Financial Risk Management with R

Adair Neto

July 7, 2023

Retrieving FRED Data

Which risk factors are important?

What is the distribution of risk factor returns? Is it normal? If not, does it have heavy tails?

How predictable are risk factor returns? Serial correlation? Volatility clustering?

Important admin: add the command `RNGkind(sample.kind="Rounding")` before any instance of the `set.seed()`.

```
library(quantmod)
```

```
## Carregando pacotes exigidos: xts
```

```
## Carregando pacotes exigidos: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Carregando pacotes exigidos: TTR
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
wilsh <- getSymbols("WILL5000IND", src="FRED", auto.assign=FALSE)
wilsh <- na.omit(wilsh)
wilsh <- wilsh["1979-12-31/2017-12-31"]
names(wilsh) <- "TR"

head(wilsh, 3)
```

```
##              TR
## 1979-12-31 1.90
## 1980-01-02 1.86
## 1980-01-04 1.88
```

```
tail(wilsh, 3)
```

```
##                TR
## 2017-12-27 124.04
## 2017-12-28 124.33
## 2017-12-29 123.67
```

To compute daily returns, a natural idea is to define a **discrete return**:

$$\text{ret}_t = \frac{\text{wilsh}_t}{\text{wilsh}_{t-1}} - 1$$

However, this is not symmetric. Thus, we use the **log return** (also known as **continuously compound return**):

$$\text{logret}_t = \log(1 + \text{ret}_t)$$

It can be shown that

$$\text{logret}_t = \log(\text{wilsh}_t) - \log(\text{wilsh}_{t-1})$$

and

$$\text{ret}_t = \exp(\text{logret}_t) - 1$$

```
logret <- diff(log(wilsh$TR))[-1]
round(head(logret,3),6)
```

```
##                    TR
## 1980-01-02 -0.021277
## 1980-01-04  0.010695
## 1980-01-07  0.005305
```

```
ret <- exp(logret)-1
round(head(ret,3),6)
```

```
##                    TR
## 1980-01-02 -0.021053
## 1980-01-04  0.010753
## 1980-01-07  0.005319
```

For longer periods of time, e.g. n-days, we compute $\text{logret}_t + \cdots + \text{logret}_{t-n+1}$ for the log return and $\exp(\text{logret}_t + \cdots + \text{logret}_{t-n+1}) - 1$ for the discrete return.

In R, we use the function `apply.weekly` to compute weekly log return.

```
logret_w <- apply.weekly(logret, sum)
round(head(logret_w,3), 6)
```

```
##                    TR
## 1980-01-04 -0.010582
## 1980-01-11  0.036558
## 1980-01-18  0.010204
```

```
ret_w <- exp(logret_w) - 1
```

Also available: `apply.monthly`, `apply.quarterly`, and `apply.yearly`.

Risk Management Under Normal Distributions

Estimating μ and σ

Assume that logret $\sim N(\mu, \sigma)$.

We estimate μ using the sample mean and σ using the sample standard deviation.

```
mu <- mean(logret)
sigma <- sd(logret)
```

Value-at-Risk (VaR)

Definition. (Value-at-risk) The VaR is the amount that a portfolio might lose, with a given probability $(1-\alpha)$ (confidence level), over a given time period.

The VaR is the alpha quantile of the pdf.

```
alpha <- 0.05
var <- qnorm(alpha, mu, sigma)
```

Example. Suppose that a hedge fund is investing $1000 million in US equities. The VaR of the daily change in its assets, with 95% confidence level, is given by

```
HFvar <- 1000 * (exp(var) - 1)
```

Expected Shortfall (ES)

Definition. (Expected Shortfall) ES is the expected return given that the return is worse than the associated VaR.

The average loss is:

```
es <- mu - sigma * dnorm(qnorm(alpha, 0, 1), 0, 1)/alpha
```

In the previous example,

```
HFes <- 1000 * (exp(es) - 1)
round(HFes, 1)
```

```
## [1] -21.4
```

Using Simulation to Estimate VaR and ES

We'll simulate some data, and take the α-quantile of the simulated data.

One way of doing that is by drawing 100,000 outcomes from $N(\mu, \sigma)$ distribution. Here, we assume normality.

```
RNGkind(sample.kind="Rounding")
```

```
## Warning in RNGkind(sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
set.seed(123789)

rvec <- rnorm(100000, mu, sigma)
VaR <- quantile(rvec, alpha)
ES <- mean(rvec[rvec < VaR])

round(VaR, 6)
```

```
##       5%
## -0.01729
```

```
round(ES, 6)
```

```
## [1] -0.021892
```

Another way: draw 100,000 outcomes (with replacement) from the vector of daily log returns. Here, we do not assume normality.

```
RNGkind(sample.kind="Rounding")
```

```
## Warning in RNGkind(sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
set.seed(123789)

rvec <- sample(as.vector(logret), 100000, replace = TRUE)
VaR <- quantile(rvec, alpha)
ES <- mean(rvec[rvec < VaR])

round(VaR, 6)
```

```
##        5%
## -0.015862
```

```
round(ES, 6)
```

```
## [1] -0.025279
```

Possible problem: the data is not normal.

Risk Management Under Non-normal Distributions

Skewness and Kurtosis

Most common departures from normality:

1. **Skewness:** one tail is longer than the other.
2. **Kurtosis:** tails are heavier or thinner than the normal.

```
# install.packages("moments")
library(moments)
rvec <- as.vector(logret)
round(skewness(rvec), 2)
```

```
## [1] -0.91
```

If the coefficient of skewness is negative, it is left-skewed.

Heavy-tailed distributions are called **leptokurtic** and thin-tailed are called **platykurtic**.

Coefficient of kurtosis:

1. 3 for normal.
2. < 3 for thin-tailed.
3. > 3 for heavy-tailed.

```
round(kurtosis(rvec), 2)
```

```
## [1] 21.8
```

The Jarque-Bera test for normality

```
jarque.test(rvec)
```

```
##
##  Jarque-Bera Normality Test
##
## data:  rvec
## JB = 142451, p-value < 2.2e-16
## alternative hypothesis: greater
```

Thus, reject normality.

Other tests: QQPlot and Kolmogorov-Smirnov.

Student-t Distribution

Has one parameter: ν degrees of freedom.

Mean: 0.

Variance:

$$\frac{\nu}{\nu - 2}$$

for $\nu > 2$ and ∞ for $1 < \nu \leq 2$, otherwise undefined.

Skewness: 0 for $\nu > 3$, otherwise undefined.

Kurtosis:

$$3 + \frac{6}{\nu - 4}$$

for $\nu > 4$, ∞ for $2 < \nu \leq 4$, otherwise undefined.

Has heavier tails than the normal. When $\nu \to \infty$, the distribution converges to $N(0, 1)$.

We use the standard student-t distribution with standard deviation 1.

Assume that logret $= \mu + \sigma\varepsilon$, in which ε is a rescaled student-t distribution with dof ν. The mean of logret is μ and the standard deviation is σ.

We'll use the Maximum Likelihood Estimation (MLE) to estimate (μ, σ, ν). For that, we use `fitdistr` function in the "MASS" package.

```
#install.packages("MASS")
library(MASS)
t.fit <- fitdistr(rvec, "t")
```

```
## Warning in dt((x - m)/s, df, log = TRUE): NaNs produzidos

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produzidos

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produzidos

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produzidos

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produzidos

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produzidos

## Warning in log(s): NaNs produzidos

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produzidos

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produzidos

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produzidos

## Warning in log(s): NaNs produzidos

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produzidos
```

```
## Warning in log(s): NaNs produzidos

## Warning in log(s): NaNs produzidos

## Warning in log(s): NaNs produzidos

## Warning in log(s): NaNs produzidos

## Warning in log(s): NaNs produzidos

## Warning in log(s): NaNs produzidos

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produzidos

## Warning in log(s): NaNs produzidos

## Warning in dt((x - m)/s, df, log = TRUE): NaNs produzidos
```

```
round(t.fit$estimate, 6)
```

```
##        m        s       df
## 0.000754 0.006643 3.045229
```

Simulation

To estimate VaR and ES for student-t, we use simulation like in the normal case.

```
alpha <- 0.05
RNGkind(sample.kind="Rounding")
```

```
## Warning in RNGkind(sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
set.seed(123789)
library(metRology)
```

```
##
## Attaching package: 'metRology'

## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```
rvec <- rt.scaled(100000, mean = t.fit$estimate[1], sd = t.fit$estimate[2], df = t.fit$estimate[3])
VaR <- quantile(rvec, alpha)
ES <- mean(rvec[rvec < VaR])

round(VaR, 6)
```

```
##        5%
## -0.014706
```

```
round(ES, 6)
```

```
## [1] -0.0243
```

Multi-day Horizon

We'll compute VaR and ES for a 10-day horizon in 3 ways.

1. Simulate from the estimated student-t distribution.

```
alpha <- 0.05
RNGkind(sample.kind="Rounding")
```

```
## Warning in RNGkind(sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
set.seed(123789)
library(metRology)

rvec <- rep(0, 100000)
for (i in 1:10) {
    rvec <- rvec + rt.scaled(100000, mean = t.fit$estimate[1], sd = t.fit$estimate[2], df = t.fit$estima
}
# VaR <- quantile(rvec, alpha)
ES <- mean(rvec[rvec < VaR])

round(VaR, 6)
```

```
##        5%
## -0.014706
```

```
round(ES, 6)
```

```
## [1] -0.036801
```

2. Simulate from the empirical distribution with i.i.d. draws.

```
alpha <- 0.05
RNGkind(sample.kind="Rounding")
```

```
## Warning in RNGkind(sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
set.seed(123789)
library(metRology)

rvec <- rep(0, 100000)
for (i in 1:10) {
    rvec <- rvec + sample(as.vector(logret), 100000, replace = TRUE)
}
VaR <- quantile(rvec, alpha)
ES <- mean(rvec[rvec < VaR])

round(VaR, 6)
```

```
##        5%
## -0.050808
```

```
round(ES, 6)
```

```
## [1] -0.071943
```

3. Simulate from the empirical distribution with block draws.

```
alpha <- 0.05
RNGkind(sample.kind="Rounding")
```

```
## Warning in RNGkind(sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```r
set.seed(123789)
library(metRology)

rvec <- rep(0, 100000)
rdat <- as.vector(logret)
posn <- seq(from = 1, to = length(rdat) - 9, by = 1)
rpos <- sample(posn, 100000, replace = TRUE)
for (i in 1:10) {
    rvec <- rvec + rdat[rpos]
    rpos <- rpos + 1
}
VaR <- quantile(rvec, alpha)
ES <- mean(rvec[rvec < VaR])

round(VaR, 6)
```

```
##        5%
## -0.047006
```

```r
round(ES, 6)
```

```
## [1] -0.077524
```

Risk Management under Volatility Clustering

Serial Correlation, Volatility Clustering, and GARCH

Assumption 1: the future distribution of the log returns is the same as its historical distribution.

Assumption 2: the parameters of the historical distribution are estimated without paying attention to the ordering of the data, i.e., the ordering of the data is not important.

To test if the ordering of the data is important, we'll use some tests. The first is serial correlation.

Serial correlation

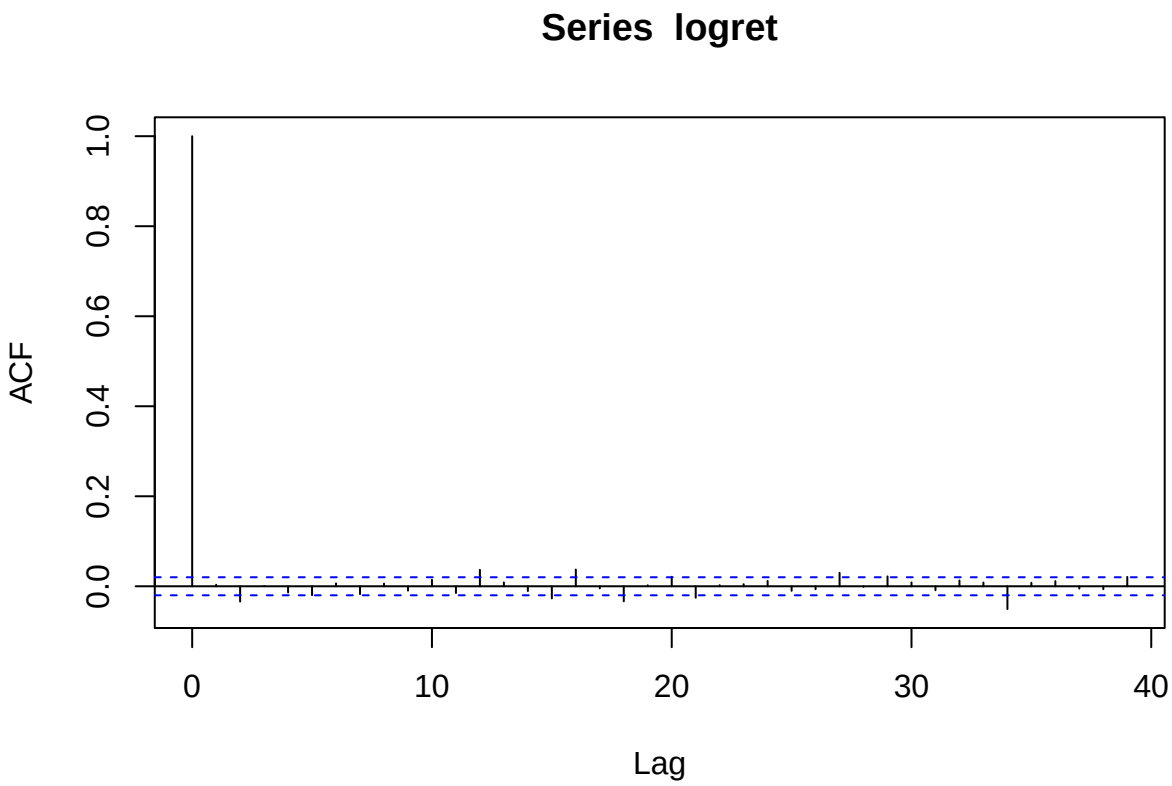
Let x_t be the value of a time series on day t.

The **autocorrelation coefficient** at lag j is given by

$$\rho_j = \text{cor}(x_t, x_{t-j})$$

And the **autocorrelation coefficient function (ACF)** is the graph of ρ_j for $j = 0, 1, \ldots$. Notice that $\rho_0 = 1$.

```r
acf(logret)
```



If most values are outside the dash lines (95% confidence level), then there's evidence of strong correlation.

Volatility Clustering

High (low) volatility tends to be followed by high (low) volatility.

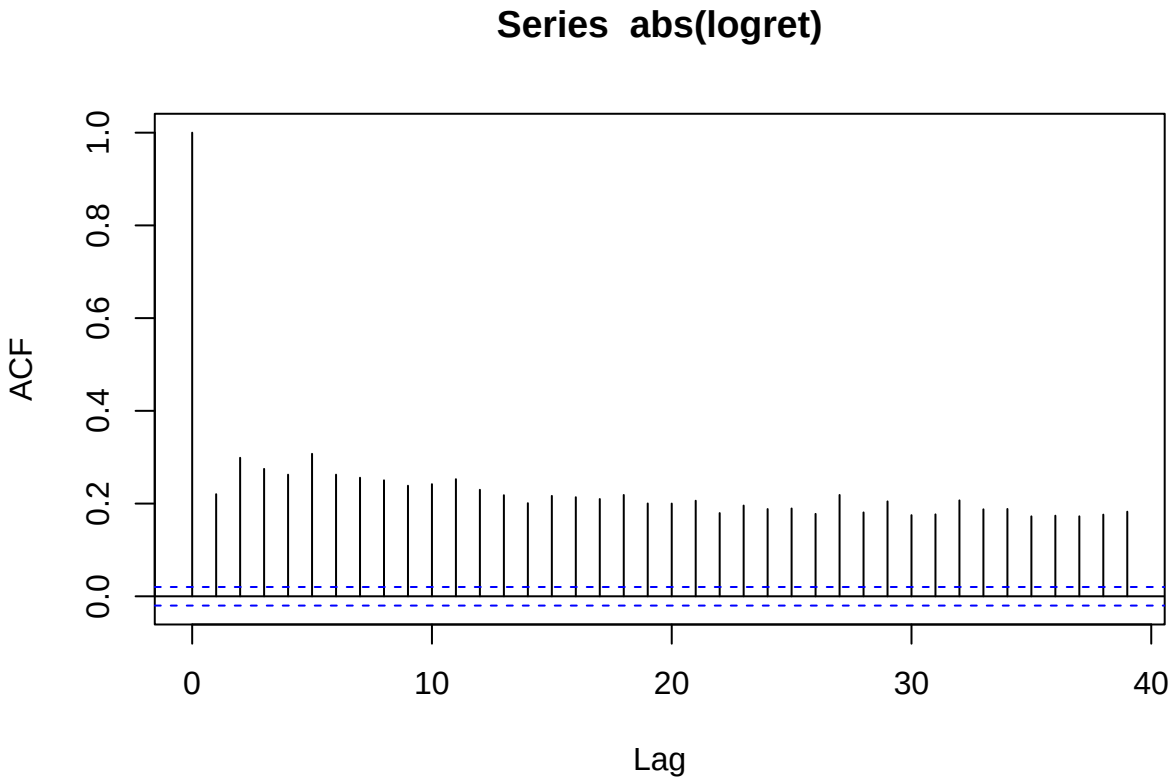
Here we use x_t as before and define the autocorrelation coefficient at lag j by

$$\rho_{|j|} = \text{cor}(|x_t|, |x_{t-j}|)$$

knitr::opts_chunk$set(eval=FALSE)

Then apply `acf()` to the absolute values.

```r
acf(abs(logret))
```



GARCH - A volatility prediction model

The GARCH(1,1) normal model:

- Mean equation: $r_t = a_0 + \sqrt{h_t}\varepsilon_t$.
- Variance equation: $h_t = \alpha_0 + \beta_1 h_{t-1} + \alpha_1 \varepsilon_{t-1}^2$.
- Distribution equation: $\varepsilon_t \sim N(0,1)$.

Where r_t is the return series with time varying volatility, a_0 is its expected return (typically close to 0), $\sqrt{h_t}\varepsilon_t$ is the unexpected return, with h_t the predictable variance.

If we have constant variance, with $\beta_1 = 0 = \alpha_1$, then the model gives $H_t = \alpha_0$, which is the normal model of log returns: $r_t = \mu + \sigma\varepsilon_t$.

```
library(rugarch)
```

```
## Carregando pacotes exigidos: parallel
```

```
##
## Attaching package: 'rugarch'
```

```
## The following object is masked from 'package:stats':
##
##     sigma
```

```
garch.N <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1,1)),
                      mean.model = list(armaOrder = c(0,0), include.mean = TRUE),
                      distribution.model = "norm")
fit.garch.N <- ugarchfit(spec = garch.N, data = logret)
fit.garch.N
```

```
##
## *---------------------------------*
```

```
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## ----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(0,0,0)
## Distribution : norm
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.000677    0.000079   8.5625 0.000000
## omega   0.000002    0.000001   2.9210 0.003489
## alpha1  0.090826    0.007896  11.5032 0.000000
## beta1   0.893296    0.008515 104.9037 0.000000
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.000677    0.000085  7.92872  0.00000
## omega   0.000002    0.000006  0.30702  0.75883
## alpha1  0.090826    0.059850  1.51755  0.12913
## beta1   0.893296    0.070415 12.68623  0.00000
##
## LogLikelihood : 31767.82
##
## Information Criteria
## ------------------------------------
##
## Akaike       -6.6299
## Bayes        -6.6269
## Shibata      -6.6299
## Hannan-Quinn -6.6289
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic   p-value
## Lag[1]                      24.71 6.673e-07
## Lag[2*(p+q)+(p+q)-1][2]     24.71 2.588e-07
## Lag[4*(p+q)+(p+q)-1][5]     27.61 1.491e-07
## d.o.f=0
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                     0.3118  0.5766
## Lag[2*(p+q)+(p+q)-1][5]    2.8937  0.4268
## Lag[4*(p+q)+(p+q)-1][9]    3.8085  0.6214
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
```

```
## ARCH Lag[3]   0.01177 0.500 2.000  0.9136
## ARCH Lag[5]   0.01910 1.440 1.667  0.9988
## ARCH Lag[7]   0.75552 2.315 1.543  0.9498
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  178.0231
## Individual Statistics:
## mu      0.07094
## omega  22.32967
## alpha1  0.12665
## beta1   0.18631
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.07 1.24 1.6
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value      prob sig
## Sign Bias            2.191 2.849e-02  **
## Negative Sign Bias   2.018 4.364e-02  **
## Positive Sign Bias   2.793 5.229e-03 ***
## Joint Effect        48.333 1.809e-10 ***
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     478.9    1.513e-89
## 2    30     419.2    9.396e-71
## 3    40     815.3   6.631e-146
## 4    50     790.9   1.655e-134
##
##
## Elapsed time : 0.5066292
```

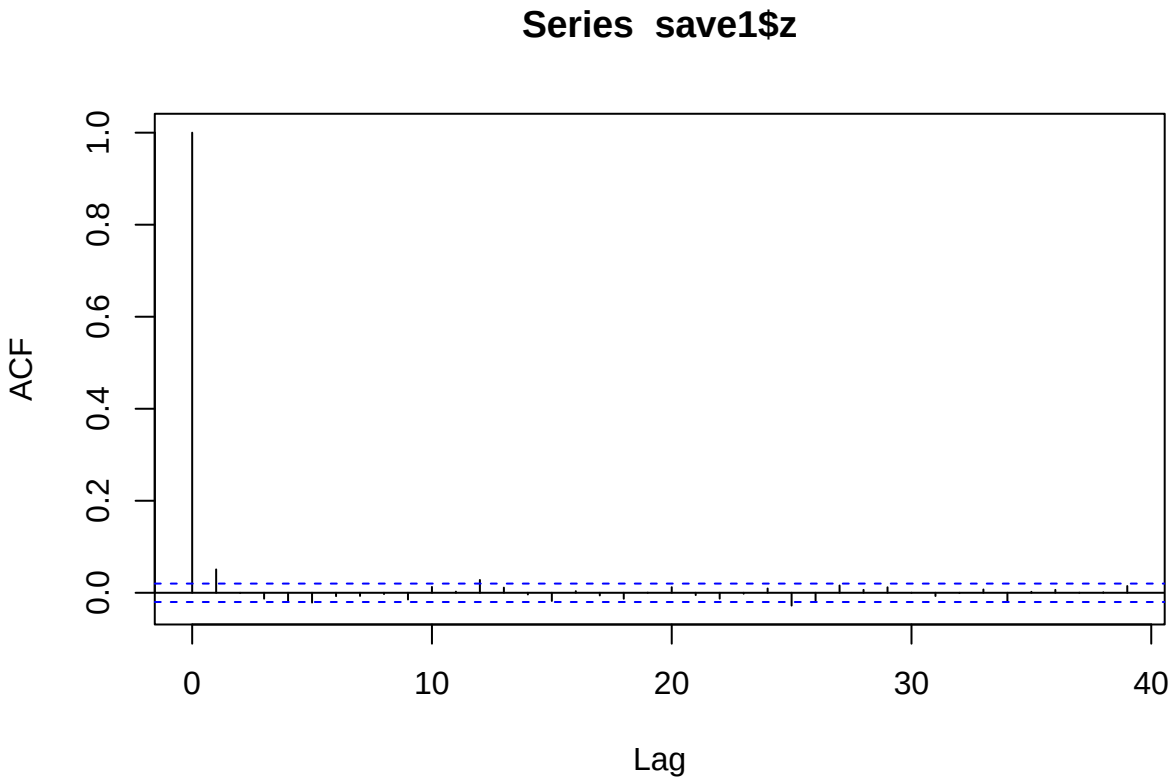
Save fitted values:

```
save1 <- cbind(logret, fit.garch.N@fit$sigma, fit.garch.N@fit$z)
names(save1) <- c("logret", "s", "z")
```

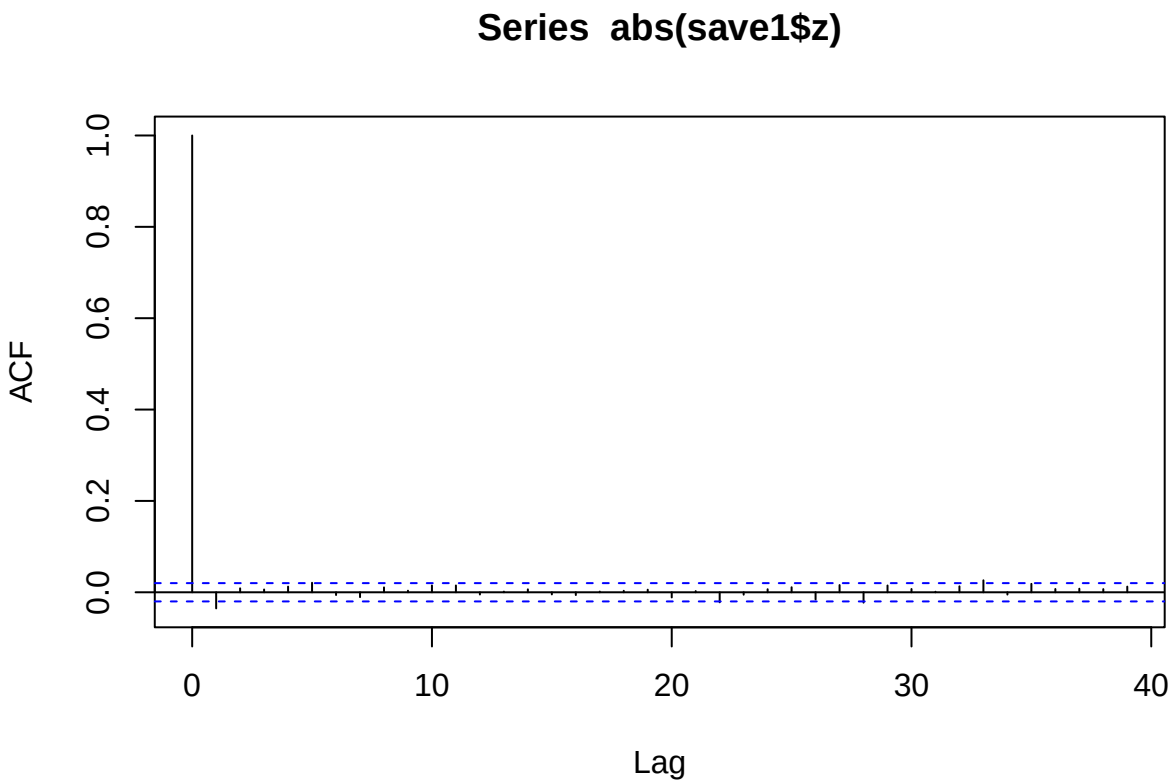
Diagnostic test: the z_t (fitted value for ε_t) must be normal. Compare mean, sd, skewness, and kurtosis. The Jarque-Bera test can also be used.

Also check the serial correlation.

```
acf(save1$z)
```




```
acf(abs(save1$z))
```



Now, we change the distribution equation from the normal to the rescaled student-t distribution. Thus, the

distribution model becomes

$$\varepsilon_t \sim \frac{t(\nu)}{\sqrt{\frac{\nu}{\nu-2}}}$$

```
garch.t <- ugarchspec(variance.model = list(model = "sGARCH", garchOrder = c(1,1)),
                      mean.model = list(armaOrder = c(0,0), include.mean = TRUE),
                      distribution.model = "std")
fit.garch.t <- ugarchfit(spec = garch.t, data = logret)
fit.garch.t
```

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(0,0,0)
## Distribution : std
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.000792    0.000073   10.916  0.0e+00
## omega   0.000001    0.000000    4.156  3.2e-05
## alpha1  0.071230    0.003849   18.504  0.0e+00
## beta1   0.922833    0.003776  244.426  0.0e+00
## shape   6.143072    0.371702   16.527  0.0e+00
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.000792    0.000072  10.9883 0.000000
## omega   0.000001    0.000000   1.6434 0.100305
## alpha1  0.071230    0.016979   4.1951 0.000027
## beta1   0.922833    0.015966  57.8011 0.000000
## shape   6.143072    0.478406  12.8407 0.000000
##
## LogLikelihood : 32054.33
##
## Information Criteria
## ------------------------------------
##
## Akaike       -6.6895
## Bayes        -6.6857
## Shibata      -6.6895
## Hannan-Quinn -6.6882
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic   p-value
## Lag[1]                      26.41 2.767e-07
## Lag[2*(p+q)+(p+q)-1][2]     26.41 9.278e-08
## Lag[4*(p+q)+(p+q)-1][5]     29.35 4.820e-08
```

```
## d.o.f=0
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                      3.467 0.06260
## Lag[2*(p+q)+(p+q)-1][5]     9.208 0.01466
## Lag[4*(p+q)+(p+q)-1][9]    10.368 0.04181
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]    0.4462 0.500 2.000  0.5042
## ARCH Lag[5]    0.5127 1.440 1.667  0.8799
## ARCH Lag[7]    1.0014 2.315 1.543  0.9133
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  1034.21
## Individual Statistics:
## mu       0.06115
## omega  208.87381
## alpha1   0.48179
## beta1    0.38116
## shape    0.62958
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.28 1.47 1.88
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value      prob sig
## Sign Bias            1.939 5.251e-02   *
## Negative Sign Bias   2.651 8.038e-03 ***
## Positive Sign Bias   2.496 1.259e-02  **
## Joint Effect        47.842 2.301e-10 ***
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     174.7    3.475e-27
## 2    30     397.1    2.939e-66
## 3    40     370.0    1.623e-55
## 4    50     432.6    7.677e-63
##
##
## Elapsed time : 0.5673711
```

```
save1 <- cbind(logret, fit.garch.t@fit$sigma, fit.garch.t@fit$z)
names(save1) <- c("logret", "s", "z")
```

```
parm1 <- fit.garch.t@fit$coef
```

VaR and ES for GARCH Bootstrap

The ugarchboot Function

```
RNGkind(sample.kind="Rounding")
```

```
## Warning in RNGkind(sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
set.seed(123789)

boot.garch <- ugarchboot(fit.garch.t,
              method = "Partial",
              sampling = "raw",
              n.ahead = 1,
              n.bootpred = 100000,
              solver = "solnp")

rvec <- boot.garch@fseries
alpha <- 0.05
VaR <- quantile(rvec, alpha)
ES <- mean(rvec[rvec < VaR])
```

The ugarchrool Function

```
library(bigrquery)
n2016 <- length(logret["1980-01-01/2016-12-31"])
roll.garch <- ugarchroll(spec = garch.t,
              data = logret,
              n.ahead = 1,
              forecast.length = 1,
              n.start = n2016,
              refit.every = 1,
              refit.window = "recursive",
              calculate.VaR = TRUE,
              VaR.alpha = 0.05,
              keep.coef = TRUE)
```