Package 'qrmtools'

May 29, 2016

Version 0.0-6

Encoding UTF-8

Title Tools for Quantitative Risk Management

Description Functions and data sets for reproducing selected results from the book ``Quantitative Risk Management: Concepts, Techniques and Tools". Furthermore, new developments and auxiliary functions for Quantitative Risk Management practice.

Author Marius Hofert [aut, cre], Kurt Hornik [aut]

Maintainer Marius Hofert <marius.hofert@uwaterloo.ca>

Depends R (>= 3.2.0)

Imports graphics, lattice, quantmod, Quandl, zoo, methods, grDevices, stats, rugarch, utils, xts

Suggests combinat, copula, knitr, mvtnorm, QRM, sfsmisc

Enhances

License GPL-2 | GPL-3

NeedsCompilation yes

VignetteBuilder knitr

Repository CRAN

Date/Publication 2016-05-29 21:14:06

R topics documented:

ARMA_GARCH *Fitting ARMA-GARCH Processes*

Description

Fail-safe componentwise fitting of univariate ARMA-GARCH processes.

Usage

```
fit_ARMA_GARCH(x, ugarchspec.list = ugarchspec(), verbose = TRUE, ...)
```

Arguments

x	A matrix-like data structure, possibly an xts object.
ugarchspec.list	An object of class uGARCHspec (as returned by ugarchspec()) or a list of such. In case of a list, its length has to be equal to the number of columns of x. ugarchspec.list provides the ARMA-GARCH specifications for each of the time series (columns of x).
verbose	A logical indicating whether verbose output is given.
...	Additional arguments passed to the underlying ugarchfit().

Value

If x consists of one column only (e.g. a vector), ARMA_GARCH() returns the fitted object; otherwise it returns a list of such.

Author(s)

Marius Hofert

Examples

```
library(rugarch)
library(copula)

## Read the data, build -log-returns
data(SMI.12) # Swiss Market Index data
stocks <- c("CSGN", "BAER", "UBSN", "SREN", "ZURN") # components we work with
x <- SMI.12[, stocks]
X <- -log_returns(x)
n <- nrow(X)
d <- ncol(X)
```

```
## Fit ARMA-GARCH models to the -log-returns
## Note: - Our choice here is purely for demonstration purposes.
##         The models are not necessarily adequate
##       - The sample size n is *too* small here for properly capturing GARCH effects.
##         Again, this is only for demonstration purposes here.
uspec <- c(rep(list(ugarchspec(distribution.model = "std")), d-2), # ARMA(1,1)-GARCH(1,1)
           list(ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(2,2)),
                           distribution.model = "std")),
           list(ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(2,1)),
                           mean.model = list(armaOrder = c(1,2), include.mean = TRUE),
                           distribution.model = "std")))
fitAG <- fit_ARMA_GARCH(X, ugarchspec.list = uspec)
str(fitAG, max.level = 1) # list with components fit, warning, error
stopifnot(sapply(fitAG$error, is.null)) # NULL = no error
stopifnot(sapply(fitAG$warning, is.null)) # NULL = no warning

## Pick out the standardized residuals, plot them and fit a t copula to them
## Note: ugarchsim() needs the residuals to be standardized; so working with
##       standardize = FALSE still requires to simulate them from the
##       respective standardized marginal distribution functions.
Z <- sapply(fitAG$fit, residuals, standardize = TRUE)
U <- pobs(Z)
pairs(U, gap = 0)
fitC <- fitCopula(tCopula(dim = d, dispstr = "un"), data = U, method = "mpl")

## Simulate (standardized) Z
set.seed(271)
U. <- rCopula(n, fitC@copula) # simulate from the fitted copula
nu <- sapply(1:d, function(j) fitAG$fit[[j]]@fit$coef["shape"]) # extract (fitted) d.o.f. nu
Z. <- sapply(1:d, function(j) sqrt((nu[j]-2)/nu[j]) * qt(U.[,j], df = nu[j])) # Z

## Simulate from fitted model
X. <- sapply(1:d, function(j)
    fitted(ugarchsim(fitAG$fit[[j]], n.sim = n, m.sim = 1, startMethod = "sample",
                     rseed = 271, custom.dist = list(name = "sample",
                                                     distfit = Z.[,j, drop = FALSE]))))

## Plots original vs simulated -log-returns
opar <- par(no.readonly = TRUE)
layout(matrix(1:(2*d), ncol = d)) # layout
ran <- range(X, X.)
for(j in 1:d) {
    plot(X[,j],  type = "l", ylim = ran, ylab = paste(stocks[j], "-log-returns"))
    plot(X.[,j], type = "l", ylim = ran, ylab = "Simulated -log-returns")
}
par(opar)
```

Black_Scholes *Black–Scholes formula and the Greeks*

Description

Compute the Black–Scholes formula and the Greeks.

Usage

```
Black_Scholes(t, S, r, sigma, K, T, type=c("call", "put"))
Black_Scholes_Greeks(t, S, r, sigma, K, T)
```

Arguments

`t`	initial or current time t (in years).
`S`	stock price at time t.
`r`	risk-free annual interest rate.
`sigma`	annual volatility (standard deviation).
`K`	strike.
`T`	maturity (in years).
`type`	`character` string indicating whether the price of a call (the default) or of put option is to be computed.

Value

`Black_Scholes()` returns the value of a European-style call or put option (depending on the chosen `type`) on a non-dividend paying stock.

`Black_Scholes_Greeks()` returns the first-order derivatives delta, theta, rho, vega and the second-order derivatives gamma, vanna and vomma (in this order).

Author(s)

Marius Hofert

References

McNeil, A. J., Frey, R., and Embrechts, P. (2015). *Quantitative Risk Management: Concepts, Techniques, Tools*. Princeton University Press.

`catch` *Catching Results, Warnings and Errors Simultaneously*

Description

Catches results, warnings and errors.

Usage

```
catch(expr)
```

Arguments

expr	An expression to be evaluated, typically a function call.

Details

This function is particularly useful for large(r) simulation studies to not fail until finished.

Value

A list with components:

value	The value of expr or NULL in case of an error.
warning	The warning message (see simpleWarning or warning()) or NULL in case of no warning.
error	The error message (see simpleError or stop()) or NULL in case of no error.

Author(s)

Marius Hofert (based on doCallWE() and tryCatch.W.E() in the R package **simsalapar**).

Examples

```
catch(log(2))
catch(log(-1))
catch(log("a"))
```

---

density_plot_matrix *Density Plot of the Values from a Lower Triangular Matrix*

Description

Density plot of all values in the lower triangular part of a matrix.

Usage

```
density_plot_matrix(x, xlab = "Entries in the lower triangular matrix",
                    main = "", text = NULL, side = 4, line = 1, adj = 0, ...)
```

Arguments

x	A matrix-like object.
xlab	The x-axis label.
main	The title.
text	See mtext(). The text = "", it is omitted.
side	See mtext().
line	See mtext().
adj	See mtext().
...	Additional arguments passed to the underlying plot().

Details

density_plot_matrix() is typically used for symmetric matrices (like correlation matrices, matrices of pairwise Kendall's tau or tail dependence parameters) to check the distribution of their off-diagonal entries.

Value

invisible().

Author(s)

Marius Hofert

Examples

```
## Generate a random correlation matrix
d <- 50
L <- diag(1:d)
set.seed(271)
L[lower.tri(L)] <- runif(choose(d,2))
Sigma <- L
P <- cor(Sigma)
## Density of its lower triangular entries
density_plot_matrix(P)
```

---

get_data *Tools for Getting and Working with Data*

Description

Download (and possibly) merge data from freely available databases.

Usage

```
get_data(x, from=NULL, to=NULL, src=c("yahoo", "quandl", "oanda", "FRED", "google"),
         FUN=NULL, verbose=TRUE, warn=TRUE, ...)
```

Arguments

x	A vector of ticker symbols (e.g. "^GSPC" if src="yahoo" or "EUR/USD" if src="oanda").
from	start date as a Date object or character string (in international date format "yyyy-mm-dd"); if NULL, the earliest date with available data is picked.
to	end date as a Date object or character string (in international date format "yyyy-mm-dd"); if NULL, the last date with available data is picked.
src	A character string specifying the data source (e.g. "yahoo" for stocks or "oanda" for FX data); see getSymbols() and Quandl().

FUN	A function to be applied to the data before being returned. This can be **the identity** if the data could not be retrieved (and is thus replaced by NA); **the given** FUN if FUN has been provided; **a useful default** if FUN=NULL; the default uses the adjusted close price Ad() if src="yahoo", the close price Cl() if src="google" and the identity otherwise.
verbose	A logical indicating whether progress monitoring should be done.
warn	A logical indicating whether a warning is given showing the error message when fetching x fails.
...	Additional arguments passed to the underlying function getSymbols() from **quantmod** or Quandl() from **Quandl** (if src="quandl").

Details

FUN is typically one of **quantmod**'s Op, Hi, Lo, Cl, Vo, Ad or one of the combined functions OpCl, ClCl, HiCl, LoCl, LoHi, OpHi, OpLo, OpOp.

Value

An xts object containing the data with column name(s) adjusted to be the ticker symbol (in case lengths match; otherwise the column names are not adjusted); NA if data is not available.

Author(s)

Marius Hofert

Examples

```
## Get stock and volatility data (for all available trading days)
dat <- get_data(c("^GSPC", "^VIX")) # note: this needs a working internet connection
## Plot them (Alternative: plot.xts() from xtsExtra)
library(zoo)
plot.zoo(dat, screens = 1, main = "", xlab = "Trading day", ylab = "Value")
```

GEV *Generalized Extreme Value Distribution*

Description

Density, distribution function, quantile function and random variate generation for the generalized extreme value distribution (GEV).

Usage

```
dGEV(x, xi, mu=0, sigma=1, log=FALSE)
pGEV(q, xi, mu=0, sigma=1, lower.tail=TRUE, log.p=FALSE)
qGEV(p, xi, mu=0, sigma=1, lower.tail=TRUE, log.p=FALSE)
rGEV(n, xi, mu=0, sigma=1)
```

Arguments

`x, q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations.
`xi`	GEV shape parameter, a real number.
`mu`	GEV location parameter, a real number.
`sigma`	GEV scale parameter, a positive number.
`lower.tail`	logical; if TRUE (default) probabilities are $P(X \leq x)$ otherwise, $P(X > x)$.
`log, log.p`	logical; if TRUE, probabilities p are given as log(p).

Details

The distribution function of the generalized extreme value distribution is given by

$$F(x) = \begin{cases} \exp(-(1-\xi(x-\mu)/\sigma)^{-1/\xi}), & \text{if } \xi \neq 0,\ 1+\xi(x-\mu)/\sigma > 0, \\ \exp(-e^{-(x-\mu)/\sigma}), & \text{if } \xi = 0, \end{cases}$$

where $\sigma > 0$.

Value

`dGEV()` computes the density, `pGEV()` the distribution function, `qGEV()` the quantile function and `rGEV()` random variates of the generalized extreme value distribution.

Author(s)

Marius Hofert

References

McNeil, A. J., Frey, R., and Embrechts, P. (2015). *Quantitative Risk Management: Concepts, Techniques, Tools*. Princeton University Press.

Examples

```
## Basic sanity checks
plot(pGEV(rGEV(1000, xi=0.5), xi=0.5)) # should be U[0,1]
curve(dGEV(x, xi=0.5), from=-3, to=5)
```

GPD *(Generalized) Pareto Distribution*

Description

Density, distribution function, quantile function and random variate generation for the (generalized) Pareto distribution (GPD).

Usage

```
dGPD(x, xi, beta, log=FALSE)
pGPD(q, xi, beta, lower.tail=TRUE, log.p=FALSE)
qGPD(p, xi, beta, lower.tail=TRUE, log.p=FALSE)
rGPD(n, xi, beta)

dPar(x, theta, log=FALSE)
pPar(q, theta, lower.tail=TRUE, log.p=FALSE)
qPar(p, theta, lower.tail=TRUE, log.p=FALSE)
rPar(n, theta)
```

Arguments

x, q	vector of quantiles.
p	vector of probabilities.
n	number of observations.
xi	GPD shape parameter, a real number.
beta	GPD scale parameter, a positive number.
theta	Pareto parameter, a positive number.
lower.tail	logical; if TRUE (default) probabilities are $P(X \leq x)$ otherwise, $P(X > x)$.
log, log.p	logical; if TRUE, probabilities p are given as log(p).

Details

The distribution function of the generalized Pareto distribution is given by

$$F(x) = \begin{cases} 1 - (1 + \xi x/\beta)^{-1/\xi}, & \text{if } \xi \neq 0, \\ 1 - \exp(-x/\beta), & \text{if } \xi = 0, \end{cases}$$

where $\beta > 0$ and $x \geq 0$ if $\xi \geq 0$ and $x \in [0, -\beta/\xi]$ if $\xi < 0$.

The distribution function of the (standard) Pareto distribution is given by

$$F(x) = 1 - (1 + x)^{-\theta}, \; x \geq 0,$$

where $\theta > 0$.

In contrast to dGPD(), pGPD(), qGPD() and rGPD(), the functions dPar(), pPar(), qPar() and rPar() are vectorized in both their main argument and θ.

Value

dGPD() computes the density, pGPD() the distribution function, qGPD() the quantile function and rGPD() random variates of the generalized Pareto distribution.

Similary for dPar(), pPar(), qPar() and rPar() for the (standard) Pareto distribution.

Author(s)

Marius Hofert

References

McNeil, A. J., Frey, R., and Embrechts, P. (2015). *Quantitative Risk Management: Concepts, Techniques, Tools*. Princeton University Press.

Examples

```
## Basic sanity checks
plot(pGPD(rGPD(1000, xi=0.5, beta=3), xi=0.5, beta=3)) # should be U[0,1]
curve(dGPD(x, xi=0.5, beta=3), from=-1, to=5)
```

---

plot_matrix *Graphical Tool for Visualizing Matrices*

Description

Plot of a matrix.

Usage

```
plot_matrix(x, xlab = "Column", ylab = "Row",
           scales = list(alternating = c(1,1), tck = c(1,0), x = list(rot = 90)),
           at = NULL, colorkey = NULL, col = c("royalblue3", "white", "maroon3"),
             col.regions = NULL, ...)
```

Arguments

x	A matrix-like object.
xlab	The x-axis label.
ylab	The y-axis label.
scales	See levelplot(); if NULL, labels and ticks are omitted.
at	See levelplot(). If NULL, a useful default is computed based on the given values in x.
colorkey	See levelplot(). If NULL, a useful default is computed based on at.
col	A vector of length 3 providing the color key's default colors.
col.regions	See levelplot(). If NULL, a useful default is computed based on at.
...	Additional arguments passed to the underlying function levelplot().

Details

Plot of a matrix.

Value

The plot, a Trellis object.

Author(s)

Marius Hofert

Examples

```
## Generate a random correlation matrix
d <- 50
L <- diag(1:d)
set.seed(271)
L[lower.tri(L)] <- runif(choose(d,2))
Sigma <- L
P <- cor(Sigma)
## Default
plot_matrix(P)
## Default if nonnegative
plot_matrix(abs(P))
## Without diagonal
P. <- abs(P)
diag(P.) <- NA
plot_matrix(P.)
## Default if nonpositive
plot_matrix(-abs(P))
## Extending the color key to [-1,1] with darker color for |rho| >> 0
## Note: When specifying 'at', one most likely also wants to provide 'col.regions'
plot_matrix(P, at = seq(-1, 1, length.out = 200),
          col.regions = grey(c(seq(0, 1, length.out = 100), seq(1, 0, length.out = 100))))
```

---

plot_NA *Graphical Tool for Visualizing NAs in a Data Set*

Description

Plot NAs in a data set.

Usage

```
plot_NA(x, col = c("black", "white"), xlab = "Time", ylab = "Component",
        text = "Black: NA; White: Available data",
        side = 4, line = 1, adj = 0, ...)
```

Arguments

x	A matrix (ideally an xts object).
col	A bivariate vector containing the colors for missing and available data, respectively.
xlab	The x-axis label.
ylab	The y-axis label.
text	See mtext(). The text = "", it is omitted.
side	See mtext().
line	See mtext().
adj	See mtext().
...	Additional arguments passed to the underlying function image().

Details

Indicate NAs in a data set.

Value

invisible().

Author(s)

Marius Hofert

Examples

```
## Generate some data
n <- 1000 # sample size
d <- 100 # dimension
set.seed(271) # set seed
x <- matrix(runif(n*d), ncol = d) # generate data

## Assign missing data
k <- ceiling(d/4) # fraction of columns with some NAs
j <- sample(1:d, size = k) # columns j with NAs
i <- sample(1:n, size = k) # 1:i will be NA in each column j
X <- x
for(k. in seq_len(k)) X[1:i[k.], j[k.]] <- NA # put in NAs

## Plot NAs
plot_NA(X) # indicate NAs
```

returns *Compute Log-Returns or the Inverse Transformation*

Description

Compute log-returns or data from given log-returns (the inverse transformation).

Usage

```
log_returns(x, inverse = FALSE, start.value, drop = TRUE)
```

Arguments

x	A matrix of values to be turned into log-returns (if `inverse = FALSE`) or log-returns (if `inverse = TRUE`).
inverse	A logical indicating whether the inverse transformation (data from given log-returns) is to be computed (if TRUE, this requires `start.value` to be specified).
start.value	If `inverse = TRUE`, the last available value of the time series to be constructed from the given log-returns `x`.
drop	A logical indicating whether 1-column matrices (or vectors) are returned as vectors.

Details

For negative log-returns, use `-log_returns(x)` or `log_returns(-x, inverse = TRUE, start.value = ...)`.

Value

A matrix with the same number of columns as `x` (if one, then a vector), just one row less (if `inverse = FALSE`) or one row more (if `inverse = TRUE`).

Author(s)

Marius Hofert

Examples

```
## Generate some data
n <- 1000
set.seed(271)
S <- cumsum(sample(c(-1,1), size = n, replace = TRUE))
S <- if(any(S <= 0)) S <- 1 - min(S) + S

## Log-returns
X <- log_returns(S) # build log-returns
Y <- log_returns(X, inverse = TRUE, start.value = S[1]) # transform back (first date is lost)
stopifnot(all.equal(S, Y))
```

```
## -Log-returns
X <- -log_returns(S) # build -log-returns
Y <- log_returns(-X, inverse = TRUE, start.value = S[1]) # transform back
stopifnot(all.equal(S, Y))
```

`risk_measures` *Risk Measures*

Description

Computing risk measures for various distributions.

Usage

```
VaR_t(alpha, mu=0, sigma=1, df=Inf)
ES_t(alpha, mu=0, sigma=1, df=Inf)
VaR_Par(alpha, theta)
ES_Par(alpha, theta)
```

Arguments

`alpha`	confidence level (in [0,1]).
`mu`	location parameter.
`sigma`	scale parameter, a positive number.
`df`	degrees of freedom, a positive number; choose `df=Inf` for the normal distribution.
`theta`	Pareto parameter, a positive number.

Details

The distribution function of the (standard) Pareto distribution is given by

$$F(x) = 1 - (1 + x)^{-\theta},\ x \geq 0,$$

where $\theta > 0$.

Value

`VaR_t()`, `ES_t()` compute Value-at-Risk and expected shortfall for the t (or normal) distribution.

`VaR_Par()`, `ES_Par()` compute Value-at-Risk and expected shortfall for the Pareto distribution.

Author(s)

Marius Hofert

References

McNeil, A. J., Frey, R., and Embrechts, P. (2015). *Quantitative Risk Management: Concepts, Techniques, Tools*. Princeton University Press.

Examples

```
## Example 2.16 in McNeil, Frey, Embrechts (2015)
V <- 10000 # value of the portfolio today
sig <- 0.2/sqrt(250) # daily volatility (annualized volatility of 20%)
nu <- 4 # degrees of freedom for the t distribution
alpha <- seq(0.001, 0.999, length.out=256) # confidence levels
VaRnorm <- VaR_t(alpha, sigma=V*sig, df=Inf)
VaRt4 <- VaR_t(alpha, sigma=V*sig*sqrt((nu-2)/nu), df=nu)
ESnorm <- ES_t(alpha, sigma=V*sig, df=Inf)
ESt4 <- ES_t(alpha, sigma=V*sig*sqrt((nu-2)/nu), df=nu)
ran <- range(VaRnorm, VaRt4, ESnorm, ESt4)
plot(alpha, VaRnorm, type="l", ylim=ran, xlab=expression(alpha), ylab="")
lines(alpha, VaRt4, col="royalblue3")
lines(alpha, ESnorm, col="darkorange2")
lines(alpha, ESt4, col="maroon3")
legend("bottomright", bty="n", lty=rep(1,4), col=c("black",
       "royalblue3", "darkorange3", "maroon3"),
       legend=c(expression(VaR[alpha]~~"for normal model"),
                expression(VaR[alpha]~~"for "*t[4]*" model"),
                expression(ES[alpha]~~"for normal model"),
                expression(ES[alpha]~~"for "*t[4]*" model")))
```

VaR_bounds *Best and Worst Value-at-Risk for Given Margins*

Description

Compute the best and worst Value-at-Risk for given marginal distributions.

Usage

```
crude_VaR_bounds(alpha, qF, ...)
VaR_bounds_hom(alpha, d, method=c("Wang", "Wang.Par", "dual"),
               interval=NULL, tol=NULL, ...)
dual_bound(s, d, pF, tol=.Machine$double.eps^0.25, ...)

rearrange(X, tol=0, tol.type=c("relative", "absolute"), max.ra=Inf,
          method=c("worst", "best"), sample=TRUE, is.sorted=FALSE, trace=FALSE)
RA(alpha, qF, N, abstol=0, max.ra=Inf, method=c("worst", "best"),
   sample=TRUE)
ARA(alpha, qF, N.exp=seq(8, 19, by=1), reltol=c(0, 0.01),
    max.ra=10*length(qF), method=c("worst", "best"), sample=TRUE)
```

Arguments

alpha	Value-at-Risk confidence level (e.g., 0.99).
d	Dimension (number of risk factors; ≥ 2).
qF	The marginal quantile function (in the homogeneous case) or a d-list containing the marginal quantile functions (in the general case).
method	A character string. For VaR_bounds_hom(): method="Wang" and method="Wang.Par" apply the approach of McNeil et al. (2015, Proposition 8.32) for computing best (i.e., smallest) and worst (i.e., largest) Value-at-Risk. The latter method assumes Pareto margins and thus does not require numerical integration. method="dual" applies the dual bound approach as in Embrechts et al. (2013, Proposition 4) for computing worst Value-at-Risk (no value for the best Value-at-Risk can be obtained with this approach and thus NA is returned for the best Value-at-Risk). rearrange(), RA(), ARA(): method indicates whether bounds for the best or for the worst Value-at-Risk should be computed. These bounds are termed $\underline{s}_N$ and $\overline{s}_N$ in the literature (and below) and are theoretically not guaranteed bounds of (best or worst) Value-at-Risk; however, they are treated as such in practice and are typically in line with results from VaR_bounds_hom() in the homogeneous case, for example.
interval	Initial interval (a numeric(2)) for computing worst Value-at-Risk. If not provided, these are the defaults chosen: method="Wang": The initial interval is $[0, (1-\alpha)/d]$. method="Wang.Par": The initial interval is $[c_l, c_u]$, where c_l and c_u are chosen as in Hofert et al. (2015). method="dual": In this case, no good defaults are known. Note that the lower endpoint of the initial interval has to be sufficiently large in order for the the inner root-finding algorithm to find a root; see Details.
tol	VaR_bounds_hom(): The tolerance for uniroot() for computing worst Value-at-Risk. This defaults (for tol=NULL) to $2.2204 * 10^{-16}$ for method="Wang" or method="Wang.Par" (where a smaller tolerance is crucial) and to uniroot()'s default .Machine$double.eps^0.25 otherwise. Note that for method="dual", tol is used for both the outer and the inner root-finding procedure. rearrange(): (Absolute or relative) tolerance to determine (the individual) convergence. This should normally be a number greater than or equal to 0, but we also allow tol=NULL which rearranges the columns until all columns are oppositely ordered to the sum of all other columns (this should only be used by experts).
tol.type	character string indicating the type of convergence tolerance function to be used ("relative" for relative tolerance and "absolute" for absolute tolerance).
s	Dual bound evaluation point.
pF	The marginal loss distribution function.
X	An (N, d)-matrix of quantiles (to be rearranged). If is.sorted it is assumed that the columns of X are sorted in *increasing* order.

max.ra — Maximal number of (considered) column rearrangements of the underlying matrix of quantiles (can be set to Inf).

N — The number of discretization points.

N.exp — The exponents of the number of discretization points (a vector) over which the algorithm iterates to find the smallest number of discretization points for which the desired accuracy (specified by reltol) is attained; for each number of discretization points, at most max.ra-many column rearrangements are of the underlying matrix of quantiles are considered.

abstol — Absolute convergence tolerance ϵ to determine the individual convergence, i.e., the change in the computed minimal (for method="worst") or maximal (for method="best") row sums for the lower bound $\underline{s}_N$ and the upper bound $\overline{s}_N$. abstol is typically ≥ 0; it can also be NULL, see tol above.

reltol — A vector of length one or two containing the relative convergence tolerances. If reltol is of length two, the first component is the the individual relative tolerance (used to determine convergence of the minimal (for method="worst") or maximal (for method="best") row sums for $\underline{s}_N$ and $\overline{s}_N$) and the second component is the joint relative tolerance (i.e., the relative tolerance between the computed $\underline{s}_N$ and $\overline{s}_N$ with respect to $\overline{s}_N$). If reltol is of length one, it denotes the joint relative tolerance; the individual relative tolerance is then taken as NULL (see tol above).

sample — A logical indicating whether each column of the two underlying matrices of quantiles (see Step 3 of the Rearrangement Algorithm in Embrechts et al. (2013)) are randomly permuted before the rearrangements begin. This typically has quite a positive effect on run time (as most of the time is spent (oppositely) ordering columns).

is.sorted — A logical indicating whether the columns of X are sorted in increasing order.

trace — A logical indicating whether the underlying matrix is printed after each rearrangement step. See vignette("VaR_bounds", package="qrmtools") for how to interpret the output.

... — crude_VaR_bounds(): Ellipsis argument passed to (all provided) quantile functions.

VaR_bounds_hom(): The case method="Wang" requires the quantile function qF() to be provided and additional arguments passed via the ellipsis argument are passed on to
the underlying integrate(). For method="Wang.Par" the ellipsis argument must contain the parameter $\theta > 0$ of the Pareto distribution. For method="dual", the ellipsis argument must contain the distribution function pF() and the initial interval interval for the outer root finding procedure (not for d=2); additional arguments are passed on to the underlying integrate() for computing the dual bound $D(s)$.

dual_bound(): The ellipsis argument is passed to the underlying integrate().

Details

For d=2, VaR_bounds_hom() uses the method of Embrechts et al. (2013, Proposition 2). For method="Wang" and method="Wang.Par" the method presented in McNeil et al. (2015, Prop.

8.32) is implemented; this goes back to Embrechts et al. (2014, Prop. 3.1; note that the published version of this paper contains typos for both bounds). This requires one `uniroot()` and, for the generic `method="Wang"`, one `integrate()`. The critical part for the generic `method="Wang"` is the lower endpoint of the initial interval for `uniroot()`. If the (marginal) distribution function has finite first moment, this can be taken as 0. However, if it has infinite first moment, the lower endpoint has to be positive (but must lie below the unknown root). Note that the upper endpoint $(1-\alpha)/d$ also happens to be a root and thus one needs a proper initional interval containing the root and being strictly contained in $(0,(1-\alpha)/d$. In the case of Pareto margins, Hofert et al. (2015) have derived such an initial (which is used by `method="Wang.Par"`). Also note that the chosen smaller default tolerances for `uniroot()` in case of `method="Wang"` and `method="Wang.Par"` are crucial for obtaining reliable Value-at-Risk values; see Hofert et al. (2015).

For `method="dual"` for computing worst Value-at-Risk, the method presented of Embrechts et al. (2013, Proposition 4) is implemented. This requires two (nested) `uniroot()`, and an `integrate()`. For the inner root-finding procedure to find a root, the lower endpoint of the provided initial `interval` has to be "sufficiently large".

Note that these approaches for computing the Value-at-Risk bounds in the homogeneous case are numerically non-trivial; see the source code and `vignette("VaR_bounds", package="qrmtools")` for more details. As a rule of thumb, use `method="Wang"` if you have to (i.e., if the margins are not Pareto) and `method="Wang.Par"` if you can (i.e., if the margins are Pareto). It is not recommended to use (the numerically even more challenging) `method="dual"`.

Concerning the inhomogeneous case, `rearrange()` is an auxiliary function which is called by `RA()` and `ARA()`. After a column of `X` has been rearranged, the tolerance between the minimal (for the worst Value-at-Risk) or maximal (for the best Value-at-Risk) row sum after this rearrangement and the one of d steps before (so typically when that column was rearranged the last time) is computed and convergence determined. For performance reasons, no checking is done and `rearrange()` can change in future versions to (futher) improve run time. Overall it should only be used by experts.

For the Rearrangement Algorithm `RA()`, convergence of $\underline{s}_N$ and $\overline{s}_N$ is determined if the minimal (for the worst Value-at-Risk) or maximal (for the best Value-at-Risk) row sum satisfies the specified `abstol` (so $\leq \epsilon$) after at most `max.ra`-many column rearrangements. This is different from Embrechts et al. (2013) who use $< \epsilon$ and only check for convergence after an iteration through all columns of the underlying matrix of quantiles has been completed.

For the Adaptive Rearrangement Algorithm `ARA()`, convergence of $\underline{s}_N$ and $\overline{s}_N$ is determined if, after at most `max.ra`-many column rearrangements, the (the individual relative tolerance) `reltol[1]` is satisfied *and* the relative (joint) tolerance between both bounds is at most `reltol[2]`.

Note that both `RA()` and `ARA()` need to evalute the 0-quantile (for the lower bound for the best Value-at-Risk) and the 1-quantile (for the upper bound for the worst Value-at-Risk). As the algorithms can only handle finite values, the 0-quantile and the 1-quantile need to be adjusted if infinite. Instead of the 0-quantile, the $\alpha/(2N)$-quantile is computed and instead of the 1-quantile the $\alpha+(1-\alpha)(1-1/(2N))$-quantile is computed for such margins (if the 0-quantile or the 1-quantile is finite, no adjustment is made).

As a rule of thumb (see the examples below, `vignette("VaR_bounds", package="qrmtools")` and Hofert et al. (2015) for the reasons), it is recommended to use `ARA()` instead of `RA()`.

On the theoretical side, let us again stress the following. `rearrange()`, `RA()` and `ARA()` compute $\underline{s}_N$ and $\overline{s}_N$ which are, from a practical point of view, treated as bounds for the worst (i.e., largest) or the best (i.e., smallest) Value-at-Risk (whatever is chosen with `method`), but which are not known to be such bounds from a theoretical point of view; see also above. Calling them "bounds" for worst

or best Value-at-Risk is thus theoretically not correct (unless proven) but "practical". The literature thus speaks of $(\underline{s}_N, \overline{s}_N)$ as the rearrangement range (rather than an interval containing worst or best Value-at-Risk).

Value

`crude_VaR_bounds()` returns crude lower and upper bounds for Value-at-Risk at confidence level α for any d-dimensional model with marginal quantile functions specified by `qF`.

`VaR_bounds_hom()` returns the best and worst Value-at-Risk at confidence level α for d risks with equal distribution function specified by the ellipsis ....

`dual_bound()` returns the value of the dual bound $D(s)$ as given in Embrechts, Puccetti, Rüschendorf (2013, Eq. (12)).

`rearrange()` returns a `list` containing

`bound`: The computed $\underline{s}_N$ or $\overline{s}_N$.

`tol`: The reached tolerance (i.e., the (absolute or relative) change of the minimal (for `method="worst"`) or maximal (for `method="best"`) row sum after the last rearranged column).

`converged`: A `logical` indicating whether the desired (absolute or relative) tolerance `tol` has been reached.

`m.row.sums`: A `vector` containing the computed minimal (for `method="worst"`) or maximal (for `method="best"`) row sums after each (considered) column rearrangement.

`X.rearranged`: An (`N`, `d`)-`matrix` containing the rearranged `X`.

`RA()` returns a `list` containing

`bounds`: A bivariate vector containing the computed $\underline{s}_N$ and $\overline{s}_N$ (the so-called rearrangement range) which are typically treated as bounds for (worst or best) Value-at-Risk; see also above.

`rel.ra.gap`: The reached relative tolerance (also known as relative rearrangement gap) between $\underline{s}_N$ and $\overline{s}_N$ computed with respect to $\overline{s}_N$.

`ind.abs.tol`: A bivariate `vector` containing the reached individual absolute tolerances (i.e., the absolute change of the minimal (for `method="worst"`) or maximal (for `method="best"`) row sums for computing $\underline{s}_N$ and $\overline{s}_N$; see also `tol` returned by `rearrange()` above).

`converged`: A bivariate `logical` vector indicating convergence of the computed $\underline{s}_N$ and $\overline{s}_N$ (i.e., whether the desired tolerances were reached).

`num.ra`: A bivariate vector containing the number of column rearrangments of the underlying matrices of quantiles for $\underline{s}_N$ and $\overline{s}_N$.

`m.row.sums`: A `list` of length two containing minimal (for `method="worst"`) or maximal (for `method="best"`) row sums (after each (considered) column rearrangement) for the computed $\underline{s}_N$ and $\overline{s}_N$.

`X`: The initially constructed (`N`, `d`)-matrices of quantiles for computing $\underline{s}_N$ and $\overline{s}_N$.

`X.rearranged`: The rearranged matrices `X` (for $\underline{s}_N$ and $\overline{s}_N$).

`ARA()` returns a `list` containing

`bounds`: See `RA()`.

`rel.ra.gap`: See `RA()`.

`rel.tol`: A trivariate `vector` containing the reached individual relative tolerances and the reached joint relative tolerance (computed with respect to $\overline{s}_N$).

`converged`: A trivariate `logical vector` indicating individual convergence of the computed $\underline{s}_N$ (first entry) and $\overline{s}_N$ (second entry) and indicating joint convergence of the two bounds according to the attained joint relative tolerance (third entry).

`N.used`: The actual `N` used for computing the (final) $\underline{s}_N$ and $\overline{s}_N$.

`num.ra`: See `RA()`; computed for `N.used`.

`m.row.sums`: See `RA()`; computed for `N.used`.

`X`: See `RA()`; computed for for `N.used`.

`X.rearranged`: See `RA()`; computed for for `N.used`.

Author(s)

Marius Hofert

References

Embrechts, P., Puccetti, G., Rüschendorf, L., Wang, R., Beleraj, A. (2014). An Academic Response to Basel 3.5. *Risks* **2**(1), 25–48.

Embrechts, P., Puccetti, G., Rüschendorf, L. (2013). Model uncertainty and VaR aggregation. *Journal of Banking \& Finance* **37**, 2750–2764.

McNeil, A. J., Frey, R., and Embrechts, P. (2015). *Quantitative Risk Management: Concepts, Techniques, Tools*. Princeton University Press.

Hofert, M., Memartoluie, A., Saunders, D., Wirjanto, T. (2015). Improved Algorithms for Computing Worst Value-at-Risk: Numerical Challenges and the Adaptive Rearrangement Algorithm. See http://arxiv.org/abs/1505.02281.

See Also

`vignette("VaR_bounds", package="qrmtools")` for more example calls, numerical challenges encoutered and a comparison of the different methods for computing the worst (i.e., largest) Value-at-Risk.

Examples

```
## Pareto setup
alpha <- 0.99 # VaR confidence level
th <- 2 # Pareto parameter theta
qF <- function(p, theta=th) qPar(p, theta=theta) # Pareto quantile function
pF <- function(q, theta=th) pPar(q, theta=theta) # Pareto distribution function


## d=2: Compute best/worst VaR explicitly (hom. case) and compare with (A)RA ###

d <- 2 # dimension

## Explicit
VaRbounds <- VaR_bounds_hom(alpha, d=d, qF=qF) # (best VaR, worst VaR)
```

```
## Adaptive Rearrangement Algorithm (ARA)
set.seed(271) # set seed (for reproducibility)
ARAbest  <- ARA(alpha, qF=rep(list(qF), d), method="best")
ARAworst <- ARA(alpha, qF=rep(list(qF), d))

## Rearrangement Algorithm (RA) with N as in ARA()
RAbest  <- RA(alpha, qF=rep(list(qF), d), N=ARAbest$N.used, method="best")
RAworst <- RA(alpha, qF=rep(list(qF), d), N=ARAworst$N.used)

## Compare
stopifnot(all.equal(c(ARAbest$bounds[1], ARAbest$bounds[2],
                      RAbest$bounds[1],  RAbest$bounds[2]),
                    rep(VaRbounds[1], 4), tolerance=0.004, check.names=FALSE))
stopifnot(all.equal(c(ARAworst$bounds[1], ARAworst$bounds[2],
                      RAworst$bounds[1],  RAworst$bounds[2]),
                    rep(VaRbounds[2], 4), tolerance=0.003, check.names=FALSE))


## d=8: Compute best/worst VaR (hom. case) and compare with (A)RA ###############

d <- 8 # dimension

## Compute VaR bounds with various methods
I <- crude_VaR_bounds(alpha, qF=rep(list(qF), d)) # crude bound
VaR.W     <- VaR_bounds_hom(alpha, d=d, method="Wang", qF=qF)
VaR.W.Par <- VaR_bounds_hom(alpha, d=d, method="Wang.Par", theta=th)
VaR.dual  <- VaR_bounds_hom(alpha, d=d, method="dual", interval=I, pF=pF)

## Adaptive Rearrangement Algorithm (ARA) (with different relative tolerances)
set.seed(271) # set seed (for reproducibility)
ARAbest  <- ARA(alpha, qF=rep(list(qF), d), reltol=c(0.001, 0.01), method="best")
ARAworst <- ARA(alpha, qF=rep(list(qF), d), reltol=c(0.001, 0.01))

## Rearrangement Algorithm (RA) with N as in ARA and abstol (roughly) chosen as in ARA
RAbest  <- RA(alpha, qF=rep(list(qF), d), N=ARAbest$N.used,
              abstol=mean(tail(abs(diff(ARAbest$m.row.sums$low)), n=1),
                          tail(abs(diff(ARAbest$m.row.sums$up)), n=1)),
              method="best")
RAworst <- RA(alpha, qF=rep(list(qF), d), N=ARAworst$N.used,
              abstol=mean(tail(abs(diff(ARAworst$m.row.sums$low)), n=1),
                          tail(abs(diff(ARAworst$m.row.sums$up)), n=1)))

## Compare
stopifnot(all.equal(c(VaR.W[1], ARAbest$bounds, RAbest$bounds),
                    rep(VaR.W.Par[1],5), tolerance=0.004, check.names=FALSE))
stopifnot(all.equal(c(VaR.W[2], VaR.dual[2], ARAworst$bounds, RAworst$bounds),
                    rep(VaR.W.Par[2],6), tolerance=0.003, check.names=FALSE))


## Using (some of) the additional results computed by (A)RA() ##################

xlim <- c(1, max(sapply(RAworst$m.row.sums, length)))
```

```
ylim <- range(RAworst$m.row.sums)
plot(RAworst$m.row.sums[[2]], type="l", xlim=xlim, ylim=ylim,
     xlab="Number or rearranged columns",
     ylab=paste0("Minimal row sum per rearranged column"),
     main=substitute("Worst VaR minimal row sums ("*alpha==a.*","~d==d.*" and Par("*
                     th.*"))", list(a.=alpha, d.=d, th.=th)))
lines(1:length(RAworst$m.row.sums[[1]]), RAworst$m.row.sums[[1]], col="royalblue3")
legend("bottomright", bty="n", lty=rep(1,2),
       col=c("black", "royalblue3"), legend=c("upper bound", "lower bound"))
## => One should use ARA() instead of RA()


## "Reproducing" examples from Embrechts et al. (2013) ##########################

## "Reproducing" Table 1 (but seed and eps are unknown)
## Left-hand side of Table 1
N <- 50
qPar <- rep(list(qF), 3)
p <- alpha + (1-alpha)*(0:(N-1))/N # for 'worst' (= largest) VaR
X <- sapply(qPar, function(qF) qF(p))
cbind(X, rowSums(X))
## Right-hand side of Table 1
set.seed(271)
res <- RA(alpha, qF=qPar, N=N)
row.sum <- rowSums(res$X.rearranged$low)
cbind(res$X.rearranged$low, row.sum)[order(row.sum),]

## "Reproducing" Table 3 for alpha=0.99 (but seed is unknown)
N <- 2e4 # we use a smaller N here to save run time
eps <- 0.1 # absolute tolerance
xi <- c(1.19, 1.17, 1.01, 1.39, 1.23, 1.22, 0.85, 0.98)
beta <- c(774, 254, 233, 412, 107, 243, 314, 124)
qF.lst <- lapply(1:8, function(j){ function(p) qGPD(p, xi=xi[j], beta=beta[j])})
set.seed(271)
res.best <- RA(0.99, qF=qF.lst, N=N, abstol=eps, method="best")
print(format(res.best$bounds, scientific=TRUE), quote=FALSE) # close to first value of 1st row
res.worst <- RA(0.99, qF=qF.lst, N=N, abstol=eps)
print(format(res.worst$bounds, scientific=TRUE), quote=FALSE) # close to last value of 1st row
```

Index