from 1995-01-02 to 2013-09-13, dim 4880x1 (t, x, z), 4880x8 (df). Interst rates in Switzerland. Include negative prices at the end of the dataset. Care is required to calculate the returns! (Use fatreturns and elevate). See the examples if you decide or need to remove it from the list.

Function extractData extracts 8 financial series from matrix mData (DEXIA and CHF are not included) and converts them into a 2 dimensions object with any of the following class:

- "tss" is for timeSeries with a timeDate index.
- "xts" is for xts with a POSIXct index.
- "zoo" is for zoo with a Date index.
- "dfr" is the usual R data.frame with the Date as index.
- "bfr" is a data.frame with the Date in the first column.
- "mat" is a matrix with the Date in the rownames.

The start date must be posterior to "2007-01-01" (default) and the end date must be anterior to "2013-12-31" (default).

References

P. Kiener, Explicit models for bilateral fat-tailed distributions and applications in finance with the package FatTailsR, 8th R/Rmetrics Workshop and Summer School, Paris, 27 June 2014. Download it from: https://www.inmodelia.com/exemples/2014-0627-Rmetrics-Kiener-en.pdf

P. Kiener, Fat tail analysis and package FatTailsR, 9th R/Rmetrics Workshop and Summer School, Zurich, 27 June 2015. https://www.inmodelia.com/exemples/2015-0627-Rmetrics-Kiener-en.pdf

See Also

tData, xData, zData, dfData, getDSdata.

Examples

```
library(zoo)
library(xts)
library(timeSeries)

### dfData, tData, xData, zData : prices only
attributes(dfData); attributes(tData); attributes(xData); attributes(zData)
lapply(dfData, head, 3)
lapply( mData, head, 3)
lapply( tData, head, 3)
lapply( xData, head, 3)
lapply( zData, head, 3)

### extractData : prices and logreturns
head(ptD <- extractData("p", "tss", "2009-01-01", "2012-12-31")) ; tail(ptD)
head(rtD <- extractData("r", "tss"))
```

```
head(pxD <- extractData("p", "xts"))
head(rxD <- extractData("r", "xts"))
head(pzD <- extractData("p", "zoo"))
head(rzD <- extractData("r", "zoo"))
head(pbD <- extractData("p", "bfr"))
head(rbD <- extractData("r", "bfr"))
head(pmD <- extractData("p", "mat"))
head(rmD <- extractData("r", "mat"))

### Remove item CHF (negative prices) from dfData, tData, xData, zData
Z <- dfData[names(dfData)[1:9]]; attributes(Z)
Z <- tData[names(tData)[1:9]]; attributes(Z)
Z <- xData[names(xData)[1:9]]; attributes(Z)
Z <- zData[names(zData)[1:9]]; attributes(Z)
```

`fatreturns` *Simple and Elaborated Prices to Returns*

Description

`fatreturns` is an elaborated function to compute prices to returns. It includes a pre-treatment for negative prices. It computes either log-returns (default) or percentage-returns. It handles properly NA values in the input vector, replacing them by 0 in the output vector. Doing so, it warrants that the sum of the log-returns (when selected) is equal to the difference of the log-prices. It works with vector, matrix, data.frame, timeSeries, xts, zoo, list, list of lists and even list of vector, data.frame, timeSeries, xts, zoo mixed together. The returned object is of same dimension and same class than the input object with the first line filled with 0. The results may be as per one, per cent (default), per thousand and per ten thousand.

`logreturns` is an improved version of function `100*diff(log(x))` to handle vector, matrix, data.frame and list. It handles properly the first line and the NA values. It does not control time, rownames and colnames but may return them.

Usage

```
fatreturns(x, log = TRUE, per = "cent", e = NULL, dfrcol = 1, na.rm = TRUE)

logreturns(x)

replaceNA(x)
```

Arguments

`x`	The prices (vector, data.frame, matrix, timeSeries, xts, zoo, list).
`log`	boolean. log returns or percentage returns.
`per`	character. Either "one", "cent, "thousand", "tenthousand" or "o", "c", "th", "te". Multiply the result by 1, 100, 1000, 10000.

e	NULL or positive numeric. NULL is for no change f(x)=x. A positive numeric designates the focal point of the hyperbola to turn negative prices into positive prices, keeping the hierarchy: f(x)=(x+sqrt(x*x+e*e))/2. There is currently no rules of thumb for the optimal value of e.
dfrcol	integer. For data.frame only, designates the column that handles the time and must be processed separately. Use dfrcol = 0 if all columns must be processed and there is no time (or turn the data.frame to a matrix).
na.rm	boolean. Replace x[t]=NA with the previous non-NA value available in the price serie such that (x[t-1],x[t]=x[t-1],x[t+1]) and calculate the returns accordingly. Force 0 in the first line of the returns if x[1]=NA.

Details

Negative prices in financial markets, like interest rates in Europe, are a nightmare as the rough calculation of the returns generates non-sense values. elevate uses an hyperbola and implements the following formula:

$$elevate(x, e) = (x + sqrt(x * x + e * e))/2$$

There is currently no rule of thumb to calculate e. When $e = NULL$, there is no change and the output is identical to the input. When $e = 0$, all negative values are turned to 0.

Examples

```
fatreturns(extractData())
logreturns(getDSdata())
```

---

fitkienerX *Estimation and Regression Functions for Kiener Distributions*

Description

Several functions to estimate the parameters of asymmetric Kiener distributions and display the results in a numeric vector or in a matrix. Algorithm "reg" (the default) uses a nonlinear regression and handle difficult cases. Algorithm "estim" has been completely rewritten in version 1.8-0 and is now very accurate, even for k<1. Adjustement on extreme quantiles can be controlled very precisely.

Usage

```
fitkienerX(X, algo = c("r", "reg", "e", "estim"), ord = 7, maxk = 10,
  mink = 1.53, maxe = 0.5, probak = pprobs2, dgts = NULL,
  exfitk = NULL, dimnames = FALSE, ncores = 1)

paramkienerX(X, algo = c("r", "reg", "e", "estim"), ord = 7, maxk = 10,
  mink = 1.53, maxe = 0.5, dgts = 3, parnames = TRUE,
```

```
  dimnames = FALSE, ncores = 1)

paramkienerX7(X, dgts = 3, n = 10, maxk = 20, maxe = 0.9,
  parnames = TRUE, dimnames = FALSE, ncores = 1)

paramkienerX5(X, dgts = 3, i = 4, maxk = 20, maxe = 0.9,
  parnames = TRUE, dimnames = FALSE, ncores = 1)
```

Arguments

X	numeric. Vector, matrix, array or list of quantiles.
algo	character. The algorithm used: "r" or "reg" for regression (default) and "e" or "estim" for quantile estimation.
ord	integer. Option for probability selection and treatment.
maxk	numeric. The maximum value of tail parameter k.
mink	numeric. The minimum value of tail parameter k.
maxe	numeric. The maximum value of absolute tail parameter |e|.
probak	numeric. Ordered vector of probabilities.
dgts	integer. The rounding of output parameters.
exfitk	character. A vector of parameter names to subset the output.
dimnames	boolean. Display dimnames.
ncores	integer. The number of cores for parallel processing of arrays.
parnames	boolean. Display parameter names.
n	integer. The 1:n and (N+i-n):N elements of X used to calculate synthetic quantiles at probability levels p1 and 1-p1.
i	integer. The i-th and (N-i)-th elements of X used to extract probabilities p1 and 1-p1 and quantiles x(p) and x(1-p).

Details

FatTailsR package currently uses two different algorithms to estimate the parameters of Kiener distributions K1, K2, K3 and K4.

- Functions fitkienerX(algo = "reg"), paramkienerX(algo = "reg") and regkienerLX use an unweighted nonlinear regression from logit(p) to X over the whole dataset. Depending the size of the dataset, calculation can be slow but is usually accurate and describes very well the last 1-10 points in the tails (except if there is a huge outlier).
- Functions fitkienerX(algo = "estim"), paramkienerX(algo = "estim"), paramkienerX5 and paramkienerX7 estimate the parameters with just 5 to 11 quantiles, 5 being the minimum. For averaging purpose, 11 quantiles are proposed (see below). Computation is almost instantaneous and reasonnably accurate. This is the recommanded method for intensive computation.

A typical input is a numeric vector or a matrix that describes the returns of a stock. A matrix must be in the format DS with DATES as rownames, STOCKS as colnames and (log-)returns as the content of the matrix. An array must be in the format DSL with DATES as rownames, STOCKS as

colnames LAGS in the third dimension and (log-)returns as the content of the array. A list can be a list of numeric but neither a list of matrix, a list of data.frame or a list of arrays.

Conversion from a (possible) time series format to a sorted numeric vector is done automatically and without any check of the initial format. Empirical probabilities of each point in the sorted dataset is calculated with the function `ppoints` whose parameter `a` has been set to `a = 0` as large datasets are very common in finance. The lowest acceptable size of a dataset is not clear at this moment. A minimum of 11 points has been set in `"reg"` algorithm and a minimum of 15 points has been set in `"estim"` algorithm. It might change in the future. If possible, use at least 21 points.

Parameter `algo` controls the algorithm used. Default is "reg".

When `algo = "reg"` (or `algo = "r"`), a nonlinear regression is performed with `nlsLM` from the logit of the empirical probabilities `logit(p)` over the quantiles X with the function `qlkiener4`. The maximum value of the tail parameter `k` is controlled by `maxk`. An upper value `maxk = 10` is appropriate for datasets of low and medium size, less than 20.000 or 50.000 points. For larger datasets, the upper limit can be extended up to `maxk = 20`. When this limit is reached, the shape of the distribution is very similar to the logistic distribution (at least when `e = 0`) and the use of this distribution should be considered. Remember that value `k < 2` describes a distribution with no stable variance and `k < 1` describes a distribution with no stable mean.

When `algo = "estim"` (or `algo = "e"`), 5 to 11 quantiles are used to estimate the parameters. The minimum is 5 quantiles : the median x.50, two quantiles at medium distance to the median, usually x.25 and x.75 and two quantiles located close to the extremes of the dataset, for instance x.01 and x.99 if the dataset `X` has more than 100 points, x.0001 and x.9999 if the dataset `X` has more than 10.000 points and so on if the dataset is larger. These quantiles are extracted with function `fiveprobs`. Small datasets must contain at least 15 different points.

With the idea of averaging the results (but without any guarantee of better estimates), calculation has been extended to 11 probabilities extracted from `X` with the function `elevenprobs` where p1, p2 and p3 are the most extreme probabilities of the dataset `X` with values finishing either by `.x01` or `.x025` or `.x05`:

- `p11 = c(p1,p2,p3,0.25,0.35,0.50,0.65,0.75,1-p3,1-p2,1-p1)`

Selection of subsets among these 11 probabilities is controlled with the option `ord` which can take 12 different values. For instance, the default `ord = 7` computes the parameters at probabilities `c(p1,0.25,0.50,0.75,1-p1)` and `c(p2,0.25,0.50,0.75,1-p2)`. Parameters `d` and `k` are averaged first and the results of these averages are used to compute the other parameters `g,a,w,e`. Small dataset should consider `ord = 5` and large dataset can consider `ord = 12`. The 12 possible values of `ord` are:

1. `c(p1,0.35,0.50,0.65,1-p1)`
2. `c(p2,0.35,0.50,0.65,1-p2)`
3. `c(p1,p2,0.35,0.50,0.65,1-p2,1-p1)`
4. `c(p1,p2,p3,0.35,0.50,0.65,1-p3,1-p2,1-p1)`
5. `c(p1,0.25,0.50,0.75,1-p1)`
6. `c(p2,0.25,0.50,0.75,1-p2)`
7. `c(p1,p2,0.25,0.50,0.75,1-p2,1-p1)`
8. `c(p1,p2,p3,0.25,0.50,0.75,1-p3,1-p2,1-p1)`
9. `c(p1,0.25,0.35,0.50,0.65,0.75,1-p1)`

10. c(p2,0.25,0.35,0.50,0.65,0.75,1-p2)
11. c(p1,p2,0.25,0.35,0.50,0.65,0.75,1-p2,1-p1)
12. c(p1,p2,p3,0.25,0.35,0.50,0.65,0.75,1-p3,1-p2,1-p1)

paramkienerX5 is a simplified version of paramkienerX with predefined values algo = "estim", ord = 5, maxk = 10 and direct access to internal subfunctions. It uses the following probabilities:

- p5 = c(p1,0.25,0.50,0.75,1-p1)

paramkienerX7 is a simplified version of paramkienerX with predefined values algo = "estim", ord = 7, maxk = 10 and direct access to internal subfunctions. It uses the following probabilities:

- p7 = c(p1,p2,0.25,0.50,0.75,1-p2,1-p1)

The quantiles corresponding to the above probabilities are then extracted with the function quantile whose parameter type has been set to type = 6 as it returns the closest values to the true quantiles (according to our experience) for all k > 1.9. (Note: when k < 1.5, algorithm algo = "reg" returns better results). Both probabilities and quantiles are then transfered to estimkiener11 for calculation.

probak controls the probabilities at which the model is tested with the parameter estimates. fitkienerX and regkienerLX share the same subroutines. The default for fitkienerX and regkienerLX is pprobs2 = c(0.01,0.025,0.05,0.95,0.975,0.99) as those values are usual in finance. Other sets of values are provided at pprobs0.

Rounding the results is useful to display nice results, especially in a matrix or in a data.frame. dgts = 13 is recommanded as a, k, w are usually significant at 1 digit.

- dgts = NULL does not perform any rounding.
- dgts = 0 to 9 rounds all parameters at the same level.
- dgts = 10 to 27 rounds the parameters at various levels for nice display. See roundcoefk for the details. (Note: the rounding 10 to 27 currently works with paramkienerX, paramkienerX5, paramkienerX7 but not yet with fitkienerX).

Extracting the most useful parameters from the (quite long) vector/matrix fitk is controlled by parameter exfitk that calls user-defined or predefined parameter subsets like exfit0, ..., exfit7. IMPORTANT: never subset fitk by rank number as new items may be added in the future and rank may vary.

Calculation of vectors, matrices and lists is not parallelized. Parallelization of code for arrays was introduced in version 1.5-0 and improved in version 1.5-1. ncores controls the number of cores allowed to the process (through parApply which runs on Unices and Windows and requires about 2 seconds to start). ncores = 1 means no parallelization. ncores = 0 is the recommanded option. It uses the maximum number of cores available on the computer, as detected by detectCores, minus 1 core, which gives the best performance in most cases. Although appealing, this automatic selection may be sometimes dangerous. For instance, the instruction f(X,ncores_max) -f(X,ncores_max), a nice way to compute an array of 0, will call 2 ncores_max and crash R. ncores = 2,..,99 sets manually the number of cores. If the requested value is larger than the maximum number of cores, this value is automatically reduced (with a warning) to this maximum. Hence, this latest option provides one core more than option ncores = 0.

NOTE: fitkienerLX, regkienerX, estimkiener(X,5,7) were introduced in v1.2-0 and replaced in version v1.4-1 by fitkienerX and paramkiener(X,5,7) to accomodate vector, matrix, arrays and lists. We apologize to early users who need to rewrite their codes.

Value

`paramkienerX`: a vector (or a matrix) of parameter estimates `c(m,g,a,k,w,d,e)`.

`fitkienerX`: a vector (or a matrix) made of several parts:

- `ret` : the return over the period calculated with `sum(x)`. Thus, assume log-returns.
- `m,g,a,k,w,d,e` : the parameter estimates.
- `m1,sd,sk,ke` : the mean, standard deviation, skewness and excess of kurtosis computed from the parameter estimates.
- `m1x,sdx,skx,kex` : The mean, standard deviation, skewness and excess of kurtosis computed from the dataset.
- `lh` : the length of the dataset over the period.
- `q.` : quantile estimated with the parameter estimates.
- `VaR.` : Value-at-Risk, positive in most cases.
- `c.` : corrective tail coefficient = (q - m) / (q_logistic_function - m).
- `ltm.` : left tail mean (signed ES on the left tail, usually negative).
- `rtm.` : right tail mean (signed ES on the right tail, usually positive).
- `dtmq.` : (p<=0.5 left, p>0.5 right) tail mean minus quantile.
- `ES.` : expected shortfall, positive in most cases.
- `h.` : corrective ES = (ES - m) / (ES_logistic_function - m).
- `desv.` : ES - VaR, usually positive.
- `l.` : quantile estimated by the tangent logistic function.
- `dl.` : quantile - quantile_logistic_function.
- `g.` : quantile estimated by the Laplace-Gauss function.
- `dg.` : quantile - quantile_Laplace_Gauss_function.

IMPORTANT : if you need to subset `fitk`, always subset it by parameter names and never subset it by rank number as new items may be added in the future and rank may vary. Use for instance `exfit0`, ..., `exfit7`.

References

P. Kiener, Fat tail analysis and package FatTailsR, 9th R/Rmetrics Workshop and Summer School, Zurich, 27 June 2015. https://www.inmodelia.com/exemples/2015-0627-Rmetrics-Kiener-en.pdf

See Also

`regkienerLX`, `estimkiener11`, `roundcoefk`, `exfit6`.

Examples

```
require(minpack.lm)
require(timeSeries)

### Load the datasets and choose j in 1:16
DS     <- getDSdata()
j      <- 5

### and run this block
probak <- c(0.01, 0.05, 0.95, 0.99)
X      <- DS[[j]] ; names(DS)[j]
elevenprobs(X)
fitkienerX(X, algo = "reg", dgts = 3, probak = probak)
fitkienerX(X, algo = "estim", ord = 5, probak = probak, dgts = 3)
paramkienerX(X)
paramkienerX5(X)

### Compare the 12 values of paramkienerX(ord/row = 1:12) and paramkienerX (row 13)
compare <- function(ord, X) { paramkienerX(X, ord, algo = "estim", dgts = 13) }
rbind(t(sapply( 1:12, compare, X)), paramkienerX(X, algo = "reg", dgts = 13))

### Analyze DS in one step
t(sapply(DS, paramkienerX, algo = "reg", dgts = 13))
t(sapply(DS, paramkienerX, algo = "estim", dgts = 13))
paramkienerX(DS, algo = "reg", dgts = 13)
paramkienerX(DS, algo = "estim", dgts = 13)
system.time(fitk_rDS <- fitkienerX(DS, algo = "r", probak = pprobs2, dgts = 3))
system.time(fitk_eDS <- fitkienerX(DS, algo = "e", probak = pprobs2, dgts = 3))
fitk_rDS
fitk_eDS

### Subset rDS and eDS with exfit0,..,exfit7
fitk_rDS[,exfit4]
fitk_eDS[,exfit7]
fitkienerX(DS, algo = "e", probak = pprobs2, dgts = 3, exfitk = exfit7)

### Array (new example introduced in v1.5-1)
### Increase the number of cores and crash R.
## Not run:
arr <- array(rkiener1(3000), c(4,3,250))
paramkienerX7(arr, ncores = 2)
## paramkienerX7(arr, ncores = 2) - paramkienerX(arr, ncores = 2)
## End(Not run)

### End
```

getDSdata *Get DS Dataset*

Description

A function to extract the log-returns of 16 financial series and time series provided by the packages `datasets` (EuStockMarkets, sunspot.year) and `timeSeries` (USDCHF, MSFT, LPP2005REC). The 16 datasets are converted to a list of numeric without any reference to the original dates. This list is usually called `DS`, hence the name.

Usage

```
getDSdata()
```

Details

The dataset is usually created by the instruction `DS <-getDSdata()`. Then, it is used with a call to DS[[j]] with j in 1:16.

1. "USDCHF" (USDCHF, timeSeries)
2. "MSFT" (MSFT, timeSeries)
3. "DAX" (EuStockMarkets, datasets)
4. "SMI" (EuStockMarkets, datasets)
5. "CAC" (EuStockMarkets, datasets)
6. "FTSE" (EuStockMarkets, datasets)
7. "SBI" (LPP2005REC, timeSeries)
8. "SPI" (LPP2005REC, timeSeries)
9. "SII" (LPP2005REC, timeSeries)
10. "LMI" (LPP2005REC, timeSeries)
11. "MPI" (LPP2005REC, timeSeries)
12. "ALT" (LPP2005REC, timeSeries)
13. "LPP25" (LPP2005REC, timeSeries)
14. "LPP40" (LPP2005REC, timeSeries)
15. "LPP60" (LPP2005REC, timeSeries)
16. "sunspot" (sunspot.year, datasets)

Note that `sunspot.year` is regularly updated with each new version of `R`. The generated dataset is `logreturn(sunspot.year + 1000)`.

See Also

`EuStockMarkets`, `sunspot.year`, `TimeSeriesData`, `regkienerLX`, `fitkienerX`

Examples

```
require(timeSeries)

getDSdata
DS  <- getDSdata()
attributes(DS)
sapply(DS, length)
sapply(DS, head)
```

---

getnamesk *Generate a list of vectors of characters from a vector of probabilities*

Description

Generate vector of characters from a vector of probabilities, replacing `0.` by letters:

- `p.` : probability.
- `q.` : quantile.
- `VaR.` : Value-at-Risk, positive in most cases.
- `c.` : corrective tail coefficient = (q - m) / (q_logistic_function - m).
- `ltm.` : left tail mean (signed ES on the left tail, usually negative).
- `rtm.` : right tail mean (signed ES on the right tail, usually positive).
- `dtmq.` : (p<=0.5 left, p>0.5 right) tail mean minus quantile.
- `ES.` : expected shortfall, positive in most cases.
- `h.` : corrective ES = (ES - m) / (ES_logistic_function - m).
- `desv.` : ES - VaR, usually positive.
- `l.` : quantile of the tangent logistic function.
- `dl.` : quantile - quantile_logistic_function.
- `g.` : quantile of the Laplace-Gauss function.
- `dg.` : quantile - quantile_Laplace_Gauss_function.

, `q.`, `VaR.`, `c.`, `ltm.`, `rtm.`, `ES.`, `h.`, `l.`, `dl.`, `g.`, `dg.`. The result is a list of vectors.

Usage

```
getnamesk(probak = pprobs2, check = TRUE)

getnprobak(probak = pprobs2, check = TRUE)
```

Arguments

probak	a vector of ordered probabilities with 0 and 1 excluded.
check	boolean. Apply checkquantiles function.

See Also

Probabilities: pprobs0

Examples

```
getnamesk(pprobs1)
getnamesk(pprobs8)
```

---

kashp *Kashp Function*

Description

kashp, which stands for kappa times arc-sine-hyperbola-power is the nonlinear transformation of x at the heart of power hyperbolas, power hyperbolic functions and symmetric Kiener distributions. dkashp_dx is its derivative with respect to x. ashp is provided for convenience.

Usage

```
kashp(x, k = 1)

dkashp_dx(x, k = 1)

ashp(x, k = 1)
```

Arguments

x	a numeric value, vector or matrix.
k	a numeric value or vector, preferably strictly positive.

Details

ashp function is defined for x in (-Inf, +Inf) by:

$$ashp(x, k) = asinh(x/2/k)$$

kashp function is defined for x in (-Inf, +Inf) by:

$$kashp(x, k) = k * asinh(x/2/k)$$

dkashp_dx function is defined for x in (-Inf, +Inf) by:

$$dkashp_dx(x,k) = 1/sqrt(x*x/k/k+4) = 1/2/cosh(ashp(x,k))$$

If k is a vector, then the use of the function `outer` is recommanded.

The undesired case k=0 returns 0 for kashp and dkashp_dx, 1 for exphp, -Inf, NaN, +Inf for ashp.

See Also

The power hyperbolas and the power hyperbolic functions `exphp`.

Examples

```
require(graphics)

### Example 1
x    <- (-3:3)*3 ; names(x) <- x
kashp(x, k=2)
k    <- c(-2, 0, 1, 2, 3, 5, 10) ; names(k) <- k
outer(x, k, kashp)
outer(x, k, exphp)

### Example 2
xmin   <- -10
xd     <- 0.5
x      <- seq(xmin, -xmin, xd) ; names(x) <- x
k      <- c(0.6, 1, 1.5, 2, 3.2, 10) ; names(k) <- k
olty   <- c(2, 1, 2, 1, 2, 1, 1)
olwd   <- c(1, 1, 2, 2, 3, 4, 2)
ocol   <- c(2, 2, 4, 4, 3, 3, 1)
op     <- par(mfrow = c(2,2), mgp = c(1.5,0.8,0), mar = c(3,3,2,1))

Tkashp <- ts(cbind(outer(x, k, kashp), "x/2" = x/2), start = xmin, deltat = xd)
plot(Tkashp, plot.type = "single", ylim = c(-5, +5),
       lty = olty, lwd = olwd, col = ocol, xaxs = "i", yaxs = "i", xlab = "",
       ylab = "", main = "kashp(x, k)" )
legend("topleft", title = expression(kappa), legend = colnames(Tkashp),
       inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

Tdkashp <- ts(cbind(outer(x, k, dkashp_dx)), start = xmin, deltat = xd)
plot(Tdkashp, plot.type = "single", ylim = c(0, 0.8),
       lty = olty, lwd = olwd, col = ocol, xaxs = "i", yaxs = "i", xlab = "",
       ylab = "", main="dkashp_dx(x, k)" )
legend("topleft", title = expression(kappa), legend = colnames(Tdkashp),
       inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

Tashp <- ts(cbind(outer(x, k, ashp), "x/2" = x/2), start = xmin, deltat = xd)
plot(Tashp, plot.type = "single", ylim = c(-5, +5),
       lty = olty, lwd = olwd, col = ocol, xaxs = "i", yaxs = "i", xlab = "",
       ylab = "", main = "ashp(x, k)" )
legend("topleft", title = expression(kappa), legend = colnames(Tashp),
```

```
       inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )
### End example 2
```

---

kiener1 *Symmetric Kiener Distribution K1*

Description

Density, distribution function, quantile function, random generation, value-at-risk, expected shortfall (+ signed left/right tail mean) and additional formulae for symmetric Kiener distribution K1. This distribution is similar to the power hyperbola logistic distribution but with additional parameters for location (m) and scale (g).

Usage

```
dkiener1(x, m = 0, g = 1, k = 3.2, log = FALSE)

pkiener1(q, m = 0, g = 1, k = 3.2, lower.tail = TRUE, log.p = FALSE)

qkiener1(p, m = 0, g = 1, k = 3.2, lower.tail = TRUE, log.p = FALSE)

rkiener1(n, m = 0, g = 1, k = 3.2)

dpkiener1(p, m = 0, g = 1, k = 3.2, log = FALSE)

dqkiener1(p, m = 0, g = 1, k = 3.2, log = FALSE)

lkiener1(x, m = 0, g = 1, k = 3.2)

dlkiener1(lp, m = 0, g = 1, k = 3.2, log = FALSE)

qlkiener1(lp, m = 0, g = 1, k = 3.2, lower.tail = TRUE)

varkiener1(p, m = 0, g = 1, k = 3.2, lower.tail = TRUE, log.p = FALSE)

ltmkiener1(p, m = 0, g = 1, k = 3.2, lower.tail = TRUE, log.p = FALSE)

rtmkiener1(p, m = 0, g = 1, k = 3.2, lower.tail = TRUE, log.p = FALSE)

dtmqkiener1(p, m = 0, g = 1, k = 3.2, lower.tail = TRUE, log.p = FALSE)

eskiener1(p, m = 0, g = 1, k = 3.2, lower.tail = TRUE, log.p = FALSE,
  signedES = FALSE)
```

Arguments

x	vector of quantiles.
m	numeric. The median.
g	numeric. The scale parameter, preferably strictly positive.
k	numeric. The tail parameter, preferably strictly positive.
log	logical. If TRUE, densities are given in log scale.
q	vector of quantiles.
lower.tail	logical. If TRUE, use p. If FALSE, use 1-p.
log.p	logical. If TRUE, probabilities p are given as log(p).
p	vector of probabilities.
n	number of observations. If length(n) > 1, the length is taken to be the number required.
lp	vector of logit of probabilities.
signedES	logical. FALSE (default) returns positive numbers for left and right tails. TRUE returns negative number (= ltmkiener1) for left tail and positive number (= rtmkiener1) for right tail.

Details

Kiener distributions use the following parameters, some of them being redundant. See aw2k and pk2pk for the formulas and the conversion between parameters:

- m (mu) is the median of the distribution,.
- g (gamma) is the scale parameter.
- a (alpha) is the left tail parameter.
- k (kappa) is the harmonic mean of a and w and describes a global tail parameter.
- w (omega) is the right tail parameter.
- d (delta) is the distortion parameter.
- e (epsilon) is the eccentricity parameter.

Kiener distributions K1(m,g,k,...) describe distributions with symmetric left and right fat tails with tail parameter k. This parameter is the power exponent mentionned in Pareto formula and Karamata theorems.

m is the median of the distribution. g is the scale parameter and the inverse of the density at the median: $g = 1/8/f(m)$. As a first estimate, it is approximatively one fourth of the standard deviation $g \approx \sigma/4$ but is independant from it.

dkiener1 function is defined for x in (-Inf, +Inf) by:

$$dkiener1(x, m, g, k) = 1/4/g/cosh(ashp((x-m)/g, k))/(1 + cosh(kashp((x-m)/g, k)))$$

pkiener1 function is defined for q in (-Inf, +Inf) by:

$$pkiener1(q, m, g, k) = 1/(1 + exp(-kashp((q-m)/g, k)))$$

qkiener1 function is defined for p in (0, 1) by:

$$qkiener1(p, m, g, k) = m + 2 * g * k * sinh(logit(p)/k)$$

rkiener1 generates n random quantiles.

In addition to the classical d, p, q, r functions, the prefixes dp, dq, l, dl, ql are also provided.

dpkiener1 is the density function calculated from the probability p. It is defined for p in (0, 1) by:

$$dpkiener1(p, m, g, k) = p * (1 - p)/2/g/cosh(logit(p)/k)$$

dqkiener1 is the derivate of the quantile function calculated from the probability p. It is defined for p in (0, 1) by:

$$dqkiener1(p, m, g, k) = 2 * g/p/(1 - p) * cosh(logit(p)/k)$$

lkiener1 function is equivalent to kashp function but with additional parameters m and g. Being computed from the x (or q) vector, it can be compared to the logit of the empirical probability logit(p) through a nonlinear regression with ordinary or weighted least squares to estimate the distribution parameters. It is defined for x in (-Inf, +Inf) by:

$$lkiener1(x, m, g, k) = kashp((x - m)/g, k)$$

dlkiener1 is the density function calculated from the logit of the probability lp = logit(p). It is defined for lp in (-Inf, +Inf) by:

$$dlkiener1(lp, m, g, k) = p * (1 - p)/2/g/cosh(lp/k)$$

qlkiener1 is the quantile function calculated from the logit of the probability lp = logit(p). It is defined for lp in (-Inf, +Inf) by:

$$qlkiener1(lp, m, g, k) = m + g * k * 2 * sinh(lp/k)$$

varkiener1 designates the Value a-risk and turns negative numbers into positive numbers with the following rule:

$$varkiener1 < -if(p <= 0.5)(-qkiener1)else(qkiener1)$$

Usual values in finance are p = 0.01, p = 0.05, p = 0.95 and p = 0.99. lower.tail = FALSE uses 1-p rather than p.

ltmkiener1, rtmkiener1 and eskiener1 are respectively the left tail mean, the right tail mean and the expected shortfall of the distribution (sometimes called average VaR, conditional VaR or tail VaR). Left tail mean is the integrale from -Inf to p of the quantile function qkiener1 divided by p. Right tail mean is the integrale from p to +Inf of the quantile function qkiener1 divided by 1-p. Expected shortfall turns negative numbers into positive numbers with the following rule:

$$eskiener1 < -if(p <= 0.5)(-ltmkiener1)else(rtmkiener1)$$

Usual values in finance are p = 0.01, p = 0.025, p = 0.975 and p = 0.99. lower.tail = FALSE uses 1-p rather than p.

dtmqkiener1 is the difference between the left tail mean and the quantile when (p <= 0.5) and the difference between the right tail mean and the quantile when (p > 0.5). It is in quantile unit and is an indirect measure of the tail curvature.

References

P. Kiener, Explicit models for bilateral fat-tailed distributions and applications in finance with the package FatTailsR, 8th R/Rmetrics Workshop and Summer School, Paris, 27 June 2014. Download it from: https://www.inmodelia.com/exemples/2014-0627-Rmetrics-Kiener-en.pdf

P. Kiener, Fat tail analysis and package FatTailsR, 9th R/Rmetrics Workshop and Summer School, Zurich, 27 June 2015. Download it from: https://www.inmodelia.com/exemples/2015-0627-Rmetrics-Kiener-en.pdf

C. Acerbi, D. Tasche, Expected shortfall: a natural coherent alternative to Value at Risk, 9 May 2001. Download it from: https://www.bis.org/bcbs/ca/acertasc.pdf

See Also

Power hyperbola logistic distribution logishp, asymmetric Kiener distributions K2, K3, K4 and K7 kiener2, kiener3, kiener4, kiener7, regression function regkienerLX.

Examples

```
require(graphics)

### Example 1
pp <- c(ppoints(11, a = 1), NA, NaN) ; pp
qkiener1(p = pp, k = 4)


### Example 2: Try various value of k = 1.5, 3, 5, 10
k       <- 5  # 1.5, 3, 5, 10
set.seed(2014)
mainTC  <- paste("qkiener1(p, m = 0, g = 1, k = ", k, ")")
mainsum <- paste("cumulated qkiener1(p, m = 0, g = 1, k = ", k, ")")
T       <- 500
C       <- 4
TC      <- qkiener1(p = runif(T*C), m = 0, g = 1, k = k)
matTC   <- matrix(TC, nrow = T, ncol = C, dimnames = list(1:T, letters[1:C]))
head(matTC)
plot.ts(matTC, main = mainTC)
#
matsum  <- apply(matTC, MARGIN=2, cumsum)
head(matsum)
plot.ts(matsum, plot.type = "single", main = mainsum)
### End example 2


### Example 3 (four plots: probability, density, logit, logdensity)
x  <- q  <- seq(-15, 15, length.out=101)
k     <- c(0.6, 1, 1.5, 2, 3.2, 10) ; names(k) <- k ; k
olty  <- c(2, 1, 2, 1, 2, 1, 1)
olwd  <- c(1, 1, 2, 2, 3, 3, 2)
ocol  <- c(2, 2, 4, 4, 3, 3, 1)
lleg  <- c("logit(0.999) = 6.9", "logit(0.99)   = 4.6", "logit(0.95)   = 2.9",
           "logit(0.50)   = 0", "logit(0.05)   = -2.9", "logit(0.01)   = -4.6",
```

```
           "logit(0.001) = -6.9  ")
op    <- par(mfrow=c(2,2), mgp=c(1.5,0.8,0), mar=c(3,3,2,1))

plot(x, plogis(x, scale = 2), type = "b", lwd = 2, ylim = c(0, 1),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "pkiener1(q, m, g, k)")
for (i in 1:length(k)) lines(x, pkiener1(x, k = k[i]),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", title = expression(kappa), legend = c(k, "logistic"),
       cex = 0.7, inset = 0.02, lty = olty, lwd = olwd, col = ocol )

plot(x, dlogis(x, scale = 2), type = "b", lwd = 2, ylim = c(0, 0.14),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "", main = "dkiener1(x, m, g, k)" )
for (i in 1:length(k)) lines(x, dkiener1(x, k = k[i]),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topright", title = expression(kappa), legend = c(k, "logistic"),
       cex = 0.7, inset = 0.02, lty = olty, lwd = olwd, col = ocol )

plot(x, x/2, type = "b", lwd = 2, ylim = c(-7.5, 7.5), yaxt="n", xaxs = "i",
     yaxs = "i", xlab = "", ylab = "", main = "logit(pkiener1(q, m, g, k))")
axis(2, las=1, at=c(-6.9, -4.6, -2.9, 0, 2.9, 4.6, 6.9) )
for (i in 1:length(k)) lines(x, lkiener1(x, k = k[i]),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
lines(x, logit(pnorm(x, 0, 3.192)), type="l", lty=1, lwd=3, col=7) # erfx
legend("topleft", legend = lleg, cex = 0.7, inset = 0.02 )
legend("bottomright", title = expression(kappa),
       legend = c(k, "logistic", "Gauss"), cex = 0.7, inset = 0.02,
       lty = c(olty, 1), lwd = c(olwd, 3), col = c(ocol , 7) )

plot(x, log(dlogis(x, scale = 2)), lwd = 2, type = "b", ylim = c(-8, -1.5),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "", main = "log(dkiener1(x, m, g, k))")
for (i in 1:length(k)) lines(x, log(dkiener1(x, k = k[i])),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
lines(x, dnorm(x, 0, 3.192, log = TRUE), type = "l", lty = 1, lwd = 3, col = 7)
legend("bottom", title = expression(kappa), legend = c(k, "logistic", "Gauss"),
       cex = 0.7, inset = 0.02, lty = c(olty, 1), lwd = c(olwd, 3), col = c(ocol , 7) )
### End example 3


### Example 4 (four plots: quantile, derivate, density and quantiles from p)
p   <- ppoints(199, a=0)
k   <- c(0.6, 1, 1.5, 2, 3.2, 10) ; names(k) <- k ; k
op  <- par(mfrow=c(2,2), mgp=c(1.5,0.8,0), mar=c(3,3,2,1))
plot(p, qlogis(p, scale = 2), type = "o", lwd = 2, ylim = c(-15, 15),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "qkiener1(p, m, g, k)")
for (i in 1:length(k)) lines(p, qkiener1(p, k = k[i]),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", title = expression(kappa), legend = c(k, "qlogis(x/2)"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

plot(p, 2/p/(1-p), type = "o", lwd = 2, xlim = c(0, 1), ylim = c(0, 100),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "",
```

```
     main = "dqkiener1(p, m, g, k)")
for (i in 1:length(k)) lines(p, dqkiener1(p, k = k[i]),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("top", title = expression(kappa), legend = c(k, "p*(1-p)/2"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

plot(qlogis(p, scale = 2), p*(1-p)/2, type = "o", lwd = 2, xlim = c(-15, 15),
     ylim = c(0, 0.14), xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "qkiener1, dpkiener1(p, m, g, k)")
for (i in 1:length(k)) lines(qkiener1(p, k = k[i]), dpkiener1(p, k = k[i]),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", title = expression(kappa), legend = c(k, "p*(1-p)/2"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )
### End example 4


### Example 5 (q and VaR, ltm, rtm, and ES)
pp <- c(0.001, 0.0025, 0.005, 0.01, 0.025, 0.05,
        0.10, 0.20, 0.35, 0.5, 0.65, 0.80, 0.90,
        0.95, 0.975, 0.99, 0.995, 0.9975, 0.999)
m <- -10 ; g <- 1 ; k <- 4
round(c(m = m, g = g, a = k, k = k, w = k, d = 0, e = 0), 2)
plot(qkiener1(pp, m, g, k), pp, type = "b")
round(cbind(p = pp, "1-p" = 1-pp,
q   =   qkiener1(pp, m, g, k),
ltm = ltmkiener1(pp, m, g, k),
rtm = rtmkiener1(pp, m, g, k),
es  =  eskiener1(pp, m, g, k),
VaR = varkiener1(pp, m, g, k)), 4)
round(kmean(c(m, g, k), model = "K1"), 4) # limit value of ltm, rtm
round(cbind(p = pp, "1-p" = 1-pp,
q   =   qkiener1(pp, m, g, k, lower.tail = FALSE),
ltm = ltmkiener1(pp, m, g, k, lower.tail = FALSE),
rtm = rtmkiener1(pp, m, g, k, lower.tail = FALSE),
es  =  eskiener1(pp, m, g, k, lower.tail = FALSE),
VaR = varkiener1(pp, m, g, k, lower.tail = FALSE)), 4)
### End example 5
```

---

kiener2 *Asymmetric Kiener Distribution K2*

Description

Density, distribution function, quantile function, random generation, value-at-risk, expected shortfall (+ signed left/right tail mean) and additional formulae for asymmetric Kiener distribution K2.

Usage

```
dkiener2(x, m = 0, g = 1, a = 3.2, w = 3.2, log = FALSE)

pkiener2(q, m = 0, g = 1, a = 3.2, w = 3.2, lower.tail = TRUE, log.p = FALSE)

qkiener2(p, m = 0, g = 1, a = 3.2, w = 3.2, lower.tail = TRUE, log.p = FALSE)

rkiener2(n, m = 0, g = 1, a = 3.2, w = 3.2)

dpkiener2(p, m = 0, g = 1, a = 3.2, w = 3.2, log = FALSE)

dqkiener2(p, m = 0, g = 1, a = 3.2, w = 3.2, log = FALSE)

lkiener2(x, m = 0, g = 1, a = 3.2, w = 3.2)

dlkiener2(lp, m = 0, g = 1, a = 3.2, w = 3.2, log = FALSE)

qlkiener2(lp, m = 0, g = 1, a = 3.2, w = 3.2, lower.tail = TRUE)

varkiener2(p, m = 0, g = 1, a = 3.2, w = 3.2, lower.tail = TRUE, log.p = FALSE)

ltmkiener2(p, m = 0, g = 1, a = 3.2, w = 3.2, lower.tail = TRUE, log.p = FALSE)

rtmkiener2(p, m = 0, g = 1, a = 3.2, w = 3.2, lower.tail = TRUE, log.p = FALSE)

dtmqkiener2(p, m = 0, g = 1, a = 3.2, w = 3.2, lower.tail = TRUE,
  log.p = FALSE)

eskiener2(p, m = 0, g = 1, a = 3.2, w = 3.2, lower.tail = TRUE,
  log.p = FALSE, signedES = FALSE)
```

Arguments

x	vector of quantiles.
m	numeric. The median.
g	numeric. The scale parameter, preferably strictly positive.
a	numeric. The left tail parameter, preferably strictly positive.
w	numeric. The right tail parameter, preferably strictly positive.
log	logical. If TRUE, densities are given in log scale.
q	vector of quantiles.
lower.tail	logical. If TRUE, use p. If FALSE, use 1-p.
log.p	logical. If TRUE, probabilities p are given as log(p).
p	vector of probabilities.
n	number of observations. If length(n) > 1, the length is taken to be the number required.

lp	vector of logit of probabilities.
signedES	logical. FALSE (default) returns positive numbers for left and right tails. TRUE returns negative number (= ltmkiener4) for left tail and positive number (= rtmkiener4) for right tail.

Details

Kiener distributions use the following parameters, some of them being redundant. See aw2k and pk2pk for the formulas and the conversion between parameters:

- m (mu) is the median of the distribution,.
- g (gamma) is the scale parameter.
- a (alpha) is the left tail parameter.
- k (kappa) is the harmonic mean of a and w and describes a global tail parameter.
- w (omega) is the right tail parameter.
- d (delta) is the distortion parameter.
- e (epsilon) is the eccentricity parameter.

Kiener distributions K2(m,g,a,w) are distributions with asymmetrical left and right fat tails described by the parameters a (alpha) for the left tail and w (omega) for the right tail. These parameters correspond to the power exponent that appear in Pareto formula and Karamata theorems.

As a and w are highly correlated, the use of Kiener distributions (K3(...,k,d) K4 (K4(...,k,e) is an alternate solution.

m is the median of the distribution. g is the scale parameter and the inverse of the density at the median: $g = 1/8/f(m)$. As a first estimate, it is approximatively one fourth of the standard deviation $g \approx \sigma/4$ but is independant from it.

The d, p functions have no explicit forms. They are provided here for convenience. They are estimated from a reverse optimization on the quantile function and can be (very) slow, depending the number of points to estimate. We recommand to use the quantile function as far as possible. WARNING: Results may become inconsistent when a or w are smaller than 1. Hopefully, this case seldom happens in finance.

qkiener2 function is defined for p in (0, 1) by:

$$qkiener2(p, m, g, a, w) = m + g * k * (-exp(-logit(p)/a) + exp(logit(p)/w))$$

where k is the harmonic mean of the tail parameters a and w calculated by $k = aw2k(a, w)$.

rkiener2 generates n random quantiles.

In addition to the classical d, p, q, r functions, the prefixes dp, dq, l, dl, ql are also provided.

dpkiener2 is the density function calculated from the probability p. It is defined for p in (0, 1) by:

$$dpkiener2(p, m, g, a, w) = p * (1 - p)/k/g/(exp(-logit(p)/a)/a + exp(logit(p)/w)/w$$

dqkiener2 is the derivate of the quantile function calculated from the probability p. It is defined for p in (0, 1) by:

$$dqkiener2(p, m, g, a, w) = k * g/p/(1 - p) * (exp(-logit(p)/a)/a + exp(logit(p)/w)/w)$$

`lkiener2` function is estimated from a reverse optimization and can be (very) slow depending the number of points to estimate. Initialization is done by assuming a symmetric distribution `lkiener1` around the harmonic mean `k`, then optimization is performed to take into account the true values `a` and `w`. The result can be then compared to the empirical probability logit(p). WARNING: Results may become inconsistent when `a` or `w` are smaller than 1. Hopefully, this case seldom happens in finance.

`dlkiener2` is the density function calculated from the logit of the probability lp = logit(p). it is defined for lp in (-Inf, +Inf) by:

$$dlkiener2(lp, m, g, a, w) = p * (1 - p)/k/g/(exp(-lp/a)/a + exp(lp/w)/w)$$

`qlkiener2` is the quantile function calculated from the logit of the probability. It is defined for lp in (-Inf, +Inf) by:

$$qlkiener2(lp, m, g, a, w) = m + g * k * (-exp(-lp/a) + exp(lp/w))$$

`varkiener2` designates the Value a-risk and turns negative numbers into positive numbers with the following rule:

$$varkiener2 < -if(p <= 0.5) - qkiener2 elseqkiener2$$

Usual values in finance are `p = 0.01`, `p = 0.05`, `p = 0.95` and `p = 0.99`. `lower.tail = FALSE` uses `1-p` rather than p.

`ltmkiener2`, `rtmkiener2` and `eskiener2` are respectively the left tail mean, the right tail mean and the expected shortfall of the distribution (sometimes called average VaR, conditional VaR or tail VaR). Left tail mean is the integrale from `-Inf` to p of the quantile function `qkiener2` divided by p. Right tail mean is the integrale from p to `+Inf` of the quantile function `qkiener2` divided by 1-p. Expected shortfall turns negative numbers into positive numbers with the following rule:

$$eskiener2 < -if(p <= 0.5) - ltmkiener2 elsertmkiener2$$

Usual values in finance are `p = 0.01`, `p = 0.025`, `p = 0.975` and `p = 0.99`. `lower.tail = FALSE` uses `1-p` rather than p.

`dtmqkiener2` is the difference between the left tail mean and the quantile when (p <= 0.5) and the difference between the right tail mean and the quantile when (p > 0.5). It is in quantile unit and is an indirect measure of the tail curvature.

References

P. Kiener, Explicit models for bilateral fat-tailed distributions and applications in finance with the package FatTailsR, 8th R/Rmetrics Workshop and Summer School, Paris, 27 June 2014. Download it from: https://www.inmodelia.com/exemples/2014-0627-Rmetrics-Kiener-en.pdf

P. Kiener, Fat tail analysis and package FatTailsR, 9th R/Rmetrics Workshop and Summer School, Zurich, 27 June 2015. Download it from: https://www.inmodelia.com/exemples/2015-0627-Rmetrics-Kiener-en.pdf

C. Acerbi, D. Tasche, Expected shortfall: a natural coherent alternative to Value at Risk, 9 May 2001. Download it from: https://www.bis.org/bcbs/ca/acertasc.pdf

See Also

Symmetric Kiener distribution K1 kiener1, asymmetric Kiener distributions K3, K4 and K7 kiener3, kiener4, kiener7, conversion functions aw2k, estimation function fitkienerX, regression function regkienerLX.

Examples

```
require(graphics)

### Example 1
pp <- c(ppoints(11, a = 1), NA, NaN) ; pp
lp <- logit(pp) ; lp
qkiener2(  p = pp, m = 2, g = 1.5, a = 4, w = 6)
qkiener2(  p = pp, m = 2, g = 1.5, a = 4, w = 6)
qlkiener2(lp = lp, m = 2, g = 1.5, a = 4, w = 6)
dpkiener2( p = pp, m = 2, g = 1.5, a = 4, w = 6)
dlkiener2(lp = lp, m = 2, g = 1.5, a = 4, w = 6)
dqkiener2( p = pp, m = 2, g = 1.5, a = 4, w = 6)


### Example 2
a       <- 6
w       <- 4
set.seed(2014)
mainTC  <- paste("qkiener2(p, m = 0, g = 1, a = ", a, ", w = ", w, ")")
mainsum <- paste("cumulated qkiener2(p, m = 0, g = 1, a = ", a, ", w = ", w, ")")
T       <- 500
C       <- 4
TC      <- qkiener2(p = runif(T*C), m = 0, g = 1, a = a, w = w)
matTC   <- matrix(TC, nrow = T, ncol = C, dimnames = list(1:T, letters[1:C]))
head(matTC)
plot.ts(matTC, main = mainTC)
#
matsum  <- apply(matTC, MARGIN=2, cumsum)
head(matsum)
plot.ts(matsum, plot.type = "single", main = mainsum)
### End example 2


### Example 3 (four plots: probability, density, logit, logdensity)
x     <- q  <- seq(-15, 15, length.out=101)
w     <- c(0.6, 1, 1.5, 2, 3.2, 10) ; names(w) <- w
olty  <- c(2, 1, 2, 1, 2, 1, 1)
olwd  <- c(1, 1, 2, 2, 3, 3, 2)
ocol  <- c(2, 2, 4, 4, 3, 3, 1)
lleg  <- c("logit(0.999) = 6.9", "logit(0.99)   = 4.6", "logit(0.95)   = 2.9",
           "logit(0.50)   = 0", "logit(0.05)   = -2.9", "logit(0.01)   = -4.6",
           "logit(0.001) = -6.9  ")
op    <- par(mfrow=c(2,2), mgp=c(1.5,0.8,0), mar=c(3,3,2,1))

plot(x, plogis(x, scale = 2), type = "n", lwd = 2, ylim = c(0, 1),
```

```
     xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "pkiener2(q, m, g, a=2, w=...)")
for (i in 1:length(w)) lines(x, pkiener2(x, a = 2, w = w[i]),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", title = expression(omega), legend = c(w),
       cex = 0.7, inset = 0.02, lty = olty, lwd = olwd, col = ocol )

plot(x, dlogis(x, scale = 2), type = "n", lwd = 2, ylim = c(0, 0.17),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "dkiener2(q, m, g, a=2, w=...)")
for (i in 1:length(w)) lines(x, dkiener2(x, a = 2, w = w[i]),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topright", title = expression(omega), legend = c(w),
       cex = 0.7, inset = 0.02, lty = olty, lwd = olwd, col = ocol )

plot(x, x/2, type = "n", lwd = 1, ylim = c(-7.5, 7.5), yaxt="n", xaxs = "i",
     yaxs = "i", xlab = "", ylab = "",
     main = "logit(pkiener2(q, m, g, a=2, w=...))")
axis(2, las=1, at=c(-6.9, -4.6, -2.9, 0, 2.9, 4.6, 6.9) )
for (i in 1:length(w)) lines(x, lkiener2(x, a = 2, w = w[i]),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", legend = lleg, cex = 0.7, inset = 0.02 )
legend("bottomright", title = expression(omega), legend = c(w),
       cex = 0.7, inset = 0.02, lty = c(olty), lwd = c(olwd), col = c(ocol) )

plot(x, dlogis(x, scale = 2, log=TRUE), type = "n", lwd = 2, ylim = c(-8, -1.5),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "log(dkiener2(q, m, g, a=2, w=...))")
for (i in 1:length(w)) lines(x, dkiener2(x, a = 2, w = w[i], log=TRUE),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("bottom", title = expression(omega), legend = c(w),
       cex = 0.7, inset = 0.02, lty = olty, lwd = olwd, col = ocol )
### End example 3


### Example 4 (four plots: quantile, derivate, density and quantiles from p)
p     <- ppoints(199, a=0)
w     <- c(0.6, 1, 1.5, 2, 3.2, 10) ; names(w) <- w ; w
op    <- par(mfrow=c(2,2), mgp=c(1.5,0.8,0), mar=c(3,3,2,1))

plot(p, qlogis(p, scale = 2), type = "l", lwd = 2, xlim = c(0, 1),
     ylim = c(-15, 15), xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "qkiener2(p, m, g, a=2, w=...)")
for (i in 1:length(w)) lines(p, qkiener2(p, a = 2, w = w[i]),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", title = expression(omega), legend = c(w, "qlogis(x/2)"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

plot(p, 2/p/(1-p), type = "l", lwd = 2, xlim = c(0, 1), ylim = c(0, 100),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "dqkiener2(p, m, g, a=2, w=...)")
for (i in 1:length(w)) lines(p, dqkiener2(p, a = 2, w = w[i]),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
```

```
legend("top", title = expression(omega), legend = c(w, "p*(1-p)/2"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

plot(qlogis(p, scale = 2), p*(1-p)/2, type = "l", lwd = 2, xlim = c(-15, 15),
     ylim = c(0, 0.18), xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "qkiener2, dpkiener2(p, m, g, a=2, w=...)")
for (i in 1:length(w)) {
     lines(qkiener2(p, a = 2, w = w[i]), dpkiener2(p, a = 2, w = w[i]),
           lty = olty[i], lwd = olwd[i], col = ocol[i] ) }
legend("topleft", title = expression(omega), legend = c(w, "p*(1-p)/2"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

plot(qlogis(p, scale = 2), p, type = "l", lwd = 2, xlim = c(-15, 15),
     ylim = c(0, 1), xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "inverse axis qkiener2(p, m, g, a=2, w=...)")
for (i in 1:length(w)) lines(qkiener2(p, a = 2, w = w[i]), p,
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", title = expression(omega), legend = c(w, "qlogis(x/2)"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )
### End example 4


### Example 5 (q and VaR, ltm, rtm, and ES)
pp <- c(0.001, 0.0025, 0.005, 0.01, 0.025, 0.05,
        0.10, 0.20, 0.35, 0.5, 0.65, 0.80, 0.90,
        0.95, 0.975, 0.99, 0.995, 0.9975, 0.999)
m <- -10 ; g <- 1 ; a <- 5 ; w = 3
k <- aw2k(a, w) ; d <- aw2d(a, w) ; e <- aw2e(a, w)
round(c(m = m, g = g, a = a, k = k, w = w, d = d, e = e), 2)
plot(qkiener2(pp, m, g, a, w), pp, type = "b")
round(cbind(p = pp, "1-p" = 1-pp,
q   =   qkiener2(pp, m, g, a, w),
ltm = ltmkiener2(pp, m, g, a, w),
rtm = rtmkiener2(pp, m, g, a, w),
ES  =  eskiener2(pp, m, g, a, w),
VaR = varkiener2(pp, m, g, a, w)), 4)
round(kmean(c(m, g, a, w), model = "K2"), 4) # limit value for ltm and rtm
round(cbind(p = pp, "1-p" = 1-pp,
q   =   qkiener2(pp, m, g, a, w, lower.tail = FALSE),
ltm = ltmkiener2(pp, m, g, a, w, lower.tail = FALSE),
rtm = rtmkiener2(pp, m, g, a, w, lower.tail = FALSE),
ES  =  eskiener2(pp, m, g, a, w, lower.tail = FALSE),
VaR = varkiener2(pp, m, g, a, w, lower.tail = FALSE)), 4)
### End example 5
```

---

kiener3 *Asymmetric Kiener Distribution K3*

Description

Density, distribution function, quantile function, random generation, value-at-risk, expected shortfall (+ signed left/right tail mean) and additional formulae for asymmetric Kiener distribution K3.

Usage

```
dkiener3(x, m = 0, g = 1, k = 3.2, d = 0, log = FALSE)

pkiener3(q, m = 0, g = 1, k = 3.2, d = 0, lower.tail = TRUE, log.p = FALSE)

qkiener3(p, m = 0, g = 1, k = 3.2, d = 0, lower.tail = TRUE, log.p = FALSE)

rkiener3(n, m = 0, g = 1, k = 3.2, d = 0)

dpkiener3(p, m = 0, g = 1, k = 3.2, d = 0, log = FALSE)

dqkiener3(p, m = 0, g = 1, k = 3.2, d = 0, log = FALSE)

lkiener3(x, m = 0, g = 1, k = 3.2, d = 0)

dlkiener3(lp, m = 0, g = 1, k = 3.2, d = 0, log = FALSE)

qlkiener3(lp, m = 0, g = 1, k = 3.2, d = 0, lower.tail = TRUE)

varkiener3(p, m = 0, g = 1, k = 3.2, d = 0, lower.tail = TRUE, log.p = FALSE)

ltmkiener3(p, m = 0, g = 1, k = 3.2, d = 0, lower.tail = TRUE, log.p = FALSE)

rtmkiener3(p, m = 0, g = 1, k = 3.2, d = 0, lower.tail = TRUE, log.p = FALSE)

dtmqkiener3(p, m = 0, g = 1, k = 3.2, d = 0, lower.tail = TRUE, log.p = FALSE)

eskiener3(p, m = 0, g = 1, k = 3.2, d = 0, lower.tail = TRUE,
  log.p = FALSE, signedES = FALSE)
```

Arguments

x	vector of quantiles.
m	numeric. The median.
g	numeric. The scale parameter, preferably strictly positive.
k	numeric. The tail parameter, preferably strictly positive.
d	numeric. The distortion parameter between left and right tails.
log	logical. If TRUE, densities are given in log scale.
q	vector of quantiles.
lower.tail	logical. If TRUE, use p. If FALSE, use 1-p.
log.p	logical. If TRUE, probabilities p are given as log(p).

p	vector of probabilities.
n	number of observations. If length(n) > 1, the length is taken to be the number required.
lp	vector of logit of probabilities.
signedES	logical. FALSE (default) returns positive numbers for left and right tails. TRUE returns negative number (= ltmkiener3) for left tail and positive number (= rtmkiener3) for right tail.

Details

Kiener distributions use the following parameters, some of them being redundant. See aw2k and pk2pk for the formulas and the conversion between parameters:

- m (mu) is the median of the distribution,.
- g (gamma) is the scale parameter.
- a (alpha) is the left tail parameter.
- k (kappa) is the harmonic mean of a and w and describes a global tail parameter.
- w (omega) is the right tail parameter.
- d (delta) is the distortion parameter.
- e (epsilon) is the eccentricity parameter.

Kiener distributions K3(m,g,k,d,...) are distributions with asymmetrical left and right fat tails described by a global tail parameter k and a distortion parameter d.

Distributions K3 (kiener3) with parameters k (kappa) and d (delta) and distributions K4 (kiener4) with parameters k (kappa) and e (epsilon)) have been created to disantangle the parameters a (alpha) and w (omega) of distributions of distribution K2 (kiener2). The tiny difference between distributions K3 and K4 ($d = e/k$) has not yet been fully evaluated. Both should be tested at that moment.

k is the harmonic mean of a and w and represents a global tail parameter.

d is a distortion parameter between the left tail parameter a and the right tail parameter w. It verifies the inequality: $-k < d < k$ (whereas e of distribution K4 verifies $-1 < e < 1$). The conversion functions (see aw2k) are:

$$1/k = (1/a + 1/w)/2$$

$$d = (-1/a + 1/w)/2$$

$$1/a = 1/k - d$$

$$1/w = 1/k + d$$

d (and e) should be of the same sign than the skewness. A negative value $d < 0$ implies $a < w$ and indicates a left tail heavier than the right tail. A positive value $d > 0$ implies $a > w$ and a right tail heavier than the left tail.

m is the median of the distribution. g is the scale parameter and the inverse of the density at the median: $g = 1/8/f(m)$. As a first estimate, it is approximatively one fourth of the standard deviation $g \approx \sigma/4$ but is independant from it.

The d, p functions have no explicit forms. They are provided here for convenience. They are estimated from a reverse optimization on the quantile function and can be (very) slow, depending the number of points to estimate. We recommand to use the quantile function as far as possible. WARNING: Results may become inconsistent when k is smaller than 1 or for very large absolute values of d. Hopefully, this case seldom happens in finance.

`qkiener3` function is defined for p in (0, 1) by:

$$qkiener3(p, m, g, k, d) = m + 2 * g * k * sinh(logit(p)/k) * exp(d * logit(p))$$

`rkiener3` generates n random quantiles.

In addition to the classical d, p, q, r functions, the prefixes dp, dq, l, dl, ql are also provided.

`dpkiener3` is the density function calculated from the probability p. The formula is adapted from distribution K2. It is defined for p in (0, 1) by:

$$dpkiener3(p, m, g, k, d) = p * (1 - p)/k/g/(exp(-logit(p)/a)/a + exp(logit(p)/w)/w$$

with a and w defined from k and d with the formula presented above.

`dqkiener3` is the derivate of the quantile function calculated from the probability p. The formula is adapted from distribution K2. It is defined for p in (0, 1) by:

$$dqkiener3(p, m, g, k, d) = k * g/p/(1 - p) * (exp(-logit(p)/a)/a + exp(logit(p)/w)/w)$$

with a and w defined above.

`lkiener3` function is estimated from a reverse optimization and can be (very) slow depending the number of points to estimate. Initialization is done with a symmetric distribution `lkiener1` of parameter k (thus $d = 0$). Then optimization is performed to take into account the true value of d. The results can then be compared to the empirical probability logit(p). WARNING: Results may become inconsistent when k is smaller than 1 or for very large absolute values of d. Hopefully, this case seldom happens in finance.

`dlkiener3` is the density function calculated from the logit of the probability lp = logit(p). The formula is adapted from distribution K2. it is defined for lp in (-Inf, +Inf) by:

$$dlkiener3(lp, m, g, k, d) = p * (1 - p)/k/g/(exp(-lp/a)/a + exp(lp/w)/w)$$

with a and w defined above.

`qlkiener3` is the quantile function calculated from the logit of the probability. It is defined for lp in (-Inf, +Inf) by:

$$qlkiener3(lp, m, g, k, d) = m + 2 * g * k * sinh(lp/k) * exp(d * lp)$$

`varkiener3` designates the Value a-risk and turns negative numbers into positive numbers with the following rule:

$$varkiener3 < -if(p <= 0.5) - qkiener3 else qkiener3$$

Usual values in finance are p = `0.01`, p = `0.05`, p = `0.95` and p = `0.99`. `lower.tail` = `FALSE` uses `1-p` rather than p.

`ltmkiener3`, `rtmkiener3` and `eskiener3` are respectively the left tail mean, the right tail mean and the expected shortfall of the distribution (sometimes called average VaR, conditional VaR or

tail VaR). Left tail mean is the integrale from -Inf to p of the quantile function qkiener3 divided by p. Right tail mean is the integrale from p to +Inf of the quantile function qkiener3 divided by 1-p. Expected shortfall turns negative numbers into positive numbers with the following rule:

$$eskiener3 < -if(p <= 0.5) - ltmkiener3 elsertmkiener3$$

Usual values in finance are p = 0.01, p = 0.025, p = 0.975 and p = 0.99. lower.tail = FALSE uses 1-p rather than p.

dtmqkiener3 is the difference between the left tail mean and the quantile when (p <= 0.5) and the difference between the right tail mean and the quantile when (p > 0.5). It is in quantile unit and is an indirect measure of the tail curvature.

References

P. Kiener, Explicit models for bilateral fat-tailed distributions and applications in finance with the package FatTailsR, 8th R/Rmetrics Workshop and Summer School, Paris, 27 June 2014. Download it from: https://www.inmodelia.com/exemples/2014-0627-Rmetrics-Kiener-en.pdf

P. Kiener, Fat tail analysis and package FatTailsR, 9th R/Rmetrics Workshop and Summer School, Zurich, 27 June 2015. Download it from: https://www.inmodelia.com/exemples/2015-0627-Rmetrics-Kiener-en.pdf

C. Acerbi, D. Tasche, Expected shortfall: a natural coherent alternative to Value at Risk, 9 May 2001. Download it from: https://www.bis.org/bcbs/ca/acertasc.pdf

See Also

Symmetric Kiener distribution K1 kiener1, asymmetric Kiener distributions K2, K4 and K7 kiener2, kiener4, kiener7, conversion functions aw2k, estimation function fitkienerX, regression function regkienerLX.

Examples

```
require(graphics)

### Example 1
pp <- c(ppoints(11, a = 1), NA, NaN) ; pp
lp <- logit(pp) ; lp
qkiener3(  p = pp, m = 2, g = 1.5, k = aw2k(4, 6), d = aw2d(4, 6))
qlkiener3(lp = lp, m = 2, g = 1.5, k = aw2k(4, 6), d = aw2d(4, 6))
dpkiener3( p = pp, m = 2, g = 1.5, k = aw2k(4, 6), d = aw2d(4, 6))
dlkiener3(lp = lp, m = 2, g = 1.5, k = aw2k(4, 6), d = aw2d(4, 6))
dqkiener3( p = pp, m = 2, g = 1.5, k = aw2k(4, 6), d = aw2d(4, 6))


### Example 2
k       <- 4.8
d       <- 0.042
set.seed(2014)
mainTC  <- paste("qkiener3(p, m = 0, g = 1, k = ", k, ", d = ", d, ")")
mainsum <- paste("cumulated qkiener3(p, m = 0, g = 1, k = ", k, ", d = ", d, ")")
```

```
T       <- 500
C       <- 4
TC      <- qkiener3(p = runif(T*C), m = 0, g = 1, k = k, d = d)
matTC   <- matrix(TC, nrow = T, ncol = C, dimnames = list(1:T, letters[1:C]))
head(matTC)
plot.ts(matTC, main = mainTC)
#
matsum  <- apply(matTC, MARGIN=2, cumsum)
head(matsum)
plot.ts(matsum, plot.type = "single", main = mainsum)
### End example 2


### Example 3 (four plots: probability, density, logit, logdensity)
x     <- q  <- seq(-15, 15, length.out=101)
k     <- 3.2
d     <- c(-0.1, -0.03, -0.01, 0.01, 0.03, 0.1) ; names(d) <- d
olty  <- c(2, 1, 2, 1, 2, 1, 1)
olwd  <- c(1, 1, 2, 2, 3, 3, 2)
ocol  <- c(2, 2, 4, 4, 3, 3, 1)
lleg  <- c("logit(0.999) = 6.9", "logit(0.99)   = 4.6", "logit(0.95)   = 2.9",
           "logit(0.50)   = 0", "logit(0.05)   = -2.9", "logit(0.01)   = -4.6",
           "logit(0.001) = -6.9  ")
op    <- par(mfrow=c(2,2), mgp=c(1.5,0.8,0), mar=c(3,3,2,1))

plot(x, pkiener3(x, k = 3.2, d = 0), type = "l", lwd = 3, ylim = c(0, 1),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "pkiener3(q, m, g, k=3.2, d=...)")
for (i in 1:length(d)) lines(x, pkiener3(x, k = 3.2, d = d[i]),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", title = expression(delta), legend = c(d, "0"),
       cex = 0.7, inset = 0.02, lty = olty, lwd = olwd, col = ocol )

plot(x, dkiener3(x, k = 3.2, d = 0), type = "l", lwd = 3, ylim = c(0, 0.14),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "dkiener3(q, m, g, k=3.2, d=...)")
for (i in 1:length(d)) lines(x, dkiener3(x, k = 3.2, d = d[i]),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topright", title = expression(delta), legend = c(d, "0"),
       cex = 0.7, inset = 0.02, lty = olty, lwd = olwd, col = ocol )

plot(x, lkiener3(x, k = 3.2, d = 0), type = "l", lwd =3, ylim = c(-7.5, 7.5),
     yaxt="n", xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "logit(pkiener3(q, m, g, k=3.2, d=...))")
axis(2, las=1, at=c(-6.9, -4.6, -2.9, 0, 2.9, 4.6, 6.9) )
for (i in 1:length(d)) lines(x, lkiener3(x, k = 3.2, d = d[i]),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", legend = lleg, cex = 0.7, inset = 0.02 )
legend("bottomright", title = expression(delta), legend = c(d, "0"),
       cex = 0.7, inset = 0.02, lty = c(olty), lwd = c(olwd), col = c(ocol) )

plot(x, dkiener3(x, k = 3.2, d = 0, log = TRUE), type = "l", lwd = 3,
     ylim = c(-8, -1.5), xaxs = "i", yaxs = "i", xlab = "", ylab = "",
```

```
     main = "log(dkiener3(q, m, g, k=2, d=...))")
for (i in 1:length(d)) lines(x, dkiener3(x, k = 3.2, d = d[i], log=TRUE),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("bottom", title = expression(delta), legend = c(d, "0"),
       cex = 0.7, inset = 0.02, lty = olty, lwd = olwd, col = ocol )
### End example 3


### Example 4 (four plots: quantile, derivate, density and quantiles from p)
p     <- ppoints(199, a=0)
d     <- c(-0.1, -0.03, -0.01, 0.01, 0.03, 0.1) ; names(d) <- d
op    <- par(mfrow=c(2,2), mgp=c(1.5,0.8,0), mar=c(3,3,2,1))

plot(p, qlogis(p, scale = 2), type = "l", lwd = 2, xlim = c(0, 1),
      ylim = c(-15, 15), xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "qkiener3(p, m, g, k=3.2, d=...)")
for (i in 1:length(d)) lines(p, qkiener3(p, k = 3.2, d = d[i]),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", title = expression(delta), legend = c(d, "qlogis(x/2)"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

plot(p, 2/p/(1-p), type = "l", lwd = 2, xlim = c(0, 1), ylim = c(0, 100),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "dqkiener3(p, m, g, k=3.2, d=...)")
for (i in 1:length(d)) lines(p, dqkiener3(p, k = 3.2, d = d[i]),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("top", title = expression(delta), legend = c(d, "p*(1-p)/2"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

plot(qlogis(p, scale = 2), p*(1-p)/2, type = "l", lwd = 2, xlim = c(-15, 15),
     ylim = c(0, 0.14), xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "qkiener3, dpkiener3(p, m, g, k=3.2, d=...)")
for (i in 1:length(d)) {
     lines(qkiener3(p, k = 3.2, d = d[i]), dpkiener3(p, k = 3.2, d = d[i]),
           lty = olty[i], lwd = olwd[i], col = ocol[i] ) }
legend("topleft", title = expression(delta), legend = c(d, "p*(1-p)/2"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

plot(qlogis(p, scale = 2), p, type = "l", lwd = 2, xlim = c(-15, 15),
     ylim = c(0, 1), xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "inverse axis qkiener3(p, m, g, k=3.2, d=...)")
for (i in 1:length(d)) lines(qkiener3(p, k = 3.2, d = d[i]), p,
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", title = expression(delta), legend = c(d, "qlogis(x/2)"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )
### End example 4


### Example 5 (q and VaR, ltm, rtm, and ES)
pp <- c(0.001, 0.0025, 0.005, 0.01, 0.025, 0.05,
        0.10, 0.20, 0.35, 0.5, 0.65, 0.80, 0.90,
        0.95, 0.975, 0.99, 0.995, 0.9975, 0.999)
m <- -10 ; g <- 1 ; k <- 4 ; d <- 0.06
```

```
a <- dk2a(d, k) ; w <- dk2w(d, k) ; e <- dk2e(d, k)
round(c(m = m, g = g, a = a, k = k, w = w, d = d, e = e), 2)
plot(qkiener3(  pp, m=m, k=k, d=d), pp, type ="b")
round(cbind(p = pp, "1-p" = 1-pp,
q   =   qkiener3(pp, m, g, k, d),
ltm = ltmkiener3(pp, m, g, k, d),
rtm = rtmkiener3(pp, m, g, k, d),
ES  =  eskiener3(pp, m, g, k, d),
VaR = varkiener3(pp, m, g, k, d)), 4)
round(kmean(c(m, g, k, d), model = "K3"), 4) # limit value for ltm and rtm
round(cbind(p = pp, "1-p" = 1-pp,
q   =   qkiener3(pp, m, g, k, d, lower.tail = FALSE),
ltm = ltmkiener3(pp, m, g, k, d, lower.tail = FALSE),
rtm = rtmkiener3(pp, m, g, k, d, lower.tail = FALSE),
ES  =  eskiener3(pp, m, g, k, d, lower.tail = FALSE),
VaR = varkiener3(pp, m, g, k, d, lower.tail = FALSE)), 4)
### End example 5
```

---

kiener4 *Asymmetric Kiener Distribution K4*

Description

Density, distribution function, quantile function, random generation, value-at-risk, expected shortfall (+ signed left/right tail mean) and additional formulae for asymmetric Kiener distribution K4.

Usage

```
dkiener4(x, m = 0, g = 1, k = 3.2, e = 0, log = FALSE)

pkiener4(q, m = 0, g = 1, k = 3.2, e = 0, lower.tail = TRUE, log.p = FALSE)

qkiener4(p, m = 0, g = 1, k = 3.2, e = 0, lower.tail = TRUE, log.p = FALSE)

rkiener4(n, m = 0, g = 1, k = 3.2, e = 0)

dpkiener4(p, m = 0, g = 1, k = 3.2, e = 0, log = FALSE)

dqkiener4(p, m = 0, g = 1, k = 3.2, e = 0, log = FALSE)

lkiener4(x, m = 0, g = 1, k = 3.2, e = 0)

dlkiener4(lp, m = 0, g = 1, k = 3.2, e = 0, log = FALSE)

qlkiener4(lp, m = 0, g = 1, k = 3.2, e = 0, lower.tail = TRUE)
```

```
varkiener4(p, m = 0, g = 1, k = 3.2, e = 0, lower.tail = TRUE, log.p = FALSE)

ltmkiener4(p, m = 0, g = 1, k = 3.2, e = 0, lower.tail = TRUE, log.p = FALSE)

rtmkiener4(p, m = 0, g = 1, k = 3.2, e = 0, lower.tail = TRUE, log.p = FALSE)

dtmqkiener4(p, m = 0, g = 1, k = 3.2, e = 0, lower.tail = TRUE, log.p = FALSE)

eskiener4(p, m = 0, g = 1, k = 3.2, e = 0, lower.tail = TRUE,
  log.p = FALSE, signedES = FALSE)
```

Arguments

x	vector of quantiles.
m	numeric. The median.
g	numeric. The scale parameter, preferably strictly positive.
k	numeric. The tail parameter, preferably strictly positive.
e	numeric. The eccentricity parameter between left and right tails.
log	logical. If TRUE, densities are given in log scale.
q	vector of quantiles.
lower.tail	logical. If TRUE, use p. If FALSE, use 1-p.
log.p	logical. If TRUE, probabilities p are given as log(p).
p	vector of probabilities.
n	number of observations. If length(n) > 1, the length is taken to be the number required.
lp	vector of logit of probabilities.
signedES	logical. FALSE (default) returns positive numbers for left and right tails. TRUE returns negative number (= ltmkiener4) for left tail and positive number (= rtmkiener4) for right tail.

Details

Kiener distributions use the following parameters, some of them being redundant. See aw2k and pk2pk for the formulas and the conversion between parameters:

- m (mu) is the median of the distribution,.
- g (gamma) is the scale parameter.
- a (alpha) is the left tail parameter.
- k (kappa) is the harmonic mean of a and w and describes a global tail parameter.
- w (omega) is the right tail parameter.
- d (delta) is the distortion parameter.
- e (epsilon) is the eccentricity parameter.

Kiener distributions `K4(m,g,k,e,...)` are distributions with asymmetrical left and right fat tails described by a global tail parameter `k` and an eccentricity parameter `e`.

Distributions K3 (`kiener3`) with parameters `k` (kappa) and `d` (delta) and distributions K4 (`kiener4`) with parameters `k` (kappa) and `e` (epsilon)) have been created to disantangle the parameters `a` (alpha) and `w` (omega) of distributions K2 (`kiener2`). The tiny difference between distributions K3 and K4 ($d = e/k$) has not yet been fully evaluated. Both should be tested at that moment.

`k` is the harmonic mean of `a` and `w` and represents a global tail parameter.

`e` is an eccentricity parameter between the left tail parameter `a` and the right tail parameter `w`. It verifies the inequality: $-1 < e < 1$ (whereas `d` of distribution K3 verifies $-k < d < k$). The conversion functions (see `aw2k`) are:

$$1/k = (1/a + 1/w)/2$$

$$e = (a - w)/(a + w)$$

$$a = k/(1 - e)$$

$$w = k/(1 + e)$$

`e` (and `d`) should be of the same sign than the skewness. A negative value $e < 0$ implies $a < w$ and indicates a left tail heavier than the right tail. A positive value $e > 0$ implies $a > w$ and a right tail heavier than the left tail.

`m` is the median of the distribution. `g` is the scale parameter and the inverse of the density at the median: $g = 1/8/f(m)$. As a first estimate, it is approximatively one fourth of the standard deviation $g \approx \sigma/4$ but is independant from it.

The `d`, `p` functions have no explicit forms. They are provided here for convenience. They are estimated from a reverse optimization on the quantile function and can be (very) slow, depending the number of points to estimate. We recommand to use the quantile function as far as possible. WARNING: Results may become inconsistent when `k` is smaller than 1 or for very large absolute values of `e`. Hopefully, these cases seldom happen in finance.

`qkiener4` function is defined for `p` in (0, 1) by:

$$qkiener4(p, m, g, k, e) = m + 2 * g * k * sinh(logit(p)/k) * exp(e/k * logit(p))$$

`rkiener4` generates `n` random quantiles.

In addition to the classical `d`, `p`, `q`, `r` functions, the prefixes `dp`, `dq`, `l`, `dl`, `ql` are also provided.

`dpkiener4` is the density function calculated from the probability `p`. The formula is adapted from distribution K2. It is defined for `p` in (0, 1) by:

$$dpkiener4(p, m, g, k, e) = p * (1 - p)/k/g/(exp(-logit(p)/a)/a + exp(logit(p)/w)/w$$

with `a` and `w` defined from `k` and `e`.

`dqkiener4` is the derivate of the quantile function calculated from the probability `p`. The formula is adapted from distribution K2. It is defined for `p` in (0, 1) by:

$$dqkiener4(p, m, g, k, e) = k * g/p/(1 - p) * (exp(-logit(p)/a)/a + exp(logit(p)/w)/w)$$

with `a` and `w` defined with the formula presented above.

lkiener4 function is estimated from a reverse optimization and can be (very) slow depending the number of points to estimate. Initialization is done with a symmetric distribution lkiener1 of parameter k (thus $e = 0$). Then optimization is performed to take into account the true value of e. The results can then be compared to the empirical probability logit(p). WARNING: Results may become inconsistent when k is smaller than 1 or for very large absolute values of e. Hopefully, these cases seldom happen in finance.

dlkiener4 is the density function calculated from the logit of the probability lp = logit(p). The formula is adapted from distribution K2. it is defined for lp in (-Inf, +Inf) by:

$$dlkiener4(lp, m, g, k, e) = p * (1 - p)/k/g/(exp(-lp/a)/a + exp(lp/w)/w)$$

with a and w defined above.

qlkiener4 is the quantile function calculated from the logit of the probability. It is defined for lp in (-Inf, +Inf) by:

$$qlkiener4(lp, m, g, k, e) = m + 2 * g * k * sinh(lp/k) * exp(e/k * lp)$$

varkiener4 designates the Value a-risk and turns negative numbers into positive numbers with the following rule:

$$varkiener4 < -if(p <= 0.5) - qkiener4elseqkiener4$$

Usual values in finance are p = 0.01, p = 0.05, p = 0.95 and p = 0.99. lower.tail = FALSE uses 1-p rather than p.

ltmkiener4, rtmkiener4 and eskiener4 are respectively the left tail mean, the right tail mean and the expected shortfall of the distribution (sometimes called average VaR, conditional VaR or tail VaR). Left tail mean is the integrale from -Inf to p of the quantile function qkiener4 divided by p. Right tail mean is the integrale from p to +Inf of the quantile function qkiener4 divided by 1-p. Expected shortfall turns negative numbers into positive numbers with the following rule:

$$eskiener4 < -if(p <= 0.5) - ltmkiener4elsertmkiener4$$

Usual values in finance are p = 0.01, p = 0.025, p = 0.975 and p = 0.99. lower.tail = FALSE uses 1-p rather than p.

dtmqkiener4 is the difference between the left tail mean and the quantile when (p <= 0.5) and the difference between the right tail mean and the quantile when (p > 0.5). It is in quantile unit and is an indirect measure of the tail curvature.

References

P. Kiener, Explicit models for bilateral fat-tailed distributions and applications in finance with the package FatTailsR, 8th R/Rmetrics Workshop and Summer School, Paris, 27 June 2014. Download it from: https://www.inmodelia.com/exemples/2014-0627-Rmetrics-Kiener-en.pdf

P. Kiener, Fat tail analysis and package FatTailsR, 9th R/Rmetrics Workshop and Summer School, Zurich, 27 June 2015. Download it from: https://www.inmodelia.com/exemples/2015-0627-Rmetrics-Kiener-en.pdf

C. Acerbi, D. Tasche, Expected shortfall: a natural coherent alternative to Value at Risk, 9 May 2001. Download it from: https://www.bis.org/bcbs/ca/acertasc.pdf

See Also

Symmetric Kiener distribution K1 kiener1, asymmetric Kiener distributions K2, K3 and K7 kiener2, kiener3, kiener7, conversion functions aw2k, estimation function fitkienerX,

Examples

```
require(graphics)

### Example 1
pp <- c(ppoints(11, a = 1), NA, NaN) ; pp
lp <- logit(pp) ; lp
qkiener4(  p = pp, m = 2, g = 1.5, k = aw2k(4, 6), e = aw2e(4, 6))
qlkiener4(lp = lp, m = 2, g = 1.5, k = aw2k(4, 6), e = aw2e(4, 6))
dpkiener4( p = pp, m = 2, g = 1.5, k = aw2k(4, 6), e = aw2e(4, 6))
dlkiener4(lp = lp, m = 2, g = 1.5, k = aw2k(4, 6), e = aw2e(4, 6))
dqkiener4( p = pp, m = 2, g = 1.5, k = aw2k(4, 6), e = aw2e(4, 6))


### Example 2
k       <- 4.8
e       <- 0.2
set.seed(2014)
mainTC  <- paste("qkiener4(p, m = 0, g = 1, k = ", k, ", e = ", e, ")")
mainsum <- paste("cumulated qkiener4(p, m = 0, g = 1, k = ", k, ", e = ", e, ")")
T       <- 500
C       <- 4
TC      <- qkiener4(p = runif(T*C), m = 0, g = 1, k = k, e = e)
matTC   <- matrix(TC, nrow = T, ncol = C, dimnames = list(1:T, letters[1:C]))
head(matTC)
plot.ts(matTC, main = mainTC)
#
matsum  <- apply(matTC, MARGIN=2, cumsum)
head(matsum)
plot.ts(matsum, plot.type = "single", main = mainsum)
### End example 2


### Example 3 (four plots: probability, density, logit, logdensity)
x     <- q  <- seq(-15, 15, length.out=101)
k     <- 3.2
e     <- c(-0.3, -0.15, -0.07, 0.07, 0.15, 0.30) ; names(e) <- e
olty  <- c(2, 1, 2, 1, 2, 1, 1)
olwd  <- c(1, 1, 2, 2, 3, 3, 2)
ocol  <- c(2, 2, 4, 4, 3, 3, 1)
lleg  <- c("logit(0.999) = 6.9", "logit(0.99)   = 4.6", "logit(0.95)   = 2.9",
           "logit(0.50)   = 0", "logit(0.05)   = -2.9", "logit(0.01)   = -4.6",
           "logit(0.001) = -6.9  ")
op    <- par(mfrow=c(2,2), mgp=c(1.5,0.8,0), mar=c(3,3,2,1))

plot(x, pkiener4(x, k = 3.2, e = 0), type = "l", lwd = 3, ylim = c(0, 1),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "",
```

```
     main = "pkiener4(q, m, g, k=3.2, e=...)")
for (i in 1:length(e)) lines(x, pkiener4(x, k = 3.2, e = e[i]),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", title = expression(epsilon), legend = c(e, "0"),
       cex = 0.7, inset = 0.02, lty = olty, lwd = olwd, col = ocol )

plot(x, dkiener4(x, k = 3.2, e = 0), type = "l", lwd = 3, ylim = c(0, 0.14),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "dkiener4(q, m, g, k=3.2, e=...)")
for (i in 1:length(e)) lines(x, dkiener4(x, k = 3.2, e = e[i]),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topright", title = expression(epsilon), legend = c(e, "0"),
       cex = 0.7, inset = 0.02, lty = olty, lwd = olwd, col = ocol )

plot(x, lkiener4(x, k = 3.2, e = 0), type = "l", lwd =3, ylim = c(-7.5, 7.5),
     yaxt="n", xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "logit(pkiener4(q, m, g, k=3.2, e=...))")
axis(2, las=1, at=c(-6.9, -4.6, -2.9, 0, 2.9, 4.6, 6.9) )
for (i in 1:length(e)) lines(x, lkiener4(x, k = 3.2, e = e[i]),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", legend = lleg, cex = 0.7, inset = 0.02 )
legend("bottomright", title = expression(epsilon), legend = c(e, "0"),
       cex = 0.7, inset = 0.02, lty = c(olty), lwd = c(olwd), col = c(ocol) )

plot(x, dkiener4(x, k = 3.2, e = 0, log = TRUE), type = "l", lwd = 3,
     ylim = c(-8, -1.5), xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "log(dkiener4(q, m, g, k=2, e=...))")
for (i in 1:length(e)) lines(x, dkiener4(x, k = 3.2, e = e[i], log=TRUE),
       lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("bottom", title = expression(epsilon), legend = c(e, "0"),
       cex = 0.7, inset = 0.02, lty = olty, lwd = olwd, col = ocol )
### End example 3


### Example 4 (four plots: quantile, derivate, density and quantiles from p)
p     <- ppoints(199, a=0)
e     <- c(-0.3, -0.15, -0.07, 0.07, 0.15, 0.30) ; names(e) <- e
op    <- par(mfrow=c(2,2), mgp=c(1.5,0.8,0), mar=c(3,3,2,1))

plot(p, qlogis(p, scale = 2), type = "l", lwd = 2, xlim = c(0, 1),
     ylim = c(-15, 15), xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "qkiener4(p, m, g, k=3.2, e=...)")
for (i in 1:length(e)) lines(p, qkiener4(p, k = 3.2, e = e[i]),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", title = expression(epsilon), legend = c(e, "qlogis(x/2)"),
        inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

plot(p, 2/p/(1-p), type = "l", lwd = 2, xlim = c(0, 1), ylim = c(0, 100),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "dqkiener4(p, m, g, k=3.2, e=...)")
for (i in 1:length(e)) lines(p, dqkiener4(p, k = 3.2, e = e[i]),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("top", title = expression(epsilon), legend = c(e, "p*(1-p)/2"),
```

```
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

plot(qlogis(p, scale = 2), p*(1-p)/2, type = "l", lwd = 2, xlim = c(-15, 15),
     ylim = c(0, 0.14), xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "qkiener4, dpkiener4(p, m, g, k=3.2, e=...)")
for (i in 1:length(e)) {
     lines(qkiener4(p, k = 3.2, e = e[i]), dpkiener4(p, k = 3.2, e = e[i]),
           lty = olty[i], lwd = olwd[i], col = ocol[i] ) }
legend("topleft", title = expression(epsilon), legend = c(e, "p*(1-p)/2"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

plot(qlogis(p, scale = 2), p, type = "l", lwd = 2, xlim = c(-15, 15),
     ylim = c(0, 1), xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "inverse axis qkiener4(p, m, g, k=3.2, e=...)")
for (i in 1:length(e)) lines(qkiener4(p, k = 3.2, e = e[i]), p,
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", title = expression(epsilon), legend = c(e, "qlogis(x/2)"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )
### End example 4


pp <- c(0.001, 0.0025, 0.005, 0.01, 0.025, 0.05,
        0.10, 0.20, 0.35, 0.5, 0.65, 0.80, 0.90,
        0.95, 0.975, 0.99, 0.995, 0.9975, 0.999)
m <- -5 ; g <- 1 ; k <- 4 ; e = -0.20
a <- ek2a(e, k) ; w <- ek2w(e, k) ; d <- ek2d(e, k)
round(c(m = m, g = g, a = a, k = k, w = w, d = d, e = e), 2)
plot(qkiener4(pp, m, g, k, e), pp, type = "b")
round(cbind(p = pp, "1-p" = 1-pp,
q   =   qkiener4(pp, m, g, k, e),
ltm = ltmkiener4(pp, m, g, k, e),
rtm = rtmkiener4(pp, m, g, k, e),
ES  =  eskiener4(pp, m, g, k, e),
VaR = varkiener4(pp, m, g, k, e)), 4)
round(kmean(c(m, g, k, e), model = "K4"), 4) # limit value for ltm and rtm
round(cbind(p = pp, "1-p" = 1-pp,
q   =   qkiener4(pp, m, g, k, e, lower.tail = FALSE),
ltm = ltmkiener4(pp, m, g, k, e, lower.tail = FALSE),
rtm = rtmkiener4(pp, m, g, k, e, lower.tail = FALSE),
ES  =  eskiener4(pp, m, g, k, e, lower.tail = FALSE),
VaR = varkiener4(pp, m, g, k, e, lower.tail = FALSE)), 4)
### End example 5
```

---

kiener7 *Asymmetric Kiener Distribution K7 (K2)*

Description

Density, distribution function, quantile function, random generation, value-at-risk, expected shortfall (+ signed left/right tail mean) and additional formulae for asymmetric Kiener distribution K7 = K2. With K7, the vector of parameters is provided as `coefk`, usually estimated with `paramkienerX` (and ~X5,~X7) or `regkienerLX$coefk`. Main inputs can be supplied as vector (`x,q,p`) and matrix (`coefk`) and the resulting output is a matrix (useful for simulation).

Usage

```
dkiener7(x, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0), log = FALSE)

pkiener7(q, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0), lower.tail = TRUE,
  log.p = FALSE)

qkiener7(p, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0), lower.tail = TRUE,
  log.p = FALSE)

rkiener7(n, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0), same_p = FALSE)

dpkiener7(p, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0), log = FALSE)

dqkiener7(p, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0), log = FALSE)

lkiener7(x, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0))

dlkiener7(lp, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0), log = FALSE)

qlkiener7(lp, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0), lower.tail = TRUE)

varkiener7(p, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0), lower.tail = TRUE,
  log.p = FALSE)

ltmkiener7(p, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0), lower.tail = TRUE,
  log.p = FALSE)

rtmkiener7(p, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0), lower.tail = TRUE,
  log.p = FALSE)

dtmqkiener7(p, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0), lower.tail = TRUE,
  log.p = FALSE)

eskiener7(p, coefk = c(0, 1, 3.2, 3.2, 3.2, 0, 0), lower.tail = TRUE,
  log.p = FALSE, signedES = FALSE)
```

Arguments

`x`	vector of quantiles.
`coefk`	vector of 7 parameters `c(m,g,a,k,w,d,e)` or matrix with 7 columns.

log	logical. If TRUE, densities are given in log scale.
q	vector of quantiles.
lower.tail	logical. If TRUE, use p. If FALSE, use 1-p.
log.p	logical. If TRUE, probabilities p are given as log(p).
p	vector of probabilities.
n	integer. Number of observations. If length(n) > 1, the length is taken to be the number required.
same_p	logical. If FALSE (default), random probabilies are generated on the fly. If TRUE, the same set of random probabilities is used for each line of coefk (if coefk is a matrix).
lp	vector of logit of probabilities.
signedES	logical. FALSE (default) returns positive numbers for left and right tails. TRUE returns negative number (= ltmkiener7) for left tail and positive number (= rtmkiener7) for right tail.

Details

Kiener distributions use the following parameters, some of them being redundant. See aw2k and pk2pk for the formulas and the conversion between parameters:

- m (mu) is the median of the distribution.
- g (gamma) is the scale parameter.
- a (alpha) is the left tail parameter.
- k (kappa) is the harmonic mean of a and w and describes a global tail parameter.
- w (omega) is the right tail parameter.
- d (delta) is the distortion parameter.
- e (epsilon) is the eccentricity parameter.

Kiener distribution K7 is designed after kiener2 but uses as input coefk rather than m, g, a and w.

The d, p functions have no explicit forms. They are provided here for convenience. They are estimated from a reverse optimization on the quantile function and can be (very) slow, depending the number of points to estimate. We recommand to use the quantile function as much as possible. WARNING: Results may become inconsistent when a or w are smaller than 1.

qkiener7 function is defined for p in (0, 1) by:

$$qkiener7(p, coefk) = m + g * k * (-exp(-logit(p)/a) + exp(logit(p)/w))$$

where k is the harmonic mean of the tail parameters a and w calculated by $k = aw2k(a, w)$.

rkiener7 generates n random quantiles.

In addition to the classical d, p, q, r functions, the prefixes dp, dq, l, dl, ql are also provided.

dpkiener7 is the density function calculated from the probability p. It is defined for p in (0, 1) by:

$$dpkiener7(p, coefk) = p * (1 - p)/k/g/(exp(-logit(p)/a)/a + exp(logit(p)/w)/w$$

dqkiener7 is the derivate of the quantile function calculated from the probability p. It is defined for p in (0, 1) by:

$$dqkiener7(p, coefk) = k * g/p/(1-p) * (exp(-logit(p)/a)/a + exp(logit(p)/w)/w)$$

lkiener7 function is estimated from a reverse optimization and can be (very) slow depending the number of points to estimate. Initialization is done by assuming a symmetric distribution lkiener1 around the harmonic mean k, then optimization is performed to take into account the true values a and w. The result can be then compared to the empirical probability logit(p). WARNING: Results may become inconsistent when a or w are smaller than 1.

dlkiener7 is the density function calculated from the logit of the probability lp = logit(p). it is defined for lp in (-Inf, +Inf) by:

$$dlkiener7(lp, coefk) = p * (1-p)/k/g/(exp(-lp/a)/a + exp(lp/w)/w)$$

qlkiener7 is the quantile function calculated from the logit of the probability. It is defined for lp in (-Inf, +Inf) by:

$$qlkiener7(lp, coefk) = m + g * k * (-exp(-lp/a) + exp(lp/w))$$

varkiener7 designates the Value a-risk and turns negative numbers into positive numbers with the following rule:

$$varkiener7 < -if(p <= 0.5) - qkiener7 elseqkiener7$$

Usual values in finance are p = 0.01, p = 0.05, p = 0.95 and p = 0.99. lower.tail = FALSE uses 1-p rather than p.

ltmkiener7, rtmkiener7 and eskiener7 are respectively the left tail mean, the right tail mean and the expected shortfall of the distribution (sometimes called average VaR, conditional VaR or tail VaR). Left tail mean is the integrale from -Inf to p of the quantile function qkiener7 divided by p. Right tail mean is the integrale from p to +Inf of the quantile function qkiener7 divided by 1-p. Expected shortfall turns negative numbers into positive numbers with the following rule:

$$eskiener7 < -if(p <= 0.5) - ltmkiener7 elsertmkiener7$$

Usual values in finance are p = 0.01, p = 0.025, p = 0.975 and p = 0.99. lower.tail = FALSE uses 1-p rather than p.

dtmqkiener7 is the difference between the left tail mean and the quantile when (p <= 0.5) and the difference between the right tail mean and the quantile when (p > 0.5). It is in quantile unit and is an indirect measure of the tail curvature.

References

P. Kiener, Explicit models for bilateral fat-tailed distributions and applications in finance with the package FatTailsR, 8th R/Rmetrics Workshop and Summer School, Paris, 27 June 2014. Download it from: https://www.inmodelia.com/exemples/2014-0627-Rmetrics-Kiener-en.pdf

P. Kiener, Fat tail analysis and package FatTailsR, 9th R/Rmetrics Workshop and Summer School, Zurich, 27 June 2015. Download it from: https://www.inmodelia.com/exemples/2015-0627-Rmetrics-Kiener-en.pdf

C. Acerbi, D. Tasche, Expected shortfall: a natural coherent alternative to Value at Risk, 9 May 2001. Download it from: https://www.bis.org/bcbs/ca/acertasc.pdf

See Also

Symmetric Kiener distribution K1 kiener1, asymmetric Kiener distributions K2, K3 and K4 kiener2, kiener3, kiener4, conversion functions aw2k, estimation function paramkienerX, estimation function fitkienerX, regression function regkienerLX.

Examples

```
head(ED <- fatreturns(extractData()))
(coefk  <- paramkienerX(ED, dgts = 3))
x  <- -4
xx <- -4:4
p  <- 0.1
pp <- pprobs2

dkiener7(x)
dkiener7(x,  coefk)
dkiener7(xx)
dkiener7(xx, coefk)

pkiener7(x)
pkiener7(x,  coefk)
pkiener7(xx)
pkiener7(xx, coefk)

qkiener7(p)
qkiener7(p,  coefk)
qkiener7(pp)
qkiener7(pp, coefk)

rkiener7(10)
rkiener7(10, coefk)

varkiener7(p)
varkiener7(p, coefk)
varkiener7(pp)
varkiener7(pp, coefk)

ltmkiener7(p)
ltmkiener7(p, coefk)
ltmkiener7(pp)
ltmkiener7(pp, coefk)

eskiener7(p)
eskiener7(p, coefk)
eskiener7(pp)
eskiener7(pp, coefk)
```

kmoments *Moments Associated To Kiener Distribution Parameters*

Description

Non-central moments, central moments, mean, standard deviation, variance, skewness, kurtosis, excess of kurtosis and cumulants associated to the parameters of Kiener distributions K1, K2, K3 and K4. All-in-one vectors `kmoments` (estimated from the parameters) and `xmoments` (estimated from the vector of quantiles) are provided.

Usage

```
kmoments(coefk, model = "K2", lengthx = NA, dgts = NULL, dimnames = FALSE)

xmoments(x, dgts = NULL, dimnames = FALSE)

kmoment(n, coefk, model = "K2", dgts = NULL)

kcmoment(n, coefk, model = "K2", dgts = NULL)

kmean(coefk, model = "K2", dgts = NULL)

kstandev(coefk, model = "K2", dgts = NULL)

kvariance(coefk, model = "K2", dgts = NULL)

kskewness(coefk, model = "K2", dgts = NULL)

kkurtosis(coefk, model = "K2", dgts = NULL)

kekurtosis(coefk, model = "K2", dgts = NULL)
```

Arguments

`coefk`	vector. Parameters of the distribution of length 3 ("K1"), length 4 (model = K2, K3, K4) and length 7 ("K7").
`model`	character. Model type, either "K2", "K3" or "K4" if `coefk` is of length 4. Type "K1" and "K7" may be provided but are ignored.
`lengthx`	integer. The length of the vector `x` used to calculate the parameters. See the details for matrix and lists.
`dgts`	integer. The rounding applied to the output.
`dimnames`	boolean. Display dimnames.
`x`	numeric. Vector of quantiles.
`n`	integer. The moment order.

Details

The non-central moments m1,m2,m3,m4,..,mn, the central moments u1,u2,u3,u4,..,un (where u stands for mu in Greek) and the cumulants k1,k2,k3,k4,..,kn (where k stands for kappa in Greek; not to be confounded with tail parameter "k" and models "K1", "K2", "K3", "K4") of order n exist only if $min(a,k,w) > n$. The mean m1 exists only if $min(a,k,w) > 1$. The standard deviation sd and the variance u2 exist only if $min(a,k,w) > 2$. The skewness sk exists only if $min(a,k,w) > 3$. The kurtosis ku and the excess of kurtosis ke exist only if $min(a,k,w) > 4$.

coefk may take five different forms :

- c(m,g,k) of length 3 for distribution "K1".
- c(m,g,a,w) of length 4 for distribution "K2".
- c(m,g,k,d) of length 4 for distribution "K3".
- c(m,g,k,e) of length 4 for distribution "K4".
- c(m,g,a,k,w,d,e) of length 7 (sometimes referred as "K7") provided by estimation/regression functions paramkienerX, fitkienerX, regkienerLX (via "reg$coefk") and conversion function pk2pk.

Forms of length 3 and 7 are automatically recognized and do not require model = "K1" or "K7" which are ignored. Forms of length 4 require model = "K2", "K3" or "K4". Visit pk2pk for details on the parameter conversion function used within kmoments.

xmoments and kmoments provide all-in-one vectors.

xmoments is the traditional mean of squares, cubic and power 4 functions of non-central and central values of x, from which NA values have been removed. Therefore, length of x ignores NA values and may be different from the true length.

kmoments calls every specialized functions from order 1 to order 4 and uses the estimated parameters as inputs, not the initial dataset x. As it does not know *a priori* the length of x, this latest can be provided separately via lengthx = length(x), lengthx = nrow(x) and lengthx = sapply(x,length) if x is a vector, a matrix or a list. See the examples.

Value

Vectors kmoments and xmoments have the following structure (with a third letter x added to xmoments):

ku	Kurtosis.
ke	Excess of kurtosis.
sk	Skewness.
sd	Standard deviation. Square root of the variance u2
m1	Mean.
m2	Non-central moment of second order.
m3	Non-central moment of third order.
m4	Non-central moment of fourth order.
u1	Central moment of first order. Should be 0.
u2	Central moment of second order. Variance

u3	Central moment of third order.
u4	Central moment of fourth order.
k1	Cumulant of first order. Should be 0.
k2	Cumulant of second order.
k3	Cumulant of third order.
k4	Cumulant of fourth order.
lh	Length of x, from which NA values were removed.
.....	.

See Also

pk2pk, paramkienerX, regkienerLX.

Examples

```
## Example 1
kcmoment(2, c(-1, 1, 6, 9), model = "K2")
kcmoment(2, c(-1, 1, 7.2, -0.2/7.2), model = "K3")
kcmoment(2, c(-1, 1, 7.2, -0.2), model = "K4")
kcmoment(2, c(-1, 1, 6, 7.2, 9, -0.2/7.2, -0.2))
kvariance(c(-1, 1, 6, 9))
kmoments(c(-1, 1, 6, 9), dgts = 3)

## Example 2: "K2" and "K7" are preferred input formats for kmoments
## Moments fall at expected parameter values (=> NA).
## apply and direct calculation (= transpose)
(mat4 <- matrix(c(rep(0,4), rep(1,4), c(1.9,2.1,3.9,4.1), rep(5,4)),
                nrow = 4, byrow = TRUE,
                dimnames = list(c("m","g","a","w"), paste0("b",1:4))))
round(mat7 <- apply(mat4, 2, pk2pk), 2)
round(rbind(mat7, apply(mat7, 2, kmoments)[2:5,]), 2)
round(cbind(t(mat7), kmoments(t(mat7), dgts = 2)[,2:5]), 2)

## Example 3: Matrix, timeSeries, xts, zoo + apply
matret    <- 100*diff(log((EuStockMarkets)))
(matcoefk <- apply(matret, 2, paramkienerX5, dgts = 2))
(matmomk  <- apply(matcoefk, 2, kmoments, lengthx = nrow(matret), dgts = 2))
(matmomx  <- apply(matret, 2, xmoments, dgts = 2))
rbind(matcoefk, matmomk[2:5,], matmomx[2:5,])

## Example 4: List + direct calculation = transpose
DS   <- getDSdata() ; dimdim(DS) ; class(DS)
(pDS <- paramkienerX5(DS, dimnames = FALSE))
(kDS <- kmoments(pDS, lengthx = sapply(DS, length), dgts = 3))
(xDS <- xmoments( DS, dgts = 3))
cbind(pDS, kDS[,2:5], xDS[,2:5])
```

`laplacegaussnorm` *Laplace-Gauss Normal Distribution Object*

Description

An object designed after regkienerLX to summarize the information related to a given dataset when the Laplace-Gauss normal distribution is applied on it.

Usage

```
laplacegaussnorm(X)
```

Arguments

`X`	vector of quantiles.

Details

This function is designed after regkienerLX to provide a similar framework.

Value

A list with the following data.frame:

- dfrXPndata.frame. X = initial quantiles. Pn = estimated normal probabilites.
- dfrXLndata.frame. X = initial quantiles. Ln = logit of estimated normal probabilites.
- dfrXDndata.frame. X = initial quantiles. Dn = estimated normal density.
- coefnnumeric. The mean and the standard deviation of the dataset.
- dfrQnPndata.frame. Qn = estimated quantiles of interest. Pn = probability.
- dfrQnPndata.frame. Qn = estimated quantiles of interest. Pn = logit of probability.

See Also

The regression function `regkienerLX`.

Examples

```
prices2returns <- function(x) { 100*diff(log(x)) }
CAC  <- prices2returns(as.numeric(EuStockMarkets[,3]))
lgn  <- laplacegaussnorm( CAC )
attributes(lgn)
head(lgn$dfrXPn)
head(lgn$dfrXLn)
head(lgn$dfrXDn)
lgn$coefn
lgn$dfrQnPn
lgn$dfrQnLn
```

loghp *Inverse Power Hyperbolas and Inverse Power Hyperbolic Functions*

Description

The inverse power hyperbolas and the inverse power hyperbolic functions: arc-cosine-hp, arc-sine-hp, arc-tangent-hp, arc-secant-hp, arc-cosecant-hp and arc-cotangent-hp.

Usage

```
loghp(x, k = 1)

acoshp(x, k = 1)

asinhp(x, k = 1)

atanhp(x, k = 1)

asechp(x, k = 1)

acosechp(x, k = 1)

acotanhp(x, k = 1)
```

Arguments

x	a numeric value, vector or matrix.
k	a numeric value, preferably strictly positive.

Details

loghp function is defined on (0, +Inf) by:

$$logshp(x, k) = 2 * k * sinh(log(x)/k)$$

acoshp function is defined on [1, +Inf) by:

$$acoshp(x, k) = 2 * k * sinh(acosh(x)/k)$$

asinhp function is defined on (-Inf, +Inf) by:

$$asinhp(x, k) = 2 * k * sinh(asinh(x)/k)$$

atanhp function is defined on (-1, +1) by:

$$atanhp(x, k) = 2 * k * sinh(atanh(x)/k)$$

asechp function is defined on (0, +1] by:

$$asechp(x, k) = 2 * k * sinh(acosh(1/x)/k)$$

acosechp function is defined on (-Inf, 0) U (0, +Inf) by:

$$acosechp(x, k) = 2 * k * sinh(asinh(1/x)/k)$$

acotanhp function is defined on (-Inf, -1) U (1, +Inf) by:

$$acotanhp(x, k) = 2 * k * sinh(atanh(1/x)/k)$$

If k is a vector of length > 1, then the use of the function `outer` is recommanded.

See Also

The power hyperbolic functions `exphp`.

Examples

```
### Example 1 (acoshp, asinhp, atanhp)
 loghp( c(ppoints(10), 1, 1/rev(ppoints(10))), k = 2)
acoshp( 1:10, k = 2)
asinhp( -5:5, k = 2)
atanhp( seq(-1, 1, by = 0.1), k = 2)
asechp( ppoints(20), k = 2)
acosechp( -5:5, k = 2)
acotanhp( c( -1/ppoints(10), 1/rev(ppoints(10))), k = 2)

x  <- (-3:3)*3
 loghp(exphp(x, k = 4), k = 4)
acoshp(coshp(x, k = 4), k = 4)
asinhp(sinhp(x, k = 4), k = 4)
atanhp(tanhp(x, k = 4), k = 4)


### Example 2 (loghp, acoshp, asinhp, atanhp)
k     <- c(0.6, 1, 1.5, 2, 3.2, 10) ; names(k) <- k
olty  <- c(2, 1, 2, 1, 2, 1, 1)
olwd  <- c(1, 1, 2, 2, 3, 4, 2)
ocol  <- c(2, 2, 4, 4, 3, 3, 1)
op    <- par(mfrow = c(2, 2), mgp = c(1.5, 0.8, 0), mar = c(3, 3, 2, 1))

xld     <- 0.05
xl      <- seq(0.05, 20, xld) ; names(xl) <- xl
Tlcoshp <- ts(cbind(outer(xl, k, loghp), "2*log(x)" = 2*log(xl)),
              start = xl[1], deltat = xld)
plot(Tlcoshp, plot.type = "single", xlim = c(0,20), ylim = c(-5,15),
     lty = olty, lwd = olwd, col = ocol, xaxs = "i", yaxs = "i",
     xlab="", ylab = "", main = "loghp(x, k)" )
legend("bottomright", title = expression(kappa), legend = colnames(Tlcoshp),
     inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

## acoshp(x, k)
xcd     <- 0.5
xc      <- seq(1, 20, xcd) ; names(xc) <- xc
```

```
Tacoshp <- ts(cbind(outer(xc, k, acoshp), "2*acosh(x)" = 2*acosh(xc)),
              start = xc[1], deltat = xcd)
plot(Tacoshp, plot.type = "single", ylim = c(0,15), lty = olty, lwd = olwd, col = ocol,
        xaxs = "i", yaxs = "i", xlab = "", ylab = "", main = "acoshp(x, k)" )
legend("bottomright", title = expression(kappa), legend = colnames(Tacoshp),
        inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

## asinhp(x, k)
xsd     <- 0.5
xs      <- seq(-10, 10, xsd) ; names(xs) <- xs
Tasinhp <- ts(cbind(outer(xs, k, asinhp), "2*asinh(x)" = 2*asinh(xs)),
              start = xs[1], deltat = xsd)
plot(Tasinhp, plot.type = "single", ylim = c(-10,10), lty = olty, lwd = olwd, col = ocol,
        xaxs = "i", yaxs = "i", xlab = "", ylab = "", main = "asinhp(x, k)" )
legend("topleft", title = expression(kappa), legend = colnames(Tasinhp),
        inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

## atanhp(x, k)
xtd     <- 0.01
xt      <- seq(-1, 1, xtd) ; names(xt) <- xt
Tatanhp <- ts(cbind(outer(xt, k, atanhp), "2*atanh(x)" = 2*atanh(xt)),
              start = xt[1], deltat = xtd)
plot(Tatanhp, plot.type = "single", ylim = c(-10,10), lty = olty, lwd = olwd, col = ocol,
        xaxs = "i", yaxs = "i", xlab = "", ylab = "", main = "atanhp(x, k)" )
legend("topleft", title = expression(kappa), legend = colnames(Tatanhp),
        inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )
### End Example 2
```

---

logishp *The Power Hyperbola Logistic Distribution*

Description

Density, distribution function, quantile function and random generation for the power hyperbola logistic distribution.

Usage

```
dlogishp(x, k = 1, log = FALSE)

plogishp(q, k = 1)

invkogit(q, k = 1)

qlogishp(p, k = 1)

kogit(p, k = 1)
```

```
rlogishp(n, k = 1)

dplogishp(p, k = 1, log = FALSE)

dqlogishp(p, k = 1, log = FALSE)

llogishp(x, k = 1)

dllogishp(lp, k = 1, log = FALSE)

qllogishp(lp, k = 1)
```

Arguments

x	vector of quantiles.
k	numeric. The tail parameter, preferably strictly positive. Can be a vector (see details).
log	boolean.
q	vector of quantiles.
p	vector of probabilities.
n	number of observations. If length(n) > 1, the length is taken to be the number required.
lp	vector of logit of probabilities.

Details

dlogishp function (log is available) is defined for x in (-Inf, +Inf) by:

$$dlogishp(x,k) = dkashp\_dx(x,k) * plogishp(x,k) * plogishp(-x,k)$$

invkogit=plogishp functions are defined for q in (-Inf, +Inf) by:

$$invkogit(q,k) = plogishp(q,k) = 1/(1+exp(-kashp(q,k)))$$

kogit=qlogishp functions are defined for p in (0, 1) by:

$$kogit(p,k) = qlogishp(p,k) = 2 * k * sinh(logit(p)/k)$$

rlogishp function generates n random values.

In addition to the classical formats, the prefixes dp, dq, l, dl, ql are also provided:

dplogishp function (log is available) is defined for p in (0, 1) by:

$$dplogishp(p,k=1) = p * (1-p)/2/cosh(logit(p)/k)$$

dqlogishp function (log is available) is defined for p in (0, 1) by:

$$dqlogishp(p,k=1) = 2/p/(1-p) * cosh(logit(p)/k)$$

`llogishp` function is defined for x in (-Inf, +Inf) by:

$$llogishp(x, k) = kashp(x, k)$$

`dllogishp` function is defined for lp = logit(p) in (-Inf, +Inf) by :

$$dllogishp(lp, k) = p * (1 - p)/2/cosh(lp/k)$$

`qllogishp` function is defined for lp = logit(p) in (-Inf, +Inf) by :

$$qllogishp(lp, k) = 2 * k * sinh(lp/k)$$

If k is a vector, then the use of the function `outer` is recommanded.

See Also

Kiener distribution K1 `kiener1` which has location (`m`) and scale (`g`) parameters.

Examples

```
require(graphics)

### Example 1
pp <- c(ppoints(11, a = 1), NA, NaN) ; pp
plogishp(-5:5, k = 4)
dlogishp(-5:5, k = 4)
qlogishp(pp, k = 4)
outer(-5:5, 1:6, plogishp)
outer(-5:5, 1:6, dlogishp)
outer(runif(20), 1:6, qlogishp)

### Example 2
x     <- seq(-15, 15, length.out = 101)
k     <- c(0.6, 1, 1.5, 2, 3.2, 10) ; names(k) <- k ; k
olty  <- c(2, 1, 2, 1, 2, 1, 1)
olwd  <- c(1, 1, 2, 2, 3, 4, 2)
ocol  <- c(2, 2, 4, 4, 3, 3, 1)
op    <- par(mfrow = c(2,2), mgp = c(1.5,0.8,0), mar = c(3,3,2,1))

plot(x, plogis(x, scale = 2), type = "b", lwd = 2, ylim = c(0, 1),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "", main = "plogishp(x, k)")
for (i in 1:length(k)) lines(x, plogishp(x, k = k[i]),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", title = expression(kappa), legend = c(k, "plogis(x/2)"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

plot(x, dlogis(x, scale = 2), type = "b", lwd = 2, xaxs = "i",
     yaxs = "i", xlab = "", ylab = "", main = "dlogishp(x, k)")
for (i in 1:length(k)) lines(x, dlogishp(x, k = k[i]),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )

plot(x, x/2, type = "b", lwd = 2, ylim = c(-7.5, 7.5), xaxs = "i",
```

```
     yaxs = "i", xlab = "", ylab = "", main = "logit(logishp(h, k))")
for (i in 1:length(k)) lines(x, llogishp(x, k = k[i]),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )

plot(x, log(dlogis(x, scale = 2)), lwd = 2, type = "b", xaxs = "i",
     yaxs = "i", xlab = "", ylab = "", main = "log(dlogishp(x, k))")
for (i in 1:length(k)) lines(x, dlogishp(x, k = k[i], log = TRUE),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
### End example 2

### Example 3
p <- ppoints(199, a=0)
plot(p, qlogis(p, scale = 2), type = "o", lwd = 2, ylim = c(-15, 15),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "qlogishp(p, k)")
for (i in 1:length(k)) lines(p, qlogishp(p, k = k[i]),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", title = expression(kappa), legend = c(k, "qlogis(x/2)"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

plot(p, 2/p/(1-p), type = "o", lwd = 2, xlim = c(0, 1), ylim = c(0, 100),
     xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "dqlogishp(p, k)")
for (i in 1:length(k)) lines(p, dqlogishp(p, k = k[i]),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("top", title = expression(kappa), legend = c(k, "p*(1-p)/2"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )

plot(qlogis(p, scale = 2), p*(1-p)/2, type = "o", lwd = 2, xlim = c(-15, 15),
     ylim = c(0, 0.14), xaxs = "i", yaxs = "i", xlab = "", ylab = "",
     main = "qlogishp, dplogishp(p, k)")
for (i in 1:length(k)) lines(qlogishp(p, k = k[i]), dplogishp(p, k = k[i]),
          lty = olty[i], lwd = olwd[i], col = ocol[i] )
legend("topleft", title = expression(kappa), legend = c(k, "p*(1-p)/2"),
          inset = 0.02, lty = olty, lwd = olwd, col = ocol, cex = 0.7 )
### End example 3
```

`logit` *Logit and Invlogit Functions*

Description

The logit and invlogit functions, widely used in this package, are wrappers of `qlogis` and `plogis` functions.

Functions `eslogis` is the expected shortfall of the logistic function (times a factor 2). When `p<=0.5`, it is equivalent (times -1) to the left tail mean `ltmlogis`. When `p>0.5`, it is equivalent to the right tail mean `rtmlogis`. `ltmlogis` and `rtmlogis` are used to calculate the `h` parameter in `hkiener1`, `hkiener2`, `hkiener3`, `hkiener4`.

Usage

```
logit(p)

invlogit(x)

ltmlogis(p, m = 0, g = 1, lower.tail = TRUE, log.p = FALSE)

rtmlogis(p, m = 0, g = 1, lower.tail = TRUE, log.p = FALSE)

eslogis(p, m = 0, g = 1, lower.tail = TRUE, log.p = FALSE)
```

Arguments

p	numeric. one value or a vector between 0 and 1.
x	numeric. one value or a vector of numerics.
m	numeric. a central parameter (also used in model K1, K2, K3 and K4).
g	numeric. a scale parameter (also used in model K1, K2, K3 and K4).
lower.tail	logical. If TRUE, use p. If FALSE, use 1-p.
log.p	logical. If TRUE, probabilities p are given as log(p).

Details

`logit` function is defined for p in (0, 1) by:

$$logit(p) = log(p/(1-p))$$

`invlogit` function is defined for x in (-Inf, +Inf) by:

$$invlogit(x) = exp(x)/(1+exp(x)) = plogis(x)$$

Examples

```
logit( c(ppoints(11, a = 1), NA, NaN) )
invlogit( c(-Inf, -10:10, +Inf, NA, NaN) )
```

`mData` *Datasets dfData, mData, tData, xData, zData, extractData : mData*

Description

A list of datasets in data.frame, matrix, timeSeries, xts and zoo formats. This is the matrix format. Visit `extractData` for more information.

pk2pk *Global Conversion Function Between Kiener Distribution Parameters*

Description

A conversion function between Kiener distribution parameters `K1(m,g,k)`, `K2(m,g,a,w)`, `K3(m,g,k,d)` and `K4(m,g,k,e)` to and from `coefk = c(m,g,a,k,w,d,e)` extracted from `regkienerLX` and `paramkienerX`.

Usage

```
pk2pk(coefk, model = "K2", to = "K7", dgts = NULL)
```

Arguments

`coefk`	vectors of numeric of length 3, 4 or 7.
`model`	character. Either "K1", "K2", "K3", "K4", "K7".
`to`	character. Either "K1", "K2", "K3", "K4", "K7".
`dgts`	integer. The rounding applied to the output.

Details

Kiener distributions use the following parameters, some of them being redundant. See also `aw2k` for the formulas and the conversion between parameters:

- `m` (mu) is the median of the distribution,.
- `g` (gamma) is the scale parameter.
- `a` (alpha) is the left tail parameter.
- `k` (kappa) is the harmonic mean of `a` and `w` and describes a global tail parameter.
- `w` (omega) is the right tail parameter.
- `d` (delta) is the distortion parameter.
- `e` (epsilon) is the eccentricity parameter.

`pk2pk()` performs the conversion between the various representation, from and to:

- "K1" : `kiener1(m,g,k)`
- "K2" : `kiener2(m,g,a,w)`
- "K3" : `kiener3(m,g,k,d)`
- "K4" : `kiener4(m,g,k,e)`
- "K7" : `c(m,g,a,k,w,d,e)`

`coefk` can take any of the above form. When length(coefk) is 4, `model = "K2"`, `"K3"` or `"K4"` is required to differentiate the three models. When length(coefk) is 3 or 7, recognition is automatic and `model = "K1"` or `"K7"` is ignored. The vector is assumed to be correct and there is no check of the consistency between the parameters `a`, `k`, `w`, `d` and `e`.

The output may be any of the above forms. Default is "K7" = c(m,g,a,k,w,d,e) which is coefk provided by the regression function regkienerLX or the parameter estimation function paramkienerX. It is widely in many plots.

An integer rounding parameter is provided trough dgts. Default is no rounding.

See Also

Local conversion functions aw2k, Kiener distributions K1, K2, K3 and K4: kiener1, kiener2, kiener3, kiener4

Examples

```
## Example 1
c2 <- c(1, 2, 3, 5)
pk2pk(c2, model = "K2", to = "K1") # loose the asymmetry.
pk2pk(c2, model = "K2", to = "K2")
pk2pk(c2, model = "K2", to = "K3")
pk2pk(c2, model = "K2", to = "K4")
pk2pk(c2, model = "K2", to = "K4")
(c7 <- pk2pk(c2, model = "K2", to = "K7", dgts = 3))
pk2pk(c7, model = "K7", to = "K2")

## Example 2 ("K2" to "K7")
(mat4 <- matrix( c(rep(0,9), rep(1,9), seq(0.5,4.5,0.5), seq(1,5,0.5)),
         nrow = 4, byrow = TRUE, dimnames = list(c("m","g","a","w"), paste0("b",1:9))))
(mat7 <- round(apply(mat4, 2, pk2pk), 3))
```

---

pprobs0 *Several Vectors of Probabilities*

Description

Several vectors of probabilities used in FatTailsR. Remark: pprobs5 <- sort(c(pprobs2, pprobs3, pprobs4)).

pprobs0 <- c(0.01, 0.05, 0.10, 0.25, 0.50, 0.75, 0.90, 0.95, 0.99)

pprobs1 <- c(0.01, 0.05, 0.95, 0.99)

pprobs2 <- c(0.01, 0.025, 0.05, 0.95, 0.975, 0.99)

pprobs3 <- c(0.001, 0.0025, 0.005, 0.995, 0.9975, 0.999)

pprobs4 <- c(0.0001, 0.00025, 0.0005, 0.9995, 0.99975, 0.9999)

pprobs5 <- c(0.0001, 0.00025, 0.0005, 0.001, 0.0025, 0.005, 0.01, 0.025, 0.05, 0.95, 0.975, 0.99, 0.995, 0.9975, 0.999, 0.9995, 0.99975, 0.9999)

pprobs6 <- c(0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.50, 0.95, 0.99, 0.995, 0.999, 0.9995, 0.9999)

pprobs7 <- c(0.01, 0.025, 0.05, 0.10, 0.17, 0.25, 0.33, 0.41, 0.50, 0.59, 0.67, 0.75, 0.83, 0.90, 0.95, 0.975, 0.99)

pprobs8 <- c(0.001, 0.0025, 0.005, 0.01, 0.025, 0.05, 0.10, 0.17, 0.25, 0.33, 0.41, 0.50, 0.59, 0.67, 0.75, 0.83, 0.90, 0.95, 0.975, 0.99, 0.995, 0.9975, 0.999)

pprobs9 <- c(0.0001, 0.00025, 0.0005, 0.001, 0.0025, 0.005, 0.01, 0.025, 0.05, 0.10, 0.17, 0.25, 0.33, 0.41, 0.50, 0.59, 0.67, 0.75, 0.83, 0.90, 0.95, 0.975, 0.99, 0.995, 0.9975, 0.999, 0.9995, 0.99975, 0.9999)

Usage

```
pprobs0

pprobs1

pprobs2

pprobs3

pprobs4

pprobs5

pprobs6

pprobs7

pprobs8

pprobs9
```

Format

An object of class `numeric` of length 9.

An object of class `numeric` of length 4.

An object of class `numeric` of length 6.

An object of class `numeric` of length 6.

An object of class `numeric` of length 6.

An object of class `numeric` of length 18.

An object of class `numeric` of length 13.

An object of class `numeric` of length 17.

An object of class `numeric` of length 23.

An object of class `numeric` of length 29.

See Also

The conversion function `getnamesk`

regkienerLX *Regression Function for Kiener Distributions*

Description

One function to estimate the parameters of Kiener distributions K1, K2, K3 and K4 and display the results in a list with many data.frame ready to use for plotting. This function performs an unweighted nonlinear regression of the logit of the empirical probabilities logit(p) on the quantiles X.

Usage

```
regkienerLX(X, model = "K4", pdgts = c(3, 3, 1, 1, 1, 3, 2, 4, 4, 2, 2),
  maxk = 10, mink = 0.2, app = 0, probak = pprobs2, dgts = NULL,
  exfitk = NULL)
```

Arguments

X	vector of quantiles.
model	the model used for the regression: "K1", "K2", "K3", "K4".
pdgts	vector of length 11. Control the rounding of output parameters.
maxk	numeric. The maximum value of tail parameter k.
mink	numeric. The minimum value of tail parameter k.
app	numeric. The parameter "a" in the function ppoints.
probak	vector of probabilities used in output regk$fitk. For instance pprobs0.
dgts	rounding parameter applied globally to output regk$fitk.
exfitk	character. A vector of parameter names to subset regk$fitk. For instance exfit0.

Details

This function is designed to estimate the parameters of Kiener distributions for a given dataset. It encapsulates the four distributions described in this package. "K1" uses model lqkiener1, "K2" uses model lqkiener2, "K3" uses model lqkiener3 and "K4" uses model lqkiener4.

A typical input is a numeric vector that describes the returns of a stock. Conversion from a (possible) time series format to a sorted numeric vector is done automatically and without any check of the initial format. There is also no check of missing values, Na, NaN, -Inf, +Inf. Empirical probabilities of each point in the sorted dataset is calculated with the function ppoints. The parameter app corresponds to the parameter a in ppoints but has been limited to the range (0, 0.5). Default value is 0 as large datasets are very common in finance.

A nonlinear regression is performed with nlsLM from the logit of the probabilities logit(p) over the quantiles X with one of the functions lqkiener1234. These functions have been selected as they have an explicit form in the four types (this is unfortunately not the case for dkiener234) and return satisfactory results with ordinary least squares. The median is calculated before the regression and is injected as a mandatory value in the regression function.

Kiener distributions use the following parameters, some of them being redundant. See aw2k and pk2pk for the formulas and the conversion between parameters:

- m (mu) is the median of the distribution.
- g (gamma) is the scale parameter.
- a (alpha) is the left tail parameter.
- k (kappa) is the harmonic mean of a and w and describes a global tail parameter.
- w (omega) is the right tail parameter.
- d (delta) is the distortion parameter.
- e (epsilon) is the eccentricity parameter.

Where:

- c(m, g, k) of length 3 for distribution "K1".
- c(m, g, a, w) of length 4 for distribution "K2".
- c(m, g, k, d) of length 4 for distribution "K3".
- c(m, g, k, e) of length 4 for distribution "K4".
- c(m, g, a, k, w, d, e) of length 7 extracted from object of class clregk like regkienerLX (typically "reg$coefk").

Model "K1" return results with 1+2=3 parameters and describes a (assumed) symmetric distribution. Parameters d and e are set to 0. Models "K2", "K3" and "K4" describe asymmetric distributions. They return results with 1+3=4 parameters. Model "K2" has a very clear parameter definition but unfortunately parameters a and w are highly correlated. Model "K3" has the least correlated parameters but the meaning of the distortion parameter d, usually of order 1e-3, is not simple.

Model "K4" exhibits a reasonable correlation between each parameter and should be the preferred intermediate model between "K1" and "K2" models. The eccentricity parameter e is well defined and easy to understand: $e = (a-w)/(a+w)$, $a = k/(1-e)$ and $w = k/(1+e)$. It varies between -1 and +1 and can be understood as a percentage (if times 100) of eccentricty. e = -1 corresponds to w = infinity, e = +1 corresponds to a = infinity and the model becomes a single log-logistic funtion with a right / left stopping point and a left / right tail.

Tail parameter lower and upper values are controlled by maxk and mink. An upper value $maxk = 10$ is appropriate for datasets of low and medium size, less than 50.000 points. For larger datasets, the upper limit can be extended up to $maxk = 20$. Such a limit returns results which are very closed to the logistic distribution, an alternate distribution which could be more appropriate. The lower limit mink is intended to avoid the value $k = 0$. Remind that value $k < 2$ describes distribution with no stable variance and $k < 1$ describes distribution with no stable mean.

The output is an object in a flat format of class clregk. It can be listed with the function attributes.

- First are the data.frames with the initial data and the estimated results.
- Second is the result of the regression regk0 given by nlsLM from which a few information have been extracted and listed here.
- Third are the regression parameters (without the median) in plain format (no rounding), the variance-covariance matrix, the variance-covariance matrix times 1e+6 and the correlation matrix in a rounded format. Note that regk0, coefk0, coefk0tt, vcovk0, mcork0 have a polymorphic format and changing parameters that depend from the selected model: "K1", "K2", "K3", "K4". They should be used with care in subsequent calculations.

- Fourth are the distribution parameters tailored to every model "K1", "K2", "K3", "K4" plus estimated quantiles at levels: c(0.001, 0.005, 0.01, 0.05, 0.5, 0.95, 0.99, 0.995, 0.999). They are intended to subsequent calculations.
- Fifth are the same parameters presented in a more readable format thanks to the vector `pdgts` which controls the rounding of the parameters in the following order:
- `pdgts = c("m","g","a","k","w","d","e","vcovk0","vcovk0m","mcork0","quantr")`.
- Sixth are some probabilities and the corresponding estimated quantiles and estimated Expected Shortfall stored in a data.frame format.
- Last is `fitk` which returns all parameters in the same format than `fitkienerX`, eventually subsetted by `exfitk`. IMPORTANT : if you need to subset `fitk`, always subset it by parameter names and never subset it by rank number as new items may be added in the future. Use for instance `exfitk = exfit0, ..., exfit7`.

Value

`dfrXP`	data.frame. X = initial quantiles. P = empirical probabilities.
`dfrXL`	data.frame. X = initial quantiles. L = logit of probabilities.
`dfrXR`	data.frame. X = initial quantiles. R = residuals after regression.
`dfrEP`	data.frame. E = estimated quantiles. P = probabilities.
`dfrEL`	data.frame. E = estimated quantiles. L = logit of probabilities.
`dfrED`	data.frame. E = estimated quantiles. D = estimated density (from probabilities).
`regk0`	object of class `nls` extracted from the regression function `nlsLM`.
`coefk0`	the regression parameters in plain format. The median is out of the regression.
`vcovk0`	rounded variance-covariance matrix.
`vcovk0m`	rounded 1e+6 times variance-covariance matrix.
`mcork0`	rounded correlation matrix.
`coefk`	all parameters in plain format.
`coefk1`	parameters for model "K1".
`coefk2`	parameters for model "K2".
`coefk3`	parameters for model "K3".
`coefk4`	parameters for model "K4".
`quantk`	quantiles of interest.
`coefr`	all parameters in a rounded format.
`coefr1`	rounded parameters for model "K1".
`coefr2`	rounded parameters for model "K2".
`coefr3`	rounded parameters for model "K3".
`coefr4`	rounded parameters for model "K4".
`quantr`	quantiles of interest in a rounded format.
`dfrQkPk`	data.frame. Qk = Estimated quantiles of interest. Pk = probabilities.
`dfrQkLk`	data.frame. Qk = Estimated quantiles of interest. Lk = Logit of probabilities.

dfrESkPk	data.frame. ESk = Estimated Expected Shortfall. Pk = probabilities.
dfrESkLk	data.frame. ESk = Estimated Expected Shortfall. Lk = Logit of probabilities.
fitk	Parameters, quantiles, moments, VaR, ES and other parameters (not rounded). Length of fitk depends on the choice applied to probak. IMPORTANT : if you need to subset fitk, always subset it by parameter names and never subset it by rank number as new items may be added in the future. Use for instance exfit0, ..., exfit7.

See Also

nlsLM, laplacegaussnorm, Kiener distributions K1, K2, K3 and K4: kiener1 kiener2, kiener3, kiener4. Other estimation function: fitkienerX and its derivatives. fitk subsetting: exfit0.

Examples

```
require(graphics)
require(minpack.lm)
require(timeSeries)

### Load the datasets and select one number (1-16)
DS     <- getDSdata()
j      <- 5


### and run this block
X      <- DS[[j]]
nameX  <- names(DS)[j]
reg    <- regkienerLX(X)

## Plotting
lleg   <- c("logit(0.999) = 6.9", "logit(0.99)   = 4.6",
            "logit(0.95)   = 2.9", "logit(0.50)   = 0",
            "logit(0.05)   = -2.9", "logit(0.01)   = -4.6",
            "logit(0.001) = -6.9  ")
pleg   <- c( paste("m =",  reg$coefr4[1]), paste("g  =", reg$coefr4[2]),
              paste("k  =", reg$coefr4[3]), paste("e  =", reg$coefr4[4]) )
op     <- par(mfrow=c(2,2), mgp=c(1.5,0.8,0), mar=c(3,3,2,1))
plot(X, type="l", main = nameX)
plot(reg$dfrXL, main = nameX, yaxt = "n")
axis(2, las=1, at=c(-9.2, -6.9, -4.6, -2.9, 0, 2.9, 4.6, 6.9, 9.2))
abline(h = c(-4.6, 4.6), lty = 4)
abline(v = c(reg$quantk[5], reg$quantk[9]), lty = 4)
legend("topleft", legend = lleg, cex = 0.7, inset = 0.02, bg = "#FFFFFF")
lines(reg$dfrEL, col = 2, lwd = 2)
points(reg$dfrQkLk, pch = 3, col = 2, lwd = 2, cex = 1.5)
plot(reg$dfrXP, main = nameX)
legend("topleft", legend = pleg, cex = 0.9, inset = 0.02 )
lines(reg$dfrEP, col = 2, lwd = 2)
plot(density(X), main = nameX)
```

```
lines(reg$dfrED, col = 2, lwd = 2)
round(cbind("k" = kmoments(reg$coefk, lengthx = nrow(reg$dfrXL)), "X" = xmoments(X)), 2)

## Attributes
attributes(reg)
head(reg$dfrXP)
head(reg$dfrXL)
head(reg$dfrXR)
head(reg$dfrEP)
head(reg$dfrEL)
head(reg$dfrED)
reg$regk0
reg$coefk0
reg$vcovk0
reg$vcovk0m
reg$mcork0
reg$coefk
reg$coefk1
reg$coefk2
reg$coefk3
reg$coefk4
reg$quantk
reg$coefr
reg$coefr1
reg$coefr2
reg$coefr3
reg$coefr4
reg$quantr
reg$dfrQkPk
reg$dfrQkLk
reg$dfrESkPk
reg$dfrESkLk
reg$fitk

## subset fitk
names(reg$fitk)
reg$fitk[exfit6]
reg$fitk[c(exfit1, exfit4)]
### End block
```

---

roundcoefk *Round Coefk*

Description

Round coefk parameters in a standard manner or in a special manner, the latest being useful to display nice matrix or data.frame.

Usage

```
roundcoefk(coefk, dgts = NULL, parnames = TRUE)
```

Arguments

coefk	numeric, matrix or data.frame representing parameters c(m,g,a,k,w,d,e).
dgts	integer. The number of rounded digits.
parnames	boolean. Output displayed with or without parameter names.

Details

For dgts between 1 and 9, rounding is done in the standard way and all parameters are rounded at the same number of digits.

For dgts between 10 and 27, rounding of parameters c(m,g,a,k,w,d,e) is done in the following way:

- dgts = 10 : c(0, 0, 1, 1, 1, 3, 2)
- dgts = 11 : c(1, 1, 1, 1, 1, 3, 2)
- dgts = 12 : c(2, 2, 1, 1, 1, 3, 2)
- dgts = 13 : c(3, 3, 1, 1, 1, 3, 2)
- dgts = 14 : c(4, 4, 1, 1, 1, 3, 2)
- dgts = 15 : c(5, 5, 1, 1, 1, 3, 2)
- dgts = 16 : c(0, 0, 2, 2, 2, 3, 2)
- dgts = 17 : c(1, 1, 2, 2, 2, 3, 2)
- dgts = 18 : c(2, 2, 2, 2, 2, 3, 2)
- dgts = 19 : c(3, 3, 2, 2, 2, 3, 2)
- dgts = 20 : c(4, 4, 2, 2, 2, 3, 2)
- dgts = 21 : c(5, 5, 2, 2, 2, 3, 2)
- dgts = 22 : c(0, 0, 3, 3, 3, 4, 3)
- dgts = 23 : c(1, 1, 3, 3, 3, 4, 3)
- dgts = 24 : c(2, 2, 3, 3, 3, 4, 3)
- dgts = 25 : c(3, 3, 3, 3, 3, 4, 3)
- dgts = 26 : c(4, 4, 3, 3, 3, 4, 3)
- dgts = 27 : c(5, 5, 3, 3, 3, 4, 3)

Examples

```
mat   <- matrix(runif(35), ncol=7)
coefk <- mat[1,]

roundcoefk(coefk, dgts = 2, parnames = FALSE)
```

```
roundcoefk(coefk, dgts = 15)
roundcoefk(mat, dgts = 15)
```

---

tData *Datasets dfData, mData, tData, xData, zData, extractData : tData*

Description

A list of datasets in data.frame, matrix, timeSeries, xts and zoo formats. This is the timeSeries format. Visit extractData for more information.

---

xData *Datasets dfData, mData, tData, xData, zData, extractData : xData*

Description

A list of datasets in data.frame, matrix, timeSeries, xts and zoo formats. This is the xts format. Visit extractData for more information.

---

zData *Datasets dfData, mData, tData, xData, zData, extractData : zData*

Description

A list of datasets in data.frame, matrix, timeSeries, xts and zoo formats. This is the zoo format. Visit extractData for more information.

Index