ARCH-GARCH Example with R

I. Ozkan

Saturday, April 26, 2014

ARCH-GARCH Example with BIST, Oil and TL/USD Series

The aim of this tutorial is to introduce ARCH-GARCH modelling in R. To do so, real life data sets are used. These sets are, Oil, BIST100 index and TL/USD Fx series. There are two parts of this tutorial. First part is to show how to import data sets from from csv files. This part also demonstrates some data manipulation steps necessary before modelling. Basic exploratory analysis and modelling are introduced in the second part of this tutorial.

Reading csv Files

Files that are used posted in http://yunus.hacettepe.edu.tr/~iozkan/data/ (http://yunus.hacettepe.edu.tr/~iozkan/data/) directory. We need to import these data sets and convert them into appropriate object type before modelling.

```
oil<-read.csv("http://yunus.hacettepe.edu.tr/~iozkan/data/oilcsv.csv", header=T, sep=";")

head(oil)
```

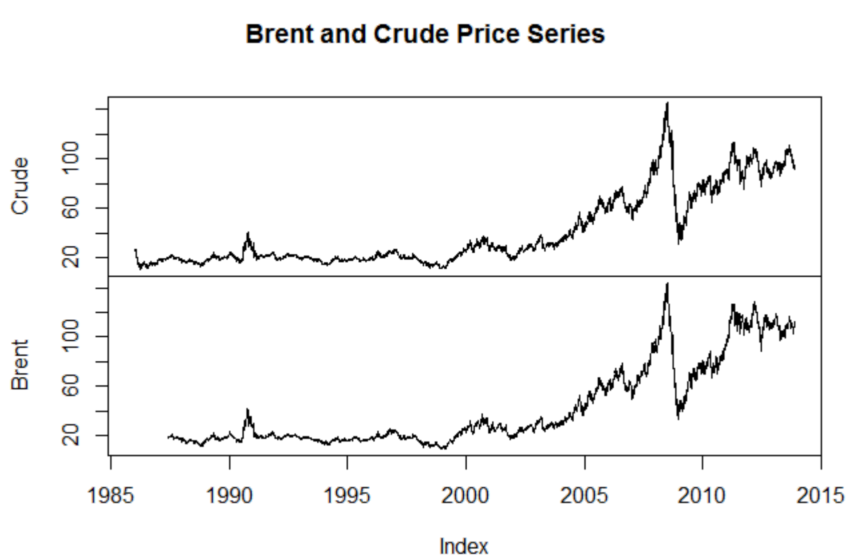
```
##         Date Crude Brent  X
## 1 02.01.1986 25.56    NA NA
## 2 03.01.1986 26.00    NA NA
## 3 06.01.1986 26.53    NA NA
## 4 07.01.1986 25.85    NA NA
## 5 08.01.1986 25.87    NA NA
## 6 09.01.1986 26.03    NA NA
```

```
tail(oil)
```

```
##            Date Crude Brent  X
## 7137 22.11.2013 94.53 111.4 NA
## 7138 25.11.2013 93.86 110.8 NA
## 7139 26.11.2013 93.41 112.0 NA
## 7140 27.11.2013 92.05 111.3 NA
## 7141 29.11.2013 92.55 111.1 NA
## 7142 02.12.2013 93.61 111.5 NA
```

It appears that there is an extra column containing only NAs (Not Available). Delete this column. Date is formatted as `dd.mm.yyyy` format. There are Brent and Crude series in the data. Lets delete column named `X` (4th column) and convert the oil series into `zoo` series. Then plot the series for the first inspection.

```
# install.packages("zoo")
library(zoo)
oil<-oil[,-4]
oil.zoo=zoo(oil[,-1], order.by=as.Date(strptime(as.character(oil[,1]), "%d.%m.%Y")))
plot(oil.zoo, main="Brent and Crude Price Series")
```



Let's import other series namely BIST100 index and TL/USD series.

```
xu100<-read.csv("http://yunus.hacettepe.edu.tr/~iozkan/data/endXU100.csv", header=T, sep=";")
head(xu100)
```

```
##   ENDEKS       DATE SESSION LOW HIGH CLOSE USD.BASED EURO.BASED
## 1  XU100 04.01.1988       0   0    0  6.89     393.5          0
## 2  XU100 05.01.1988       0   0    0  7.07     403.4          0
## 3  XU100 06.01.1988       0   0    0  7.07     399.1          0
## 4  XU100 07.01.1988       0   0    0  7.03     392.3          0
## 5  XU100 08.01.1988       0   0    0  6.96     387.3          0
## 6  XU100 11.01.1988       0   0    0  7.03     388.4          0
```

```
tail(xu100)
```

```
##       ENDEKS       DATE SESSION   LOW  HIGH CLOSE USD.BASED EURO.BASED
## 11158  XU100 04.12.2013       1 72356 72936 72672      2069       1785
## 11159  XU100 04.12.2013       2 72016 73356 73089      2081       1795
## 11160  XU100 05.12.2013       1 72454 73174 72849      2071       1785
## 11161  XU100 05.12.2013       2 71992 73032 71992      2047       1764
## 11162  XU100 06.12.2013       1 71778 72235 72070      2058       1765
## 11163  XU100 06.12.2013       2 71612 73538 73378      2095       1797
```

```
usd<-read.csv("http://yunus.hacettepe.edu.tr/~iozkan/data/usd.csv", header=T, sep=";")
head(usd)
```

```
##         DATE   USD
## 1 02.01.1950 3e-06
## 2 03.01.1950 3e-06
## 3 04.01.1950 3e-06
## 4 05.01.1950 3e-06
## 5 06.01.1950 3e-06
## 6 07.01.1950    NA
```

```
tail(usd)
```

```
##             DATE   USD
## 23348 04.12.2013 2.039
## 23349 05.12.2013 2.049
## 23350 06.12.2013 2.052
## 23351 07.12.2013    NA
## 23352 08.12.2013    NA
## 23353 09.12.2013 2.043
```

Date formats appear to be the same as the Date format of oil series. BIST index contains some columns that we are not going to use. These are `ENDEKS and SESSION` columns. But the tail of index data shows that after a certain day stock exchange reports of session's closing values. To find the date when session's closing values reported, we can check first observations of session values 1 or 2, or last observation where session value is 0. Stock exchange provides each session closing values starting January first 1995.

```
head(xu100[xu100[,"SESSION"]==1,], 2)
```

```
##      ENDEKS       DATE SESSION   LOW  HIGH CLOSE USD.BASED EURO.BASED
## 1753  XU100 02.01.1995       1 265.1 272.6 265.2     399.3          0
## 1755  XU100 03.01.1995       1 250.2 257.2 254.2     371.1          0
```

```
head(xu100[xu100[,"SESSION"]==2,], 2)
```

```
##      ENDEKS       DATE SESSION   LOW  HIGH CLOSE USD.BASED EURO.BASED
## 1754  XU100 02.01.1995       2 249.1 265.2 250.8     377.6          0
## 1756  XU100 03.01.1995       2 254.2 261.6 260.8     380.8          0
```

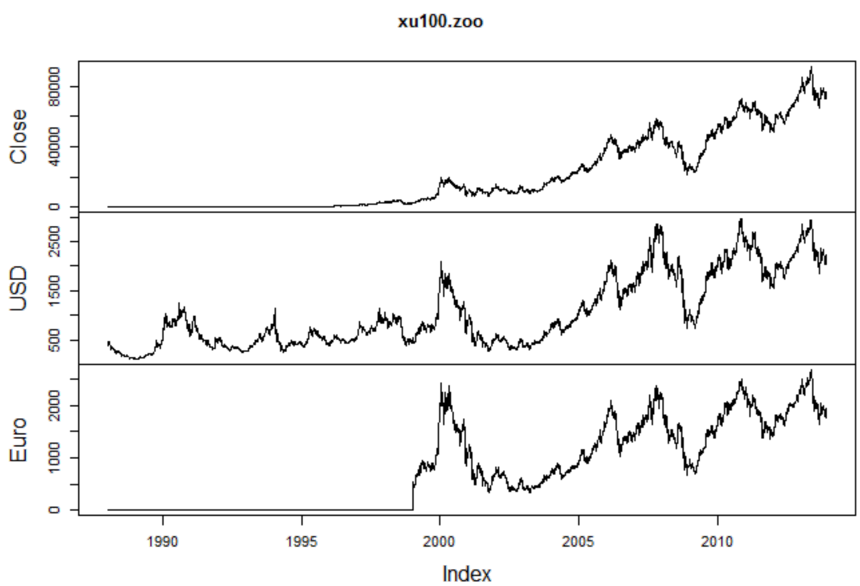
```
tail(xu100[xu100[,"SESSION"]==0,], 2)
```

```
##      ENDEKS       DATE SESSION LOW HIGH CLOSE USD.BASED EURO.BASED
## 1751  XU100 29.12.1994       0   0    0 277.9     422.1          0
## 1752  XU100 30.12.1994       0   0    0 272.6     413.3          0
```

Lets convert both index and TL/USD series into zoo and perform inital steps to clean the data. Only day's closing index values will be used in this tutorial. In addition, some columns will be deleted since we do not need them.

The TL/USD series contains weekend data as NA. These data values should be deleted. However we need to delete only weekend data in order to have the Fx series to be NA for national holidays. Here is the steps of performing this using `chron` package.

```r
#install.packages("chron")
library(chron)

xu100 <- xu100[xu100[,"SESSION"]!=1,-c(1,3,4,5)]

# now convert to zoo
xu100.zoo=zoo(xu100[,-1], order.by=as.Date(strptime(as.character(xu100[,1]), "%d.%m.%Y")))
colnames(xu100.zoo) <- c("Close", "USD", "Euro")

plot(xu100.zoo)
```



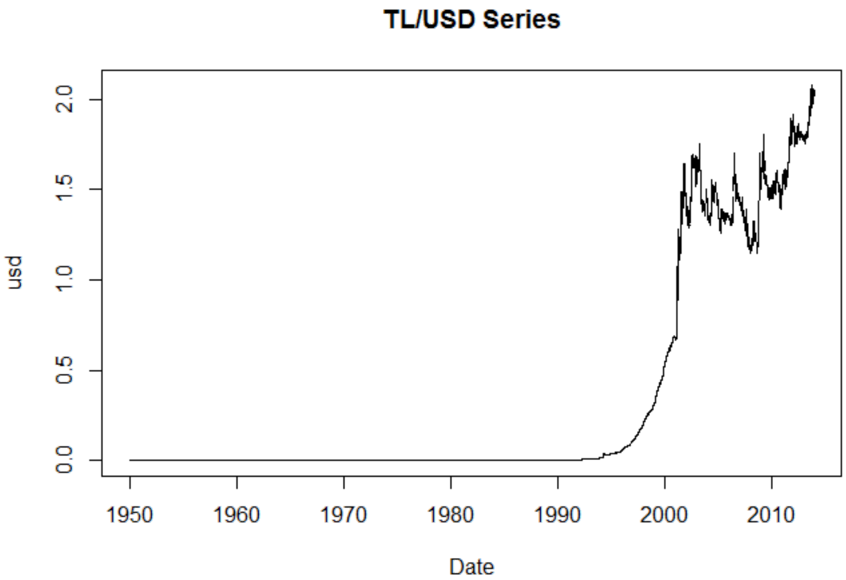

```r
usd.zoo=zoo(usd[,-1], order.by=as.Date(strptime(as.character(usd[,1]), "%d.%m.%Y")))

head(is.weekend(time(usd.zoo)))
```

```
## [1] FALSE FALSE FALSE FALSE FALSE  TRUE
```

```r
tail(is.weekend(time(usd.zoo)))
```

```
## [1] FALSE FALSE FALSE  TRUE  TRUE FALSE
```

```r
# usd2 contains weekdays obs.. others are holidays.. Do not delete..
usd <- usd.zoo[!is.weekend(time(usd.zoo))]
plot(usd, main="TL/USD Series", xlab="Date")
```

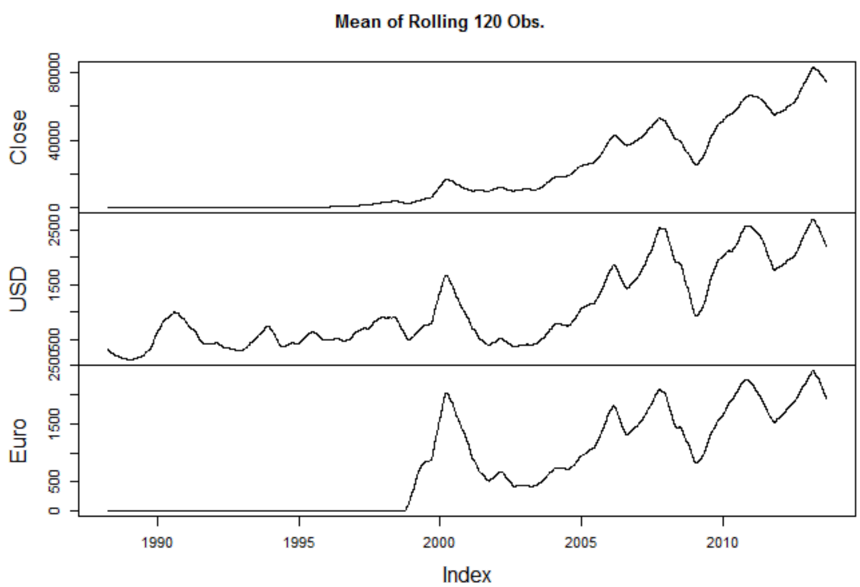


Some Useful Functions for Time Series

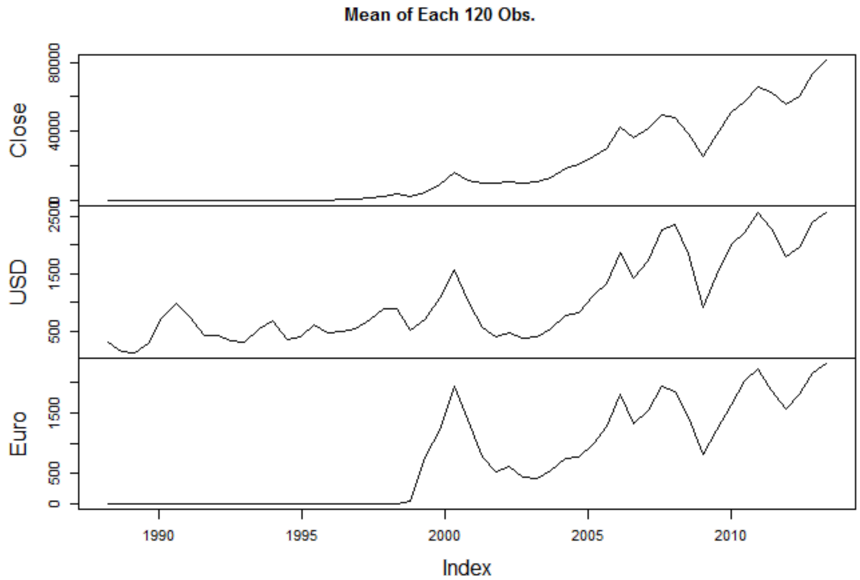
Rolling Window (Sliding Window) Analysis Example

Let's have fun with this data using `zoo` package function `rollapply()`. Lets plot the rolling window mean estimation for index data. In the first figure it is a sliding window and hence results in more smooth change in values. The second figure is obtained by estimation of mean values of sliding time windows that each has non-overlapped 120 observations.

```
# Overlapping rolling window means.. width of window is 120
plot(rollapply(xu100.zoo, width=120, mean, na.rm=T), main="Mean of Rolling 120 Obs.")
```

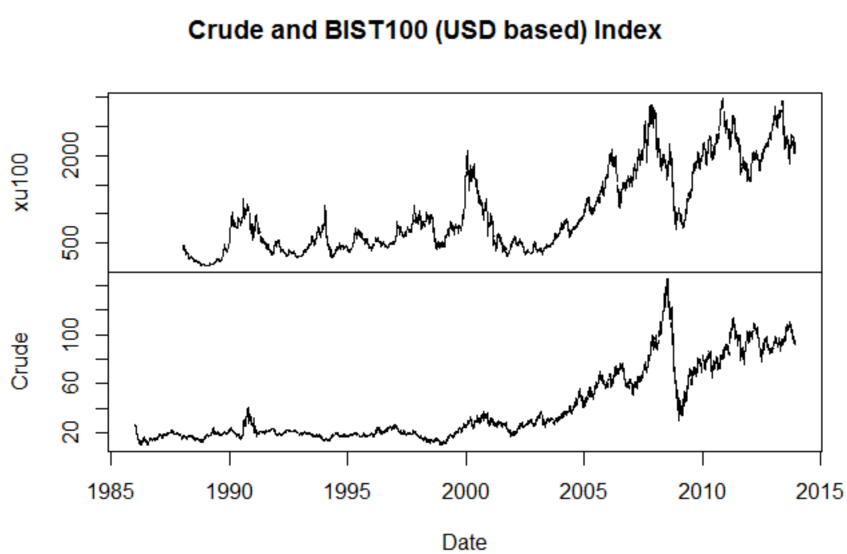



```
# Non-overlapping rolling window means.. width of window is 120
plot(rollapply(xu100.zoo, width=120, mean, by=120, na.rm=T),main="Mean of Each 120 Obs.")
```

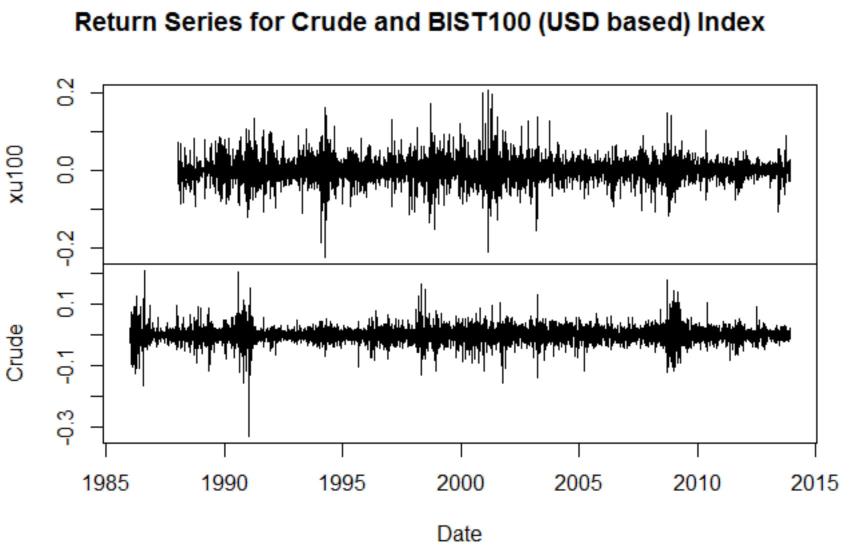


`rollapply()` function is a very useful one to get rolling window analysis. Let's perform regression analysis where dependent variable used `Close` of `xu100.zoo` object and `Crude` price is the independent variable. USD based index and crude price series are assumed to be stationary in this setting. One can perform regression on top of return series as well. For the sake of presentation I add both of them. I use `PerformanceAnalytics` package for calculating returns. I highly recommend this package for those interested in financial modelling.

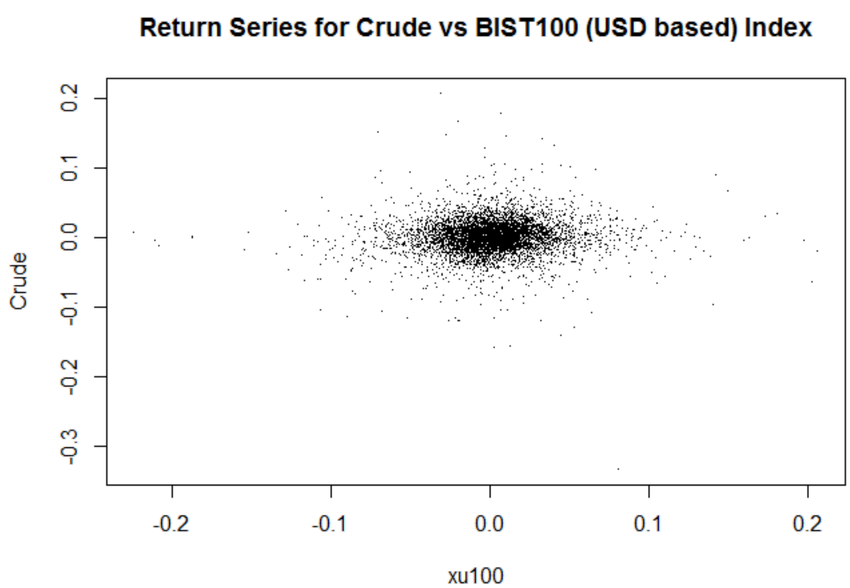
```
# install.packages("PerformanceAnalytics")
library(PerformanceAnalytics)
# Create a new zoo object containint Close values of xu100 and Crude of oil prices.
tmp <- merge(xu100.zoo[,"USD"], oil.zoo[,"Crude"])
tmp.ret <- CalculateReturns(tmp)

colnames(tmp) <- colnames(tmp.ret) <- c("xu100","Crude")
plot(tmp, main="Crude and BIST100 (USD based) Index", xlab="Date")
```



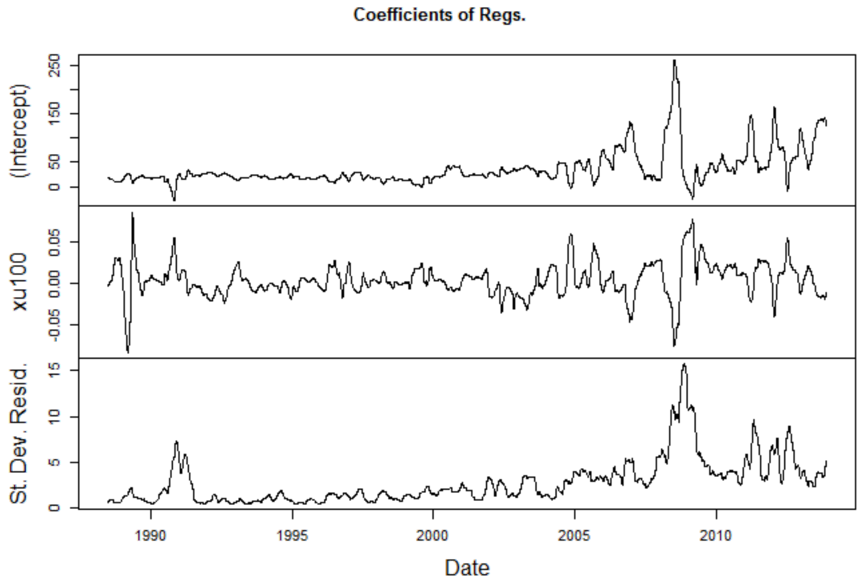

```
plot(tmp.ret, main="Return Series for Crude and BIST100 (USD based) Index", xlab="Date")
```




```
plot(coredata(na.omit(tmp.ret)), pch=".", main="Return Series for Crude vs BIST100 (USD based) Index")
```



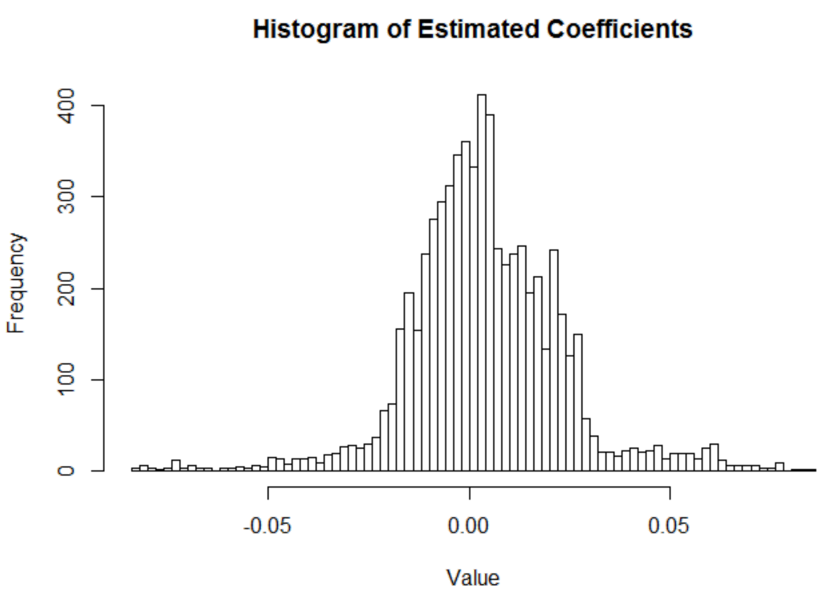

```
rr <- rollapply(na.approx(na.trim(tmp, side="both")), width = 120,
            FUN = function(z) coef(lm(Crude ~ xu100, data = as.data.frame(z))),
            by.column = FALSE, align = "right")

rr.var <- rollapply(na.approx(na.trim(tmp, side="both")), width = 120,
            FUN = function(z) sd(residuals(lm(Crude ~ xu100, data = as.data.frame(z)))),
            by.column = FALSE, align = "right")

res <- merge(rr,rr.var)
colnames(res)[3] <- "St. Dev. Resid."
plot(res, main="Coefficients of Regs.", xlab="Date")
```



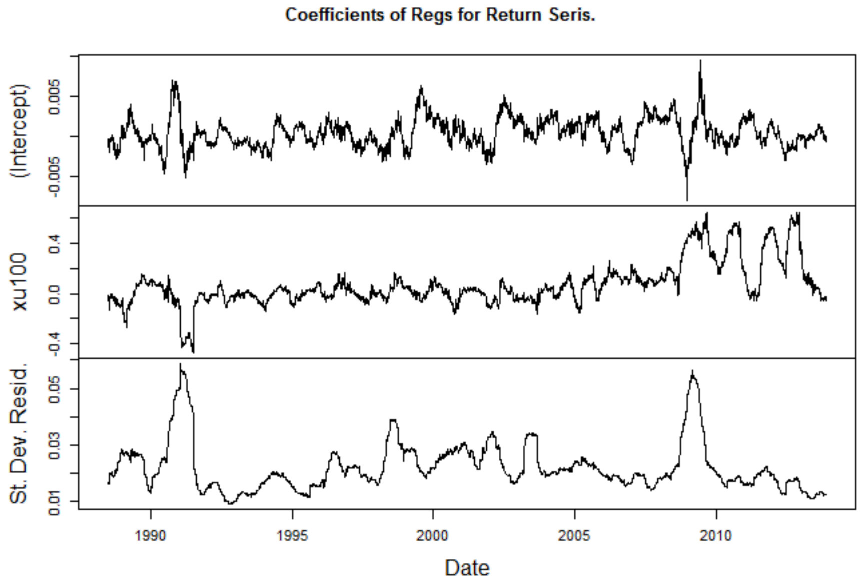

```
summary(res)
```

```
##      Index             (Intercept)        xu100           St. Dev. Resid.
##  Min.   :1988-06-17   Min.   :-29.0   Min.   :-0.08279   Min.   : 0.47
##  1st Qu.:1994-10-27   1st Qu.: 17.3   1st Qu.:-0.00704   1st Qu.: 1.08
##  Median :2001-03-14   Median : 23.7   Median : 0.00267   Median : 2.01
##  Mean   :2001-03-11   Mean   : 36.2   Mean   : 0.00390   Mean   : 2.82
##  3rd Qu.:2007-07-23   3rd Qu.: 41.0   3rd Qu.: 0.01460   3rd Qu.: 3.68
##  Max.   :2013-12-02   Max.   :260.2   Max.   : 0.08576   Max.   :15.74
```

```
hist(res[,"xu100"], breaks=100, main="Histogram of Estimated Coefficients", xlab="Value")
```



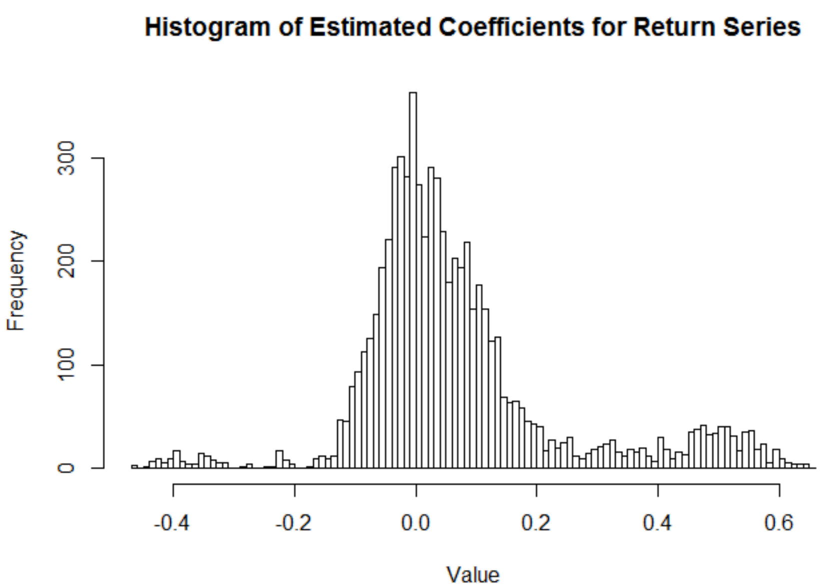

```
# For return series
rr <- rollapply(na.approx(na.trim(tmp.ret, side="both")), width = 120,
            FUN = function(z) coef(lm(Crude ~ xu100, data = as.data.frame(z))),
            by.column = FALSE, align = "right")

rr.var <- rollapply(na.approx(na.trim(tmp.ret, side="both")), width = 120,
            FUN = function(z) sd(residuals(lm(Crude ~ xu100, data = as.data.frame(z)))),
            by.column = FALSE, align = "right")

res.ret <- merge(rr,rr.var)
colnames(res.ret)[3] <- "St. Dev. Resid."
plot(res.ret, main="Coefficients of Regs for Return Seris.", xlab="Date")
```




```
summary(res.ret)
```

```
##      Index              (Intercept)            xu100
##  Min.   :1988-06-20   Min.   :-0.008092   Min.   :-0.4691
##  1st Qu.:1994-10-28   1st Qu.:-0.000833   1st Qu.:-0.0264
##  Median :2001-03-15   Median : 0.000274   Median : 0.0309
##  Mean   :2001-03-11   Mean   : 0.000426   Mean   : 0.0701
##  3rd Qu.:2007-07-24   3rd Qu.: 0.001621   3rd Qu.: 0.1130
##  Max.   :2013-12-02   Max.   : 0.009510   Max.   : 0.6514
##  St. Dev. Resid.
##  Min.   :0.00894
##  1st Qu.:0.01659
##  Median :0.01976
##  Mean   :0.02201
##  3rd Qu.:0.02528
##  Max.   :0.05875
```

```
hist(res.ret[,"xu100"], breaks=100, main="Histogram of Estimated Coefficients for Return Series", xlab="Value")
```



Applying Functions to Each Specific Periods

We have three data objects - all are zoo objects - and it is now easy to perform some preprocessing steps such as missing value tratment, aggregation (changing from high frequency to low frequency), outlier detectin and treatment etc. There are several functions you may find in different packages. Among them, eXtensible Time Series () package has (i) `apply.daily()`, `apply.weekly()`, `apply.monthly()`, `apply.quarterly()`, `apply.yearly()` for applying functions in commonly used periods, (ii) `to.daily()`, `to.weekly()`, `to.monthly()`, `to.quarterly()`, `to.yearly()` for converting high frequency data to low frequency data.

For example,

```
# install.packages("xts")
library(xts)
tail(to.yearly(usd))
```

```
## Warning: missing values removed from data
```

```
##            usd.Open usd.High usd.Low usd.Close
## 2008-12-31    1.165    1.704   1.150     1.520
## 2009-12-31    1.529    1.804   1.443     1.513
## 2010-12-31    1.494    1.605   1.395     1.554
## 2011-12-30    1.545    1.916   1.503     1.916
## 2012-12-31    1.898    1.898   1.742     1.791
## 2013-12-09    1.786    2.077   1.754     2.043
```

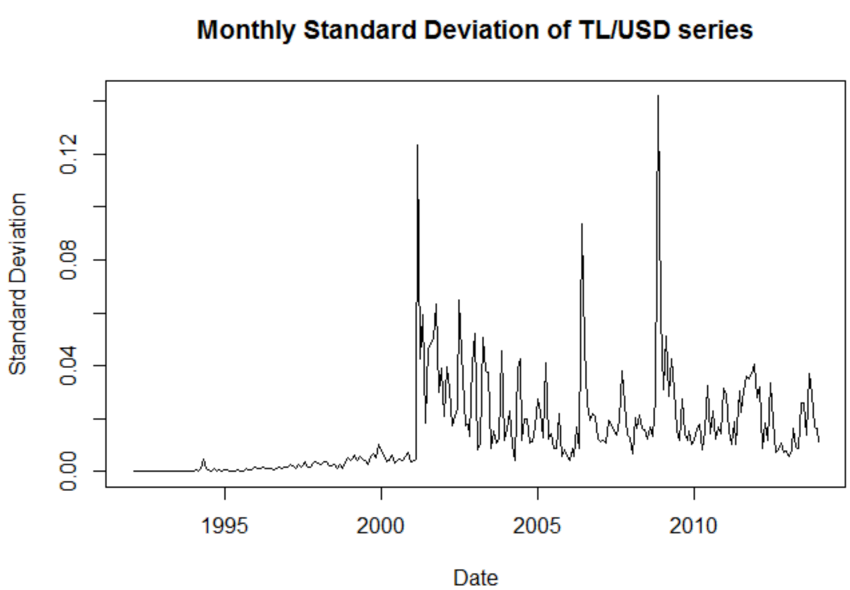
```
start(usd)
```

```
## [1] "1950-01-02"
```

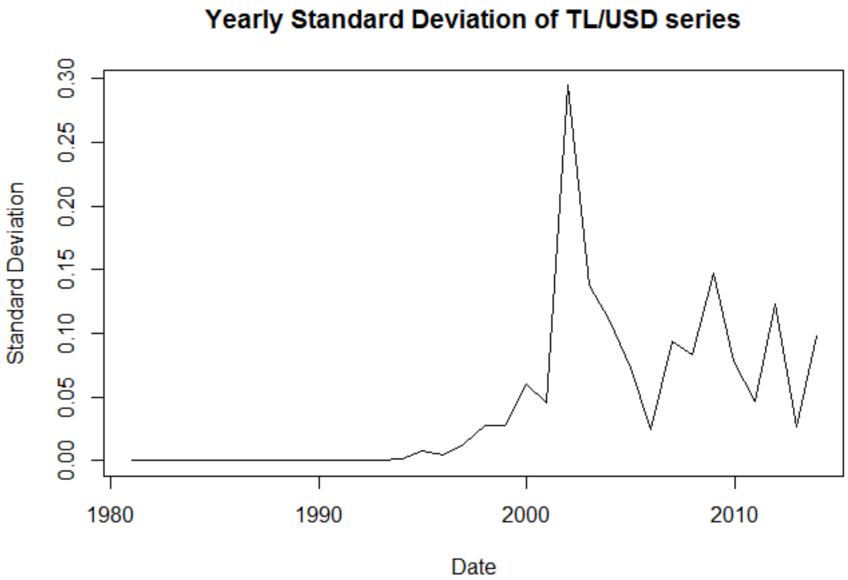
```
end(usd)
```

```
## [1] "2013-12-09"
```

```
plot(apply.monthly(window(usd, start="1992-01-01") ,function(x) sd(x, na.rm=TRUE)), main="Monthly Standard Deviation of TL/USD series", ylab="Standard De
viation", xlab="Date")
```




```
plot(apply.yearly(window(usd, start="1980-01-01") ,function(x) sd(x, na.rm=TRUE)), main="Yearly Standard Deviation of TL/USD series", ylab="Standard Devi
ation", xlab="Date")
```




```
tail(to.monthly(usd))
```

```
## Warning: missing values removed from data
```

```
##          usd.Open usd.High usd.Low usd.Close
## Tem 2013    1.928    1.961   1.910     1.928
## Ağu 2013    1.934    2.059   1.927     2.059
## Eyl 2013    2.035    2.077   1.956     2.038
## Eki 2013    2.040    2.040   1.977     1.992
## Kas 2013    1.993    2.055   1.993     2.020
## Ara 2013    2.021    2.052   2.021     2.043
```

```
tail(to.yearly(usd))
```

```
## Warning: missing values removed from data
```

```
##            usd.Open usd.High usd.Low usd.Close
## 2008-12-31    1.165    1.704   1.150     1.520
## 2009-12-31    1.529    1.804   1.443     1.513
## 2010-12-31    1.494    1.605   1.395     1.554
## 2011-12-30    1.545    1.916   1.503     1.916
## 2012-12-31    1.898    1.898   1.742     1.791
## 2013-12-09    1.786    2.077   1.754     2.043
```

Univariate Time Series Model Examples

Many indtroductory to advanced levels time series analysis books using R software published Over the last 5-6 years. Springer Use R! (http://www.springer.com/series/6991), CRS Press The R Series (http://www.crcpress.com/browse/series/crctherser,), O'Reilly R books (http://search.oreilly.com/?q=R+books&x=0&y=0) are only a few examples of R related books. See also R Documentation (http://www.r-project.org/other-docs.html) page for a list of books and other documents.

I am going to show two very convenient packages related with univariate time series modelling. These are `forecast` and `TSA` packages.

```
# install.packages(c("forecast","TSA"))
library(forecast)
library(TSA)

ls("package:forecast")
```

```
##  [1] "accuracy"             "Acf"                  "arfima"
##  [4] "Arima"                "arima.errors"         "arimaorder"
##  [7] "auto.arima"           "bats"                 "bizdays"
## [10] "BoxCox"               "BoxCox.lambda"        "croston"
## [13] "CV"                   "dm.test"              "dshw"
## [16] "easter"               "ets"                  "fitted.Arima"
## [19] "forecast"             "forecast.ar"          "forecast.Arima"
## [22] "forecast.bats"        "forecast.ets"         "forecast.fracdiff"
## [25] "forecast.HoltWinters" "forecast.lm"          "forecast.nnetar"
## [28] "forecast.stl"         "forecast.StructTS"    "forecast.tbats"
## [31] "fourier"              "fourierf"             "gas"
## [34] "getResponse"          "gold"                 "holt"
## [37] "hw"                   "InvBoxCox"            "logLik.ets"
## [40] "ma"                   "meanf"                "monthdays"
## [43] "msts"                 "na.interp"            "naive"
## [46] "ndiffs"               "nnetar"               "nsdiffs"
## [49] "Pacf"                 "plot.bats"            "plot.ets"
## [52] "plot.forecast"        "plot.splineforecast"  "plot.tbats"
## [55] "rwf"                  "seasadj"              "seasonaldummy"
## [58] "seasonaldummyf"       "seasonplot"           "ses"
## [61] "simulate.ar"          "simulate.Arima"       "simulate.ets"
## [64] "simulate.fracdiff"    "sindexf"              "snaive"
## [67] "splinef"              "stlf"                 "taylor"
## [70] "tbats"                "tbats.components"     "thetaf"
## [73] "tsclean"              "tsdisplay"            "tslm"
## [76] "tsoutliers"           "wineind"              "woolyrnq"
```

```
ls("package:TSA")
```

```
##  [1] "acf"                 "arima"               "arima.boot"
##  [4] "arimax"              "ARMAspec"            "armasubsets"
##  [7] "BoxCox.ar"           "detectAO"            "detectIO"
## [10] "eacf"                "fitted.Arima"        "garch.sim"
## [13] "gBox"                "harmonic"            "Keenan.test"
## [16] "kurtosis"            "lagplot"             "LB.test"
## [19] "McLeod.Li.test"      "periodogram"         "plot.Arima"
## [22] "plot.armasubsets"    "plot1.acf"           "predict.TAR"
## [25] "prewhiten"           "qar.sim"             "rstandard.Arima"
## [28] "runs"                "season"              "skewness"
## [31] "spec"                "summary.armasubsets" "tar"
## [34] "tar.sim"             "tar.skeleton"        "tlrt"
## [37] "Tsay.test"           "tsdiag.Arima"        "tsdiag.TAR"
## [40] "zlag"
```

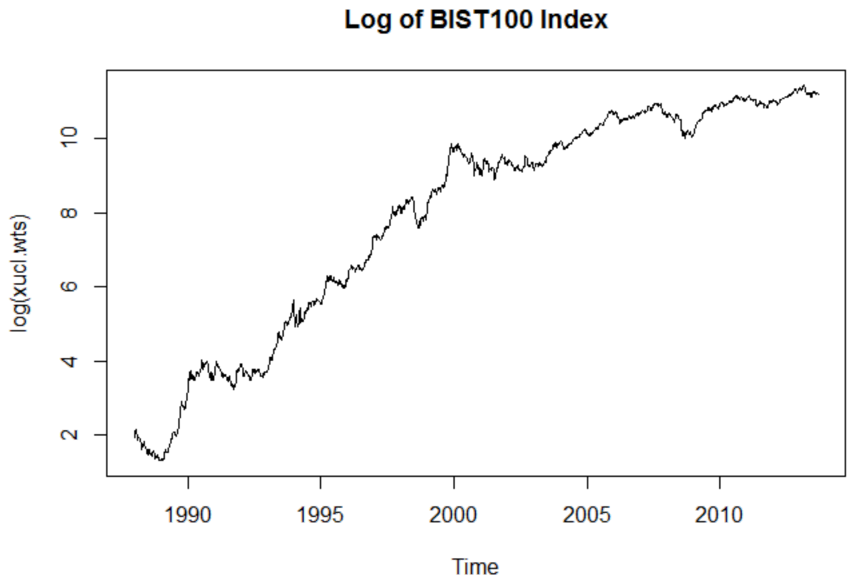
```
# For outliers
# ?detectIO #Innovative Outlier Detection
# ?detectAO #Additive Outlier Detection

# Ok.. Lets play with our data..
# Need ts object for forecast
xucl <- na.approx(na.trim(xu100.zoo[,"Close"], side="both"))
head(to.weekly(xucl))
```

```
##            xucl.Open xucl.High xucl.Low xucl.Close
## 1988-01-08      6.89      7.07     6.89       6.96
## 1988-01-15      7.03      7.83     7.03       7.69
## 1988-01-22      7.89      8.52     7.89       8.32
## 1988-01-29      8.58      8.58     8.35       8.58
## 1988-02-05      8.56      8.56     7.79       7.79
## 1988-02-12      7.59      7.59     7.23       7.47
```

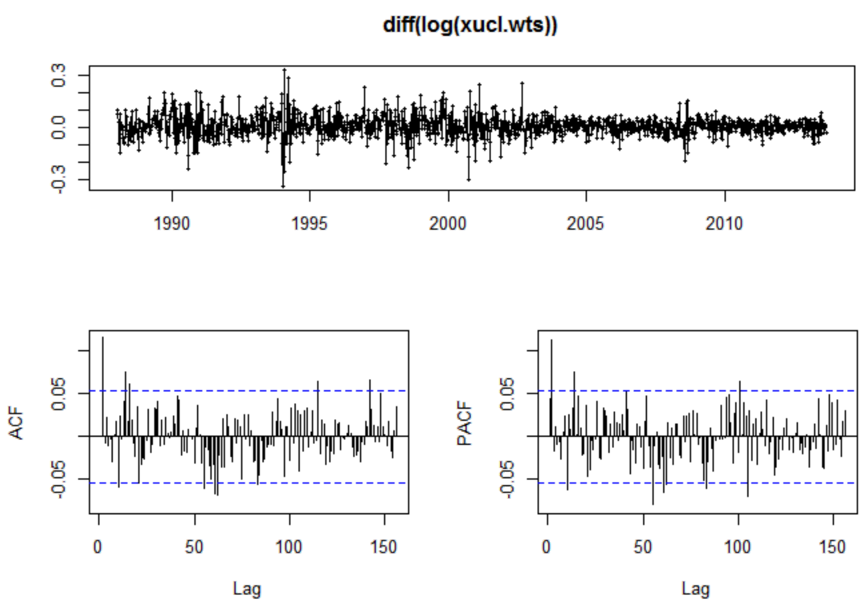
```
xucl.w <- to.weekly(xucl)[,"xucl.Close"]
# A dirty weekly close oil as time series.. beware!!!!
start(xucl.w)
```

```
## [1] "1988-01-08"
```

```
xucl.wts <- ts(xucl.w, start=c(1988,1,8), freq=52)

#
plot(log(xucl.wts), main="Log of BIST100 Index")
```




```
tsdisplay(diff(log(xucl.wts))) ## There appears a stationary in mean..
```




```
# Looks like .... .. (fill in the blank) Also one need to chech the seasonality.. I skip here..

# auto.arima function of forecast package..
auto.arima(log(xucl.wts))
```

```
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
## Warning: Unable to check for unit roots
```

```
## Series: log(xucl.wts)
## ARIMA(0,1,2)(0,0,1)[52] with drift
##
## Coefficients:
##         ma1    ma2   sma1  drift
##       0.042  0.115  0.042  0.007
## s.e.  0.027  0.027  0.027  0.002
##
## sigma^2 estimated as 0.00394:  log likelihood=1801
## AIC=-3592   AICc=-3592   BIC=-3566
```

```
# Series: log(xucl.wts)
# ARIMA(0,1,2)(0,0,1)[52] with drift

# Coefficients:
#   ma1     ma2    sma1   drift
# 0.0415  0.1152  0.0415  0.0069
# s.e.  0.0273  0.0266  0.0270  0.0021

# sigma^2 estimated as 0.003938:  log likelihood=1800.77
# AIC=-3591.55   AICc=-3591.5   BIC=-3565.56

# Assume that candidates are are from 1,1,0 to 2,1,2
# Lets try only 1,1,1 and 2,1,2 and 2,1,0
arima(log(xucl.wts), order = c(1,1,1))
```

```
## Series: x
## ARIMA(1,1,1)
##
## Coefficients:
##         ar1     ma1
##       0.705  -0.630
## s.e.  0.140   0.152
##
## sigma^2 estimated as 0.004:  log likelihood=1790
## AIC=-3577   AICc=-3577   BIC=-3561
```

```
arima(log(xucl.wts), order = c(2,1,2))
```

```
## Series: x
## ARIMA(2,1,2)
##
## Coefficients:
##          ar1    ar2    ma1     ma2
##       -0.077  0.265  0.128  -0.137
## s.e.   0.208  0.195  0.214   0.198
##
## sigma^2 estimated as 0.00397:  log likelihood=1796
## AIC=-3583   AICc=-3583   BIC=-3557
```

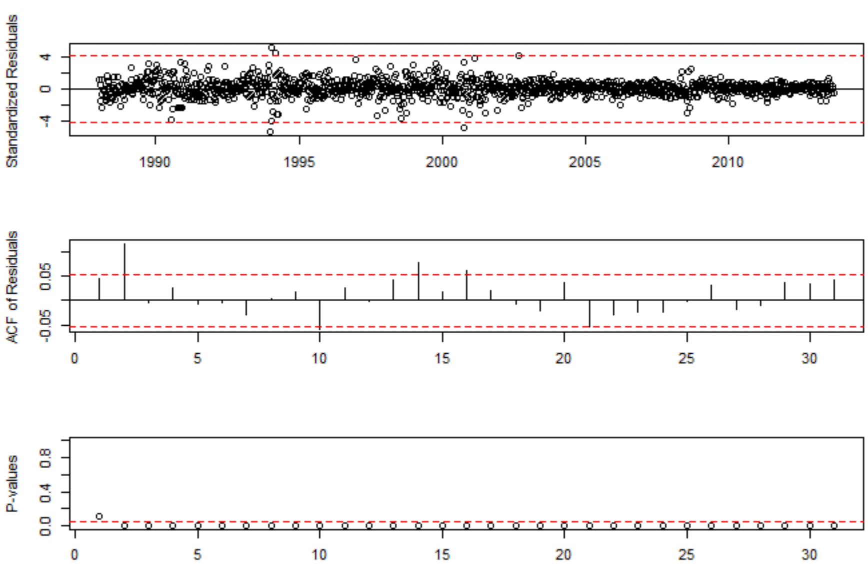
```
arima(log(xucl.wts), order = c(2,1,0))
```

```
## Series: x
## ARIMA(2,1,0)
##
## Coefficients:
##         ar1    ar2
##       0.049  0.123
## s.e.  0.027  0.027
##
## sigma^2 estimated as 0.00398:  log likelihood=1795
## AIC=-3587   AICc=-3587   BIC=-3571
```

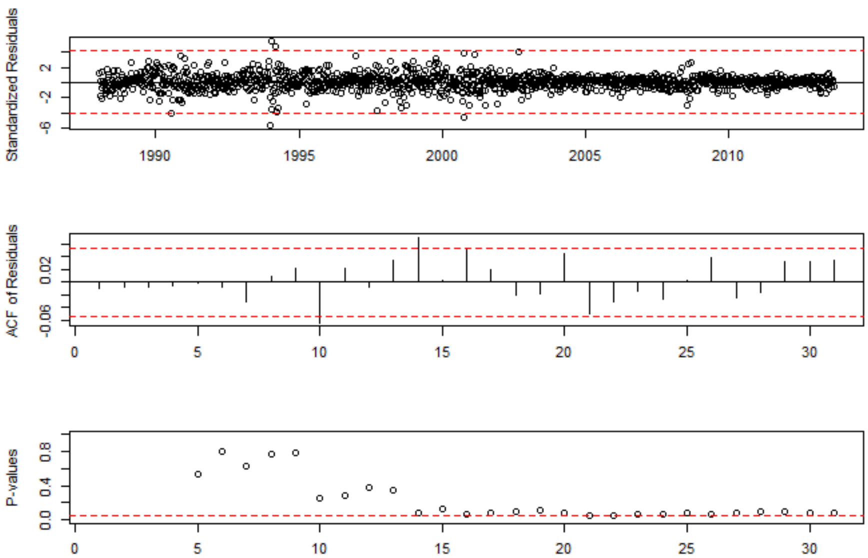
```
# Oh.. does return series (diff(log)) follow random walk??
arima(log(xucl.wts), order = c(0,1,0))
```

```
## Series: x
## ARIMA(0,1,0)
##
## sigma^2 estimated as 0.00405:  log likelihood=1783
## AIC=-3566   AICc=-3566   BIC=-3561
```

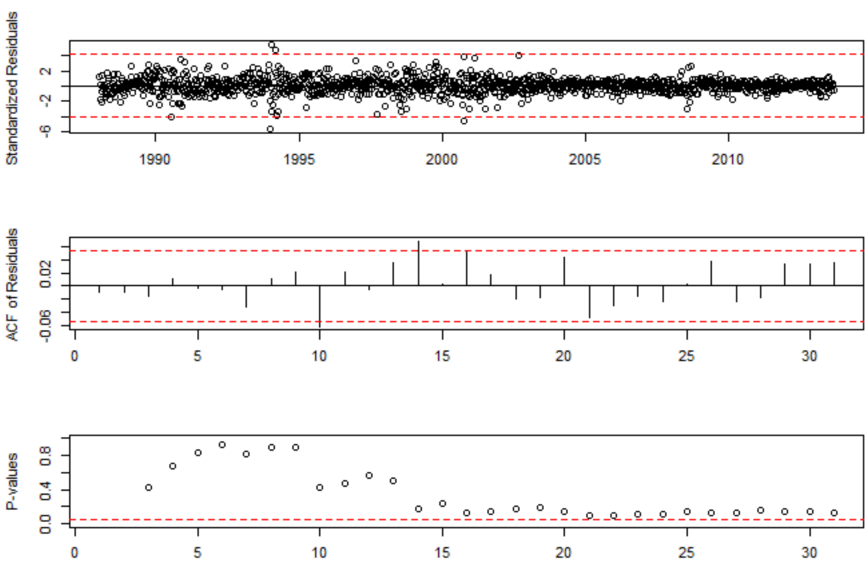
```
# One may select ARIMA(2,1,0) (beware: Seasonality Ignored..)
# based on AIC. Check Diagnostic..

#Random Walk..
tsdiag(arima(log(xucl.wts), order = c(0,1,0)))
```




```
# Two competing models.
tsdiag(arima(log(xucl.wts), order = c(2,1,2)))
```




```
tsdiag(arima(log(xucl.wts), order = c(2,1,0)))
```



Let's now go through the basic steps of ARCH-GARCH modelling. For the details of modelling please do refer to our suggested readings.

ARCH Models (Autoregressive Conditional Heteroskedasticity)

Let's $\epsilon_t=\sigma_t a_t$ is return residual obtained after modelling a mean process or in other terms, $r_t=~\mu_t+ \sigma_t^2 a_t$. The random variable a_t is white noise. The variance of residual series σ_t^2 is modelled as;

$\sigma_t^2=\alpha_0+\alpha_1 \epsilon_{t-1}^2+\cdots+\alpha_q \epsilon_{t-q}^2 = \alpha_0 + \sum_{i=1}^q \alpha_{i} \epsilon_{t-i}^2$
where $\alpha_0>0$, $\alpha_i \geq 0$ and $i > 0$.

Steps for building

1. Specify a mean equation by testing for serial dependence in the data and, if necessary, building an econometric model (e.g., an ARMA model) for the return series to remove any linear dependence.
2. Use the residuals of the mean equation to test for ARCH effects. Ljung-Box Test for ϵ_t^2, or Lagrange Multiplier Test (Ex: ArchTest in FinTS package) for example see page 102 of Tsay's book (http://www.amazon.com/Analysis-Financial-Time-Series-Ruey/dp/0470414359) (Analysis of financial time series, John Wiley & Sons).

```
# Tsay page 102
require(FinTS)
data(m.intc7303)
intcLM <- ArchTest(log(1+as.numeric(m.intc7303)), lag=12)
```

3. Specify a volatility model if ARCH effects are statistically significant and perform a joint estimation of the mean and volatility equations.
4. Check the fitted model carefully and refine it if necessary.

Weaknesses of ARCH models

- The model assumes that positive and negative shocks have the same effects on volatility because it depends on the square of the previous shocks. In practice, it is well known that price of a financial asset responds differently to positive and negative shocks.
- The ARCH model is rather restrictive. The constraint becomes complicated for higher order ARCH models. In practice, it limits the ability of ARCH models with Gaussian innovations to capture excess kurtosis.
- The ARCH model does not provide any new insight for understanding the source of variations of a financial time series. It merely provides a mechanical way to describe the behavior of the conditional variance. It gives no indication about what causes such behavior to occur.
- ARCH models are likely to overpredict the volatility because they respond slowly to large isolated shocks to the return series.

Order Determination

For order determination Partial Autocorrelation of squared series is used.

GARCH models..

GARCH Bollerslev (1986) (http://www.sciencedirect.com/science/article/pii/0304407686900631) (Bollerslev, Tim. "Generalized autoregressive conditional heteroskedasticity." Journal of econometrics 31.3 (1986): 307-327.) model is the natural generalization of ARCH models and is given by;

\(\sigma_t^2=\alpha_0 + \alpha_1 \epsilon_{t-1}^2 + \cdots + \alpha_q \epsilon_{t-q}^2 + \beta_1 \sigma_{t-1}^2 + \cdots + \beta_p\sigma_{t-p}^2 = \alpha_0 + \sum_{i=1}^q \alpha_i \epsilon_{t-i}^2 + \sum_{i=1}^p \beta_i \sigma_{t-i}^2\)

Steps for building

- Start with lower orders.
- Estimate model
- Diagnose the model
- Increase the order and go through previous steps.
- Do follow these steps until a desired model obtained by comparing each models.

Most of the time **GARCH(1,1)** does good job. In practice, up to GARCH(2,2) model is used.

```
# An estimation example..
require(rmgarch)
```

```
## Loading required package: rmgarch
## Loading required package: rugarch
## KernSmooth 2.23 loaded
## Copyright M. P. Wand 1997-2009
##
## Attaching package: 'rmgarch'
##
## The following objects are masked from 'package:xts':
##
##     first, last
```

```
require(PerformanceAnalytics)

# Lets use usd series.. starting from 2007
usd1 <- window(usd, start="2007-01-01")

# Remove NA's
usd1 <- na.approx(na.trim(CalculateReturns(usd1), side="both"))

# start with default GARCH spec.
spec = ugarchspec()
print(spec)
```

```
##
## *---------------------------------*
## *       GARCH Model Spec          *
## *---------------------------------*
##
## Conditional Variance Dynamics
## ------------------------------------
## GARCH Model      : sGARCH(1,1)
## Variance Targeting   : FALSE
##
## Conditional Mean Dynamics
## ------------------------------------
## Mean Model       : ARFIMA(1,0,1)
## Include Mean     : TRUE
## GARCH-in-Mean        : FALSE
##
## Conditional Distribution
## ------------------------------------
## Distribution :  norm
## Includes Skew    :  FALSE
## Includes Shape   :  FALSE
## Includes Lambda  :  FALSE
```

```
def.fit = ugarchfit(spec = spec, data = usd1)
print(def.fit)
```

```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(1,0,1)
## Distribution : norm
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.000075    0.000151  0.49671  0.61940
## ar1     0.244584    0.237513  1.02977  0.30312
## ma1    -0.124114    0.243650 -0.50939  0.61048
## omega   0.000001    0.000001  0.69588  0.48650
## alpha1  0.145368    0.026687  5.44706  0.00000
## beta1   0.853186    0.023309 36.60366  0.00000
##
## Robust Standard Errors:
##         Estimate  Std. Error   t value Pr(>|t|)
## mu      0.000075     0.00015  0.501948 0.615704
## ar1     0.244584     0.30358  0.805654 0.420442
## ma1    -0.124114     0.30555 -0.406197 0.684598
## omega   0.000001     0.00001  0.072872 0.941908
## alpha1  0.145368     0.19748  0.736121 0.461657
## beta1   0.853186     0.18272  4.669431 0.000003
##
## LogLikelihood : 6433
##
## Information Criteria
## ------------------------------------
##
## Akaike       -7.1133
## Bayes        -7.0950
## Shibata      -7.1133
## Hannan-Quinn -7.1065
##
## Q-Statistics on Standardized Residuals
## ------------------------------------
##               statistic p-value
## Lag[1]           0.9397  0.3323
## Lag[p+q+1][3]    2.4793  0.1154
## Lag[p+q+5][7]    7.7397  0.1712
## d.o.f=2
## H0 : No serial correlation
##
## Q-Statistics on Standardized Squared Residuals
## ------------------------------------
##               statistic p-value
## Lag[1]           0.2334  0.6290
## Lag[p+q+1][3]    1.5376  0.2150
## Lag[p+q+5][7]    2.4872  0.7784
## d.o.f=2
##
## ARCH LM Tests
## ------------------------------------
##              Statistic DoF P-Value
## ARCH Lag[2]     0.2487   2  0.8831
## ARCH Lag[5]     2.3257   5  0.8025
## ARCH Lag[10]    5.1267  10  0.8826
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  85.38
## Individual Statistics:
## mu      0.05651
## ar1     0.02600
## ma1     0.02469
## omega  15.50043
## alpha1  0.57176
## beta1   0.40496
##
```

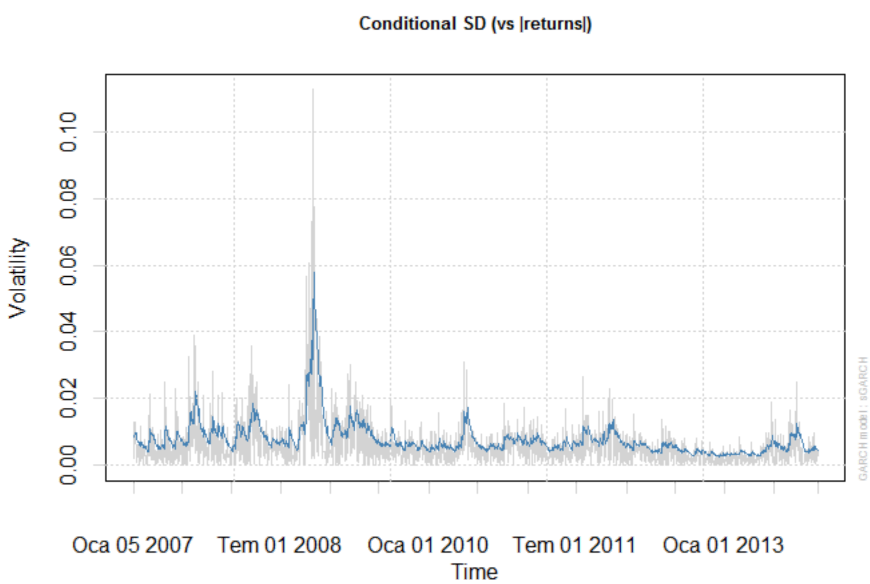
```
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.49 1.68 2.12
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value    prob sig
## Sign Bias           1.3368 0.18145
## Negative Sign Bias  0.2289 0.81899
## Positive Sign Bias  0.9973 0.31873
## Joint Effect        7.5687 0.05582   *
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     30.40     0.046935
## 2    30     50.13     0.008749
## 3    40     59.76     0.017792
## 4    50     65.30     0.059529
##
##
## Elapsed time : 0.242
```

```
# ARMA effect, AIC etc, Q-Stat (Ljung and Box 1978) for resid, and some other tests for misspecification is given in output.
# ARCH LM test; null: no ARCH effect
# Q-Stat: high p-values idicates little chance for serial correlation
# sign bias test; null: no significiant negative and positive reaction shocks (if exist apARCH type models)

# Goodness of Fit; with 20 to 50 bins of Chi-squared test.

# Nymblom test: the parameter stability test (should we switch to TGARCH models?) Here Omega and alpha are worth discussing.. But Omega is not stat sign.
 coefficient..


# Lets chnage GARCH Specs.. Lets use GARCH(1,1)..

garch11.spec = ugarchspec(mean.model = list(armaOrder = c(0,0)),
               variance.model = list(garchOrder = c(1,1),
               model = "sGARCH"), distribution.model = "norm")

# Fit the model
garch.fit = ugarchfit(garch11.spec, data = usd1, fit.control=list(scale=TRUE))

print(garch.fit)
```

```
##
## *---------------------------------*
## *          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      0.000073    0.000135  0.54228  0.58762
## omega   0.000001    0.000001  0.70675  0.47972
## alpha1  0.144868    0.027917  5.18931  0.00000
## beta1   0.853067    0.024528 34.77891  0.00000
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.000073    0.000151 0.481739 0.629991
## omega   0.000001    0.000010 0.074048 0.940973
## alpha1  0.144868    0.210283 0.688919 0.490874
## beta1   0.853067    0.194949 4.375837 0.000012
##
## LogLikelihood : 6421
##
## Information Criteria
## ------------------------------------
##
## Akaike       -7.1020
## Bayes        -7.0898
## Shibata      -7.1020
## Hannan-Quinn -7.0975
##
## Q-Statistics on Standardized Residuals
## ------------------------------------
##               statistic   p-value
## Lag[1]            30.77 2.908e-08
## Lag[p+q+1][1]     30.77 2.908e-08
## Lag[p+q+5][5]     33.79 2.620e-06
## d.o.f=0
## H0 : No serial correlation
##
## Q-Statistics on Standardized Squared Residuals
## ------------------------------------
##               statistic p-value
## Lag[1]           0.5958  0.4402
## Lag[p+q+1][3]    1.7031  0.1919
## Lag[p+q+5][7]    2.8182  0.7280
## d.o.f=2
##
## ARCH LM Tests
## ------------------------------------
##              Statistic DoF P-Value
## ARCH Lag[2]     0.6126   2  0.7362
## ARCH Lag[5]     2.3663   5  0.7965
## ARCH Lag[10]    6.2159  10  0.7968
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  82.4
## Individual Statistics:
## mu      0.06676
## omega  14.49200
## alpha1  0.59884
## beta1   0.46025
##
## 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           2.8035 0.00511 ***
## Negative Sign Bias  0.4418 0.65866
## Positive Sign Bias  0.4680 0.63984
## Joint Effect       14.7022 0.00209 ***
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     38.39     0.005292
## 2    30     46.77     0.019624
## 3    40     59.72     0.017959
## 4    50     75.15     0.009535
##
##
## Elapsed time : 0.14
```

```
plot(garch.fit, which=3)
```




```
# try plot(garch.fit)
# plot(garch.fit, which=10)

# plot(garch.fit, which=11)

# plot(garch.fit, which=8)

# plot(garch.fit, which="all")

# One may want to clean data from outliers.. Beware the
```

For some other ARCH/GARCH related models, see Glossary to ARCH (GARCH), Tim Bollerslev (ftp://ftp.econ.au.dk/creates/rp/08/rp08_49.pdf).

Dynamic Conditional Correlation

Dynamic Conditional Correlation (DCCR) (http://www.tandfonline.com/doi/pdf/10.1198/073500102288618487)(Engle, R. (2002). Dynamic conditional correlation: A simple class of multivariate generalized autoregressive conditional heteroskedasticity models. Journal of Business & Economic Statistics, 20(3), 339-350.) is a class of multivariate models that have the flexibility of univariate GARCH models coupled with parsimonious parametric models for the correlations. However as nothing comes without its own limits. There are discussions about these type of models also. As an example see Ten Things You Should Know About the Dynamic Conditional Correlation Representation (http://eprints.ucm.es/21803/).

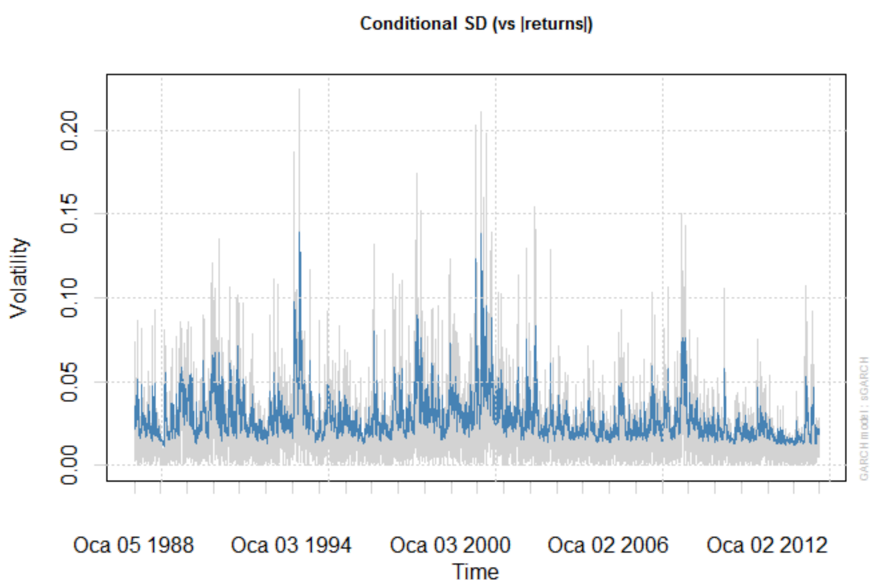
R has `rmgarch` package for performing DCCR estimations. As an example we are going to use our BIST and TL/USD data.

```r
# For DCCR
require(rmgarch)
require(PerformanceAnalytics)

xusd <- merge(xu100.zoo,usd)
# Lets use USD based xu100 and usd series..
xusd <- xusd[,c("USD","usd")]
colnames(xusd)<-c("BIST","TL-USD")

# First GARCH Specs.. Lets use GARCH(1,1) for both of them just to show..
garch11.spec = ugarchspec(mean.model = list(armaOrder = c(0,0)),
               variance.model = list(garchOrder = c(1,1),
               model = "sGARCH"), distribution.model = "norm")

# dcc specification - GARCH(1,1) for conditional correlations
dcc.garch11.spec = dccspec(uspec = multispec( replicate(2, garch11.spec) ), dccOrder = c(1,1), distribution = "mvnorm")

# Obtain some clean return series..
tst <- na.approx(na.trim(CalculateReturns(xusd), side="both"))
# Fit DCC
garch.fit = ugarchfit(garch11.spec, data = tst[,1], fit.control=list(scale=TRUE))

print(garch.fit)
```

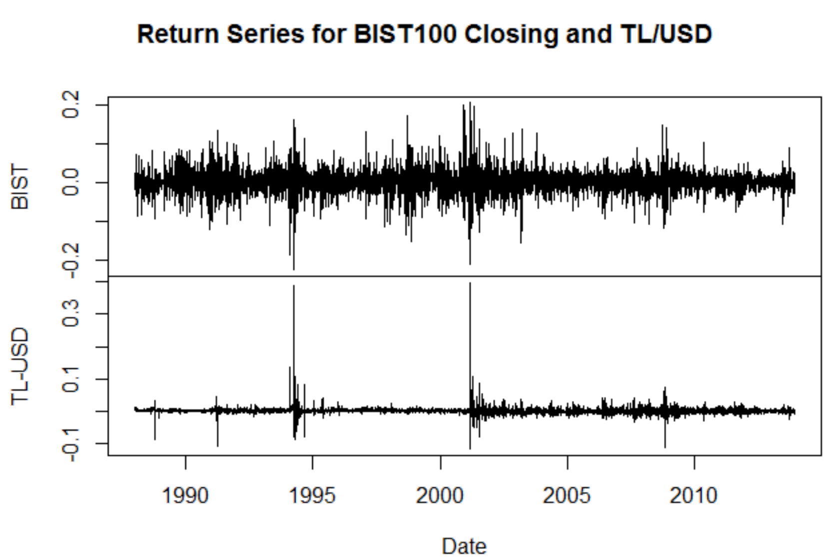
```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(0,0,0)
## Distribution : norm
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.001305    0.000273   4.7841    2e-06
## omega   0.000027    0.000004   6.4621    0e+00
## alpha1  0.191637    0.014768  12.9762    0e+00
## beta1   0.790801    0.015637  50.5720    0e+00
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## mu      0.001305    0.000353   3.6948 0.000220
## omega   0.000027    0.000009   2.9324 0.003363
## alpha1  0.191637    0.029301   6.5404 0.000000
## beta1   0.790801    0.035164  22.4891 0.000000
##
## LogLikelihood : 15217
##
## Information Criteria
## ------------------------------------
##
## Akaike       -4.4982
## Bayes        -4.4942
## Shibata      -4.4982
## Hannan-Quinn -4.4968
##
## Q-Statistics on Standardized Residuals
## ------------------------------------
##               statistic p-value
## Lag[1]            205.1       0
## Lag[p+q+1][1]     205.1       0
## Lag[p+q+5][5]     245.2       0
## d.o.f=0
## H0 : No serial correlation
##
## Q-Statistics on Standardized Squared Residuals
## ------------------------------------
##               statistic   p-value
## Lag[1]            8.268 0.0040344
## Lag[p+q+1][3]     8.817 0.0029849
## Lag[p+q+5][7]    25.049 0.0001363
## d.o.f=2
##
## ARCH LM Tests
## ------------------------------------
##              Statistic DoF   P-Value
## ARCH Lag[2]      8.293   2 0.0158223
## ARCH Lag[5]     14.321   5 0.0136959
## ARCH Lag[10]    34.029  10 0.0001826
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  3.892
## Individual Statistics:
## mu     0.4481
## omega  2.1216
## alpha1 0.8395
## beta1  0.9180
##
## 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           1.7689 0.076951   *
## Negative Sign Bias  1.5447 0.122461
## Positive Sign Bias  0.6672 0.504677
## Joint Effect       12.4362 0.006029 ***
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     85.88    1.769e-10
## 2    30    109.15    3.196e-11
## 3    40    119.02    5.012e-10
## 4    50    133.19    1.029e-09
##
##
## Elapsed time : 0.402
```

```
plot(garch.fit, which=3)
```




```
plot(tst, main="Return Series for BIST100 Closing and TL/USD", xlab="Date")
```



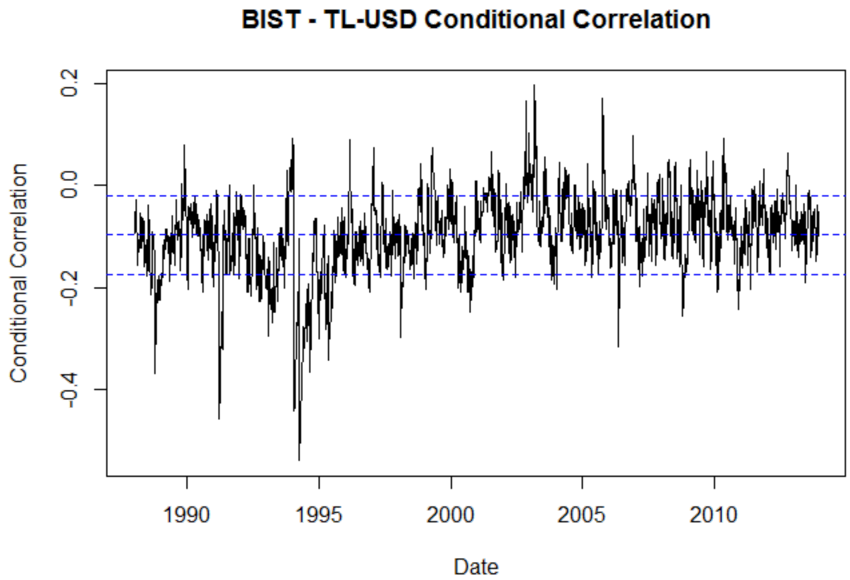

```
dcc.fit = dccfit(dcc.garch11.spec, data = tst, fit.control=list(scale=TRUE))
```

```
## Warning:
## ugarchfit-->warning: solver failer to converge.
```

```
# Get the result of fitting..
print(dcc.fit)
```

```
##
## *---------------------------------*
## *          DCC GARCH Fit          *
## *---------------------------------*
##
## Distribution         :  mvnorm
## Model                :  DCC(1,1)
## No. Parameters       :  11
## [VAR GARCH DCC UncQ] : [0+8+2+1]
## No. Series           :  2
## No. Obs.             :  6764
## Log-Likelihood       :  35186
## Av.Log-Likelihood    :  5.2
##
## Optimal Parameters
## -----------------------------------
##                   Estimate  Std. Error   t value Pr(>|t|)
## [BIST].mu         0.001305    0.000293  4.451966 0.000009
## [BIST].omega      0.000027    0.000008  3.300415 0.000965
## [BIST].alpha1     0.191637    0.029008  6.606422 0.000000
## [BIST].beta1      0.790801    0.033926 23.309493 0.000000
## [TL-USD].mu       0.001292    0.024593  0.052550 0.958091
## [TL-USD].omega    0.000001    0.004274  0.000343 0.999726
## [TL-USD].alpha1   0.166067    1.889442  0.087892 0.929962
## [TL-USD].beta1    0.774243    8.531518  0.090751 0.927691
## [Joint]dcca1      0.026934    0.007171  3.756245 0.000172
## [Joint]dccb1      0.944227    0.039837 23.702056 0.000000
##
## Information Criteria
## ---------------------
##
## Akaike       -10.401
## Bayes        -10.389
## Shibata      -10.401
## Hannan-Quinn -10.397
##
##
## Elapsed time : 9.824
```

```
# Obtain conditional Correlation..
r1=rcor(dcc.fit, type="R")
r1.z=zoo(r1[1,2,], order.by=time(tst))

# Lets plot it..
plot(r1.z, main=paste(colnames(xusd)[1],"-", colnames(xusd)[2], "Conditional Correlation", sep=" "),
     ylab="Conditional Correlation",
     xlab="Date")
abline(h=mean(r1.z), lty=2, lwd=1, col="blue")
abline(h=(mean(r1.z)+sd(r1.z)), lty=2, lwd=1, col="blue")
abline(h=(mean(r1.z)-sd(r1.z)), lty=2, lwd=1, col="blue")
```




```
# Lets plot starting January 2013..
mr1z <- mean(window(r1.z, start="2013-01-01"))
sr1z <- sd(window(r1.z, start="2013-01-01"))

plot(window(r1.z, start="2013-01-01"), main=paste(colnames(xusd)[1],"-", colnames(xusd)[2], "Conditional Correlation", sep=" "),
     ylab="Conditional Correlation", sub=paste("mean:", round(mr1z,3),"sd:", round(sr1z,3)),
     xlab="Date")

abline(h=mr1z, lty=2, lwd=1, col="blue")

abline(h=(mr1z+sr1z), lty=2, lwd=1, col="red")
abline(h=(mr1z-sr1z), lty=2, lwd=1, col="red")
```