Final Project

Mikhail Stukalo, Rishabh Ghosh, Wail Choudar

5/10/2020

```r
library(quantmod) # Financial data
library(PerformanceAnalytics) # Financial analysis
library(xts) # Time indexed data frames
library(tidyr) # Tidy data
library(ggplot2) # Plotting
library(fitdistrplus) # Fitting distributions
library(tibble)  # More advanced type of a data frame
require(dplyr) # Data manipulation
library(fGarch) # skewed normal distribution
library(EnvStats) # Boxcox transformation
```

Note

For the full code please refer to Code.R. It is important to run Code.R first, as it installs all necessary but missing libraries.

General Overview

This paper is designed to illustrate the use of the mathematical and statistical concepts learned in Math 23C. We apply some of the learned concepts to financial data, in particular the volatility index, VIX, and the returns of VIXY, an exchange traded fund (ETF), linked to the performance of VIX.

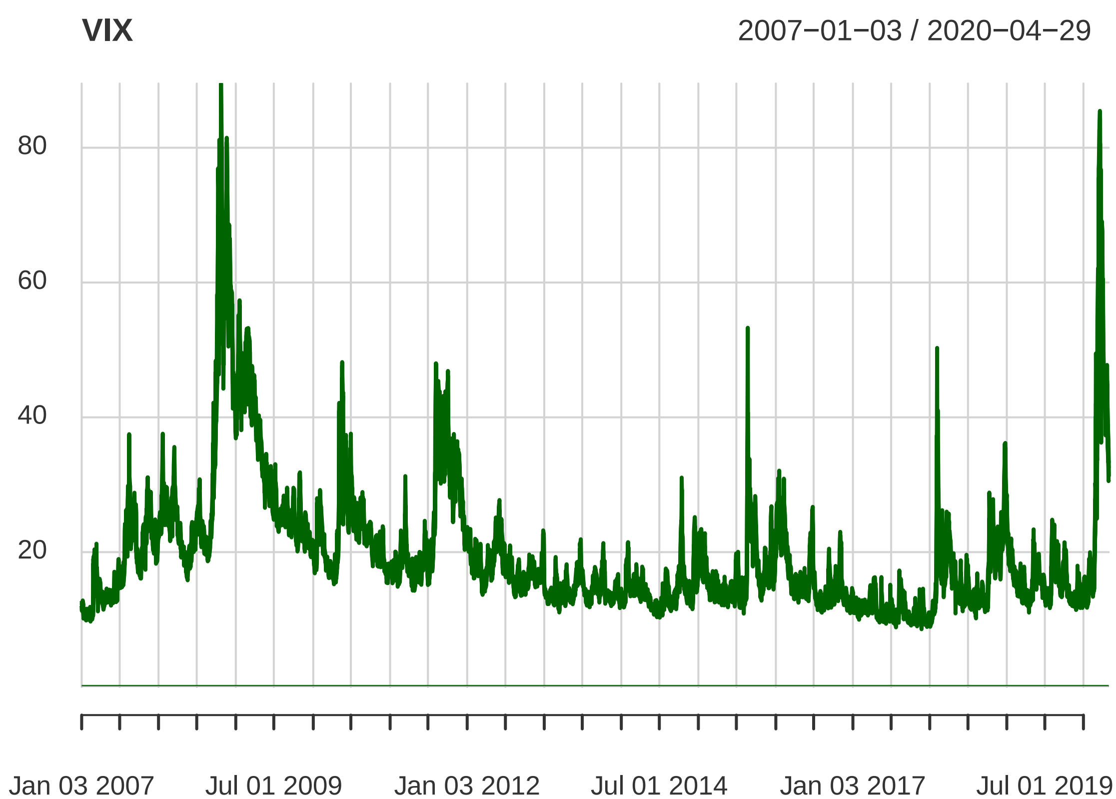
It is important to stress, that by no means this paper is supposed to reflect our views on financial markets neither does it suggest any tools for financial analysis. WE did our best to make sure that the concepts illustrated in the paper are somewhat reasonable from the practical point of view. However, the main goal was the application of mathematical concepts.

VIX Index

VIX is an index maintained by the Chicago Board Options Exchange (CBOE). Volatility Index is ameasure of the stock market's expectation of volatility of S&P500 Index.

```r
# We download data for VIX index using quantmod package
# To ensure the consistency of the analysis, we saved the data into .RDS files
# We still provide the code for downloading the data sets, but we comment it out
# Download data
#vix = getSymbols("^VIX", auto.assign = F) # Download VIX data
# Save for future use
# saveRDS(vix, "./data/vix.RDS")
# write.csv(fortify(vix), "./data/vix.csv", row.names = F)
vix = read.csv("./data/vix.csv")
```

```r
# Convert to xts
# Helper function
to_xts = function(df){
  xts(df[,2:ncol(df)], order.by = as.Date(df[,1]))
}

vix = to_xts(vix)


# Calculate returns
vix_r = CalculateReturns(vix)$VIX.Adjusted


# Plot VIX using Performance Analytics
charts.TimeSeries(vix, main = "VIX", colorset = 'darkgreen')
```



An investor can gain exposure to VIX by purchasing shares exchange traded funds (ETFs), linked to VIX. In this paper, we look at performance of one of the largest VIX-linked ETF, ProShares VIX Short-Term Futures ETF (VIXY), and analyze performance of this ETF relative to the indexx in various market conditions.

Performance of VIXY relative to VIX.

It has been noted, that because VIX is particularly hard to track, many ETFs that technically are linked to VIX, in practice show suboptimal tracking performance. E.g.

https://www.barrons.com/articles/no-your-etf-doesnt-track-the-vix-volatility-index-and-here-are-the-numbers-1403010972

Using *quantmod* package and YahooFinance as the source, we download stock prices of VXX and calculate daily returns.

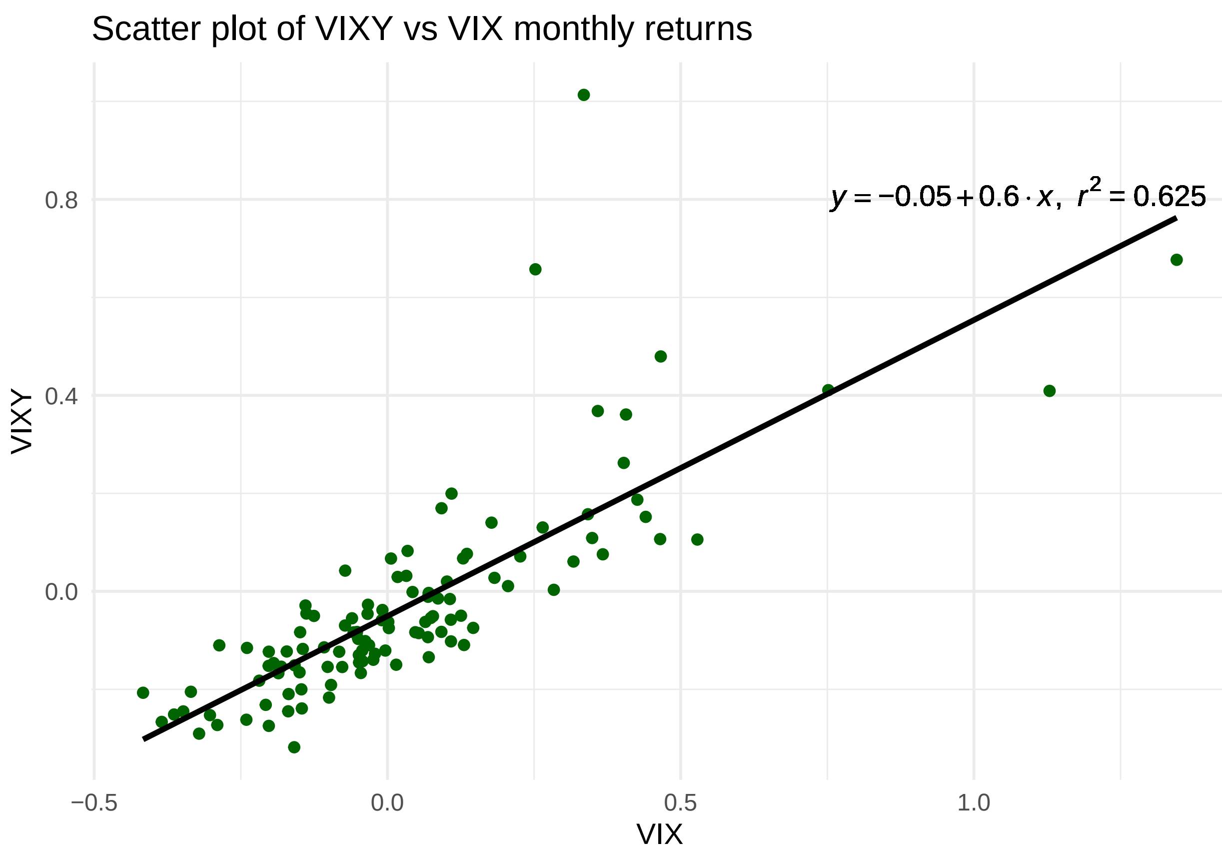
```
# Download VIXY
#vixy = getSymbols("VIXY", auto.assign = F)
# saveRDS(vixy, "./data/vixy.RDS") #Save for future use
#write.csv(fortify(vixy), "./data/vixy.csv", row.names = F)
vixy = read.csv("./data/vixy.csv")
vixy = to_xts(vixy)

# Calculate returns
vixy_r = CalculateReturns(vixy)$VIXY.Adjusted


# Plot
charts.TimeSeries(vixy$VIXY.Adjusted, main = "VIXY stock price", colorset = 'darkgreen')
```

Just by inspection, it is clear that VIXY returns in the long-run do not follow the index. This is known as "beta decay". However, we can explore if the beta of monthly returns of VIXY is somewhat close to VIX index percentage change.

Please see Question 1 in Code.R for more detailed analysis.

```
# Calculate monthly returns
vix_m = monthlyReturn(vix)
vixy_m = monthlyReturn(vixy)
```

```r
# Combine in one df
colnames(vix_m) = "VIX"
colnames(vixy_m) = "VIXY"

df = merge.xts(vixy_m, vix_m,  join ="left" )

# Drop first observation (incomplete month)
df = df[2:nrow(df),]
df = fortify(df) # Convert to a dataframe


# Helper function to plot.
# Partially reused the code from https://groups.google.com/forum/#!topic/ggplot2/1TgH-kG5XMA

lm_eqn <- function(x, y, xlab="", ylab="", main=""){
    tt = data.frame(x = x, y = y)
    m <- lm(y ~ x, tt);
    eq <- substitute(italic(y) == a + b %.% italic(x)*","~~italic(r)^2~"="~r2,
         list(a = format(unname(coef(m)[1]), digits = 2),
              b = format(unname(coef(m)[2]), digits = 2),
             r2 = format(summary(m)$r.squared, digits = 3)))
    eq = as.character(as.expression(eq))

    ggplot(tt, aes(x,y)) + geom_point(color=c('darkgreen')) + theme_minimal() + ggtitle(main) +
      xlab(xlab) + ylab(ylab) + geom_smooth(method = "lm", se=FALSE, color="black", formula = y ~ x) +
      geom_text(x = max(x)*0.8, y = max(y)*0.8, label = eq, parse = TRUE)
}



# Scatter plot
lm_eqn(df$VIX, df$VIXY, xlab='VIX', ylab='VIXY', main = 'Scatter plot of VIXY vs VIX monthly returns')
```



Although we can see that the OLS model describes the relathionship between returns quite well (R^2=0.63), it is far from ideal. To get a perfect exposure to the index, and ETF should exhibit the slope (aka beta) close to 1. Also, as we are fitting a single variable regression, we can calculate the correlation between monthly returns as $\sqrt{R^2} = 0.79$. Again, it is quite far from desired 1.

We can also notice that the fit gets worse with extremely high positive changes in VIX Index, which is associated with increased market turbulence.

An easy way to check this assumtion is to calculate correlation between the absolute value of residuals and VIX returns.

```
# Fit regression
ols = lm(VIXY~VIX, df)

# Extract residulas
res = ols$residuals

cor(abs(res), df$VIX)
```

```
## [1] 0.2860571
```

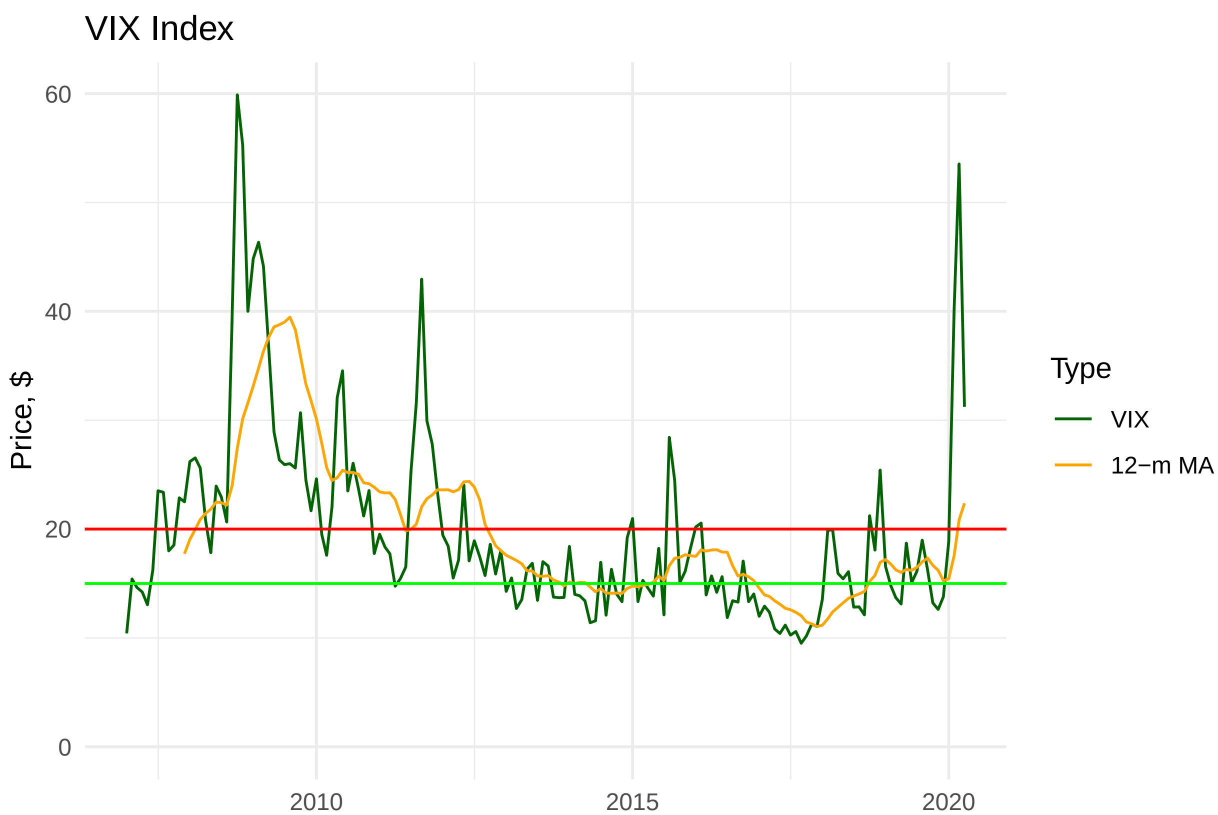
The correlation coefficient is 0.28 which supports our initial assumption.

Another way to analyze the discrepancy between the fund performance and VIX (and to finally make use of the class study material) is to perform a permutation test. But first, we encode VIX returns as categorical variables.

Categorical variables and in-month beta

We encode various vix regimes as categorical variables. There are tow categorical variables we introduced: RiskOn and RiskLevel. *OnOff* is a Boolean variable that is TRUE if the closing monthly value of VIX index is higher than the rolling 12-month moving average. *RiskLevel* is encoded as High, Normal, and Low risk, depending on whether the closing value of Index for this month is higher, between or lower than upper or lower bar we set. For this particular analysis, we set upper bound at 20 and lower bound at 15. These values are somewhat subjective. However, they look reasonable given the history of VIX Index.

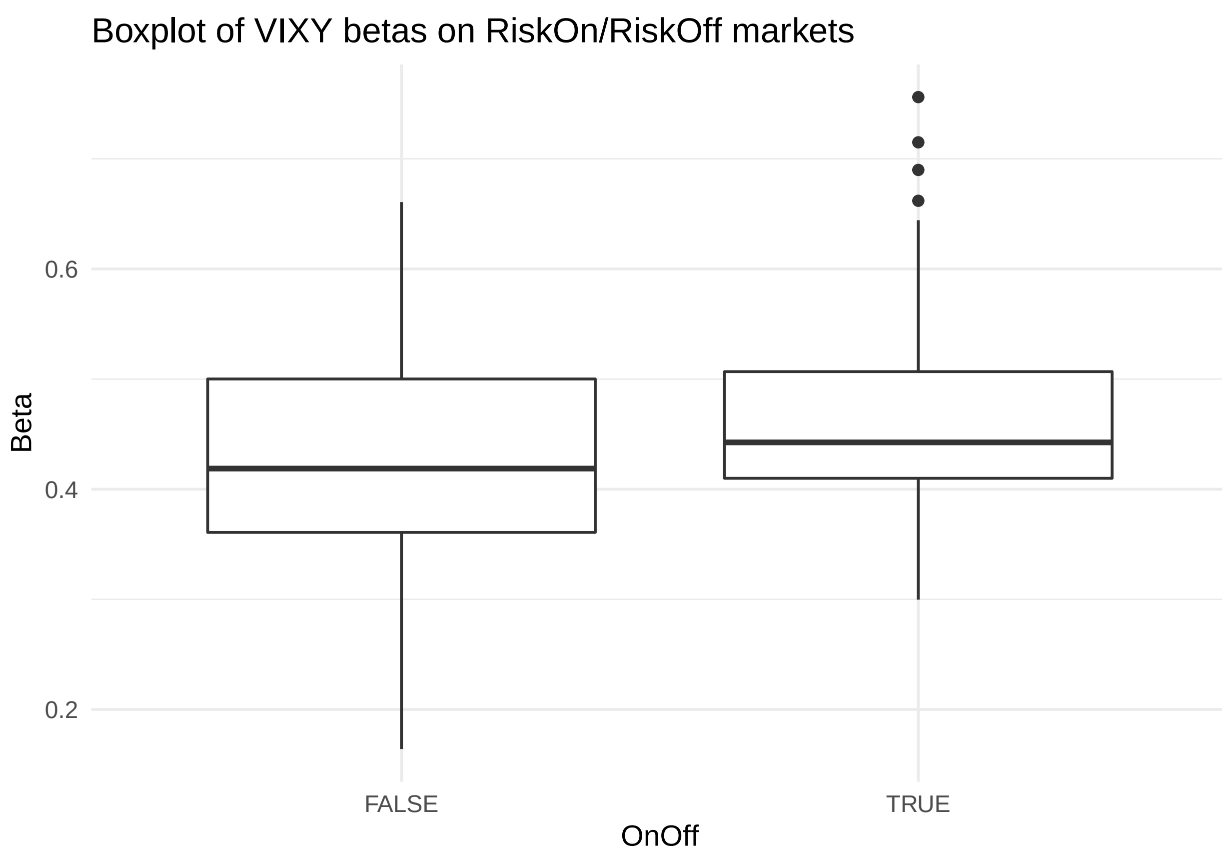
```
thresholds = c(15,20)
# Create vix dataframe
vix_df = to.monthly(vix)$vix.Adjusted
colnames(vix_df) = "VIX_Price"

# Calculate and add 12 month moving average
ma = rollapply(data = vix_df, width = 12, FUN = mean)
colnames(ma) = "MA"
vix_df = merge.xts(vix_df, ma)

# Plot
for_plot = data.frame(vix_df) %>% rownames_to_column("DT") %>%
  gather(., key="Type", value="Price", -DT)
for_plot$Type = factor(for_plot$Type, levels = c("VIX_Price", "MA"))
ggplot(for_plot, aes(x=as.Date(as.yearmon(DT)), y = Price, color=Type)) +
  geom_line() + xlab("") + ylab("Price, $") + ggtitle("VIX Index") +
  theme_minimal() +
  scale_color_manual(values=c("darkgreen", "orange"), labels=c("VIX", "12-m MA")) +
  geom_hline(aes(yintercept=thresholds[1]), color='green') +
  geom_hline(aes(yintercept=thresholds[2]), color='red') +
  ylim(0, max(for_plot$Price))
```



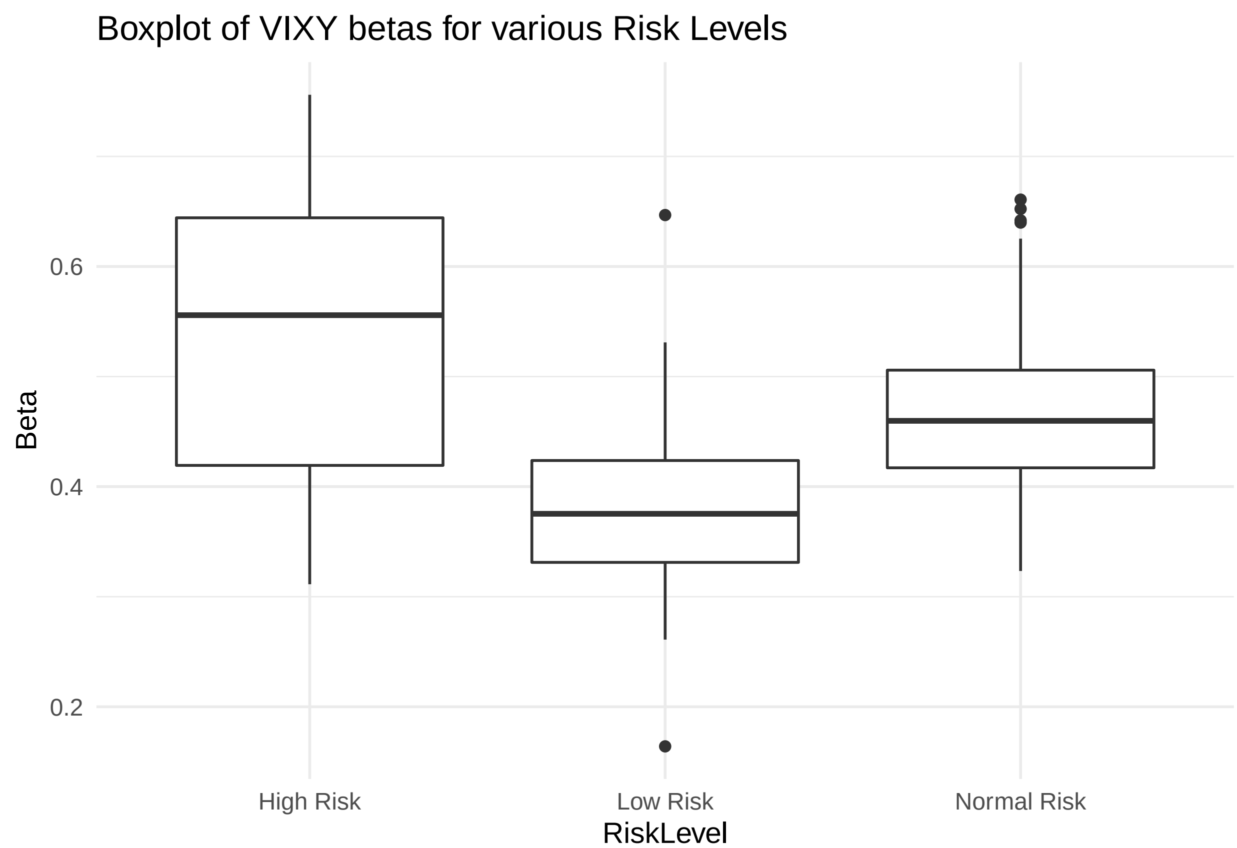
Next, we plot boxplots to see if the VIXY mean beta is different for various market regimes.

```
# Caluclate beta of daily returns
rr = merge.xts(vix_r[which(index(vix_r)>=min(index(vixy_r))),], vixy_r)
colnames(rr) = c("VIX","VIXY")


# Beta is a slope coefficient of the linear regression. We calculate it as cov/var

calcBeta = function(x,y){
  cov(x,y, use="complete.obs")/var(x, na.rm=T)
}


# Calculate monthly betas
df_m = apply.monthly(rr, function(x) calcBeta(x=x$VIX, y=x$VIXY))

# Create categorical variables
vix_df = data.frame(vix_df) %>% rownames_to_column('DT') %>%
  mutate(OnOff = VIX_Price > MA,
         RiskLevel = ifelse(VIX_Price > thresholds[2], 'High Risk',
                            ifelse(VIX_Price < thresholds[1], 'Low Risk',
                                   'Normal Risk')),
         RiskLevel = factor(RiskLevel),
         DT = as.yearmon(DT)
  )
```

```r
# Add betas
colnames(df_m) = "Beta"
df_m = data.frame(df_m) %>% rownames_to_column('DT')
df_m$DT = as.yearmon(df_m$DT)
vix_df$DT = as.yearmon(vix_df$DT)
vix_df = merge(df_m, vix_df, by='DT', all.x=T)

# Boxplots
for_plot = vix_df %>% dplyr::select(Beta, OnOff)
ggplot(for_plot, aes(x = OnOff, y=Beta)) + geom_boxplot() +
  theme_minimal() + ggtitle("Boxplot of VIXY betas on RiskOn/RiskOff markets")
```

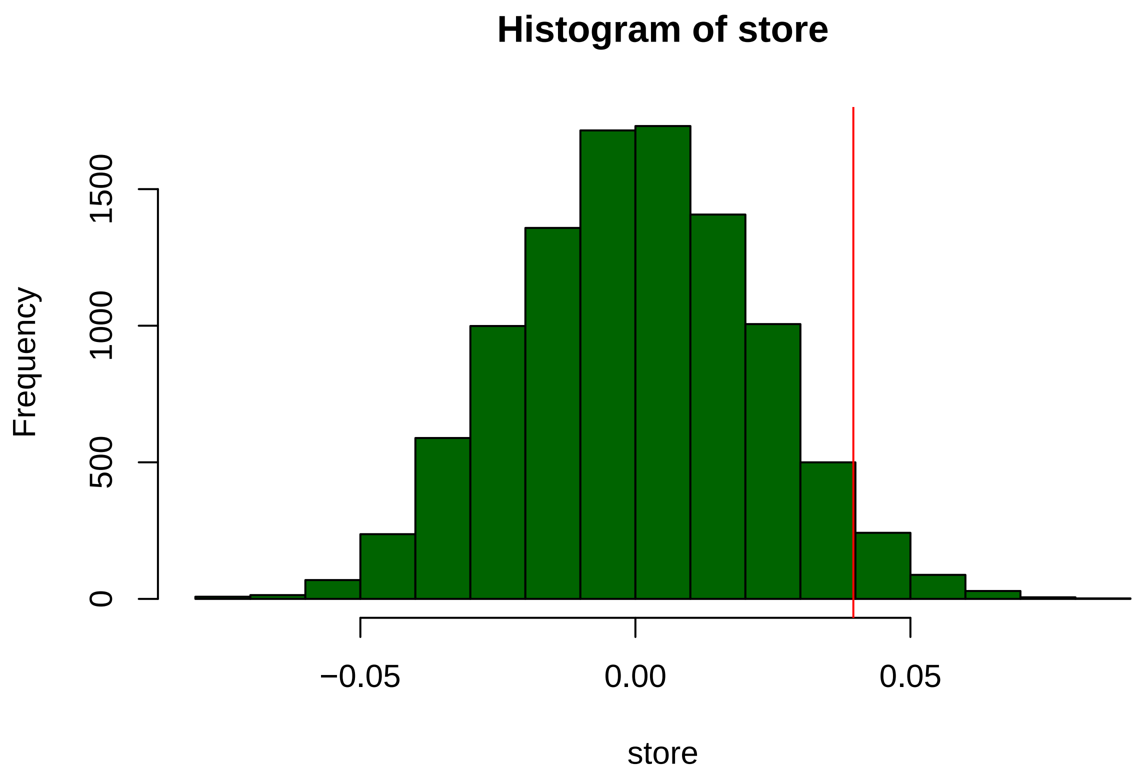



```r
# Boxplots
for_plot = vix_df %>% dplyr::select(Beta, RiskLevel)
ggplot(for_plot, aes(x = RiskLevel, y=Beta)) + geom_boxplot() +
  theme_minimal() + ggtitle("Boxplot of VIXY betas for various Risk Levels")
```



We see, that probably betas in various market regimes differ. We can formally check it by using a permutation test. Here, we compare if the mean beta differs for RiskOn and RiskOff months.

```
# Permutation test
# We check if the ETF beta differs in risk-on and risk-off months
tmp = vix_df %>% dplyr::select(OnOff, Beta)
N_on = nrow(tmp[tmp$OnOff,])

obs_diff = mean(tmp[tmp$OnOff==T,'Beta']) - mean(tmp[tmp$OnOff==F,'Beta'])

N = 10000
store = numeric(N)

for (i in 1:N){
  ind_on = sample(1:nrow(tmp), size = N_on, replace = F)
  mu_on = mean(tmp$Beta[ind_on])
  mu_off = mean(tmp$Beta[-ind_on])


  # Store simulated  diff
  store[i] = mu_on - mu_off
}


# Simple plot
hist(store, col='darkgreen')
abline(v = obs_diff, col='red')
```




```
# P-value
print(paste0("P-value:",(sum(store>obs_diff) + sum(store<(-obs_diff)))/N))
```

```
## [1] "P-value:0.0732"
```

With 10% confidence we may conclude that the mean betas for RiskOn and RiskOff periods differ.

We performed the same exercise (with similar conclusion) for correlation of daily returns. You can see it in details in Code.R (Question 2).

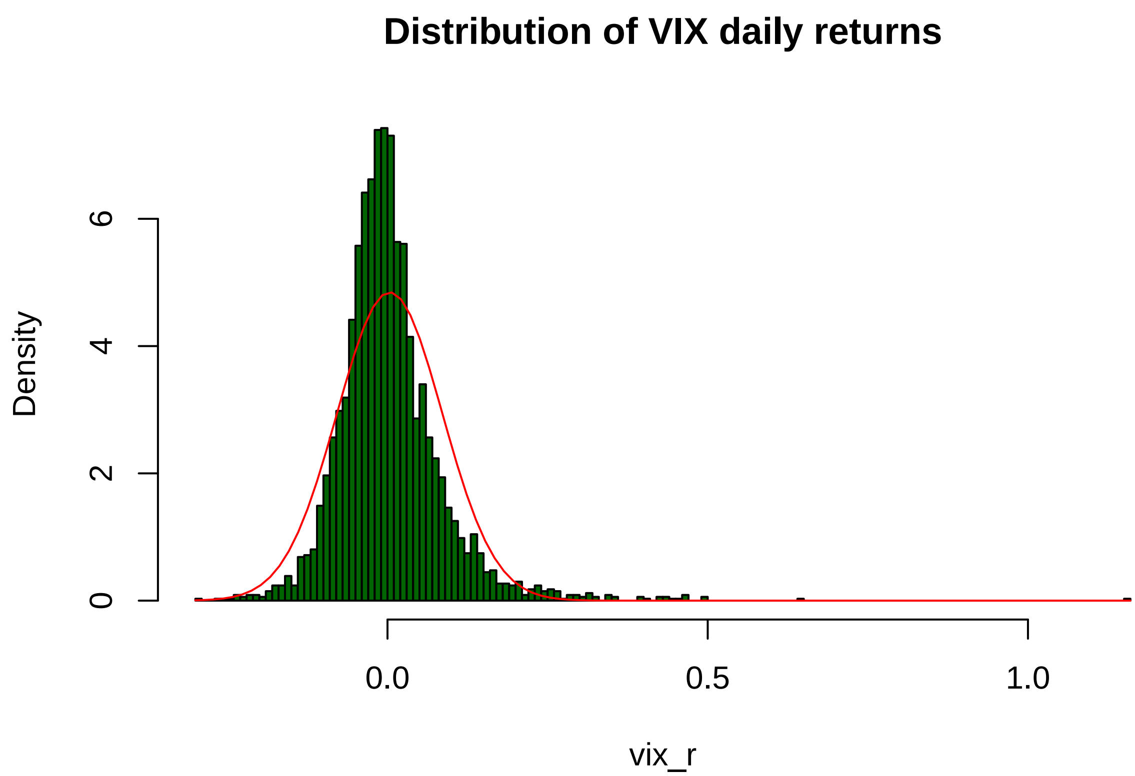
Distribution of VIX returns

Next, we switch our focus on VIX index itself. We would like to see if we can use the tools from Math23C to model VIX index.

First, we check if the index returns can be described by a normal distribution. Unfortunately, in financial theory returns of assets are often assumed to be normally distributed, which is often not the case in real life. (For more details, see Question 4 in Code.R)

```
# Can the returns of VIX index be described by normal distribution?
# First, let's compare the distribution of VIX returns with the normal distribution
ret = as.vector(vix_r$VIX.Adjusted) # Extract vector
ret = ret[!is.na(ret)] # Remove NA
mu = mean(ret)
sigma = sd(ret)
```

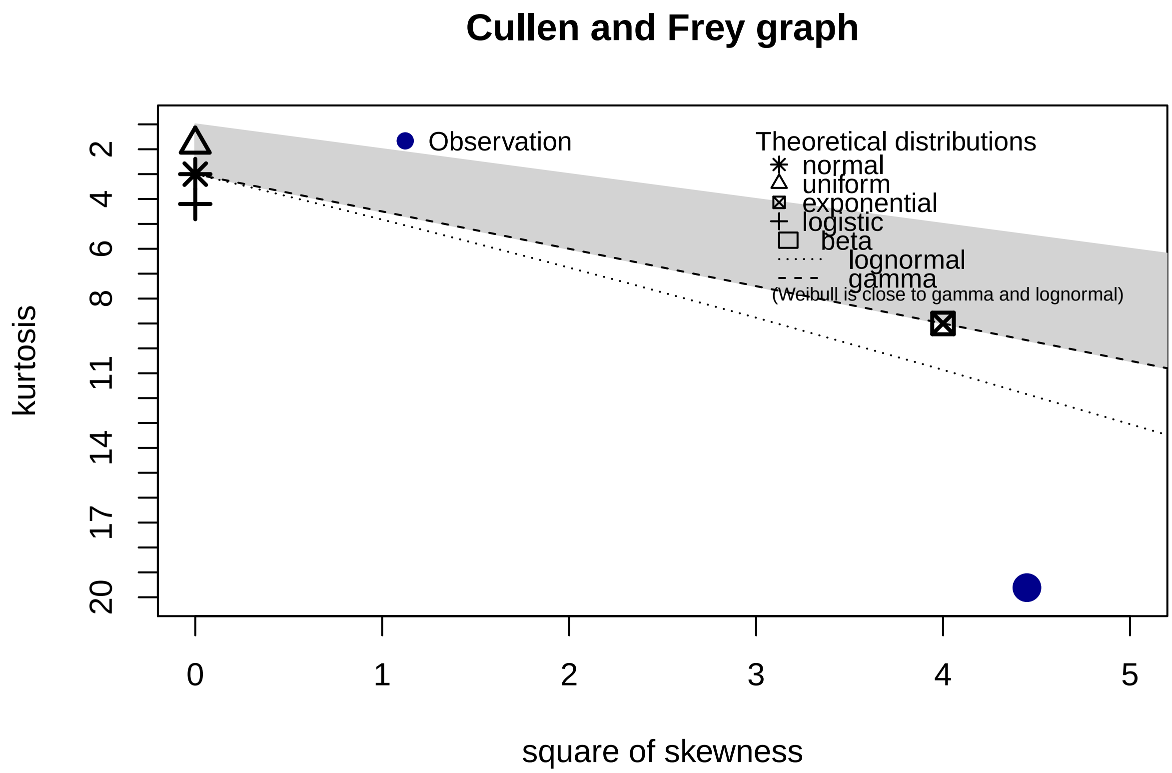
```r
hist(vix_r, col='darkgreen', main = "Distribution of VIX daily returns",
     breaks = 'fd', probability = T)
curve(dnorm(x, mean=mu, sd=sigma), add=T, col='red')
```



From the graph we can see that the distribution is far from normal. It is positively skewed and has high kurtosis.

We can check the parameters of the distribution using *fitdistrplus* package.

```r
descdist(ret)
```




```
## summary statistics
## ------
## min:  -0.2957265   max:  1.155979
## median:  -0.005982906
## mean:  0.003408601
## estimated sd:  0.08231429
## estimated skewness:  2.109208
## estimated kurtosis:  19.6144
```

```
# Probably it is going to be hard to find a well defined destribution for our set
```

Another way we can show the the fit is bad, is by using deciles analysis.

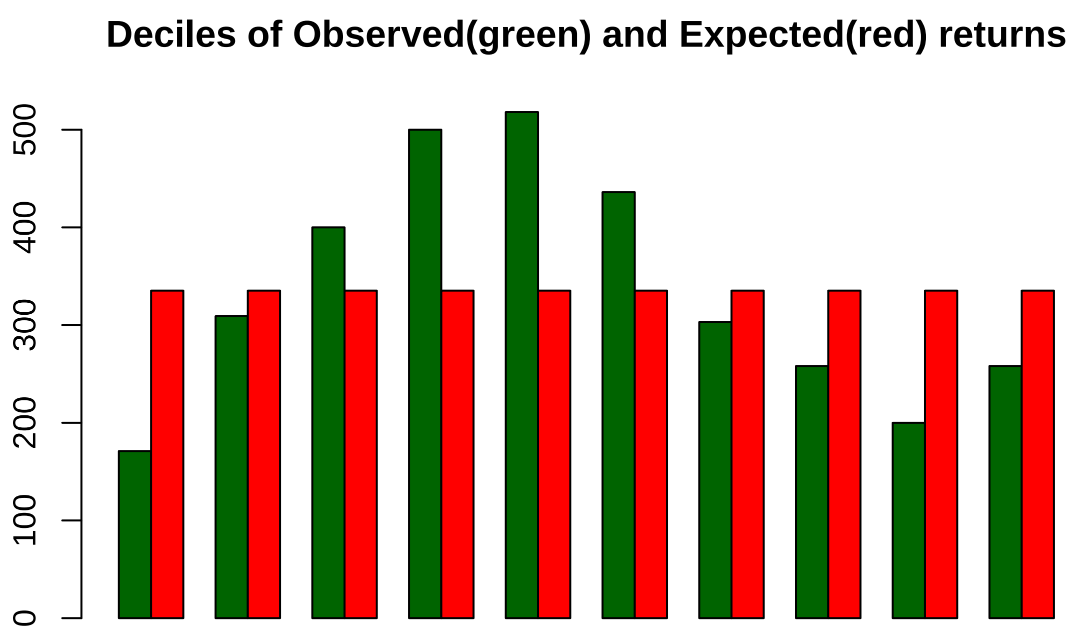
```
# We can formally check the fit
# Using deciles

nob = length(ret) # Number of observations
dec = qnorm(seq(0.0, 1, by = 0.1), mu, sigma) # Deciles
Exp = rep(nob/10,10) # Number Of expected observations per bin


binsim = numeric(10) # Bins
for (i in 1:10){
  binsim[i] <- sum((ret >= dec[i]) & (ret <= dec[i+1]) )
}
```

```
chisq.val = sum((binsim - Exp)^2/Exp)
pval =  pchisq(chisq.val, df = 7, lower.tail = F)
pval
```

```
## [1] 3.72066e-82
```

With p-value that low, we can definitely rejecty the hypothesis that the VIX return distribution can be described by normal distribution.

```
# We can further illustrate it by plotting expected vs observed bins

barplot(rbind(binsim, Exp), beside=T, col = c('darkgreen', 'red'),
        main = 'Deciles of Observed(green) and Expected(red) returns')
```



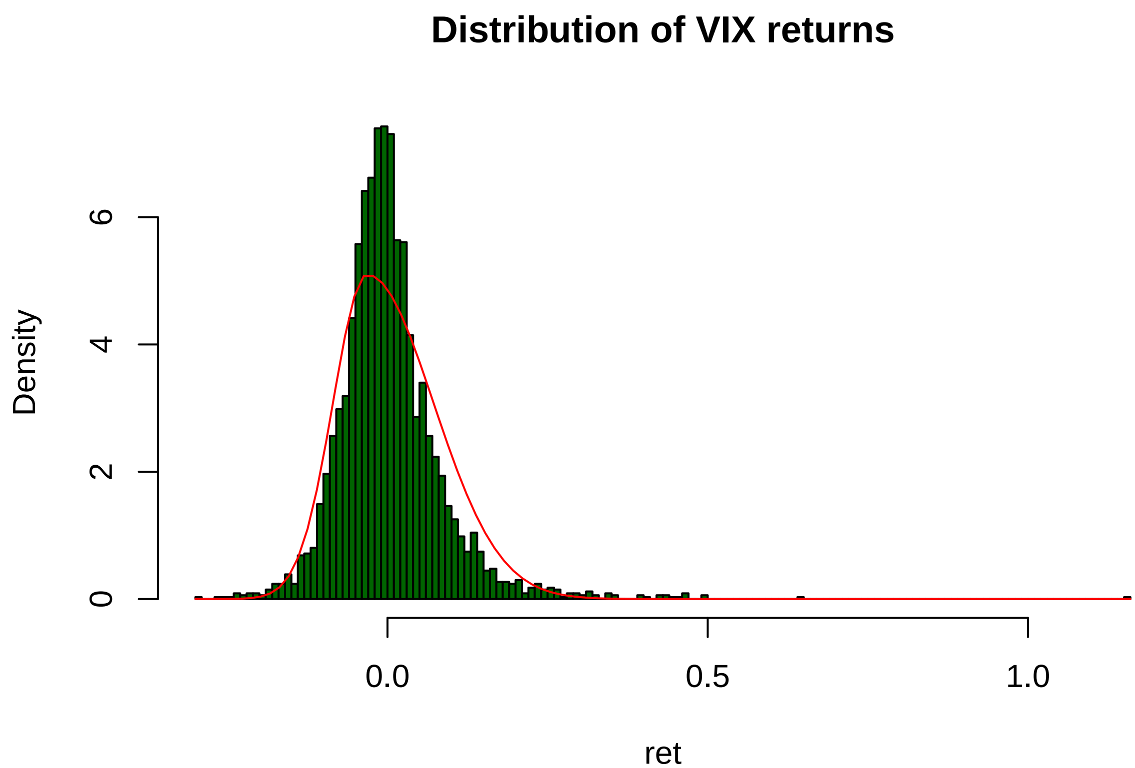
We tried different types of distributions, neither of which worked well (see Question 6 in Code.R).

Skewed normal distribution

Using *fGarch* package we tried to fit skewed normal distribution.

```
# Get VIX returns
ret = vix_r$VIX.Adjusted
ret = as.vector(ret[!is.na(ret)])

# Fit skewed normal
sn_par = snormFit(ret) # Use built in function to fit skewed distribution
mu = sn_par$par['mean']
sigma = sn_par$par['sd']
```

```r
xi = sn_par$par['xi'] #Skew

hist(ret, breaks="fd", col='darkgreen', main='Distribution of VIX returns', probability = T)
curve(dsnorm(x, mean=mu, sd=sigma, xi=xi), add=T, col="red")
```

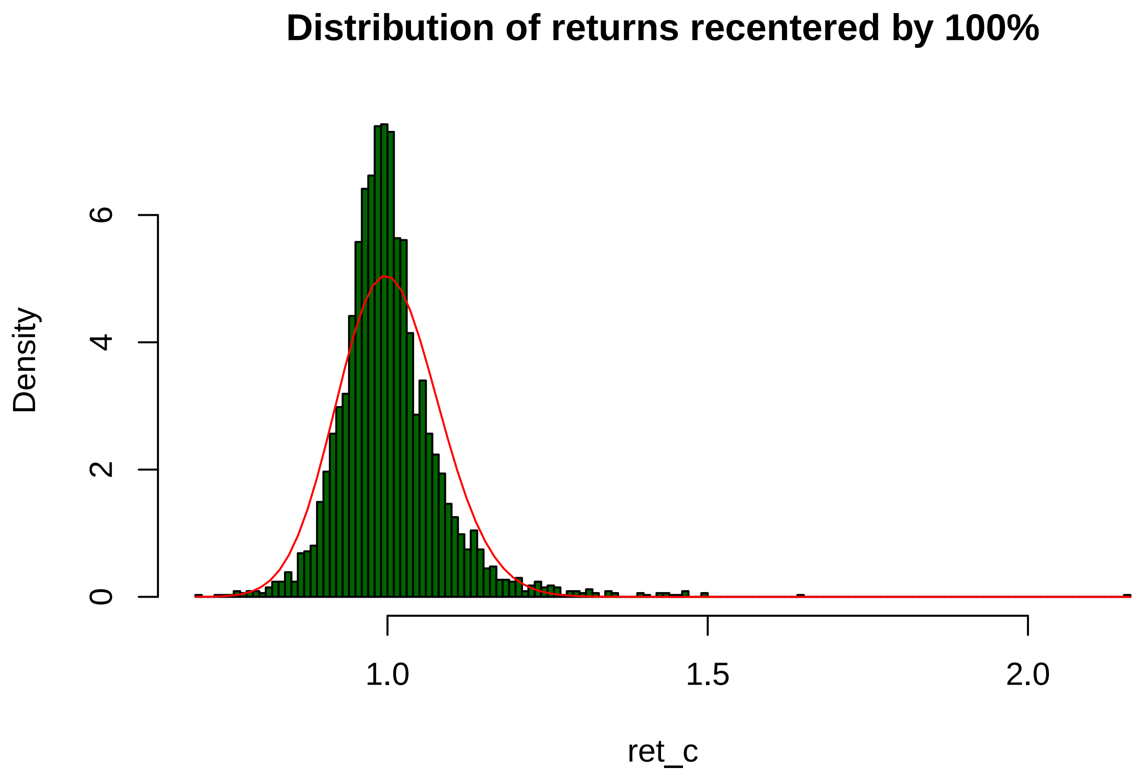


It dealt with skeweness, but not with the kurtosis. The p-value of ChiSq test was very low (O -50). Therefore, we rejected the hypothesis of skewed normal distribution of VIX returns.

Gamma distribution

Next, we recentered our observation based on the assumption that daily returns cannot be less than -100% (VIX index has to be positive). Now, when we are in positive range, we can fit gamma distribution.

```r
# Therefore, without the loss of generality we recenter the distribution of returns
ret_c = ret + 1 # Move distribution by one
ret_c = as.vector(ret_c)

# Now we can fit gamma distribution
pars = fitdist(ret_c, distr = "gamma")
shape = pars$estimate['shape']
rate = pars$estimate['rate']

# Plot histogram and curve
hist(ret_c, breaks='fd', col='darkgreen',
     main='Distribution of returns recentered by 100%', probability = T)
curve(dgamma(x, shape, rate), col='red', add=T)
```

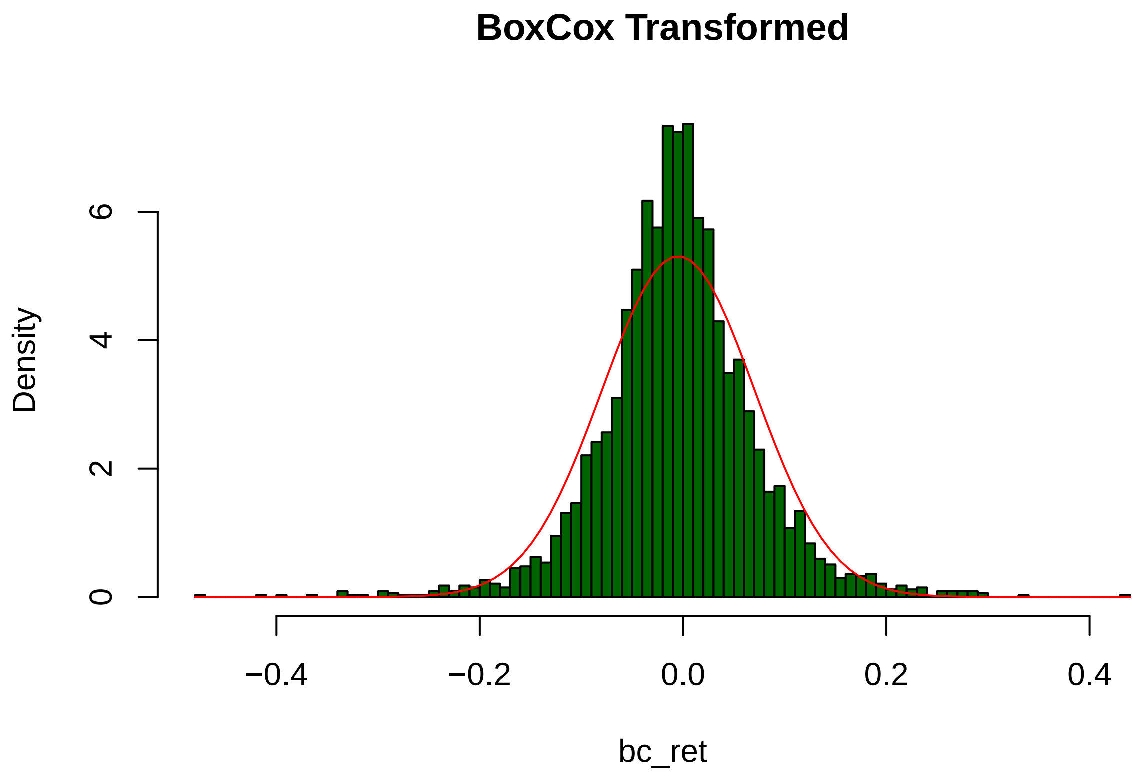


This trick still did not work: the culprit is kurtosis again. P-value was very low again (O-79) so we rejected this fit too.

BoxCox (power) transformation

Next, we tried to transform recentered returns data using power transformation in the form $\frac{y^\lambda - 1}{y}$. Theoretically, this transformation showd make data more normal-like.

```
lam = boxcox(ret_c, optimize = T)$lambda # Find optimal lambda.
                                          # We use returns +100% to make
                                          # sure that all data > 0
bc_ret = boxcoxTransform(ret_c, lam)
mm = mean(bc_ret)
sig = sd(bc_ret)

hist(bc_ret, breaks='fd', probability = T, col='darkgreen', main='BoxCox Transformed')
curve(dnorm(x, mean=mm, sd = sig), add=T, col='red')
```

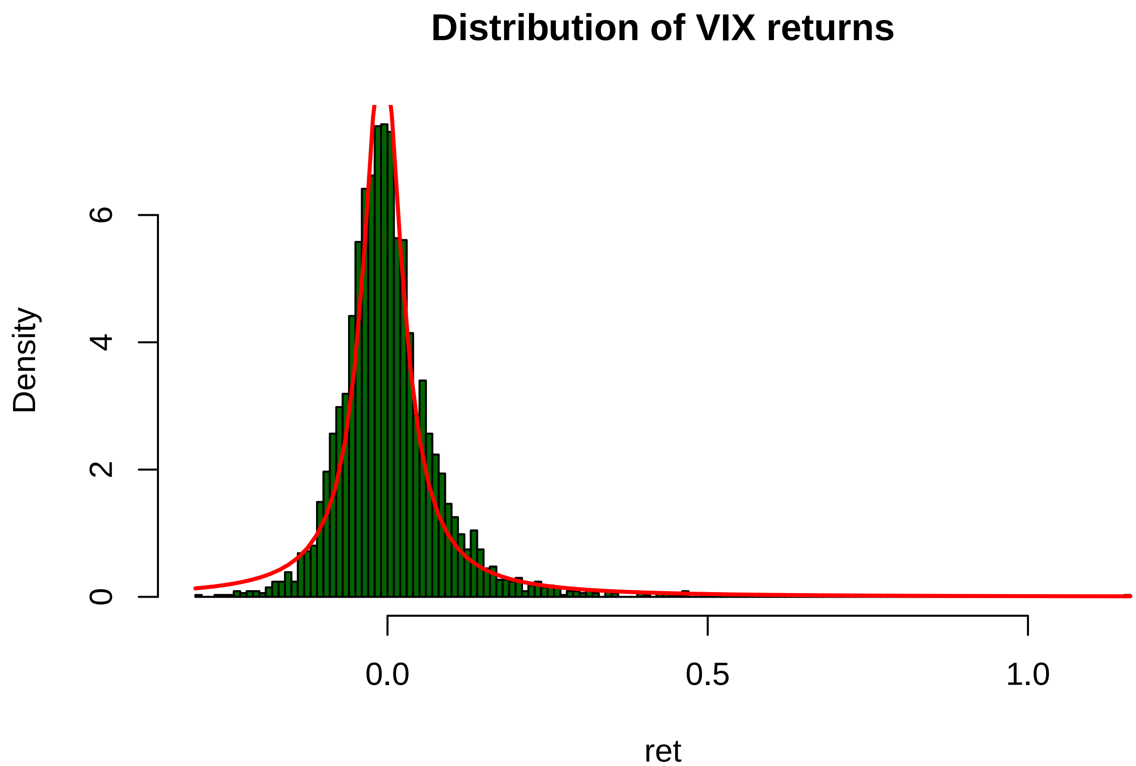


Although, the transformed data looks more symmetrical, the kurtosis is still too high to model with normal distribution. After performing the ChiSquare test we rejected this fit as well.

Cauchy distribution

Finally, we tested Cauchy distribution.

```
ret = as.vector(vix_r$VIX.Adjusted)
ret = ret[2:length(ret)]
pars =fitdist(ret, "cauchy")
loc =  pars$estimate['location']
sc = pars$estimate['scale']

# Plotting
hist(ret, breaks="fd", probability = T, col='darkgreen',
     main='Distribution of VIX returns')
curve(dcauchy(x, loc, sc), add=T, col='red',
      lwd=2)
```



The fit looked much better, but unfortunately, it failed to model tail risks well. Given that we had so many observations, our p-value was still to low to accept this fit.

Modelling VIX as a process with drift and noise

Important disclaimer. What follows next is more of an illustration of using math tools for various modelling purposes and is not intended to be perfectly accurate from the point of view of financial engineering.

The motivation of this chapter comes from the fact that in financial engineering stock returns are usually modelled as a stochastic process in the form $\frac{dS_t}{S_t} = \mu dt + \sigma dX_t$, where S_t is a price of an asset, μ is expected return, σ is standard deviation of returns, and dX_t is Brownian motion. Implicitly, the model assumes that stock returns are normally distributed, s.t. you can describe the process using mean and standard deviation.

We consider if we can separately model the noise term based on observed standard deviations of daily returns, and the drift term, using Lagrange interpolation.

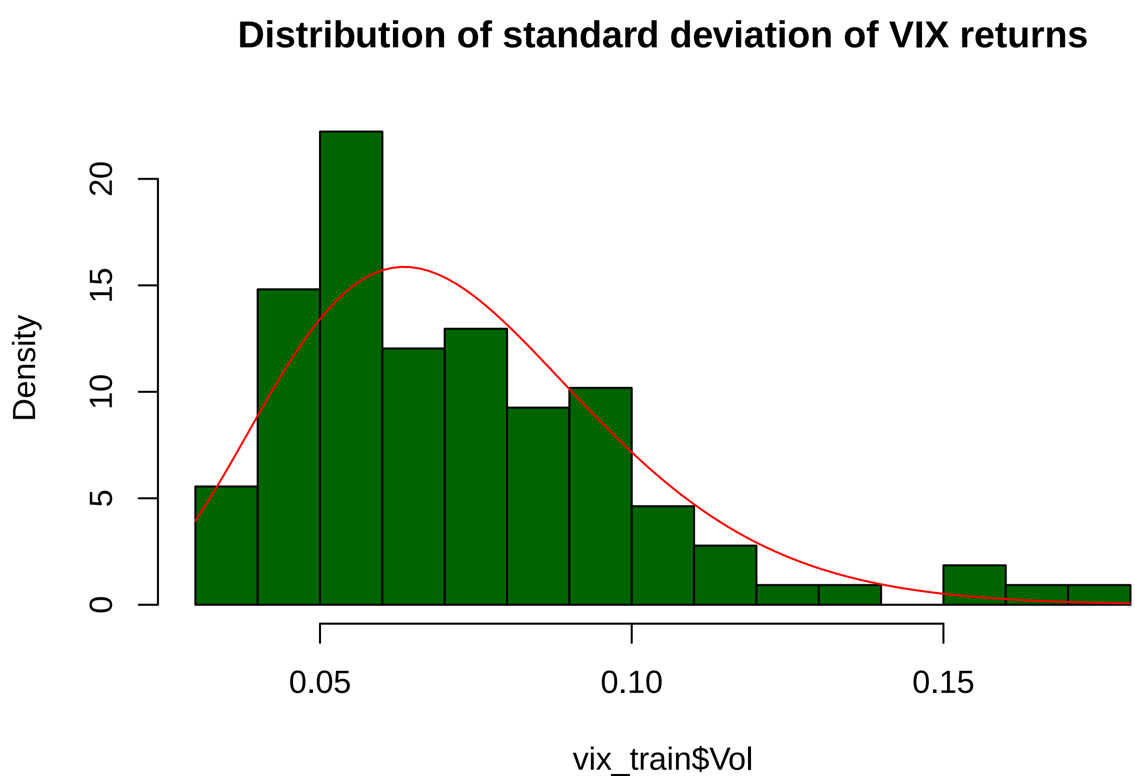
Modelling standard deviation of VIX returns.

For more information, please refer to Question 3 in Code.R

First, we divide the dataset into train and test dataset, so that we could assess the performance of the model in out-of-sample period. We use January 2016 as the cut-off time. We calculate volatility (standard deviation) of daily returns for each month in training period and fit beta distribution to model it.

```
vix_vol = vix_r[-1,] # Remove the first day (it is NA)
colnames(vix_vol) = 'VIX'

# Convert to dataframe, calculate monthly volatility of daily returns
```

```r
vix_vol = data.frame(vix_vol) %>% rownames_to_column('DT') %>%
  mutate(Month = as.yearmon(DT)) %>% group_by(Month) %>%
  summarize(Vol = sd(VIX))

# Divide to train and test sets to check how well it works out-of-sample
date_split = as.yearmon(as.Date("2016-01-01"))
vix_train = vix_vol[which(vix_vol$Month<date_split),]
vix_test = vix_vol[which(vix_vol$Month>=date_split),]


# Fit beta distribution
fd =  fitdist(vix_train$Vol, 'beta')

coefs = fd$estimate
shape1 = coefs[1]
shape2 = coefs[2]

# Plot fit
hist(vix_train$Vol, breaks='FD', col='darkgreen', probability = T,
     main = 'Distribution of standard deviation of VIX returns')
curve(dbeta(x, shape1, shape2), col='red', add=T)
```



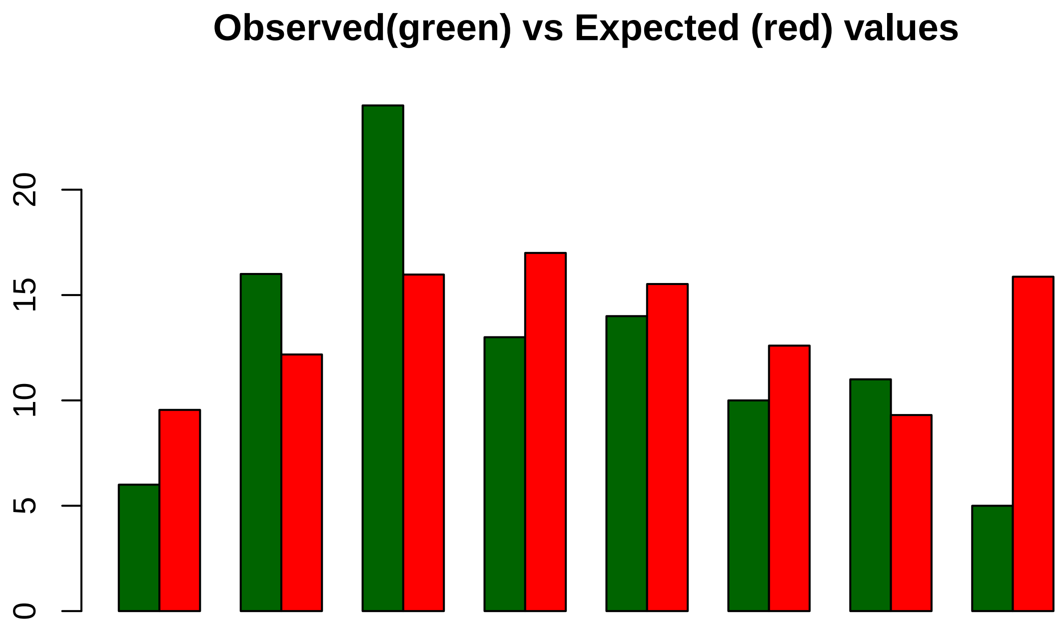
We can check the fit using ChiSq test.

```r
# Check fit
h = hist(vix_train$Vol, breaks='FD', plot = F) # Get histogram object
```

```r
cnt = h$counts
br  = h$breaks

# Create dataframe for test
test_df = data.frame(Breaks = br[2:length(br)],
                     Obs = cnt)

#Combine observations to get at lest 5
which(test_df$Obs<=5)
```

```
## [1]  8  9 10 11 12 13 14 15
```

```r
cut_off = 8
test_df$Breaks[cut_off] = sum(test_df$Obs[cut_off:nrow(test_df)])
test_df = test_df[1:cut_off,]

# Add expected
N_obs = sum(cnt)

tot = 0 # TO store cumulative probability
for (i in 1:nrow(test_df)){
  if (i!=nrow(test_df)){
    test_df[i,'Exp'] = N_obs * (pbeta(test_df$Breaks[i], shape1, shape2) - tot)
    tot = pbeta(test_df$Breaks[i], shape1, shape2)
  }
else {
    test_df[i,'Exp'] = N_obs * (1 - tot)
}
}


# Plot
barplot(rbind(test_df$Obs, test_df$Exp), beside = T, col=c('darkgreen', 'red'),
        main = 'Observed(green) vs Expected (red) values')
```



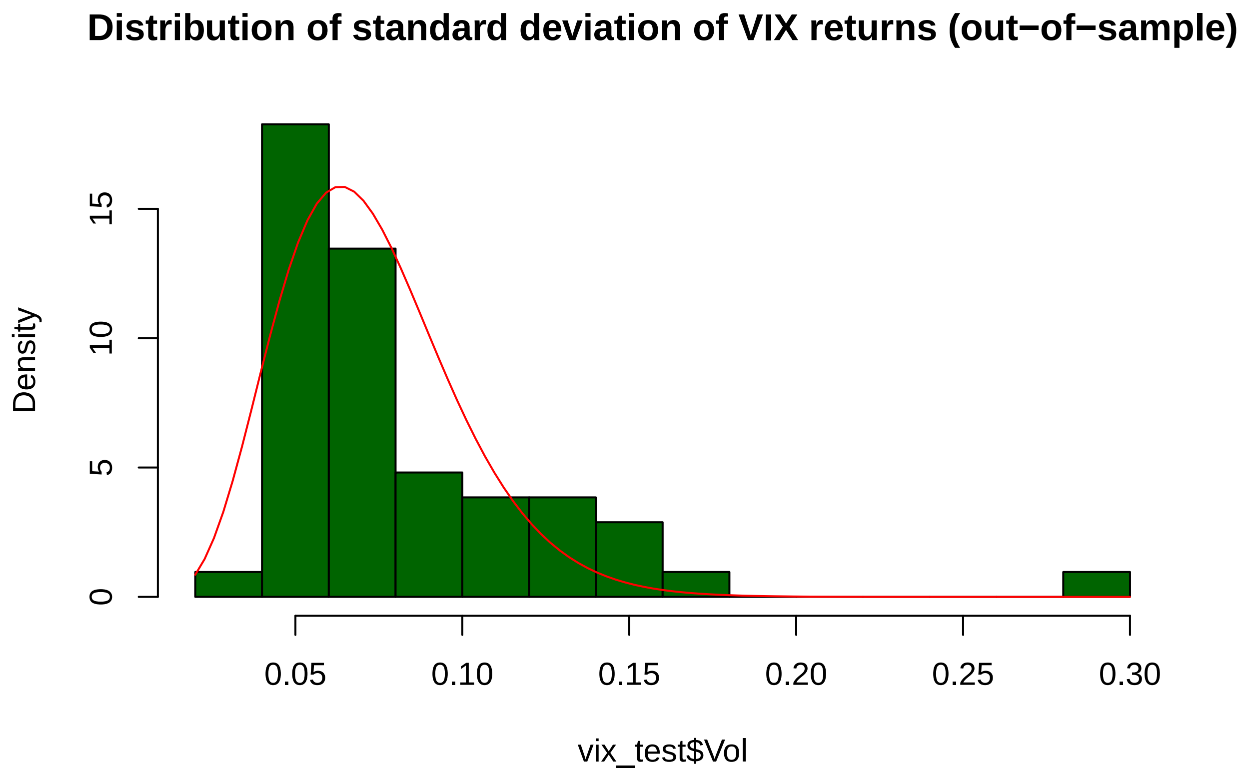

```r
# Calculate chisq
chisq.val = sum((test_df$Obs - test_df$Exp)^2/test_df$Exp)

# Calculate p-value
pval =  pchisq(chisq.val, df = length(cnt)-3, lower.tail = F)
print(paste0("p-value: ", pval))
```

```
## [1] "p-value: 0.194345819194613"
```

p-value is above 10%, so with 10% confidence level we cannot reject the null hypothesis that beta distribution is a good fit for the distribution of VIX index return volatility.

Next, we can test it on the out-of-sample data.

```r
# Plot fit
hist(vix_test$Vol, breaks='FD', col='darkgreen', probability = T,
     main = 'Distribution of standard deviation of VIX returns (out-of-sample)')
curve(dbeta(x, shape1, shape2), col='red', add=T)
```

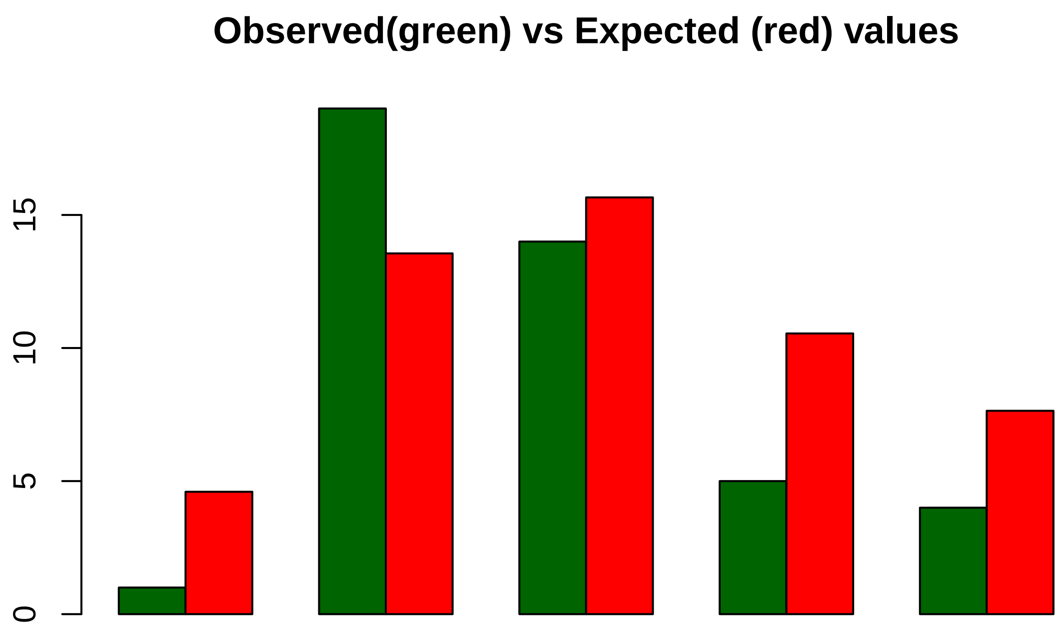



```r
# Check fit
h = hist(vix_test$Vol, breaks='FD', plot = F) # Get histogram object
cnt = h$counts
br  = h$breaks

# Create dataframe for test
test_df = data.frame(Breaks = br[2:length(br)],
                     Obs = cnt)

#Combine observations to get at lest 5
which(test_df$Obs<=5)
```

```
##  [1]  1  4  5  6  7  8  9 10 11 12 13 14
```

```r
cut_off = 5
test_df$Breaks[cut_off] = sum(test_df$Obs[cut_off:nrow(test_df)])
test_df = test_df[1:cut_off,]

# Add expected
N_obs = sum(cnt)

tot = 0 # TO store cumulative probability
for (i in 1:nrow(test_df)){
  if (i!=nrow(test_df)){
    test_df[i,'Exp'] = N_obs * (pbeta(test_df$Breaks[i], shape1, shape2) - tot)
    tot = pbeta(test_df$Breaks[i], shape1, shape2)
  }
```

```r
  else {
    test_df[i,'Exp'] = N_obs * (1 - tot)
  }
}


# Plot
barplot(rbind(test_df$Obs, test_df$Exp), beside = T, col=c('darkgreen', 'red'),
        main = 'Observed(green) vs Expected (red) values')
```




```r
# Calculate chisq
chisq.val = sum((test_df$Obs - test_df$Exp)^2/test_df$Exp)

# Calculate p-value
pval =  pchisq(chisq.val, df = length(cnt)-3, lower.tail = F)
print(paste0("p-value: ", pval))
```

```
## [1] "p-value: 0.545669908739016"
```

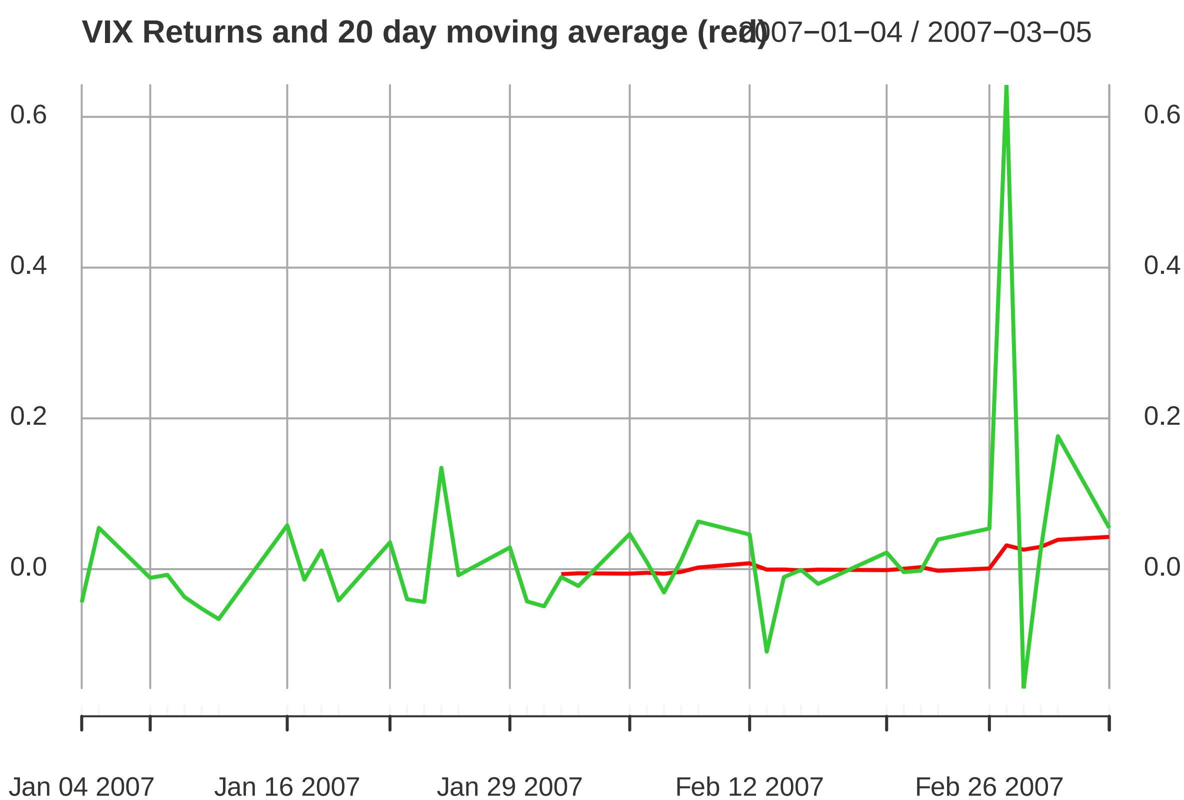
p-value is even higher. And again we fail to reject the null, and conclude that beta distribution may be a good fit.

Using Lagrange extrapolation to forcast VIX

For more detail, refer to Question 5 in Code.R

We use exponensial 20-day moving average to smoothen the percent change (return) of VIX index, then we use Lagrange interpolation (in fact, extrapolation) to forecast the expected return of the VIX Index.

```
# First, an example script, and then we will put it into a function
tmp = vix_r$VIX.Adjusted[2:42]

# Calculate MA
tmp$MA = rollapply(tmp$VIX.Adjusted, 20, mean)


# Plot VIX vs EMA
plot(tmp, col=c("limegreen", "red"), main='VIX Returns and 20 day moving average (red)')
```




```
# Exctract vector of MA
vec = as.vector(tmp$MA)[(nrow(tmp)-19):nrow(tmp)]; vec
```

```
##  [1] -0.0059124099 -0.0048618700 -0.0060361658 -0.0036075832  0.0021688551
##  [6]  0.0077780199 -0.0005978069 -0.0004313978 -0.0017078468 -0.0006125877
## [11] -0.0012936295  0.0005073440  0.0025853203 -0.0021740068  0.0009208246
## [16]  0.0315908872  0.0258387652  0.0295992857  0.0389450719  0.0428009504
```

```
pnt = c(1, 5, 10, 15, 20) # Fourth degreepolynomial

PInv = cbind(rep(1, length(pnt)), pnt, pnt^2, pnt^3, pnt^4); PInv
```

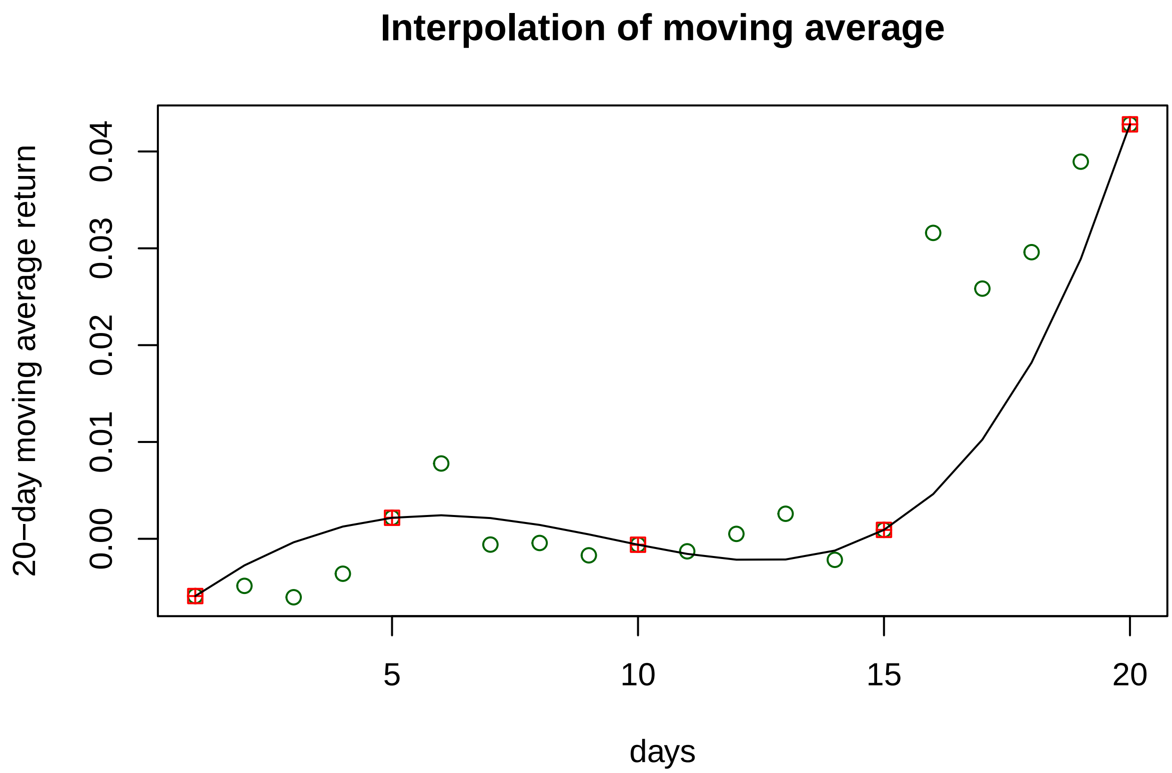
```
##         pnt
## [1,] 1   1   1    1      1
## [2,] 1   5  25  125    625
## [3,] 1  10 100 1000  10000
```

```
## [4,] 1  15 225 3375  50625
## [5,] 1  20 400 8000 160000
```

```
P = solve(PInv); P
```

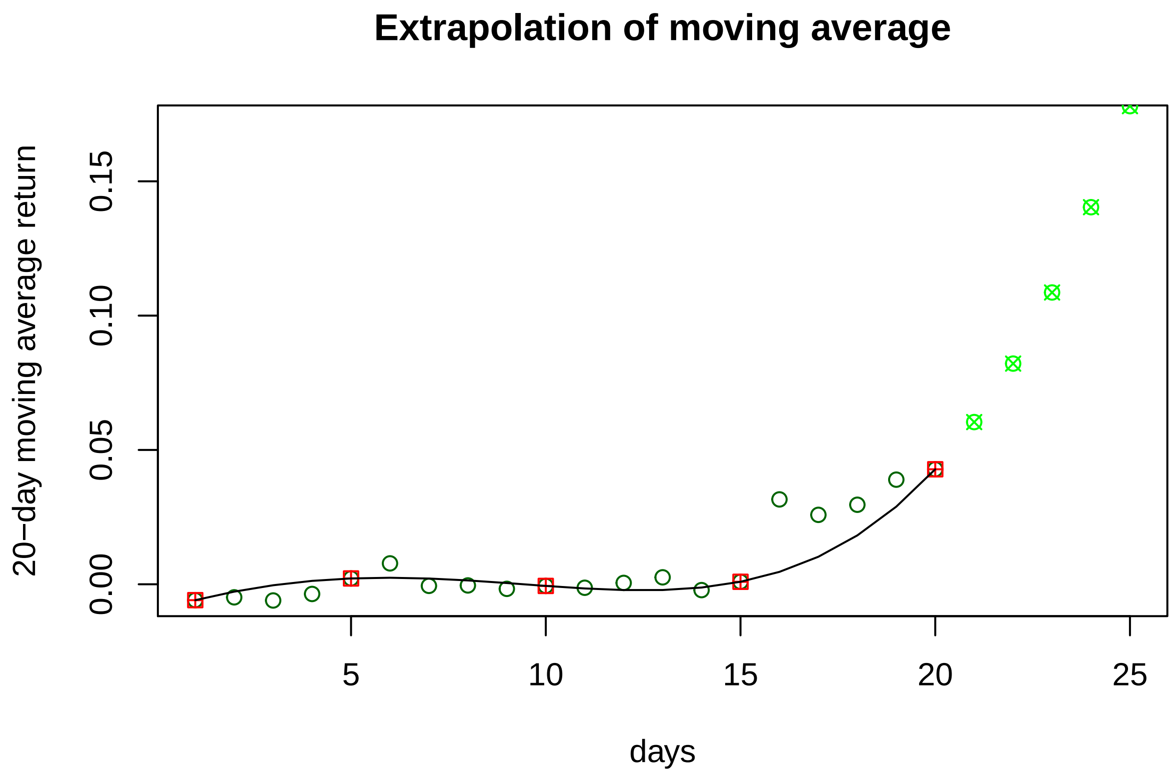
```
##               [,1]          [,2]          [,3]          [,4]          [,5]
##      1.5664160401 -1.0000000000  0.6666666667 -0.2857142857  5.263158e-02
## pnt -0.6526733500  1.2166666667 -0.8777777778  0.3857142857 -7.192982e-02
##      0.0913742690 -0.2316666667  0.2288888889 -0.1100000000  2.140351e-02
##     -0.0052213868  0.0153333333 -0.0182222222  0.0102857143 -2.175439e-03
##      0.0001044277 -0.0003333333  0.0004444444 -0.0002857143  7.017544e-05
```

```
# Interpolation function is based on Paul's lecture code
interp = function(x, val){
  c(1,x,x^2,x^3,x^4) %*% P %*% val
}

# Making the function accept vectors
vect_int = function(v){
  out = c()
  for (i in v){
    out= c(out, interp(i, vec[pnt]))
  }

  out
}


# Test interpolation function, extrapolate
plot(vec, col='darkgreen', main = 'Interpolation of moving average',
     xlab="days", ylab = '20-day moving average return') # Observed values
points(pnt, vec[pnt], pch=12, col="red") # Highlight points used to interpolate
lines(vect_int(c(1:20)), type='l')
```



Now we can use it to extrapolate expected returns.

```
# Use it to extrapolate
plot(c(vec, rep(NA,5)), ylim = c(min(vec)*0.8,max(vec)*4),
     col='darkgreen', main = 'Extrapolation of moving average',
     xlab="days", ylab = '20-day moving average return') # Observed values
points(pnt, vec[pnt], pch=12, col="red") # Highlight points used to interpolate
lines(vect_int(c(1:20)), type='l')
points(21:25, vect_int(c(21:25)), pch=13, col='green')
```



We already can see a potential problem: if the observed values exhibit exponential growth at the end of the period, the model will likely forecast the continuation of the exponential growth. But again, the goal of this paper is to show application of tools rather than make a break through in financial theory.

Putting things together

We create a function that uses extrapolated mean returns and MonteCarlo simulation based on volatility term drawn from beta distribution to make a forecast for the next 5 trading days.

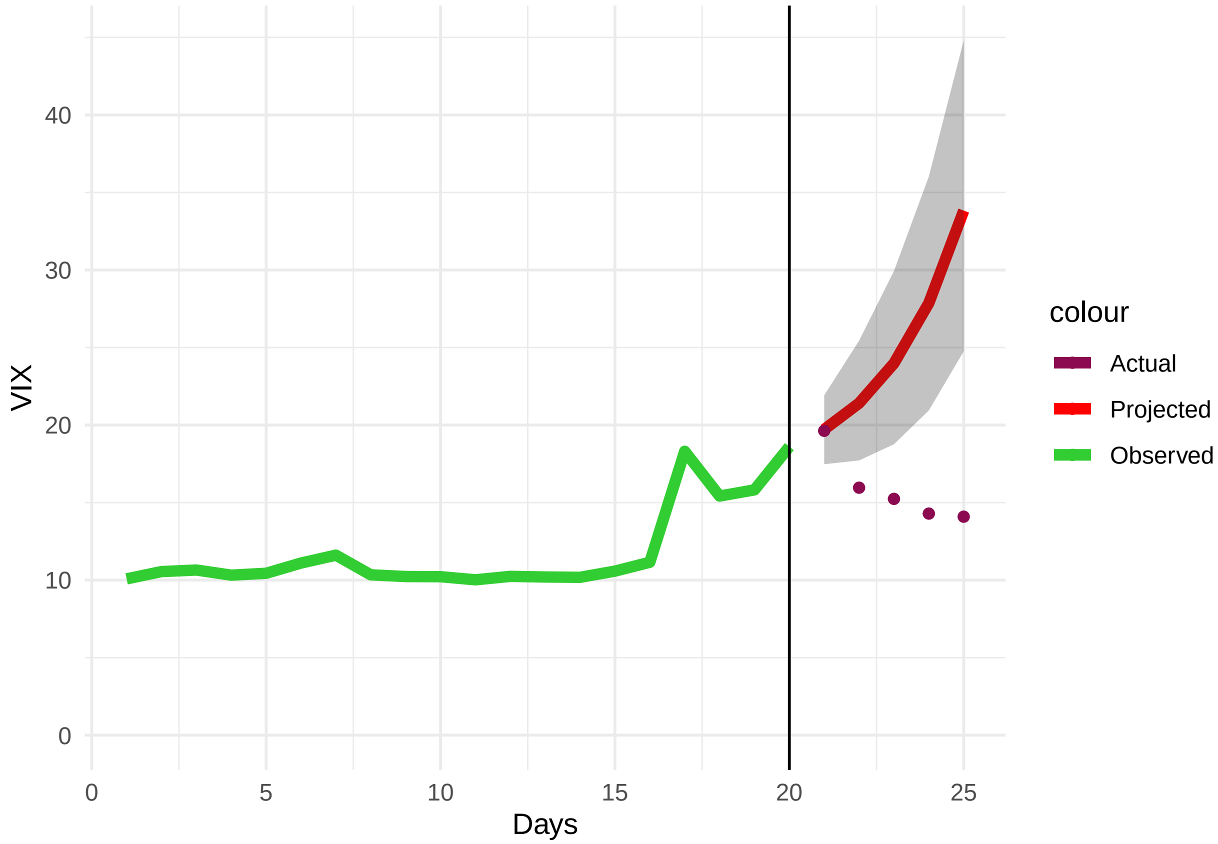
```
# Lets put together a versatile function.
ModelVol = function(observed, shape1, shape2, plot = TRUE, known = NA){
  # Calculate returns from prices
  ret = observed/lag(observed) -1
  ret = ret[2:nrow(observed),]

  # Calculate moving average


  # Check that we have enough data
  if (nrow(ret)<40){
    stop("Please supply at least 41 observations")
  }

  # Keep last 20 observations
  vec = as.vector(rollapply(ret, 20, mean))
  vec = vec[(length(vec)-19):length(vec)]
```

```r
# Use Lagrange extrapolation
pnt = c(1, 5, 10, 15, 20) # Fourth degreepolynomial

PInv = cbind(rep(1, length(pnt)), pnt, pnt^2, pnt^3, pnt^4)
P = solve(PInv)


# Making the function accept vectors
vect_int = function(v){

  interp = function(x, val){
    c(1,x,x^2,x^3,x^4) %*% P %*% val
  }

  out = c()
  for (i in v){
    out= c(out, interp(i, vec[pnt]))
  }

  out
}

# Extrapolate ma for the next five days
extra = vect_int(21:25)

# Simulate 10000 return paths
N = 10000

noise = matrix(rbeta(5*N, shape1, shape2), ncol=5)

paths = matrix(rep(extra, N), ncol=5, byrow = T) + noise * matrix(sample(c(-1,1),5*N, replace = T), n

paths = t(apply(paths ,1, function(x) cumprod(1+x))) # Compound return

last_price = as.vector(observed[nrow(observed),])
prices =  matrix(rep(last_price, 5*N), ncol=5) * paths

# Find top/bottom quantiles
bound = apply(prices,2, function(x) quantile(x, probs = c(0.025, 0.975)))



# Create dataframe
obs = as.vector(observed[(nrow(observed)-19):nrow(observed),])
tmp = data.frame(Observed = c(obs,
                              rep(NA, 5)),
                 Trend = c(rep(NA,20), last_price * cumprod(1+extra)),
                 Upper = c(rep(NA,20), bound[2,]),
                 Lower = c(rep(NA,20), bound[1,])
                 )
```

```r
  # Add known if we have it
  if (!is.na(known)){
    tmp$Actual[21:25] = known[1:5,1]
  } else {
    tmp$Actual = NA
  }

  # Plot
  if (plot==TRUE){
     g = ggplot(tmp, aes(x=1:25, y=Trend)) + geom_line(aes(y=Trend, color='limegreen'), size=2)  +
      geom_line(aes(y=Observed, color = 'red'), size=2) +
      geom_ribbon(aes(ymin=Lower, ymax=Upper), alpha=0.3) +
      geom_point(aes(y=Actual, color='deeppink4')) +
      ylim(0,max(tmp))  +
      xlim(1,25) +
      scale_color_manual(values = c('deeppink4', 'red', 'limegreen'),
                                      labels=c("Actual","Projected","Observed")) +
      theme_minimal() + xlab("Days") + ylab("VIX") +
      geom_vline(aes(xintercept=20))
  }

  list(DF = tmp, Plot=g)
}
```

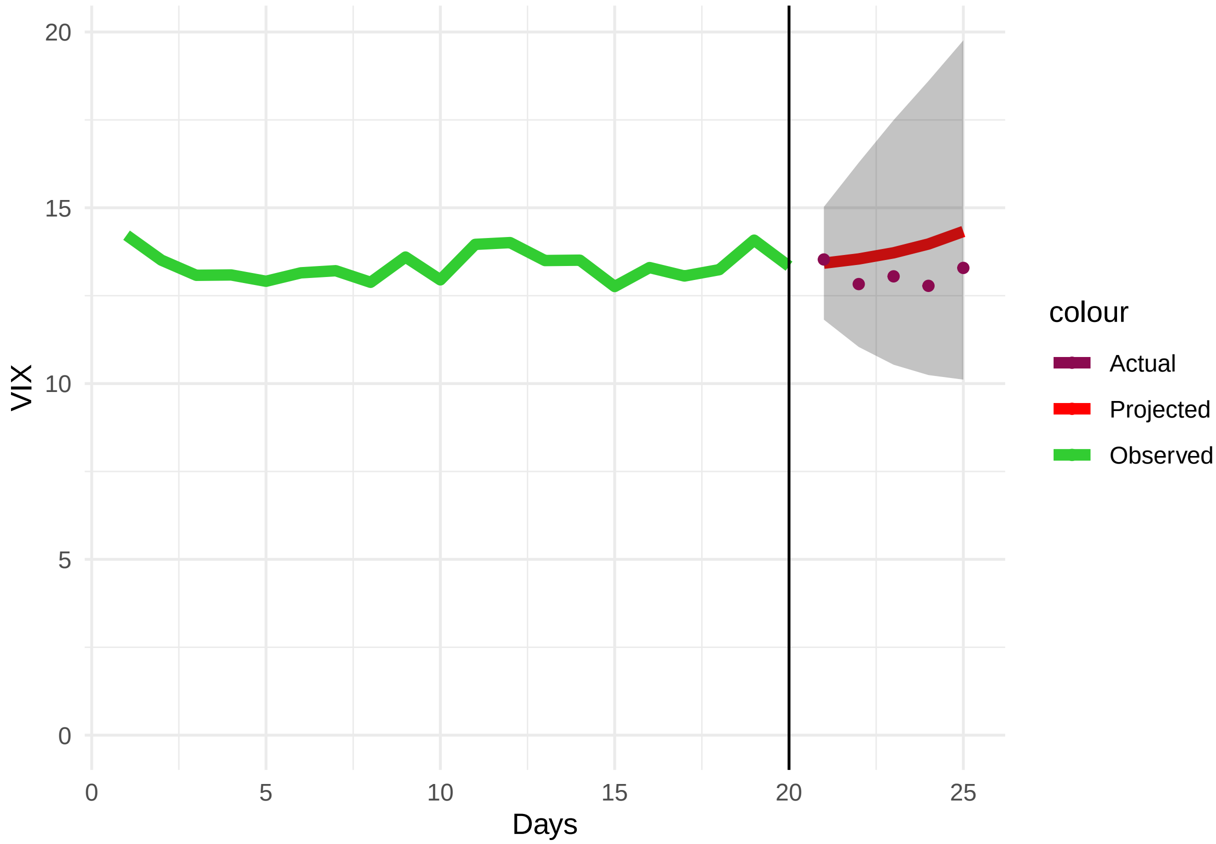
The resulting model, as expected, does a bad job for market regimes that change abruptly.

```r
# Does terrible job for market shocks
ModelVol(vix$VIX.Adjusted[1:41], shape1, shape2, known = vix$VIX.Adjusted[42:47])$Plot
```



However, the performance for fairly calm markets is adequate.

```
# But ok job for calm markets
ModelVol(vix$VIX.Adjusted[60:100], shape1, shape2,, known = vix$VIX.Adjusted[101:105])$Plot
```



Using historic data to forecast the future

Another way to deal with the nasty distribution we have on hand, is to draw directly from historic distribution of returns. See Question 7 in Code.R

Using historic data, we get the expectation and top and bottom 2.5% of VIX returns, and use them to model future returns.

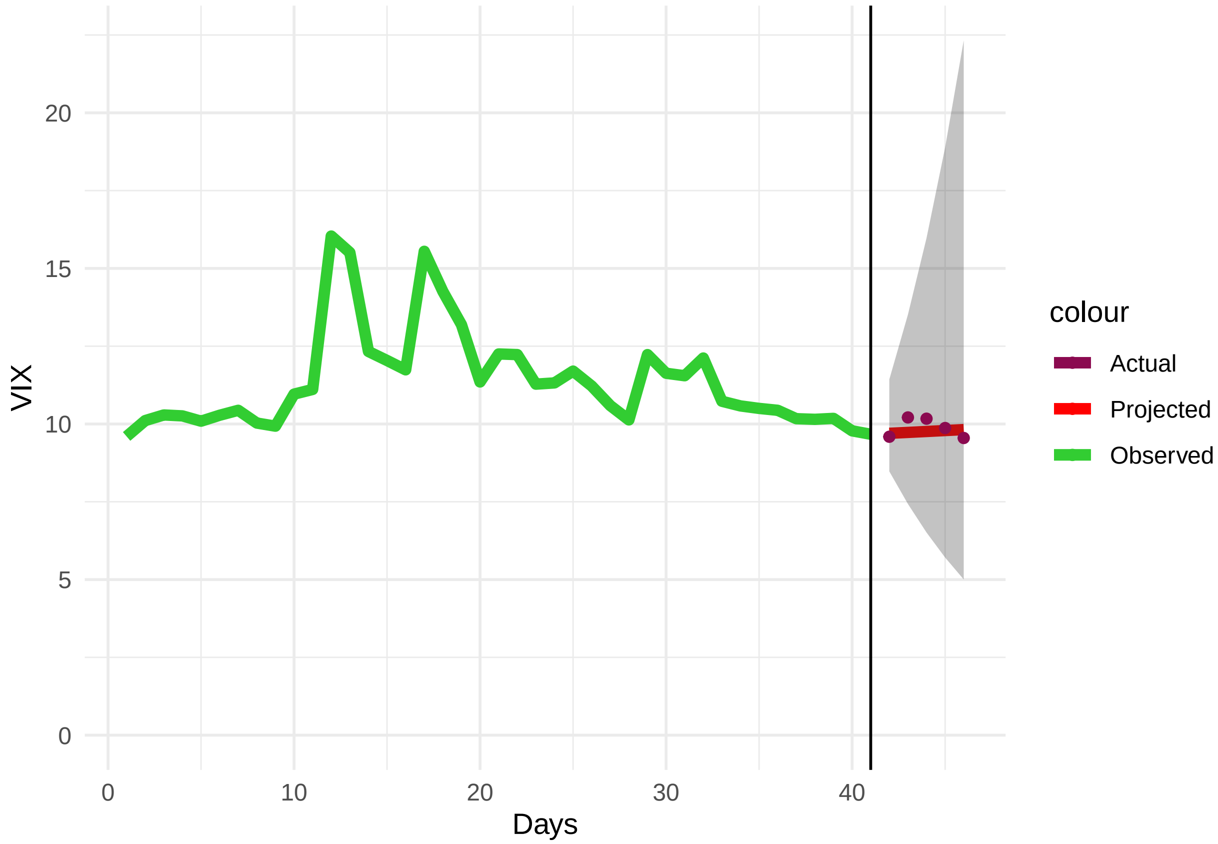
To test the performance of this approach we come back to the idea of dividing the dataset into train and test.

```
# Divide returns into train and test
spl_ind = max(which(as.yearmon(index(vix_r))<=date_split))
train = vix_r[2:spl_ind,] # Start with 2 to drop first NA
test = vix_r[(spl_ind+1):nrow(vix_r),]
test_vix = vix[(spl_ind+1):nrow(vix_r),]

# Calculate historic expectation and lower and upper percentiles
train_ret = as.vector(train$VIX.Adjusted)
exp_ret = mean(train_ret)
up_bound = quantile(train_ret, probs=c(0.025, (1-0.025)))[2]
low_bound = quantile(train_ret, probs=c(0.025, (1-0.025)))[1]


# Lets put together a versatile function.
ModelVix = function(observed, exp_ret,  up_bound,
                    low_bound, plot = TRUE, known = NA){
```

```
  observed = as.vector(observed[,1])

  # Create prediction for next 5 days
  last_pr = observed[length(observed)]
  e_p = last_pr * cumprod(1+rep(exp_ret,5)) # Using expectation
  u_p = last_pr * cumprod(1+rep(up_bound,5)) # Upper bound
  l_p = last_pr * cumprod(1+rep(low_bound,5)) # Lower bound


  # Create dataframe

  tmp = data.frame(Observed = c(observed,
                                rep(NA, 5)),
                   Trend = c(rep(NA,length(observed)), e_p),
                   Upper = c(rep(NA,length(observed)), u_p),
                   Lower = c(rep(NA,length(observed)), l_p)
  )




  # Add known if we have it
  if (!is.na(known)){
    tmp$Actual = c(rep(NA,length(observed)), as.vector(known[1:5,1]))
  } else {
    tmp$Actual = NA
  }

  # Plot
  if (plot==TRUE){
    g = ggplot(tmp, aes(x=1:nrow(tmp), y=Trend)) + geom_line(aes(y=Trend, color='limegreen'), size=2)  +
      geom_line(aes(y=Observed, color = 'red'), size=2) +
      geom_ribbon(aes(ymin=Lower, ymax=Upper), alpha=0.3) +
      geom_point(aes(y=Actual, color='deeppink4')) +
      ylim(0,max(tmp))  +
      xlim(1,nrow(tmp)) +
      scale_color_manual(values = c('deeppink4', 'red', 'limegreen'),
                         labels=c("Actual","Projected","Observed")) +
      theme_minimal() + xlab("Days") + ylab("VIX") +
      geom_vline(aes(xintercept=nrow(tmp)-5))
  }

  list(DF = tmp, Plot=g)
}
```

We can test the performance of this approach

```
set.seed(123)

# Random sampling data
day_T = sample(1:(nrow(test_vix)-5),1)

ModelVix(test_vix$VIX.Adjusted[(day_T-40):day_T], exp_ret,  up_bound,
         low_bound, known = test_vix$VIX.Adjusted[(day_T+1):(day_T+5)])$Plot
```



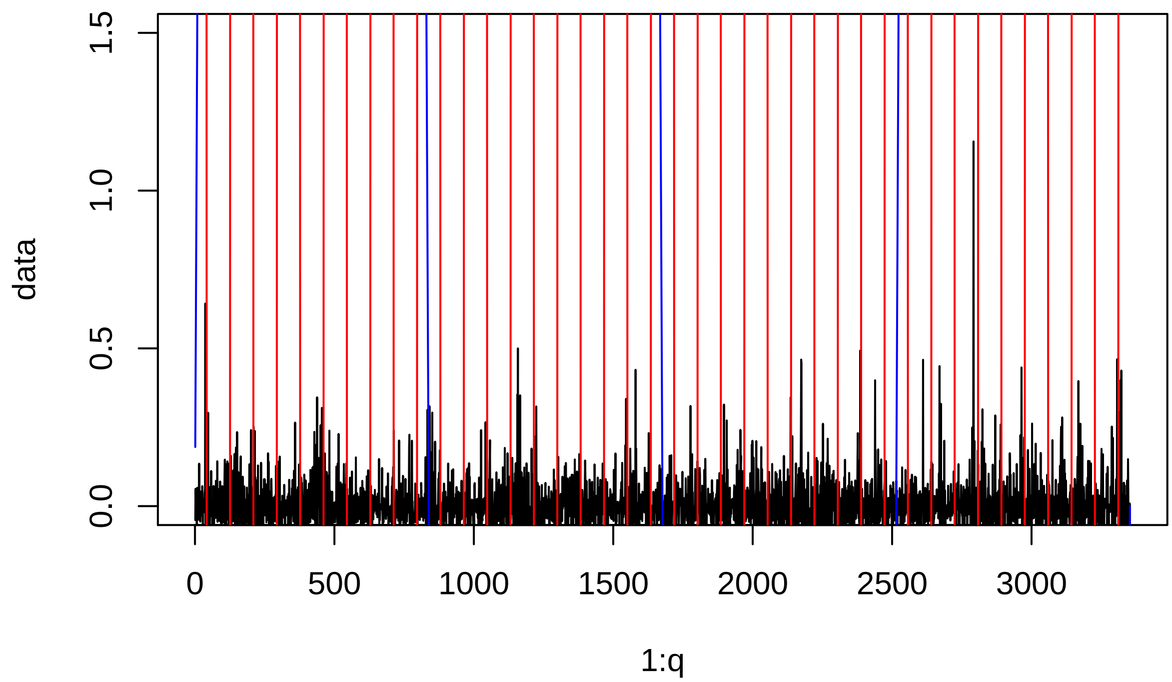
Several out-of-sample tests showed that this approach is fairly accurate. However, the downside is that the confidence interval is very large.

Fourier analysis of VIX Index

```
# Question 8: Using Fourier analysis.
################################################################################
#The goal of this analysis is to model the daily returns and observe the overall
#smoothness of the VIX's

# Apply Fourier analysis to VIX returns
data = as.numeric(ret)
q <- length(data);q
```

```
## [1] 3353
```

```
plot(1:q,data, ylim = c(0,1.5),type = "l")


myCos <- function(m) cos((1:q)*m*2*pi/q)
mySin <- function(m) sin((1:q)*m*2*pi/q)
plot(1:q,data, ylim = c(0,1.5),type = "l")
points(1:q, 100*myCos(1),type = "l", col = "blue")
points(1:q, 100*mySin(1),type = "l", col = "blue")
points(1:q, 100*myCos(20),type = "l", col = "red")
```

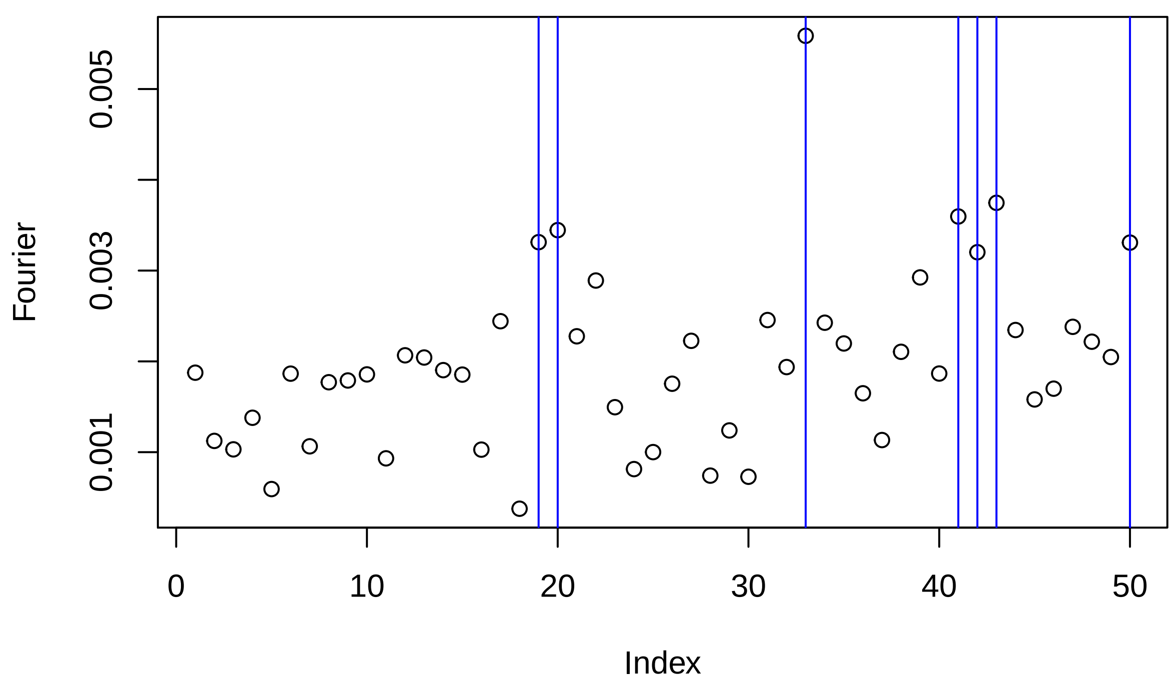



```r
#Currently looks ugly...

coeffA <- function(m){
  sum(data*myCos(m)/(q/2))
}
coeffB <- function(m){
  sum(data*mySin(m)/(q/2))
}

#We compute Fourier Coefficients
FourierA <- sapply(1:50,coeffA)
FourierB <- sapply(1:50,coeffB)
Fourier <- sqrt(FourierA^2+FourierB^2)
Fourier
```

```
##  [1] 0.0018755737 0.0011240312 0.0010309025 0.0013790947 0.0005934681
##  [6] 0.0018650025 0.0010645439 0.0017702656 0.0017896972 0.0018567425
## [11] 0.0009320736 0.0020668690 0.0020424256 0.0019036663 0.0018543041
## [16] 0.0010280596 0.0024424937 0.0003771998 0.0033131701 0.0034456356
## [21] 0.0022763996 0.0028905676 0.0014954700 0.0008136298 0.0010007845
## [26] 0.0017537514 0.0022265373 0.0007415096 0.0012396499 0.0007294235
## [31] 0.0024552474 0.0019374936 0.0055860782 0.0024258406 0.0021973315
## [36] 0.0016486768 0.0011333543 0.0021053572 0.0029250015 0.0018661212
## [41] 0.0035961125 0.0032023913 0.0037464502 0.0023455197 0.0015808936
## [46] 0.0016994446 0.0023806134 0.0022167259 0.0020479809 0.0033083579
```

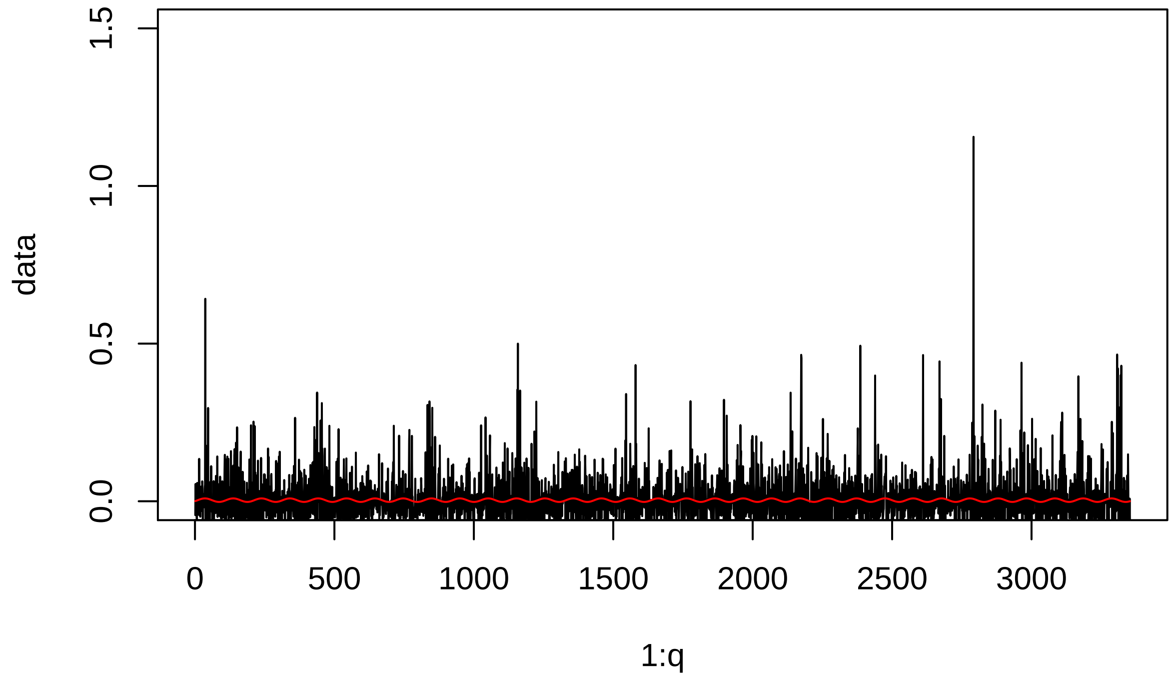
Let's quickly plot the coefficients and figure out which values are significant…

```r
plot(Fourier)
abline(v=33, col="blue")
abline(v=50, col="blue")
abline(v=43, col="blue")
abline(v=20, col="blue")
abline(v=19, col="blue")
abline(v=41, col="blue")
abline(v=42, col="blue")
```

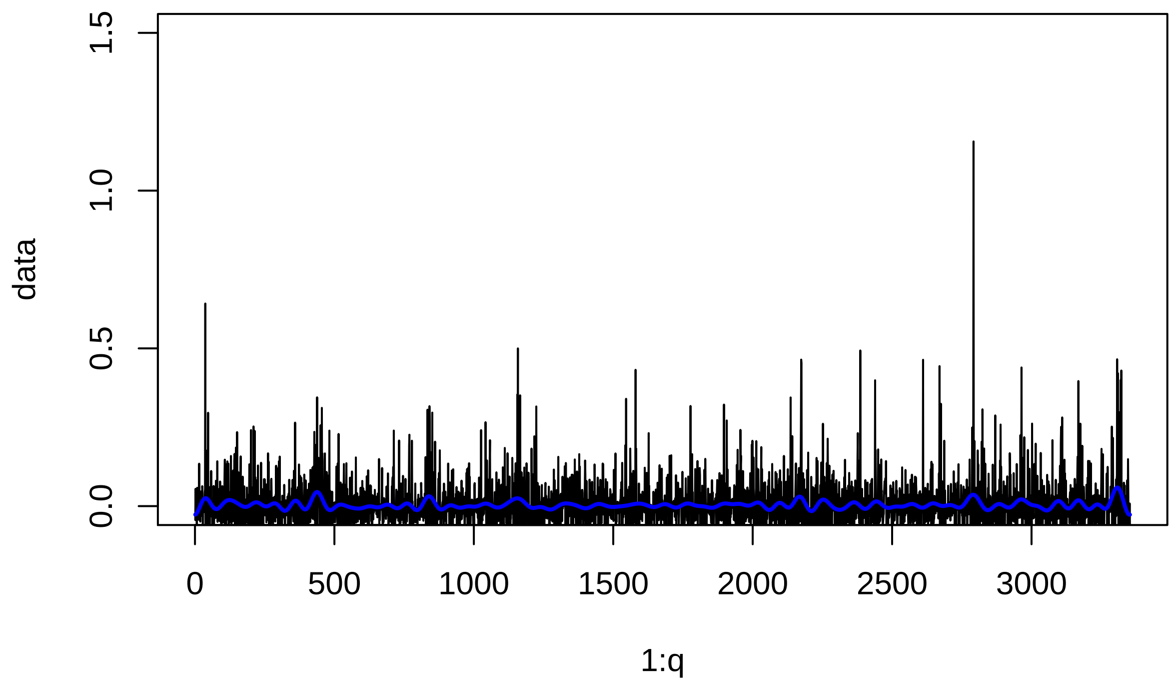


We See That A maximum is at 33, but there is no significant difference in order between values, alluding to the fact that the data is most likely not sinusoidal

```r
#Let's try reconstructing with 1 vector
plot(1:q,data, ylim = c(0,1.5),type = "l")
points(1:q,mean(data)+FourierA[33]*myCos(33)+FourierB[33]*mySin(33), type = "l", col = "red")
```




```r
#What if we use more?
recon <- mean(data)    #this is a_0
for (m in 1:50) {
  recon <- recon + FourierA[m]*myCos(m)+FourierB[m]*mySin(m)
}
plot(1:q,data, ylim = c(0,1.5),type = "l")
points(1:q,recon, type = "l", col = "blue",lwd = 2)
```

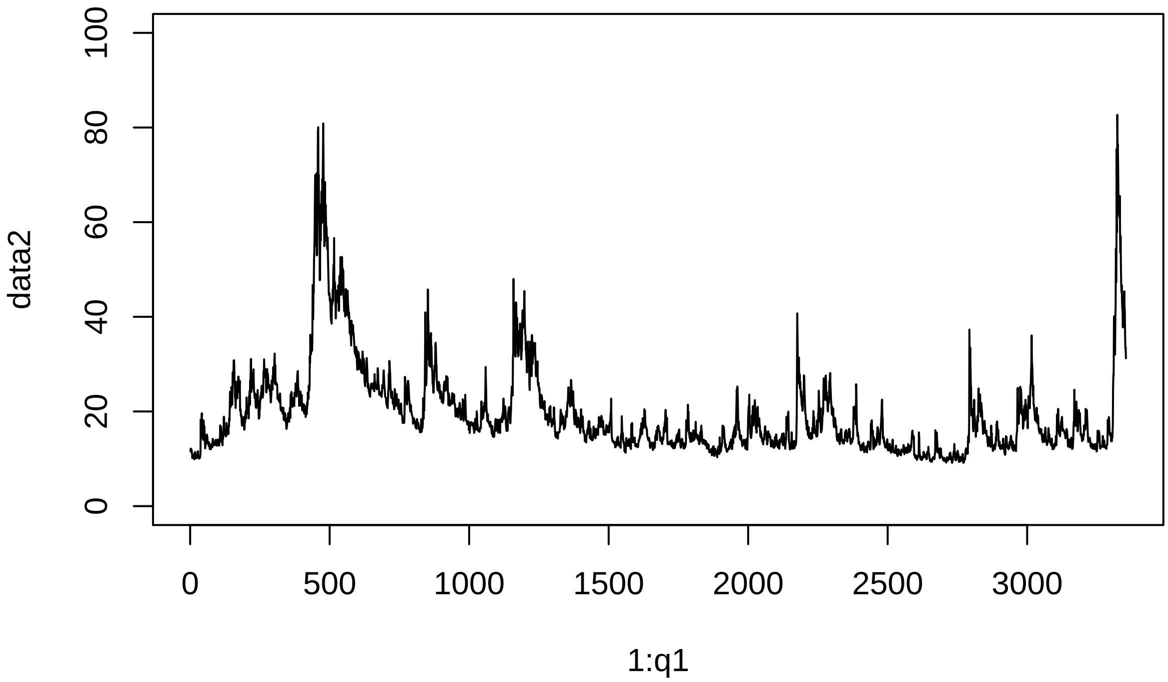


This doesn't look good, It can thus not be mapped periodically, What we conclude from this is that there is a significant amount of noise in daily VIX returns, if we look at the levels instead we can see a much better equation as $\sigma^2 = \frac{2}{T}\sum_i \frac{\Delta K_i}{K_i^2} e^{RT} Q\left(K_i\right) - \frac{1}{T}\left[\frac{F}{K_0} - 1\right]^2$ is the formula used to calculate the VIX given by CBOE.

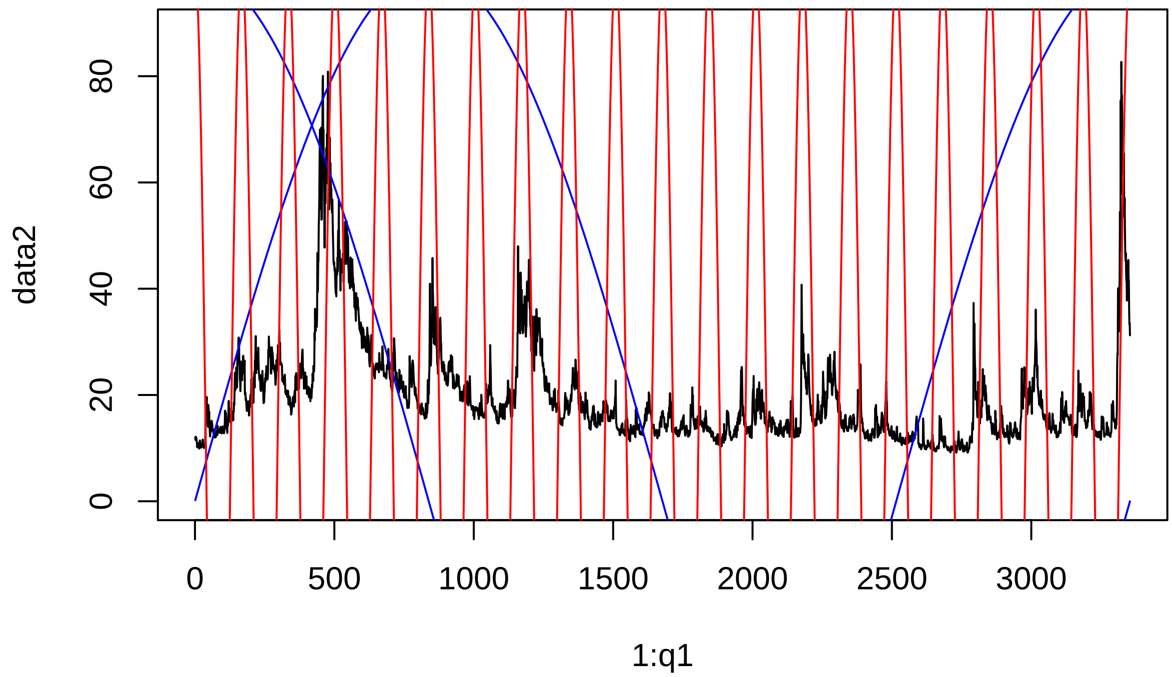
```
# What if we look at VIX Index levels instead of returns
#Let us take a look at the backwardation effect
data2 = as.numeric(vix$VIX.Close)
q1 <- length(data2);q1
```

```
## [1] 3354
```

```
plot(1:q1,data2, ylim = c(0,100),type = "l")
```




```r
myCos2 <- function(m) cos((1:q1)*m*2*pi/q1)
mySin2 <- function(m) sin((1:q1)*m*2*pi/q1)
plot(1:q1,data2, ylim = c(0,89),type = "l")
points(1:q1, 100*myCos2(1),type = "l", col = "blue")
points(1:q1, 100*mySin2(1),type = "l", col = "blue")
points(1:q1, 100*myCos2(20),type = "l", col = "red")
```



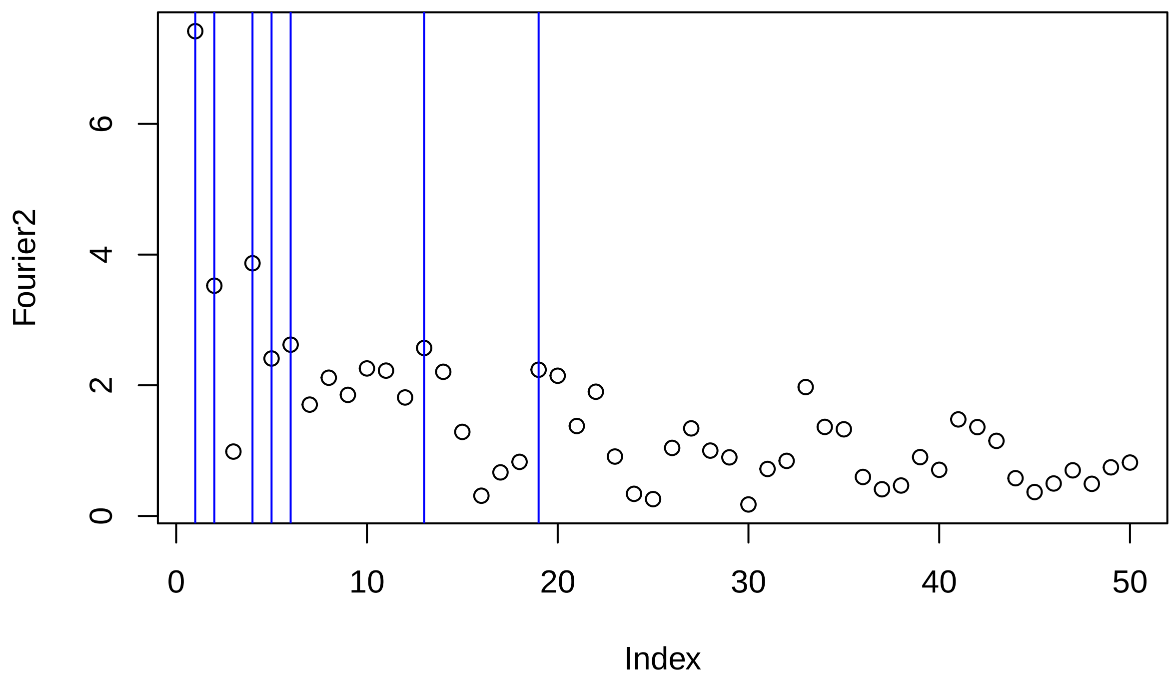

```
#Currently looks ugly...
coeffA2 <- function(m){
  sum(data2*myCos(m)/(q1/2))
}
coeffB2 <- function(m){
  sum(data2*mySin(m)/(q1/2))
}

#We compute Fourier Coefficients
FourierA2 <- sapply(1:50,coeffA2)
FourierB2 <- sapply(1:50,coeffB2)
Fourier2 <- sqrt(FourierA2^2+FourierB2^2)
Fourier2
```

```
##  [1] 7.4181455 3.5237381 0.9852924 3.8674833 2.4097682 2.6213732 1.7040391
##  [8] 2.1157917 1.8530415 2.2578573 2.2243373 1.8137134 2.5706269 2.2062514
## [15] 1.2867200 0.3101890 0.6673683 0.8282944 2.2380002 2.1460719 1.3769085
## [22] 1.9013058 0.9095309 0.3391314 0.2582038 1.0417045 1.3408234 1.0006370
## [29] 0.8971214 0.1766368 0.7198173 0.8431864 1.9728128 1.3631407 1.3264745
## [36] 0.5972836 0.4094253 0.4663819 0.9008071 0.7063728 1.4778414 1.3600586
## [43] 1.1495429 0.5782209 0.3665004 0.4973817 0.6989533 0.4920286 0.7447009
## [50] 0.8178063
```

Let's again quickly plot the coefficients and figure out which values are significant…

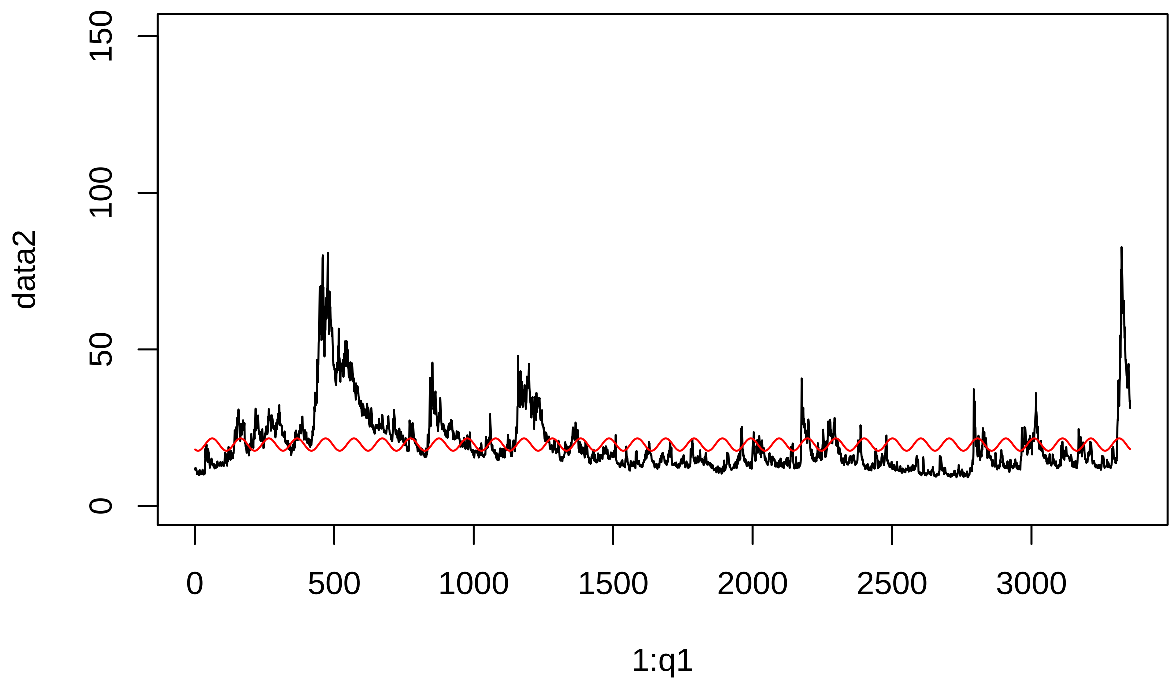
```
plot(Fourier2)
abline(v=1, col="blue")
abline(v=2, col="blue")
```

```r
abline(v=4, col="blue")
abline(v=6, col="blue")
abline(v=19, col="blue")
abline(v=13, col="blue")
abline(v=5, col="blue")
```

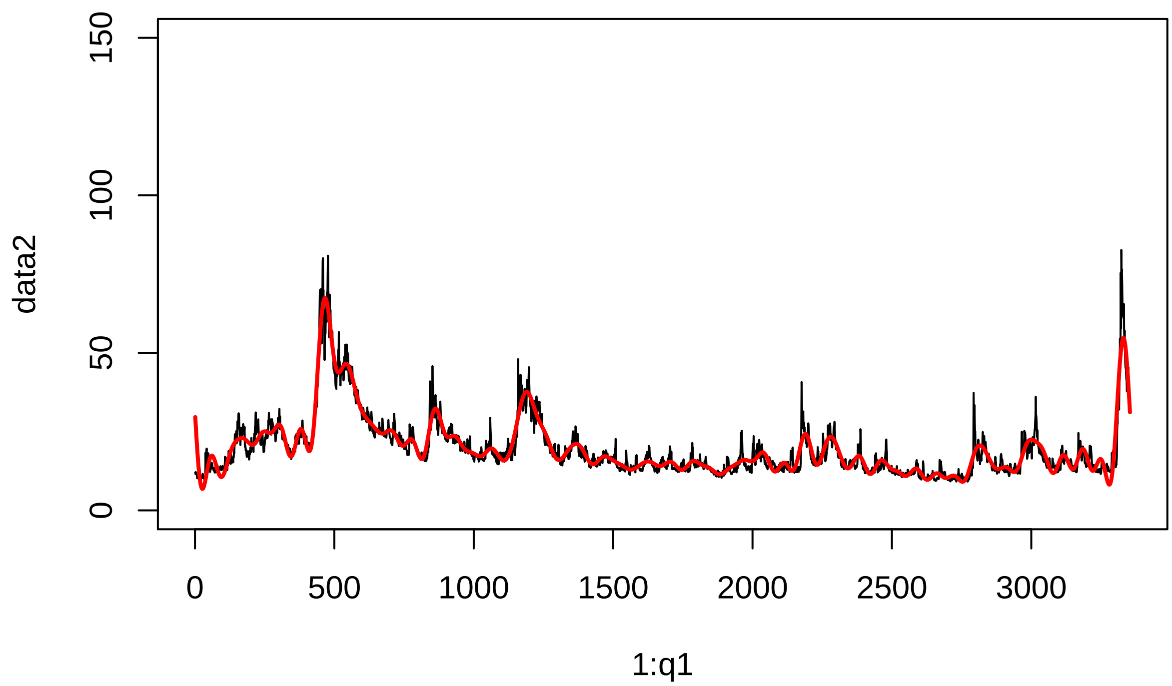


Let's try reconstructing with 1 vector

```r
plot(1:q1,data2, ylim = c(0,151),type = "l")
points(1:q1,mean(data2)+FourierA2[33]*myCos2(33)+FourierB2[33]*mySin2(33), type = "l", col = "red")
```




```r
#What if we use more?
recon2 <- mean(data2)   #this is a_0
for (m in 1:50) {
  recon2 <- recon2 + FourierA2[m]*myCos2(m)+FourierB2[m]*mySin2(m)
}
plot(1:q1,data2, ylim = c(0,150),type = "l")
points(1:q1,recon2, type = "l", col = "red",lwd = 2)
```




```
#This actually looks fairly accurate.
```

The conclusion that we can draw from this is that certain effects on the VIX such as expansion and weighting is able to be modeled quite well by a Fourier analysis, and as a result, we can conclude that it is relatively smooth. However, daily returns are in comparison very noisy, most as the volumes of the S&P contracts vary and volume is not consistent. As a result, it is quite impossible to smoothen the data without having the volume inflow and sticking to a smaller granularity.