GARCH parameters and quantiles estimation

Jose Augusto Fiorucci

20/11/2020

Input

```
symbol = "EMBR3.SA"#"BOVA11.SA"#
from=as.Date('2000-01-01')#2012
to=as.Date('2017-12-31')#'2018-12-31'
C_Trend = 0.95
C_Reaction = 0.50
```

Data download

```
getSymbols.yahoo(symbol, from=from, to=to, env=globalenv())
```

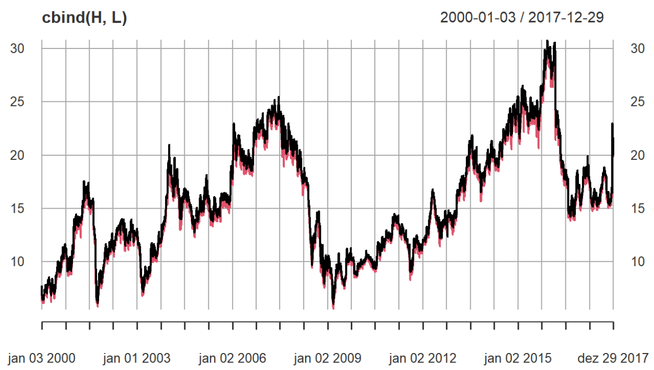
```
## Warning: EMBR3.SA contains missing values. Some functions will not work if
## objects contain missing values in the middle of the series. Consider using
## na.omit(), na.approx(), na.fill(), etc to remove or replace them.
```

```
## [1] "EMBR3.SA"
```

```
x <- get(symbol, envir=globalenv())
rm(list = symbol, envir=globalenv())
```

High and Low

```
H <- Hi(x)
L <- Lo(x)
plot(cbind(H,L))
```



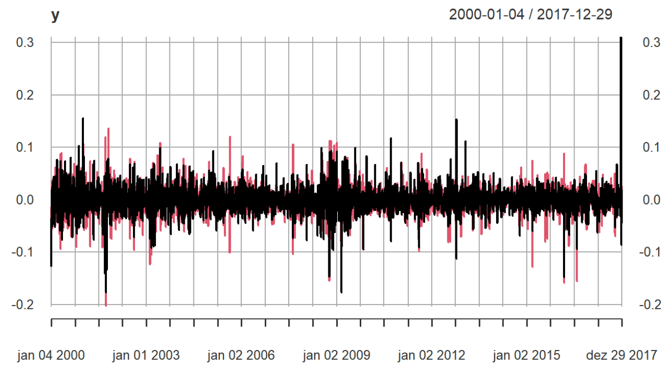
Loading [MathJax]/jax/output/HTML-CSS/jax.js

Returns

```
y <- cbind( diff(log(H)),  diff(log(L)) )
y <- na.omit(y)
y %>% cor() # Returns correlation
```

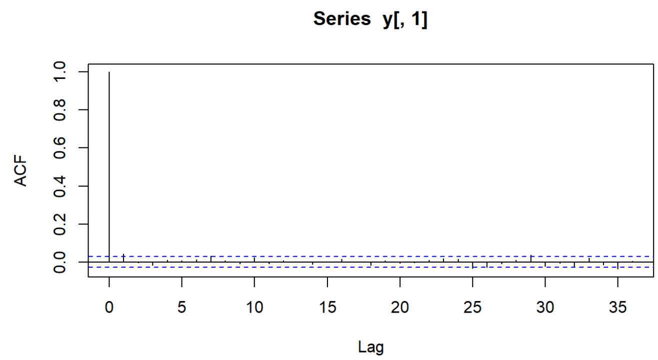
```
##                EMBR3.SA.High EMBR3.SA.Low
## EMBR3.SA.High      1.0000000    0.5421755
## EMBR3.SA.Low       0.5421755    1.0000000
```

```
plot(y)
```

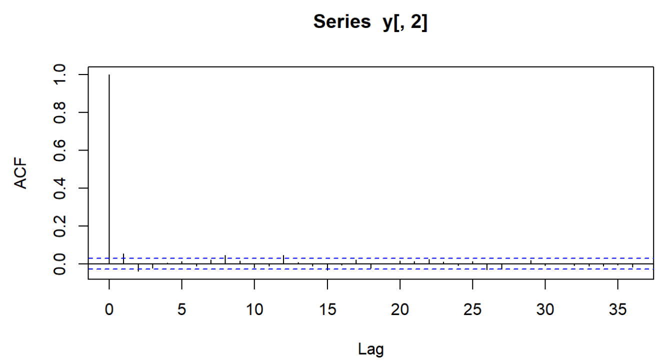


Autocorrelation

```
acf(y[,1])
```

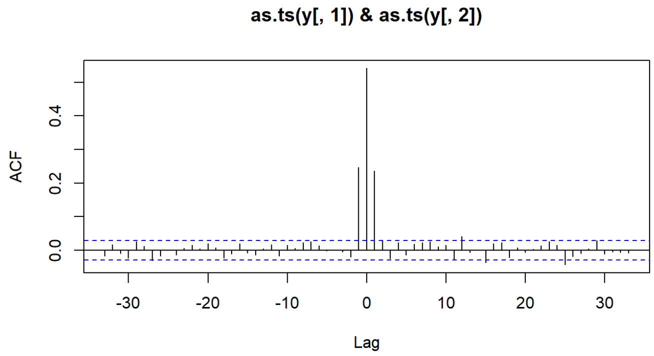



```
acf(y[,2])
```



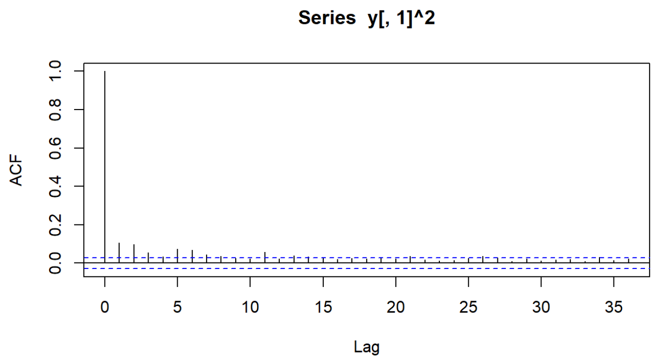
Cross correlation

```
ccf(as.ts(y[,1]),as.ts(y[,2]))
```

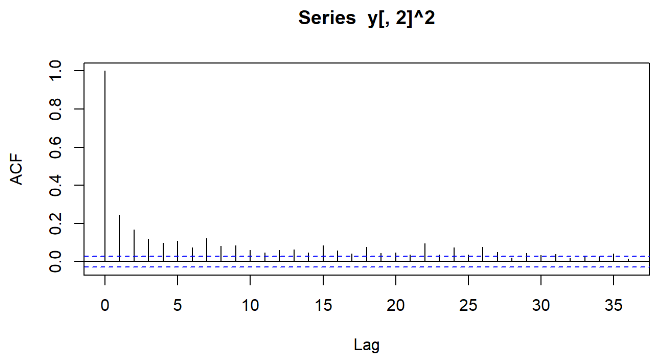


Volatility verification

```
acf(y[,1]^2)
```




```
acf(y[,2]^2)
```



Bivariate DCC-GARCH

We will consider the DCC-GARCH to model the volatility of $y = (r_H, r_L)'$, where r_H and r_L denote the $100 \times$ log-returns from hight's and low's observations.

Loading [MathJax]/jax/output/HTML-CSS/jax.js

```
# returns
mY <- 100*y

# generates the Markov Chain
start <- Sys.time()

out <- bayesDccGarch(mY, control=list(print=FALSE))
```

```
## Maximizing the log-posterior density function.
## Done.
## One approximation for covariance matrix of parameters cannot be directly computed through the hessian matrix.
## Calibrating the standard deviations for simulation:
## Accept Rate:
##  phi_1  phi_2  phi_3  phi_4  phi_5  phi_6  phi_7  phi_8  phi_9 phi_10 phi_11
##   0.30   0.10   0.16   0.15   0.15   0.11   0.18   0.11   0.14   0.30   0.53
## Accept Rate:
##  phi_1  phi_2  phi_3  phi_4  phi_5  phi_6  phi_7  phi_8  phi_9 phi_10 phi_11
##   0.31   0.18   0.15   0.22   0.25   0.17   0.19   0.20   0.25   0.31   0.48
## Accept Rate:
##  phi_1  phi_2  phi_3  phi_4  phi_5  phi_6  phi_7  phi_8  phi_9 phi_10 phi_11
##   0.27   0.16   0.27   0.21   0.24   0.15   0.17   0.19   0.23   0.31   0.44
## Accept Rate:
##  phi_1  phi_2  phi_3  phi_4  phi_5  phi_6  phi_7  phi_8  phi_9 phi_10 phi_11
##   0.30   0.17   0.28   0.21   0.25   0.26   0.19   0.18   0.22   0.32   0.45
## Computing the covariance matrix of pilot sample.
```

```
## Warning in if (class(control$cholCov) != "try-error") {: a condição tem
## comprimento > 1 e somente o primeiro elemento será usado
```

```
## Done.
## Calibrating the Lambda coefficient:
## lambda: 0.4
## Accept Rate: 0.37
## Done.
## Starting the simulation by one-block random walk Metropolis-Hasting algorithm.
## Done.
```

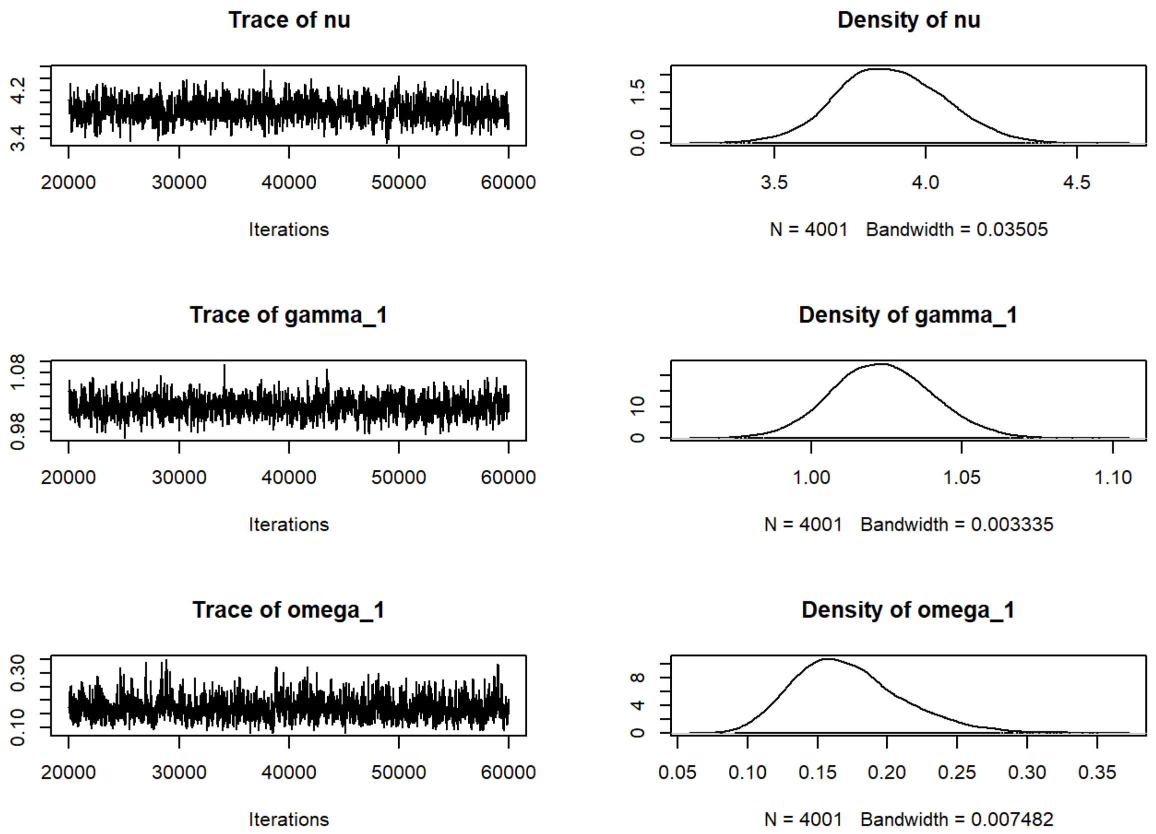
```
out2 <- increaseSim(out, nSim=50000)
```

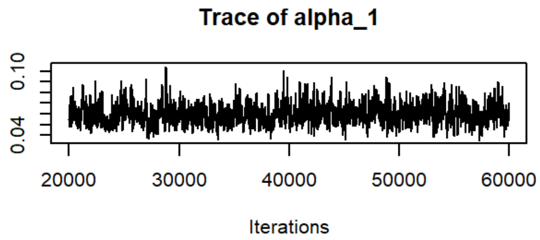
```
## Calibrating the Lambda coefficient:
## lambda: 0.4
## Accept Rate: 0.37
## Done.
## Starting the simulation by one-block random walk Metropolis-Hasting algorithm.
## Done.
```

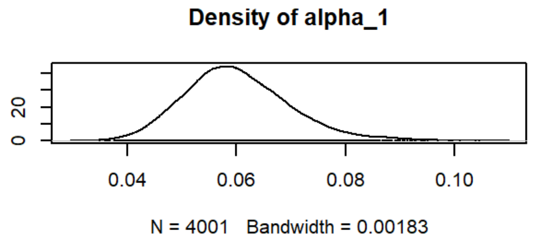
```
out <- window(out2, start=20000, thin=10)
rm(out2)

end <- Sys.time()

# elapsed time
end-start
```

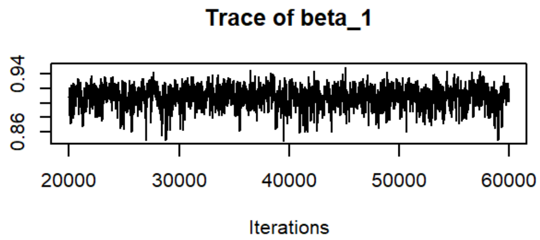
```
## Time difference of 5.882798 mins
```

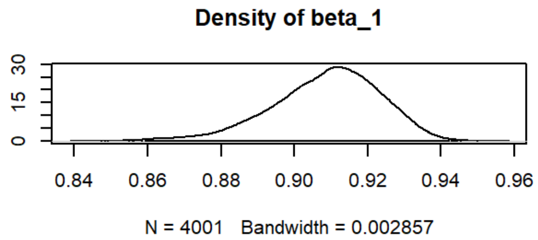
```
# plot Markov Chain
plot(out$MC)
```

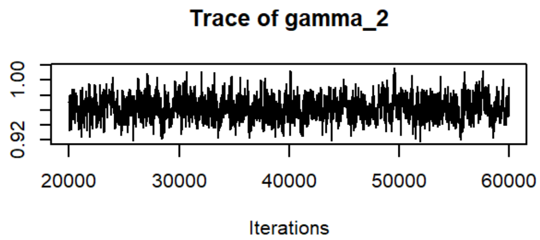


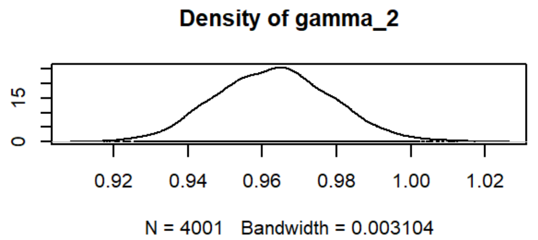


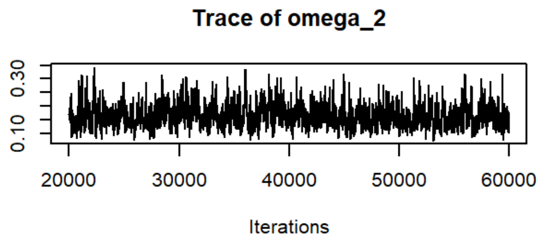


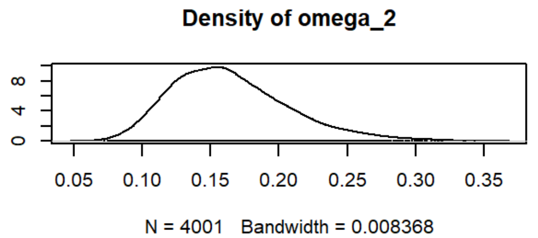


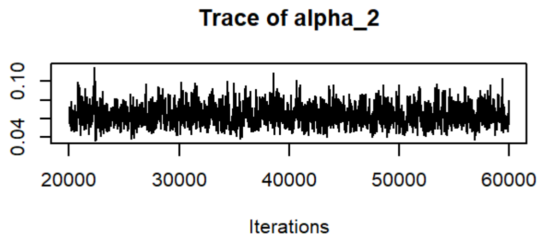


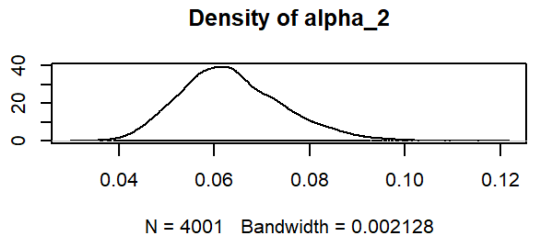


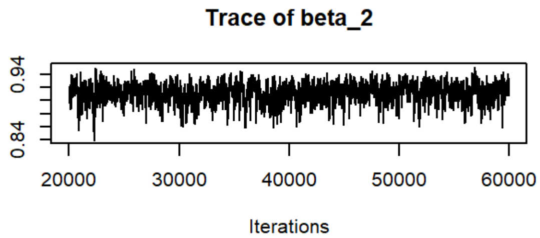


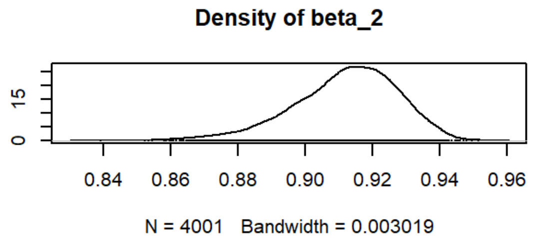




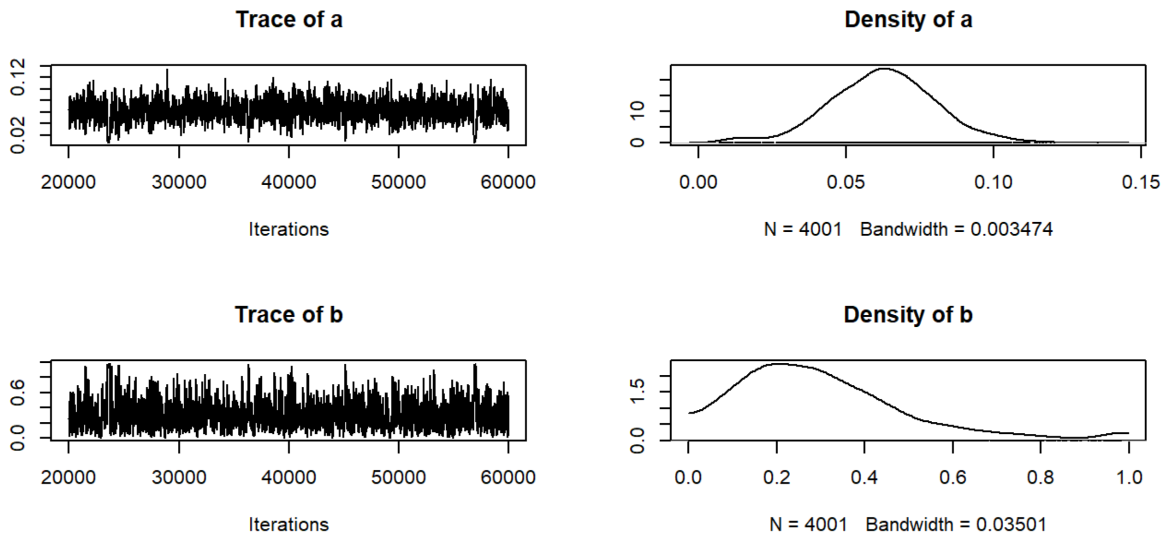








Loading [MathJax]/jax/output/HTML-CSS/jax.js




```
## Estimative of parameters
out$MC %>% summary()
```

```
##
## Iterations = 20000:60000
## Thinning interval = 10
## Number of chains = 1
## Sample size per chain = 4001
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean       SD  Naive SE Time-series SE
## nu      3.88217 0.173700 0.0027461      0.0066025
## gamma_1 1.02344 0.016526 0.0002613      0.0006235
## omega_1 0.17214 0.038975 0.0006162      0.0015216
## alpha_1 0.06011 0.009579 0.0001514      0.0004151
## beta_1  0.90862 0.014755 0.0002333      0.0005825
## gamma_2 0.96317 0.015382 0.0002432      0.0006211
## omega_2 0.16444 0.042417 0.0006706      0.0018995
## alpha_2 0.06364 0.010660 0.0001685      0.0003755
## beta_2  0.91187 0.015426 0.0002439      0.0006420
## a       0.06220 0.017827 0.0002818      0.0006096
## b       0.30436 0.191948 0.0030346      0.0082975
##
## 2. Quantiles for each variable:
##
##            2.5%     25%     50%     75%   97.5%
## nu      3.55174 3.76201 3.87631 4.00067 4.22594
## gamma_1 0.99207 1.01212 1.02316 1.03443 1.05709
## omega_1 0.10872 0.14467 0.16766 0.19436 0.26025
## alpha_1 0.04359 0.05369 0.05926 0.06584 0.08141
## beta_1  0.87508 0.89999 0.91008 0.91897 0.93338
## gamma_2 0.93421 0.95253 0.96315 0.97334 0.99388
## omega_2 0.09796 0.13344 0.15903 0.18901 0.26395
## alpha_2 0.04554 0.05622 0.06271 0.07036 0.08688
## beta_2  0.87657 0.90271 0.91346 0.92276 0.93763
## a       0.02456 0.05055 0.06242 0.07362 0.09801
## b       0.03358 0.16596 0.27099 0.39848 0.81435
```

```
# Prepare input for the expert advisor
parEst <- summary(out)$statistics[,'Mean']

## High
#HBOP
High_UB_HBOP = qsstd(p=1-(1-C_Trend)/2, mean = 0, sd = 1, nu = parEst['nu'], xi = parEst['gam
ma_1'])
#S1
High_UB_S1 = qsstd(p=1-(1-C_Reaction)/2, mean = 0, sd = 1, nu = parEst['nu'], xi = parEst['ga
mma_1'])


## Low
#B1
Low_LB_B1 = qsstd(p=(1-C_Reaction)/2, mean = 0, sd = 1, nu = parEst['nu'], xi = parEst['gamma
_2'])
#LBOP
Low_LB_LBOP = qsstd(p=(1-C_Trend)/2, mean = 0, sd = 1, nu = parEst['nu'], xi = parEst['gamma_
2'])


m = matrix(NA,nrow=10,ncol=1)
rownames(m) = c("High_UB_HBOP","High_UB_S1","Low_LB_B1","Low_LB_LBOP",
               "High_omega", "High_alpha","High_beta",
                     "Low_omega",  "Low_alpha", "Low_beta" )
colnames(m) = 'Value'

m["High_UB_HBOP",1] = High_UB_HBOP
m["High_UB_S1",1] = High_UB_S1
m["Low_LB_B1",1] =  Low_LB_B1
m["Low_LB_LBOP",1] = Low_LB_LBOP

m["High_omega",1] = parEst["omega_1"]
m["High_alpha",1] = parEst["alpha_1"]
m["High_beta",1] = parEst["beta_1"]

m["Low_omega",1] = parEst["omega_2"]
m["Low_alpha",1] = parEst["alpha_2"]
m["Low_beta",1] = parEst["beta_2"]

# Input for expert advisor
print(round(m,3))
```

```
##                Value
## High_UB_HBOP  1.985
## High_UB_S1    0.512
## Low_LB_B1    -0.508
## Low_LB_LBOP  -2.003
## High_omega    0.172
## High_alpha    0.060
## High_beta     0.909
## Low_omega     0.164
## Low_alpha     0.064
## Low_beta      0.912
```

Loading [MathJax]/jax/output/HTML-CSS/jax.js