GARCH parameters and quantiles estimation

Jose Augusto Fiorucci

20/11/2020

Input

```
symbol = "BRL=X"#"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: BRL=X 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] "BRL=X"
```

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

High and Low

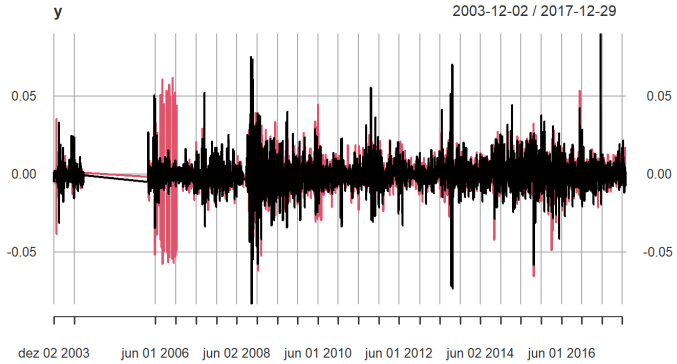
```
H <- Hi(x)
L <- Lo(x)
plot(cbind(H,L))
```

Returns

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

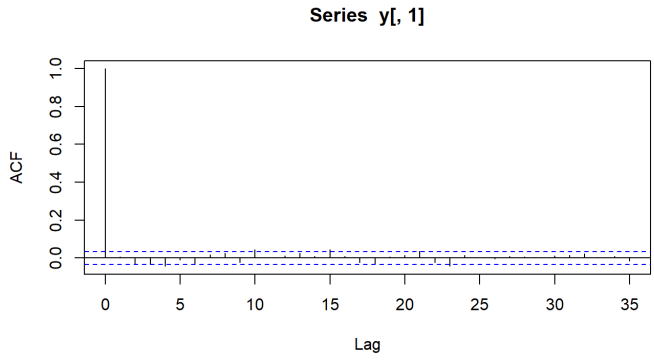
```
##            BRL.X.High BRL.X.Low
## BRL.X.High  1.0000000 0.5239676
## BRL.X.Low   0.5239676 1.0000000
```

```
plot(y)
```

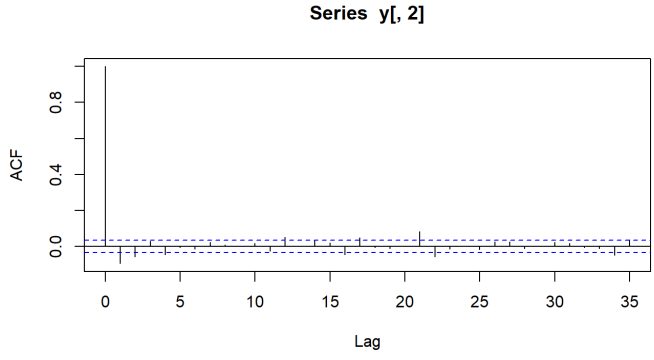


Autocorrelation

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

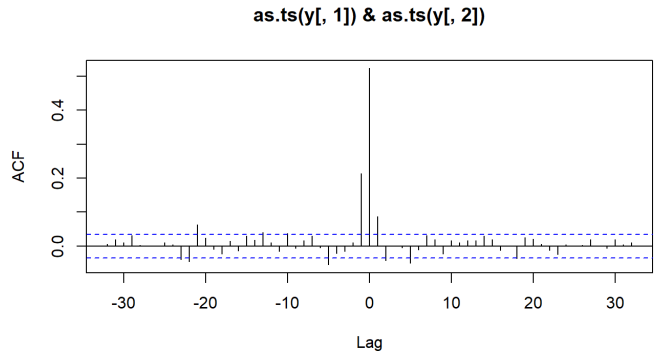



```
acf(y[,2])
```



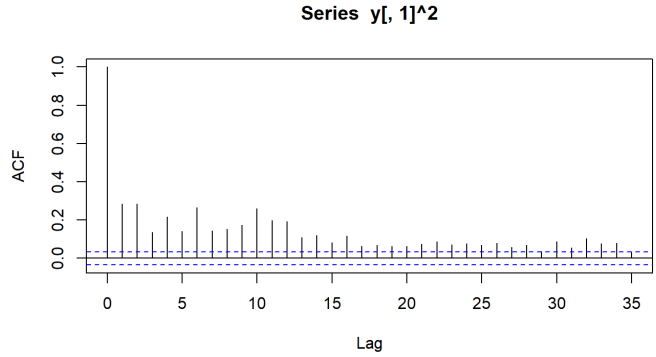
Cross correlation

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

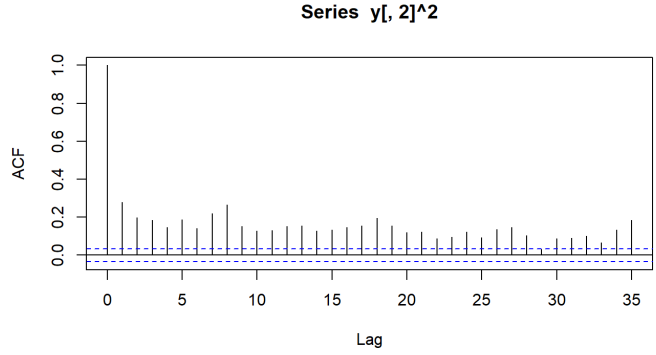


Volatility verification

```
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.

```
# 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.
```

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

```
## Calibrating the Lambda coefficient:
## lambda: 0.4
## Accept Rate: 0.33
## Done.
## Starting the simulation by one-block random walk Metropolis-Hasting algorithm.
## Done.
```

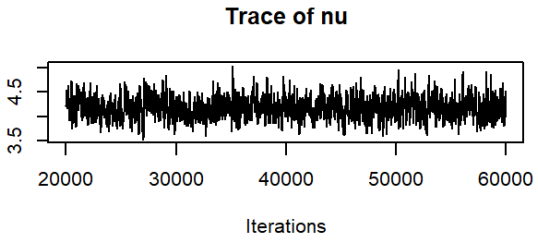
```
out2 <- increaseSim(out, nSim=50000)
```

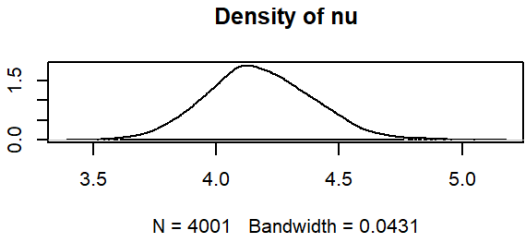
```
## Calibrating the Lambda coefficient:
## lambda: 0.4
## Accept Rate: 0.4
## Done.
## Starting the simulation by one-block random walk Metropolis-Hasting algorithm.
## Done.
```

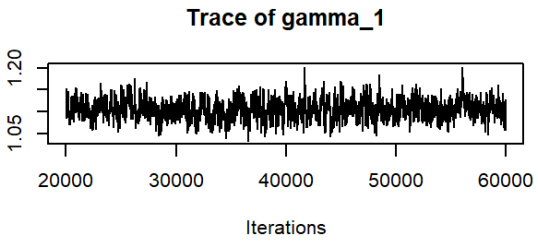
```
out <- window(out2, start=20000, thin=10)
rm(out2)

end <- Sys.time()

# elapsed time
end-start
```

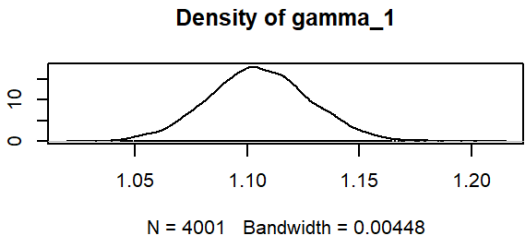
```
## Time difference of 2.554791 mins
```

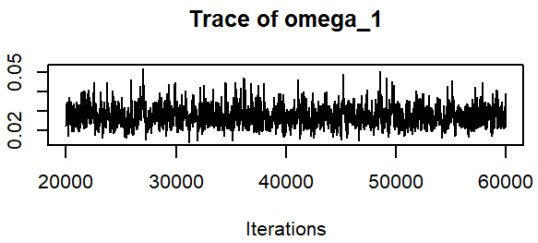
```
# plot Markov Chain
plot(out$MC)
```

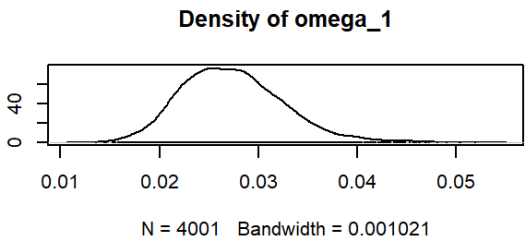

Trace of nu
3.5
4.5
20000
30000
40000
50000
60000
Iterations

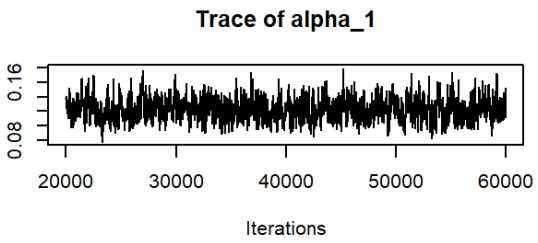

Density of nu
0.0
1.5
3.5
4.0
4.5
5.0
N = 4001 Bandwidth = 0.0431

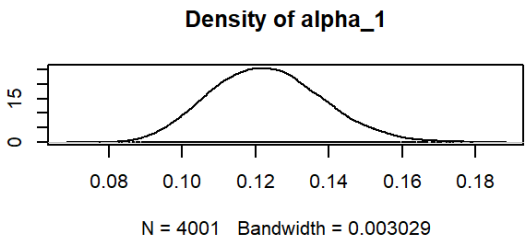

Trace of gamma_1
1.05
1.20
20000
30000
40000
50000
60000
Iterations

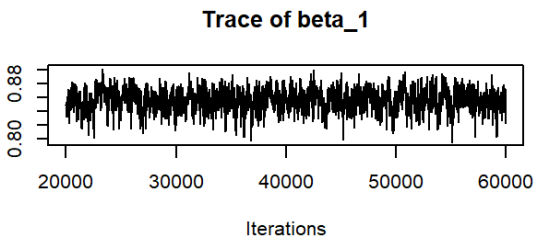

Density of gamma_1
0
10
1.05
1.10
1.15
1.20
N = 4001 Bandwidth = 0.00448

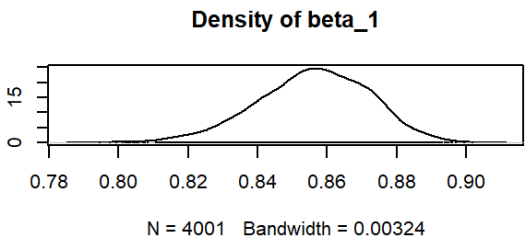

Trace of omega_1
0.02
0.05
20000
30000
40000
50000
60000
Iterations

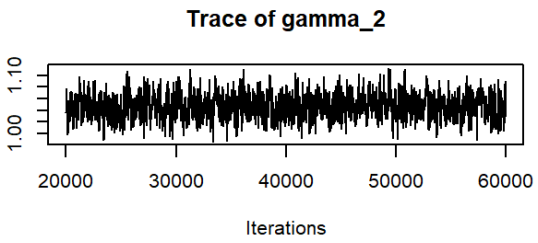

Density of omega_1
0
40
0.01
0.02
0.03
0.04
0.05
N = 4001 Bandwidth = 0.001021

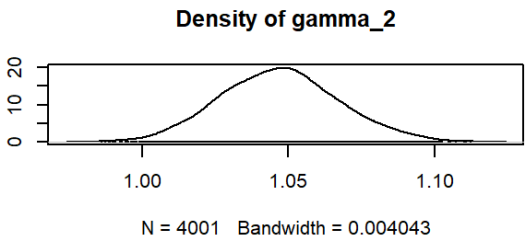

Trace of alpha_1
0.08
0.16
20000
30000
40000
50000
60000
Iterations

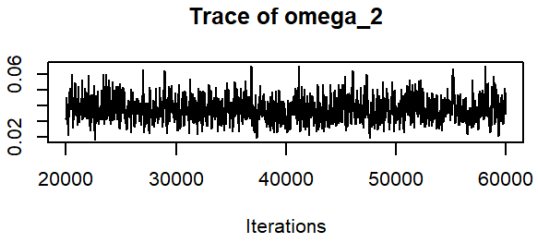

Density of alpha_1
0
15
0.08
0.10
0.12
0.14
0.16
0.18
N = 4001 Bandwidth = 0.003029

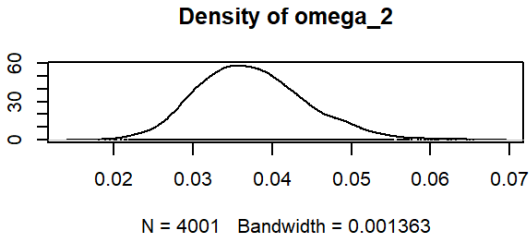

Trace of beta_1
0.80
0.88
20000
30000
40000
50000
60000
Iterations

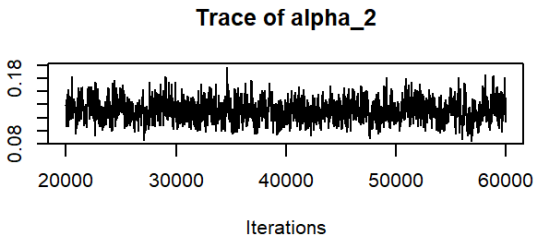

Density of beta_1
0
15
0.78
0.80
0.82
0.84
0.86
0.88
0.90
N = 4001 Bandwidth = 0.00324

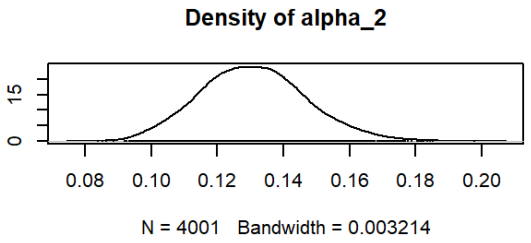

Trace of gamma_2
1.00
1.10
20000
30000
40000
50000
60000
Iterations

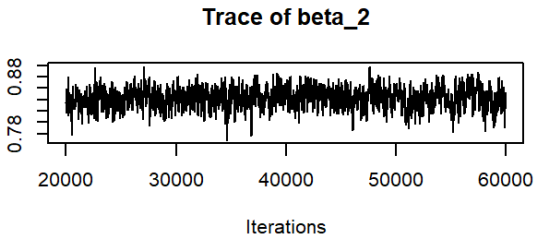

Density of gamma_2
0
10
20
1.00
1.05
1.10
N = 4001 Bandwidth = 0.004043

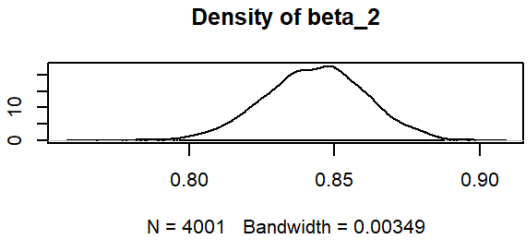


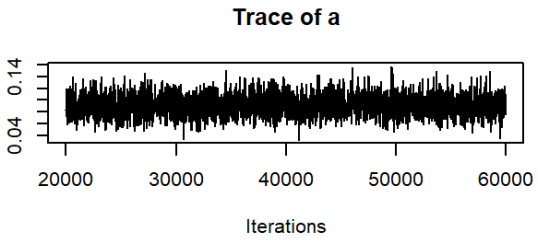


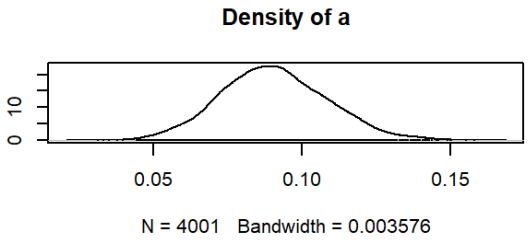


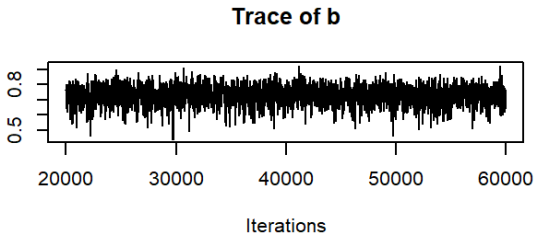


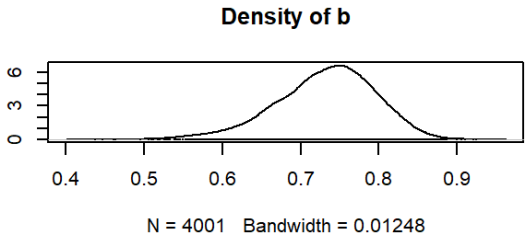













```
## Estimative of parameters
out$MC %>% summary()
```

```
##
## 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      4.17600 0.214778 3.396e-03      0.0082699
## gamma_1 1.10531 0.022432 3.546e-04      0.0010014
## omega_1 0.02738 0.005137 8.121e-05      0.0002020
## alpha_1 0.12311 0.015010 2.373e-04      0.0006239
## beta_1  0.85591 0.016057 2.538e-04      0.0006941
## gamma_2 1.04690 0.020117 3.180e-04      0.0007989
## omega_2 0.03742 0.006844 1.082e-04      0.0002711
## alpha_2 0.13087 0.015927 2.518e-04      0.0006390
## beta_2  0.84315 0.017317 2.738e-04      0.0007364
## a       0.09055 0.017727 2.803e-04      0.0003856
## b       0.73154 0.063726 1.007e-03      0.0016529
##
## 2. Quantiles for each variable:
##
##             2.5%     25%     50%     75%   97.5%
## nu      3.77732 4.03104 4.16597 4.31728 4.60985
## gamma_1 1.06180 1.09022 1.10460 1.11997 1.15005
## omega_1 0.01874 0.02373 0.02702 0.03052 0.03902
## alpha_1 0.09623 0.11233 0.12254 0.13289 0.15432
## beta_1  0.82176 0.84557 0.85663 0.86756 0.88495
## gamma_2 1.00893 1.03292 1.04691 1.05978 1.08768
## omega_2 0.02566 0.03258 0.03688 0.04164 0.05206
## alpha_2 0.10111 0.11973 0.13037 0.14132 0.16397
## beta_2  0.80864 0.83184 0.84367 0.85502 0.87614
## a       0.05716 0.07845 0.08987 0.10219 0.12692
## b       0.58885 0.69345 0.73753 0.77636 0.84076
```

```
# 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  2.087
## High_UB_S1    0.509
## Low_LB_B1    -0.543
## Low_LB_LBOP  -1.914
## High_omega    0.027
## High_alpha    0.123
## High_beta     0.856
## Low_omega     0.037
## Low_alpha     0.131
## Low_beta      0.843
```