ARIMA Model EUR And JPY

Jane

29/04/2021

# Forcasting Exchange Rate Using ARIMA Model for EUR And US Dollar

## Reading EUR and EUR Currency into r

library(readr)  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

EURJPYARIMA <- read.csv ("EURJPY\_Candlestick\_1\_D\_BID\_01.01.2000-31.12.2020.csv")%>%  
 select('GMT.TIME', CLOSE)%>%  
 rename(Date = ('GMT.TIME'), RateEURJPY = ("CLOSE"))  
  
   
head(EURJPYARIMA)

## Date RateEURJPY  
## 1 2000-01-03 104.00  
## 2 2000-01-04 106.28  
## 3 2000-01-05 107.57  
## 4 2000-01-06 108.68  
## 5 2000-01-07 108.42  
## 6 2000-01-10 107.78

## Conversion of Gmt time to date format

library(dplyr)  
library(lubridate)

##   
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':  
##   
## date, intersect, setdiff, union

EURJPYARIMA$Date <- lubridate::ymd(EURJPYARIMA$Date)  
head(EURJPYARIMA)

## Date RateEURJPY  
## 1 2000-01-03 104.00  
## 2 2000-01-04 106.28  
## 3 2000-01-05 107.57  
## 4 2000-01-06 108.68  
## 5 2000-01-07 108.42  
## 6 2000-01-10 107.78

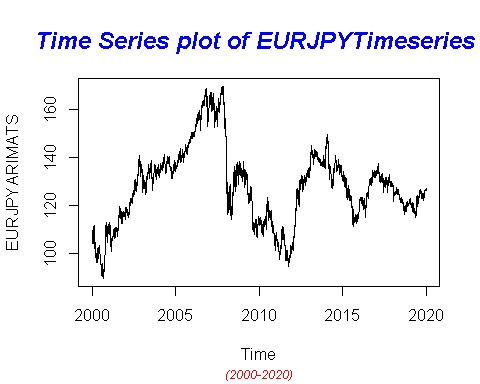
##Checking for obvious errors or missingg value

#Checking for obvious errors  
which(is.na(EURJPYARIMA))

## integer(0)

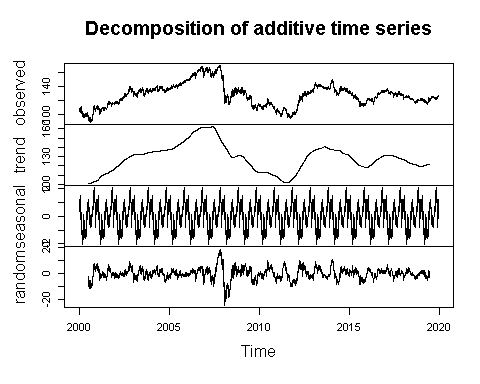
##Converting the data set into time series object

#Converting the data set into time series object  
EURJPYARIMATS<- ts(as.vector(EURJPYARIMA$Rate), frequency = 320, start= c(2000,01,03))  
plot.ts(EURJPYARIMATS)  
title("Time Series plot of EURJPYTimeseries ", sub = "(2000-2020)",  
 cex.main = 1.5, font.main= 4, col.main= "blue",  
 cex.sub = 0.75, font.sub = 3, col.sub = "red")



## Finding the component of the Time Series

ComponentEURJPY <- decompose(EURJPYARIMATS)  
plot(ComponentEURJPY)



## To To achieve stationarity by differencing the data – compute the differences between consecutive observations

library("fUnitRoots")

## Warning: package 'fUnitRoots' was built under R version 4.0.5

## Loading required package: timeDate

## Warning: package 'timeDate' was built under R version 4.0.4

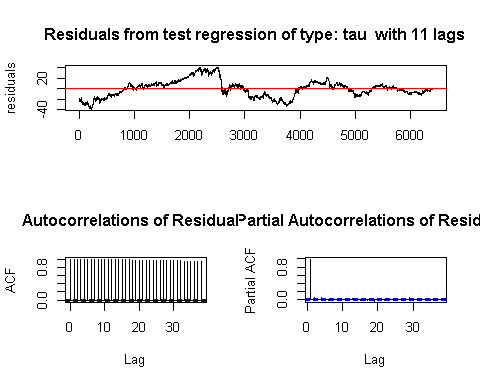
## Loading required package: timeSeries

## Warning: package 'timeSeries' was built under R version 4.0.5

## Loading required package: fBasics

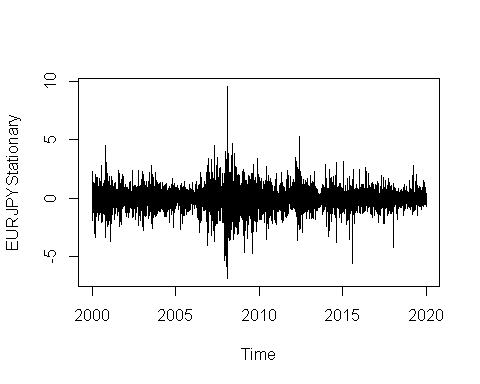
## Warning: package 'fBasics' was built under R version 4.0.5

urkpssTest(EURJPYARIMATS, type = c("tau"), lags = c("short"),use.lag = NULL, doplot = TRUE)



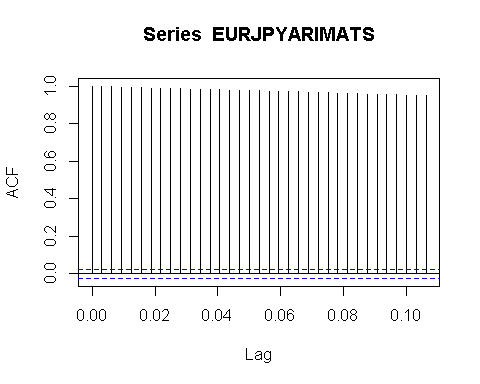
##   
## Title:  
## KPSS Unit Root Test  
##   
## Test Results:  
## NA  
##   
## Description:  
## Thu May 06 09:51:42 2021 by user: janeo

EURJPYStationary= diff(EURJPYARIMATS, differences=1)  
plot(EURJPYStationary)

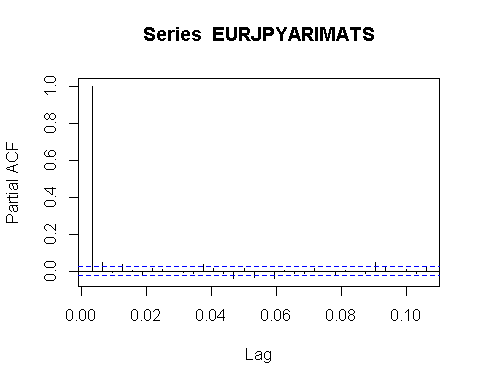


## Calculating Autocorrlation function and partil autocorlation function

acf(EURJPYARIMATS,lag.max=34)

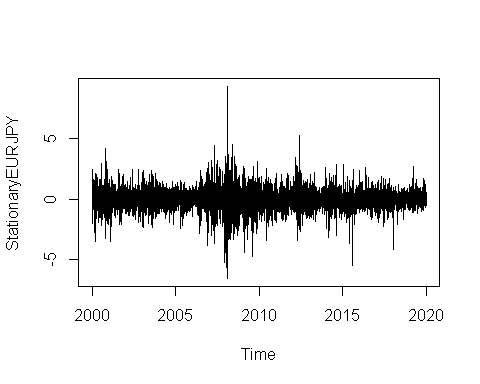


pacf(EURJPYARIMATS, lag.max = 34)



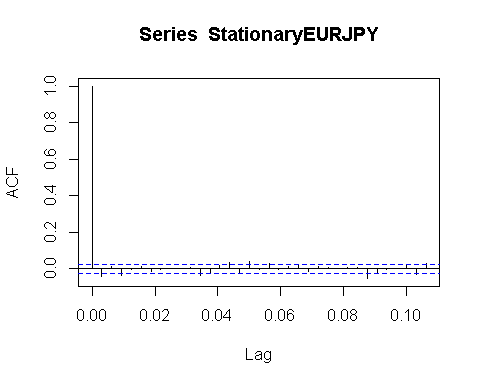
## Adjusting and ensuring there are no seasonality

TSseasonallyadjustedEURJPY <- EURJPYARIMATS- ComponentEURJPY$seasonal   
StationaryEURJPY <- diff(TSseasonallyadjustedEURJPY, differences=1)  
plot(StationaryEURJPY)

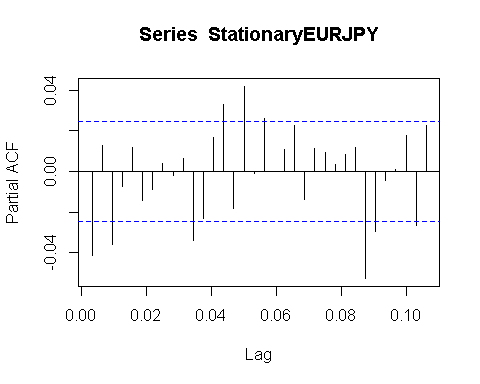


## Calculating again for ACF and PACF after finding stationality

acf(StationaryEURJPY, lag.max=34)



pacf(StationaryEURJPY, lag.max=34)



# Fitting The ARIMA Model

## ARIMA fitting (1,0,4)

fitArima1EURJPY <- arima(EURJPYARIMATS, order = c(1,0,4), include.mean = TRUE)  
fitArima1EURJPY

##   
## Call:  
## arima(x = EURJPYARIMATS, order = c(1, 0, 4), include.mean = TRUE)  
##   
## Coefficients:  
## ar1 ma1 ma2 ma3 ma4 intercept  
## 0.9987 -0.0396 0.0131 -0.0391 -0.0031 127.7247  
## s.e. 0.0006 0.0125 0.0125 0.0127 0.0126 7.2754  
##   
## sigma^2 estimated as 0.7531: log likelihood = -8175.53, aic = 16365.07

##Arima Fitting (1,0,3)

fitArima2EURJPY <- arima(EURJPYARIMATS, order = c(1,0,3), include.mean = TRUE)  
fitArima2EURJPY

##   
## Call:  
## arima(x = EURJPYARIMATS, order = c(1, 0, 3), include.mean = TRUE)  
##   
## Coefficients:  
## ar1 ma1 ma2 ma3 intercept  
## 0.9987 -0.0398 0.0133 -0.0391 127.7816  
## s.e. 0.0006 0.0125 0.0126 0.0127 7.2699  
##   
## sigma^2 estimated as 0.7531: log likelihood = -8175.57, aic = 16363.13

## Arima Fitting (4,0,0)

fitArima3EURJPY <- arima(EURJPYARIMATS, order = c(4,0,0), include.mean = TRUE)  
fitArima3EURJPY

##   
## Call:  
## arima(x = EURJPYARIMATS, order = c(4, 0, 0), include.mean = TRUE)  
##   
## Coefficients:  
## ar1 ar2 ar3 ar4 intercept  
## 0.9593 0.0498 -0.0469 0.0365 127.7675  
## s.e. 0.0125 0.0173 0.0173 0.0125 7.2429  
##   
## sigma^2 estimated as 0.7532: log likelihood = -8175.86, aic = 16363.71

##Fitting Arima (2,0,4)

fitArima4EURJPY <- arima(EURJPYARIMATS, order = c(2,0,4), include.mean = TRUE)  
fitArima4EURJPY

##   
## Call:  
## arima(x = EURJPYARIMATS, order = c(2, 0, 4), include.mean = TRUE)  
##   
## Coefficients:  
## ar1 ar2 ma1 ma2 ma3 ma4 intercept  
## 1.4936 -0.4942 -0.5347 0.0330 -0.0457 0.0198 127.8783  
## s.e. 1.8666 1.8643 1.8652 0.0752 0.0284 0.0662 7.2892  
##   
## sigma^2 estimated as 0.7531: log likelihood = -8175.57, aic = 16367.14

##Best possible model is selected by AIC scores of the models

library(dLagM)

## Warning: package 'dLagM' was built under R version 4.0.5

## Loading required package: nardl

## Warning: package 'nardl' was built under R version 4.0.5

## Registered S3 method overwritten by 'quantmod':  
## method from  
## as.zoo.data.frame zoo

## Loading required package: dynlm

## Loading required package: zoo

##   
## Attaching package: 'zoo'

## The following object is masked from 'package:timeSeries':  
##   
## time<-

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

ARIMAModelSelectionEURJPY = AIC(fitArima1EURJPY,fitArima2EURJPY,fitArima3EURJPY,fitArima4EURJPY)  
sortScore(ARIMAModelSelectionEURJPY, score ="aic")

## df AIC  
## fitArima2EURJPY 6 16363.13  
## fitArima3EURJPY 6 16363.71  
## fitArima1EURJPY 7 16365.07  
## fitArima4EURJPY 8 16367.14

### Base on the above the fitArima1CanJap is selected

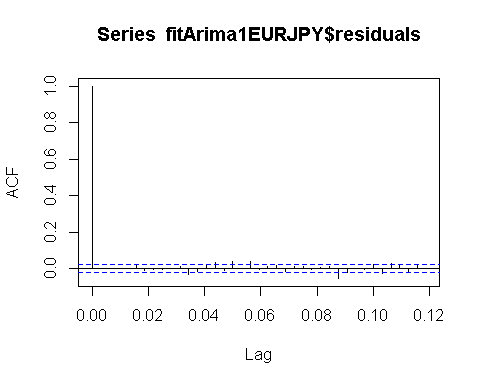
## 

confint(fitArima1EURJPY)

## 2.5 % 97.5 %  
## ar1 0.99755614 0.99992631  
## ma1 -0.06415461 -0.01500970  
## ma2 -0.01145425 0.03772620  
## ma3 -0.06390324 -0.01422998  
## ma4 -0.02786963 0.02163855  
## intercept 113.46529540 141.98416788

## Runing code to obtain Box Test Rest

acf(fitArima1EURJPY$residuals)



library(FitAR)

## Warning: package 'FitAR' was built under R version 4.0.5

## Loading required package: lattice

## Loading required package: leaps

## Loading required package: ltsa

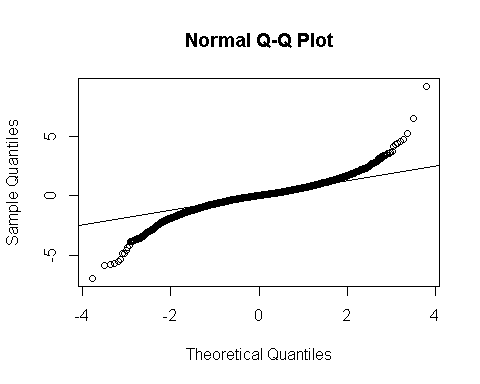
## Loading required package: bestglm

## Warning: package 'bestglm' was built under R version 4.0.5

library(bestglm)  
 Box.test(resid(fitArima1EURJPY),type="Ljung",lag=20,fitdf=1)

##   
## Box-Ljung test  
##   
## data: resid(fitArima1EURJPY)  
## X-squared = 40.675, df = 19, p-value = 0.002666

qqnorm(fitArima1EURJPY$residuals)  
qqline(fitArima1EURJPY$residuals)



## Using Auto.arima to find the best model fit

library(forecast)

## Warning: package 'forecast' was built under R version 4.0.5

##   
## Attaching package: 'forecast'

## The following object is masked from 'package:FitAR':  
##   
## BoxCox

## The following object is masked from 'package:dLagM':  
##   
## forecast

auto.arima(EURJPYARIMATS, trace=TRUE)

##   
## Fitting models using approximations to speed things up...  
##   
## ARIMA(2,1,2)(1,0,1)[320] with drift : Inf  
## ARIMA(0,1,0) with drift : 16371.89  
## ARIMA(1,1,0)(1,0,0)[320] with drift : Inf  
## ARIMA(0,1,1)(0,0,1)[320] with drift : Inf  
## ARIMA(0,1,0) : 16369.99  
## ARIMA(0,1,0)(1,0,0)[320] with drift : Inf  
## ARIMA(0,1,0)(0,0,1)[320] with drift : Inf  
## ARIMA(0,1,0)(1,0,1)[320] with drift : Inf  
## ARIMA(1,1,0) with drift : 16357.31  
## ARIMA(1,1,0)(0,0,1)[320] with drift : Inf  
## ARIMA(1,1,0)(1,0,1)[320] with drift : Inf  
## ARIMA(2,1,0) with drift : 16356.97  
## ARIMA(2,1,0)(1,0,0)[320] with drift : Inf  
## ARIMA(2,1,0)(0,0,1)[320] with drift : Inf  
## ARIMA(2,1,0)(1,0,1)[320] with drift : Inf  
## ARIMA(3,1,0) with drift : 16349.43  
## ARIMA(3,1,0)(1,0,0)[320] with drift : Inf  
## ARIMA(3,1,0)(0,0,1)[320] with drift : Inf  
## ARIMA(3,1,0)(1,0,1)[320] with drift : Inf  
## ARIMA(4,1,0) with drift : 16352.02  
## ARIMA(3,1,1) with drift : 16351.3  
## ARIMA(2,1,1) with drift : 16358.64  
## ARIMA(4,1,1) with drift : 16353.45  
## ARIMA(3,1,0) : 16347.5  
## ARIMA(3,1,0)(1,0,0)[320] : Inf  
## ARIMA(3,1,0)(0,0,1)[320] : Inf  
## ARIMA(3,1,0)(1,0,1)[320] : Inf  
## ARIMA(2,1,0) : 16355.04  
## ARIMA(4,1,0) : 16350.1  
## ARIMA(3,1,1) : 16349.37  
## ARIMA(2,1,1) : 16356.68  
## ARIMA(4,1,1) : 16351.39  
##   
## Now re-fitting the best model(s) without approximations...  
##   
## ARIMA(3,1,0) : 16355.37  
##   
## Best model: ARIMA(3,1,0)

## Series: EURJPYARIMATS   
## ARIMA(3,1,0)   
##   
## Coefficients:  
## ar1 ar2 ar3  
## -0.0401 0.0098 -0.0371  
## s.e. 0.0125 0.0125 0.0125  
##   
## sigma^2 estimated as 0.754: log likelihood=-8173.68  
## AIC=16355.36 AICc=16355.37 BIC=16382.42

## forecasting using Best model: ARIMA(0,1,0)

forecastarimaEURJPY<- predict(fitArima1EURJPY,n.ahead = 100)  
forecastarimaEURJPY

## $pred  
## Time Series:  
## Start = c(2019, 320)   
## End = c(2020, 99)   
## Frequency = 320   
## [1] 126.1434 126.1373 126.1668 126.1709 126.1729 126.1748 126.1768 126.1787  
## [9] 126.1807 126.1826 126.1846 126.1865 126.1884 126.1904 126.1923 126.1942  
## [17] 126.1962 126.1981 126.2000 126.2019 126.2038 126.2057 126.2077 126.2096  
## [25] 126.2115 126.2134 126.2153 126.2172 126.2191 126.2210 126.2229 126.2248  
## [33] 126.2267 126.2285 126.2304 126.2323 126.2342 126.2361 126.2379 126.2398  
## [41] 126.2417 126.2435 126.2454 126.2473 126.2491 126.2510 126.2528 126.2547  
## [49] 126.2565 126.2584 126.2602 126.2621 126.2639 126.2658 126.2676 126.2694  
## [57] 126.2713 126.2731 126.2749 126.2767 126.2786 126.2804 126.2822 126.2840  
## [65] 126.2858 126.2876 126.2895 126.2913 126.2931 126.2949 126.2967 126.2985  
## [73] 126.3003 126.3021 126.3038 126.3056 126.3074 126.3092 126.3110 126.3128  
## [81] 126.3145 126.3163 126.3181 126.3199 126.3216 126.3234 126.3252 126.3269  
## [89] 126.3287 126.3304 126.3322 126.3339 126.3357 126.3374 126.3392 126.3409  
## [97] 126.3427 126.3444 126.3462 126.3479  
##   
## $se  
## Time Series:  
## Start = c(2019, 320)   
## End = c(2020, 99)   
## Frequency = 320   
## [1] 0.8678218 1.2024849 1.4683888 1.6759039 1.8588021 2.0248482 2.1779010  
## [8] 2.3205337 2.4545644 2.5813341 2.7018659 2.8169612 2.9272621 3.0332923  
## [15] 3.1354858 3.2342069 3.3297650 3.4224257 3.5124188 3.5999450 3.6851806  
## [22] 3.7682816 3.8493866 3.9286199 4.0060929 4.0819065 4.1561518 4.2289119  
## [29] 4.3002626 4.3702735 4.4390084 4.5065261 4.5728808 4.6381230 4.7022993  
## [36] 4.7654533 4.8276253 4.8888534 4.9491728 5.0086168 5.0672166 5.1250016  
## [43] 5.1819992 5.2382356 5.2937354 5.3485218 5.4026170 5.4560417 5.5088160  
## [50] 5.5609585 5.6124872 5.6634193 5.7137709 5.7635577 5.8127943 5.8614951  
## [57] 5.9096736 5.9573427 6.0045148 6.0512020 6.0974155 6.1431665 6.1884654  
## [64] 6.2333223 6.2777471 6.3217492 6.3653375 6.4085208 6.4513075 6.4937057  
## [71] 6.5357232 6.5773676 6.6186462 6.6595661 6.7001340 6.7403566 6.7802403  
## [78] 6.8197913 6.8590156 6.8979191 6.9365072 6.9747856 7.0127596 7.0504342  
## [85] 7.0878146 7.1249055 7.1617118 7.1982380 7.2344885 7.2704678 7.3061800  
## [92] 7.3416294 7.3768198 7.4117553 7.4464396 7.4808764 7.5150693 7.5490220  
## [99] 7.5827378 7.6162200

par(mfrow = c(1,1))