Package 'cvar'

November 3, 2022

Type Package

Title Compute Expected Shortfall and Value at Risk for Continuous Distributions

Version 0.5

Description Compute expected shortfall (ES) and Value at Risk (VaR) from a quantile function, distribution function, random number generator or probability density function. ES is also known as Conditional Value at Risk (CVaR). Virtually any continuous distribution can be specified. The functions are vectorized over the arguments. The computations are done directly from the definitions, see e.g. Acerbi and Tasche (2002) <doi:10.1111/1468-0300.00091>. Some support for GARCH models is provided, as well.

URL https://geobosh.github.io/cvar/ (doc), https://github.com/GeoBosh/cvar (devel)

BugReports https://github.com/GeoBosh/cvar/issues

Imports gbutils, Rdpack (>= 0.8)

RdMacros Rdpack

License GPL (>= 2)

Collate VaR.R cvar-package.R garch.R

RoxygenNote 7.2.0

Suggests testthat, fGarch, PerformanceAnalytics

NeedsCompilation no

Author Georgi N. Boshnakov [aut, cre]

Maintainer Georgi N. Boshnakov <georgi.boshnakov@manchester.ac.uk>

Repository CRAN

Date/Publication 2022-11-03 10:00:06 UTC

R topics documented:

`cvar-package` *Compute Conditional Value-at-Risk and Value-at-Risk*

Description

Compute expected shortfall (ES) and Value at Risk (VaR) from a quantile function, distribution function, random number generator or probability density function. ES is also known as Conditional Value at Risk (CVaR). Virtually any continuous distribution can be specified. The functions are vectorised over the arguments. Some support for GARCH models is provided, as well.

Details

There is a huge number of functions for computations with distributions in core R and in contributed packages. Pdf's, cdf's, quantile functions and random number generators are covered comprehensively. The coverage of expected shortfall is more patchy but a large collection of distributions, including functions for expected shortfall, is provided by Nadarajah et al. (2013). Peterson and Carl (2018) and Dutang et al. (2008) provide packages covering comprehensively various aspects of risk measurement, including some functions for expected shortfall.

Package **cvar** is a small package with, essentially, two main functions — `ES` for computing the expected shortfall and `VaR` for Value at Risk. The user specifies the distribution by supplying one of the functions that define a continuous distribution—currently this can be a quantile function (qf), cumulative distribution function (cdf) or probability density function (pdf). Virtually any continuous distribution can be specified.

The functions are vectorised over the parameters of the distributions, making bulk computations more convenient, for example for forecasting or model evaluation.

The name of this package, "cvar", comes from *Conditional Value at Risk* (CVaR), which is an alternative term for expected shortfall.

We chose to use the standard names `ES` and `VaR`, despite the possibility for name clashes with same named functions in other packages, rather than invent possibly difficult to remember alternatives. Just call the functions as `cvar::ES` and `cvar::VaR` if necessary.

Locations-scale transformations can be specified separately from the other distribution parameters. This is useful when such parameters are not provided directly by the distribution at hand. The use of these parameters often leads to more efficient computations and better numerical accuracy even if the distribution has its own parameters for this purpose. Some of the examples for `VaR` and `ES` illustrate this for the Gaussian distribution.

Since VaR is a quantile, functions computing it for a given distribution are convenience functions. `VaR` exported by **cvar** could be attractive in certain workflows because of its vectorised distribution parameters, the location-scale transformation, and the possibility to compute it from cdf's when quantile functions are not available.

Some support for GARCH models is provided, as well. It is currently under development, see `predict.garch1c1` for current functionality.

In practice, we may need to compute VaR associated with data. The distribution comes from fitting a model. In the simplest case, we fit a distribution to the data, assuming that the sample is i.i.d. For example, a normal distribution $N(\mu, \sigma^2)$ can be fitted using the sample mean and sample variance as estimates of the unknown parameters μ and σ^2, see section 'Examples'. For other common distributions there are specialised functions to fit their parameters and if not, general optimisation routines can be used. More soffisticated models may be used, even time series models such as GARCH and mixture autoregressive models.

Author(s)

Georgi N. Boshnakov

References

Christophe Dutang, Vincent Goulet, Mathieu Pigeon (2008). "actuar: An R Package for Actuarial Science." *Journal of Statistical Software*, **25**(7), 38. doi: 10.18637/jss.v025.i07.

Saralees Nadarajah, Stephen Chan, Emmanuel Afuecheta (2013). *VaRES: Computes value at risk and expected shortfall for over 100 parametric distributions*. R package version 1.0, https://CRAN.R-project.org/package=VaRES.

Brian G. Peterson, Peter Carl (2018). *PerformanceAnalytics: Econometric Tools for Performance and Risk Analysis*. R package version 1.5.2, https://CRAN.R-project.org/package=PerformanceAnalytics.

See Also

`ES`, `VaR`

Examples

```
## see the examples for ES(), VaR(), predict.garch1c1()
```

`ES` *Compute expected shortfall (ES) of distributions*

Description

Compute the expected shortfall for a distribution.

Usage

```
ES(dist, p_loss = 0.05, ...)

## Default S3 method:
ES(
  dist,
  p_loss = 0.05,
  dist.type = "qf",
  qf,
  ...,
  intercept = 0,
  slope = 1,
  control = list(),
  x
)

## S3 method for class 'numeric'
ES(
  dist,
  p_loss = 0.05,
  dist.type = "qf",
  qf,
  ...,
  intercept = 0,
  slope = 1,
  control = list(),
  x
)
```

Arguments

`dist`	specifies the distribution whose ES is computed, usually a function or a name of a function computing quantiles, cdf, pdf, or a random number generator, see Details.
`p_loss`	level, default is 0.05.
`...`	passed on to `dist`.
`dist.type`	a character string specifying what is computed by `dist`, such as "qf" or "cdf".
`qf`	quantile function, only used if `dist.type = "pdf"`.
`intercept, slope`	compute ES for the linear transformation `intercept + slope*X`, where `X` has distribution specified by `dist`, see Details.
`control`	additional control parameters for the numerical integration routine.
`x`	deprecated and will soon be removed. `x` was renamed to `p_loss`, please use the latter.

Details

ES computes the expected shortfall for distributions specified by the arguments. dist is typically a function (or the name of one). What dist computes is determined by dist.type, whose default setting is "qf" (the quantile function). Other possible settings of dist.type include "cdf" and "pdf". Additional arguments for dist can be given with the "..." arguments.

Argument dist can also be a numeric vector. In that case the ES is computed, effectively, for the empirical cumulative distribution function (ecdf) of the vector. The ecdf is not created explicitly and the quantile function is used instead for the computation of VaR. Arguments in "..." are passed eventually to quantile() and can be used, for example, to select a non-defult method for the computation of quantiles.

Except for the exceptions discussed below, a function computing VaR for the specified distribution is constructed and the expected shortfall is computed by numerically integrating it. The numerical integration can be fine-tuned with argument control, which should be a named list, see integrate for the available options.

If dist.type is "pdf", VaR is not computed, Instead, the partial expectation of the lower tail is computed by numerical integration of x * pdf(x). Currently the quantile function is required anyway, via argument qf, to compute the upper limit of the integral. So, this case is mainly for testing and comparison purposes.

A bunch of expected shortfalls is computed if argument x or any of the arguments in "..." are of length greater than one. They are recycled to equal length, if necessary, using the normal R recycling rules.

intercept and slope can be used to compute the expected shortfall for the location-scale transformation Y = intercept + slope * X, where the distribution of X is as specified by the other parameters and Y is the variable of interest. The expected shortfall of X is calculated and then transformed to that of Y. Note that the distribution of X doesn't need to be standardised, although it typically will.

The intercept and the slope can be vectors. Using them may be particularly useful for cheap calculations in, for example, forecasting, where the predictive distributions are often from the same family, but with different location and scale parameters. Conceptually, the described treatment of intercept and slope is equivalent to recycling them along with the other arguments, but more efficiently.

The names, intercept and slope, for the location and scale parameters were chosen for their expressiveness and to minimise the possibility for a clash with parameters of dist (e.g., the Gamma distribution has parameter scale).

Value

a numeric vector

See Also

VaR for VaR,

predict for examples with fitted models

Examples

```
ES(qnorm)
```

```
## Gaussian
ES(qnorm, dist.type = "qf")
ES(pnorm, dist.type = "cdf")

## t-dist
ES(qt, dist.type = "qf", df = 4)
ES(pt, dist.type = "cdf", df = 4)

ES(pnorm, 0.95, dist.type = "cdf")
ES(qnorm, 0.95, dist.type = "qf")
## - VaRES::esnormal(0.95, 0, 1)
## - PerformanceAnalytics::ETL(p=0.05, method = "gaussian", mu = 0,
##                             sigma = 1, weights = 1)             # same

cvar::ES(pnorm, dist.type = "cdf")
cvar::ES(qnorm, dist.type = "qf")
cvar::ES(pnorm, 0.05, dist.type = "cdf")
cvar::ES(qnorm, 0.05, dist.type = "qf")

## this uses "pdf"
cvar::ES(dnorm, 0.05, dist.type = "pdf", qf = qnorm)


## this gives warning (it does more than simply computing ES):
## PerformanceAnalytics::ETL(p=0.95, method = "gaussian", mu = 0, sigma = 1, weights = 1)

## run this if VaRRES is present
## Not run:
x <- seq(0.01, 0.99, length = 100)
y <- sapply(x, function(p) cvar::ES(qnorm, p, dist.type = "qf"))
yS <- sapply(x, function(p) - VaRES::esnormal(p))
plot(x, y)
lines(x, yS, col = "blue")

## End(Not run)
```

`GarchModel` *Specify a GARCH model*

Description

Specify a GARCH model.

Usage

```
GarchModel(model = list(), ..., model.class = NULL)
```

Arguments

model	a GARCH model or a list.
...	named arguments specifying the GARCH model.
model.class	a class for the result. By default GarchModel() decides the class of the result.

Details

Argument model can be the result of a previous call to GarchModel. Arguments in "..." overwrite current components of model.

GarchModel guarantees that code using it will continue to work transparently for the user even if the internal represedtation of GARCH models in this package is changed or additional functionality is added.

Value

an object from suitable GARCH-type class

Examples

```
## GARCH(1,1) with Gaussian innovations
mo1a <- GarchModel(omega = 1, alpha = 0.3, beta = 0.5)
mo1b <- GarchModel(omega = 1, alpha = 0.3, beta = 0.5, cond.dist = "norm")

## equivalently, the parameters can be given as a list
p1 <- list(omega = 1, alpha = 0.3, beta = 0.5)
mo1a_alt <- GarchModel(p1)
mo1b_alt <- GarchModel(p1, cond.dist = "norm")
stopifnot(identical(mo1a, mo1a_alt), identical(mo1b, mo1b_alt))

## additional arguments modify values already in 'model'
mo_alt <- GarchModel(p1, beta = 0.4)

## set also initial values
mo2 <- GarchModel(omega = 1, alpha = 0.3, beta = 0.5, esp0 = - 1.5, h0 = 4.96)

##  GARCH(1,1) with standardised-t_5
mot <- GarchModel(omega = 1, alpha = 0.3, beta = 0.5, cond.dist = list("std", nu = 5))
```

---

predict.garch1c1 *Prediction for GARCH(1,1) time series*

Description

Predict GARCH(1,1) time series.

Usage

```
## S3 method for class 'garch1c1'
predict(object, n.ahead = 1, Nsim = 1000, eps, sigmasq, seed = NULL, ...)
```

Arguments

object	an object from class "garch1c1".
n.ahead	maximum horizon (lead time) for prediction.
Nsim	number of Monte Carlo simulations for simulation based quantities.
eps	the time series to predict, only the last value is used.
sigmasq	the (squared) volatilities, only the last value is used.
seed	an integer, seed for the random number generator.
...	currently not used.

Details

Plug-in prediction intervals and predictive distributions are obtained by inserting the predicted volatility in the conditional densities. For predictions more than one lag ahead these are not the real predictive distributions but the prediction intervals are usually adequate.

For simulation prediction intervals we generate a (large) number of continuations of the given time series. Prediction intervals can be based on sample quantiles. The generated samples are stored in the returned object and can be used for further exploration of the predictive distributions. dist_sim$eps contains the simulated future values of the time series and dist_sim$h the corresponding (squared) volatilities. Both are matrices whose i-th rows contain the predicted quantities for horizon i.

The random seed at the start of the simulations is saved in the returned object. A speficific seed can be requested with argument seed. In that case the simulations are done with the specified seed and the old state of the random number generator is restored before the function returns. This setup is similar to sim_garch1c1.

Value

an object from S3 class "predict_garch1c1" containing the following components:

eps	point predictions (conditional expectations) of the time series (equal to zero for pure GARCH).
h	point predictions (conditional expectations)of the squared volatilities.
model	the model.
call	the call.
pi_plugin	Prediction intervals for the time series, based on plug-in distributions, see Details.
pi_sim	Simulation based prediction intervals for the time series, see Details.
dist_sim	simulation samples from the predictive distributions of the time series and the volatilties.

Note

This function is under development and may be changed.

Examples

```
op <- options(digits = 4)

## set up a model and simulate a time series
mo <- GarchModel(omega = 0.4, alpha = 0.3, beta = 0.5)
a1 <- sim_garch1c1(mo, n = 1000, n.start = 100, seed = 20220305)

## predictions for T+1,...,T+5 (T = time of last value)
## Nsim is small to reduce the load on CRAN, usually Nsim is larger.
a.pred <- predict(mo, n.ahead = 5, Nsim = 1000, eps = a1$eps,
  sigmasq = a1$h, seed = 1234)

## preditions for the time series
a.pred$eps

## PI's for eps - plug-in and simulated
a.pred$pi_plugin
a.pred$pi_sim

## a DIY calculation of PI's using the simulated sample paths
t(apply(a.pred$dist_sim$eps, 1, function(x) quantile(x, c(0.025, 0.975))))

## further investigate the predictive distributions
t(apply(a.pred$dist_sim$eps, 1, function(x) summary(x)))

## compare predictive densities for horizons 2 and 5:
h2 <- a.pred$dist_sim$eps[2, ]
h5 <- a.pred$dist_sim$eps[5, ]
main <- "Predictive densities: horizons 2 (blue) and 5 (black)"
plot(density(h5), main = main)
lines(density(h2), col = "blue")

## predictions of sigma_t^2
a.pred$h

## plug-in predictions of sigma_t
sqrt(a.pred$h)

## simulation predictive densities (PD's) of sigma_t for horizons 2 and 5:
h2 <- sqrt(a.pred$dist_sim$h[2, ])
h5 <- sqrt(a.pred$dist_sim$h[5, ])
main <- "PD's of sigma_t for horizons 2 (blue) and 5 (black)"
plot(density(h2), col = "blue", main = main)
lines(density(h5))

## VaR and ES for different horizons
cbind(h = 1:5,
      VaR = apply(a.pred$dist_sim$eps, 1, function(x) VaR(x, c(0.05))),
```

```
      ES = apply(a.pred$dist_sim$eps, 1, function(x) ES(x, c(0.05))) )

## fit a GARCH(1,1) model to exchange rate data and predict
gmo1 <- fGarch::garchFit(formula = ~garch(1, 1), data = fGarch::dem2gbp,
  include.mean = FALSE, cond.dist = "norm", trace = FALSE)
mocoef <- gmo1@fit$par
mofitted <- GarchModel(omega = mocoef["omega"], alpha = mocoef["alpha1"],
  beta = mocoef["beta1"])
gmo1.pred <- predict(mofitted, n.ahead = 5, Nsim = 1000, eps = gmo1@data,
  sigmasq = gmo1@h.t, seed = 1234)
gmo1.pred$pi_plugin
gmo1.pred$pi_sim

op <- options(op) # restore options(digits)
```

`sim_garch1c1` *Simulate GARCH(1,1) time series*

Description

Simulate GARCH(1,1) time series.

Usage

```
sim_garch1c1(model, n, n.start = 0, seed = NULL)
```

Arguments

`model`	a GARCH(1,1) model, an object obtained from `GarchModel`.
`n`	the length of the generated time series.
`n.start`	number of warm-up values, which are then dropped.
`seed`	an integer to use for setting the random number generator.

Details

The simulated time series is in component `eps` of the returned value. For exploration of algorithms and eestimation procedures, the volatilities and the standardised innovations are also returned.

The random seed at the start of the simulations is saved in the returned object. A speficific seed can be requested with argument `seed`. In that case the simulations are done with the specified seed and the old state of the random number generator is restored before the function returns.

Value

a list with components:

`eps`	the time series,
`h`	the (squared) volatilities,
`eta`	the standardised innovations,
`model`	the GARCH(1,1) model,
`.sim`	a list containing the parameters of the simulation,
`call`	the call.

Note

This function is under development and may be changed.

`VaR` *Compute Value-at-Risk (VaR)*

Description

`VaR` computes the Value-at-Risk of the distribution specified by the arguments. The meaning of the parameters is the same as in `ES`, including the recycling rules.

Usage

```
VaR(dist, p_loss = 0.05, ...)

VaR_qf(
  dist,
  p_loss = 0.05,
  ...,
  intercept = 0,
  slope = 1,
  tol = .Machine$double.eps^0.5,
  x
)

VaR_cdf(
  dist,
  p_loss = 0.05,
  ...,
  intercept = 0,
  slope = 1,
  tol = .Machine$double.eps^0.5,
  x
)
```

```
## Default S3 method:
VaR(
  dist,
  p_loss = 0.05,
  dist.type = "qf",
  ...,
  intercept = 0,
  slope = 1,
  tol = .Machine$double.eps^0.5,
  x
)

## S3 method for class 'numeric'
VaR(dist, p_loss = 0.05, ..., intercept = 0, slope = 1, x)
```

Arguments

`dist`	specifies the distribution whose ES is computed, usually a function or a name of a function computing quantiles, cdf, pdf, or a random number generator, see Details.
`p_loss`	level, default is 0.05.
`...`	passed on to `dist`.
`intercept, slope`	compute VaR for the linear transformation `intercept + slope*X`, where `X` has distribution specified by `dist`, see Details.
`tol`	tollerance
`x`	deprecated and will soon be removed. `x` was renamed to `p_loss`, please use the latter.
`dist.type`	a character string specifying what is computed by `dist`, such as "qf" or "cdf".

Details

`VaR` is S3 generic. The meaning of the parameters for its default method is the same as in `ES`, including the recycling rules.

`VaR_qf` and `VaR_cdf` are streamlined, non-generic, variants for the common case when the `"..."` parameters are scalar. The parameters `x`, `intercept`, and `slope` can be vectors, as for `VaR`.

Argument `dist` can also be a numeric vector. In that case the ES is computed, effectively, for the empirical cumulative distribution function (ecdf) of the vector. The ecdf is not created explicitly and the `quantile` function is used instead for the computation of VaR. Arguments in `"..."` are passed eventually to `quantile()` and can be used, for example, to select a non-defult method for the computation of quantiles.

In practice, we may need to compute VaR associated with data. The distribution comes from fitting a model. In the simplest case, we fit a distribution to the data, assuming that the sample is i.i.d. For example, a normal distribution $N(\mu, \sigma^2)$ can be fitted using the sample mean and sample variance as estimates of the unknown parameters μ and σ^2, see section 'Examples'. For other common

distributions there are specialised functions to fit their parameters and if not, general optimisation routines can be used. More soffisticated models may be used, even time series models such as GARCH and mixture autoregressive models.

Note

We use the traditional definition of VaR as the negated lower quantile. For example, if X are returns on an asset, $\text{VAR}_\alpha = -q_\alpha$, where q_α is the lower α quantile of X. Equivalently, VAR_α is equal to the lower $1 - \alpha$ quantile of $-X$.

See Also

ES for ES,

predict for examples with fitted models

Examples

```
cvar::VaR(qnorm, c(0.01, 0.05), dist.type = "qf")

## the following examples use these values, obtained by fitting a normal distribution to
## some data:
muA <- 0.006408553
sigma2A <- 0.0004018977

## with quantile function, giving the parameters directly in the call:
res1 <- cvar::VaR(qnorm, 0.05, mean = muA, sd = sqrt(sigma2A))
res2 <- cvar::VaR(qnorm, 0.05, intercept = muA, slope = sqrt(sigma2A))
abs((res2 - res1)) # 0, intercept/slope equivalent to mean/sd

## with quantile function, which already knows the parameters:
my_qnorm <- function(p) qnorm(p, mean = muA, sd = sqrt(sigma2A))
res1_alt <- cvar::VaR(my_qnorm, 0.05)
abs((res1_alt - res1))

## with cdf the precision depends on solving an equation
res1a <- cvar::VaR(pnorm, 0.05, dist.type = "cdf", mean = muA, sd = sqrt(sigma2A))
res2a <- cvar::VaR(pnorm, 0.05, dist.type = "cdf", intercept = muA, slope = sqrt(sigma2A))
abs((res1a - res2)) # 3.287939e-09
abs((res2a - res2)) # 5.331195e-11, intercept/slope better numerically

## as above, but increase the precision, this is probably excessive
res1b <- cvar::VaR(pnorm, 0.05, dist.type = "cdf",
                   mean = muA, sd = sqrt(sigma2A), tol = .Machine$double.eps^0.75)
res2b <- cvar::VaR(pnorm, 0.05, dist.type = "cdf",
                 intercept = muA, slope = sqrt(sigma2A), tol = .Machine$double.eps^0.75)
abs((res1b - res2)) # 6.938894e-18 # both within machine precision
abs((res2b - res2)) # 1.040834e-16

## relative precision is also good
abs((res1b - res2)/res2) # 2.6119e-16 # both within machine precision
abs((res2b - res2)/res2) # 3.91785e-15
```

```
## an extended example with vector args, if "PerformanceAnalytics" is present
if (requireNamespace("PerformanceAnalytics", quietly = TRUE)) withAutoprint({
    data(edhec, package = "PerformanceAnalytics")
    mu <- apply(edhec, 2, mean)
    sigma2 <- apply(edhec, 2, var)
    musigma2 <- cbind(mu, sigma2)

    ## compute in 2 ways with cvar::VaR
    vAz1 <- cvar::VaR(qnorm, 0.05, mean = mu, sd = sqrt(sigma2))
    vAz2 <- cvar::VaR(qnorm, 0.05, intercept = mu, slope = sqrt(sigma2))

    vAz1a <- cvar::VaR(pnorm, 0.05, dist.type = "cdf",
                       mean = mu, sd = sqrt(sigma2))
    vAz2a <- cvar::VaR(pnorm, 0.05, dist.type = "cdf",
                       intercept = mu, slope = sqrt(sigma2))

    vAz1b <- cvar::VaR(pnorm, 0.05, dist.type = "cdf",
                   mean = mu, sd = sqrt(sigma2),
                   tol = .Machine$double.eps^0.75)
    vAz2b <- cvar::VaR(pnorm, 0.05, dist.type = "cdf",
                   intercept = mu, slope = sqrt(sigma2),
                   tol = .Machine$double.eps^0.75)

    ## analogous calc. with PerformanceAnalytics::VaR
    vPA <- apply(musigma2, 1, function(x)
        PerformanceAnalytics::VaR(p = .95, method = "gaussian", invert = FALSE,
                                  mu = x[1], sigma = x[2], weights = 1))
    ## the results are numerically the same
    max(abs((vPA - vAz1))) # 5.551115e-17
    max(abs((vPA - vAz2))) #   ""

    max(abs((vPA - vAz1a))) # 3.287941e-09
    max(abs((vPA - vAz2a))) #  1.465251e-10, intercept/slope better

    max(abs((vPA - vAz1b))) # 4.374869e-13
    max(abs((vPA - vAz2b))) # 3.330669e-16
})
```

Index