Analysis of the EUR/GBP currency pair

Audrey Ekuban

28 May 2016

The euro British pound (EUR/GBP, EURGBP) is a currency pairing whereby the euro and the pound are traded against each other. You short (sell) the euro and go long (buy) the pound or go long (buy) the euro and short (sell) the pound. The EUR/GBP is one of the most traded currency pairs and also, one of the most stable (though that may change). EUR is the base currency. So EURGBP = 0.8 means that 1 EUR is exchanged for 0.8 GBP. If the EURGBP rises to 0.85 then GBP is weakening against EUR.

Types Of Analysis Used In Forex

Forex analysis is used to determine whether to buy or sell a currency pair at any one time. Forex analysis could be technical in nature, using charting tools, or fundamental in nature, using economic indicators and/or news based events. A trader's currency trading system use analysis that create buy or sell decisions when they point in the same direction.

Fundamental Analysis

Fundamental analysis is often used to analyze changes in the forex market by monitoring factors, such as interest rates, unemployment rates, gross domestic product (GDP) and many other economic releases that come out of the countries in question. For example, a trader analyzing the EUR/GBP currency pair fundamentally, would be interested in how a Brexit decision would impact interest rates in the UK. They would also want to be on top of any significant news releases which could indicate that a Brexit decision is likely.

Technical Analysis

Technical analysis is a method of evaluating securities by analyzing statistics generated by market activity, such as past prices and volume. Technical analysts do not attempt to measure a security's intrinsic value, but instead use charts and other tools to identify patterns that can suggest future activity.

The field of technical analysis is based on three assumptions:

1. The market discounts everything.
2. Price moves in trends.
3. History tends to repeat itself.

A technical indicator is a method of chart analysis, usually of price over time or volume, using various formulae designed to highlight specific characteristics and provide signals to

help forecast market movements. Indicators can also be derived from the use of trendlines and price patterns, which are not based on formulae and are more subjective.

Time Series Analysis

Time Series analysis has two main goals: Identifying the nature of a sequence of observations and predicting future values using historical observations (forecasting). In time series analysis it is assumed that the data consists of a systematic pattern and random noise which usually makes the pattern difficult to identify. Most time series analysis techniques use filtering to remove noise.

There are two general components of Time series patterns: Trend and Seasonality. The trend is a linear or non-linear component and it does not repeat within the time range. The Seasonality repeats itself in systematic intervals over time.

Trend analysis is a technique used to identify a trend component in the time series data. In many cases data can be approximated by a linear function, but logarithmic, exponential and polynomial functions can also be used.

Regression Analysis, used in trend analysis, is the study of relationships among variables, and its purpose is to predict or estimate the value of one variable from known values of other variables related to it. Any method of fitting equations to data may be called regression, and these equations are useful for making predictions and judging the strength of relationships.

Forecasting and extrapolation from present values to the future values is not a function of regression analysis. To predict the future time series analysis is used.

The least-squares method is the most common predictive function, and it calculates the minimum average squared deviations between the points and the estimated function. However maximimising Log-Likelihoods and minimising AIcc (Akaike information criterion) is also used.

Analysis Steps

Step 1 - Download the data

I downloaded currency data from www.forexite.com.The website provides free intraday data for 16 currencies. The data goes back to 2001. The R script below was used to download and unzip the data files.

```
# Download currency data from forexite.com

getFiles <- function(URL) {
  zipfile = unlist(strsplit(URL, "/")[[1]])[7]
  txtfile = replace(zipfile, '.zip', '.txt')
  download.file(URL, zipfile)
  data <- read.table(unzip(zipfile))
```

```
}

#setwd("RawData")

# Download all data that exists between 3rd January 2001 and 14th May 2016
inclusive
downloadDates = format(seq(as.Date("2001/01/03"), as.Date("2016/05/14"),
"days"),
                       format="%Y/%m/%d%m%y")
# Set download URL
downloadURL = paste('http://www.forexite.com/free_forex_quotes/',
downloadDates, '.zip', sep='')

# Start Download
for (i in downloadURL) {
  print(i)
  tryCatch({
    getFiles(i)
  }, error=function(e){})
}
```

Step 2 - Extract the EURGBP data and merge into one file

The downloaded files contain a number of currency pairs.Since I was only interested in the EUR / GBP I needed to extract the lines that started with EURGBP. I did so using the script below.

```
library(dplyr)

setwd("RawData")

# We are only interested in *.txt files
myFiles <- list.files(pattern = ".txt")

# Create empty dataframe
df <- data.frame(V2 = (character()),
                 V3 = character(),
                 V4 = character(),
                 V5 = character(),
                 V6 = character(),
                 V7 = character()

)

# Read tge data from the files into the dataframe
for (f in myFiles) {
  tryCatch({
    if (file.size(f) > 0){
      df2 = read.csv(f, skip=1, header=FALSE, colClasses=c('character'))
```

```
      df2 = filter(df2, V1 == 'EURGBP') %>% select(-V1)
      df = rbind_list(df, df2)
    }
  }, error = function(err) {
    print(paste(f, ' : ', err))
  })
}

write.table(df, '../Data/EURGBP.csv', row.names = FALSE, col.names =
c('DATE', 'TIME', 'OPEN', 'HIGH', 'LOW', 'CLOSE'), quote = FALSE, sep=",")
```

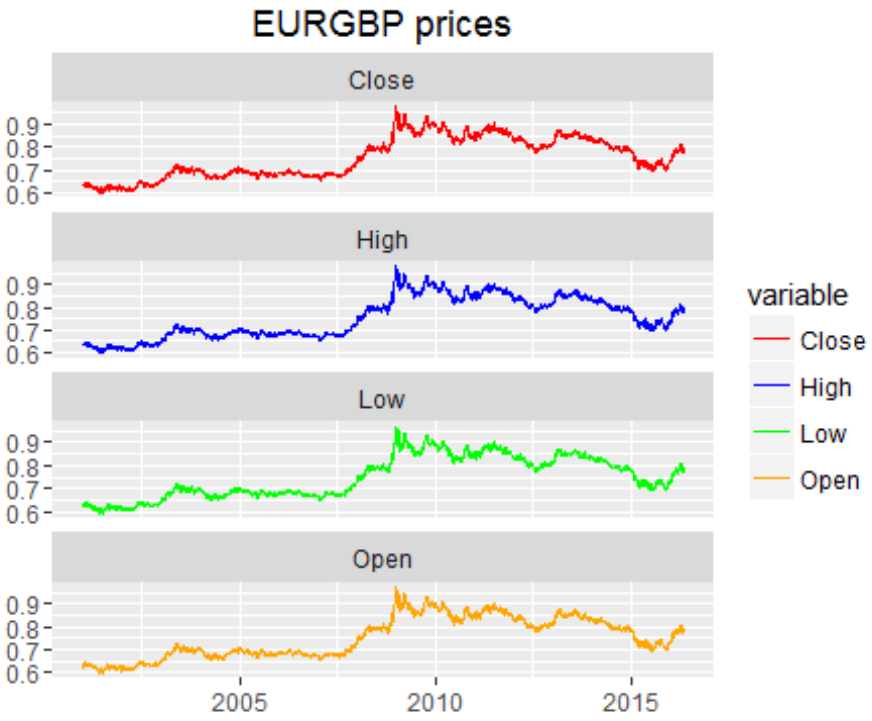
Step 3 - Data Preparation

The data in EURGBP.csv contained a separate DATE and TIMe column. The data is a weakly regular time series i.e. there was some missing data.However the missing data was due to the FOREX market being closed so in reality the data wasn't missing. Converting the data to a Time Series object in R created a strictly regular time series and this is what was used in the analysis.

The data shows that during the 2007/2008 Financial crisis GBP was at is weakest point against EUR.

```
setwd("Data")

#The below produces an error
#install.packages('ggfortify')

#install.packages("devtools")
#library(devtools)

#install_github('sinhrks/ggfortify')
library(ggfortify)

## Loading required package: ggplot2

#install.packages('xts')
library(xts)

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

#install.packages('forecast')
library(forecast)

## Loading required package: timeDate
```

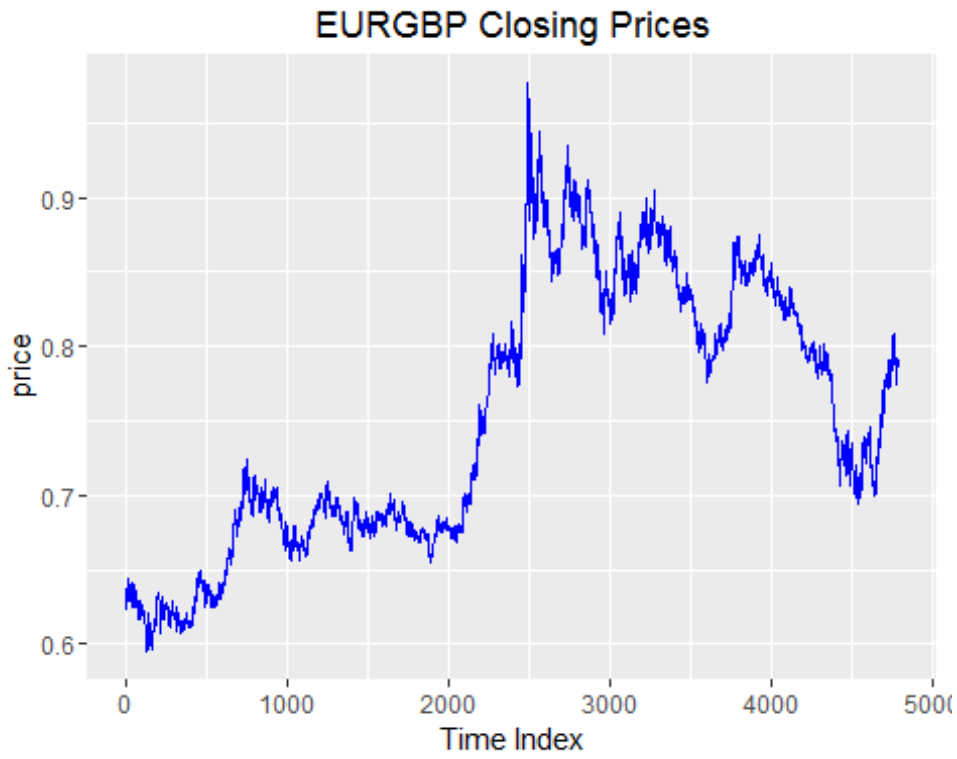
```
## This is forecast 7.1

# read the data
EURGBP = read.csv("EURGBP.csv", colClasses=c(rep("character", 2),
rep("numeric", 4)))
datetime <- as.POSIXct(paste(EURGBP$DATE, EURGBP$TIME), format = "%Y%m%d
%H%M%OS")
EURGBP = EURGBP[c(3:6)]
summary(EURGBP)

##       OPEN             HIGH             LOW              CLOSE
##  Min.   :0.5954   Min.   :0.5954   Min.   :0.5954   Min.   :0.5954
##  1st Qu.:0.6812   1st Qu.:0.6812   1st Qu.:0.6812   1st Qu.:0.6812
##  Median :0.7569   Median :0.7570   Median :0.7569   Median :0.7570
##  Mean   :0.7586   Mean   :0.7586   Mean   :0.7585   Mean   :0.7586
##  3rd Qu.:0.8387   3rd Qu.:0.8388   3rd Qu.:0.8387   3rd Qu.:0.8387
##  Max.   :0.9801   Max.   :0.9801   Max.   :0.9796   Max.   :0.9800

# Make sure ordering is correct and convert to daily
EURGBP.xt <- xts(x = EURGBP, order.by = datetime)
EURGBP.xt.daily = to.daily(EURGBP.xt)

names(EURGBP.xt.daily) = c('Open', 'High', 'Low', 'Close')
head(EURGBP.xt.daily)

##              Open   High    Low  Close
## 2001-01-03 0.6328 0.6353 0.6212 0.6222
## 2001-01-04 0.6222 0.6340 0.6220 0.6328
## 2001-01-05 0.6328 0.6376 0.6323 0.6374
## 2001-01-07 0.6367 0.6371 0.6367 0.6371
## 2001-01-08 0.6371 0.6374 0.6314 0.6325
## 2001-01-09 0.6325 0.6342 0.6299 0.6337

tail(EURGBP.xt.daily)

##              Open   High    Low  Close
## 2016-05-08 0.7904 0.7904 0.7898 0.7901
## 2016-05-09 0.7901 0.7926 0.7866 0.7895
## 2016-05-10 0.7896 0.7902 0.7869 0.7870
## 2016-05-11 0.7870 0.7918 0.7864 0.7908
## 2016-05-12 0.7907 0.7917 0.7844 0.7874
## 2016-05-13 0.7875 0.7881 0.7857 0.7870

# Plot the multi-variate time series
pallet = c('red', 'blue', 'green', 'orange')
autoplot(EURGBP.xt.daily, ts.colour = 'variable', main = 'EURGBP prices') +
scale_colour_manual(values=pallet)
```




```
# Check if data is weakly or strictly regular
is.regular(EURGBP.xt.daily)

## [1] TRUE

is.regular(EURGBP.xt.daily, strict = TRUE)

## [1] FALSE

# Convert data to Time Series for analysis
EURGBP.ts.daily = ts(EURGBP.xt.daily, names=c('Open', 'High', 'Low',
'Close'))
is.regular(EURGBP.ts.daily, strict = TRUE)

## [1] TRUE

head(EURGBP.ts.daily)

##        Open   High    Low  Close
## [1,] 0.6328 0.6353 0.6212 0.6222
## [2,] 0.6222 0.6340 0.6220 0.6328
## [3,] 0.6328 0.6376 0.6323 0.6374
## [4,] 0.6367 0.6371 0.6367 0.6371
## [5,] 0.6371 0.6374 0.6314 0.6325
## [6,] 0.6325 0.6342 0.6299 0.6337

tail(EURGBP.ts.daily)
```

```
##           Open   High    Low  Close
## [4783,] 0.7904 0.7904 0.7898 0.7901
## [4784,] 0.7901 0.7926 0.7866 0.7895
## [4785,] 0.7896 0.7902 0.7869 0.7870
## [4786,] 0.7870 0.7918 0.7864 0.7908
## [4787,] 0.7907 0.7917 0.7844 0.7874
## [4788,] 0.7875 0.7881 0.7857 0.7870
```

```
# Focus on closing prices
EURGBP.price = EURGBP.ts.daily[,'Close']

autoplot(as.zoo(EURGBP.price), ts.colour = 'blue') +
  ggtitle('EURGBP Closing Prices') + xlab('Time Index') + ylab('price')
```




```
# Split the time series into Training and Testing
EURGBP.training = window(EURGBP.price, 1, 3000)
EURGBP.test = window(EURGBP.price, 3000)
```

Step 4 - Plot Data and examine

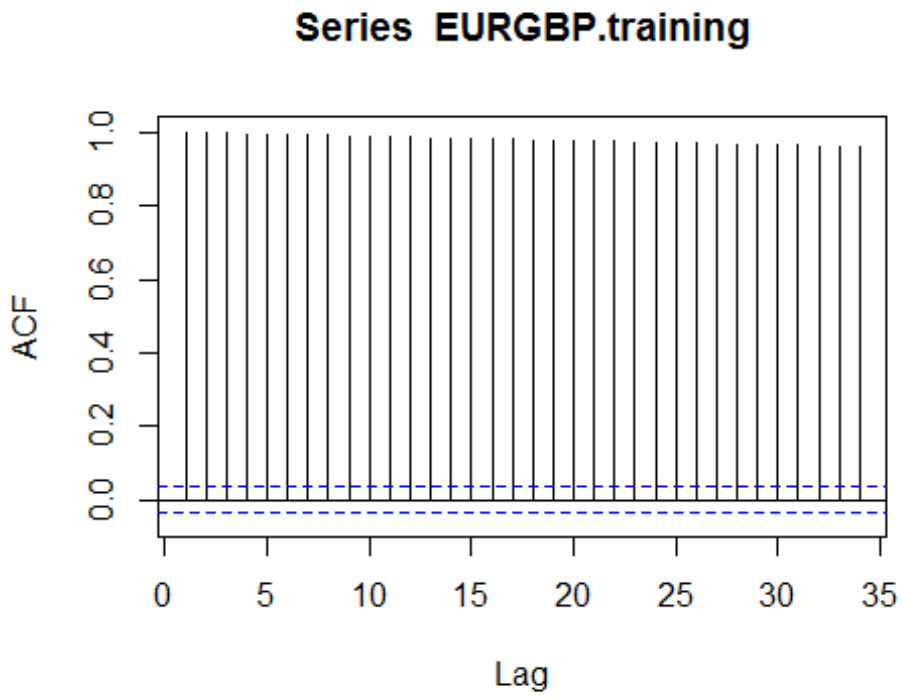
A stationary time series is one whose statistical properties such as mean, variance, autocorrelation, etc. are all constant over time. The stationarized series can be prediced as the properties will be the same in the future as they have been in the past.

The Autocorrelation Function (ACF) and the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test was used to check for stationarity.

Based on the KPSS p-value stated below, the hypothesis of stationarity is rejected.In addition, the ACF plot shows a slow linear decay, indicative of a non-stationary time series.

```
autoplot(as.zoo(EURGBP.training), ts.colour = 'blue') +
  ggtitle('Training Data') + xlab('Time Index') + ylab('Closing Price')
```


```
Acf(EURGBP.training)
```




```
tseries::kpss.test(EURGBP.training)
```

```
## Warning in tseries::kpss.test(EURGBP.training): p-value smaller than
## printed p-value
```

```
##
##  KPSS Test for Level Stationarity
##
## data:  EURGBP.training
## KPSS Level = 16.974, Truncation lag parameter = 12, p-value = 0.01
```

```
# Check - How many differences do we need to take
ndiffs(EURGBP.training)
```

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

```
#The below produces an error as the data is not seasonal
#nsdiffs(EURGBP.training)
```

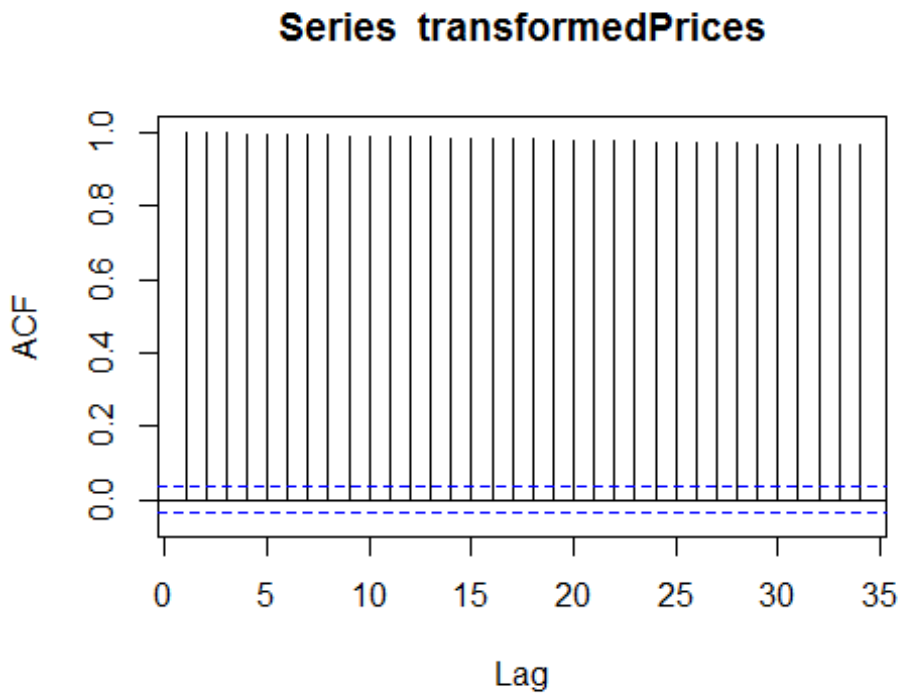
Step 5 - Transform the data and examine

Based on the KPSS p-value stated below, the hypothesthat that the time series is stationarity is rejected.In addition, the ACF plot shows a slow linear decay, indicative of a non-stationary time series.

```
# Use BoxCox to transform the data
lambda = forecast::BoxCox.lambda(EURGBP.training,lower=-2)
lambda
```

```
## [1] -1.613177

transformedPrices = forecast::BoxCox(EURGBP.training, lambda)

# Train the data
autoplot(as.zoo(transformedPrices), ts.colour = 'blue') +
  ggtitle('Training Data') + xlab('Time Index') + ylab('transformed Closing
Price')
```


```
Acf(transformedPrices)
```



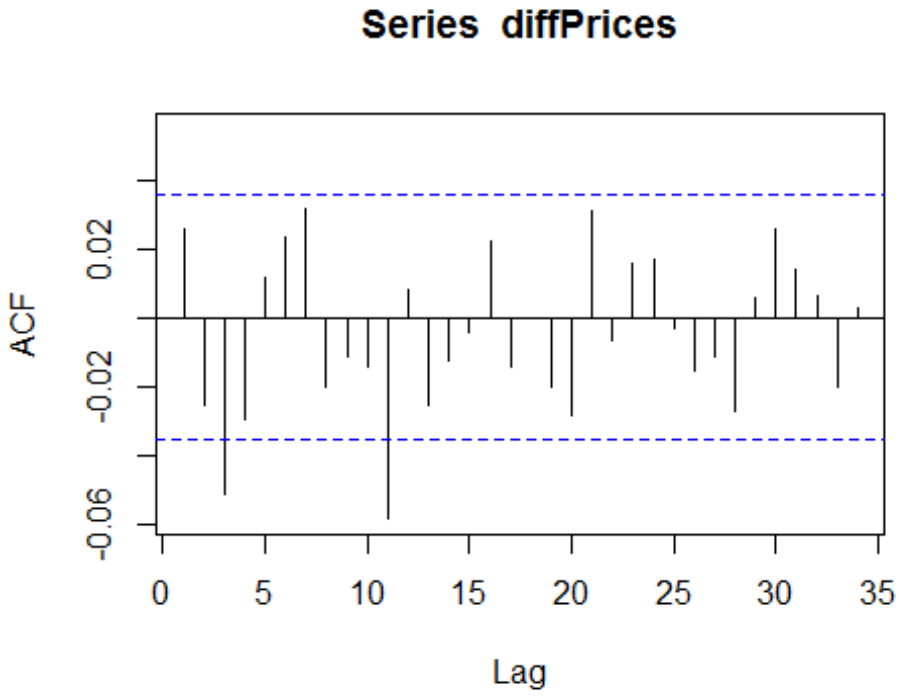

```
tseries::kpss.test(transformedPrices)

## Warning in tseries::kpss.test(transformedPrices): p-value smaller than
## printed p-value

##
##  KPSS Test for Level Stationarity
##
## data:  transformedPrices
## KPSS Level = 17.857, Truncation lag parameter = 12, p-value = 0.01

# Check - How many differences do we need to take
ndiffs(transformedPrices)

## [1] 1

#The below produces an error as the data is not seasonal
#nsdiffs(transformedPrices)
```

Step 6 - Take 1st differences of the transformed data

Based on the KPSS p-value stated below, the hypothesthat that the time series is stationarity is not rejected.In addition, The ACF plot also indicates a stationary time series.

```
diffPrices = diff(transformedPrices)
autoplot(as.zoo(diffPrices), ts.colour = 'blue') +
  ggtitle('First Difference Training Data') + xlab('Time Index') +
ylab('transformed Closing Price')
```


```
Acf(diffPrices)
```




```
tseries::kpss.test(diffPrices)
```

```
## Warning in tseries::kpss.test(diffPrices): p-value greater than printed p-
## value

##
##  KPSS Test for Level Stationarity
##
## data:  diffPrices
## KPSS Level = 0.056454, Truncation lag parameter = 12, p-value =
## 0.1
```

```
summary(diffPrices)
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.0323000 -0.0037580  0.0000000  0.0001634  0.0040120  0.0519400
```

Step 7 - Model Identification and Estimation

The Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) of the stationary series now need to be examined o estimate the ARIMA(p,d,q) model.

It should be noted that we can use auto.arima in R to give an estimate of the model.

The d in ARIMA(p,d,q) stands for the number of times the data had to be differenced to become to stationary. In this case d = 1.

The p in ARIMA(p,d,q) measures the order of the autoregressive component. The time-series has an autoregressive order of 1, called AR(1), then we should see only the first partial autocorrelation coefficient as significant. However this is not the case

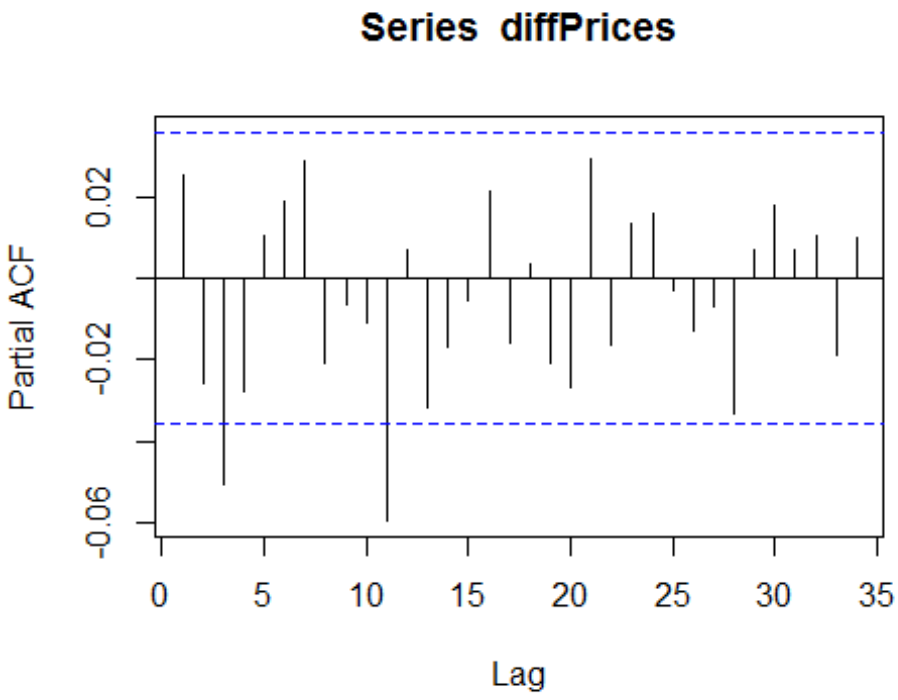
The q in ARIMA(p,d,q) is the number of lagged forecast errors in the prediction equation. This is usually referred to as the Moving Average (MA) term.

Generally, PACF will have significant correlations up to lag p, and will quickly drop to near zero values after lag p. Further work is needed.

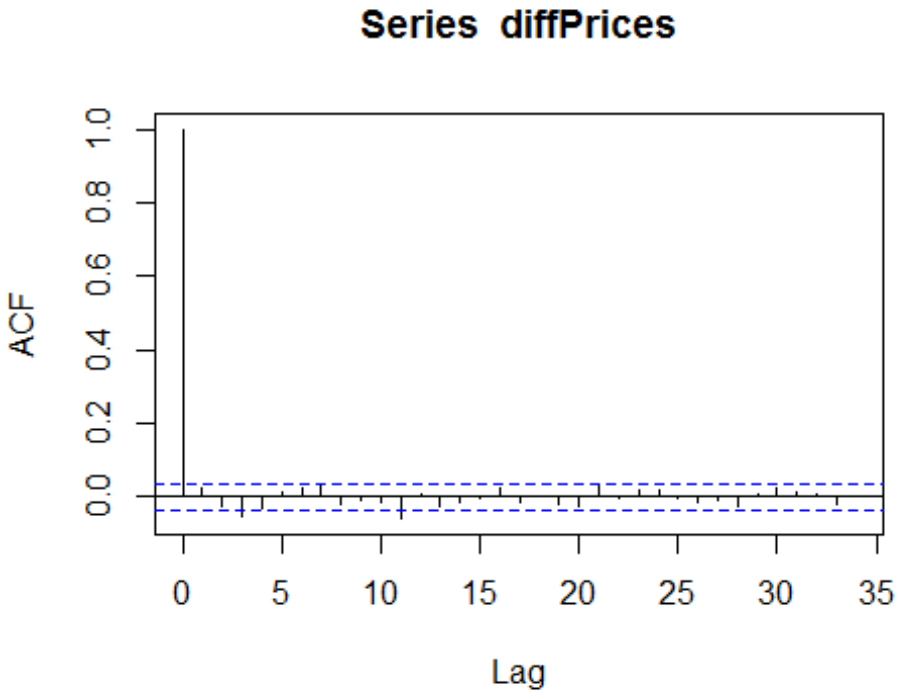
```
auto.arima(transformedPrices)
```

```
## Series: transformedPrices
## ARIMA(2,1,2)
##
## Coefficients:
##          ar1      ar2      ma1     ma2
##       1.3409  -0.9536  -1.3392  0.9350
## s.e.  0.0170   0.0284   0.0206  0.0315
##
## sigma^2 estimated as 5.744e-05:  log likelihood=10388.7
## AIC=-20767.4   AICc=-20767.38   BIC=-20737.37
```

```
pacf(diffPrices)
```



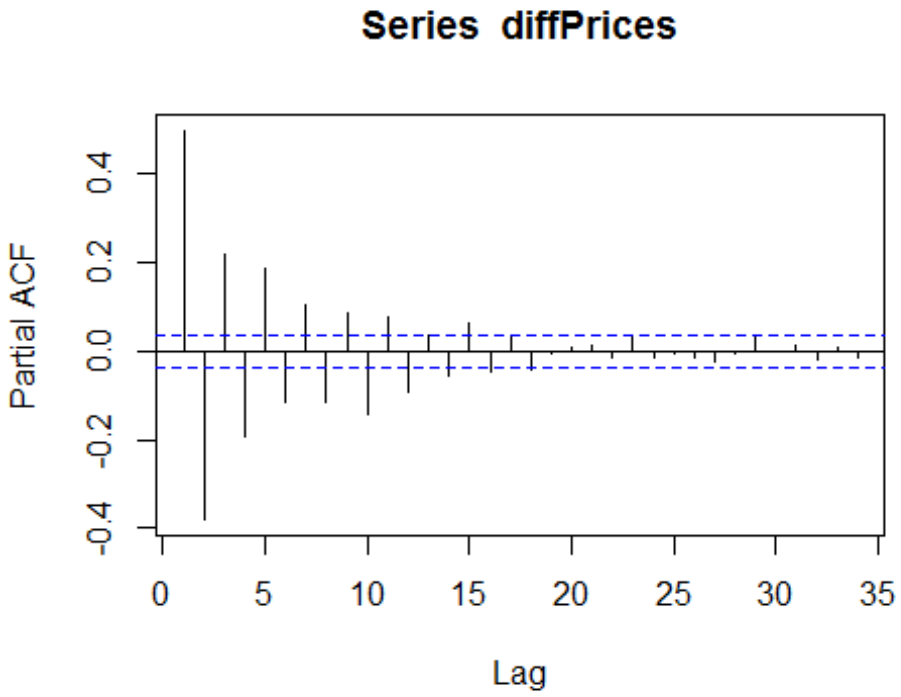

```r
# Use acf rather tha Acf
acf(diffPrices)
```



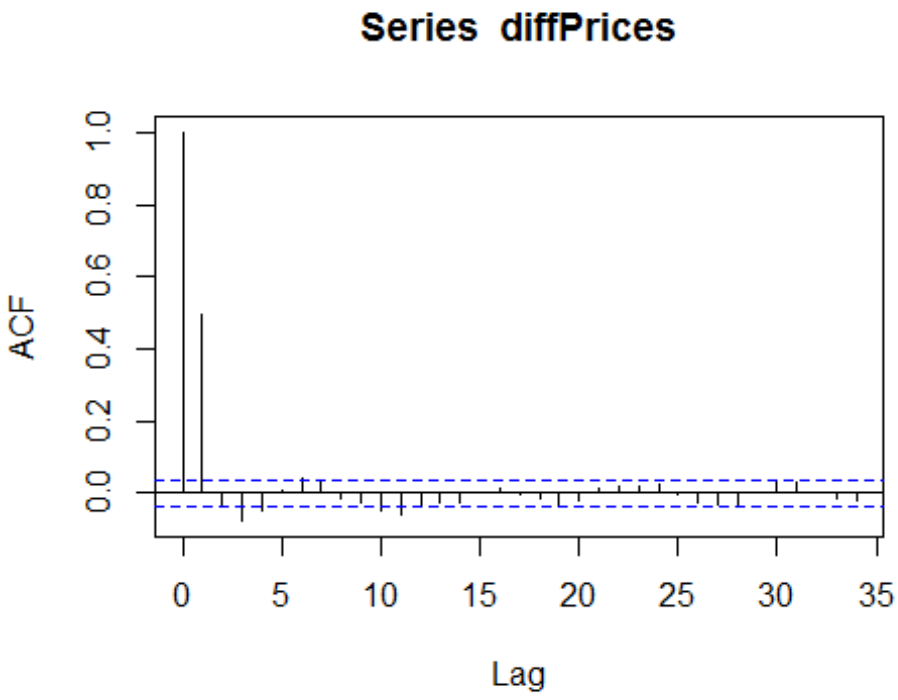
Increase lags in differenced data

The above differened data uses a lag of 1 i.e. the difference is taken of 2 consecutive data points. For a log of 2, the differences between the third and the first value, between the fourth and the second value, between the fifth and the third value etc. are calculated.

```
diffPrices = diff(transformedPrices, lag = 2, diff = 1)

pacf(diffPrices)
```



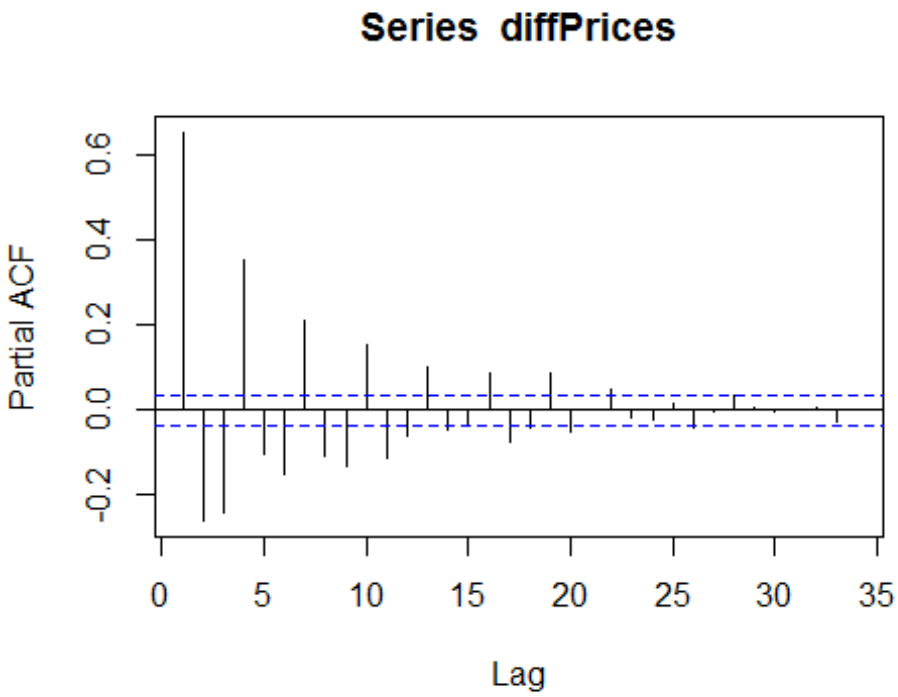

```
# Use acf rather tha Acf
acf(diffPrices)
```



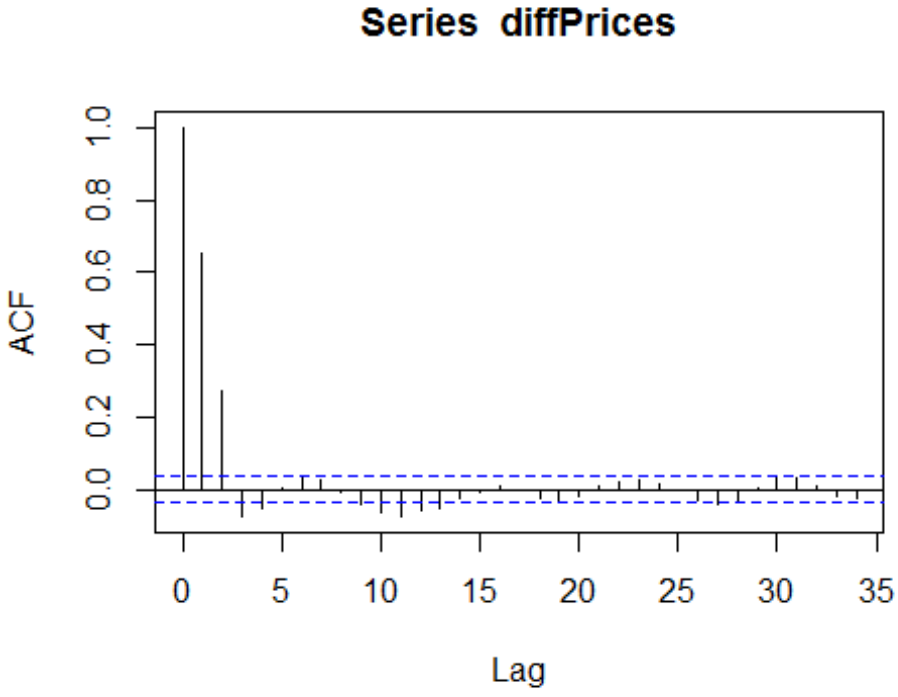

```
# Increase maximum order
auto.arima(transformedPrices, approximation=FALSE, stepwise=FALSE,
max.order=15)

## Series: transformedPrices
## ARIMA(3,1,2)
##
## Coefficients:
##          ar1      ar2     ar3      ma1     ma2
##       1.3705  -1.0046  0.0196  -1.3488  0.9618
## s.e.  0.0249   0.0306  0.0189   0.0168  0.0203
##
## sigma^2 estimated as 5.742e-05:  log likelihood=10389.91
## AIC=-20767.83   AICc=-20767.8   BIC=-20731.79
```

The auto.arima results above indicate that ARIMA(p = 3, d = 1, q = 2) might be a best fit (it has lowest (AICc value). Try one more lag and calculate various ARIMA's to verify that this is feasible.

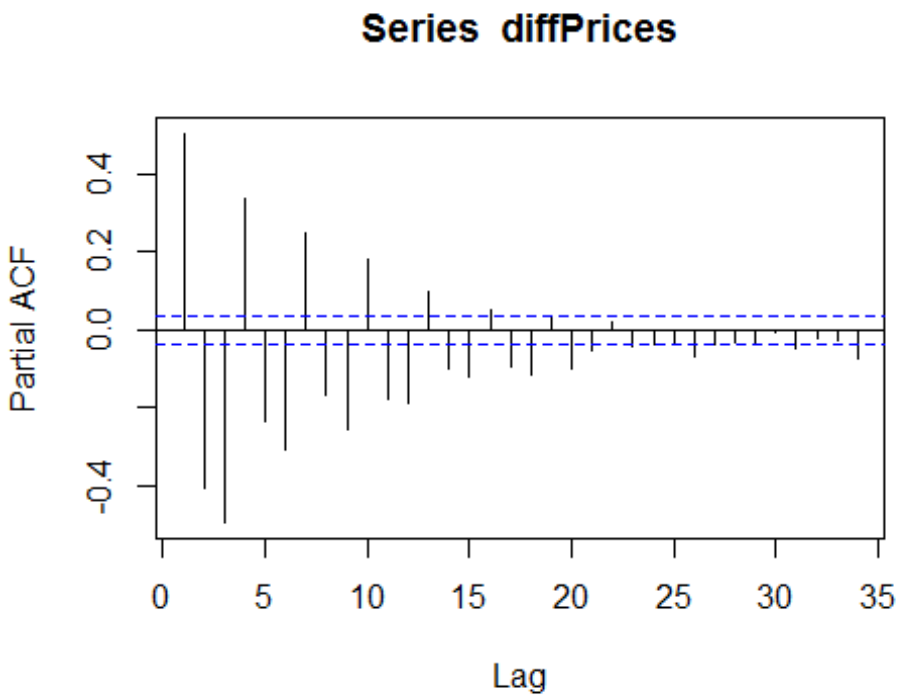
```
diffPrices = diff(transformedPrices, lag = 3, diff = 1)

pacf(diffPrices)
```



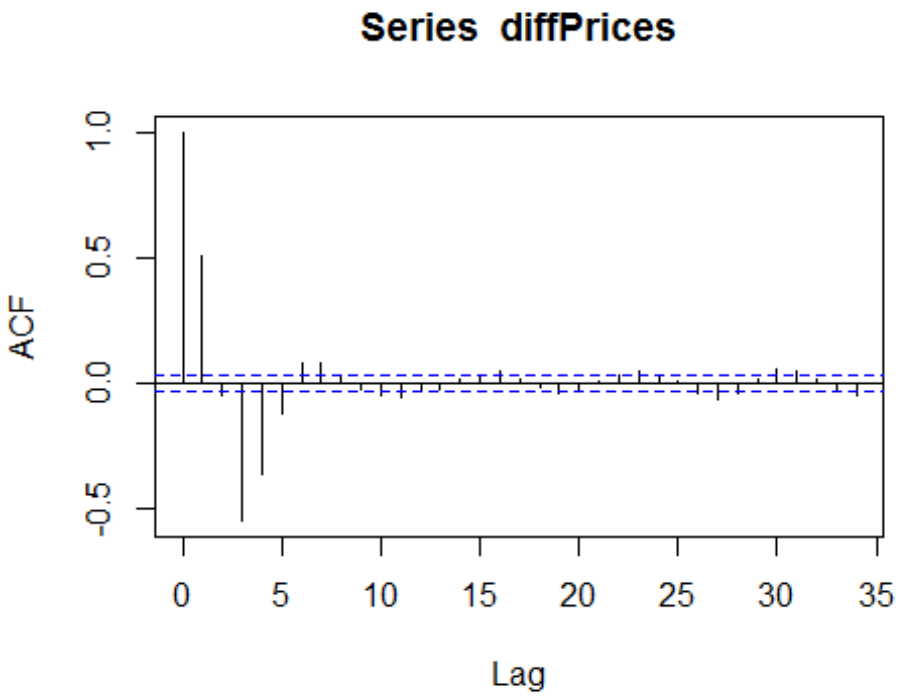

```r
# Use acf rather tha Acf
acf(diffPrices)
```



```r
diffPrices = diff(transformedPrices, lag = 3, diff = 2)

pacf(diffPrices)
```




```r
# Use acf rather tha Acf
acf(diffPrices)
```




```
AIC(Arima(transformedPrices, c(3,1,2)),
    Arima(transformedPrices, c(4,1,3)),
    Arima(transformedPrices, c(5,1,4)))
```

```
##                                       df       AIC
## Arima(transformedPrices, c(3, 1, 2))  6 -20767.83
## Arima(transformedPrices, c(4, 1, 3))  8 -20756.21
## Arima(transformedPrices, c(5, 1, 4)) 10 -20769.68
```

It should be noted that the plots indicate a very high-ordered model. However, it is possible for an AR term and an MA term to cancel each other's effects, even though both may appear significant in a model.

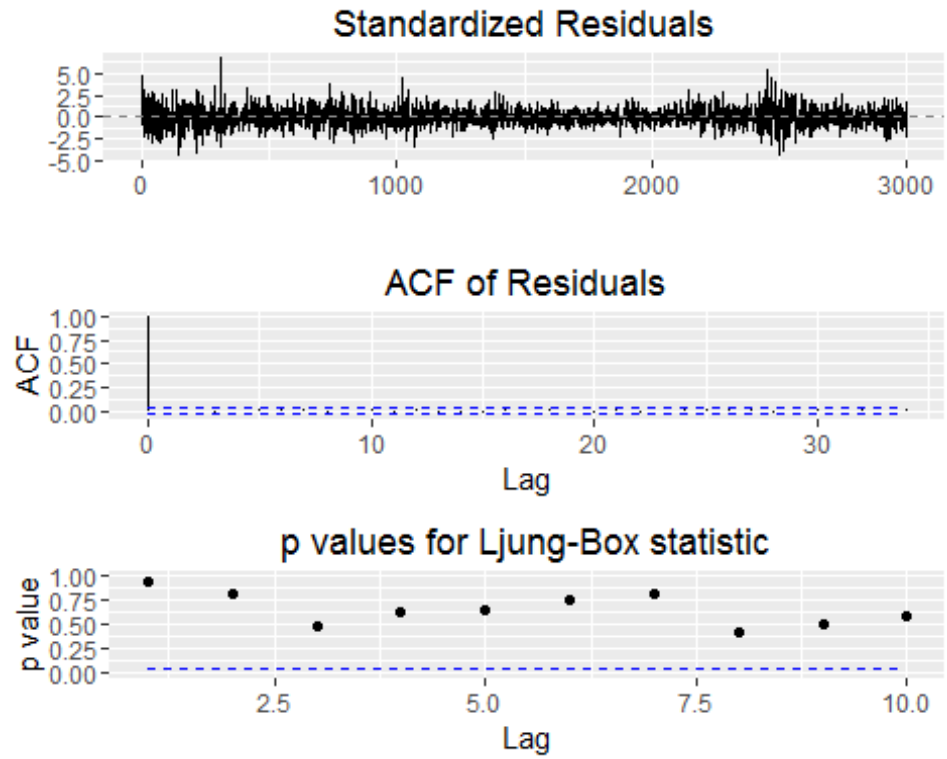
Step 7 - Forecasting

Apply one step forecasting of the model to the Test set. The test data set is modelled extremely well and errors approximately match the errors in the Training set.

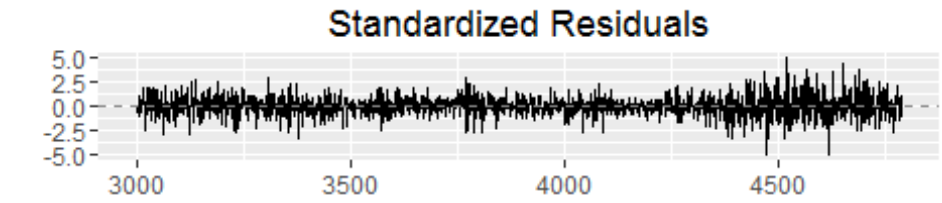
However using the model to predict if EUR value will increase or decrease would be incorrect. That is for another Capstone. But I was curious. Taking lags into consideration, it can be seen that the model will correctly predict the direction of the price in approximately 73% of cases. Not good enough to risk your cash.

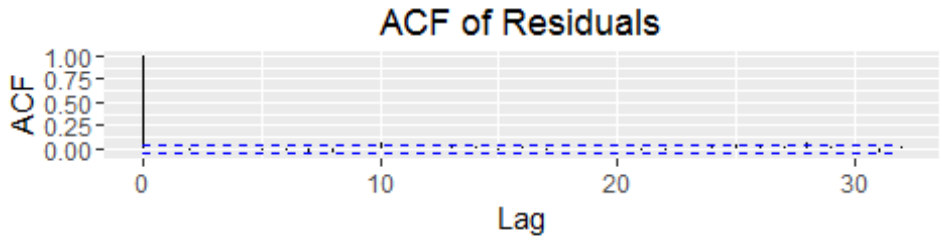
```
testTransformedPrices = forecast::BoxCox(EURGBP.test, lambda)

# Model
```

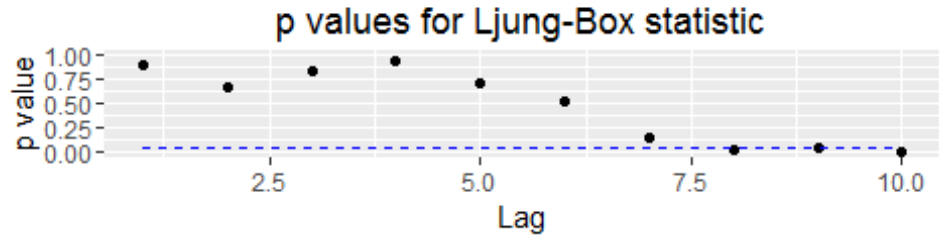
```
transformedPrices.fit <- Arima(transformedPrices, c(3,1,2))
ggtsdiag(transformedPrices.fit)
```



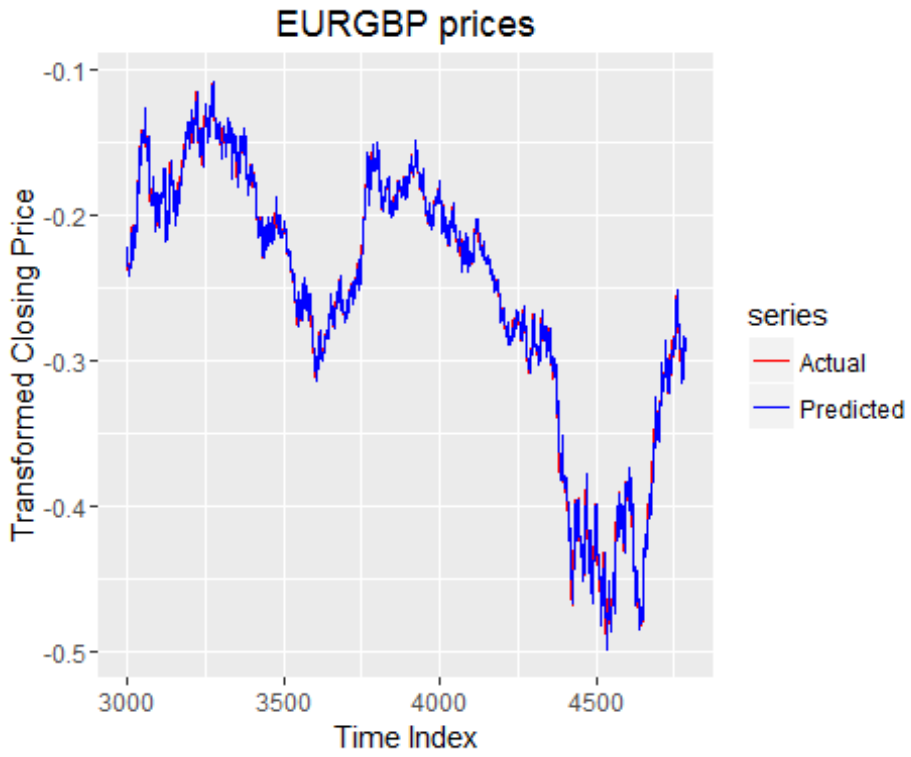

```
summary(transformedPrices.fit)

## Series: transformedPrices
## ARIMA(3,1,2)
##
## Coefficients:
##          ar1      ar2     ar3      ma1     ma2
##       1.3705  -1.0046  0.0196  -1.3488  0.9618
## s.e.  0.0249   0.0306  0.0189   0.0168  0.0203
##
## sigma^2 estimated as 5.742e-05:  log likelihood=10389.91
## AIC=-20767.83   AICc=-20767.8   BIC=-20731.79
##
## Training set error measures:
##                       ME        RMSE         MAE        MPE     MAPE
## Training set 0.0001632831 0.007569703 0.005454472 -0.1033683 1.741767
##                    MASE         ACF1
## Training set 0.9981741 -0.001260125

# Apply model to test
testTransformedPrices.fit = Arima(testTransformedPrices,
model=transformedPrices.fit)
ggtsdiag(testTransformedPrices.fit)
```








```
accuracy(fitted(testTransformedPrices.fit), testTransformedPrices)

##                       ME        RMSE         MAE         MPE     MAPE
## Test set -3.920594e-05 0.006561965 0.004697599 -0.02400249 1.988625
##                  ACF1 Theil's U
## Test set 0.003043687  1.002799

EURGBP.transformed = cbind(testTransformedPrices,
fitted(testTransformedPrices.fit))
colnames(EURGBP.transformed) = c('Actual', 'Predicted')

pallet = c('red', 'blue')
autoplot(EURGBP.transformed, ts.colour = 'variable', main = 'EURGBP prices')
+
  scale_colour_manual(values=pallet) +
  xlab('Time Index') + ylab('Transformed Closing Price')
```

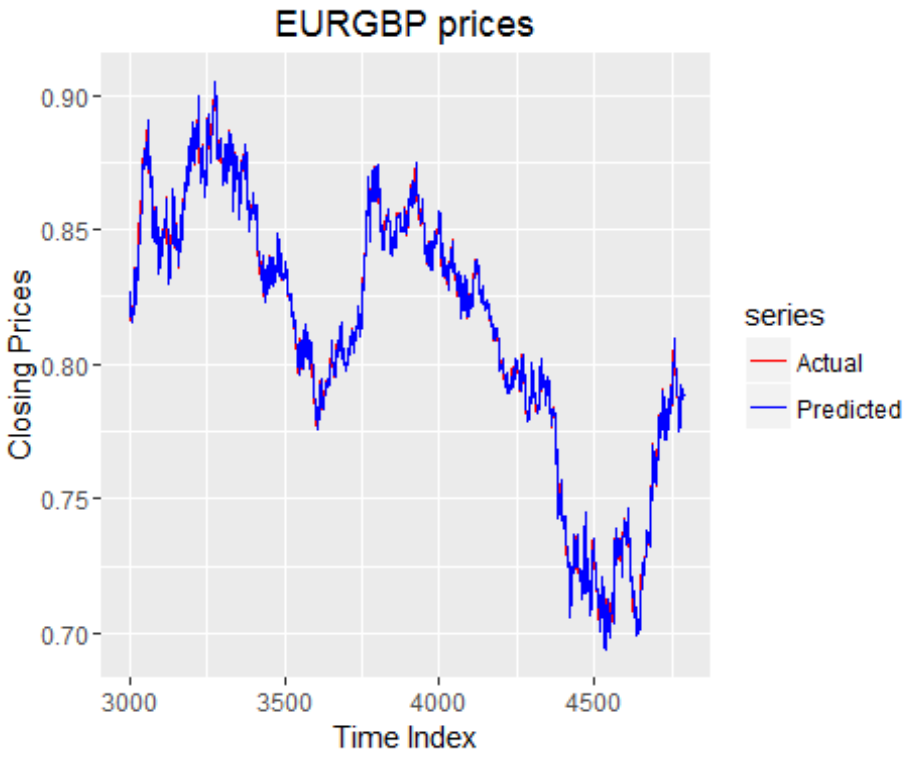



```
# Long Training
actualDirection = sapply(transformedPrices, function(x) diff(x, 3, 1) > 0)
predictedDirection = sapply(fitted(transformedPrices.fit), function(x)
diff(x,3,1) > 0)
table(actualDirection, predictedDirection)

##                 predictedDirection
## actualDirection FALSE TRUE
##           FALSE  1099  407
##           TRUE    393 1098

#Short Training
actualDirection = sapply(transformedPrices, function(x) diff(x, 3, 1) < 0)
predictedDirection = sapply(fitted(transformedPrices.fit), function(x)
diff(x, 3, 1) < 0)
table(actualDirection, predictedDirection)

##                 predictedDirection
## actualDirection FALSE TRUE
##           FALSE  1106  415
##           TRUE    399 1077

# Long Test
actualDirection = sapply(testTransformedPrices, function(x) diff(x, 3, 1) >
0)
predictedDirection = sapply(fitted(testTransformedPrices.fit), function(x)
```

```
diff(x,3,1) > 0)
table(actualDirection, predictedDirection)

##                predictedDirection
## actualDirection FALSE TRUE
##           FALSE   669  240
##           TRUE    235  642

#Short Test
actualDirection = sapply(testTransformedPrices, function(x) diff(x, 3, 1) <
0)
predictedDirection = sapply(fitted(testTransformedPrices.fit), function(x)
diff(x, 3, 1) < 0)
table(actualDirection, predictedDirection)

##                predictedDirection
## actualDirection FALSE TRUE
##           FALSE   651  245
##           TRUE    231  659
```

Verify untransformed data

```
# Untransform the fitted values
fittedPrices = forecast::InvBoxCox(fitted(testTransformedPrices.fit), lambda)

EURGBP.test.prices = cbind(EURGBP.test, fittedPrices)
colnames(EURGBP.test.prices) = c('Actual', 'Predicted')

pallet = c('red', 'blue')
autoplot(EURGBP.test.prices, ts.colour = 'variable', main = 'EURGBP prices')
+
  scale_colour_manual(values=pallet) +
  xlab('Time Index') + ylab('Closing Prices')
```


EURGBP prices
Closing Prices
0.90
0.85
0.80
0.75
0.70
3000
3500
4000
4500
Time Index
series
Actual
Predicted