

Time Series Analysis of Monthly VIX Index

University of Toronto Mississauga, STA457 Project

Professor: Masoud Ataei

Written By: David Quan

Date: April 8, 2022

Abstract

The stock market is watched by all types of individuals, ranging from CEOs, brokers, investors, and many more. There are many external factors that can influence the prices of the market. A popular tool to gauge the possible direction the market will head in, is called the Volatility Index (VIX). The VIX represents the amount of fear individuals have towards the stock market. This report is a time series analysis of the monthly VIX prices. Our goal is to provide future forecasts of the future monthly VIX average prices.

Throughout the process of utilizing the Box-Jenkins Framework, issues with normality of the error terms were encountered, which was the main roadblock. It is important to note that the monthly VIX prices is largely dependent on external factors rather than the past prices. As a result, we will consider segmentations of the data (particularly ranging from January 2010 to December 2019), rather than considering the data as a whole.

We will begin by explaining some important concepts, and addressing questions about the nature of the VIX. Then, using the Box-Jenkins Framework, we will develop several candidate models to analyze and test. The main model that was selected and found to be the best was ARMA(2,0,2), as it performed relatively the same compared to others, however it was the most parsimonious. The conclusions we reached is that the monthly VIX is difficult to predict due to the many external factors that can influence the sentiment towards the stock market.

Introduction

In the United States of America, an immense amount of money is transferred within the stock market everyday. Many companies offer stocks for the public to purchase, in which the individual will in turn own a share of the company. The success of companies plays a vital part in the economy, and often reflect in their stock prices. The trading of stocks has been both a form of investments for individuals and a profession for many. As a result, it is important for all types of traders to obtain as much information before making financial investments and decisions.

The Standard and Poor's 500, commonly known as the S&P500 looks at the collective performance of 500 of the United State's leading companies such as Microsoft, Walmart, and Disney, just to name

a few. These 500 companies provide an accurate representation of the US stock market as a whole.

The Chicago Board Options Exchange (CBOE) created the Volatility Index (VIX) in the year of 1993. The VIX provides an estimate of the overall stock market's implied volatility, namely the future expectations of the market's uncertainty. The prices of the VIX are largely dependent on the attitude of the investors towards the stock market. This is known as the market sentiment and can either be bullish or bearish. Bullish refers to the attitude of investors believing the overall market will excel, whereas bearish refers to a negative attitude, believing it will not perform well.

The dataset provided contains the monthly average of the VIX Index prices. In the first column, we are given the dates (the year along with the month), ranging from February 1990 to December 2021. For each month, an average of the daily VIX Index is taken and that is what occupies the second column of our dataset. It is important to note that the data given is the monthly average of the VIX, rather than the daily prices. This gives us more room for error, because if we were forecasting a single day ahead, there is no room for error. But since we are forecasting an average, it gives us more leniency.

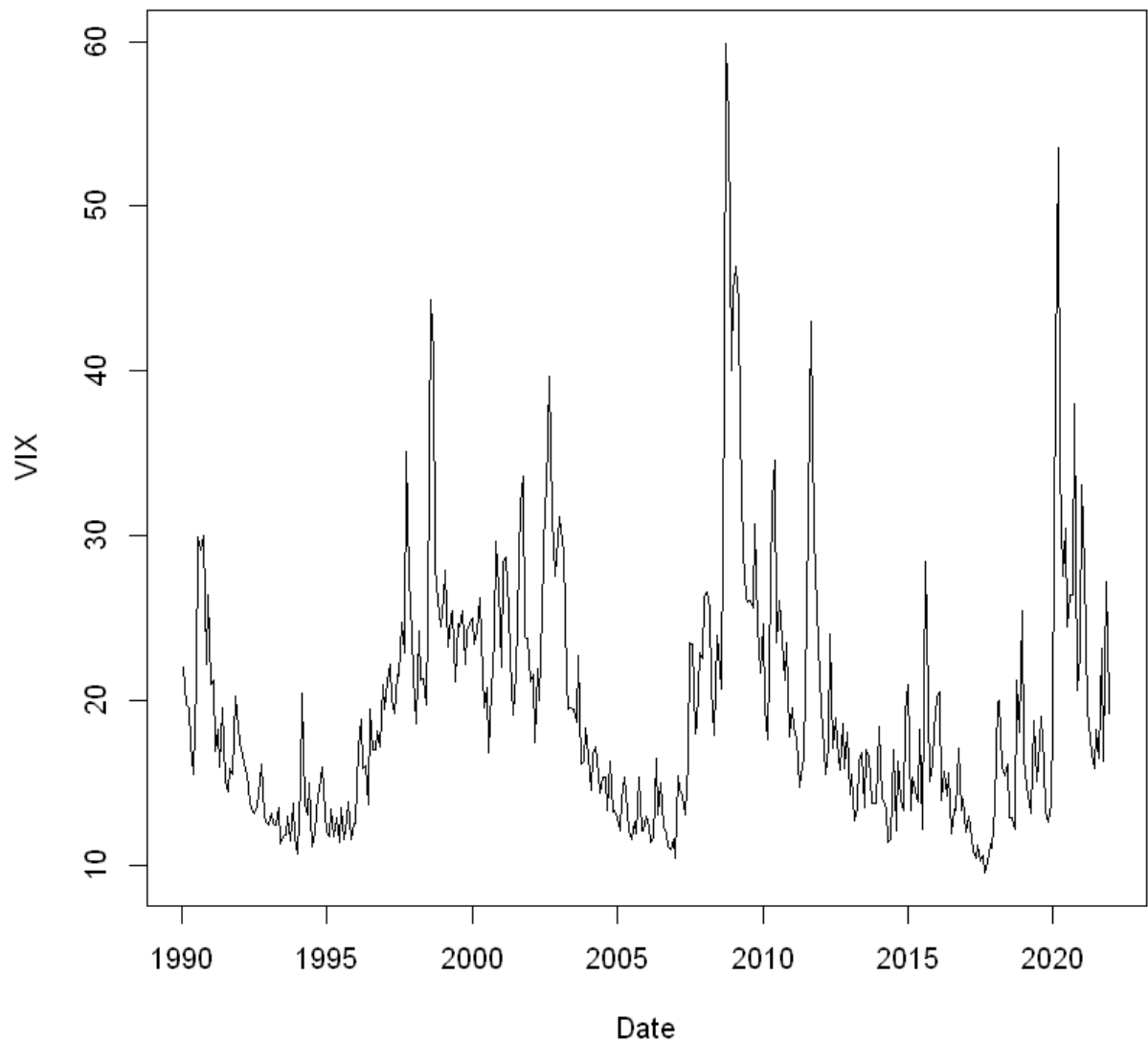
This investigation on the monthly VIX is worthy of analysis because determining an accurate model can help investors know when to buy and sell their stocks. Personally speaking, I just started investing in some technology stocks, however without any knowledge of the stock market I faced many challenges. In particular, when I would check company updates and negative news would come out, this did instill fear a sense of fear in me. In particular, I would keep an eye on Reddit's group, called Wallstreet Bets, and gather a lot of information on individuals' sentiment to stocks like Gamestop (GME). Though it should be mentioned that the individuals on the Reddit page are always in favour for such stocks, so it misrepresents the sentiment of all people, which the VIX is an indicator of. That is to say, if it is possible to detect possible patterns or even know the general direction of the market, this can be a strong asset to investors.

Below is a time plot of our time series data, to get a better idea of how the VIX may work.

```
In [6]: # Time Plot of Whole data
data <- read_excel("./Project_Data.xlsx")

plot(data$Date, data$VIX, type = "l", xlab = "Date", ylab = "VIX", main = "Time Plot of
```

Time Plot of Monthly VIX

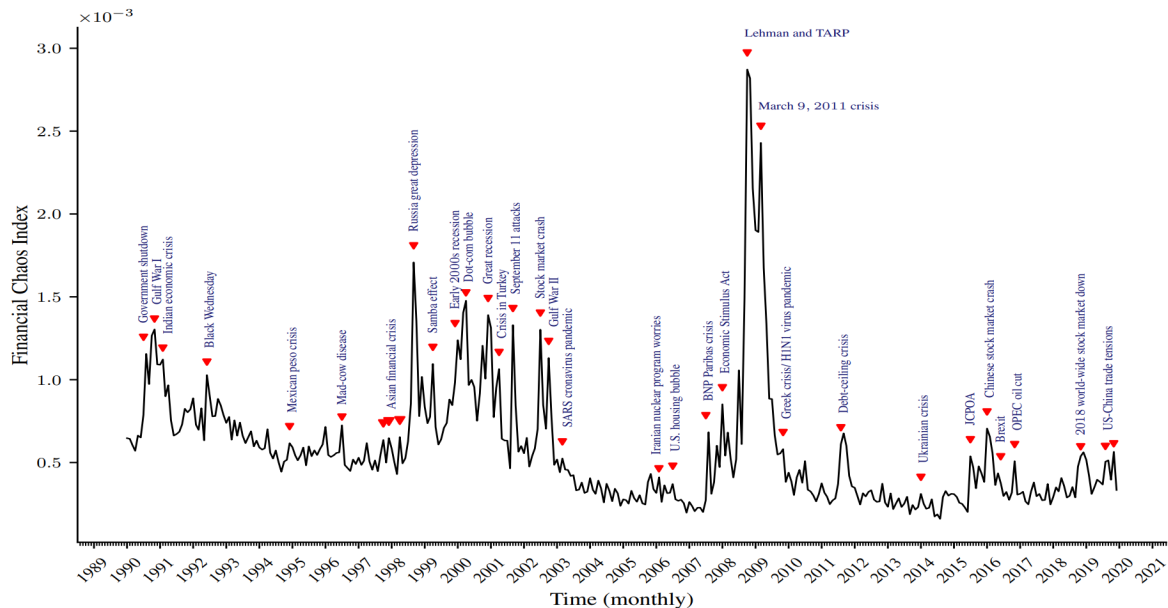


Looking at the time plot of the whole data, there are some noticeable peaks (particularly five of them) in the VIX prices. Below are the following time periods, along with the possible significant global event that occurred during that particular time, and may largely be the reason of the spike in VIX. Note that these events were obtained from wikipedia's word events page.

They occur in the following time periods:

Date	Event
1998, September	1998 Russian Financial Crisis
2002, September	After Effects of September 11 Attacks
2008, October	Financial Crisis of 2007-2008
2011, September	August 2011 Stock Markets Fall
2020, March	Covid-19 Pandemic

Now considering the graphs that Professor Ataei prepared, we notice that the peaks aren't exactly the same (for example in september of 2011, the VIX price was far higher than the graph below suggests). Despite the slight discrepancy, the general shape of the time plot and the graph provided is the same. To ensure there is no confusion, the graph below is cut off towards the end, however it should resemble a spike during March 2020 due to COVID-19.



We will now address the question: *Is it possible to forecast the monthly VIX index?*

First off, considering the peaks of the volatility index, it is not clear what category (if any) the world events occurring at these points in time fall under. We have crises relating to financials/the economy, unexpected attacks (9-11), and a pandemic. This suggests that there are several factors at play that influences the sentiment of the masses. It appears that a major crisis can trigger a quick change in the VIX, however the question at hand becomes: are such major crises are predictable? Considering events such as recessions, these are typically cyclical, however the period doesn't always hold true. For events such as 9-11, and COVID-19, these were things that the world did not see coming. By this logic, it appears that major crises are not predictable, and so the VIX is not predictable either due to the sheer number of factors impacting it.

A question that may arise for some individuals is: *Given the past data, is it sufficient to construct a predictive model for forecasting the VIX trend?* Further into the report, we arrive to the conclusion that the past data is useful in certain sections of the data. There comes a point in which the data doesn't impact what will happen in the present, and this is based on what segments of the data we are looking at. We will show that it is possible to create a predictive model to capture the general trend, but it's more difficult to report exact values.

A question to consider: *Are there any factors influencing the value of the VIX index at every month? If so, is it possible to omit these additional elements from our analysis and simplify the problem to studying only the univariate time series of the VIX observations? What would the main drawbacks of such simplification?* No it is not possible to attribute the monthly VIX index to one factor. Such

simplification would result in ignoring many other factors that still influence the data, but perhaps to a lesser degree. As mentioned before, there are many factors that have an influence in raising and dropping the VIX index, and to eliminate factors would result in not properly representing the nature of the stock market.

Model Specification

To begin, it is important to note that the nature of the VIX stock prices is time dependent (it is a stochastic process). This will affect how we split our data into a training and testing set. We will have to utilize the first time point and it's subsequent observations (up to a certain point) as the testing dataset, rather than using random assignment.

Let's now split our data into a training dataset and a testing dataset. We have 384 observations, which is a moderate amount of data considering this is time series data. Since time series data heavily relies on the training dataset to provide an accurate forecast on the testing dataset, we will apply a 95/5 split. So the first 95% of observations will be our training dataset (the first 365 observations), and the remaining 5% (observations 366-384) will make up our testing dataset.

```
In [7]: # Computing the size of our training and testing datasets
x <- data$VIX
n_total <- length(x)
pTest = 0.05

nTrain <- n_total - floor(pTest * n_total)
nTest <- n_total - (n_total - floor(pTest*n_total) + 1) + 1

print("Size of Training Dataset:")
nTrain

print("Size of Testing Dataset:")
nTest

# Split up dataset into a training and testing dataset
# Training Dataset
training_dataset <- data[1:365,]
training_dates <- training_dataset$Date
training_VIX_Values <- training_dataset$VIX

# Testing Dataset
testing_dataset <- data[366:384,]
testing_dates <- testing_dataset$Date
testing_VIX_Values <- testing_dataset$VIX
```

```
[1] "Size of Training Dataset:"
365
[1] "Size of Testing Dataset:"
19
```

Prior to the model building, the first step is to analyze our data (the training dataset) to obtain a better idea of what we are working with. To do this, we will consider the time plot of the Monthly VIX, and since our sample size is moderate in size, we will also analyze the plot of the cumulative mean level. Upon first glance of the time plot, there is no such horizontal line that the data will be relatively centered around. This implies that our data seems to be non-stationary in nature. Considering the different types of non-stationary models, we can rule out the following:

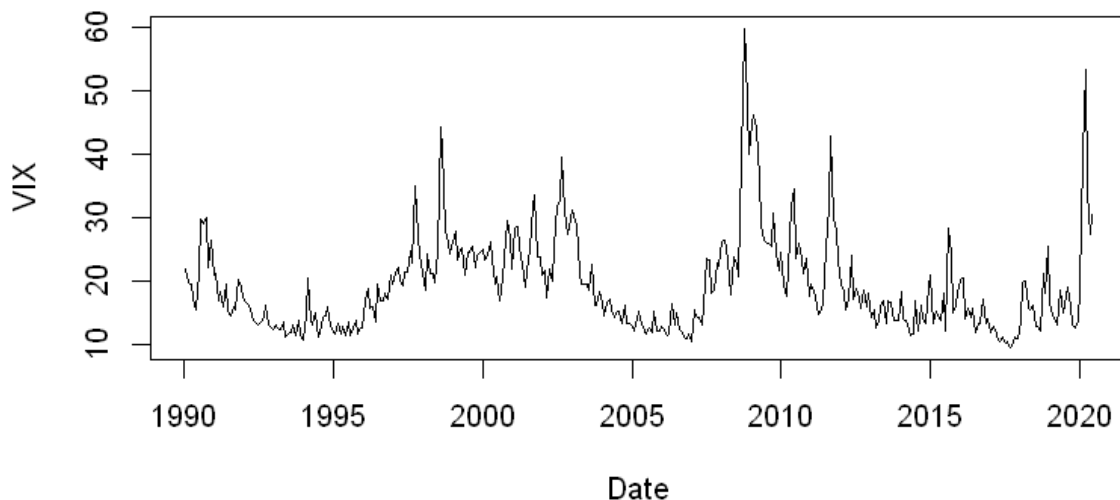
trend stationary, as the mean is not exactly increasing; and heteroscedacity since we don't have a constant mean. Now this leaves the level stationary, and difference stationary to keep in mind as we proceed. The plot of the cumulative Mean shows that towards the end of the time series, it sort of tapers out to a mean level of about 19.5 from 2002 onwards. It can be observed that there is a moderate amount of fluctuation in the mean level. This suggests our data is non-stationary.

```
In [8]: par(mfrow = c(2,1), mar = c(4, 4, 4, 4))

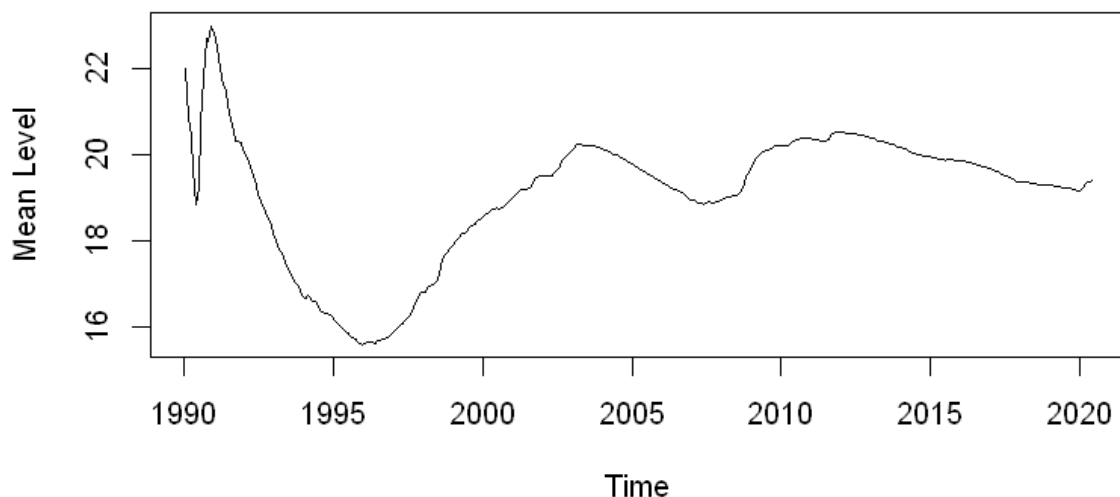
# Time Plot
plot(training_dates, training_VIX_Values, type = "l", xlab = "Date", ylab = "VIX", main = "Time Plot of Monthly VIX")

# Cumulative mean
cummeanVIX <- cumsum(training_VIX_Values) / seq_along(training_VIX_Values)
plot(training_dates, cummeanVIX, type = "l", xlab="Time", ylab = "Mean Level", main = "Plot of the Cumulative Mean")
```

Time Plot of Monthly VIX



Plot of the Cumulative Mean



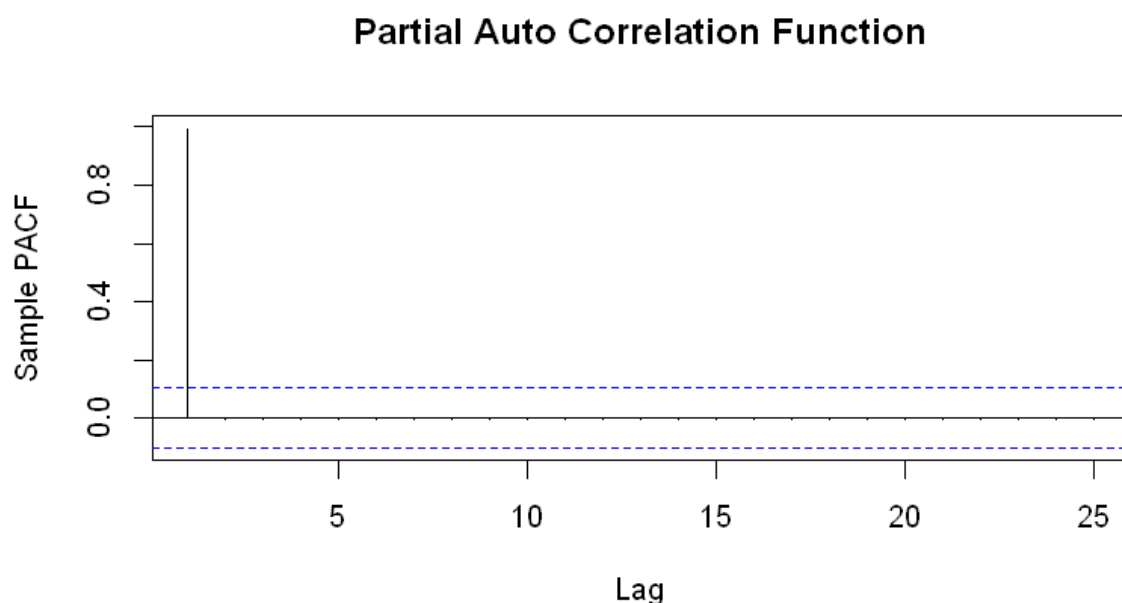
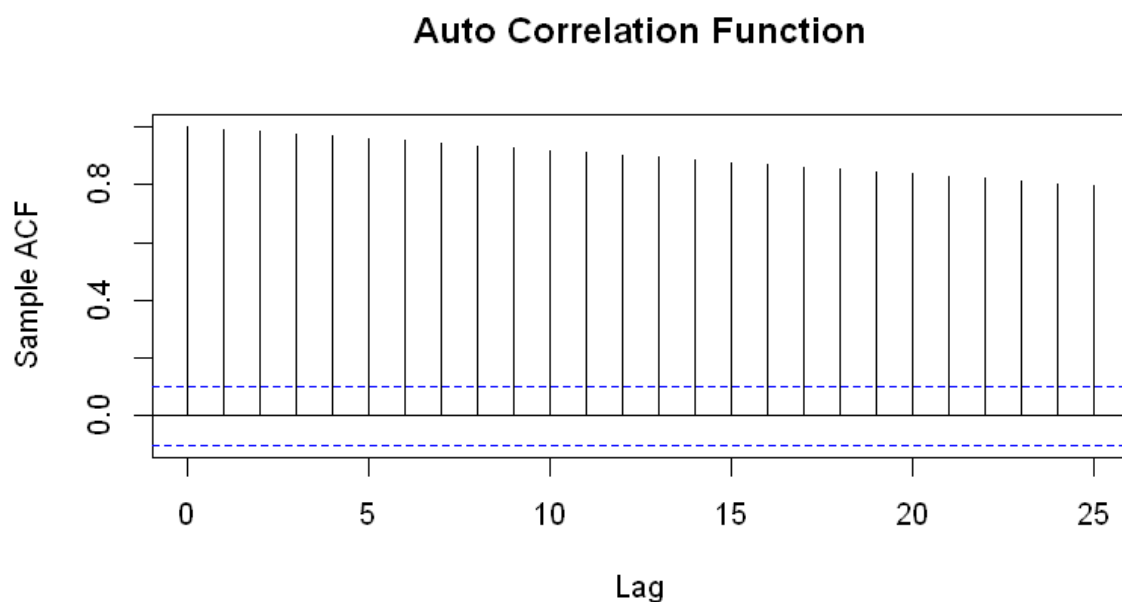
To obtain a general idea of whether the time series will have an Autoregressive (AR) or a Moving Average (MA) signature, one can analyze the Sample Autocorrelation Function (ACF) and the Partial Autocorrelation Function Plots (PACF).

In the ACF plot, notice that at each lag given, the bars exceed the blue threshold bands. So this tells us there is structure in our time series and that we have evidence against pure randomness. Now this plot indicates that the signature is largely Autoregressive because the Sample ACF doesn't reach zero at some lag point.

Now in the Partial ACF plot, our above conclusion is solidified because at lag 1 and onwards we observe that the sample PACF is zero. Therefore, we can conclude that our model's AR part is more powerful than the MA part.

In [9]:

```
#ACF, PACF
par(mfrow = c(2,1), mar = c(4, 4, 4, 4))
acf(training_dates, xlab = "Lag", ylab = "Sample ACF", main = "Auto Correlation Function")
acf(training_dates, type = "partial", xlab = "Lag", ylab = "Sample PACF", main = "Partial Auto Correlation Function")
```



In order to further analyze the stationarity of our time series data, we will apply the Augmented Dickey-Fuller (ADF) Test and the Kwiatkowski–Phillips–Schmidt–Shin (KPSS) Test.

The ADF Test considers the null hypothesis $H_0 : X_t \sim I(1)$, namely there is a unit root versus the alternative hypothesis $H_1 : X_t \sim I(0)$, namely the time series is stationary, under the assumption that the data has an ARMA structure. Upon performing the test, we obtain a p-value of 0.01, which is less than alpha at 0.05. Therefore we conclude that there is evidence against the null hypothesis, and so we reject it. Now this implies that our data is stationary.

The KPSS Test tests the null hypothesis, H_0 : the time series data is stationary versus the alternative hypothesis H_1 : the time series data is non-stationary because of a unit root. Now for the KPSS Test for level stationarity, we obtain a p-value of 0.01, which is less than alpha at 0.05. Therefore, we reject the null hypothesis, and conclude that the time series is non-stationary. However, for the KPSS Test for trend stationarity, we obtain a p-value of 0.1, which is greater than alpha at 0.05. Therefore, we fail to reject the null hypothesis, and conclude that there is not enough evidence that the model isn't trend stationary.

Recalling back to our analysis of the time plot, we ruled out the trend stationary model because there was no clear way to fit a line representing the mean around the data given. The results of the KPSS test contradicts our visual assumption, and so we can't firmly say if the model is non-stationary or not.

```
In [10]: # Augmented Dickey-Fuller (ADF) Test
training_dates.adf <- adf.test(training_dates, alternative = "stationary")
training_dates.adf

# Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test
training_dates.kpss_level <- kpss.test(training_dates, null = "Level")
training_dates.kpss_level
training_dates.kpss_trend <- kpss.test(training_dates, null = "Trend")
training_dates.kpss_trend
```

```
Warning message in adf.test(training_dates, alternative = "stationary"):
"p-value smaller than printed p-value"
Augmented Dickey-Fuller Test
```

```
data: training_dates
Dickey-Fuller = -8.0294, Lag order = 7, p-value = 0.01
alternative hypothesis: stationary
```

```
Warning message in kpss.test(training_dates, null = "Level"):
"p-value smaller than printed p-value"
KPSS Test for Level Stationarity
```

```
data: training_dates
KPSS Level = 6.1822, Truncation lag parameter = 5, p-value = 0.01
Warning message in kpss.test(training_dates, null = "Trend"):
"p-value greater than printed p-value"
KPSS Test for Trend Stationarity
```

```
data: training_dates
KPSS Trend = 0.019934, Truncation lag parameter = 5, p-value = 0.1
```

Since we have mixed reviews on the stationarity of our time series, we know that typically time series are non-stationary. Following the Box Jenkins Analysis, this means we must apply some transformation(s). Below we will perform differencing, which will make our model stationary.

In [11]:

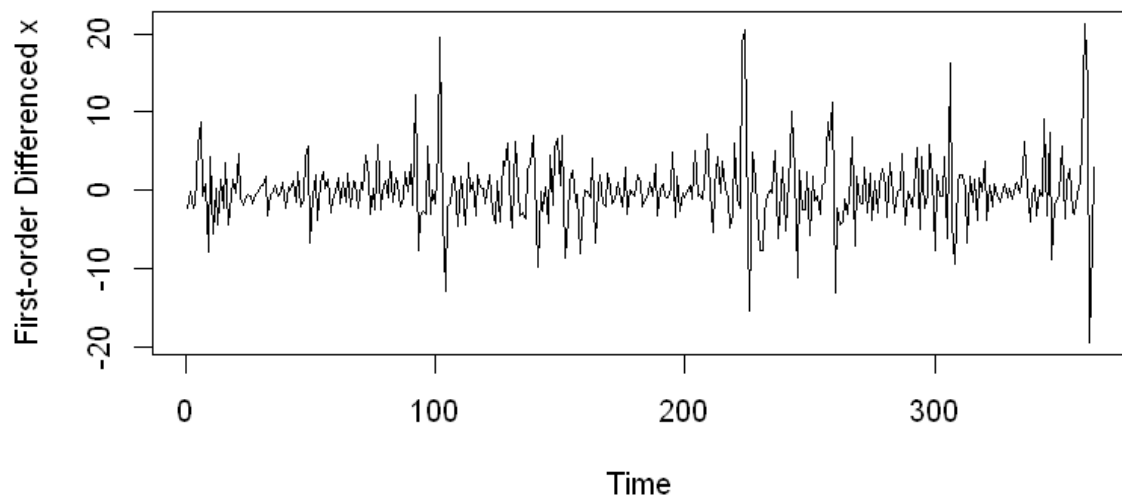
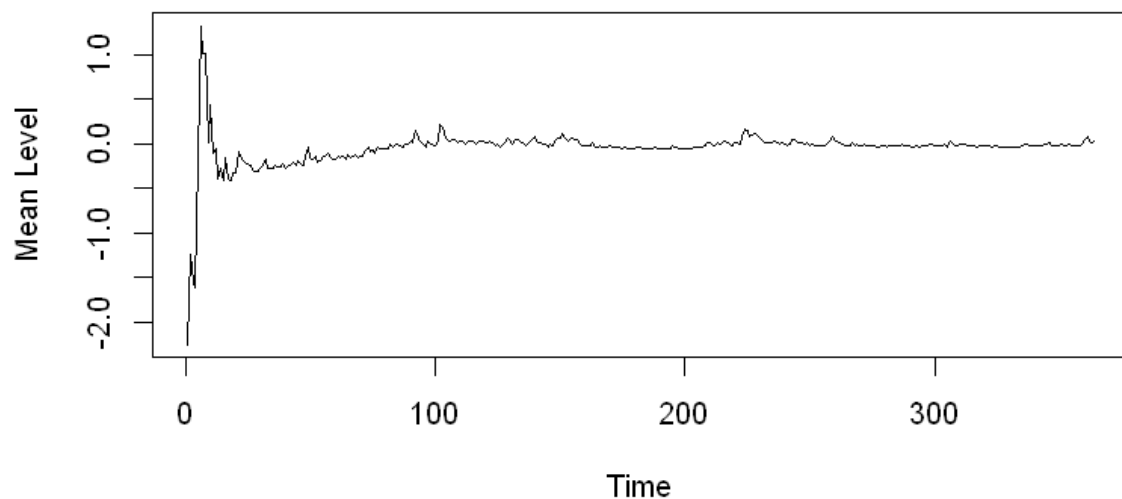
```
# Differenced Series
y = diff(training_VIX_Values)

cummean_y <- cumsum(y) / seq_along(y)

par(mfrow = c(2,1), mar = c(4, 4, 4, 4))

# Time Plot
plot(y, type = "l", xlab = "Time", ylab = "First-order Differenced x", main = "Time Plot")

# Plot of Cumulative Mean
plot(cummean_y, type = "l", xlab = "Time", ylab = "Mean Level", main = "Plot of Cumulative Mean")
```

Time Plot (Differenced)**Plot of Cumulative Mean (Differenced)**

The above time plot does indicate stationarity, as we can fit a horizontal line at 0 to accurately balance the data around. Also the plot of the cumulative mean indicates stationarity since we don't have any major fluctuations towards the end of the plot.

The ADF Test yields a p-value of 0.01, which is less than alpha at 0.05. Therefore, we reject the ADF test's null hypothesis, and thus this implies that our the differenced series is stationary.

The KPSS Test yields a p-value of 0.1 for both cases. We fail to reject each of our null hypotheses because the p-value is greater than alpha at 0.05. Therefore for the differenced series, there is no evidence to conclude that the series is non-stationary because of a unit root.

In [12]:

```
# Augmented Dickey-Fuller (ADF) Test for Differenced Series
y.adf <- adf.test(y, alternative = "stationary")
y.adf

# Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test for Differenced Series
y.kpss_level <- kpss.test(y, null = "Level")
y.kpss_level
y.kpss_trend <- kpss.test(y, null = "Trend")
y.kpss_trend
```

```
Warning message in adf.test(y, alternative = "stationary"):
"p-value smaller than printed p-value"
      Augmented Dickey-Fuller Test
```

```
data: y
Dickey-Fuller = -8.8805, Lag order = 7, p-value = 0.01
alternative hypothesis: stationary
Warning message in kpss.test(y, null = "Level"):
"p-value greater than printed p-value"
      KPSS Test for Level Stationarity
```

```
data: y
KPSS Level = 0.028478, Truncation lag parameter = 5, p-value = 0.1
Warning message in kpss.test(y, null = "Trend"):
"p-value greater than printed p-value"
      KPSS Test for Trend Stationarity
```

```
data: y
KPSS Trend = 0.021838, Truncation lag parameter = 5, p-value = 0.1
```

For our differenced series, we have that the ADF test suggests the differenced series is stationary. As for the KPSS tests, they both yields a p-value of 0.1. Thus, we fail to reject each of our null hypotheses because the p-value is greater than alpha at 0.05. Therefore for the differenced series, there is no evidence to conclude that the series is non-stationary because of a unit root. Therefore our differenced series is stationary.

Now we will proceed into looking at the Extended ACF Table which will help indicate what models are valid to consider and not.

In [13]:

```
# Extended ACF Table
y.eacf <- eacf(y, ar.max = 5, ma.max = 5)

#NOTE: Due to some issues with Jupyter Notebook and the version of R,
#       I wasn't able to obtain the EACF Table. Instead I used my
#       R-Studio instead, and obtained the EACF Table.
```

Error in eacf(y, ar.max = 5, ma.max = 5): could not find function "eacf"
Traceback:

```
> print("By: David Quan")
[1] "By: David Quan"
> y.eacf <- eacf(y, ar.max = 5, ma.max = 5)
AR/MA
  0 1 2 3 4 5
0 o x x o o o
1 x x o o o o
2 x x o o o o
3 x x o o o o
4 x o x x o o
5 x x x o x o
```

Given the Extended ACF Table, we won't accept any of the models (based on the orders) that have an 'X' and consider the ones with an 'O'.

In [14]:

```
y.aic <- matrix(0, 5, 5)
y.bic <- matrix(0, 5, 5)

for (i in 0:4) for (j in 0:4){
  y.fit <- arima(y, order = c(i,0,j), method = "ML", include.mean = TRUE)
  y.aic[i + 1, j + 1] <- y.fit$aic
  y.bic[i + 1, j + 1] <- BIC(y.fit)
}

y.aic_vec <- sort(unmatrix(y.aic, byrow = FALSE))[1:13]
y.bic_vec <- sort(unmatrix(y.bic, byrow = FALSE))[1:13]

#NOTE: Due to some issues with Jupyter Notebook and the version of R,
#       the AIC and BIC vectors were done in my R-studio
```

Error in unmatrix(y.aic, byrow = FALSE): could not find function "unmatrix"
Traceback:

1. sort(unmatrix(y.aic, byrow = FALSE))

```
> print("By: David Quan")
[1] "By: David Quan"
> y.aic_vec <- sort(unmatrix(y.aic, byrow = FALSE))[1:13]
> y.bic_vec <- sort(unmatrix(y.bic, byrow = FALSE))[1:13]
>
> y.aic_vec
r3:c2 r2:c3 r1:c4 r2:c4 r1:c5 r4:c2 r3:c3 r5:c4 r2:c5 r4:c4 r5:c2 r3:c4 r4:c3
2092.957 2093.090 2094.761 2094.809 2094.824 2094.880 2094.898 2096.444 2096.591 2096.685 2096.744 2096.805 2096.936
> y.bic_vec
r3:c2 r2:c3 r2:c2 r1:c4 r2:c4 r1:c5 r4:c2 r3:c3 r1:c3 r4:c1 r2:c5 r5:c2 r3:c4
2114.442 2114.575 2115.940 2116.246 2120.192 2120.207 2120.262 2120.280 2121.039 2125.848 2125.872 2126.024 2126.085
```

Given all these models with various orders p and q, there are several things to keep in mind as we choose our candidate models. Firstly, we want the models that prioritize the AR part over the MA part because as we observed in the ACF and Partial ACF charts, our model's AR part is more powerful than the MA part. Secondly, the Principle of Parsimony states that we favour models that are simple (or parsimonious) over complex models. Lastly, the concept of AIC and BIC is important to comprehend. Essentially, AIC overestimates the number of parameters in the model where as BIC underestimates.

Now looking at the AIC and BIC values for our possible models, they do not vary by much (no real significant change). So there's isn't one to really consider, and thus we will use the principle of

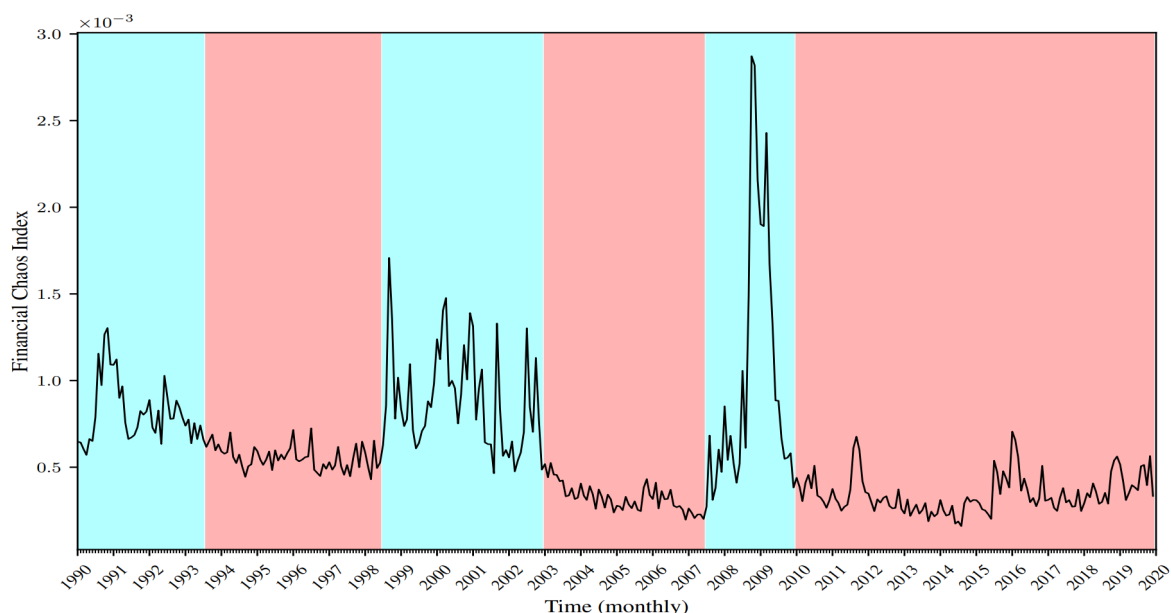
parsimony, as mentioned before, to select the best models. The results show that both the AIC and BIC methods select the model with ARMA(3,2) as the first best, and ARMA(2,3) as the second best. These have lower orders ($p+q$) compared to other models which is good. Lastly since we know the AR part should be prioritized, we expect the ARMA(3,2) model to be superior to the ARMA(2,3) model. Even though, the ARMA(2,2) yields a relatively low BIC value and is quite parsimonious, isn't anywhere to be seen in our AIC vector, thus we won't consider it. Also the ARMA(1,4) model isn't considered as it has a high order for the MA part, which shouldn't be prioritized. Lastly, note that these are valid models to consider based on the Extended ACF Table.

We will proceed with the two models: ARMA(3,2) and ARMA(2,3).

Model Specification: Regimes

Thus far, we encountered some issues with the fitting and diagnostics stage. The ARMA(3,2) and ARMA(2,3) models do indeed have uncorrelated error terms which is desired, however the QQ plot and the Shapiro Test suggests that the error terms are not normal. The Box-Jenkins framework requires us to have normality of the error terms to proceed to the model forecasting stage.

In the introduction, we explored the events that stuck out in spiking the Volatility Index. As suggested by Professor Ataei, one approach we can take is to split the time series data into segments, also known as regimes. The graph below shows the Volatility Index regimes (with COVID-19 creating a 7th regime). Here the blue and red colours simply are just simply used to distinguish between regimes, and there is no special relation between all the blue (or red) segments. In each isolated regime, there are difference forces driving the Volatility Index, which including (but not limited to): Psychological, economical, political, geopolitical, fundamentals, traders, and uncertainty. Since the factors vary in each time period, we cannot consider the time series as a whole, and instead should analyze each segment individually. This is known as the regime switching modelling.



Segment	Time Period	Segment	Time Period
1	1990/01 – 1993/06	4	2003/01 – 2007/09
2	1993/07 – 1998/06	5	2007/10 – 2009/12
3	1998/07 – 2002/12	6	2010/01 – 2019/12

The table above shows the corresponding time periods for each segment in our graph above. For the purpose of our project, we will only consider the 6th regime and analyze that. We will consider the 6th segment as our entire dataset, and perform all the necessary steps in the Box-Jenkins framework.

The first step is to isolate the sixth segment into it's own data. Since we now have significantly less observations in our time series, we will consider a training dataset containing the first 98% of observations and the remaining 2% will make up our testing dataset.

In [15]:

```
# REGIME 6 Data
regime6_data <- data[240:359,]

# Computing the size of our new training and testing datasets
new_x <- regime6_data$VIX
new_n_total <- length(new_x)
new_pTest = 0.02

new_nTrain <- new_n_total - floor(new_pTest * new_n_total)
new_nTest <- new_n_total - (new_n_total - floor(new_pTest*new_n_total) + 1) + 1

print("Size of the New Training Dataset:")
new_nTrain

print("Size of the New Testing Dataset:")
new_nTest

# Regime 6 Training Dataset
regime6_training_dataset <- regime6_data[1:118,]
regime6_training_dates <- regime6_training_dataset$Date
regime6_training_VIX_Values <- regime6_training_dataset$VIX

# Testing Dataset
regime6_testing_dataset <- regime6_data[119:120,]
regime6_testing_dates <- regime6_testing_dataset$Date
regime6_testing_VIX_Values <- regime6_testing_dataset$VIX
```

```
[1] "Size of the New Training Dataset:"
118
[1] "Size of the New Testing Dataset:"
2
```

With the new testing dataset for our regime 6 data, let's analyze the time plot. It appears that our data is not stationary since we can't really balance the data around a mean VIX value.

We will also look at the plot of the Cumulative mean level, however since we do not have many observations, we will take the results with a grain of salt. The plot of cumulative mean level suggests our data is also not stationary.

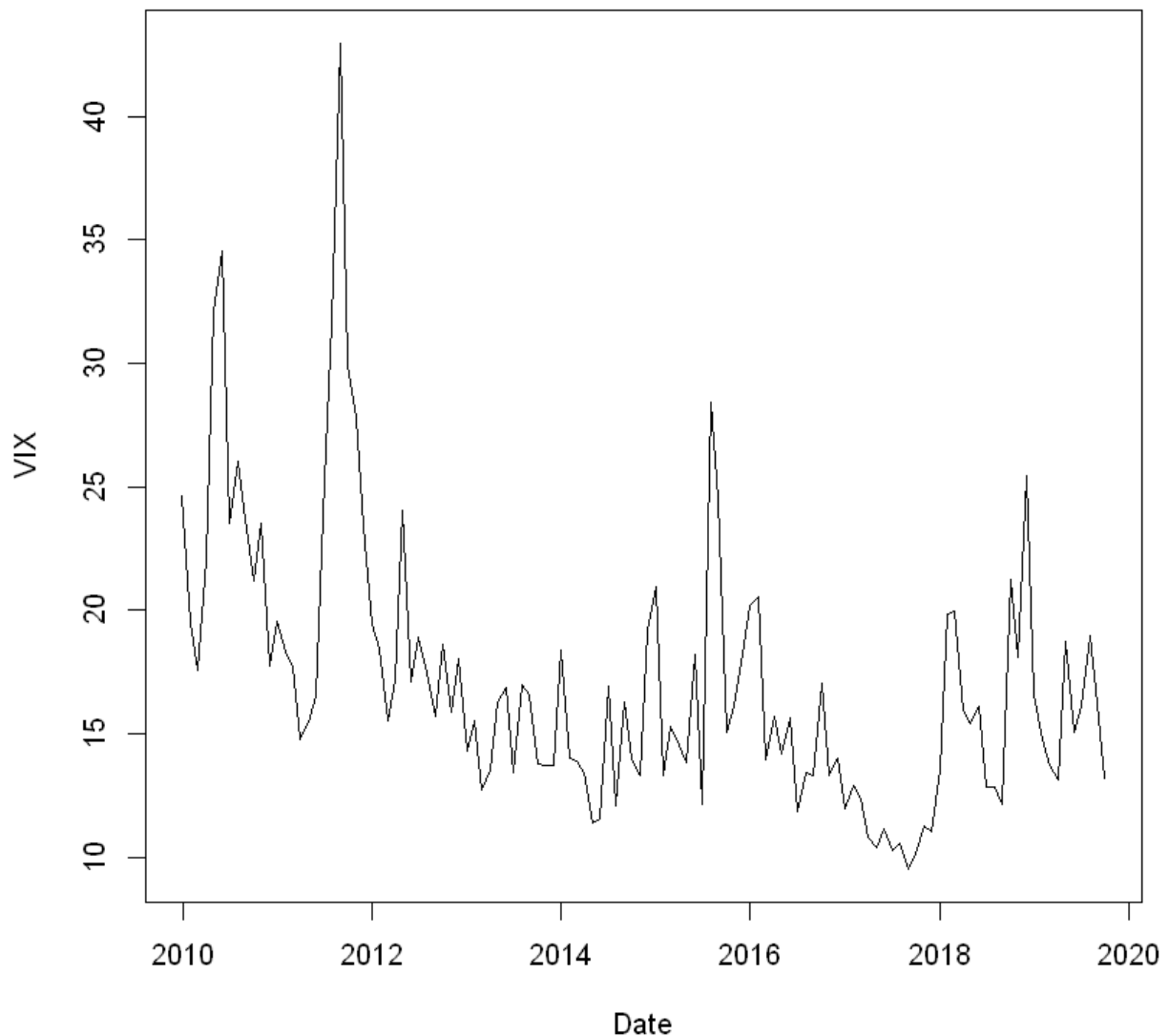
So far our data appears to be non-stationary, but we will conduct further tests.

In [16]:

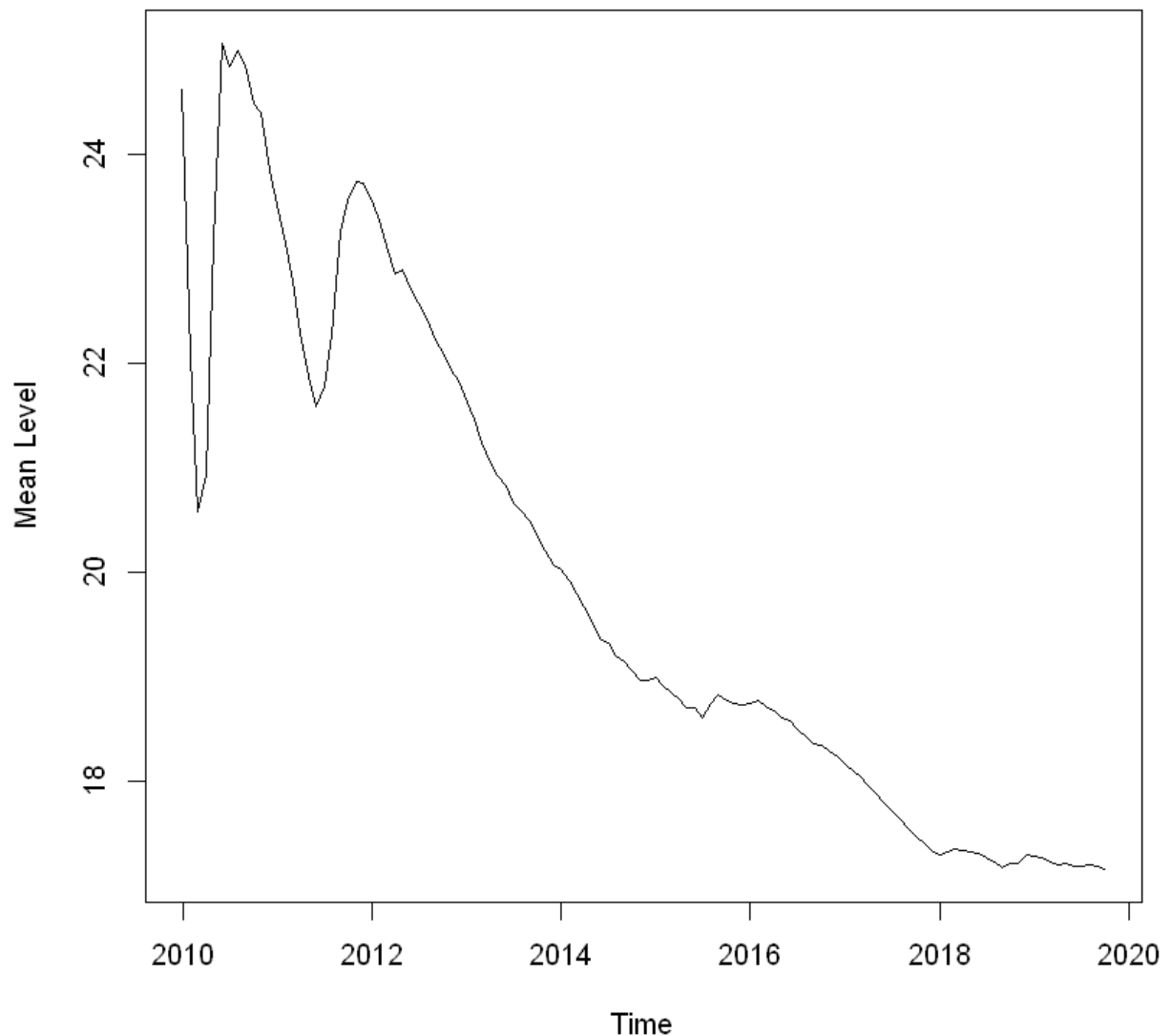
```
# Time Plot
plot(regime6_training_dates, regime6_training_VIX_Values, type = "l", xlab = "Date", ylab = "VIX")

# Cumulative mean
regime6_cummeanVIX <- cumsum(regime6_training_VIX_Values) / seq_along(regime6_training_VIX_Values)
plot(regime6_training_dates, regime6_cummeanVIX, type = "l", xlab="Time", ylab = "Mean")
```

Regime 6, Time Plot of Monthly VIX



Regime 6, Plot of the Cumulative Mean



In the ACF plot we see that at each lag given, the bars exceed the blue threshold bands. As a result, this indicates that the signature is largely Autoregressive.

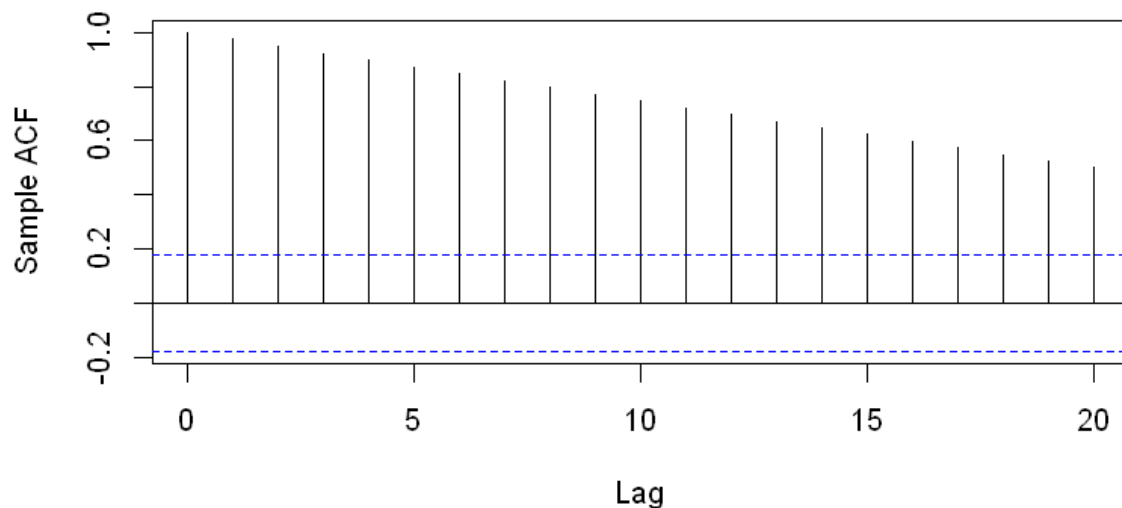
In the Partial ACF plot, our above conclusion is solidified because at lag 1 and onwards we observe that the sample PACF is zero. Therefore, we can conclude that our model's AR part is more powerful than the MA part.

In [17]:

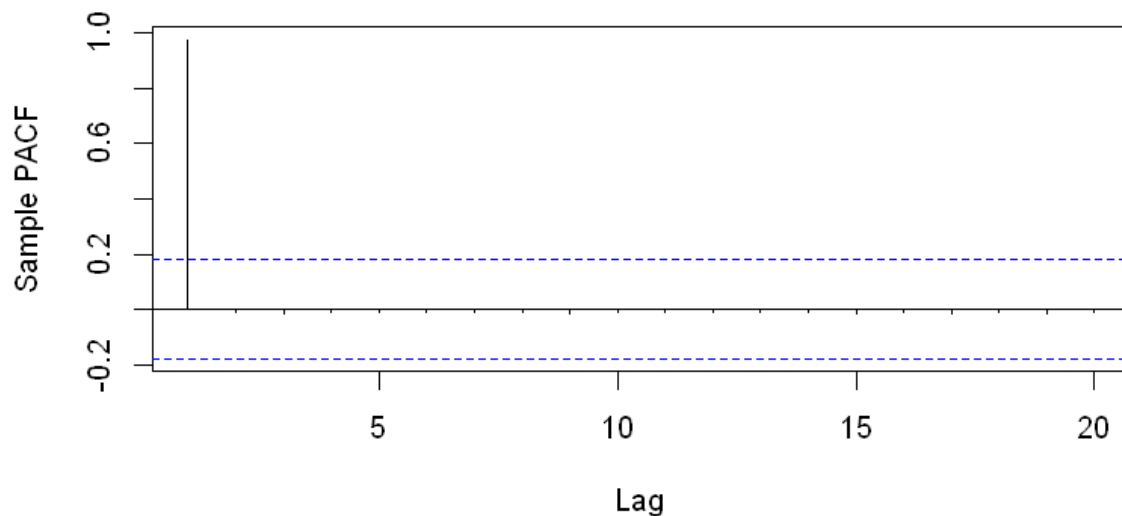
```
#Regime 6

#ACF, PACF
par(mfrow = c(2,1), mar = c(4, 4, 4, 4))
acf(regime6_training_dates, xlab = "Lag", ylab = "Sample ACF", main = "Regime 6, Auto C
acf(regime6_training_dates, type = "partial", xlab = "Lag", ylab = "Sample PACF", main
```

Regime 6, Auto Correlation Function



Regime 6, Partial Auto Correlation Function



The ADF and KPSS Tests yield the following results:

ADF: P-value = 0.01 suggests we reject the null hypothesis and conclude that our data is stationary.

KPSS Test for Level Stationarity: P-value = 0.01 suggests we reject the null hypothesis and conclude that the time series is non-stationary.

KPSS Test for Trend Stationarity: P-value = 0.1 suggests we fail to reject the null hypothesis and conclude that there is not enough evidence to say that the model isn't trend stationary.

```
In [18]: # Augmented Dickey-Fuller (ADF) Test
regime6_training_dates.adf <- adf.test(regime6_training_dates, alternative = "stationar
regime6_training_dates.adf

# Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test
regime6_training_dates.kpss_level <- kpss.test(regime6_training_dates, null = "Level")
regime6_training_dates.kpss_level
```



```
regime6_training_dates.kpss_trend <- kpss.test(regime6_training_dates, null = "Trend")
regime6_training_dates.kpss_trend
```

Warning message in `adf.test(regime6_training_dates, alternative = "stationary")`:
 "p-value smaller than printed p-value"
 Augmented Dickey-Fuller Test

data: regime6_training_dates
 Dickey-Fuller = -5.0094, Lag order = 4, p-value = 0.01
 alternative hypothesis: stationary

Warning message in `kpss.test(regime6_training_dates, null = "Level")`:
 "p-value smaller than printed p-value"
 KPSS Test for Level Stationarity

data: regime6_training_dates
 KPSS Level = 2.4602, Truncation lag parameter = 4, p-value = 0.01
 Warning message in `kpss.test(regime6_training_dates, null = "Trend")`:
 "p-value greater than printed p-value"
 KPSS Test for Trend Stationarity

data: regime6_training_dates
 KPSS Trend = 0.059378, Truncation lag parameter = 4, p-value = 0.1

Since there is no conclusive argument that our data is stationary and since the time plot suggests non-stationary, we will transform our time series by the method of differencing.

In [19]:

```
# Regime 6
# Differenced Series
r6_y = diff(regime6_training_VIX_Values)

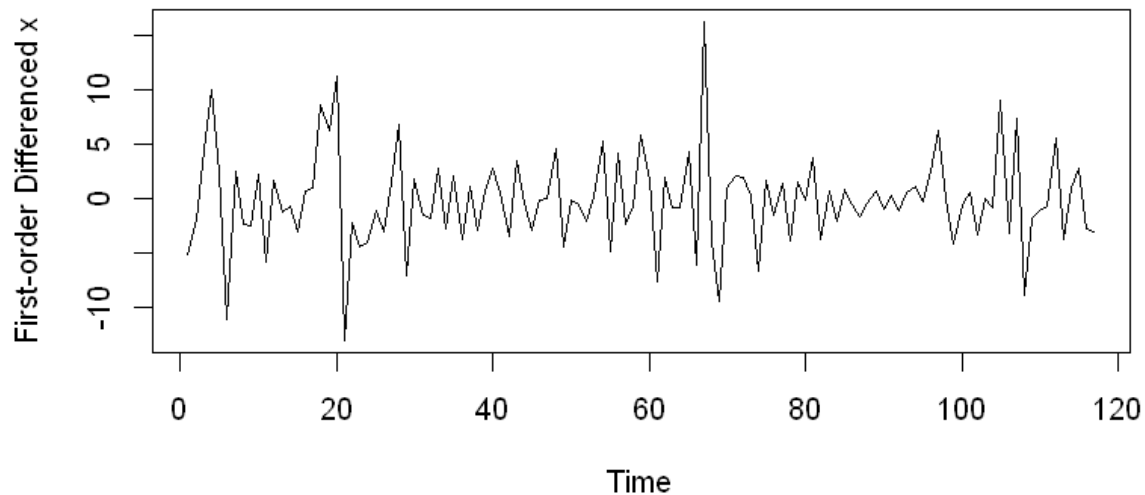
r6_cummean_y <- cumsum(r6_y) / seq_along(r6_y)

par(mfrow = c(2,1), mar = c(4, 4, 4, 4))

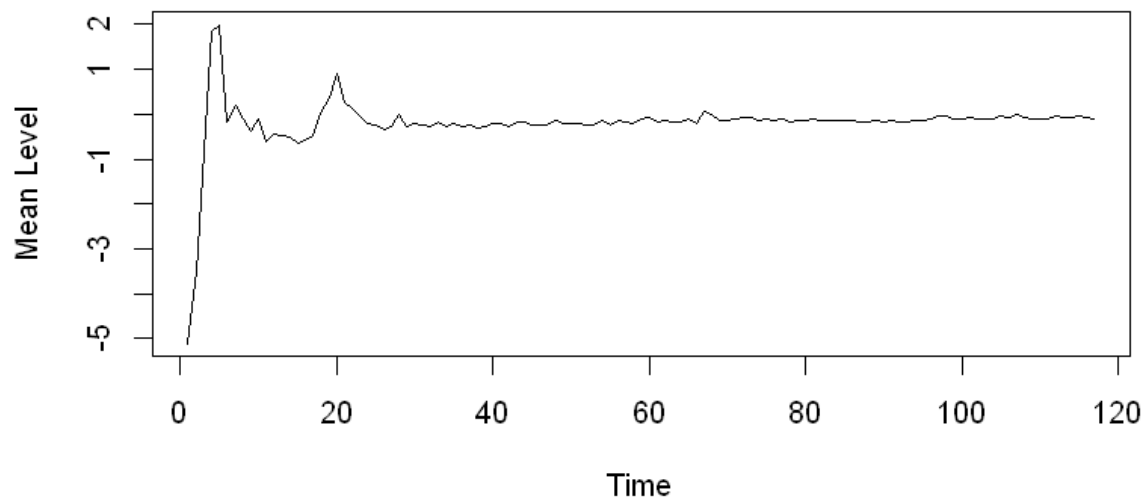
# Time Plot
plot(r6_y, type = "l", xlab = "Time", ylab = "First-order Differenced x", main = "Regim

# Plot of Cumulative Mean
plot(r6_cummean_y, type = "l", xlab = "Time", ylab = "Mean Level", main = "Regime 6, Plo
```

Regime 6, Time Plot (Differenced)



Regime 6, Plot of Cumulative Mean (Differenced)



The above plots suggests that our data is now stationary.

In [20]:

```
# Augmented Dickey-Fuller (ADF) Test for Differenced Series
r6_y.adf <- adf.test(r6_y, alternative = "stationary")
r6_y.adf

# Kwiatkowski-Phillips-Schmidt-Shin (KPSS) Test for Differenced Series
r6_y.kpss_level <- kpss.test(r6_y, null = "Level")
r6_y.kpss_level
r6_y.kpss_trend <- kpss.test(r6_y, null = "Trend")
r6_y.kpss_trend
```

```
Warning message in adf.test(r6_y, alternative = "stationary"):
"p-value smaller than printed p-value"
Augmented Dickey-Fuller Test
```

```
data: r6_y
Dickey-Fuller = -6.7699, Lag order = 4, p-value = 0.01
alternative hypothesis: stationary
```

```
Warning message in kpss.test(r6_y, null = "Level"):
"p-value greater than printed p-value"
      KPSS Test for Level Stationarity
```

```
data: r6_y
KPSS Level = 0.026673, Truncation lag parameter = 4, p-value = 0.1
Warning message in kpss.test(r6_y, null = "Trend"):
"p-value greater than printed p-value"
      KPSS Test for Trend Stationarity
```

```
data: r6_y
KPSS Trend = 0.021355, Truncation lag parameter = 4, p-value = 0.1
```

The ADF Test on the regime 6 differenced data yields a p-value of 0.01, which is less than alpha at 0.05. Therefore we reject the ADF test's null hypothesis, and thus this implies that our the differenced series is stationary.

The KPSS Test yields a p-value of 0.1 for both cases. We fail to reject each of our null hypotheses because the p-value is greater than alpha at 0.05. Therefore for the differenced series, there is no evidence to conclude that the series is non-stationary because of a unit root.

Now that we have stationary data, we can proceed to determining the candidate models.

In [21]:

```
# Extended ACF Table
r6_y.eacf <- eacf(r6_y, ar.max = 5, ma.max = 5)

#NOTE: Due to some issues with Jupyter Notebook and the version of R,
#       I wasn't able to obtain the EACF Table. Instead I used my
#       R-Studio instead, and obtained the EACF Table.
```

```
Error in eacf(r6_y, ar.max = 5, ma.max = 5): could not find function "eacf"
Traceback:
```

```
> # David Quan
> # Extended ACF Table
> r6_y.eacf <- eacf(r6_y, ar.max = 5, ma.max = 5)
AR/MA
  0 1 2 3 4 5
0 x o o o o o
1 x o o o o o
2 x o o o o o
3 x o x o o o
4 x x x o o o
5 x o x o o o
```

In [22]:

```
r6_y.aic <- matrix(0, 5, 5)
r6_y.bic <- matrix(0, 5, 5)

for (i in 0:4) for (j in 0:4){
  r6_y.fit <- arima(r6_y, order = c(i,0,j), method = "ML", include.mean = TRUE)
  r6_y.aic[i + 1, j + 1] <- r6_y.fit$aic
  r6_y.bic[i + 1, j + 1] <- BIC(r6_y.fit)
}

r6_y.aic_vec <- sort(unmatrix(r6_y.aic, byrow = FALSE))[1:13]
r6_y.bic_vec <- sort(unmatrix(r6_y.bic, byrow = FALSE))[1:13]
```

```
#NOTE: Due to some issues with Jupyter Notebook and the version of R,
#       the AIC and BIC vectors were done in my R-studio
```

Warning message in log(s2):

"NaNs produced"Warning message in log(s2):

"NaNs produced"Warning message in log(s2):

"NaNs produced"Warning message in log(s2):

"NaNs produced"Warning message in log(s2):

"NaNs produced"Warning message in log(s2):

"NaNs produced"Warning message in log(s2):

"NaNs produced"

Error in unmatrix(r6_y.aic, byrow = FALSE): could not find function "unmatrix"
Traceback:

```
1. sort(unmatrix(r6_y.aic, byrow = FALSE))
```

```
> #David Quan
> r6_y.aic_vec
  r4:c4  r2:c2  r4:c5  r5:c4  r3:c2  r2:c3  r5:c5  r1:c5  r1:c4  r2:c4  r3:c5  r4:c2  r3:c3
654.8780 656.1999 656.8779 656.8779 657.9450 657.9985 658.8136 658.8320 658.8880 658.9254 658.9843 659.1996 659.7109
> r6_y.bic_vec
  r2:c2  r3:c2  r2:c3  r1:c4  r1:c5  r2:c4  r4:c2  r3:c3  r1:c3  r1:c2  r4:c4  r2:c1  r2:c5
669.2486 673.7559 673.8093 674.6988 677.4051 677.4985 677.7727 678.2839 678.5879 678.9408 678.9754 680.5741 681.4660
```

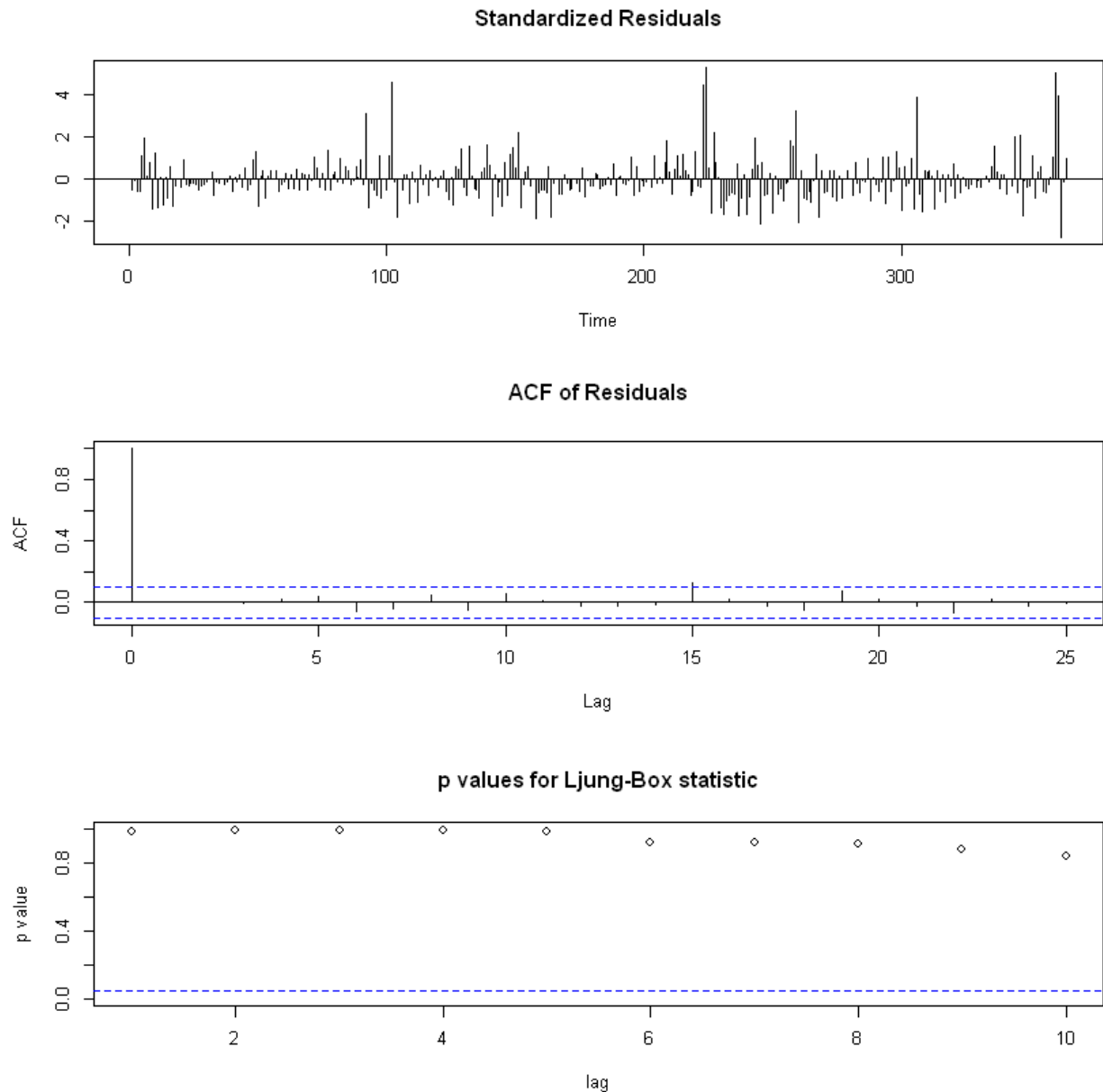
Similar to before, there is not a huge significant change in the AIC and BIC for our candidate models. Therefore, we want to choose a model that is parsimonious and is also prioritizes the AR part over the MA part. The ARMA(2,2) model and the ARMA(3,2) models look promising. We will carry out our fitting and Diagnostics in the next section, titled *Fitting and Diagnostics: Regimes*.

Fitting and Diagnostics

Model 1: ARMA(3,2)

In [23]:

```
#ARMA(3,2) Model
y.fit_1 <- arima(y, order = c(3,0,2), method = "ML", include.mean = TRUE)
tsdiag(y.fit_1)
```



For the plot of the standardized residuals, they are more or less scattered around the horizontal line at zero, however it seems like there are slightly more negative values. Also, the extreme values: around observation 105, 225, and 360 are all above the horizontal line. Since there appears to be a slight pattern, this suggests that the residuals don't have a white noise behaviour.

Next we have the Plot of ACF of Residuals. Notice that at lag point 15, the blue band/threshold is passed. This means that there is evidence against pure randomness, and so there exists structure.

Lastly, we have the Ljung-Box test which checks whether the residuals are uncorrelated or not. The null hypothesis states that the error terms are uncorrelated. Since in the graph, no points fall under the 0.05 blue dotted line, we fail to reject the null hypothesis. Therefore we conclude that the error terms are uncorrelated.

There are two tools to help check whether the residuals are normally distributed.

First there is The Shapiro Test, which states the following hypotheses: H_0 : Residuals are normally distributed, versus H_1 : Residuals are not normally distributed. Now the p-value of the Shapiro Test is very small, which implies that we reject the null hypothesis. Therefore, The residuals are not normally distributed.

The second test, which is a visual inspection of the QQ Plot, clearly indicates that the residuals are not normally distributed since the points largely deviate from the QQ line.

In [24]:

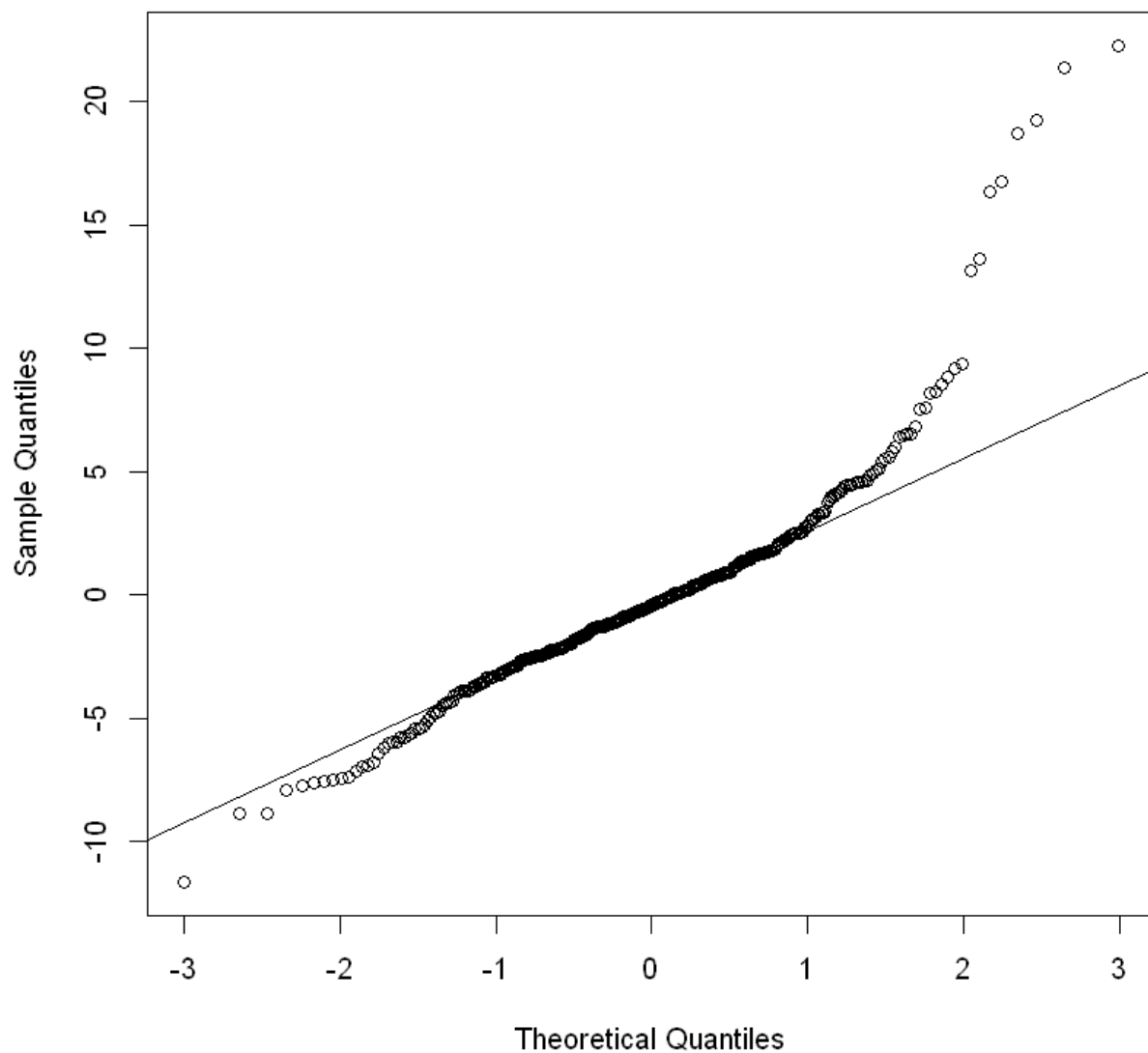
```
#QQplot
qqnorm(residuals(y.fit_1))
qqline(residuals(y.fit_1))

#Shapiro Test
shapiro.test(residuals(y.fit_1))
```

Shapiro-Wilk normality test

```
data: residuals(y.fit_1)
W = 0.88493, p-value = 6.935e-16
```

Normal Q-Q Plot



Model 2: ARMA(2,3)

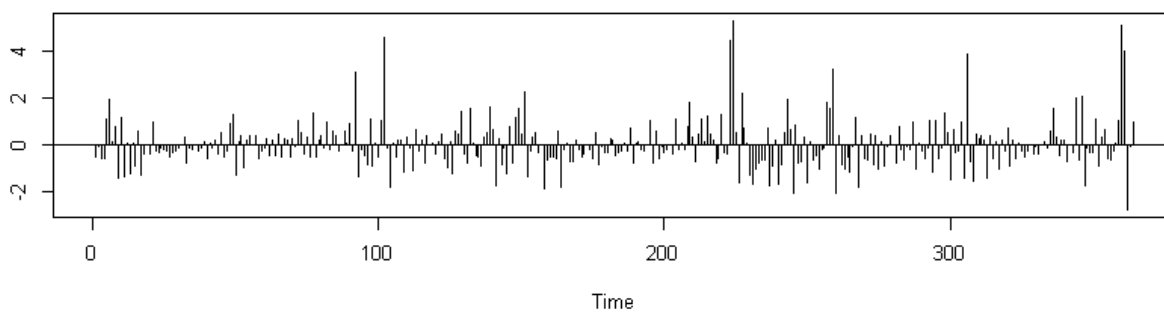
In [25]:

```
# ARMA(2,3) Model
y.fit_2 <- arima(y, order = c(2,0,3), method = "ML", include.mean = TRUE)
tsdiag(y.fit_2)

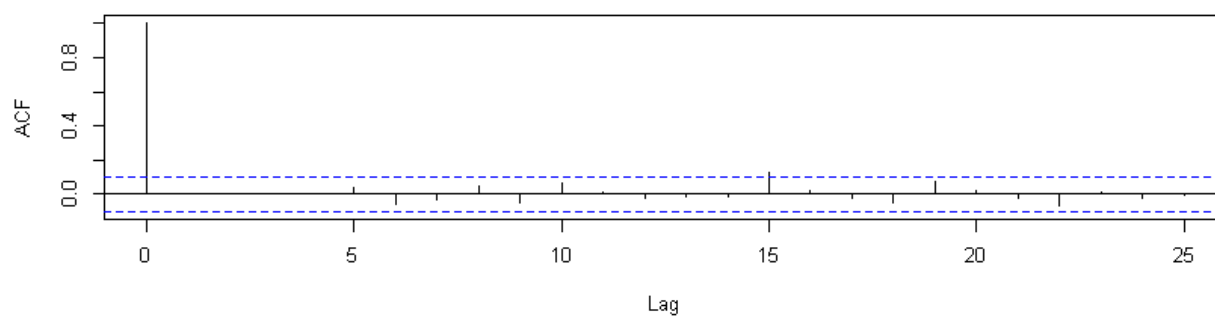
# QQ Plot
qqnorm(residuals(y.fit_2))
qqline(residuals(y.fit_2))

#Shapiro Test
shapiro.test(residuals(y.fit_2))
```

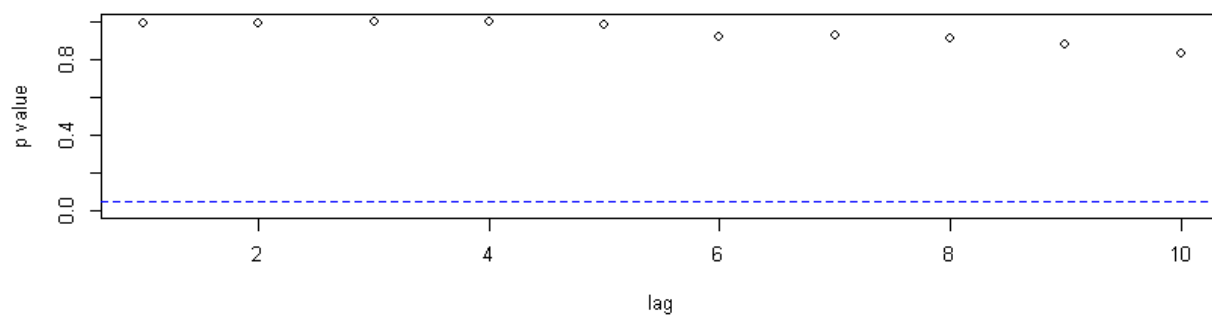
Standardized Residuals



ACF of Residuals



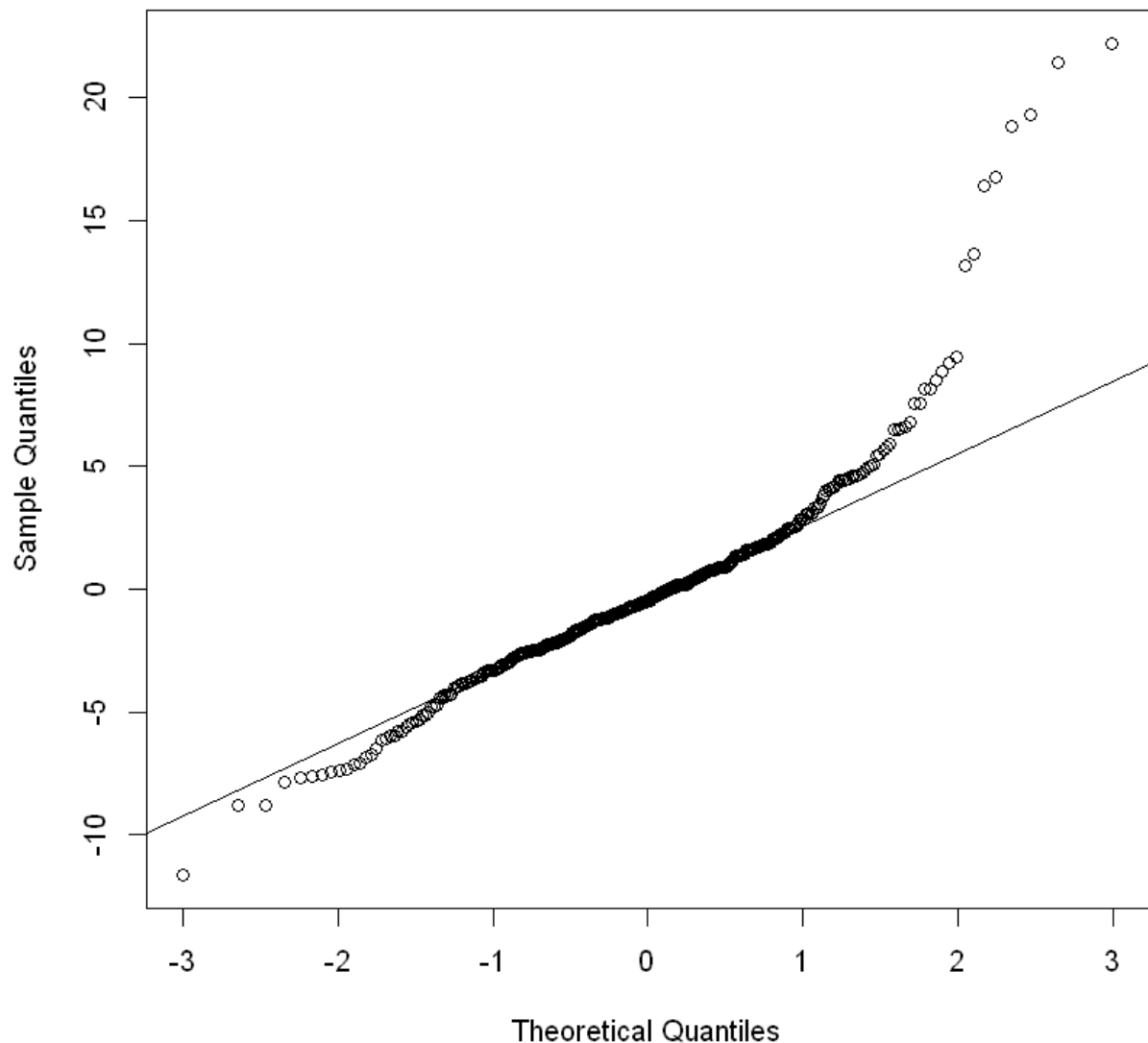
p values for Ljung-Box statistic



Shapiro-Wilk normality test

```
data: residuals(y.fit_2)  
W = 0.8846, p-value = 6.558e-16
```


Normal Q-Q Plot



Now applying the same steps as was performed for Model 1, we obtain similar results.

The plot of the standardized residuals suggests that the residuals don't have a white noise behaviour due to a slight pattern. The plot of ACF of residuals also has the threshold being exceeded at lag point 15, which also implies we don't have pure randomness. The Ljung Box-test suggests that we fail to reject the null hypothesis and therefore conclude that the error terms are uncorrelated.

As for checking normality, in the Shapiro Test, the p-value is also very small, thus we reject the null hypothesis and conclude the residuals are not normally distributed. The Normal QQ Plot also confirms this result of not being normally distributed because the points largely deviate from the QQ line.

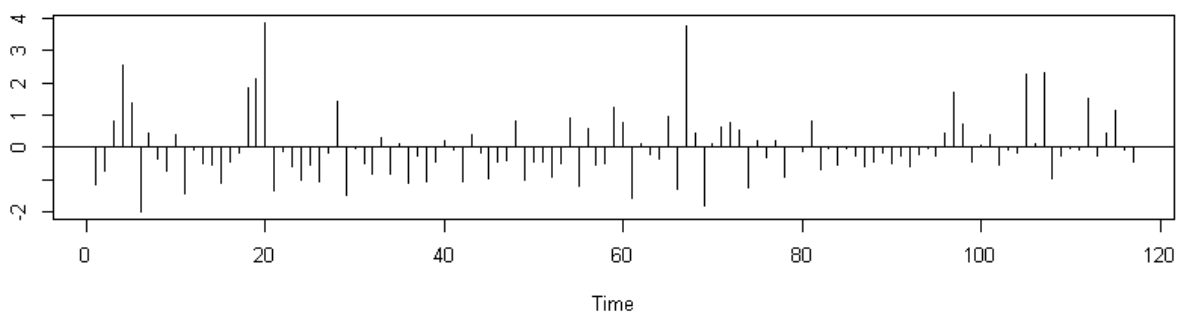
Fitting and Diagnostics: Regimes

Model 1: ARMA(2,2)

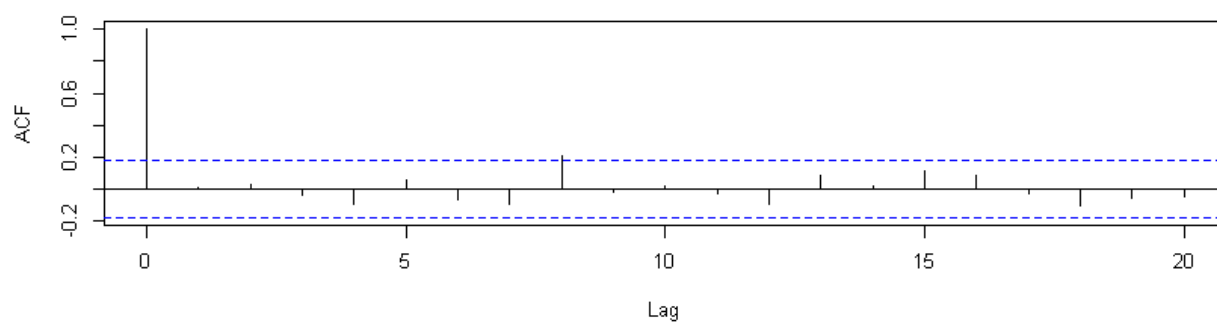
```
In [26]: # The Regime 6 Data

# ARMA (2,2)
r6_y.fit_1 <- arima(r6_y, order = c(2,0,2), method = "ML", include.mean = TRUE)
tsdiag(r6_y.fit_1)
```

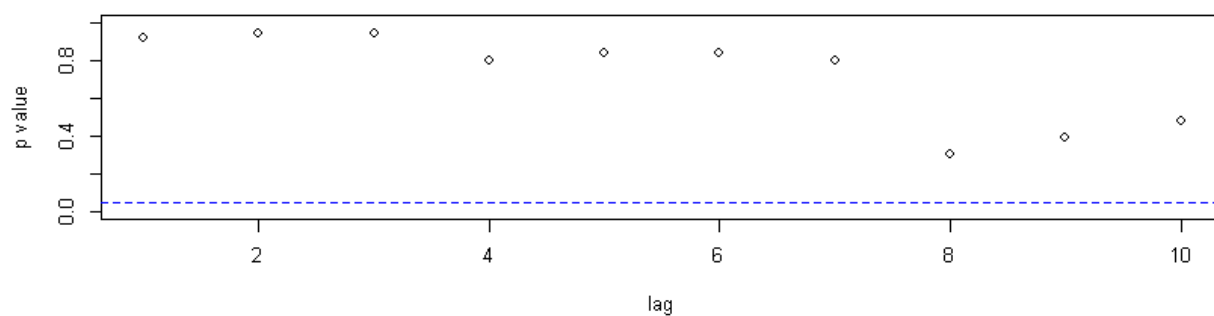
Standardized Residuals



ACF of Residuals



p values for Ljung-Box statistic

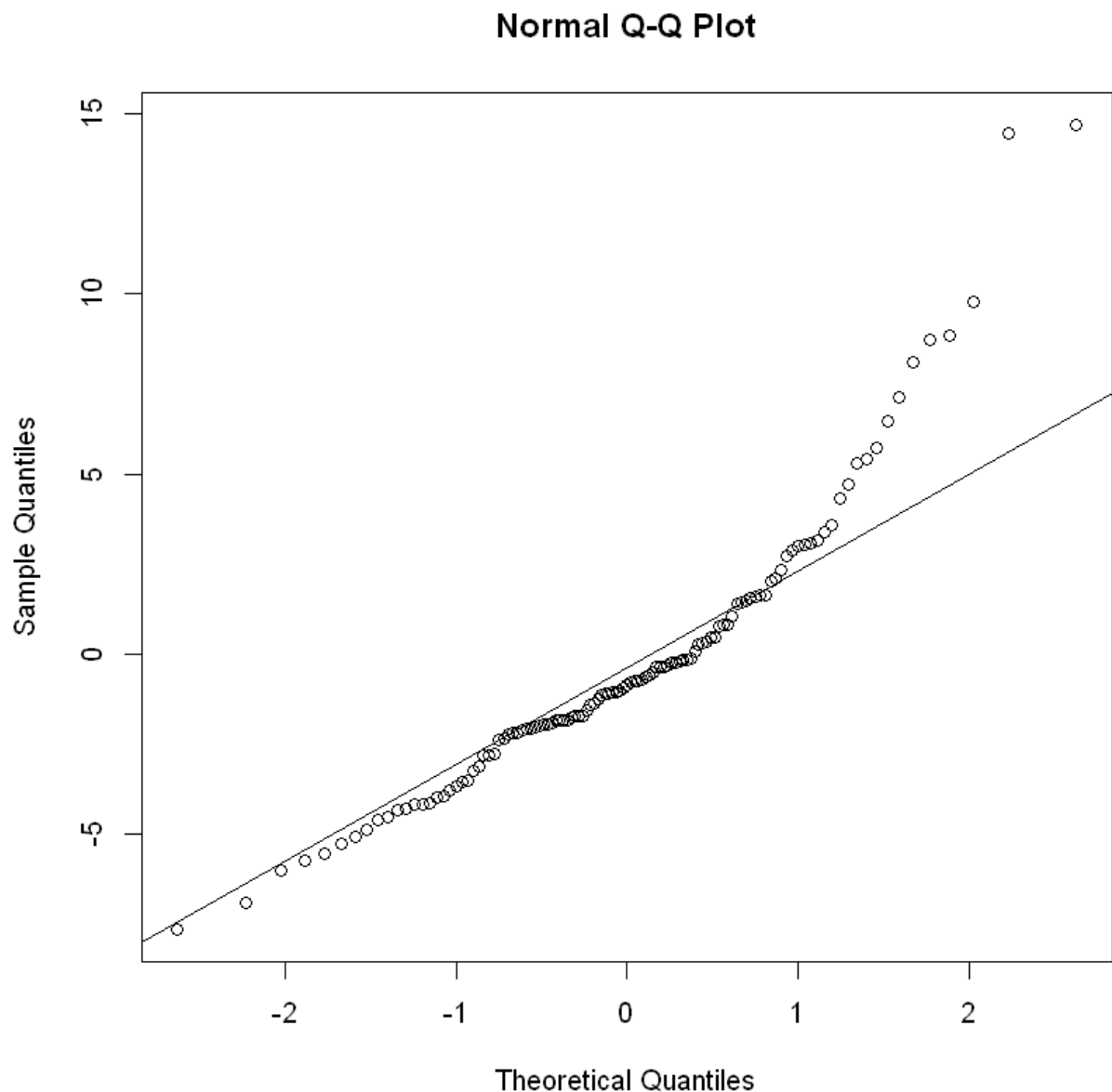


```
In [27]: # QQ Plot
qqnorm(residuals(r6_y.fit_1))
qqline(residuals(r6_y.fit_1))

# Shapiro Test
shapiro.test(residuals(r6_y.fit_1))
```

Shapiro-Wilk normality test

data: residuals(r6_y.fit_1)
W = 0.90848, p-value = 7.19e-07



We still obtain that for this model, we have:

The plot of the standardized residuals suggests that the residuals don't have a white noise behaviour due to a slight pattern. The plot of ACF of residuals also has the threshold being exceeded at lag point 8, which also implies we don't have pure randomness. The Ljung Box-test suggests that we fail to reject the null hypothesis and therefore conclude that the error terms are uncorrelated.

As for checking normality, in the Shapiro Test, the p-value is also very small, thus we reject the null hypothesis and conclude the residuals are not normally distributed. The Normal QQ Plot also confirms this result of not being normally distributed because the points largely deviate from the QQ line.

Model 2: ARMA(3,2)

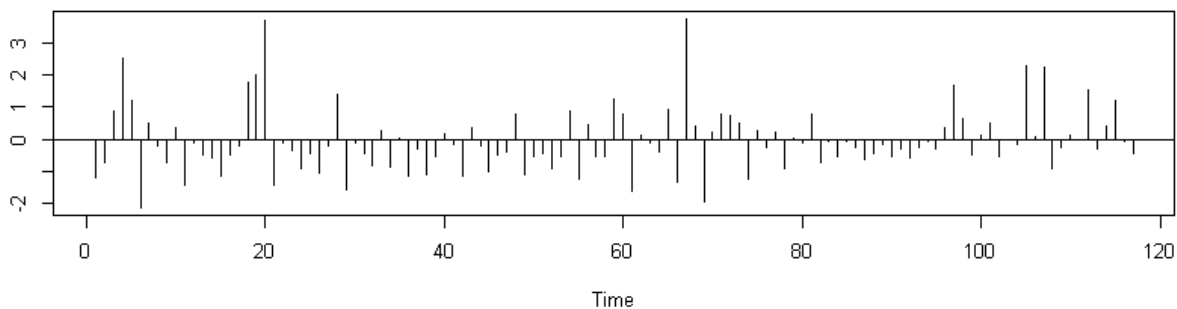
```
In [28]: # The Regime 6 Data
```

```
# ARMA (3,2)
r6_y.fit_2 <- arima(r6_y, order = c(3,0,2), method = "ML", include.mean = TRUE)
tsdiag(r6_y.fit_2)

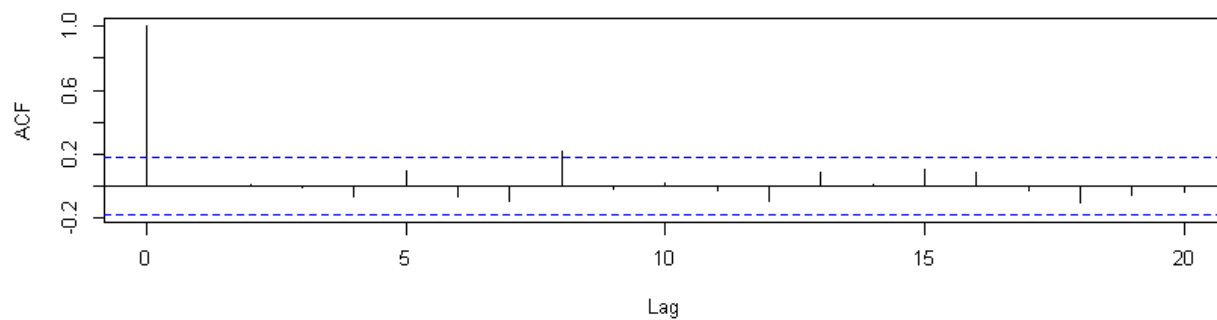
# QQ Plot
qqnorm(residuals(r6_y.fit_2))
qqline(residuals(r6_y.fit_2))

# Shapiro Test
shapiro.test(residuals(r6_y.fit_2))
```

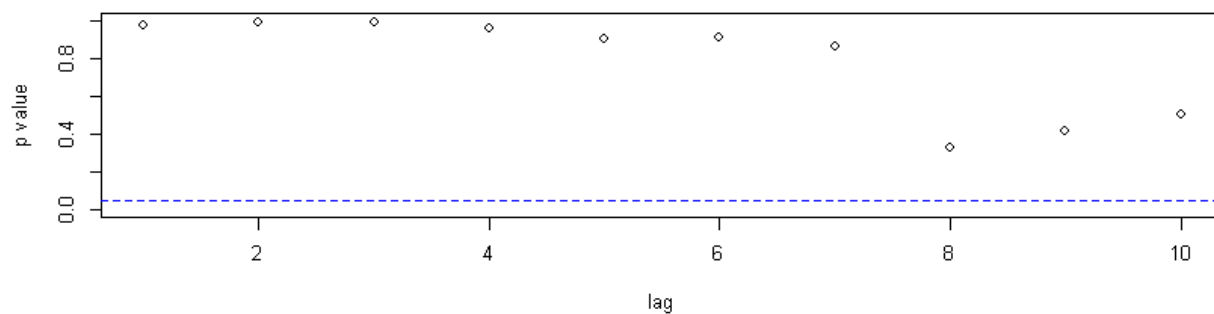
Standardized Residuals



ACF of Residuals

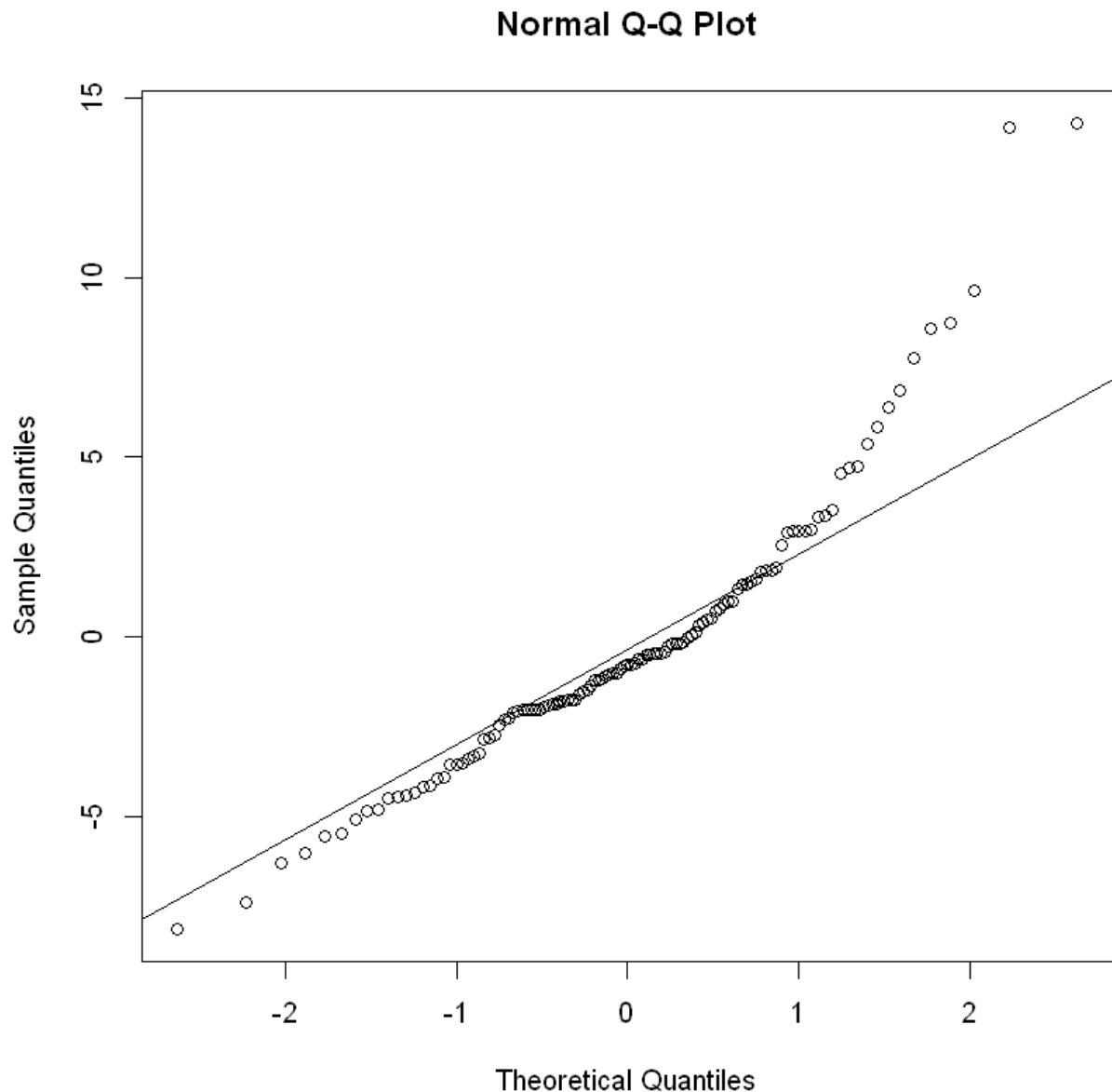


p values for Ljung-Box statistic



Shapiro-Wilk normality test

```
data: residuals(r6_y.fit_2)
W = 0.92042, p-value = 3.292e-06
```



We still obtain that for this model, we have:

The plot of the standardized residuals suggests that the residuals don't have a white noise behaviour due to a slight pattern. The plot of ACF of residuals also has the threshold being exceeded at lag point 8, which also implies we don't have pure randomness. The Ljung Box-test suggests that we fail to reject the null hypothesis and therefore conclude that the error terms are uncorrelated.

As for checking normality, in the Shapiro Test, the p-value is also very small, thus we reject the null hypothesis and conclude the residuals are not normally distributed. The Normal QQ Plot also confirms this result of not being normally distributed because the points largely deviate from the QQ line.

Since after all our fitting and diagnostics, we still obtain models that don't have normality in the error terms, we will still proceed to selecting the ARMA(2,2) model because it's the most parsimonious. We did have that the Autoregressive part should be prioritized over the Moving

average part, however if two models are both resulting in the same performance, we favour the simpler model by the principle of parsimony.

Forecasting

We will now perform forecasting for ARIMA(2,1,2). If we recall, we took the difference of our model once to convert it to stationary, and therefore $d = 1$. Therefore, the ARMA(2,2) model we selected is really ARIMA(2,1,2).

Below we will first compare the 1st step ahead forecast using the predict function, versus our ARIMA (2,0,2) Model. It appears that the forecast that we computed: -0.6022366 is close to the forecast computed by the predict function: -0.5958528. This is a good sign.

In [45]:

```
# Take the regime6_data
regime6_data <- data[240:359,]

x <- regime6_data$VIX

y <- diff(x)
n <- length(y)

y.fit <- arima(y, order = c(2,0,2), method = "ML", include.mean = TRUE)
n <- length(y)

# Finding the coefficients
y.fit$coef
```

```
ar1      0.188301424216041
ar2      0.296658188892545
ma1      -0.610344079315992
ma2      -0.389649378950634
intercept -0.0772823795409107
```

In [30]:

```
# 1 Step ahead forecast
y.forecast_pred <- predict(y.fit, 1)
y.forecast_pred

# Return the last tail end 6 residuals of the model
tail(y.fit$residuals)

# Want the last 2 residuals as our MA part is of order 2
# Will need the previous 2 observations to forecast because of the order 2 for the AR p

# First term: Intercept -> replaced with the drift term.
# Second term: Dot product btw the last 2 observation and the 2 AR coefficients
# Third term: Dot product btw the last 2 residuals and the 2 MA coefficients
# Finding the drift term, delta:
drift <- y.fit$coef[5] * (1 - sum(y.fit$coef[1:2]))

y.forecast <- drift + y[(n-1):n] %*% rev(y.fit$coef[1:2]) + y.fit$residuals[(n-1):n] %*%
y.forecast
```

*#Notice that our forecast that we computed: -0.6022366 is close to the
#forecast computed by the predict function: -0.5958528*

ERROR while rich displaying an object: Error in if (many_rows) {: argument is of length zero

Traceback:

```
1. FUN(X[[i]], ...)
2. tryCatch(withCallingHandlers({
  .   if (!mime %in% names(repr::mime2repr))
  .     stop("No repr_* for mimetype ", mime, " in repr::mime2repr")
  .   rpr <- repr::mime2repr[[mime]](obj)
  .   if (is.null(rpr))
  .     return(NULL)
  .   prepare_content(is.raw(rpr), rpr)
  . }, error = error_handler), error = outer_handler)
3. tryCatchList(expr, classes, parentenv, handlers)
4. tryCatchOne(expr, names, parentenv, handlers[[1L]])
5. doTryCatch(return(expr), name, parentenv, handler)
6. withCallingHandlers({
  .   if (!mime %in% names(repr::mime2repr))
  .     stop("No repr_* for mimetype ", mime, " in repr::mime2repr")
  .   rpr <- repr::mime2repr[[mime]](obj)
  .   if (is.null(rpr))
  .     return(NULL)
  .   prepare_content(is.raw(rpr), rpr)
  . }, error = error_handler)
7. repr::mime2repr[[mime]](obj)
8. repr_html.list(obj)
9. repr_list_generic(obj, "html", "\t<li>%s</li>\n", "\t<dt>$$s</dt>\n\t\t<dd>%s</dd>\n",
  .   "<strong>$$s</strong> = %s", "<ol>\n%s</ol>\n", "<dl>\n%s</dl>\n",
  .   numeric_item = "\t<dt>[[s]]</dt>\n\t\t<dd>%s</dd>\n", escape_fun = html_escape)
10. lapply(vec, format2repr[[fmt]])
11. FUN(X[[i]], ...)
12. repr_html.ts(X[[i]], ...)
13. repr_ts_generic(obj, repr_html.matrix, ...)
14. repr_func(m, ..., rows = nrow(m), cols = ncol(m))
15. repr_matrix_generic(obj, "<table>\n%s</table>\n", "<thead><tr>%s</tr></thead>\n",
  .   "<th></th>", "<th scope=col>%s</th>", "<tbody>\n%s</tbody>\n",
  .   "\t<tr>%s</tr>\n", "<th scope=row>%s</th>", "<td>%s</td>",
  .   escape_fun = html_escape_vec, ...)
16. ellip_limit_arr(flatten(x), rows, cols)
17. arr_partition(a, rows, cols)
ERROR while rich displaying an object: Error in repr_matrix_generic(obj, "\n%s</table>\n", sprintf("%s\n|s|\n", : formal argument "cols" matched by multiple actual arguments
```

Traceback:

```
1. FUN(X[[i]], ...)
2. tryCatch(withCallingHandlers({
  .   if (!mime %in% names(repr::mime2repr))
  .     stop("No repr_* for mimetype ", mime, " in repr::mime2repr")
  .   rpr <- repr::mime2repr[[mime]](obj)
  .   if (is.null(rpr))
  .     return(NULL)
  .   prepare_content(is.raw(rpr), rpr)
  . }, error = error_handler), error = outer_handler)
3. tryCatchList(expr, classes, parentenv, handlers)
4. tryCatchOne(expr, names, parentenv, handlers[[1L]])
```

```

5. doTryCatch(return(expr), name, parentenv, handler)
6. withCallingHandlers({
  . if (!mime %in% names(repr::mime2repr))
  .   stop("No repr_* for mimetype ", mime, " in repr::mime2repr")
  . rpr <- repr::mime2repr[[mime]](obj)
  . if (is.null(rpr))
  .   return(NULL)
  . prepare_content(is.raw(rpr), rpr)
  . }, error = error_handler)
7. repr::mime2repr[[mime]](obj)
8. repr_markdown.list(obj)
9. repr_list_generic(obj, "markdown", "%s. %s\n", "%$s\n: %s\n",
  .   "***$s** = %s", "%s\n\n", numeric_item = "[[%s]]\n: %s\n",
  .   item_uses_numbers = TRUE, escape_fun = html_escape)
10. lapply(vec, format2repr[[fmt]])
11. FUN(X[[i]], ...)
12. repr_markdown.ts(X[[i]], ...)
13. repr_ts_generic(obj, repr_markdown.matrix, ...)
14. repr_func(m, ..., rows = nrow(m), cols = ncol(m))
ERROR while rich displaying an object: Error in rep(colspec$col, ncol(obj)): invalid 'ti
mes' argument

```

Traceback:

```

1. FUN(X[[i]], ...)
2. tryCatch(withCallingHandlers({
  . if (!mime %in% names(repr::mime2repr))
  .   stop("No repr_* for mimetype ", mime, " in repr::mime2repr")
  . rpr <- repr::mime2repr[[mime]](obj)
  . if (is.null(rpr))
  .   return(NULL)
  . prepare_content(is.raw(rpr), rpr)
  . }, error = error_handler), error = outer_handler)
3. tryCatchList(expr, classes, parentenv, handlers)
4. tryCatchOne(expr, names, parentenv, handlers[[1L]])
5. doTryCatch(return(expr), name, parentenv, handler)
6. withCallingHandlers({
  . if (!mime %in% names(repr::mime2repr))
  .   stop("No repr_* for mimetype ", mime, " in repr::mime2repr")
  . rpr <- repr::mime2repr[[mime]](obj)
  . if (is.null(rpr))
  .   return(NULL)
  . prepare_content(is.raw(rpr), rpr)
  . }, error = error_handler)
7. repr::mime2repr[[mime]](obj)
8. repr_latex.list(obj)
9. repr_list_generic(obj, "latex", "\\item %s\n", "\\item[\\$s] %s\n",
  .   "\\textbf{\\$s} = %s", enum_wrap = "\\begin{enumerate}\n%s\\end{enumerate}\n",
  .   named_wrap = "\\begin{description}\n%s\\end{description}\n",
  .   numeric_item = "\\item{[[[%s]]} %s\n", escape_fun = latex_escape)
10. lapply(vec, format2repr[[fmt]])
11. FUN(X[[i]], ...)
12. repr_latex.ts(X[[i]], ...)
13. repr_ts_generic(obj, repr_latex.matrix, ..., colspec = colspec)
14. repr_func(m, ..., rows = nrow(m), cols = ncol(m))
15. paste0(paste(rep(colspec$col, ncol(obj)), collapse = ""), colspec$end)
16. paste(rep(colspec$col, ncol(obj)), collapse = "")
$pred
Time Series:
Start = 120
End = 120

```



```
Frequency = 1
[1] -0.5958528
```

```
$se
Time Series:
Start = 120
End = 120
Frequency = 1
[1] 3.817462
```

```
1. 1.62325354004944
2. 4.29756700616518
3. -0.318148751766632
4. -1.82993515017384
5. -0.406871469328646
6. 1.24750125953188
```

```
-0.6022366
```

Since we use differencing we can proceed with the following code on the **testing data**.

The code below is for us to define the function to compute the Root MSE and the Harmonic Mean.

```
In [31]: # Root MSE Function
RMSE <- function(x) {
  fval <- sqrt(sum(x^2) / length(x))
  return(fval)
}

# Harmonic Mean Function
# this is robust to outliers. Captures the smaller magnitudes more compared to large o
harmonic_mean <- function(x){
  fval <- length(x) / sum(1/abs(x))
  return(fval)
}
```

```
In [32]: regime6_data <- data[240:359,]

x <- regime6_data$VIX

# The model we selected
x.train.fit <- arima(x[1:118], order = c(2,1,2), method = "ML", include.mean = TRUE)

# The model we rejected.
x.train.fit2 <- arima(x[1:118], order = c(3,1,2), method = "ML", include.mean = TRUE)

# Root Mean Squared Error
print("RMSE of ARMA(2,2)")
RMSE(x.train.fit$residuals)
print("RMSE of ARMA(3,2)")
RMSE(x.train.fit2$residuals)

# Harmonic Mean
print("Harmonic Mean of ARMA(2,2)")
harmonic_mean(x.train.fit$residuals)
```

```
print("Harmonic Mean of ARMA(3,2)")
harmonic_mean(x.train.fit2$residuals)
```

```
[1] "RMSE of ARMA(2,2)"
3.89915672828466
[1] "RMSE of ARMA(3,2)"
3.87610381273993
[1] "Harmonic Mean of ARMA(2,2)"
0.646795745115671
[1] "Harmonic Mean of ARMA(3,2)"
0.697479186099151
```

Note that for the two models we were looking at were ARMA(2,2) and ARMA(3,2). Now ARMA(2,2) performs better in terms of the harmonic mean as it is slightly lower for the ARMA(2,2) model. However ARMA(3,2) has an insignificant edge for the RMSE. We know that the harmonic mean is more robust to extreme residual outliers, and is therefore a better measurement to analyze than RMSE. In addition ARMA(2,2) is more parsimonious, therefore we chose the better model out of the two.

In [43]:

```
nTrain = 118
nTest = 2

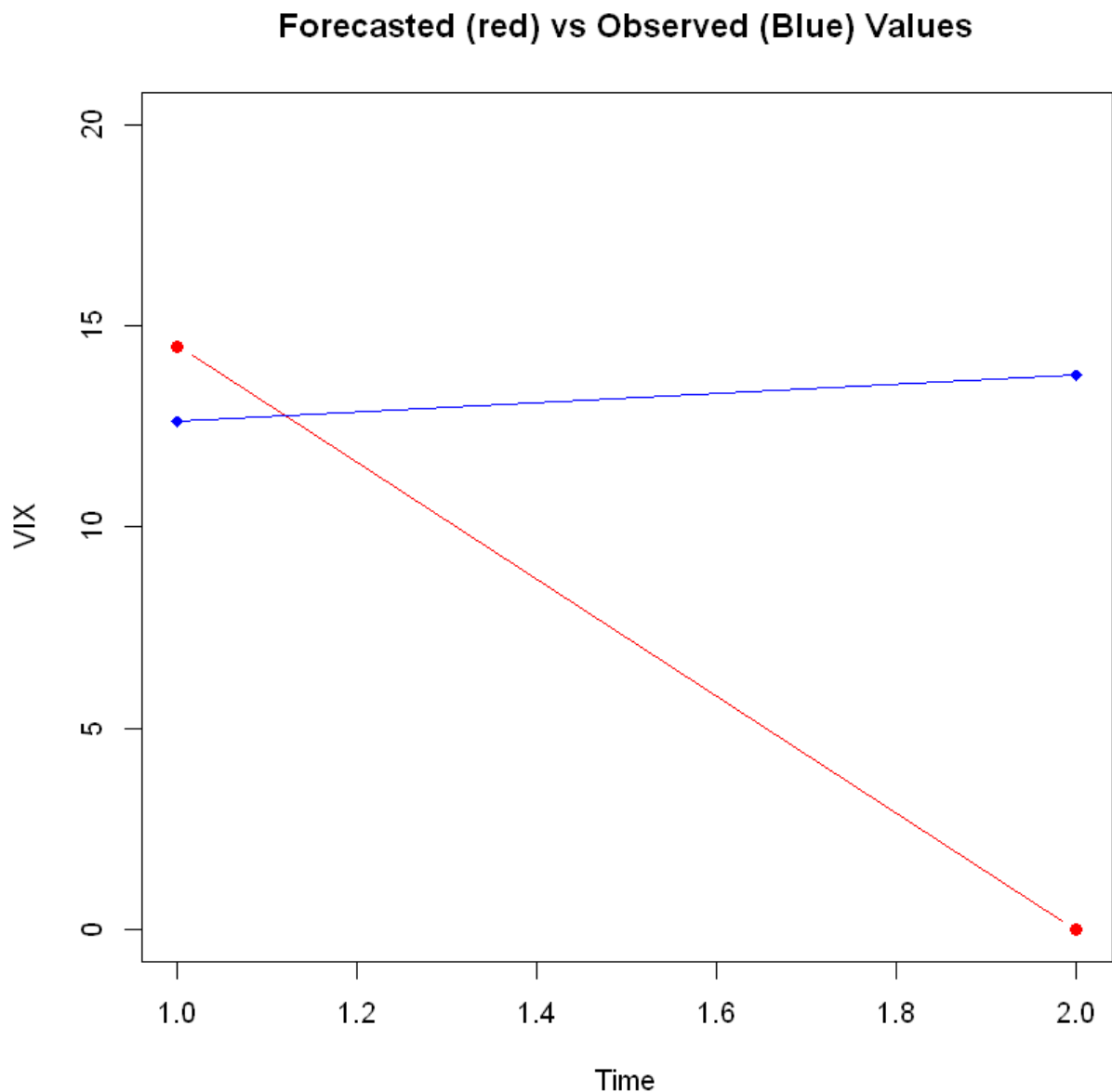
#creating 3 matrices (forecast, residuals, and SE)
x.test.forecast <- matrix(0, nTest, 1)
x.test.residual <- matrix(0, nTest, 1)
x.test.se <- matrix(0, nTest, 1)

# We have 2 test data points

# We will perform 1 step ahead prediction
for (i in (nTrain : (n-1)) ){
  x.test.fit <- arima(x[1:i], order = c(2,0,2), method = "ML", include.mean = TRUE)
  output <- predict(x.test.fit, 1)
  x.test.forecast[i - nTrain + 1, 1] <- as.numeric(output$pred)
  x.test.residual[i - nTrain + 1, 1] <- as.numeric(output$pred) - x[i+1]
  x.test.se[i - nTrain + 1, 1] <- as.numeric(output$se)
}

plot(unlist(x.test.forecast), xlim=c(1,2), ylim=c(0,20), type = "b", pch = 19, col = "r")
points(1,12.6199999999999, col = "blue", pch=18)
points(2,13.7799999999999, col = "blue", pch=18)

segments(1,12.6199999999999, 2,13.7799999999999, col = "blue")
```



NOTE: As I was compiling all my Jupyter codes, suddenly my graph above changed. I may have re-named a variable, or changed an index and that caused the change. Previously, the graph that was outputted has the 2 step ahead forecast was just under the observed value.

Discussion

Throughout the project, we ran into several issues. Firstly, we had issues in finding a model that had normally distributed errors, however they were indeed uncorrelated. There was difficulties in coming up with a good model, and after trial and error work, Professor Ataei suggested we consider splitting up the time series data into segments, also known as regimes. Now we discussed that in each regime, there are difference forces driving the Volatility Index, which include (but are not limited to): Psychological, economical, political, geopolitical, fundamentals, traders, and uncertainty. Since the factors vary in each time period, we cannot consider the time series as a whole, and instead should analyze each segment individually. We performed analysis specifically on the 6th

segment, and after following the box-jenkins framework, the model that we obtained that best described the VIX index was ARMA(2,2).

There were some encountered issues with jupyter (since it was my first time using it), such as not being able to create the EACF Tables and AIC/BIC Vectors in the Jupyter notebook due to a mismatch in versions. Here, certain functions had error messages saying the function can't be found, however it worked just fine in R-studio. One main issue was that as I was compiling and re-running all the cells to submit, my forecasted time series plot suddenly changed, and I did not have time to fix it unfortunately. If an individual were to read this report, they would think that the model is not up to par since it forecasts the 2 step ahead VIX as 0.


Aside from the issues that I experienced, some issues in practice with data analysis is the accuracy of data. Not all data is representative, as it is possible for there to be missing data, or even incorrectly inputted data. For our purposes though, we were provided with data that was already clean, so we didn't have to go through this process. Another issue in deriving conclusions is that there are sometimes some factors that weren't considered. These unknown factors may be playing a role in the outcome, so it is important to have a strong background on the sector you're working with to cover all such factors.

To conclude, it is only possible to predict the VIX index if you consider segments, due to the various factors at play, and still it is quite difficult in general to do so.

Bibliography


Professor Ataei:

Lecture slides, volatility historical graph, Segmentation Graph and corresponding Dates Table

TD Ameritrade: <https://www.youtube.com/watch?v=JwNQbsFQkHc!>
(attachment:image.png)

The Volatility Index (VIX) Explained by ClayTrader: <https://www.youtube.com/watch?v=oNVD0scicGw>

World Events: https://en.wikipedia.org/wiki/List_of_stock_market_crashes_and_bear_markets

Using Markdown in Jupyter: <https://towardsdatascience.com/write-markdown-latex-in-the-jupyter-notebook-10985edb91fd!>
(attachment:image.png)

Appendix

In [3]:

```
# Install the following packages
```

```
install.packages("readxl")  
install.packages("tseries")  
install.packages("TSA")  
install.packages("gdata")  
install.packages("dependency")
```

```

    There is a binary version available but the source version is later:
      binary source needs_compilation
readxl 1.3.1 1.4.0 TRUE

```

```

  Binaries will be installed
package 'readxl' successfully unpacked and MD5 sums checked

```

```
Warning message:
```

```

"cannot remove prior installation of package 'readxl'"Warning message in file.copy(saved
copy, lib, recursive = TRUE):
"problem copying C:\Users\david\anaconda3\Lib\R\library\00LOCK\readxl\libs\x64\readxl.dll
to C:\Users\david\anaconda3\Lib\R\library\readxl\libs\x64\readxl.dll: Permission denie
d"Warning message:
"restored 'readxl'"

```

```

The downloaded binary packages are in
  C:\Users\david\AppData\Local\Temp\RtmpUjNli5\downloaded_packages

```

```

    There is a binary version available but the source version is later:
      binary source needs_compilation
tseries 0.10-48 0.10-50 TRUE

```

```

  Binaries will be installed
package 'tseries' successfully unpacked and MD5 sums checked

```

```
Warning message:
```

```

"cannot remove prior installation of package 'tseries'"Warning message in file.copy(save
dcopy, lib, recursive = TRUE):
"problem copying C:\Users\david\anaconda3\Lib\R\library\00LOCK\tseries\libs\x64\tseries.
dll to C:\Users\david\anaconda3\Lib\R\library\tseries\libs\x64\tseries.dll: Permission d
enied"Warning message:
"restored 'tseries'"

```

```

The downloaded binary packages are in
  C:\Users\david\AppData\Local\Temp\RtmpUjNli5\downloaded_packages

```

```
Warning message:
```

```

"dependency 'locfit' is not available"
package 'TSA' successfully unpacked and MD5 sums checked

```

```

The downloaded binary packages are in
  C:\Users\david\AppData\Local\Temp\RtmpUjNli5\downloaded_packages
package 'gdata' successfully unpacked and MD5 sums checked

```

```

The downloaded binary packages are in
  C:\Users\david\AppData\Local\Temp\RtmpUjNli5\downloaded_packages

```

```
Warning message:
```

```

"package 'dependency' is not available (for R version 3.6.1)"

```

In [4]:

```
# Import the following libraries
```

```

library("readxl")
library("tseries")
library("TSA")
library("gdata")
library("locfit")

```

```
Warning message:
```

```

"package 'readxl' was built under R version 3.6.3"Warning message:
"package 'tseries' was built under R version 3.6.3"Registered S3 method overwritten by
'quantmod':
  method from
as.zoo.data.frame zoo

```

Warning message:

"package 'TSA' was built under R version 3.6.3"

Error: package or namespace load failed for 'TSA' in loadNamespace(j <- i[[1L]], c(lib.loc, .libPaths()), versionCheck = vI[[j]]):

there is no package called 'locfit'

Traceback:

```
1. library("TSA")
2. tryCatch({
  .   attr(package, "LibPath") <- which.lib.loc
  .   ns <- loadNamespace(package, lib.loc)
  .   env <- attachNamespace(ns, pos = pos, deps, exclude, include.only)
  . }, error = function(e) {
  .   P <- if (!is.null(cc <- conditionCall(e)))
  .     paste(" in", deparse(cc)[1L])
  .   else ""
  .   msg <- gettextf("package or namespace load failed for %s%s:\n %s",
  .     sQuote(package), P, conditionMessage(e))
  .   if (logical.return)
  .     message(paste("Error:", msg), domain = NA)
  .   else stop(msg, call. = FALSE, domain = NA)
  . })
3. tryCatchList(expr, classes, parentenv, handlers)
4. tryCatchOne(expr, names, parentenv, handlers[[1L]])
5. value[[3L]](cond)
6. stop(msg, call. = FALSE, domain = NA)
```

In [5]:

```
# Read our VIX dataset
data <- read_excel("./Project_Data.xlsx")
```