Brief guide to R package cvar

Georgi N. Boshnakov
University of Manchester

Abstract

Package **cvar** is a small package with, essentially, two 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.

This vignette is part of package **cvar**, version 0.1-0.

Keywords: expected shortfall, ES, CVaR, VaR, value at risk.

1. Introduction

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 **?**. **?** and **?** provide packages covering comprehensively various aspects of risk measurement, including some functions for expected shortfall.

Package **cvar** is a small package with, essentially two 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 computations are done directly from the definitions, see e.g. **?**.

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.

2. Value at Risk and Expected Shortfall

We use the traditional definition of VaR as the negated lower quantile. More specifically, let Y be the variable of interes, such as return on a financial asset. Suppose that it is modelled as a random variable with distribution function $F^Y(y)$. Then VaR is defined as[1]

$$\mathrm{VaR}_\alpha(Y) = -q_\alpha^Y.$$

where q_α^Y is the lower α-quantile of Y, i.e. we have

$$\alpha = F(q_\alpha^Y) = P(Y \leq q_\alpha^Y).$$

Typical values for α are 0.05 and 0.01. Equivalently, we could write[2]

$$\mathrm{VaR}_\alpha(Y) = q_{1-\alpha}^{-Y}.$$

The expected shortfall is the (partial) expectation of $\mathrm{VaR}_\alpha(Y)$:

$$\mathrm{ES}_\alpha(Y) = \frac{1}{\alpha}\int_0^\alpha \mathrm{VaR}_\gamma(Y)d\gamma.$$

Suppose now that Y is obtained from another random variable, X, by a linear transformation:

$$Y = a + bX$$

When this is the case, there is a simple relation between the VaR's and ES's of Y and X:

$$\mathrm{VaR}_\alpha(Y) = -a + b\mathrm{VaR}_\alpha(X)$$
$$\mathrm{ES}_\alpha(Y) = -a + b\mathrm{ES}_\alpha(X)$$

In practice, X is often chosen to be standardised but this is not necessary. Note also that if a bunch of VaR's and ES's are needed, say for normally distributed variables, this can be computed very efficiently using this property.

3. VaR

Here we compute the VaR associated with a standard normal r.v. The three variants are equivalent since 0.5 and `"qf"` are default for the last two arguments.

```
> cvar::VaR(qnorm, x = 0.05, dist.type = "qf")

[1] 1.644854

> cvar::VaR(qnorm, x = 0.05)

[1] 1.644854

> cvar::VaR(qnorm)

[1] 1.644854
```

`x` can be a vector:

[1] Beware that a definition without negation is also used in both the literature and in the R packages.

[2] This equation shows why some authors use the "big numbers", e.g. 0.95 and 0.99, in the notation for the VaR.

```
> cvar::VaR(qnorm, x = c(0.01, 0.05))

[1] 2.326348 1.644854
```

Let's set some more realistic values for the parameters of the normal distribution:

```
> muA <- 0.006408553
> sigma2A <- 0.0004018977
```

This demonstrates the use of `intercept` and `slope`. These arguments use the linear transformation property discussed in the theoretical section. For the normal disribution the parameters are precisely the intercept and the slope:

```
> res1 <- cvar::VaR(qnorm, x = 0.05, mean = muA, sd = sqrt(sigma2A))
> res2 <- cvar::VaR(qnorm, x = 0.05, intercept = muA, slope = sqrt(sigma2A))
> abs((res2 - res1)) # 0, intercept/slope equivalent to mean/sd

[1] 0
```

If we compute VaR using the cdf, the intercept slope method has some numerical advantage:

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

[1] 3.287939e-09

> abs((res2a - res2)) # 5.331195e-11, intercept/slope better numerically

[1] 5.331195e-11
```

We can use smaller tollerance to improve the precision (the value `.Machine$double.eps^0.75` = 1.81898940354586e-12 is probably excessive):

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

[1] 6.938894e-18

> abs((res2b - res2)) # 1.040834e-16

[1] 1.040834e-16
```

The relative precision is also good.

```
> abs((res1b - res2)/res2) # 2.6119e-16 # both within machine precision

[1] 2.6119e-16

> abs((res2b - res2)/res2) # 3.91785e-15

[1] 3.91785e-15
```

For examples with vectorised distribution parameters we use data from **PerformanceAnalytics**.
Compute means and variances of the variables in a data frame and store them in vectors:

```
> ## if(require("PerformanceAnalytics")){
> data(edhec, package = "PerformanceAnalytics")
> mu <- apply(edhec, 2, mean)
> sigma2 <- apply(edhec, 2, var)
> musigma2 <- cbind(mu, sigma2)
```

We compute VaR using `PerformanceAnalytics::VaR`:

```
> ## 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 below compare to the value, `VPA`, obtained here.
The computations below are similar to the previous example but the distribution parameters are vectors. The first pair of results are numerically the same:

```
> vAz1 <- cvar::VaR(qnorm, x = 0.05, mean = mu, sd = sqrt(sigma2))
> vAz2 <- cvar::VaR(qnorm, x = 0.05, intercept = mu, slope = sqrt(sigma2))
> max(abs((vPA - vAz1))) # 5.551115e-17

[1] 5.551115e-17

> max(abs((vPA - vAz2))) #   ""

[1] 5.551115e-17
```

Computing VaR from cdf shows some advantage for the location-scale method:

```
> vAz1a <- cvar::VaR(pnorm, x = 0.05, dist.type = "cdf",
+                    mean = mu, sd = sqrt(sigma2))
> vAz2a <- cvar::VaR(pnorm, x = 0.05, dist.type = "cdf",
+                    intercept = mu, slope = sqrt(sigma2))
> max(abs((vPA - vAz1a))) # 3.287941e-09

[1] 3.287941e-09

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

[1] 1.465251e-10
```

The advantage remains for smaller tolerance:

```
> vAz1b <- cvar::VaR(pnorm, x = 0.05, dist.type = "cdf",
+                    mean = mu, sd = sqrt(sigma2),
+                    tol = .Machine$double.eps^0.75)
> vAz2b <- cvar::VaR(pnorm, x = 0.05, dist.type = "cdf",
+                    intercept = mu, slope = sqrt(sigma2),
+                    tol = .Machine$double.eps^0.75)
> max(abs((vPA - vAz1b))) # 4.374869e-13

[1] 4.374869e-13

> max(abs((vPA - vAz2b))) # 3.330669e-16

[1] 3.330669e-16
```

4. Expected shortfall

`ES` has essentially the same arguments as `VaR`. The examples below are obtained almost literally by replacing the calls to `VaR` with `ES()` ones.

Here we compute the VaR associated with a standard normal r.v. The three variants are equivalent since 0.5 and `"qf"` are default for the last two arguments.

```
> cvar::ES(qnorm, x = 0.05, dist.type = "qf")

[1] 2.062713

> cvar::ES(qnorm, x = 0.05)

[1] 2.062713

> cvar::ES(qnorm)

[1] 2.062713
```

`x` can be a vector:

```
> cvar::ES(qnorm, x = c(0.01, 0.05))

[1] 2.665214 2.062713
```

Let's set some more realistic values for the parameters of the normal distribution:

```
> muA <- 0.006408553
> sigma2A <- 0.0004018977
```

This demonstrates the use of `intercept` and `slope`. These arguments use the linear transformation property discussed in the theoretical section. For the normal disribution the parameters are precisely the intercept and the slope:

```
> res1 <- cvar::ES(qnorm, x = 0.05, mean = muA, sd = sqrt(sigma2A))
> res2 <- cvar::ES(qnorm, x = 0.05, intercept = muA, slope = sqrt(sigma2A))
> abs((res2 - res1))

[1] 9.714451e-17
```

If we compute ES using the cdf, the intercept slope method has some numerical advantage:

```
> ## with cdf the precision depends on solving an equation
> res1a <- cvar::ES(pnorm, x = 0.05, dist.type = "cdf",
+                   mean = muA, sd = sqrt(sigma2A))
> res2a <- cvar::ES(pnorm, x = 0.05, dist.type = "cdf",
+                   intercept = muA, slope = sqrt(sigma2A))
> abs((res1a - res2)) #

[1] 1.933215e-10

> abs((res2a - res2)) # intercept/slope better numerically

[1] 2.385314e-13
```

We can use smaller tollerance to improve the precision (the value `.Machine$double.eps^0.75` = 1.81898940354586e-12 is probably excessive):

```
> ## as above, but increase the precision, this is probably excessive
> res1b <- cvar::ES(pnorm, x = 0.05, dist.type = "cdf",
+                   mean = muA, sd = sqrt(sigma2A), tol = .Machine$double.eps^0.75)
> res2b <- cvar::ES(pnorm, x = 0.05, dist.type = "cdf",
+                   intercept = muA, slope = sqrt(sigma2A), tol = .Machine$double.eps^0.75)
> abs((res1b - res2))

[1] 3.958639e-14

> abs((res2b - res2))

[1] 6.938894e-16
```

The relative precision is also good.

```
> abs((res1b - res2)/res2)

[1] 1.13287e-12

> abs((res2b - res2)/res2)

[1] 1.98575e-14
```

Affiliation:

Georgi N. Boshnakov
School of Mathematics
The University of Manchester
Oxford Road, Manchester M13 9PL, UK
URL: http://www.maths.manchester.ac.uk/~gb/