Package 'QRM'

July 2, 2014

Version 0.4-10

Date 2014-05-12

Title Provides R-language Code to Examine Quantitative Risk Management Concepts

Author Bernhard Pfaff [aut, cre],Marius Hofert [ctb],Alexander McNeil [aut] (S-Plus original (QRMlib)),Scott Ulmann [trl] (First R port as package QRMlib)

Maintainer Bernhard Pfaff <bernhard@pfaffikus.de>

Description This package is designed to accompany the book Quantitative Risk Management: Concepts, Techniques and Tools by Alexander J. McNeil, Rudiger Frey, and Paul Embrechts.

Depends R (>= 3.0.1), gsl, Matrix, mvtnorm, numDeriv, timeSeries

Imports Rcpp (>= 0.11.1), mgcv

LinkingTo Rcpp

LazyData Yes

License GPL (>= 2)

Encoding UTF-8

Repository CRAN

Repository/R-Forge/Project qrm

Repository/R-Forge/Revision 82

Repository/R-Forge/DateTimeStamp 2014-05-13 15:26:28

Date/Publication 2014-05-16 20:10:49

NeedsCompilation yes

R topics documented:

`QRM-package` *Quantitative Risk Modelling*

Description

This package is designed to accompany the book Quantitative Risk Management: Concepts, Techniques and Tools by Alexander J. McNeil, Rudiger Frey and Paul Embrechts.

Overview

This package provides functions for quantitative risk management as introduced in the book "Quantitative Risk Management: Concepts, Techniques and Tools" (henceforth: QRM). The S-Plus package "QRMlib" has been made available the first author of the book and can be obtained by following the instructions on `http://www.ma.hw.ac.uk/~mcneil/book/QRMlib.html`. A R port of this package has been made available on CRAN by Scott Ulmann. However, the package failed the checks and hence has been moved to the CRAN archive (**QRMlib**, version 1.4.5.1 as of 04/25/2011). This package is based on **QRMlib**, but (i), not all functions have been ported from **QRMlib** to **QRM**, (ii) the arguments of some functions have been modified, and (iii) the manual pages have been re-ordered by topic.
A list of the not ported functions is provided in `QRM-defunct` with pointers to their replacements. This was achieved by the inclusion of dependencies to the packages **gsl**, **numDeriv** and **timeSeries** and/or resorting to functions contained in the base installation of R. Second, in particular with respect to passing down arguments to the routines used in optimizations and/or argument matching, modifications to the functions' closures were necessary. In addition, the names of arguments in similar functions have been unified. Third, to provide the user a faster access to the manual pages of certain risk concepts, the functions' documentation are now ordered by concept rather than by the name of the functions.
Without modifying the existing functions of **QRMlib** too much, neither S3- nor S4-classes and methods have been included completely by now in **QRM**, but the characteristic of the former package as a collection of functions pertinent to quantitative risk modelling have been kept intact. However, this might change in future releases of **QRM**. By now, the current package can be used almost alike **QRMlib**, but with the stated modifications.

References

McNeil, A., Frey, R. and Embrechts, P., *Quantitative Risk Management: Concepts, Techniques and Tools*, 2005, Princeton: Princeton University Press.

`BiDensPlot` *Bivariate Density Plot*

Description

Generates eiether a perspective or a contour plot of a bivariate density.

Usage

```
BiDensPlot(func, xpts = c(-2, 2), ypts = c(-2, 2), npts = 50,
           type = c("persp", "contour"), ...)
```

Arguments

func	function, the name of a bivariate density function.
xpts	vector, interval of x.
ypts	vector, interval of y.
npts	integer, number of subdivision points between x and y over the specified range xpts to ypts.
type	character, the plot type, either a perspective or a contour plot.
...	ellipsis, arguments are passed to the call of func.

Value

Returns invisibly a list of (x, y, z) triplet.

Examples

```
BiDensPlot(func = dmnorm, mu = c(0, 0), Sigma = equicorr(2, -0.7))
```

---

cac40 *CAC 40 Stock Market Index (France)*

Description

This timeSeries data set provides the daily closing values of the French CAC 40 stock index for the period 1994 to March 2004. In addition, the data set is also made available as a data.frame.

Usage

```
data(cac40)
data(cac40.df)
```

Examples

```
data(cac40)
head(cac40)
```

CopulaAC *Archimedean Copulae*

Description

Functions for ealuating densities of Archimedean copulae, generating random variates and fitting data to AC

Usage

```
dcopula.AC(u, theta, name = c("clayton", "gumbel"), log = TRUE)
dcopula.clayton(u, theta, log = FALSE)
dcopula.gumbel(u, theta, log = FALSE)
rAC(name = c("clayton", "gumbel", "frank", "BB9", "GIG"), n, d, theta)
rACp(name = c("clayton", "gumbel", "frank", "BB9", "GIG"), n, d, theta, A)
rcopula.gumbel(n, theta, d)
rcopula.clayton(n, theta, d)
rcopula.frank(n, theta, d)
rstable(n, alpha, beta = 1)
rFrankMix(n, theta)
rBB9Mix(n, theta)
rcopula.Gumbel2Gp(n = 1000, gpsizes = c(2, 2), theta = c(2, 3, 5))
rcopula.GumbelNested(n, theta)
fit.AC(Udata, name = c("clayton", "gumbel"), initial = 2, ...)
```

Arguments

A	matrix, dimension $d \times p$ containing asymmetry parameters. Rowsums must be equal to one.
alpha	numeric, parameter $0 < \alpha \leq 2$, but $\alpha \neq 1$.
beta	numeric, parameter $-1 \leq \beta \leq 1$.
d	integer, dimension of copula.
gpsizes	vector, length of two, containing the group sizes.
initial	numeric, initial value used by fit.AC() in the call to nlminb().
log	logical, whether log density values should be returned
n	integer, count of random variates.
name	character, name of copula.
theta	vector, copula parameter(s).
u	matrix, dimension $n \times d$, where d is the dimension of the copula and n is the number of vector values at which to evaluate density.
Udata	matrix, pseudo-uniform observations.
...	ellipsis, arguments are passed down to nlminb().

Details

The function dcopula.AC() is a generic function, designed such that additional copulae, or expressions for densities of higher-dimensional copulae may be added. Clayton copula works in any dimension at present but Gumbel is only implemented for $d = 2$. To extend, one must calculate the d-th derivative of the generator inverse and take the logarithm of absolute value; this is the term called loggfunc. In addition, for other copulae, one needs the generator ϕ and the log of the negative value of its first derivative lnegphidash.

The random variates from rAC() with arbitrary dimension are generated by using the mixture construction of Marshall and Olkin. It may be used in place of the other functions rcopula.clayton(), rcopula.gumbel(), and rcopula.frank(). In addition, it allows simulation of BB9 and GIG copulas which don't have individual simulation routines.

For the Clayton and Gumbel copulae, see page 192 and 222–224 in QRM. The random variates for the BB9 and Frank copula are obtained from a mixing distribution using a Laplace transform method (see page 224 of QRM). The function rcopula.Gumbel2Gp() generates sample from a Gumbel copula with two-group structure constructed using three Gumbel generators (see pages 222-224 and 227 of QRM). The function rcopula.gumbelNested() generates sample from a d-dimensional Gumbel copula with nested structure constructed using $(d - 1)$ Gumbel generators.

For the random variates of the Stable distribution, a default value $\beta = 1$ is used; combined with a value for $\alpha < 1$ yields a positive stable distribution, which is required for Gumbel copula generation; the case $\alpha = 1$ has not been implemented.

Value

vector or matrix in case of the density and random-generator related functions and a list object for the fitting function.

See Also

nlminb

Examples

```
## Gumbel
r1 <- rAC("gumbel", n = 50, d = 7, theta = 3)
head(r1)
## Weighted Gumbel
alpha <- c(0.95,0.7)
wtmatrix <- cbind(alpha, 1 - alpha)
r2 <- rACp(name = "gumbel", n = 1000, d = 2, theta = c(4, 1),
           A = wtmatrix)
head(r2)
## Gumbel with two-group structure
r3 <- rcopula.Gumbel2Gp(n = 3000, gpsizes = c(3, 4),
                        theta = c(2, 3, 5))
pairs(r3)
## Nested Gumbel
r4 <- rcopula.GumbelNested(n=3000,theta=1:6)
pairs(r4)
## Frank
r5 <- rcopula.frank(1000, 2, 4)
```

```
pairs(r5)
## Fitting of Gumbel and Clayton
data(smi)
data(ftse100)
s1 <- window(ftse100, "1990-11-09", "2004-03-25")
s1a <- alignDailySeries(s1)
s2a <- alignDailySeries(smi)
idx <- merge(s1a, s2a)
r <-returns(idx)
rp <- series(window(r, "1994-01-01", "2003-12-31"))
rp <- rp[(rp[, 1] != 0) & (rp[, 2] !=0), ]
Udata <- apply(rp, 2, edf, adjust = 1)
mod.gumbel <- fit.AC(Udata, "gumbel")
mod.clayton <- fit.AC(Udata, "clayton")
mod.clayton
```

`CopulaGauss` *Gauss Copula*

Description

Functions for evaluating the Gauss copula, generating random variates and fitting.

Usage

```
dcopula.gauss(Udata, Sigma, log = FALSE)
rcopula.gauss(n, Sigma)
fit.gausscopula(Udata, ...)
```

Arguments

`log`	`logical`, whether log density values should be returned.
`n`	`integer`, count of random variates
`Sigma`	`matrix`, correlation matrix.
`Udata`	`matrix`, pseudo-uniform data where rows are vector observations with all values in unit interval.
`...`	ellipsis argument, passed down to `nlminb()` used in optimization.

Value

For `dcopula.gauss()` a vector of density values of length n. For `rcopula.gauss()` a $n \times d$ matrix of random variates and for `fit.gausscopula()` a list with the optimization results.

See Also

`nlminb`

Examples

```
ll <- c(0.01,0.99)
BiDensPlot(func = dcopula.gauss, xpts = ll, ypts = ll,
           Sigma = equicorr(2, 0.5))
data <- rcopula.gauss(2000, Sigma = equicorr(d = 6, rho = 0.7))
pairs(data)
## Fitting Gauss Copula
data(smi)
data(ftse100)
s1 <- window(ftse100, "1990-11-09", "2004-03-25")
s1a <- alignDailySeries(s1)
s2a <- alignDailySeries(smi)
idx <- merge(s1a, s2a)
r <-returns(idx)
rp <- series(window(r, "1994-01-01", "2003-12-31"))
rp <- rp[(rp[, 1] != 0) & (rp[, 2] !=0), ]
Udata <- apply(rp, 2, edf, adjust = 1)
copgauss <- fit.gausscopula(Udata)
```

`CopulaStudent` *Student's t Copula*

Description

Functions for copula density, generating random variates and fitting

Usage

```
dcopula.t(Udata, df, Sigma, log = FALSE)
rcopula.t(n, df, Sigma)
fit.tcopula(Udata, method = c("all", "Kendall", "Spearman"),
            startdf = 5, ...)
```

Arguments

`df`	`numeric`, degrees of freedom.
`log`	`logical`, whether log density values should be returned.
`method`	`character`, method for fitting.
`n`	`integer`, count of random variates
`Sigma`	`matrix`, correlation matrix
`startdf`	`numeric`, initial DF value.
`Udata`	`matrix`, dimension $n \times d$, where d is the dimension of the copula and n is the number of pseudo-uniform values.
`...`	ellipsis, arguments are passed down to `nlminb()`.

Details

If in the call to `fit.tcopula()`, `method = "all"`, then all parameters are estimated, *i.e.*, the degrees of freedom and the dispersion parameters (initial values from Spearman correlations). In case of either `method = "Kendall"` or `method = "Spearman"`, the corresponding rank correlations are used and the optimization is only carried out with respect to the degrees of freedom parameter. The initial value for the DF is given by `startdf`. See pages 197 and 229–236 of QRM.

Value

A vector of density values of length n for `dcopula.t()`. A matrix of random variates for `rcopula.t()`. A list object containing parameter estimates and details of fit for function `fit.tcopula()`.

See Also

`nlminb`

Examples

```
ll <- c(0.01,0.99)
#create perspective plot for bivariate density:
BiDensPlot(func = dcopula.t, xpts = ll, ypts = ll, df = 4,
           Sigma = equicorr(2, 0.5))
S <- equicorr(d = 6, rho = 0.7)
data <- rcopula.t(2000, df = 4, Sigma = S)
pairs(data)
## Fitting Student's Copula
data(smi)
data(ftse100)
s1 <- window(ftse100, "1990-11-09", "2004-03-25")
s1a <- alignDailySeries(s1)
s2a <- alignDailySeries(smi)
idx <- merge(s1a, s2a)
r <-returns(idx)
rp <- series(window(r, "1994-01-01", "2003-12-31"))
rp <- rp[(rp[, 1] != 0) & (rp[, 2] !=0), ]
Udata <- apply(rp, 2, edf, adjust = 1)
copt1 <- fit.tcopula(Udata)
copt2 <- fit.tcopula(Udata, method = "Kendall")
```

Credit *Credit Risk Modelling*

Description

Functions for modelling credit risk:

- Bernoulli mixture model with prescribed default and joint default probabilities
- Bernoulli mixture model with Clayton copula dependencies of default.

- Probitnormal Mixture of Bernoullis
- Beta-Binomial Distribution
- Logitnormal-Binomial Distribution
- Probitnormal-Binomial Distribution

Usage

```
cal.beta(pi1, pi2)
cal.claytonmix(pi1, pi2)
cal.probitnorm(pi1, pi2)
dclaytonmix(x, pi, theta)
pclaytonmix(q, pi, theta)
rclaytonmix(n, pi, theta)
rtcopulamix(n, pi, rho.asset, df)
dprobitnorm(x, mu, sigma)
pprobitnorm(q, mu, sigma)
rprobitnorm(n, mu, sigma)
rbinomial.mixture(n = 1000, m = 100,
                  model = c("probitnorm", "logitnorm", "beta"), ...)
rlogitnorm(n, mu, sigma)
fit.binomial(M, m)
fit.binomialBeta(M, m, startvals = c(2, 2), ses = FALSE, ...)
fit.binomialLogitnorm(M, m, startvals = c(-1, 0.5), ...)
fit.binomialProbitnorm(M, m, startvals = c(-1, 0.5), ...)
momest(data, trials, limit = 10)
```

Arguments

data	vector, numbers of defaults in each time period.
df	numeric, degree of freedom.
limit	intgeger, maximum order of joint default probability to estimate.
M	vector, count of successes.
m	vector, count of trials.
model	character, name of mixing distribution.
mu	numeric, location parameter.
n	integer, count of random variates.
pi	numeric, default probability.
pi1	numeric, default probability.
pi2	numeric, joint default probability.
q	numeric, values at which CDF should be evaluated.
sigma	numeric, scale parameter.
ses	logical, whether standard errors should be returned.
startvals	numeric, starting values.

theta	numeric, parameter of distribution.
trials	vector, group sizes in each time period.
x	numeric, values at which density should be evaluated.
rho.asset	numeric, asset correlation parameter.
...	ellipsis, arguments are passed down to either mixing distribution or nlminb().

Details

cal.beta(): calibrates a beta mixture distribution on unit interval to give an exchangeable Bernoulli mixture model with prescribed default and joint default probabilities (see pages 354-355 in QRM).
cal.claytonmix(): calibrates a mixture distribution on unit interval to give an exchangeable Bernoulli mixture model with prescribed default and joint default probabilities. The mixture distribution is the one implied by a Clayton copula model of default (see page 362 in QRM).
cal.probitnorm(): calibrates a probitnormal mixture distribution on unit interval to give an exchangeable Bernoulli mixture model with prescribed default and joint default probabilities (see page 354 in QRM).
dclaytonmix(), pclaytonmix(), rclaytonmix(): density, cumulative probability, and random generation for a mixture distribution on the unit interval which gives an exchangeable Bernoulli mixture model equivalent to a Clayton copula model (see page 362 in QRM).
fit.binomial(): fits binomial distribution by maximum likelihood.
dprobitnorm(), pprobitnorm(), rprobitnorm(): density, cumulative probability and random number generation for distribution of random variable Q on unit interval such that the probit transform of Q has a normal distribution with parameters μ and σ (see pages 353-354 in QRM).
fit.binomialBeta(): fit a beta-binomial distribution by maximum likelihood.
fit.binomialLogitnorm(): fits a mixed binomial distribution where success probability has a logitnormal distribution. Lower and upper bounds for the input parameters M and m can be specified by means of the arguments lower and upper, which are passed to nlminb(). If convergence occurs at an endpoint of either limit, one need to reset lower and upper parameter estimators and run the function again.
fit.binomialProbitnorm(): Fits a mixed binomial distribution where success probability has a probitnormal distribution. Lower and upper bounds for the input parameters M and m can be specified by means of the arguments lower and upper, which are passed to nlminb(). If convergence occurs at an endpoint of either limit, one need to reset lower and upper parameter estimators and run the function again.
momest(): calculates moment estimator of default probabilities and joint default probabilities for a homogeneous group. First returned value is default probability estimate; second value is estimate of joint default probability for two firms; and so on (see pages 375-376 in QRM).
rbinomial.mixture(): random variates from mixed binomial distribution (see pages 354-355 and pages 375-377 of QRM).
rlogitnorm(): Random number generation for distribution of random variable Q on unit interval such that the probit transform of Q has a normal distribution with parameters μ and σ (see pages 353-354 in QRM).
rtcopulamix(): random generation for mixing distribution on unit interval yielding Student's t copula model (see page 361 in QRM, exchangeable case of this model is considered).

See Also

link[stats]{nlminb}

Examples

```
## calibrating models
pi.B <- 0.2
pi2.B <- 0.05
probitnorm.pars <- cal.probitnorm(pi.B, pi2.B)
probitnorm.pars
beta.pars <- cal.beta(pi.B, pi2.B)
beta.pars
claytonmix.pars <- cal.claytonmix(pi.B, pi2.B)
claytonmix.pars
q <- (1:1000) / 1001
q <- q[q < 0.25]
p.probitnorm <- pprobitnorm(q, probitnorm.pars[1],
                            probitnorm.pars[2])
p.beta <- pbeta(q, beta.pars[1], beta.pars[2])
p.claytonmix <- pclaytonmix(q, claytonmix.pars[1],
                            claytonmix.pars[2])
scale <- range((1 - p.probitnorm), (1 - p.beta), (1 - p.claytonmix))
plot(q, (1 - p.probitnorm), type = "l", log = "y", xlab = "q",
          ylab = "P(Q > q)",ylim=scale)
lines(q, (1 - p.beta), col = 2)
lines(q, (1 - p.claytonmix), col = 3)
legend("topright", c("Probit-normal", "Beta", "Clayton-Mixture"),
         lty=rep(1,3),col = (1:3))
## Clayton Mix
pi.B <- 0.0489603
pi2.B <- 0.003126529
claytonmix.pars <- cal.claytonmix(pi.B, pi2.B)
claytonmix.pars
q <- (1:1000) / 1001
q <- q[q < 0.25]
d.claytonmix <- dclaytonmix(q, claytonmix.pars[1], claytonmix.pars[2])
head(d.claytonmix)
## SP Data
data(spdata.raw)
attach(spdata.raw)
BdefaultRate <- Bdefaults / Bobligors
## Binomial Model
mod1a <- fit.binomial(Bdefaults, Bobligors)
## Binomial Logitnorm Model
mod1b <- fit.binomialLogitnorm(Bdefaults, Bobligors)
## Binomial Probitnorm Model
mod1c <- fit.binomialProbitnorm(Bdefaults, Bobligors)
## Binomial Beta Model
mod1d <- fit.binomialBeta(Bdefaults, Bobligors);
## Moment estimates for default probabilities
momest(Bdefaults, Bobligors)
pi.B <- momest(Bdefaults, Bobligors)[1]
pi2.B <- momest(Bdefaults, Bobligors)[2]
## Probitnorm
probitnorm.pars <- cal.probitnorm(pi.B, pi2.B)
q <- (1:1000)/1001
```

```
q <- q[ q < 0.25]
d.probitnorm <- dprobitnorm(q, probitnorm.pars[1], probitnorm.pars[2])
p <- c(0.90,0.95,0.975,0.99,0.995,0.999,0.9999,0.99999,0.999999)
sigma <- 0.2 * 10000 / sqrt(250)
VaR.t4 <- qst(p, df = 4, sd = sigma, scale = TRUE)
VaR.t4
detach(spdata.raw)
## Binomial Mixture Models
pi <- 0.04896
pi2 <- 0.00321
beta.pars <- cal.beta(pi, pi2)
probitnorm.pars <- cal.probitnorm(pi, pi2)
n <- 1000
m <- rep(500, n)
mod2a <- rbinomial.mixture(n, m, "beta", shape1 = beta.pars[1],
                           shape2 = beta.pars[2])
mod2b <- rbinomial.mixture(n, m, "probitnorm",
                           mu = probitnorm.pars[1],
                           sigma = probitnorm.pars[2])
```

---

danish *Danish Fire Losses*

Description

The danish timeSeries dataset provides the daily closing value for the Danish fire losses measured from January 1980 through December 1990. In addition, the data set is also made available as a data.frame.

Usage

```
data(danish)
data(danish.df)
```

Examples

```
data(danish)
head(danish)
```

DJ *Dow Jones 30 Stock Prices*

Description

The DJ timeSeries data set provides the closing values of the Dow Jones 30 Stocks from 1991-2000. In addition, the data set is also made available as a data.frame.

Usage

```
data(DJ)
data(DJ.df)
```

Examples

```
data(DJ)
head(DJ)
```

`dji` *Dow Jones Index*

Description

The `dji` timeSeries dataset provides the daily closing value for the Dow Jones index from January 1980 to March 2004. In addition, the data set is also made available as a `data.frame`.

Usage

```
data(dji)
data(dji.df)
```

Examples

```
data(dji)
head(dji)
```

`edf` *Empirical Distribution Function*

Description

This function calculates the empirical distribution function at each element of a vector of observations.

Usage

```
edf(v, adjust = FALSE)
```

Arguments

`v`	`vector`, observations of length n.
`adjust`	`logical`, adjustment of denominator to be (n + 1).

Value

vector

Examples

```
data(smi)
data(ftse100)
s1 <- window(ftse100, "1990-11-09", "2004-03-25")
s1a <- alignDailySeries(s1)
s2a <- alignDailySeries(smi)
idx <- merge(s1a, s2a)
r <-returns(idx)
rp <- series(window(r, "1994-01-01", "2003-12-31"))
rp <- rp[(rp[, 1] != 0) & (rp[, 2] !=0), ]
Udata <- apply(rp, 2, edf, adjust = 1)
plot(Udata)
```

---

eigenmeth *Make Matrix Positive Definite*

Description

The function adjusts a negative definite symmetric matrix to make it positive definite.

Usage

```
eigenmeth(mat, delta = 0.001)
```

Arguments

mat	matrix, a symmetric matrix
delta	numeric, new size of smallest eigenvalues

Details

See page 231 of QRM.

Value

a positive-definite matrix

equicorr *Equal Correlation Matrix*

Description

Construction of an equal correlation matrix

Usage

```
equicorr(d, rho)
```

Arguments

d	integer, dimension of matrix
rho	numeric, value of correlation

Value

matrix

Examples

```
equicorr(7, 0.5)
ll <- c(0.01, 0.99)
BiDensPlot(func = dcopula.gauss, xpts = ll, ypts = ll,
           Sigma = equicorr(2,0.5))
BiDensPlot(func = dcopula.t, xpts = ll, ypts = ll , df = 4,
           Sigma = equicorr(2, 0.5))
```

ES *Expected Shortfall*

Description

Functions for computing the expected shortfall derived from the Normal or Student's t distribution (see page 45 of QRM).

Usage

```
ESnorm(p, mu = 0, sd = 1)
ESst(p, mu = 0, sd = 1, df, scale = FALSE)
```

Arguments

p	numeric, probability
mu	numeric, location parameter
sd	numeric, scale parameter
df	numeric, degrees of freedom
scale	logical, scaling Student's t distribution to have variance one

Value

numeric

Examples

```
p <- c(0.95, 0.99)
s <- 0.2 * 10000 / sqrt(250)
ESnorm(p)
ESst(p, sd = s, df = 4, scale = TRUE)
ESst(p, df = 4)
```

---

ftse100 *FTSE 100 Stock Market Index*

Description

The ftse100 timeSeries dataset provides the daily closing value for the FTSE index from January 1980 to March 2004. In addition, the data set is also made available as a data.frame.

Usage

```
data(ftse100)
data(ftse100.df)
```

Examples

```
data(ftse100)
head(ftse100)
```

FXGBP.RAW *Sterling Exchange Rates*

Description

The FXGBP timeSeries dataset provides daily exchange rates for major currencies (US Dollar, Japanese Yen, Euro, Swiss franc) against the British Pound for the period January 1987 through March 2004. In addition, the data set is also made available as a data.frame.

Usage

```
data(FXGBP)
data(FXGBP.df)
```

Examples

```
data(FXGBP)
tail(FXGBP)
```

game *Smooth Parameter Estimation and Bootstrapping of Generalized Pareto Distributions with Penalized Maximum Likelihood Estimation*

Description

gamGPDfit() fits the parameters of a generalized Pareto distribution (GPD) depending on covariates in a non- or semiparametric way.

gamGPDboot() fits and bootstraps the parameters of a GPD distribution depending on covariates in a non- or semiparametric way. Applies the post-blackend bootstrap of Chavez-Demoulin and Davison (2005).

Usage

```
gamGPDfit(x, threshold, nextremes = NULL, datvar, xiFrhs, nuFrhs,
          init = fit.GPD(x[,datvar], threshold = threshold,
                         type = "pwm", verbose = FALSE)$par.ests,
          niter = 32, include.updates = FALSE, eps.xi = 1e-05, eps.nu = 1e-05,
          progress = TRUE, adjust = TRUE, verbose = FALSE, ...)
gamGPDboot(x, B, threshold, nextremes = NULL, datvar, xiFrhs, nuFrhs,
           init = fit.GPD(x[,datvar], threshold = threshold,
                          type = "pwm", verbose = FALSE)$par.ests,
           niter = 32, include.updates = FALSE, eps.xi = 1e-5, eps.nu = 1e-5,
          boot.progress = TRUE, progress = FALSE, adjust = TRUE, verbose = FALSE,
           debug = FALSE, ...)
```

Arguments

x	data.frame containing the losses (in some component; can be specified with the argument datvar; the other components contain the covariates).
B	number of bootstrap replications.
threshold	threshold of the peaks-over-threshold (POT) method.
nextremes	number of excesses. This can be used to determine
datvar	name of the data column in x which contains the the data to be modeled.
xiFrhs	right-hand side of the formula for ξ in the gam() call for fitting ξ.
nuFrhs	right-hand side of the formula for ν in the gam() call for fitting ν.
init	bivariate vector containing initial values for (ξ, β).
niter	maximal number of iterations in the backfitting algorithm.
include.updates	logical indicating whether updates for xi and nu are returned as well (note: this might lead to objects of large size).
eps.xi	epsilon for stop criterion for ξ.
eps.nu	epsilon for stop criterion for ν.
boot.progress	logical indicating whether progress information about gamGPDboot() is displayed.
progress	logical indicating whether progress information about gamGPDfit() is displayed. For gamGPDboot(), progress is only passed to gamGPDfit() in the case that boot.progress==TRUE.
adjust	logical indicating whether non-real values of the derivatives are adjusted.
verbose	logical indicating whether additional information (in case of undesired behavior) is printed. For gamGPDboot(), progress is only passed to gamGPDfit() if boot.progress==TRUE.
debug	logical indicating whether initial fit (before the bootstrap is initiated) is saved.
...	additional arguments passed to gam() (which is called internally; see the source code of gamGPDfitUp()).

Details

gamGPDfit() fits the parameters ξ and β of the generalized Pareto distribution $\mathrm{GPD}(\xi, \beta)$ depending on covariates in a non- or semiparametric way. The distribution function is given by

$$G_{\xi,\beta}(x) = 1 - (1 + \xi x/\beta)^{-1/\xi}, \quad x \geq 0,$$

for $\xi > 0$ (which is what we assume) and $\beta > 0$. Note that β is also denoted by σ in this package. Estimation of ξ and β by gamGPDfit() is done via penalized maximum likelihood estimation, where the estimators are computed with a backfitting algorithm. In order to guarantee convergence of this algorithm, a reparameterization of β in terms of the parameter ν is done via

$$\beta = \exp(\nu)/(1 + \xi).$$

The parameters ξ and ν (and thus β) are allowed to depend on covariates (including time) in a non- or semiparametric way, for example:

$$\xi = \xi(\boldsymbol{x}, t) = \boldsymbol{x}^\top \boldsymbol{\alpha}_\xi + h_\xi(t),$$
$$\nu = \nu(\boldsymbol{x}, t) = \boldsymbol{x}^\top \boldsymbol{\alpha}_\nu + h_\nu(t),$$

where $\boldsymbol{x}$ denotes the vector of covariates, $\boldsymbol{\alpha}_\xi$, $\boldsymbol{\alpha}_\nu$ are parameter vectors and h_ξ, h_ν are regression splines. For more details, see the references and the source code.

`gamGPDboot()` first fits the GPD parameters via `gamGPDfit()`. It then conducts the post-blackend bootstrap of Chavez-Demoulin and Davison (2005). To this end, it computes the residuals, resamples them (`B` times), reconstructs the corresponding excesses, and refits the GPD parameters via `gamGPDfit()` again.

Note that if `gam()` fails in `gamGPDfit()` or the fitting or one of the bootstrap replications in `gamGPDboot()`, then the output object contains (an) empty (sub)list(s). These failures typically happen for too small sample sizes.

Value

`gamGPDfit()` returns either an empty list (`list()`; in case at least one of the two `gam()` calls in the internal function `gamGPDfitUp()` fails) or a list with the components

`xi`: estimated parameters ξ;

`beta`: estimated parameters β;

`nu`: estimated parameters ν;

`se.xi`: standard error for ξ ((possibly adjusted) second-order derivative of the reparameterized log-likelihood with respect to ξ) multiplied by -1;

`se.nu`: standard error for ν ((possibly adjusted) second-order derivative of the reparameterized log-likelihood with respect to ν) multiplied by -1;

`xi.covar`: (unique) covariates for ξ;

`nu.covar`: (unique) covariates for ν;

`covar`: available covariate combinations used for fitting β (ξ, ν);

`y`: vector of excesses (exceedances minus threshold);

`res`: residuals;

`MRD`: mean relative distances between for all iterations, calculated between old parameters (ξ, ν) (from the last iteration) and new parameters (currently estimated ones);

`logL`: log-likelihood at the estimated parameters;

`xiObj`: R object of type `gamObject` for estimated ξ (returned by `mgcv::gam()`);

`nuObj`: R object of type `gamObject` for estimated ν (returned by `mgcv::gam()`);

`xiUpdates`: if `include.updates` is `TRUE`, updates for ξ for each iteration. This is a list of R objects of type `gamObject` which contains `xiObj` as last element;

`nuUpdates`: if `include.updates` is `TRUE`, updates for ν for each iteration. This is a list of R objects of type `gamObject` which contains `nuObj` as last element;

`gamGPDboot()` returns a list of length `B+1` where the first component contains the results of the initial fit via `gamGPDfit()` and the other `B` components contain the results for each replication of the post-blackend bootstrap. Components for which `gam()` fails (e.g., due to too few data) are given as empty lists (`list()`).

Author(s)

Marius Hofert, Valerie Chavez-Demoulin.

References

Chavez-Demoulin, V., and Davison, A. C. (2005), Generalized additive models for sample extremes, *Applied Statistics* **54**(1), 207–222.

Chavez-Demoulin, V., Embrechts, P., and Hofert, M., An extreme value approach for modeling Operational Risk losses depending on covariates.

Examples

```
## generate an example data set
years <- 2003:2012 # years
nyears <- length(years)
n <- 250 # sample size for each (different) xi
u <- 200 # threshold
rGPD <- function(n, xi, beta) ((1-runif(n))^(-xi)-1)*beta/xi # sampling GPD

set.seed(17) # setting seed
xi.true.A <- seq(0.4, 0.8, length=nyears) # true xi for group "A"
## generate losses for group "A"
lossA <- unlist(lapply(1:nyears,
                       function(y) u + rGPD(n, xi=xi.true.A[y], beta=1)))
xi.true.B <- xi.true.A^2 # true xi for group "B"
## generate losses for group "B"
lossB <- unlist(lapply(1:nyears,
                       function(y) u + rGPD(n, xi=xi.true.B[y], beta=1)))
## build data frame
time <- rep(rep(years, each=n), 2) # "2" stands for the two groups
covar <- rep(c("A","B"), each=n*nyears)
value <- c(lossA, lossB)
x <- data.frame(covar=covar, time=time, value=value)

## fit
eps <- 1e-3 # to decrease the run time for this example
require(mgcv) # due to s()
fit <- gamGPDfit(x, threshold=u, datvar="value", xiFrhs=~covar+s(time)-1,
                 nuFrhs=~covar+s(time)-1, eps.xi=eps, eps.nu=eps)
## note: choosing s(..., bs="cr") will fit cubic splines

## grab the fitted values per group and year
xi.fit <- fitted(fit$xiObj)
xi.fit. <- xi.fit[1+(0:(2*nyears-1))*n] # pick fit for each group and year
xi.fit.A <- xi.fit.[1:nyears] # fit for "A" and each year
xi.fit.B <- xi.fit.[(nyears+1):(2*nyears)] # fit for "B" and each year

## plot the fitted values of xi and the true ones we simulated from
par(mfrow=c(1,2))
plot(years, xi.true.A, type="l", ylim=range(xi.true.A, xi.fit.A),
     main="Group A", xlab="Year", ylab=expression(xi))
```

```
points(years, xi.fit.A, type="l", col="red")
legend("topleft", inset=0.04, lty=1, col=c("black", "red"),
       legend=c("true", "fitted"), bty="n")
plot(years, xi.true.B, type="l", ylim=range(xi.true.B, xi.fit.B),
     main="Group B", xlab="Year", ylab=expression(xi))
points(years, xi.fit.B, type="l", col="blue")
legend("topleft", inset=0.04, lty=1, col=c("black", "blue"),
       legend=c("true", "fitted"), bty="n")
```

---

game-aux *Auxiliary Functions for Extracting/Computing Results Related to gamGPDfit()/gamGPDboot()*

Description

`get.lambda.fit()` extracts a convenient list containing unique covariate combinations and corresponding fitted values from an object returned by `gam()`.

`lambda.predict()` computes a convenient list containing unique covariate combinations and corresponding predicted values and pointwise asymptotic confidence intervals (obtained from the estimated standard errors obtained by `predict(..., se.fit=TRUE)`).

`get.GPD.fit()` extracts a convenient list containing (for each of the GPD parameters) unique covariate combinations, the fitted GPD parameter (vector), bootstrapped pointwise two-sided 1-α confidence intervals, and a matrix of bootstrapped parameter values.

`GPD.predict()` computes a convenient list containing (for each of the GPD parameters) unique covariate combinations and corresponding predicted values.

`risk.measure()` computes the selected risk measure at a matrix of values for λ, ξ, β.

Usage

```
get.lambda.fit(x)
lambda.predict(x, newdata=NULL, alpha=0.05)
get.GPD.fit(x, alpha=0.05)
GPD.predict(x, xi.newdata=NULL, beta.newdata=NULL)

risk.measure(x, alpha, u, method = c("VaR", "ES"))
```

Arguments

`x`	For `get.lambda.fit()`, `lambda.predict()` an object as returned by `gam()`; for `get.GPD.fit()`, `GPD.predict()` an object as returned by `gamGPDboot()`; for `risk.measure()` a matrix with three columns containing λ, ξ, β (in this order).
`newdata`	object as required by `predict()`. Typically a named `data.frame` of type `expand.grid(covar1=, covar` with at least the covariates used for fitting with `gam()`; if more are provided, `predict()` returns values which are equal uniformly over all of these additional covariates. Each covariate which appears when fitting with `gam()` can have more

values than were actually used in gam(). In this case predict() "interpolates" correctly with the fitted model.

xi.newdata, beta.newdata
: as newdata, just for the GPD parameters ξ and β.

alpha
: for lambda.predict(), get.GPD.fit() the significance level (typcially 0.05); for risk.measure() the confidence level (typically close to 1).

u
: threshold.

method
: character string indicating the kind of risk measure (Value-at-Risk (VaR) or expected shortfall (ES)).

Details

Note that if gam() fails in gamGPDfit() or the fitting or one of the bootstrap replications in gamGPDboot(), then x contains (an) empty (sub)list(s). These empty lists will be removed from the output of get.GPD.fit(). Hence, the subcomponent xi$fit of the output of get.GPD.fit() can contain less columns than the chosen number of bootstrap replications for creating x (each bootstrap replication with failed gam() calls is omitted). If there is any such failure, get.GPD.fit() outputs a warning. These failures typically happen for too small sample sizes.

Value

get.lambda.fit() returns a list with components

covar: (unique/minimalized) covariate combinations;

fit: corresponding fitted values of lambda.

lambda.predict() returns a list with components

covar: covariate combinations as provided by newdata;

predict: predicted lambda;

CI.low: lower confidence interval (based on predicted values);

CI.up: upper confidence interval (based on predicted values).

get.GPD.fit() returns a list with components

xi: list with components

- covar: (possibly empty) data.frame containing the unique/minimal covariate combinations for the covariates used for fitting ξ;
- fit: corresponding fitted ξ;
- CI.low: lower confidence interval (bootstrapped pointwise two-sides 1-α);
- CI.up: upper confidence interval (bootstrapped pointwise two-sides 1-α);
- boot: matrix containing the corresponding bootstrapped ξ's (or NULL if none of the bootstrap repetitions worked).

beta: similar as for xi.

GPD.predict() returns a list with components

xi: list with components

`covar`: `data.frame` containing the covariate combinations as provided by `xi.newdata`;
`predict`: predicted ξ's;

`beta`: similar as for `xi`.

`risk.measure()` returns a vector of values of the selected risk measure.

Author(s)

Marius Hofert

References

Chavez-Demoulin, V., Embrechts, P., and Hofert, M., An extreme value approach for modeling Operational Risk losses depending on covariates.

Examples

```
## see demo(game) for how to use these functions
```

`Gauss` *Multivariate Gauss Distribution*

Description

Functions for evaluating multivariate normal density, generating random variates, fitting and testing.

Usage

```
dmnorm(x, mu, Sigma, log = FALSE)
fit.norm(data)
rmnorm(n, mu = 0, Sigma)
MardiaTest(data)
jointnormalTest(data, dist = c("chisquare", "beta"), plot = TRUE)
```

Arguments

`data`	`matrix`, data set.
`dist`	`character`, "chisquare" performs test against χ^2 distribution, which is an approximation; "beta" performs a test against a scaled beta distribution.
`log`	`logical`, whether log density values shall be returned.
`n`	`integer`, count of random variates.
`mu`	`numeric`, location parameters.
`plot`	`logical`, whether test result shall be plotted.
`Sigma`	`matrix`, covariance matrix.
`x`	`matrix`, density is evaluated per row.

Examples

```
BiDensPlot(func = dmnorm, mu = c(0, 0), Sigma = equicorr(2, -0.7))
S <- equicorr(d = 3, rho = 0.7)
data <- rmnorm(1000, Sigma = S)
fit.norm(data)
S <- equicorr(d = 10, rho = 0.6)
data <- rmnorm(1000, Sigma = S)
MardiaTest(data)
## Dow Jones Data
data(DJ)
r <- returns(DJ)
stocks <- c("AXP","EK","BA","C","KO","MSFT",
            "HWP","INTC","JPM","DIS")
ss <- window(r[, stocks], "1993-01-01", "2000-12-31")
jointnormalTest(ss)
```

GEV *Generalized Extreme Value Distribution*

Description

Density, quantiles, cumulative probability, and fitting of the Generalized Extreme Value distribution.

Usage

```
pGEV(q, xi, mu = 0, sigma = 1)
qGEV(p, xi, mu = 0, sigma = 1)
dGEV(x, xi, mu = 0, sigma = 1, log = FALSE)
rGEV(n, xi, mu = 0, sigma = 1)
fit.GEV(maxima, ...)
```

Arguments

log	logical, whether log values of density should be returned, default is FALSE.
maxima	vector, block maxima data
mu	numeric, location parameter.
n	integer, count of random variates.
p	vector, probabilities.
q	vector, quantiles.
sigma	numeric, scale parameter.
x	vector, values to evaluate density.
xi	numeric, shape parameter.
...	ellipsis, arguments are passed down to optim().

Value

numeric, probability (pGEV), quantile (qGEV), density (dGEV) or random variates (rGEV) for the GEV distribution with shape parameter ξ, location parameter μ and scale parameter σ. A list object in case of `fit.GEV()`.

See Also

GPD

Examples

```
quantValue <- 4.5
pGEV(q = quantValue, xi = 0, mu = 1.0, sigma = 2.5)
pGumbel(q = quantValue, mu = 1.0, sigma = 2.5)
## Fitting to monthly block-maxima
data(nasdaq)
l <- -returns(nasdaq)
em <- timeLastDayInMonth(time(l))
monmax <- aggregate(l, by = em, FUN = max)
mod1 <- fit.GEV(monmax)
```

GHYP *Uni- and Multivariate Generalized Hyperbolic Distribution*

Description

Values of density and random number generation for uni- and multivariate Generalized Hyperbolic distribution in new QRM parameterization (χ, ψ, γ) and in standard parametrization (α, β, δ); univariate only. See pp. 77–81 in QRM. The special case of a multivariate symmetric GHYP is implemented seperately as function `dsmghyp()`.

Usage

```
dghyp(x, lambda, chi, psi, mu = 0, gamma = 0, log = FALSE)
dmghyp(x, lambda, chi, psi, mu, Sigma, gamma, log = FALSE)
dsmghyp(x, lambda, chi, psi, mu, Sigma, log = FALSE)
dghypB(x, lambda, delta, alpha, beta = 0, mu = 0, log = FALSE)
rghyp(n, lambda, chi, psi, mu = 0, gamma = 0)
rmghyp(n, lambda, chi, psi, Sigma, mu, gamma)
rghypB(n, lambda, delta, alpha, beta = 0, mu = 0)
```

Arguments

`alpha`	`numeric`, parameter(s).
`beta`	`numeric`, skewness parameter.
`chi`	`numeric`, mixing parameter(s).
`delta`	`numeric`, parameter(s).

gamma	numeric, skewness parameter(s).
lambda	numeric, mixing parameter(s).
log	logical, should log density be returned; default is FALSE.
mu	numeric, location parameter(s).
n	integer, count of random variates.
psi	numeric, mixing parameter(s).
Sigma	matrix, dispersion matrix for multivaraiate GHYP.
x	vector, values to evaluate density.

Details

The univariate QRM parameterization is defined in terms of parameters χ, ψ, γ instead of the α, β, δ model used by Blaesild (1981). If $\gamma = 0$, a normal variance mixture where the mixing variable W has a Generalized Inverse Gaussian distribution (GIG) with parameters λ, χ, ψ is given, with heavier tails. If $\gamma > 0$, a normal mean-variance mixture where the mean is also perturbed to equal $\mu + (W * \gamma)$ which introduces asymmetry as well, is obtained. Values for λ and μ are identical in both QRM and B parameterizations. The dispersion matrix Σ does not appear as argument in the univariate case since its value is identically one.

Value

numeric, value(s) of density or log-density (dghyp, dmghyp, dsmghyp and dghypB) or random sample (rghyp, rmghyp, rghypB)

Note

Density values from dgyhp() should be identical to those from dghypB() if the α, β, δ parameters of the B type are translated to the corresponding γ, χ, ψ parameters of the QRM type by formulas on pp 79–80 in QRM.
If γ is a vector of zeros, the distribution is elliptical and `dsmghyp()` is utilised in `dmghyp()`. If $\lambda = (d+1)/2$, a d-dimensional hyperbolic density results. If $\lambda = 1$, the univariate marginals are one-dimensional hyperbolics. If $\lambda = -1/2$, the distribution is Normal Inverse Gaussian (NIG). If $\lambda > 0$ and $\chi = 0$, one obtains a Variance Gamma distribution (VG). If one can define a constant ν such that $\lambda = (-1/2) * \nu$ and $\chi = \nu$ then one obtains a multivariate skewed-t distribution. See p. 80 of QRM for details.

Examples

```
old.par <- par(no.readonly = TRUE)
par(mfrow = c(2, 2))
ll <- c(-4, 4)
BiDensPlot(func = dmghyp, xpts = ll, ypts = ll, mu = c(0, 0),
           Sigma = equicorr(2, -0.7), lambda = 1, chi = 1, psi = 1,
           gamma = c(0, 0))
BiDensPlot(func = dmghyp, type = "contour", xpts = ll, ypts = ll,
           mu = c(0, 0), Sigma = equicorr(2, -0.7), lambda = 1,
           chi = 1, psi = 1, gamma = c(0, 0))
BiDensPlot(func = dmghyp, xpts = ll, ypts = ll, mu = c(0, 0),
```

```
            Sigma = equicorr(2, -0.7), lambda = 1, chi = 1, psi = 1,
            gamma = c(0.5, -0.5))
BiDensPlot(func = dmghyp, type = "contour", xpts = ll, ypts = ll,
            mu = c(0, 0), Sigma = equicorr(2, -0.7), lambda = 1,
            chi = 1, psi = 1, gamma = c(0.5, -0.5))
par(old.par)
```

GIG *Generalized Inverse Gaussian Distribution*

Description

Calculates (log) moments of univariate generalized inverse Gaussian (GIG) distribution and generating random variates.

Usage

```
EGIG(lambda, chi, psi, k = 1)
ElogGIG(lambda, chi, psi)
rGIG(n, lambda, chi, psi, envplot = FALSE, messages = FALSE)
```

Arguments

chi	numeric, chi parameter.
envplot	logical, whether plot of rejection envelope should be created.
k	integer, order of moments.
lambda	numeric, lambda parameter.
messages	logical, whether a message about rejection rate should be returned.
n	integer, count of random variates.
psi	numeric, psi parameter.

Details

Normal variance mixtures are frequently obtained by perturbing the variance component of a normal distribution; here this is done by multiplying the square root of a mixing variable assumed to have a GIG distribution depending upon three parameters (λ, χ, ψ). See p.77 in QRM.
Normal mean-variance mixtures are created from normal variance mixtures by applying another perturbation of the same mixing variable to the mean component of a normal distribution. These perturbations create Generalized Hyperbolic Distributions. See pp. 78–81 in QRM. A description of the GIG is given on page 497 in QRM Book.

Value

(log) mean of distribution or vector random variates in case of `rgig()`.

GPD *Generalized Pareto Distribution*

Description

Density, quantiles, and cumulative probability of the Generalized Pareto distribution.

Usage

```
pGPD(q, xi, beta = 1)
qGPD(p, xi, beta = 1)
dGPD(x, xi, beta = 1, log = FALSE)
rGPD(n, xi, beta = 1)
```

Arguments

beta	numeric, scale parameter.
log	logical, whether log values of density should be returned.
n	integer, count of random variates.
p	vector, probabilities.
q	vector, quantiles.
x	vector, values to evaluate density.
xi	numeric, shape parameter.

Value

numeric, probability (pGPD), quantile (qGPD), density (dGPD) or random variates (rGPD) for the GPD with scale parameter β and shape parameter ξ.

See Also

GEV, POT

Gumbel *Gumbel Distribution*

Description

Density, quantiles, and cumulative probability of the Gumbel distribution. The standard Gumbel has μ value of 0 and σ value of one.

Usage

```
dGumbel(x, mu = 0, sigma = 1, log = FALSE)
qGumbel(p, mu = 0, sigma = 1)
pGumbel(q, mu = 0, sigma = 1)
rGumbel(n, mu = 0, sigma = 1)
```

Arguments

log	logical, whether log values of density should be returned.
mu	numeric, location parameter.
n	integer, count of random variates.
p	vector, probabilities.
q	vector, quantiles.
sigma	numeric, scale parameter.
x	vector, values to evaluate density.

Value

numeric, probability (pGumbel()), quantile (qGumbel()), density (dGumbel()) or random variates (rGumbel()) for the Gumbel distribution with location parameter μ and scale parameter σ.

Examples

```
rGumbelSim <- rGumbel(1000, 1.0, 2.5)
quantValue <- 4.5
pGEV(q = quantValue, xi = 0, mu = 1.0, sigma = 2.5)
pGumbel(q = quantValue, mu = 1.0, sigma = 2.5)
```

---

hsi *Hang Seng Stock Market Index*

Description

The hsi timeSeries dataset provides the daily closing value for the Hanh Seng Index from January 1994 to March 2004. In addition, the data set is also made available as a data.frame.

Usage

```
data(hsi)
data(hsi.df)
```

Examples

```
data(hsi)
head(hsi)
```

Kendall *Kendall's Rank Correlation*

Description

Calculates Kendall's rank correlations. The function is a wrapper to `cor()`.

Usage

```
Kendall(data, ...)
```

Arguments

`data`	`matrix` or `data.frame`.
`...`	ellipsis, arguments are passed down to `cor()`

Value

matrix

See Also

`cor`, `Spearman`

Examples

```
S <- equicorr(d = 3, rho = 0.5)
data <- rmnorm(1000, Sigma = S)
Kendall(data)
```

nasdaq *NASDAQ Stock Market Index*

Description

The `nasdaq` timeSeries dataset provides the daily closing value for the NASDAQ index from January 1994 to March 2004. In addition, the data set is also made available as a `data.frame`.

Usage

```
data(nasdaq)
data(nasdaq.df)
```

Examples

```
data(nasdaq)
head(nasdaq)
```

NH *Normal Inverse Gaussian and Hyperbolic Distribution*

Description

Functions for fitting uni- and multivariate NIG and HYP distribution.

Usage

```
fit.NH(data, case = c("NIG", "HYP"), symmetric = FALSE,
       se = FALSE, ...)
fit.mNH(data, symmetric = FALSE, case = c("NIG", "HYP"),
        kvalue = NA, nit = 2000, tol = 1e-10, ...)
MCECMupdate(data, mix.pars, mu, Sigma, gamma, optpars, optfunc,
xieval=FALSE, ...)
MCECM.Qfunc(lambda, chi, psi, delta, eta, xi)
EMupdate(data, mix.pars, mu, Sigma, gamma, symmetric,
         scaling = TRUE, kvalue = 1)
```

Arguments

case	character, whether NIG or HYP shall be used.
chi	numeric, chi parameter.
data	numeric, data.
delta	numeric, delta parameter.
eta	numeric, eta parameter.
kvalue	numeric, value to which the determinant of the dispersion matrix is constrained.
lambda	numeric, lambda parameter.
mix.pars	vector, values of lambda, chi and psi.
mu	numeric, value of location parameters.
nit	integer, maximum number of iterations.
optpars	vector, parameters to optimize over.
optfunc	function, the function to be optimized.
psi	numeric, pi parameter.
scaling	logical, whether determinant scaling of Sigma shall be fixed.
se	logical, whether standard errors should be calculated.
Sigma	matrix, value of Sigma.
symmetric	logical, whether symmetric case should be fitted.
tol	numeric, tolerance for convergence.
gamma	numeric, value of gamma
xi	numeric, xi parameter.
xieval	logical, whether log moment xi shall be evaluated.
...	ellipsis, arguments are passed down to optim().

Details

`fit.NH()`: See pages 78–80 of QRM. Case 'NIG' sets $\lambda = -1/2$; case 'HYP' sets $\lambda = 1$.
`fit.mNH()`: Fitting is accomplished by using a variant of the EM algorithm (see pages 81–83 in QRM).
`MCECMupdate()`: updates estimates of mixing parameters in EM estimation of generalized hyperbolic (see Algorithm 3.14, steps (5) and (6) on page 83 in QRM).
`MCECM.Qfunc()`: a functional form that must be optimized when fitting members of generalized hyperbolic family with an MCECM algorithm (see function Q2 on page 82 of QRM).
`EMupdate()`: updates estimates of location (μ), dispersion (Σ) and skewness (γ) parameters in EM estimation of multivariate generalized hyperbolic distributions (see pages 81–83 in QRM; in that case k is the determinant of the sample covariance matrix. "EM" is an acronym for for "Expectation-Maximization" type of algorithm used to fit proposed multivariate hyperbolic models to actual data).

Examples

```
data(DJ)
r <- returns(DJ)
s <- window(r[, "MSFT"], "1993-01-01", "2000-12-31")
mod.NIG <- fit.NH(100 * s, method = "BFGS")
## multivariate
stocks <- c("AXP","EK","BA","C","KO","MSFT",
            "HWP","INTC","JPM","DIS")
ss <- window(r[, stocks], "1993-01-01", "2000-12-31")
fridays <- time(ss)[isWeekday(time(ss), wday = 5)]
ssw <- aggregate(ss, by = fridays, FUN = sum)
mod.mNIG <- fit.mNH(ssw, symmetric = FALSE, case = "NIG")
```

---

nikkei *Nikkei Stock Market Index*

Description

The `nikkei` timeSeries dataset provides the daily closing value for the Nikkei index from January 1994 to March 2004. In addition, the data set is also made available as a `data.frame`.

Usage

```
data(nikkei)
data(nikkei.df)
```

Examples

```
data(nikkei)
head(nikkei)
```

Pconstruct *Assemble a Correlation Matrix for ML Copula Fitting*

Description

This function converts a vector of values representing the terms of a lower triangular matrix A with ones on the diagonal and returns the correlation matrix corresponding to the covariance matrix AA' (see page 235 in QRM).

Usage

```
Pconstruct(theta)
```

Arguments

theta	vector, elements of a lower triangular matrix A with ones on the diagonal.

Value

matrix

See Also

link{Pdeconstruct}

Examples

```
P <- Pconstruct(1:6)
eigen(P)
Pdeconstruct(P)
```

Pdeconstruct *Disassemble a Correlation Matrix for ML Copula Fitting*

Description

This function takes a correlation matrix P and returns the elements of a lower-triangular matrix A with ones on the diagonal such that P is the corelation matrix corresponding to the covariance matrix AA' (see page 235 in QRM).

Usage

```
Pdeconstruct(P)
```

Arguments

P	matrix, a correlation matrix

Value

vector

See Also

Pconstruct

Examples

```
P <- Pconstruct(1:6)
Pdeconstruct(P)
```

PointProcess *Point Processes*

Description

Functions for estimating point processes.

Usage

```
extremalPP(data, threshold = NA, nextremes = NA, ...)
unmark(PP)
fit.POT(PP, markdens = "GPD", ...)
fit.sePP(PP, model = c("Hawkes", "ETAS"), mark.influence = TRUE,
         std.errs = FALSE, ...)
fit.seMPP(PP, markdens = "GPD", model = c("Hawkes", "ETAS"),
          mark.influence = TRUE, predictable = FALSE,
          std.errs = FALSE, ...)
stationary.sePP(sePP)
sePP.negloglik(theta, PP, case)
seMPP.negloglik(theta, PP, case, markdens)
volfunction(anytimes, times, marks, theta, model)
## S3 method for class 'MPP'
plot(x, ...)
## S3 method for class 'PP'
plot(x, ...)
## S3 method for class 'sePP'
plot(x, ...)
```

Arguments

anytimes	vector, times at which to calculate self-excitement function.
data	timeSeries object or vector.
case	numeric, indicating Hawkes or ETAS models and whether marks may have an influence on future points.

markdens	character, name of density of mark distribution, currently only "GPD".
mark.influence	logical, whether marks of marked point process may influence the self-excitement.
marks	vector, marks associated with point events.
model	character, name of self-exciting model.
nextremes	integer, count of upper extremes to be used.
PP	list, a point process object of class PP or MPP.
predictable	logical, whether previous events may influence the scaling of mark distribution.
sePP	list, a fitted self-exciting process created with fit.sePP() or a marked self-exciting process created with fit.seMPP().
std.errs	logical, whether standard errors should be computed.
theta	vector, parameters of self-excitement function.
threshold	numeric, threshold value.
times	vector, times of point events.
x	list, a (un/marked) point process object of class PP/MPP.
...	ellipsis, arguments passed to plot() or to fit.GPD() for fit.POT() or to nlminb() for functions fit.sePP() and fit.seMPP or to julian() for extremalPP.

Details

extremalPP(): returns a list describing a marked point process (see pages 298-301 of QRM).
fit.POT(): fits the POT (peaks-over-threshold) model to a point process of class PP or MPP. Note that if point process is of class PP, then function simply esitmates the rate of a homogeneous Poisson process (see pages 301–305 of QRM).
fit.seMPP(): fits a marked self-exciting process to a point process object of class MPP.
fit.sePP(): fits self-exciting process to a point process object of class PP (unmarked) or MPP (marked).
seMPP.negloglik(): evaluates negative log-likelihood of a marked self-exciting point process model; this objective function will be passed to the optimizing function.
sePP.negloglik(): evaluates negative log-likelihood of a self-exciting point process model (unmarked).
stationary.sePP(): checks a sufficient condition for stationarity of a self-exciting model and gives information about cluster size.
unmark(): strips marks from a marked point process.
volfunction(): calculates a self-excitement function for use in the negloglik methods used in fit.sePP() and fit.seMPP().

Value

The function extremalPP() returns a list describing class MPP (marked point process) consisting of times and magnitudes of threshold exceedances:

times	vector of julian day counts (since 1/1/1960) for each exceedance
marks	vector of exceedances values (differences between value and threshold at each mark)

startime	the julian count one day prior to the first date in the entire timeSeries
endtime	value of last julian count in entire timeSeries
threshold	value of threshold above which exceedances are calculated

The functions fit.POT(), fit.seMPP(), and fit.sePP() return a list containing the fitted model. The plot-methods return invisibly the data for producing these.

See Also

GPD, nlminb

Examples

```
## Extremal PP
data(sp500)
l <- -returns(sp500)
lw <- window(l, start = "1995-12-31", end = end(l))
mod1 <- extremalPP(lw, ne = 100)
mod1$marks[1:5]
mod1$threshold
mod2a <- fit.sePP(mod1, mark.influence = FALSE, std.errs = TRUE)
mod2b <- fit.seMPP(mod1, mark.influence = FALSE, std.errs = TRUE)
stationary.sePP(mod2b)
mod2c <- fit.POT(mod1, method = "BFGS")
plot(mod1)
plot(unmark(mod1))
plot(mod2a)
```

POT *Peaks-over-Threshold Method*

Description

Functions for fitting, analysing and risk measures according to POT/GPD

Usage

```
fit.GPD(data, threshold = NA, nextremes = NA, type = c("ml", "pwm"),
        information = c("observed", "expected"),
        optfunc = c("optim", "nlminb"), verbose = TRUE, ...)
plotTail(object, ppoints.gpd = ppoints(256), main = "Estimated tail probabilities",
         xlab = "Exceedances x", ylab = expression(1-hat(F)[n](x)), ...)
showRM(object, alpha, RM = c("VaR", "ES"),
       like.num = 64, ppoints.gpd = ppoints(256),
       xlab = "Exceedances x", ylab = expression(1-hat(F)[n](x)),
       legend.pos = "topright", pre.0.4.9=FALSE, ...)
findthreshold(data, ne)
MEplot(data, omit = 3., main = "Mean-Excess Plot", xlab = "Threshold",
```

```
       ylab = "Mean Excess", ...)
xiplot(data, models = 30., start = 15., end = 500., reverse = TRUE,
       ci = 0.95, auto.scale = TRUE, labels = TRUE, table = FALSE, ...)
hill(data, k, tail.index = TRUE)
hillPlot(data, option = c("alpha", "xi", "quantile"), start = 15,
         end = NA, reverse = FALSE, p = NA, ci = 0.95,
         auto.scale = TRUE, labels = TRUE, ...)
plotFittedGPDvsEmpiricalExcesses(data, threshold = NA, nextremes = NA)
RiskMeasures(out, p)
```

Arguments

alpha	numeric, probability level(s).
auto.scale	logical, whether plot should be automatically scaled.
ci	numeric, probability for asymptotic confidence bands.
data	numeric, data vector or timesSeries.
end	integer, maximum number of exceedances to be considered.
information	character, whether standard errors should be calculated with "observed" or "expected" information. This only applies to maximum likelihood type; for "pwm" type "expected" information is used if possible.
k	number (greater than or equal to 2) of order statistics used to compute the Hill plot.
labels	logical, whether axes shall be labelled.
legend.pos	if not NULL, position of legend().
pre.0.4.9	logical, whether behavior previous to version 0.4-9 applies (returning the risk measure estimate and confidence intervals instead of just invisible()).
like.num	integer, count of evaluations of profile likelihood.
main,xlab,ylab	title, x axis and y axis labels.
models	integer, count of consecutive gpd models to be fitted; *i.e.*, the count of different thresholds at which to re-estimate ξ; this many ξ estimates will be plotted.
ne	integer, count of excesses above the threshold.
nextremes	integer, count of upper extremes to be used.
object	list, returned value from fitting GPD
omit	integer, count of upper plotting points to be omitted.
optfunc	character, function used for ML-optimization.
verbose	logical indicating whether warnings are given; currently only applies in the case where type="pwm" and xi > 0.5.
option	logical, whether "alpha", "xi" (1 / alpha) or "quantile" (a quantile estimate) should be plotted.
out	list, returned value from fitting GPD.
p	vector, probability levels for risk measures.
ppoints.gpd	points in (0,1) for evaluating the GPD tail estimate.

reverse	logical, plot ordered by increasing threshold or number of extremes.
RM	character, risk measure, either "VaR" or "ES"
start	integer, lowest number of exceedances to be considered.
table	logical, printing of a result table.
tail.index	logical indicating whether the Hill estimator of $alpha$ (the default) or $1/alpha$ is computed.
threshold	numeric, threshold value.
type	character, estimation by either ML- or PWM type.
...	ellpsis, arguments are passed down to either plot() or optim() or nlminb().

Details

hillplot(): This plot is usually calculated from the alpha perspective. For a generalized Pareto analysis of heavy-tailed data using the gpd function, it helps to plot the Hill estimates for xi. See pages 286–289 in QRM. Especially note that Example 7.28 suggests the best estimates occur when the threshold is very small, perhaps 0.1 of the sample size (10–50 order statistics in a sample of size 1000). Hence one should NOT be using a 95 percent threshold for Hill estimates.
MEplot(): An upward trend in plot shows heavy-tailed behaviour. In particular, a straight line with positive gradient above some threshold is a sign of Pareto behaviour in tail. A downward trend shows thin-tailed behaviour whereas a line with zero gradient shows an exponential tail. Because upper plotting points are the average of a handful of extreme excesses, these may be omitted for a prettier plot.
plotFittedGPDvsEmpiricalExcesses(): Build a graph which plots the GPD fit of excesses over a threshold u and the corresponding empirical distribution function for observed excesses.
RiskMeasures(): Calculates risk measures (VaR or ES) based on a generalized Pareto model fitted to losses over a high threshold.
xiplot(): Creates a plot showing how the estimate of shape varies with threshold or number of extremes.

See Also

GEV

Examples

```
data(danish)
plot(danish)

MEplot(danish)

xiplot(danish)

hillPlot(danish, option = "alpha", start = 5, end = 250, p = 0.99)
hillPlot(danish, option = "alpha", start = 5, end = 60, p = 0.99)

plotFittedGPDvsEmpiricalExcesses(danish, nextremes = 109)
u <- quantile(danish, probs=0.9, names=FALSE)
plotFittedGPDvsEmpiricalExcesses(danish, threshold = u)
```

```
findthreshold(danish, 50)
mod1 <- fit.GPD(danish, threshold = u)

RiskMeasures(mod1, c(0.95, 0.99))
plotTail(mod1)

showRM(mod1, alpha = 0.99, RM = "VaR", method = "BFGS")
showRM(mod1, alpha = 0.99, RM = "ES", method = "BFGS")

mod2 <- fit.GPD(danish, threshold = u, type = "pwm")
mod3 <- fit.GPD(danish, threshold = u, optfunc = "nlminb")

## Hill plot manually constructed based on hill()

## generate data
set.seed(1)
n <- 1000 # sample size
U <- runif(n)
X1 <- 1/(1-U) # ~ F_1(x) = 1-x^{-1}, x >= 1 => Par(1)
F2 <- function(x) 1-(x*log(x))^(-1) # Par(1) with distorted SV function
X2 <- vapply(U, function(u) uniroot(function(x) 1-(x*log(x))^(-1)-u,
                                    lower=1.75, upper=1e10)$root, NA_real_)

## compute Hill estimators for various k
k <- 10:800
y1 <- hill(X1, k=k)
y2 <- hill(X2, k=k)

## Hill plot
plot(k, y1, type="l", ylim=range(y1, y2, 1),
     xlab=expression("Number"~~italic(k)~~"of upper order statistics"),
     ylab=expression("Hill estimator for"~~alpha),
     main="Hill plot") # Hill plot, good natured case (based on X1)
lines(k, y2, col="firebrick") # Hill "horror" plot (based on X2)
lines(x=c(10, 800), y=c(1, 1), col="royalblue3") # correct value alpha=1
legend("topleft", inset=0.01, lty=c(1, 1, 1), bty="n",
       col=c("black", "firebrick", "royalblue3"),
       legend=as.expression(c("Hill estimator based on"~~
                              italic(F)(x)==1-1/x,
                             "Hill estimator based on"~~
                              italic(F)(x)==1-1/(x~log~x),
                             "Correct value"~~alpha==1)))

## via hillPlot()
hillPlot(X1, option="alpha", start=10, end=800)
hillPlot(X2, option="alpha", start=10, end=800)
```

QQplot *Generic Quantile-Quantile Plot*

Description

Constructs a quantile-quantile plot against a given reference distribution.

Usage

```
QQplot(x, a = 0.5, reference = c("normal", "exp", "student"), ...)
```

Arguments

x	vector, data for QQ-plot.
a	numeric, the offset fraction to be used in ppoints(); typically in (0, 1).
reference	character, name of reference distribution.
...	ellipsis argument, passed down to quantile function of reference distribution.

Details

Special forms like ParetoQQ plots can also be created via this function. E.g., to create a ParetoQQ plot, merely pass log(data) in place of data as the first parameter and use `reference = "exp"` as the reference distribution. The ParetoQQ plot should provide a linear graph when a log transform of the data is plotted against the exponential distribution.

Value

Produces QQ-plot and returns invisibly a list of `(x, y)` pairs.

See Also

ppoints

Examples

```
QQplot(rnorm(1000), reference = "normal")
QQplot(rexp(1000), reference = "exp", rate = 0.3)
```

QRM-defunct *Defunct Functions in Package QRM*

Description

The functions listed below which were contained in the package QRMlib are now defunct. The user is referred to the suggested functions as an alternative.

Details

aggregateMonthlySeries() is defunct. use aggregate() in package **timeSeries**.
aggregateQuarterlySeries is defunct. use aggregate() in package timeSeries.
aggregateSignalSeries() is defunct. use aggregate() in package **timeSeries**.
aggregateWeeklySeries() is defunct. use aggregate() in package **timeSeries**.
besselM3() is defunct. use besselK() in package **base**.
ConvertDFToTimeSeries() is defunct. use timeSeries() in package **timeSeries**.
CovToCor() is defunct. use cov2cor() in package **stats**.
fit.Archcopula2d() is defunct. use fit.AC().
fit.GPDb() is defunct. use fit.GPD().
fit.tcopula.rank() is defunct. use fit.tcopula().
hessb() is defunct. use hessian() in package **numDeriv**.
kurtosisSPlus() is defunct. use kurtosis() in package **timeDate**.
lbeta() is defunct. use lbeta() in package **base**.
mk.returns() is defunct. use returnSeries() in package **timeSeries**.
plotMultiTS() is defunct. use plot() in package **timeSeries**.
psifunc() is defunct. use psi() in package **gsl**.
signalSeries() is defunct. use series() in package **timeSeries**.
symmetrize() is defunct. use forceSymmetric() in package **Matrix**.

---

smi *Swiss Market Index*

Description

The smi timeSeries dataset provides the daily closing value for the Swiss Market index from November 1990 to March 2004. In addition, the data set is also made available as a data.frame.

Usage

```
data(smi)
data(smi.df)
```

Examples

```
data(smi)
head(smi)
```

sp500 *Standard and Poors 500 Index*

Description

The sp500 timeSeries dataset provides the daily closing value for the S and P 500 Index from January 1980 to March 2004. In addition, the data set is also made available as a data.frame.

Usage

```
data(dji)
data(dji.df)
```

Examples

```
data(sp500)
head(sp500)
```

spdata *Standard and Poors Default Data*

Description

The spdata timeSeries dataset contains default data for A, BBB, BB, B and C-rated companies for the years 1981 to 2000. In addition, the data set is also made available as a data.frame.

Usage

```
data(spdata)
data(spdata.df)
```

Source

Standard and Poors Credit Monitor

Examples

```
data(spdata)
head(spdata)
```

spdata.raw *Standard and Poors Default Data*

Description

The spdata.raw timeSeries contains default data for A, BBB, BB, B and C-rated companies for the years 1981 to 2000.

Usage

```
data(spdata.raw)
data(spdata.raw.df)
```

Source

Standard & Poors Credit Monitor

Examples

```
data(spdata.raw)
head(spdata.raw)
```

Spearman *Spearman's Rank Correlation*

Description

Calculates Sperman's rank correlations. The function is a wrapper to cor().

Usage

```
Spearman(data, ...)
```

Arguments

data	matrix or data.frame
...	ellipsis, arguments are passed down to cor()

Value

matrix

See Also

cor, Kendall

Examples

```
S <- equicorr(d = 3, rho = 0.5)
data <- rmnorm(1000, Sigma = S)
Spearman(data)
```

Student *Student's t Distribution*

Description

Functions for evaluating density, fitting and random variates of multivaraite Student's t distribution and routines for quantiles and fitting of univariate distribution.

Usage

```
dmt(x, df, mu, Sigma, log = FALSE)
rmt(n, df = 4, mu = 0, Sigma)
qst(p, mu = 0, sd = 1, df, scale = FALSE)
fit.st(data, ...)
fit.mst(data, nit = 2000, tol = 1e-10, ...)
```

Arguments

x	matrix, dimension $n \times d$; density is evaluated for each row.
df	numeric, degrees of freedom.
mu	numeric, location parameters.
Sigma	matrix, dispersion matrix.
log	logical, returning log density values.
data	numeric, data used for uni- and multivariate fitting.
nit	integer, number of iterations of EM-type algorithm.
tol	numeric, tolerance of improvement for stopping iteration.
p	numeric, probability.
sd	numeric, scale parameters.
scale	logical, scaling Student's t distribution.
n	integer, count of random variates.
...	ellipsis, arguments are passed down to optim() in fit.st() and to MCECMupdate() in fit.mst().

See Also

link{EMupdate}, link{MCECMupdate}, and link{MCECM.Qfunc}

Examples

```
BiDensPlot(func = dmt, xpts = c(-4, 4), ypts = c(-4, 4), mu = c(0, 0),
           Sigma = equicorr(2, -0.7), df = 4)
## Quantiles of univariate Student's t
p <- c(0.90,0.95)
s <- 0.2 * 10000/sqrt(250)
qst(p, sd = s, df = 4, scale = TRUE)
## Fitting multivariate Student's t
Sigma <- diag(c(3, 4, 5)) %*% equicorr(3, 0.6) %*% diag(c(3, 4, 5))
mu <- c(1, 2 ,3)
tdata <- rmt(1000, 4, mu = mu, Sigma = Sigma)
mod1 <- fit.mst(tdata, method = "BFGS")
## DJ data
data(DJ)
r <- returns(DJ)
s <- window(r[, "MSFT"], "1993-01-01", "2000-12-31")
mod.t1 <- fit.st(100 * s)
stocks <- c("AXP","EK","BA","C","KO","MSFT",
            "HWP","INTC","JPM","DIS")
ss <- window(r[, stocks], "1993-01-01", "2000-12-31")
fridays <- time(ss)[isWeekday(time(ss), wday = 5)]
ssw <- aggregate(ss, by = fridays, FUN = sum)
mod.t2 <- fit.mst(ssw, method = "BFGS")
```

VaRbound *Computing lower and upper bounds for the (smallest or largest) VaR*

Description

`VaRbound()` computes lower and upper bounds for the lower or upper Value-at-Risk bound.

Usage

```
VaRbound(alpha, N, qmargins, bound = c("upper", "lower"), verbose = FALSE)
```

Arguments

`alpha`	confidence level in (0,1).
`N`	tail discretization parameter; see Embrechts et al. (2013).
`qmargins`	`list` containing the marginal quantile functions.
`bound`	`character` string indicating the VaR bound to be approximated (largest (default) or smallest).
`verbose`	`logical` indicating whether progress information is displayed.

Details

Due to the nature of the rearrangement algorithm, note that this purely R based implementation can be slow.

Value

numeric vector of length two, containing the lower and upper bound for the (chosen) Value-at-Risk estimate.

Author(s)

Marius Hofert.

References

Embrechts, P., Puccetti, G., and Rüschendorf, L. (2013), Model uncertainty and VaR aggregation, *Journal of Banking and Finance* **37**(8), 2750–2764.

Examples

```
qPar <- function(p, theta) (1-p)^(-1/theta)-1
qmar <- lapply(1:3, function(j) function(p) qPar(p, theta=2.5))
## bounds for the largest VaR
VaRbound(0.99, N=50, qmargins=qmar)
## bounds for the smallest VaR
VaRbound(0.99, N=50, qmargins=qmar, bound="lower")
```

---

xdax *Xetra DAX German Index*

Description

The xdax timeSeries dataset provides the daily closing value for the German Xextra DAX index from January 1994 to March 2004. In addition, the data set is also made available as a data.frame.

Usage

```
data(xdax)
data(xdax.df)
```

Examples

```
data(xdax)
head(xdax)
```

Index