Tutorial: Univariate Volatility Models

by Kevin Kotzé

To complete this tutorial you should open the file `T-8-uvol.Rproj` .

1 Conditional Heteroscedastic Model - S&P500

The first example considers the use of an ARCH/GARCH modelling framework for the S&P500 index. This example is contained in the file `tut8a-sp500.R` . To start off we can clear all the variables from the current environment and close all the plots.

```
rm(list=ls())
graphics.off()
```

Thereafter, we will need to make use of the `tidyverse` , `fGarch` and `tsm` packages, so we make use of the `library` command.

```
library(tidyverse)
library(fGarch)
library(tsm)
```

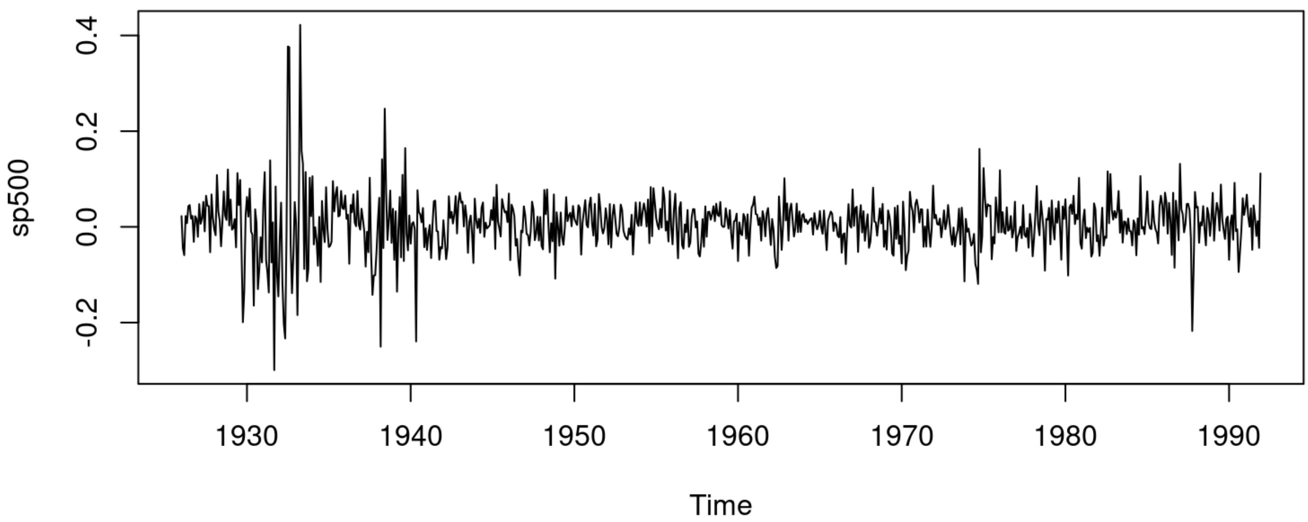
If you need install these packages run the following routine: `install.packages("fGarch")` . To install the `tsm` package we make use of the commands, `library(devtools)` and `devtools::install_github("KevinKotze/tsm")` .

We will then need to load the data and create the time series objects for the S&P500 monthly excess returns between 1926 and 1991. As the data is contained in a `.csv` file, which should be in the same folder as `T8-uvol.Rproj` , we would need to read this data into memory.

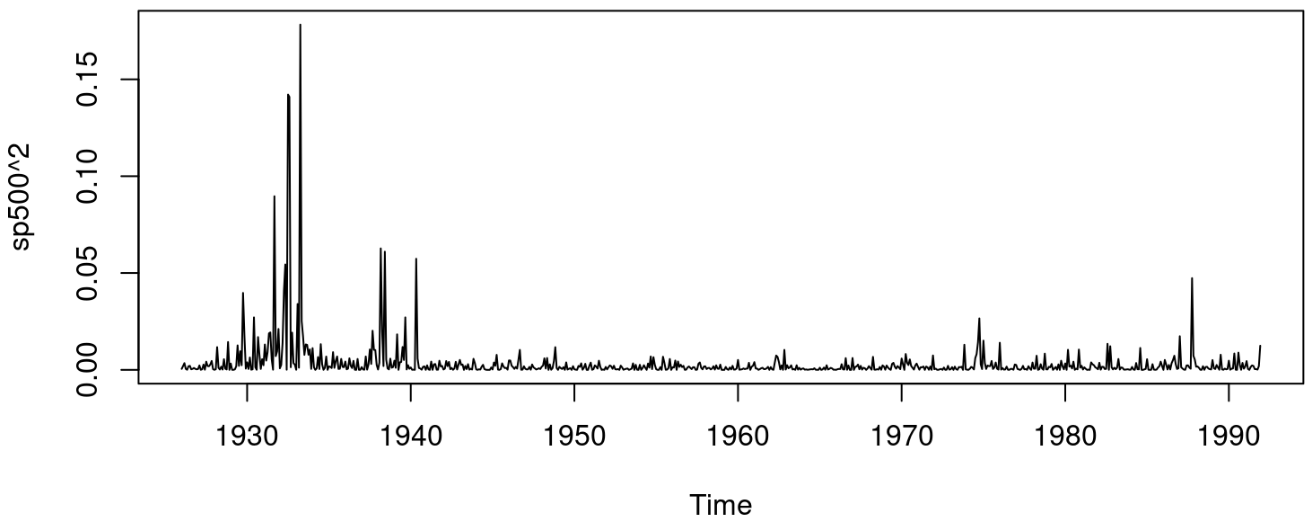
```
dat <- read_csv(file = "sp500.csv")
sp500 <- ts(dat$sp500, start = c(1926, 1), freq = 12)
```

To ensure that this has all been completed correctly, we can plot the data.

```
plot(sp500)
```

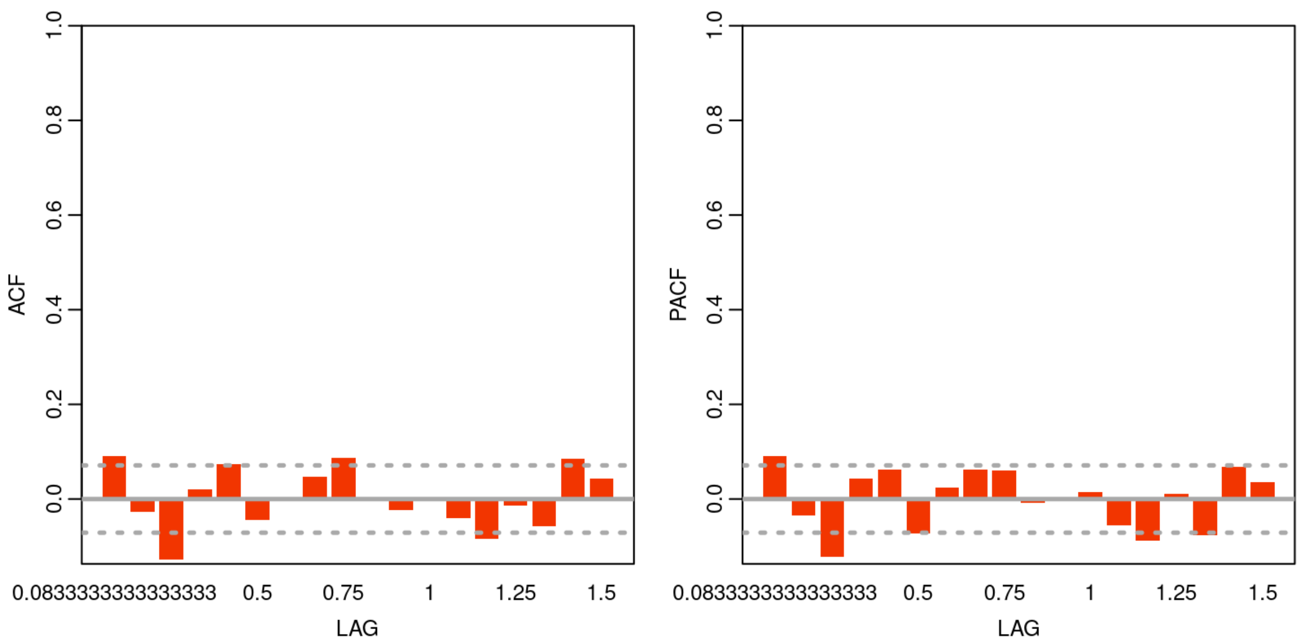



```
plot(sp500^2)
```



The first part of the ARCH modelling process is to specify an appropriate mean equation that will be able to explain the serial dependence in the first moment of the process. To complete this task, we can make use of an autocorrelation function.

```
res1 <- ac(sp500)
```

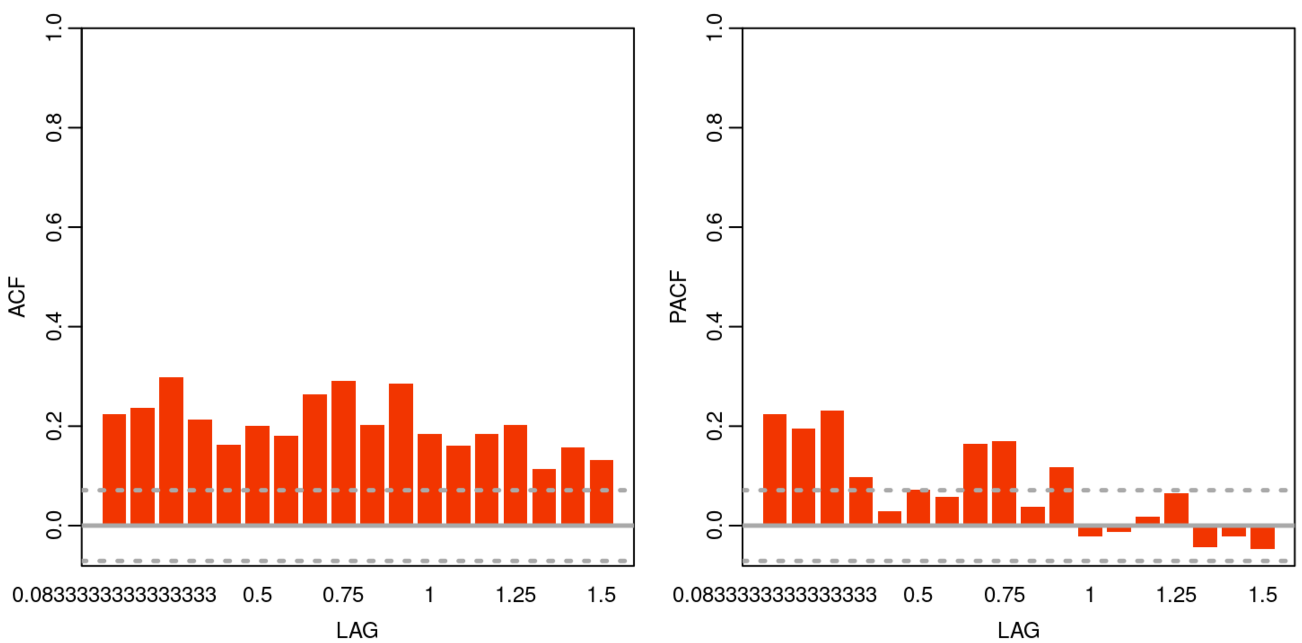



```
Box.test(sp500, lag = 12, type = 'Ljung')
```

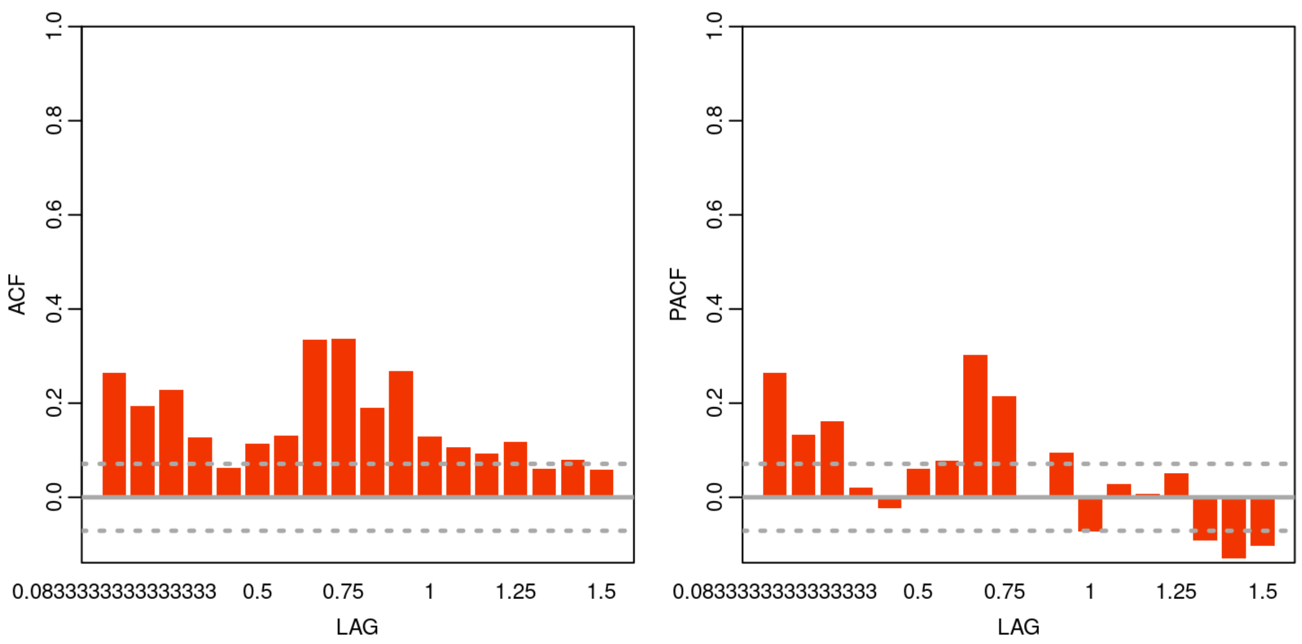
```
##
##  Box-Ljung test
##
## data:  sp500
## X-squared = 34.372, df = 12, p-value = 0.0005892
```

In this case it is noted that there is some possible persistence and the model for the mean equation may take the form of an AR(3) or a MA(3), although the coefficients in both the autocorrelation and partial autocorrelation coefficients are small. We can also have a quick look at the degree of persistence in the volatility of the process.

```
res2 <- ac(abs(sp500))
```



```
res3 <- ac(sp500^2)
```



where we note that there is definitely some persistence. To test for the significance of a mean that differs from zero we can use a *t*-test, which suggests that the mean is different from zero. Therefore, we demean the time series variable and perform further tests for volatility effects. These include the Box-Ljung and Arch test of Engle (1982).

```
t.test(sp500)
```

```
##
##  One Sample t-test
##
## data:  sp500
## t = 2.9573, df = 791, p-value = 0.003196
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  0.002065495 0.010220616
## sample estimates:
##   mean of x
## 0.006143056
```

```
at <- sp500 - mean(sp500)
Box.test(abs(at), lag = 12, type = 'Ljung')
```

```
##
##  Box-Ljung test
##
## data:  abs(at)
## X-squared = 557.08, df = 12, p-value < 2.2e-16
```

```
archTest(at, 12)
```

```
##
## Call:
## lm(formula = atsq ~ x)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.046462 -0.001960 -0.001089  0.000496  0.122800
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.0008310  0.0004186   1.985  0.04749 *
## x1           0.1112910  0.0360675   3.086  0.00210 **
## x2           0.0632316  0.0360710   1.753  0.08001 .
## x3           0.1500567  0.0361453   4.151 3.67e-05 ***
## x4           0.0344645  0.0358201   0.962  0.33628
## x5          -0.0851777  0.0347166  -2.454  0.01437 *
## x6          -0.0062502  0.0348527  -0.179  0.85772
## x7          -0.0023330  0.0348531  -0.067  0.94665
## x8           0.2463731  0.0347162   7.097 2.91e-12 ***
## x9           0.2010920  0.0358199   5.614 2.76e-08 ***
## x10         -0.0010279  0.0361446  -0.028  0.97732
## x11          0.1100626  0.0360725   3.051  0.00236 **
## x12         -0.0588513  0.0360674  -1.632  0.10315
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.01012 on 767 degrees of freedom
## Multiple R-squared:  0.2457, Adjusted R-squared:  0.2339
## F-statistic: 20.82 on 12 and 767 DF,  p-value: < 2.2e-16
```

These results are similar to what was provided earlier, where the F-statistics and p-values confirm that ARCH effects are present. We can now consider the results of a few competing models, which include an AR(3), MA(3), AR(3) with GARCH(1,1), and a GARCH(1,1) with a student-t distribution.

```
m1 <- arima(sp500, order = c(0, 0, 3))
m1
```

```
##
## Call:
## arima(x = sp500, order = c(0, 0, 3))
##
## Coefficients:
##          ma1     ma2      ma3  intercept
##       0.0949  0.0096  -0.1415     0.0062
## s.e.  0.0348  0.0355   0.0357     0.0020
##
## sigma^2 estimated as 0.003321:  log likelihood = 1136.3,  aic = -2262.6
```

```
m2 <- arima(sp500, order = c(3, 0, 0))
m2
```

```
##
## Call:
## arima(x = sp500, order = c(3, 0, 0))
##
## Coefficients:
##          ar1      ar2      ar3  intercept
##       0.0890  -0.0238  -0.1229     0.0062
## s.e.  0.0353   0.0355   0.0353     0.0019
##
## sigma^2 estimated as 0.00333:  log likelihood = 1135.25,  aic = -2260.5
```

```
m3 <- garchFit(sp500 ~ arma(3, 0) + garch(1, 1),
               data = sp500,
               trace = F)
summary(m3)
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = sp500 ~ arma(3, 0) + garch(1, 1), data = sp500,
##     trace = F)
##
## Mean and Variance Equation:
##  data ~ arma(3, 0) + garch(1, 1)
## <environment: 0x556196607a78>
##  [data = sp500]
##
## Conditional Distribution:
##  norm
##
## Coefficient(s):
##          mu          ar1          ar2          ar3
##  7.7078e-03   3.1969e-02  -3.0262e-02  -1.0650e-02
##       omega       alpha1        beta1
##  7.9746e-05   1.2425e-01   8.5302e-01
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##          Estimate  Std. Error  t value Pr(>|t|)
## mu      7.708e-03   1.607e-03    4.798 1.61e-06 ***
## ar1     3.197e-02   3.837e-02    0.833  0.40471
## ar2    -3.026e-02   3.841e-02   -0.788  0.43074
## ar3    -1.065e-02   3.756e-02   -0.284  0.77675
## omega   7.975e-05   2.810e-05    2.838  0.00454 **
## alpha1  1.242e-01   2.247e-02    5.529 3.22e-08 ***
## beta1   8.530e-01   2.183e-02   39.076  < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  1272.179    normalized:  1.606287
##
## Description:
##  Thu Oct 15 14:06:04 2020 by user: kevin
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  73.04811  1.110223e-16
##  Shapiro-Wilk Test  R    W      0.9857969 5.961684e-07
##  Ljung-Box Test     R    Q(10)  11.56752  0.3150426
##  Ljung-Box Test     R    Q(15)  17.78752  0.2740008
##  Ljung-Box Test     R    Q(20)  24.11924  0.2372224
##  Ljung-Box Test     R^2  Q(10)  10.3161   0.4132124
##  Ljung-Box Test     R^2  Q(15)  14.22818  0.5082983
##  Ljung-Box Test     R^2  Q(20)  16.79405  0.6663035
```

```
##  LM Arch Test       R    TR^2   13.34304  0.3446081
##
## Information Criterion Statistics:
##       AIC       BIC       SIC      HQIC
## -3.194897 -3.153581 -3.195051 -3.179018
```

```
m4 <- garchFit(
  sp500 ~ garch(1, 1),
  data = sp500,
  cond.dist = "std",
  trace = F
)
summary(m4)
```

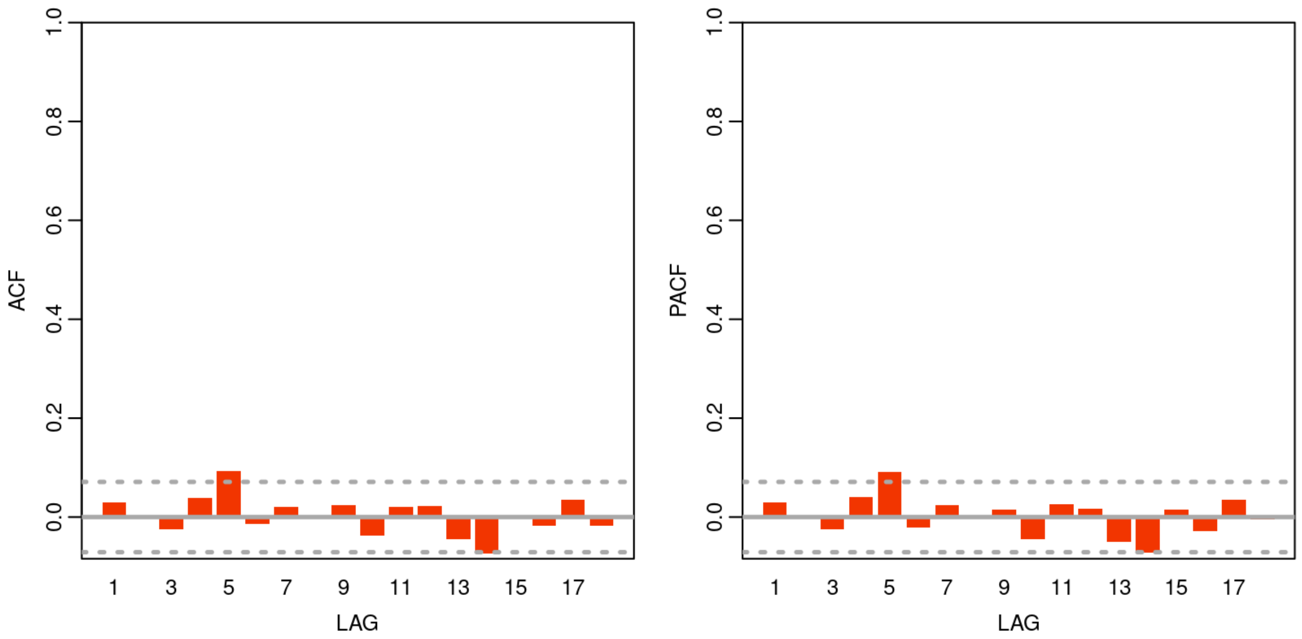
```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = sp500 ~ garch(1, 1), data = sp500, cond.dist = "std",
##     trace = F)
##
## Mean and Variance Equation:
##  data ~ garch(1, 1)
## <environment: 0x55619c1e2930>
##  [data = sp500]
##
## Conditional Distribution:
##  std
##
## Coefficient(s):
##         mu       omega      alpha1       beta1
## 0.00845503  0.00012485  0.11302615  0.84220143
##      shape
## 7.00317917
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu     8.455e-03   1.515e-03    5.581 2.39e-08 ***
## omega  1.248e-04   4.519e-05    2.763  0.00573 **
## alpha1 1.130e-01   2.693e-02    4.198 2.70e-05 ***
## beta1  8.422e-01   3.186e-02   26.432  < 2e-16 ***
## shape  7.003e+00   1.680e+00    4.169 3.06e-05 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  1283.417    normalized:  1.620476
##
## Description:
##  Thu Oct 15 14:06:04 2020 by user: kevin
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  99.61279  0
##  Shapiro-Wilk Test  R    W      0.9836345 9.727883e-08
##  Ljung-Box Test     R    Q(10)  11.37961  0.3287175
##  Ljung-Box Test     R    Q(15)  18.2163   0.2514651
##  Ljung-Box Test     R    Q(20)  24.91842  0.20457
##  Ljung-Box Test     R^2  Q(10)  10.52268  0.3958929
##  Ljung-Box Test     R^2  Q(15)  16.14587  0.3724239
##  Ljung-Box Test     R^2  Q(20)  18.93327  0.5261674
##  LM Arch Test       R    TR^2   14.88669  0.2476919
##
```

```
## Information Criterion Statistics:
##       AIC       BIC       SIC      HQIC
## -3.228325 -3.198814 -3.228404 -3.216983
```

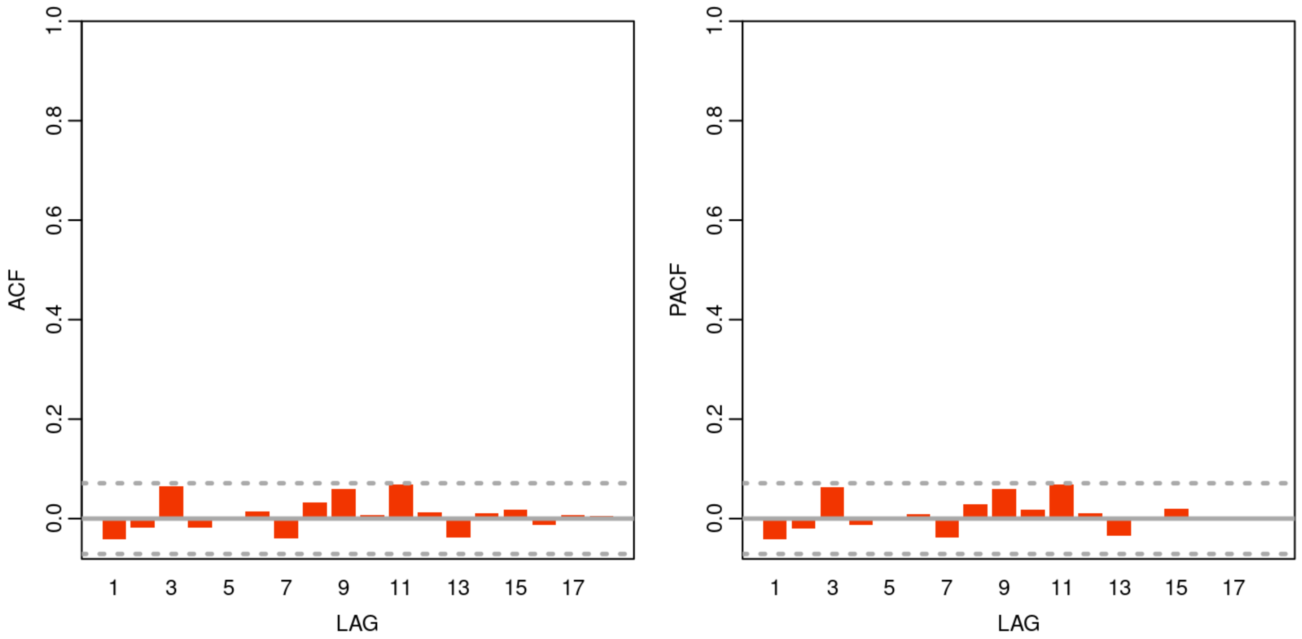
Note that either the AR(3) or MA(3) would appear to be appropriate, however, when we include a GARCH specification with the AR(3) we note that all the coefficients from the mean equation appear to be insignificant. This is due to the fact that the persistence in the second moment is also reflected by persistence in the first moment in the autocorrelation functions. As such we proceed to model a simple GARCH(1,1), where we note that all the coefficients are significant.

To ensure that there is no remaining serial correlation in the residuals of the GARCH(1,1) we make use of autocorrelation functions once again.

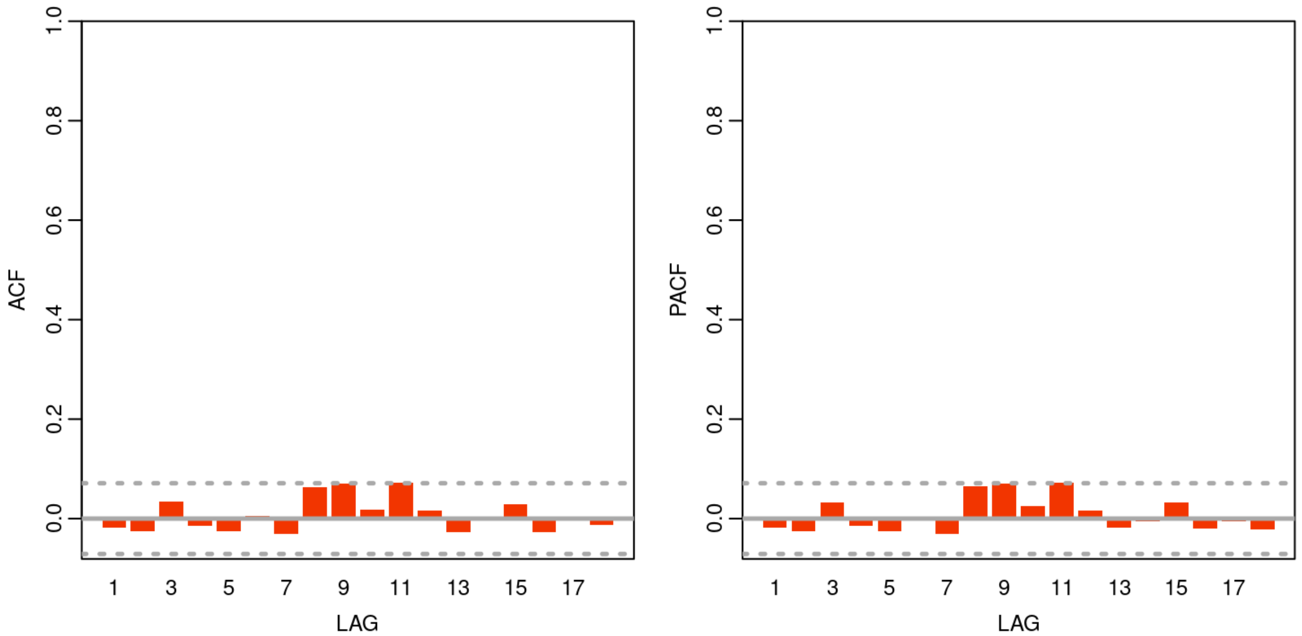
```
check1 <- ac(residuals(m4, standardize = T))
```



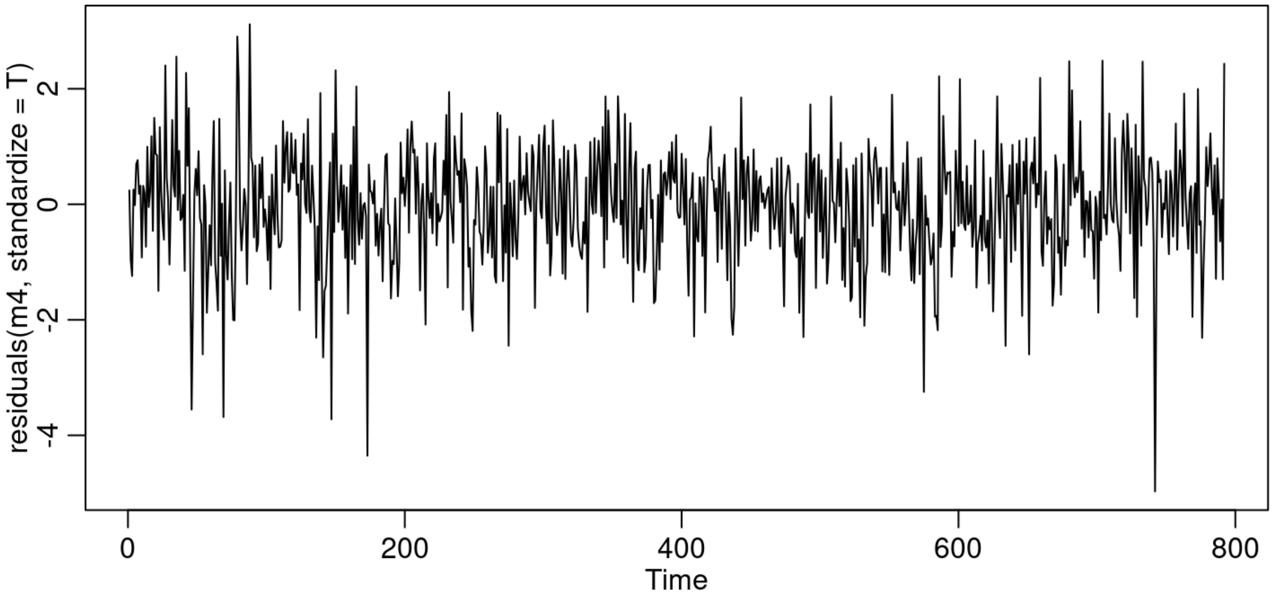

```
check2 <- ac((residuals(m4, standardize = T) ^ 2) ^ 0.5)
```




```
check3 <- ac((residuals(m4, standardize = T) ^ 2))
```




```
par(mfrow = c(1, 1))
plot.ts(residuals(m4, standardize = T))
```



where we note that the mean, absolute value and variance do not display notable degrees of serial correlation. In addition, the plot of the residuals would also appear to represent white noise.

Should we wish to generate a forecast for this model then we can do so by making use of the `predict` command.

```
predict(m4, 5)
```

```
##   meanForecast  meanError standardDeviation
## 1  0.008455033 0.05330091        0.05330091
## 2  0.008455033 0.05327888        0.05327888
## 3  0.008455033 0.05325782        0.05325782
## 4  0.008455033 0.05323770        0.05323770
## 5  0.008455033 0.05321847        0.05321847
```

2 Conditional Heteroscedastic Model - Intel Share Price

The second example considers the use of an ARCH model for the Intel share price. This example is contained in the file `tut8b-intel.R`. To start off we can clear all the variables from the current environment and close all the plots.

```
rm(list = ls())
graphics.off()
```

Thereafter, we will need to make use of the `tidyverse`, `fGarch` and `tsm` packages, so we make use of the `library` command.

```
library(tidyverse)
library(fGarch)
library(tsm)
```

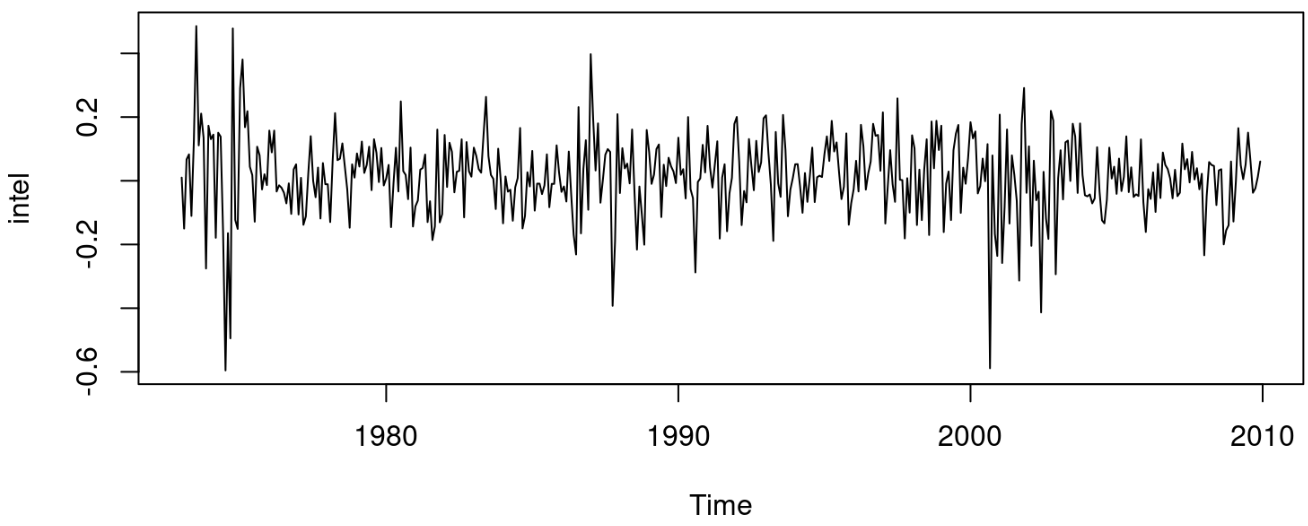
If you need install these packages run the following routine: `install.packages("fGarch")`. To install the `tsm` package we make use of the commands, `library(devtools)` and `devtools::install_github("KevinKotze/tsm")`.

As the data is contained in a `.csv` file, which should be in the same folder as `T8-uvol.Rproj` we would need to read this data into memory. After loading the data we create a time series object for the monthly logarithmic returns of the Intel share price.

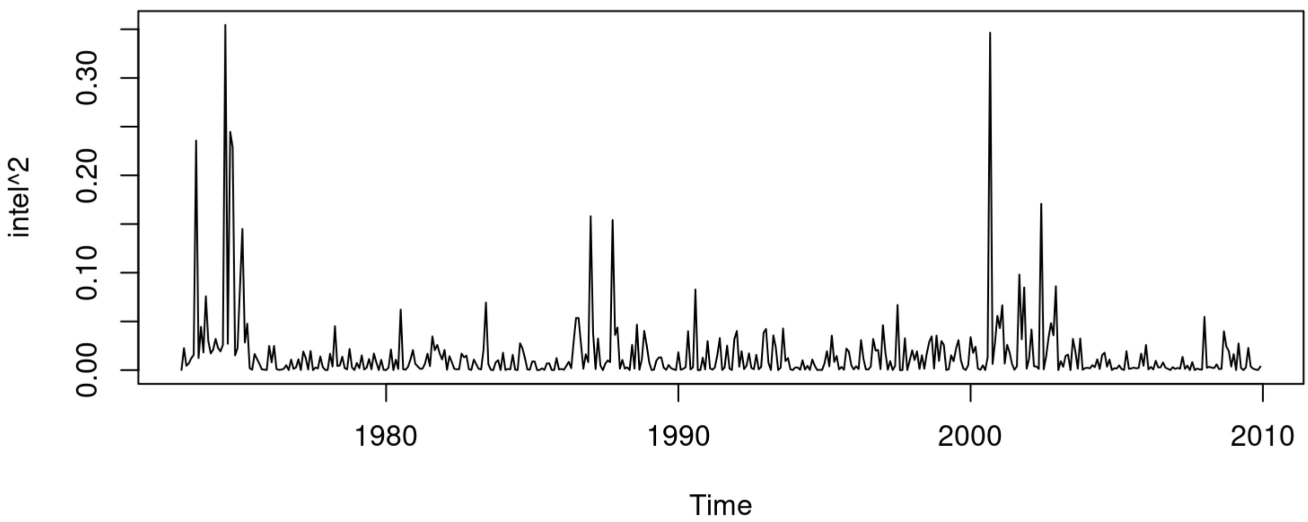
```
dat <- read_csv(file = "intel.csv")
intel <- ts((log(dat$intc + 1)), start = c(1973, 1), freq = 12)
```

To ensure that this has all been completed correctly, we can plot the data.

```
plot(intel)
```

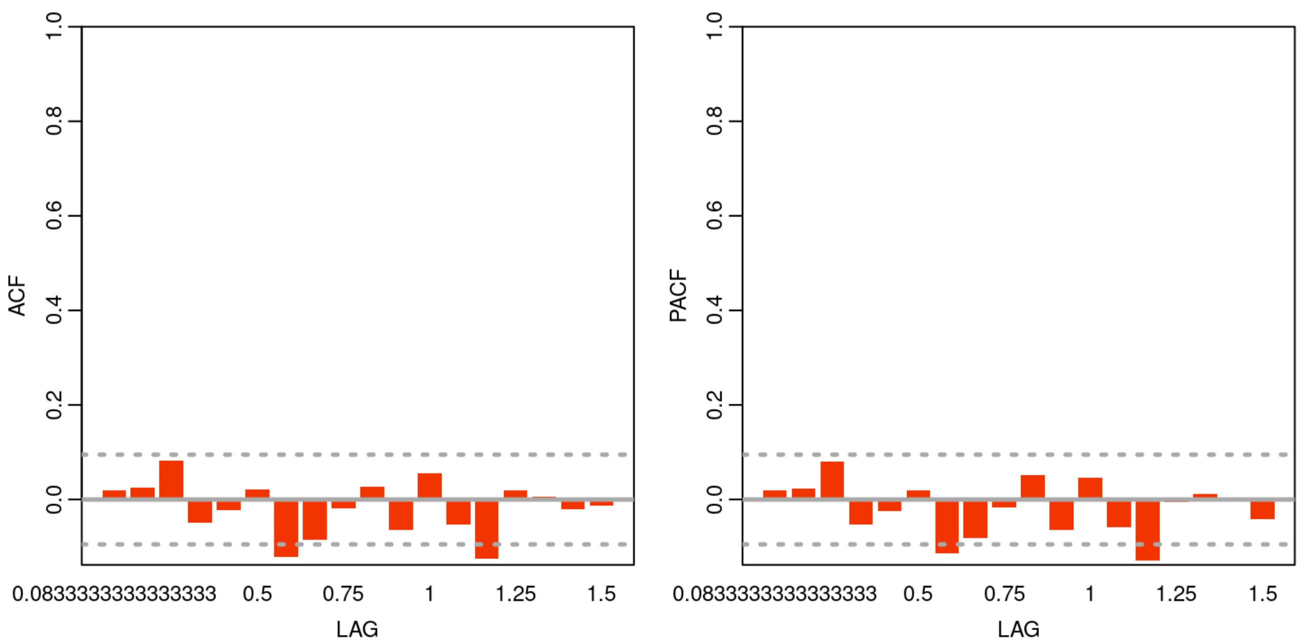



```
plot(intel ^ 2)
```



The first part of the ARCH modelling process is to specify an appropriate mean equation that will be able to explain the serial dependence in the first moment of the process. To complete this task, we can make use of an autocorrelation function.

```
res1 <- ac(intel)
```

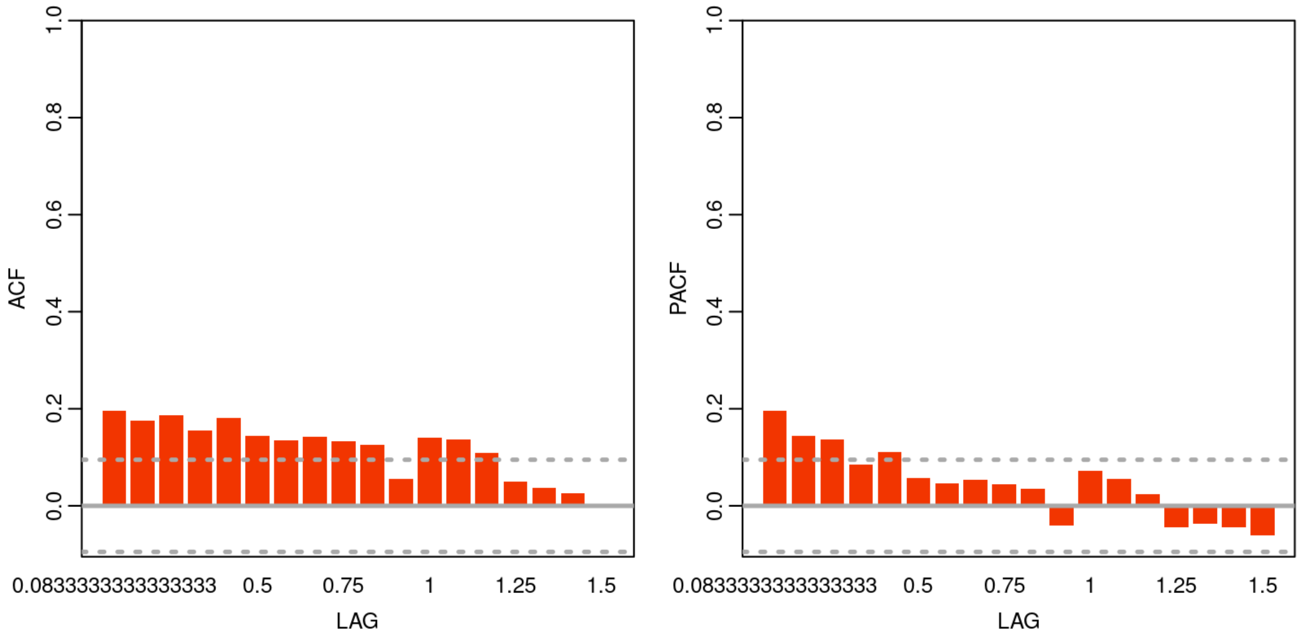



```
Box.test(intel, lag = 12, type = 'Ljung')
```

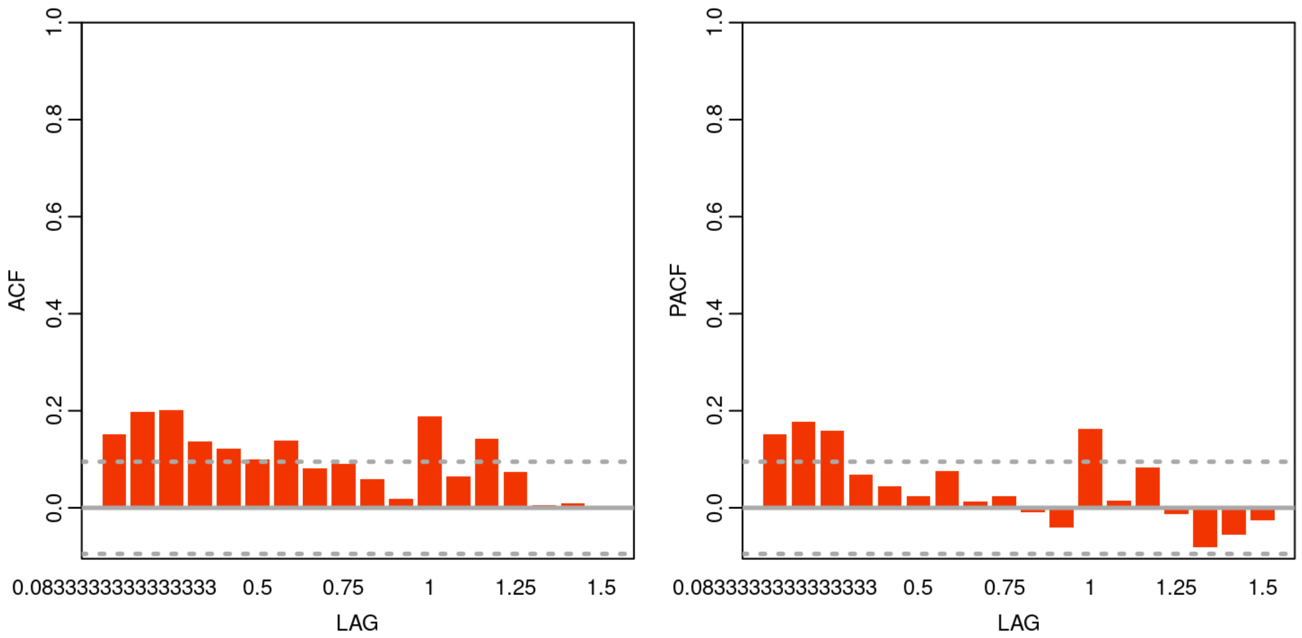
```
##
##  Box-Ljung test
##
## data:  intel
## X-squared = 18.676, df = 12, p-value = 0.09665
```

In this case it is noted that there is not much persistence in the mean equation. We can also have a quick look at the degree of persistence in the volatility of the process.

```
res2 <- ac(abs(intel))
```




```
res3 <- ac(intel ^ 2)
```




```
Box.test(abs(intel), lag = 12, type = 'Ljung')
```

```
##
##  Box-Ljung test
##
## data:  abs(intel)
## X-squared = 124.91, df = 12, p-value < 2.2e-16
```

where we note that there is definitely some persistence. To test for the significance of a mean that differs from zero we can use a *t*-test, which suggests that the mean is different from zero. Therefore, we demean the time series variable and perform further tests for volatility effects. These include the Box-Ljung and Arch test of Engle (1982).

```
t.test(intel)
```

```
##
##  One Sample t-test
##
## data:  intel
## t = 2.3788, df = 443, p-value = 0.01779
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  0.00249032 0.02616428
## sample estimates:
## mean of x
## 0.0143273
```

```
at <- intel - mean(intel)
Box.test(at ^ 2, lag = 12, type = 'Ljung')
```

```
##
##  Box-Ljung test
##
## data:  at^2
## X-squared = 92.939, df = 12, p-value = 1.332e-14
```

```
archTest(at, 12)
```

```
##
## Call:
## lm(formula = atsq ~ x)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.07440 -0.01153 -0.00658  0.00395  0.35255
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.005977   0.002249   2.658 0.008162 **
## x1           0.093817   0.048147   1.949 0.052013 .
## x2           0.153085   0.048102   3.183 0.001569 **
## x3           0.146087   0.048614   3.005 0.002815 **
## x4           0.023539   0.049126   0.479 0.632075
## x5           0.007347   0.049107   0.150 0.881139
## x6           0.010342   0.047027   0.220 0.826050
## x7           0.057183   0.047027   1.216 0.224681
## x8           0.014320   0.047079   0.304 0.761149
## x9           0.007157   0.046968   0.152 0.878965
## x10         -0.019742   0.046566  -0.424 0.671810
## x11         -0.057537   0.046041  -1.250 0.212116
## x12          0.161945   0.045965   3.523 0.000473 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03365 on 419 degrees of freedom
## Multiple R-squared:  0.1248, Adjusted R-squared:  0.0997
## F-statistic: 4.978 on 12 and 419 DF,  p-value: 9.742e-08
```

These results are similar to what was provided earlier, where the F-statistics and p-values confirm that ARCH effects are present. We can now consider the results of a few competing models, which include an AR(3), MA(3), AR(3) with GARCH(1,1), and a GARCH(1,1) with a student-t distribution.

```
m0 <- garchFit(intel ~ garch(3, 0), data = intel, trace = F)
summary(m0)
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = intel ~ garch(3, 0), data = intel, trace = F)
##
## Mean and Variance Equation:
##  data ~ garch(3, 0)
## <environment: 0x55619a911fd8>
##  [data = intel]
##
## Conditional Distribution:
##  norm
##
## Coefficient(s):
##        mu     omega    alpha1    alpha2    alpha3
## 0.012567  0.010421  0.232889  0.075069  0.051993
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.012567    0.005515    2.279   0.0227 *
## omega   0.010421    0.001238    8.418   <2e-16 ***
## alpha1  0.232889    0.111541    2.088   0.0368 *
## alpha2  0.075069    0.047305    1.587   0.1125
## alpha3  0.051993    0.045139    1.152   0.2494
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  303.9607    normalized:  0.6845963
##
## Description:
##  Thu Oct 15 14:06:05 2020 by user: kevin
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  203.3619  0
##  Shapiro-Wilk Test  R    W      0.9635971 4.898663e-09
##  Ljung-Box Test     R    Q(10)  9.260782  0.5075463
##  Ljung-Box Test     R    Q(15)  19.36748  0.197562
##  Ljung-Box Test     R    Q(20)  20.46982  0.428906
##  Ljung-Box Test     R^2  Q(10)  7.32214   0.694723
##  Ljung-Box Test     R^2  Q(15)  27.41533  0.02552902
##  Ljung-Box Test     R^2  Q(20)  28.15114  0.1058697
##  LM Arch Test       R    TR^2   25.23347  0.01375446
##
## Information Criterion Statistics:
##       AIC       BIC       SIC      HQIC
## -1.346670 -1.300546 -1.346920 -1.328481
```

```
m1 <- garchFit(intel ~ garch(1, 0), data = intel, trace = F)
summary(m1)
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = intel ~ garch(1, 0), data = intel, trace = F)
##
## Mean and Variance Equation:
##  data ~ garch(1, 0)
## <environment: 0x55619c0ba438>
##  [data = intel]
##
## Conditional Distribution:
##  norm
##
## Coefficient(s):
##       mu     omega    alpha1
## 0.013130  0.011046  0.374976
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.013130    0.005318    2.469  0.01355 *
## omega   0.011046    0.001196    9.238  < 2e-16 ***
## alpha1  0.374976    0.112620    3.330  0.00087 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  299.9247    normalized:  0.675506
##
## Description:
##  Thu Oct 15 14:06:05 2020 by user: kevin
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  144.3782  0
##  Shapiro-Wilk Test  R    W      0.9678175 2.670339e-08
##  Ljung-Box Test     R    Q(10)  12.12247  0.2769434
##  Ljung-Box Test     R    Q(15)  22.30704  0.1000021
##  Ljung-Box Test     R    Q(20)  24.33411  0.228102
##  Ljung-Box Test     R^2  Q(10)  16.57804  0.08423799
##  Ljung-Box Test     R^2  Q(15)  37.44344  0.001089753
##  Ljung-Box Test     R^2  Q(20)  38.8139   0.007031666
##  LM Arch Test       R    TR^2   27.32894  0.006926884
##
## Information Criterion Statistics:
##       AIC       BIC       SIC      HQIC
## -1.337499 -1.309824 -1.337589 -1.326585
```

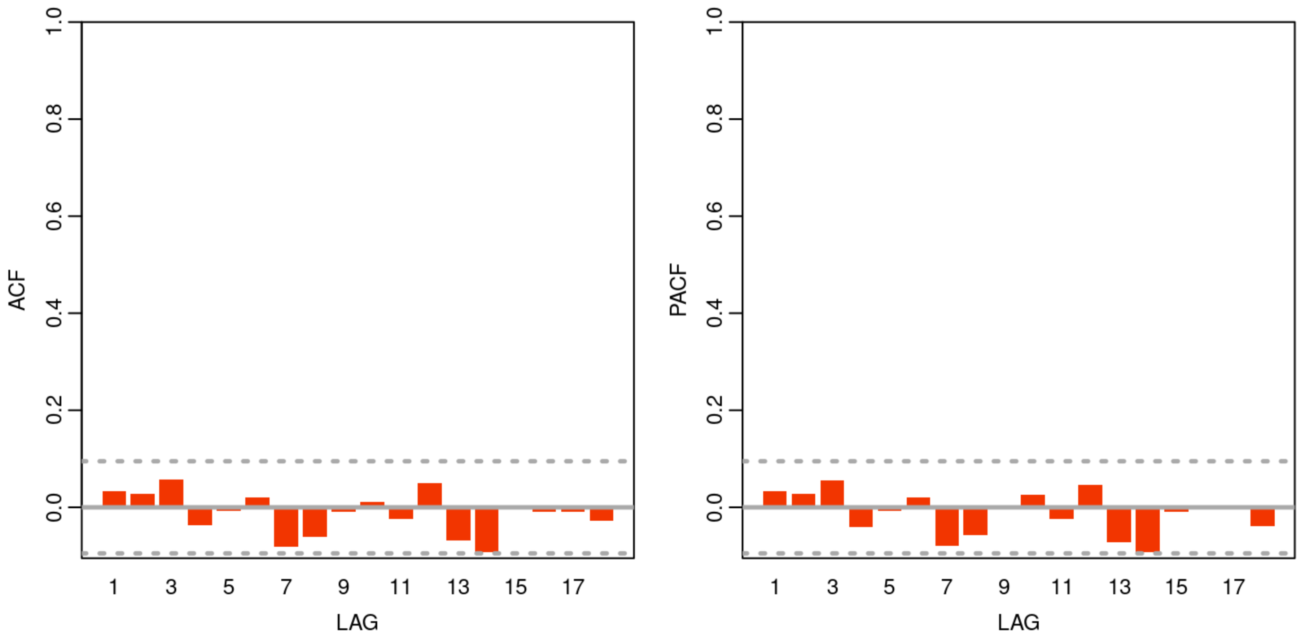
```
m2 <- garchFit(intel ~ garch(1, 1), data = intel, trace = F)
summary(m2)
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = intel ~ garch(1, 1), data = intel, trace = F)
##
## Mean and Variance Equation:
##  data ~ garch(1, 1)
## <environment: 0x556197f47548>
##  [data = intel]
##
## Conditional Distribution:
##  norm
##
## Coefficient(s):
##         mu       omega      alpha1       beta1
## 0.01126568  0.00091902  0.08643831  0.85258554
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu     0.0112657   0.0053931    2.089  0.03672 *
## omega  0.0009190   0.0003888    2.364  0.01808 *
## alpha1 0.0864383   0.0265439    3.256  0.00113 **
## beta1  0.8525855   0.0394322   21.622  < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  312.3307    normalized:  0.7034475
##
## Description:
##  Thu Oct 15 14:06:05 2020 by user: kevin
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  174.904   0
##  Shapiro-Wilk Test  R    W      0.9709615 1.030281e-07
##  Ljung-Box Test     R    Q(10)  8.016844  0.6271916
##  Ljung-Box Test     R    Q(15)  15.5006   0.4159946
##  Ljung-Box Test     R    Q(20)  16.41549  0.6905368
##  Ljung-Box Test     R^2  Q(10)  0.8746345 0.9999072
##  Ljung-Box Test     R^2  Q(15)  11.35935  0.7267295
##  Ljung-Box Test     R^2  Q(20)  12.55994  0.8954573
##  LM Arch Test       R    TR^2   10.51401  0.5709617
##
## Information Criterion Statistics:
##       AIC       BIC       SIC      HQIC
## -1.388877 -1.351978 -1.389037 -1.374326
```

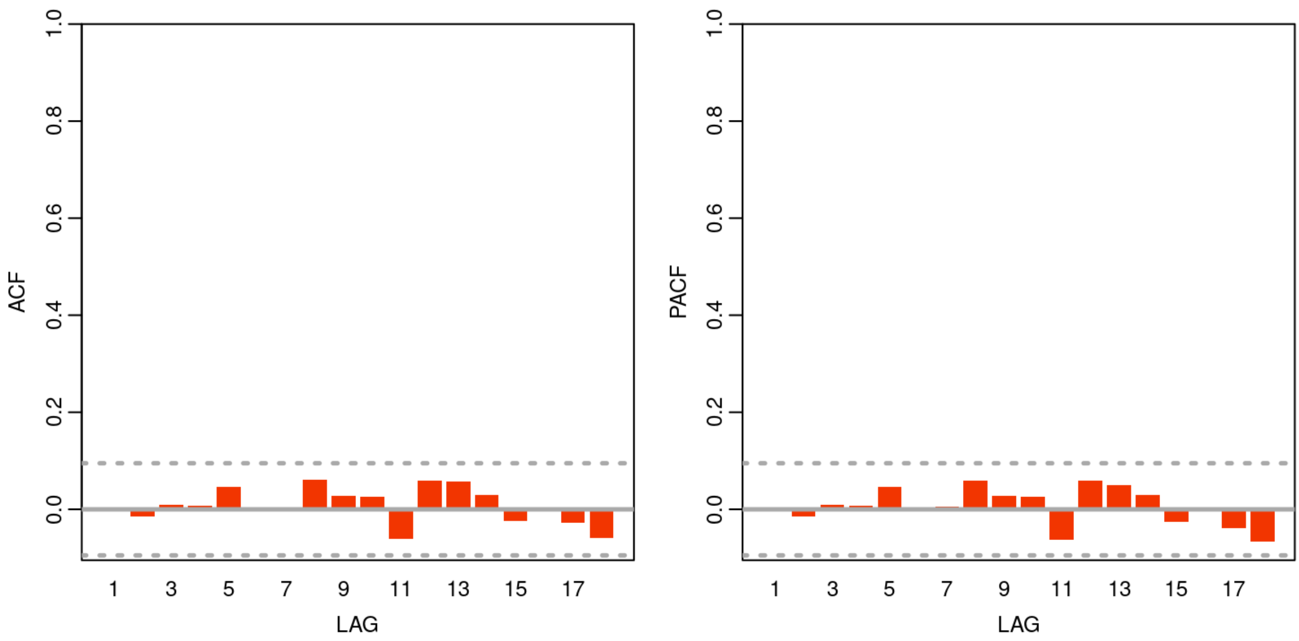
In terms of the notation, `mu` represents the constant in mean equation and `alpha` refers to the ARCH terms. Note that `alpha2` and `alpha3` appear to be insignificant in `m0`, while the `Qstat` is fairly large, which would suggest that there is no serial correlation in residual. In the second model, which represents an ARCH(1) the coefficients appear significant and the `Qstat` is acceptable at the 5% level of significance, while the Qstat for the volatility is relatively small. In the third model, which takes the form of a GARCH(1,1), we note that all the coefficients are significant.

To ensure that there is no remaining serial correlation in the residuals of the GARCH(1,1) we make use of autocorrelation functions once again.

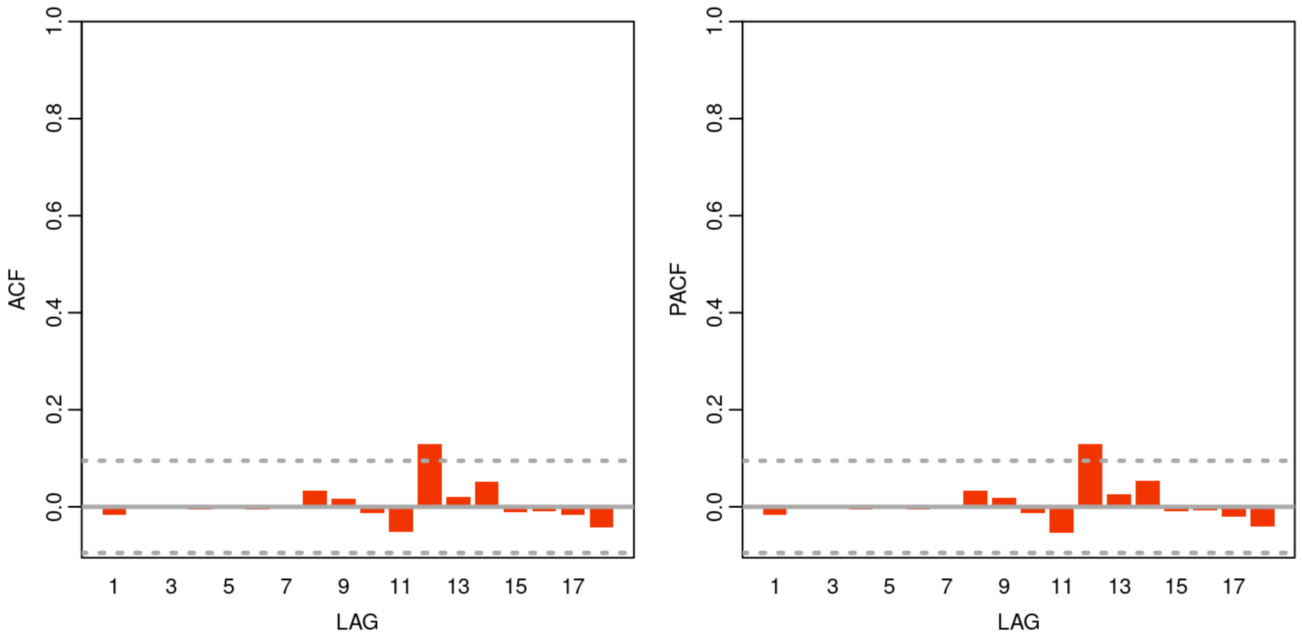
```
check1 <- ac(residuals(m2, standardize = T))
```



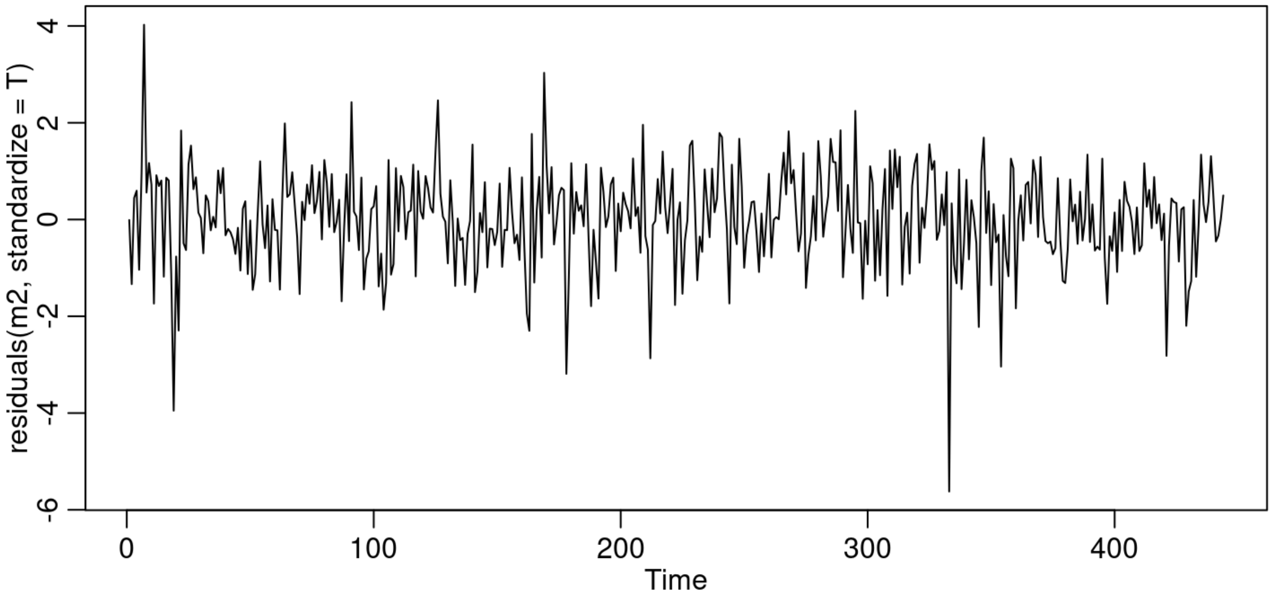

```
check2 <- ac((residuals(m2, standardize = T) ^ 2) ^ 0.5)
```



```
check3 <- ac((residuals(m2, standardize = T) ^ 2))
```




```
par(mfrow = c(1, 1))
plot.ts(residuals(m2, standardize = T))
```



where we note that the mean, absolute value and variance do not display notable degrees of serial correlation. In addition, the plot of the residuals would also appear to represent white noise.

As an alternative we could make use of a Students t-distribution or a skewed Students t-distribution.

```
m3 <- garchFit(
  intel ~ garch(1, 1),
  data = intel,
  cond.dist = "std",
  trace = F
)
summary(m3)
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = intel ~ garch(1, 1), data = intel, cond.dist = "std",
##     trace = F)
##
## Mean and Variance Equation:
##  data ~ garch(1, 1)
## <environment: 0x55619a5fb0d8>
##  [data = intel]
##
## Conditional Distribution:
##  std
##
## Coefficient(s):
##        mu      omega     alpha1      beta1      shape
## 0.0165076  0.0011576  0.1059029  0.8171298  6.7723926
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu     0.0165076   0.0051031    3.235 0.001217 **
## omega  0.0011576   0.0005782    2.002 0.045287 *
## alpha1 0.1059029   0.0372046    2.846 0.004420 **
## beta1  0.8171298   0.0580150   14.085  < 2e-16 ***
## shape  6.7723926   1.8572657    3.646 0.000266 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  326.2264    normalized:  0.734744
##
## Description:
##  Thu Oct 15 14:06:06 2020 by user: kevin
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  203.4936  0
##  Shapiro-Wilk Test  R    W      0.9687607 3.970513e-08
##  Ljung-Box Test     R    Q(10)  7.877796  0.6407723
##  Ljung-Box Test     R    Q(15)  15.55225  0.4124162
##  Ljung-Box Test     R    Q(20)  16.5048   0.6848548
##  Ljung-Box Test     R^2  Q(10)  1.066053  0.9997694
##  Ljung-Box Test     R^2  Q(15)  11.49886  0.7164967
##  Ljung-Box Test     R^2  Q(20)  12.61507  0.8932826
##  LM Arch Test       R    TR^2   10.80749  0.5454849
##
## Information Criterion Statistics:
```

```
##       AIC       BIC       SIC      HQIC
## -1.446966 -1.400841 -1.447215 -1.428776
```

```
m4 <- garchFit(
  intel ~ garch(1, 1),
  data = intel,
  cond.dist = "sstd",
  trace = F
)
summary(m4)
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = intel ~ garch(1, 1), data = intel, cond.dist = "sstd",
##     trace = F)
##
## Mean and Variance Equation:
##  data ~ garch(1, 1)
## <environment: 0x55619bddb668>
##  [data = intel]
##
## Conditional Distribution:
##  sstd
##
## Coefficient(s):
##        mu      omega     alpha1      beta1       skew
## 0.0133343  0.0011621  0.1049294  0.8177869  0.8717222
##     shape
## 7.2344212
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu     0.0133343   0.0053430    2.496 0.012573 *
## omega  0.0011621   0.0005587    2.080 0.037520 *
## alpha1 0.1049294   0.0358862    2.924 0.003456 **
## beta1  0.8177869   0.0559866   14.607  < 2e-16 ***
## skew   0.8717222   0.0629130   13.856  < 2e-16 ***
## shape  7.2344212   2.1018137    3.442 0.000577 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  328.0995    normalized:  0.7389628
##
## Description:
##  Thu Oct 15 14:06:06 2020 by user: kevin
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  195.2181  0
##  Shapiro-Wilk Test  R    W      0.9692506 4.892633e-08
##  Ljung-Box Test     R    Q(10)  7.882124  0.6403498
##  Ljung-Box Test     R    Q(15)  15.62496  0.4074053
##  Ljung-Box Test     R    Q(20)  16.57741  0.6802192
##  Ljung-Box Test     R^2  Q(10)  1.078434  0.9997569
##  Ljung-Box Test     R^2  Q(15)  11.95157  0.6826911
##  Ljung-Box Test     R^2  Q(20)  13.03793  0.8757507
##  LM Arch Test       R    TR^2   11.18828  0.5128558
```

```
##
## Information Criterion Statistics:
##       AIC       BIC       SIC      HQIC
## -1.450899 -1.395550 -1.451257 -1.429071
```

where we note that all the coefficients appear significant and the *Q*-statistic is also acceptable.

3 IGarch Model - Intel Share Price

The sixth example considers the use of an IGARCH model, which is applied to data for the Intel share price. This example is contained in the file `tut8g-Igarch.R`. To start off we can clear all the variables from the current environment and close all the plots.

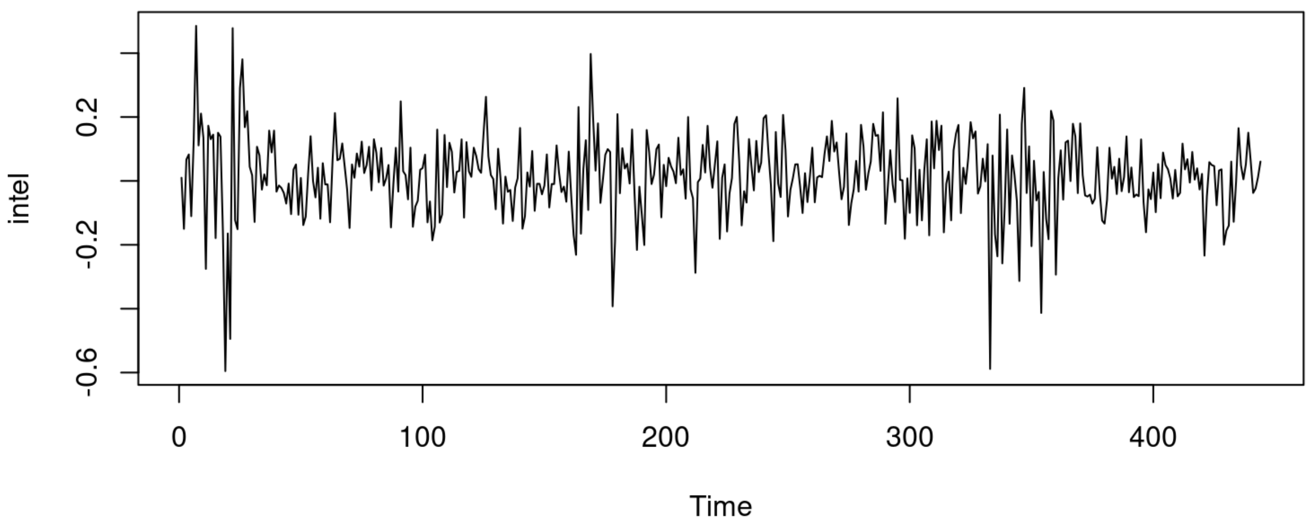
```
rm(list = ls())
graphics.off()
```

Thereafter, we will need to make use of the `tsm` package, so we make use of the `library` command.

```
library(tidyverse)
library(tsm)
```

As the data is contained in a `.csv` file, which should be in the same folder as `T8-uvol.Rproj` we would need to read this data into memory. After loading the data we create a time series object for the monthly logarithmic returns of the Intel share price.

```
dat <- read_csv(file = "intel.csv")
intel <- log(dat$intc + 1)
plot.ts(intel)
```



To model this data we make use of the `Igarch` command.

```
mm <- Igarch(intel, include.mean = T)
```

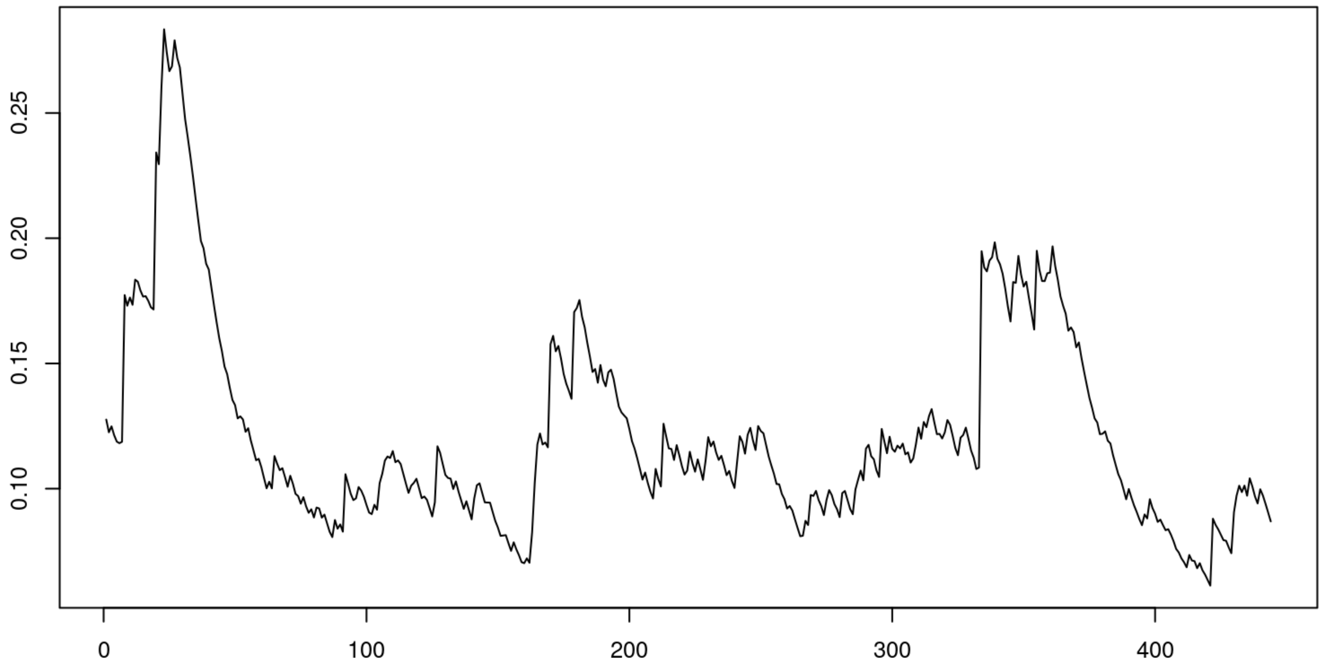
```
## Estimates:  0.009393095 0.9269987
## Maximized log-likehood:  -302.8558
##
## Coefficient(s):
##        Estimate  Std. Error  t value Pr(>|t|)
## mu   0.00939309  0.00555181   1.6919 0.090665 .
## beta 0.92699865  0.01436919  64.5130  < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
mm <- Igarch(intel)
```

```
## Estimates:  0.9217433
## Maximized log-likehood:  -301.412
##
## Coefficient(s):
##       Estimate  Std. Error  t value   Pr(>|t|)
## beta 0.9217433   0.0155534  59.2633 < 2.22e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We can then inspect the residuals with the aid of the command:

```
par(mfcol = c(1, 1),
    mar = c(2.2, 2.2, 1, 2.2),
    cex = 0.8)
plot(mm$volatility)
```



4 Garch-in-Mean Model - S&P500

The third example considers the use of an GARCH-in-Mean model, which is applied to data for the S&P500 index. This example is contained in the file `tut8c-garchM.R` . To start off we can clear all the variables from the current environment and close all the plots.

```
rm(list = ls())
graphics.off()
```

Thereafter, we will need to make use of the `tidyverse` and `tsm` packages, so we make use of the `library` command.

```
library(tidyverse)
library(tsm)
```

This allows us to load the data and create the time series objects for the S&P500 monthly excess returns between 1926 and 1991.

```
dat <- read_csv(file = "sp500.csv")
sp500 <- ts(dat$sp500, start = c(1926, 1), freq = 12)
```

To model this data we make use of the command, where `type=1` refers to the typical GARCH-in-Mean model:

```
GARCHmean = garchM(as.numeric(sp500) * 100, 1)
```

```
##   0:     2380.0229: 0.422452 0.00561296 0.806146 0.121976 0.854361
##   3:     2379.9746: 0.426034 0.00625022 0.806125 0.121755 0.855528
##   6:     2379.9185: 0.429737 0.00690734 0.806040 0.120752 0.855267
##   9:     2379.8696: 0.433530 0.00757671 0.805997 0.120872 0.856124
##  12:     2379.8151: 0.437366 0.00825132 0.805897 0.120190 0.855694
##  15:     2379.7664: 0.441241 0.00892840 0.805842 0.120504 0.856364
##  18:     2379.7131: 0.445135 0.00960611 0.805735 0.119993 0.855842
##  21:     2379.6649: 0.449054 0.0102834 0.805673 0.120392 0.856418
##  24:     2379.6129: 0.452988 0.0109598 0.805562 0.119963 0.855864
##  27:     2379.5652: 0.456943 0.0116348 0.805496 0.120390 0.856392
##  30:     2379.5144: 0.460911 0.0123078 0.805383 0.120000 0.855837
##  33:     2379.4675: 0.464898 0.0129788 0.805314 0.120428 0.856337
##  36:     2379.4180: 0.468898 0.0136467 0.805200 0.120059 0.855792
##  39:     2379.3718: 0.472915 0.0143119 0.805130 0.120479 0.856274
##  42:     2379.3236: 0.476946 0.0149727 0.805015 0.120126 0.855743
##  45:     2379.2783: 0.480993 0.0156300 0.804943 0.120535 0.856209
##  48:     2379.2314: 0.485053 0.0162816 0.804828 0.120194 0.855694
##  51:     2379.1871: 0.489128 0.0169286 0.804753 0.120591 0.856144
##  54:     2379.1415: 0.493216 0.0175682 0.804637 0.120263 0.855645
##  57:     2379.0982: 0.497319 0.0182020 0.804561 0.120649 0.856080
##  60:     2379.0540: 0.501435 0.0188265 0.804444 0.120333 0.855597
##  63:     2379.0117: 0.505564 0.0194437 0.804366 0.120707 0.856017
##  66:     2378.9689: 0.509706 0.0200494 0.804249 0.120403 0.855549
##  69:     2378.9277: 0.513862 0.0206461 0.804169 0.120766 0.855955
##  72:     2378.8863: 0.518029 0.0212289 0.804051 0.120474 0.855502
##  75:     2378.8463: 0.522211 0.0218007 0.803970 0.120825 0.855893
##  78:     2378.8063: 0.526403 0.0223564 0.803852 0.120545 0.855456
##  81:     2378.7676: 0.530610 0.0228991 0.803769 0.120884 0.855832
##  84:     2378.7290: 0.534827 0.0234238 0.803651 0.120617 0.855411
##  87:     2378.6915: 0.539058 0.0239340 0.803567 0.120944 0.855773
##  90:     2378.6543: 0.543299 0.0244253 0.803449 0.120688 0.855367
##  93:     2378.6182: 0.547553 0.0249018 0.803365 0.121004 0.855713
##  96:     2377.8108: 0.713437 0.0397951 0.799506 0.121533 0.855187
## Maximized log-likehood:  2377.81
## 
## Coefficient(s):
##        Estimate  Std. Error  t value   Pr(>|t|)    
## mu    0.7135637   0.1541249  4.62978 3.6606e-06 ***
## gamma 0.0398211   0.1408198  0.28278  0.7773453    
## omega 0.7995033   0.2832766  2.82234  0.0047674 ** 
## alpha 0.1215848   0.0219878  5.52964 3.2089e-08 ***
## beta  0.8550982   0.0217377 39.33705 < 2.22e-16 ***
## ---
## Signif. codes:  
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In this case the coefficient names are as follows: `mu` is the mean in the mean equation, `gamma` refers to the relationship between mean and the measure of volatility (which may be termed the risk-premium), `omega` is the constant in the volatility equation, `alpha` refers to the ARCH(1) coefficient and `beta` is the coefficient for the GARCH(1) term. Note that gamma is insignificant, which would suggest that there is no risk-premium.

5 GJR-Garch Model - Intel and IBM Share Price

The fourth example considers the use of a threshold GARCH model, which is applied to data for the Intel share price. This example is contained in the file `tut8d-GJR.R`. To start off we can clear all the variables from the current environment and close all the plots.

```
rm(list = ls())
graphics.off()
```

Thereafter, we will need to make use of the `tsm` package, so we make use of the `library` command.

```
library(tidyverse)
library(fGarch)
```

We can then load the data and create the time series objects for the Intel logarithmic returns.

```
dat <- read_csv(file = "intel.csv")
intel <- ts((log(dat$intc + 1)), start = c(1973, 1), freq = 12)
```

To model this data we make use of an `aparch` model, where `delta=2` would refer to the GJR-GARCH model. In this case we make use of both a normal and students-*t* distribution for the errors.

```
mod.intel.n <-
  garchFit(
    ~ aparch(1, 1),
    data = intel,
    trace = F,
    delta = 2,
    include.delta = F
  )
summary(mod.intel.n)
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = ~aparch(1, 1), data = intel, delta = 2, include.delta = F,
##     trace = F)
##
## Mean and Variance Equation:
##  data ~ aparch(1, 1)
## <environment: 0x55619a507ec0>
##  [data = intel]
##
## Conditional Distribution:
##  norm
##
## Coefficient(s):
##          mu        omega       alpha1       gamma1
##  0.01205853   0.00081818   0.08965605  -0.09613431
##       beta1
##  0.85673683
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##          Estimate  Std. Error  t value Pr(>|t|)
## mu      0.0120585   0.0054562    2.210 0.027100 *
## omega   0.0008182   0.0003679    2.224 0.026160 *
## alpha1  0.0896561   0.0270726    3.312 0.000927 ***
## gamma1 -0.0961343   0.1198265   -0.802 0.422392
## beta1   0.8567368   0.0377735   22.681  < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  312.6275    normalized:  0.7041161
##
## Description:
##  Thu Oct 15 14:06:16 2020 by user: kevin
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  146.8649  0
##  Shapiro-Wilk Test  R    W      0.9733289 3.01177e-07
##  Ljung-Box Test     R    Q(10)  8.103162  0.6187608
##  Ljung-Box Test     R    Q(15)  15.7317   0.4001035
##  Ljung-Box Test     R    Q(20)  16.62709  0.6770382
##  Ljung-Box Test     R^2  Q(10)  0.7591985 0.9999521
##  Ljung-Box Test     R^2  Q(15)  11.67255  0.7036264
##  Ljung-Box Test     R^2  Q(20)  12.93898  0.8799866
##  LM Arch Test       R    TR^2   10.69648  0.5550934
##
```

```
## Information Criterion Statistics:
##       AIC       BIC       SIC      HQIC
## -1.385710 -1.339585 -1.385959 -1.367520
```

```
mod.intel.s <-
  garchFit(
    ~ aparch(1, 1),
    data = intel,
    trace = F,
    delta = 2,
    include.delta = F,
    cond.dist = "std"
  )
summary(mod.intel.s)
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = ~aparch(1, 1), data = intel, delta = 2, cond.dist = "std",
##     include.delta = F, trace = F)
##
## Mean and Variance Equation:
##  data ~ aparch(1, 1)
## <environment: 0x556196f18e40>
##  [data = intel]
##
## Conditional Distribution:
##  std
##
## Coefficient(s):
##        mu      omega     alpha1     gamma1      beta1
## 0.0162315  0.0012378  0.1029180  0.0692275  0.8127185
##     shape
## 6.7174193
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu     0.0162315   0.0051496    3.152 0.001622 **
## omega  0.0012378   0.0006262    1.977 0.048070 *
## alpha1 0.1029180   0.0374525    2.748 0.005997 **
## gamma1 0.0692275   0.1569562    0.441 0.659168
## beta1  0.8127185   0.0594964   13.660  < 2e-16 ***
## shape  6.7174193   1.8262030    3.678 0.000235 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  326.3315    normalized:  0.7349808
##
## Description:
##  Thu Oct 15 14:06:16 2020 by user: kevin
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  228.2263  0
##  Shapiro-Wilk Test  R    W      0.9669724 1.882283e-08
##  Ljung-Box Test     R    Q(10)  7.796906  0.6486672
##  Ljung-Box Test     R    Q(15)  15.38199  0.4242678
##  Ljung-Box Test     R    Q(20)  16.37238  0.6932704
##  Ljung-Box Test     R^2  Q(10)  1.169355  0.999649
##  Ljung-Box Test     R^2  Q(15)  11.13284  0.7431223
##  Ljung-Box Test     R^2  Q(20)  12.22264  0.9081998
##  LM Arch Test       R    TR^2   10.55074  0.5677607
```

```
##
## Information Criterion Statistics:
##       AIC       BIC       SIC      HQIC
## -1.442935 -1.387586 -1.443293 -1.421107
```

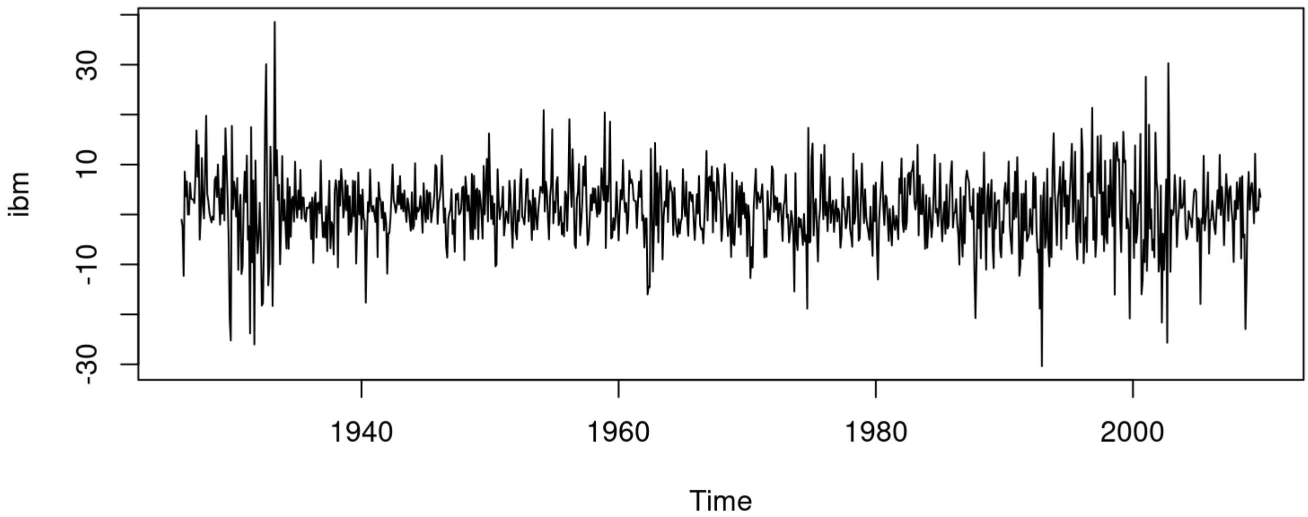
Note that as `gamma` is not significant it suggests that there are no significant leverage effects in this particular variable.

As an alternative we can make use of data for the monthly returns from the IBM share price.

```
rm(list = ls())
graphics.off()
```

We can then load the data.

```
dat <- read_csv(file = "ibm.csv")
ibm <- ts((log(dat$ibm + 1) * 100), start = c(1926, 1), freq = 12)
plot(ibm)
```



Thereafter, we can make use of similar model structures.

```
mod.ibm.n <-
  garchFit(
    ~ aparch(1, 1),
    data = ibm,
    trace = F,
    delta = 2,
    include.delta = F
  )
summary(mod.ibm.n)
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = ~aparch(1, 1), data = ibm, delta = 2, include.delta = F,
##     trace = F)
##
## Mean and Variance Equation:
##  data ~ aparch(1, 1)
## <environment: 0x55619f19aa20>
##  [data = ibm]
##
## Conditional Distribution:
##  norm
##
## Coefficient(s):
##      mu    omega   alpha1   gamma1    beta1
## 1.18657  4.33659  0.10767  0.22732  0.79467
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu       1.18657     0.20019    5.927 3.08e-09 ***
## omega    4.33659     1.34159    3.232  0.00123 **
## alpha1   0.10767     0.02548    4.225 2.39e-05 ***
## gamma1   0.22732     0.10018    2.269  0.02326 *
## beta1    0.79467     0.04554   17.450  < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  -3329.177    normalized:  -3.302755
##
## Description:
##  Thu Oct 15 14:06:16 2020 by user: kevin
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  67.07434  2.775558e-15
##  Shapiro-Wilk Test  R    W      0.9870135 8.591106e-08
##  Ljung-Box Test     R    Q(10)  16.90606  0.07646869
##  Ljung-Box Test     R    Q(15)  24.19033  0.06193097
##  Ljung-Box Test     R    Q(20)  31.89095  0.04447423
##  Ljung-Box Test     R^2  Q(10)  4.591701  0.9167336
##  Ljung-Box Test     R^2  Q(15)  11.98465  0.6801909
##  Ljung-Box Test     R^2  Q(20)  14.79516  0.788006
##  LM Arch Test       R    TR^2   7.162849  0.8466667
##
## Information Criterion Statistics:
```

```
##      AIC      BIC      SIC     HQIC
## 6.615430 6.639814 6.615381 6.624694
```

```
mod.ibm.s <-
  garchFit(
    ~ aparch(1, 1),
    data = ibm,
    trace = F,
    delta = 2,
    include.delta = F,
    cond.dist = "std"
  )
summary(mod.ibm.s)
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = ~aparch(1, 1), data = ibm, delta = 2, cond.dist = "std",
##     include.delta = F, trace = F)
##
## Mean and Variance Equation:
##  data ~ aparch(1, 1)
## <environment: 0x5561982caeb8>
##  [data = ibm]
##
## Conditional Distribution:
##  std
##
## Coefficient(s):
##      mu    omega   alpha1   gamma1    beta1    shape
## 1.20476  3.98976  0.10468  0.22366  0.80711  6.67329
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu       1.20476     0.18715    6.437 1.22e-10 ***
## omega    3.98976     1.45331    2.745 0.006046 **
## alpha1   0.10468     0.02793    3.747 0.000179 ***
## gamma1   0.22366     0.11595    1.929 0.053739 .
## beta1    0.80711     0.04825   16.727  < 2e-16 ***
## shape    6.67329     1.32779    5.026 5.01e-07 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  -3310.21    normalized:  -3.283938
##
## Description:
##  Thu Oct 15 14:06:16 2020 by user: kevin
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  67.82335  1.887379e-15
##  Shapiro-Wilk Test  R    W      0.9869701 8.215349e-08
##  Ljung-Box Test     R    Q(10)  16.91351  0.07629979
##  Ljung-Box Test     R    Q(15)  24.0869   0.06363241
##  Ljung-Box Test     R    Q(20)  31.75303  0.04600203
##  Ljung-Box Test     R^2  Q(10)  4.553244  0.9189586
##  Ljung-Box Test     R^2  Q(15)  11.66892  0.7038965
##  Ljung-Box Test     R^2  Q(20)  14.18533  0.8209762
##  LM Arch Test       R    TR^2   6.771675  0.872326
##
## Information Criterion Statistics:
```

```
##      AIC      BIC      SIC     HQIC
## 6.579782 6.609042 6.579711 6.590898
```

In this case the results suggest that there could be leverage effects in the data.

6 NGarch Model - USD/EU exchange rate data

The sixth example considers the use of an NGARCH model, which is applied to data for the USD/EU exchange rate. This example is contained in the file `tut8f-Ngarch.R`. To start off we can clear all the variables from the current environment and close all the plots.

```
rm(list = ls())
graphics.off()
```

Thereafter, we will need to make use of the `tsm` package, so we make use of the `library` command.

```
library(tidyverse)
library(tsm)
```

We can then load the data and create the time series objects for the monthly returns of the USD/EU exchange rate.

```
dat <- read_csv(file = "ex_usd_eu.csv")
fx <- log(dat$rate)
eu <- diff(fx) * 100
```

To model this data we make use of the `Ngarch` command.

```
m1 <- Ngarch(eu)
```

```
##
## Estimation results of NGARCH(1,1) model:
## estimates:  -0.001094043 0.002366721 0.9618047 0.02118565 0.7309616
## std.errors:  0.01080893 0.000580552 0.006045803 0.003604727 0.2501548
## t-ratio:  -0.1012166 4.076674 159.0863 5.877186 2.922037
```

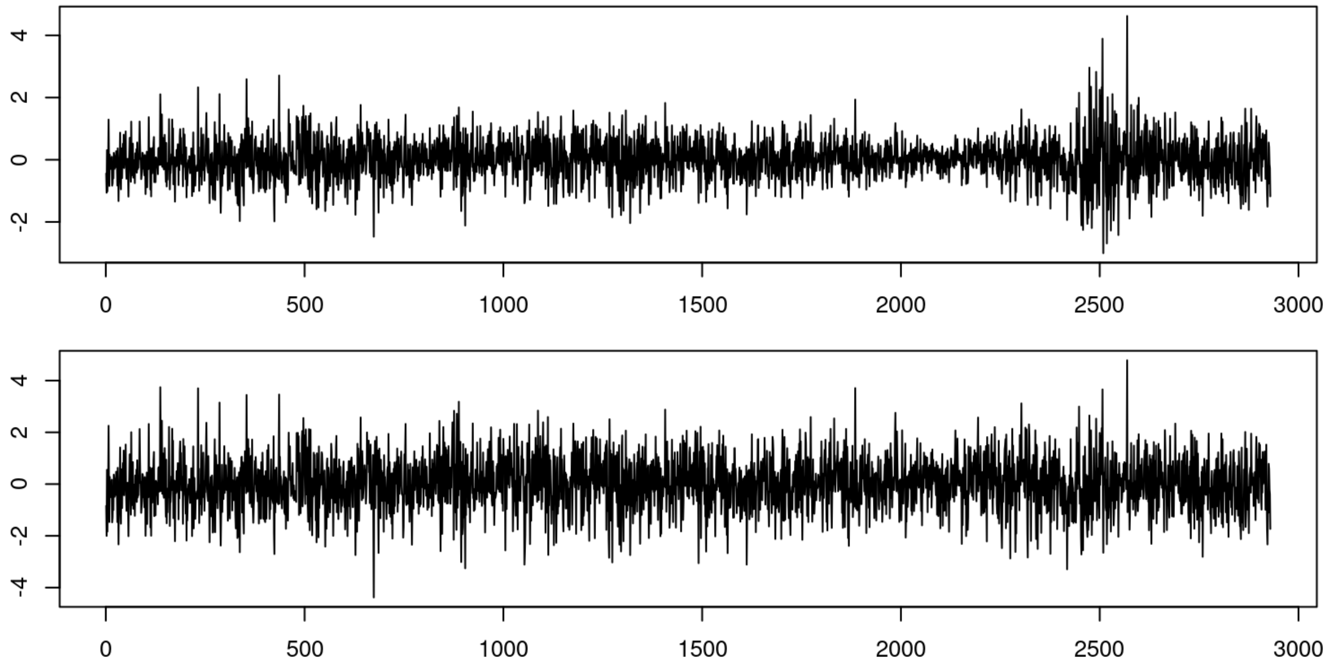
```
names(m1)
```

```
## [1] "residuals"  "volatility"
```

```
res <- m1$residuals
vol <- m1$volatility
resi <- res/vol
```

We can then compare the residuals to the actual data, with the aid of the following commands:

```
par(mfcol = c(2, 1),
    mar = c(2.2, 2.2, 1, 2.2),
    cex = 0.8)
plot.ts(eu)
plot.ts(resi)
```



Thereafter, we can check for remaining serial correlation in the residuals.

```
Box.test(resi,lag = 10, type = 'Ljung')
```

```
##
##  Box-Ljung test
##
## data:  resi
## X-squared = 14.776, df = 10, p-value = 0.1404
```

```
Box.test(abs(resi), lag = 10, type = 'Ljung')
```

```
##
##  Box-Ljung test
##
## data:  abs(resi)
## X-squared = 19.632, df = 10, p-value = 0.03293
```

While there would appear to be no serial correlation in the mean, the results for the volatility equation could be better.

7 Stochastic Volatility Model

The last example considers the use of an Stochastic Volatility model, which is applied to data for the USD/ZAR exchange rate. This example is contained in the file `tut8i-SV.R`. To start off we can clear all the variables from the current environment and close all the plots.

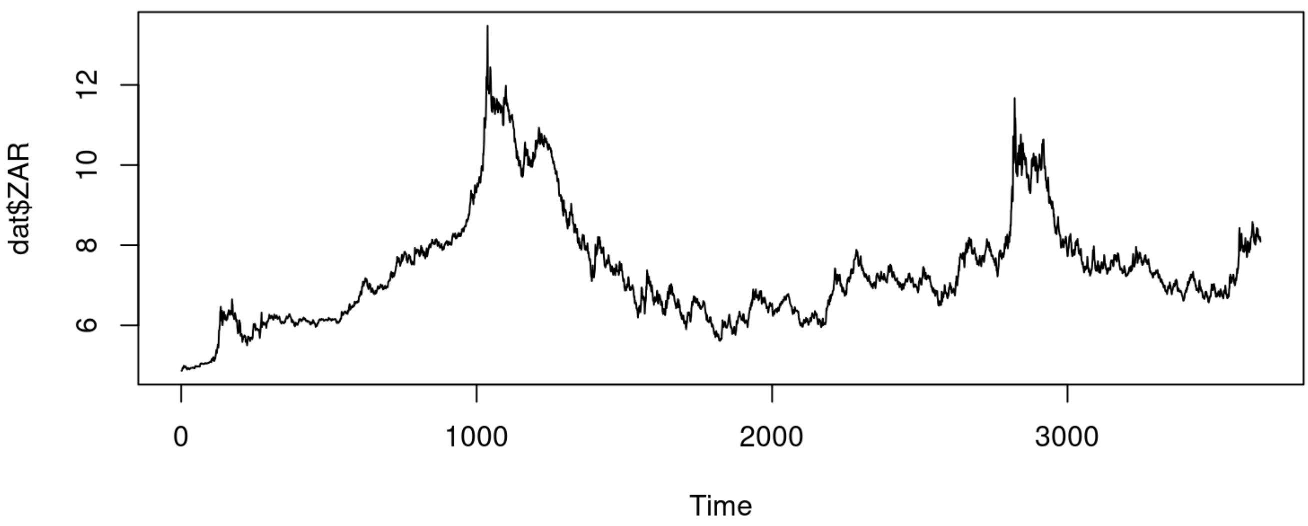
```
rm(list = ls())
graphics.off()
```

Thereafter, we will need to make use of the `tidyverse` and `stochvol` packages, so we make use of the `library` command.

```
library(tidyverse)
library(stochvol)
```

We can then load the data and create the time series objects for the monthly returns of the USD/ZAR exchange rate. In this case we are going to use the model to forecast future exchange rate, so we make use of a reduced insample period.

```
dat <- read_csv(file = "ex_data.csv")
plot.ts(dat$ZAR)
```



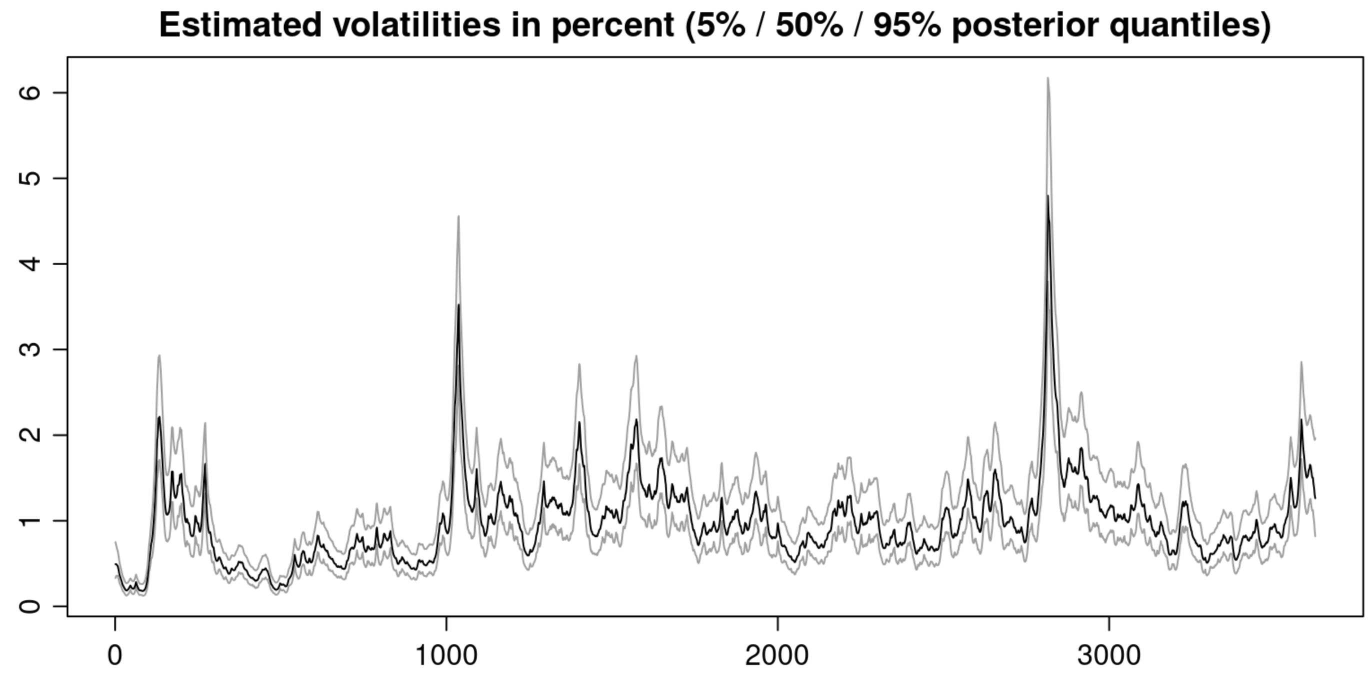

```
zar.logret <- diff(log(dat$ZAR))
zar.dat <- zar.logret - mean(zar.logret)
zar.ins <- zar.dat[1:3622]
zar.out <- zar.dat[3623:3652]
```

To model this data we make use of the `svsample` command, which executes the MCMC sampler for the SV model.

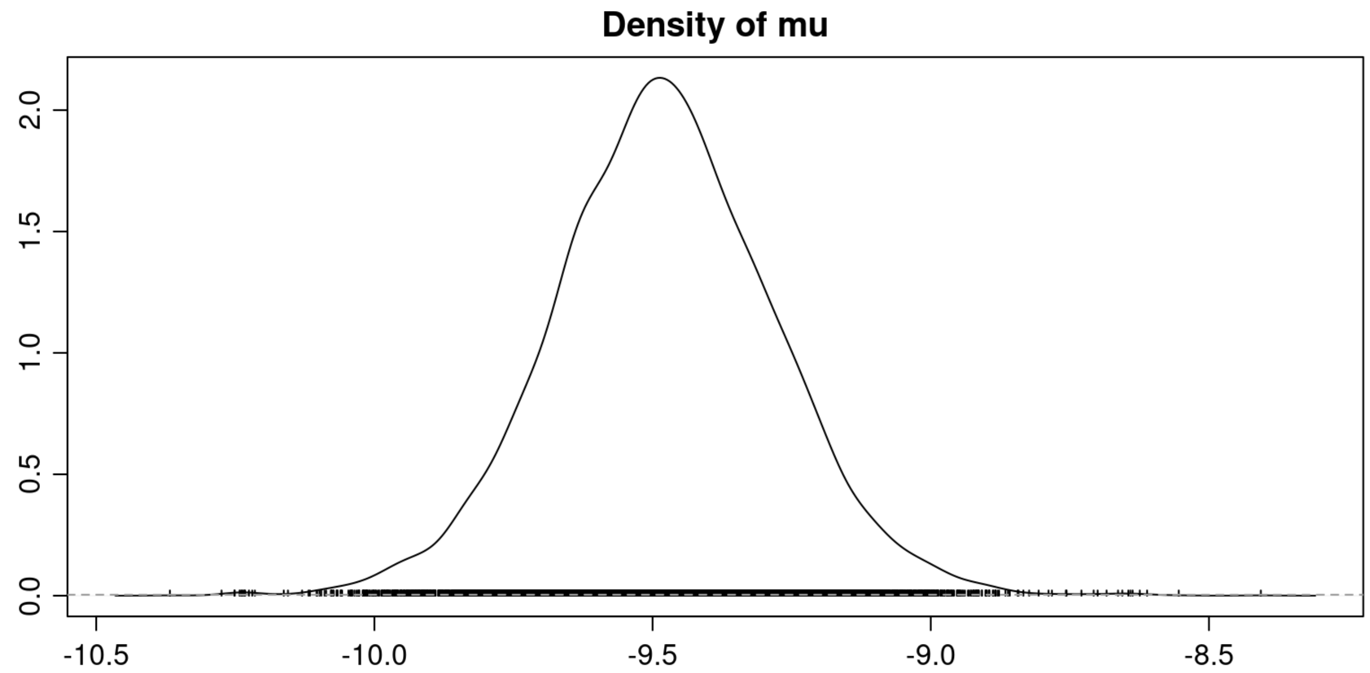
```
res.sv <- svsample(zar.ins)
```

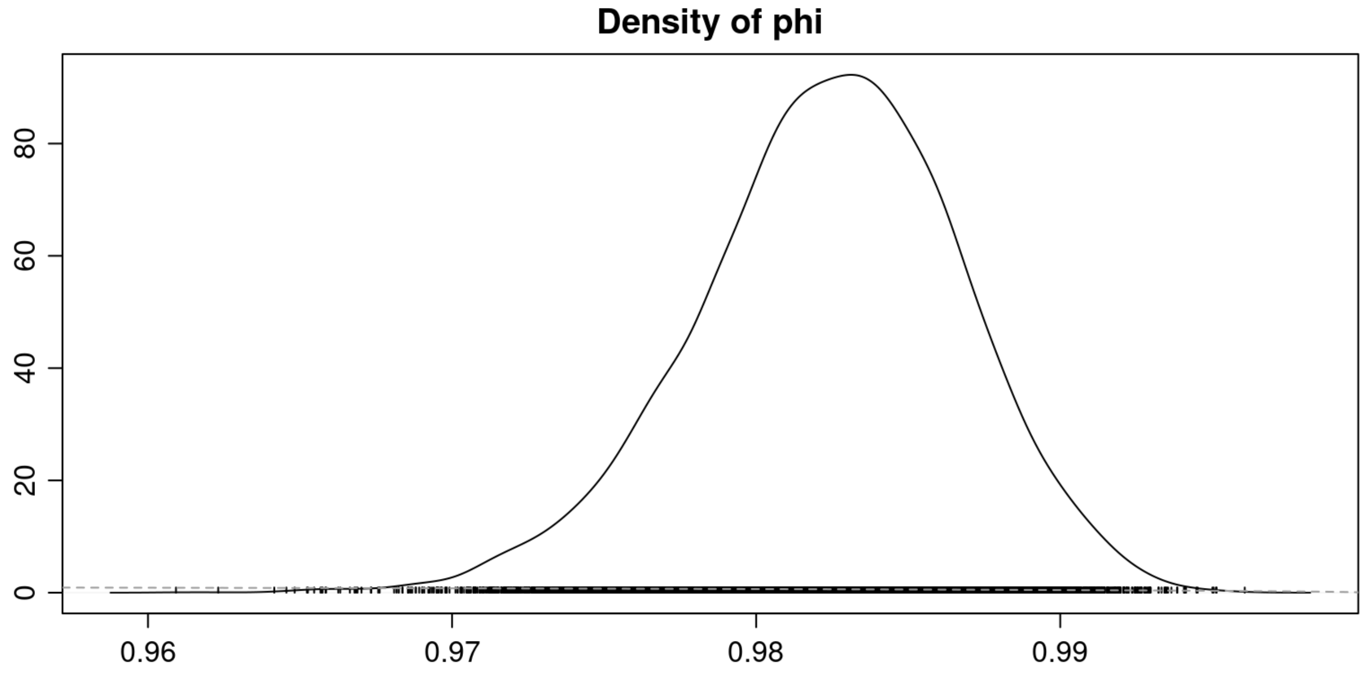
We can then consider the description of volatility and the posterior densities for the parameters:

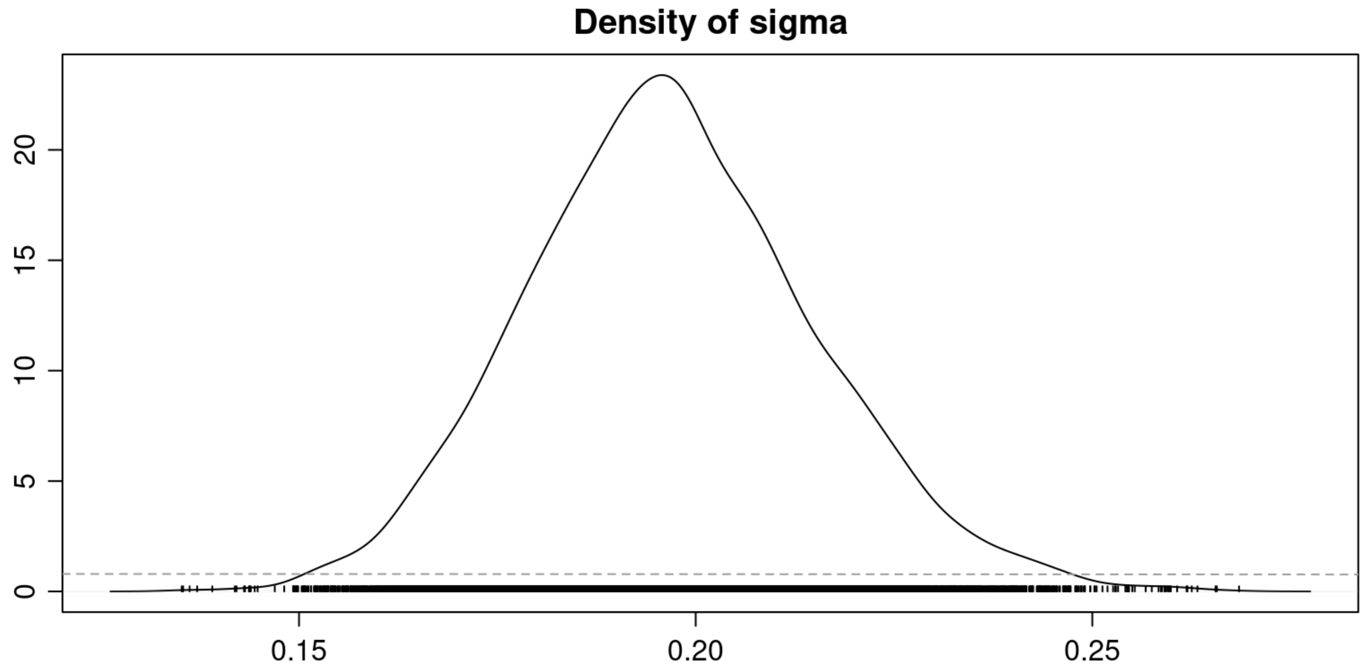
```
volplot(res.sv)
```




```
paradensplot(res.sv)
```








```
summary(res.sv)
```

```
##
## Summary of 10000 MCMC draws after a burn-in of 1000.
## Prior distributions:
## mu        ~ Normal(mean = 0, sd = 100)
## (phi+1)/2 ~ Beta(a0 = 5, b0 = 1.5)
## sigma^2   ~ 1 * Chisq(df = 1)
##
## Posterior draws of parameters (thinning = 1):
##               mean      sd      5%     50%    95%  ESS
## mu        -9.4808 0.19914 -9.8050 -9.4824 -9.157 6009
## phi         0.9822 0.00442  0.9746  0.9824  0.989  261
## sigma       0.1977 0.01873  0.1672  0.1973  0.229  137
## exp(mu/2)  0.0088 0.00088  0.0074  0.0087  0.010 6009
## sigma^2    0.0394 0.00746  0.0280  0.0389  0.052  137
##
## Posterior draws of initial and contemporaneous latents (thinning = 1):
##         mean   sd    5%   50%   95% mean(exp(h_t/2))
## h_0    -10.6 0.52 -11.4 -10.6  -9.7           0.0052
## h_1    -10.6 0.49 -11.4 -10.6  -9.8           0.0051
## h_2    -10.6 0.46 -11.4 -10.6  -9.8           0.0051
## h_3    -10.6 0.43 -11.3 -10.6  -9.9           0.0051
## h_4    -10.6 0.41 -11.3 -10.6  -9.9           0.0051
## h_5    -10.6 0.40 -11.3 -10.7 -10.0           0.0050
## h_6    -10.7 0.39 -11.3 -10.7 -10.0           0.0049
## h_7    -10.7 0.38 -11.3 -10.7 -10.1           0.0049
## h_8    -10.7 0.36 -11.3 -10.7 -10.1           0.0048
## h_9    -10.8 0.36 -11.4 -10.8 -10.2           0.0046
## h_10   -10.9 0.37 -11.5 -10.9 -10.2           0.0044
## h_11   -10.9 0.37 -11.5 -11.0 -10.3           0.0043
## h_12   -11.1 0.38 -11.7 -11.1 -10.4           0.0040
## h_13   -11.2 0.39 -11.8 -11.2 -10.5           0.0038
##        sd(exp(h_t/2))
## h_0           0.00140
## h_1           0.00130
## h_2           0.00122
## h_3           0.00113
## h_4           0.00108
## h_5           0.00104
## h_6           0.00098
## h_7           0.00094
## h_8           0.00090
## h_9           0.00087
## h_10          0.00084
## h_11          0.00081
## h_12          0.00079
## h_13          0.00077
##  [ reached getOption("max.print") -- omitted 3609 rows ]
```

To forecast forward over thirty steps, we make use of the commands:

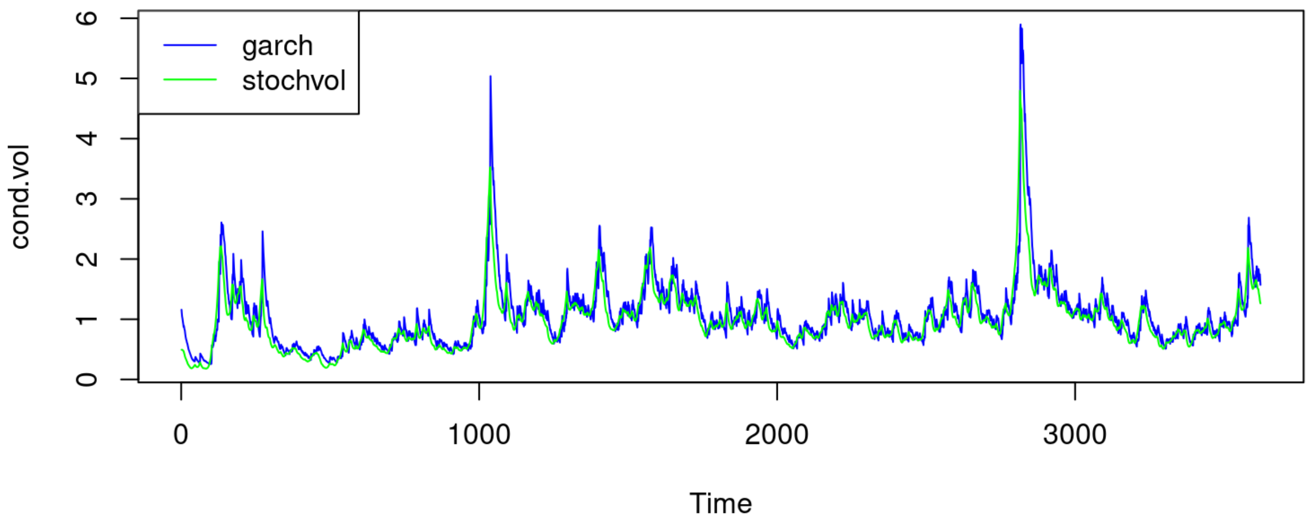
```
sv.pred <- predict(res.sv, 30)
sv.fore <-
  t(matrix(apply(
    100 * exp(sv.pred$h / 2), 2, quantile, c(0.05, 0.5, 0.95)
  ), nrow = 3))
```

To compare these forecasting results to those of the GARCH(1,1) model we could make use of the following code:

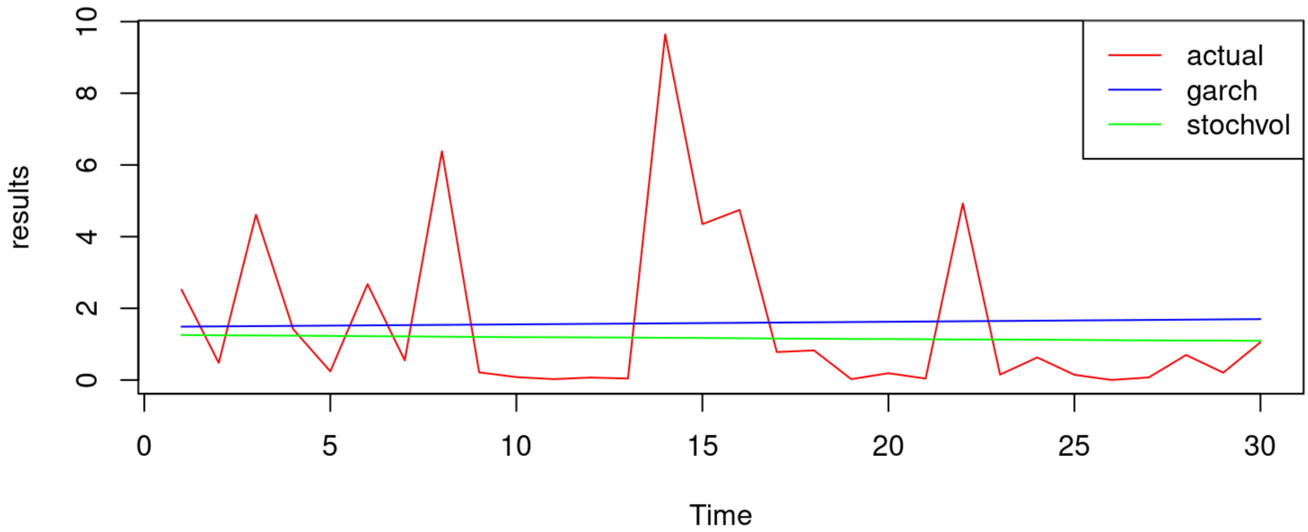
```
library(fGarch)

res.garch <- garchFit(zar.ins ~ garch(1, 1), data = zar.ins, trace = F)
summary(res.garch)
```

```
##
## Title:
##  GARCH Modelling
##
## Call:
##  garchFit(formula = zar.ins ~ garch(1, 1), data = zar.ins, trace = F)
##
## Mean and Variance Equation:
##  data ~ garch(1, 1)
## <environment: 0x556196439890>
##  [data = zar.ins]
##
## Conditional Distribution:
##  norm
##
## Coefficient(s):
##         mu       omega      alpha1       beta1
## 5.0531e-06  4.9620e-07  1.1217e-01  8.9495e-01
##
## Std. Errors:
##  based on Hessian
##
## Error Analysis:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu     5.053e-06   1.236e-04    0.041    0.967
## omega  4.962e-07   1.262e-07    3.933 8.38e-05 ***
## alpha1 1.122e-01   9.738e-03   11.518  < 2e-16 ***
## beta1  8.950e-01   8.031e-03  111.435  < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Log Likelihood:
##  11710.94    normalized:  3.23328
##
## Description:
##  Thu Oct 15 14:06:42 2020 by user: kevin
##
##
## Standardised Residuals Tests:
##                                 Statistic p-Value
##  Jarque-Bera Test   R    Chi^2  710.2088  0
##  Shapiro-Wilk Test  R    W      0.9810006 0
##  Ljung-Box Test     R    Q(10)  14.77983  0.1402974
##  Ljung-Box Test     R    Q(15)  23.66422  0.07101944
##  Ljung-Box Test     R    Q(20)  26.06154  0.1637891
##  Ljung-Box Test     R^2  Q(10)  4.012294  0.9467907
##  Ljung-Box Test     R^2  Q(15)  7.031698  0.9567642
##  Ljung-Box Test     R^2  Q(20)  8.924805  0.9837619
##  LM Arch Test       R    TR^2   4.722974  0.9665976
##
## Information Criterion Statistics:
##       AIC       BIC       SIC      HQIC
## -6.464352 -6.457510 -6.464354 -6.461914
```

```
garch.h_t = 100 * (res.garch@sigma.t)
sv.h_t <- (100 * exp(res.sv$summary$latent[, 3:5] / 2))

cond.vol <- cbind(garch.h_t, sv.h_t[, 2])

plot.ts(cond.vol,
        plot.type = "single",
        col = c("blue", "green"))
legend(
  "topleft",
  legend = c("garch", "stochvol"),
  lty = 1,
  col = c("blue", "green")
)
```




```
garch.pred <- predict(res.garch, n.ahead = 30, crit_val = 2)
garch.fore = 100 * (garch.pred)

results <- cbind((100 * zar.out) ^ 2, garch.fore[, 3], sv.fore[, 2])
plot.ts(results,
        plot.type = "single",
        col = c("red", "blue", "green"))
legend(
  "topright",
  legend = c("actual", "garch", "stochvol"),
  lty = 1,
  col = c("red", "blue", "green")
)
```



Where we note that in both cases, the models provide a poor result.