ISYE 6402 Homework 2

Part 1: Currency Conversion Analysis

Background

In this problem, we will study fluctuations in currency exchange rate over time.

File `USD-JPY.csv` download contains the daily exchange rate of USD/JPY from January 2000 through May 31st 2022. We will aggregate the data on a weekly basis, by taking the average rate within each week. The time series of interest is the weekly currency exchange. We will analyze this time series and its first order difference.

Instructions on reading the data

To read the data in `R` , save the file in your working directory (make sure you have changed the directory if different from the R working directory) and read the data using the `R` function `read.csv()`

```
fpath <- "USD-JPY.csv"
df <- read.csv(fpath, head = TRUE)
```

Here we upload the libraries needed the this data analysis:

```
library(mgcv)
```

```
## Loading required package: nlme
```

```
## This is mgcv 1.8-38. For overview type 'help("mgcv-package")'.
```

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

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

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following object is masked from 'package:nlme':
##
##     collapse
```

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

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

To prepare the data, run the following code snippet. First, aggregate by week:

```
df$date <- as.Date(df$Date, format='%Y-%m-%d')
df$week <- floor_date(df$date, "week")



df <- df[, c("week", "jpy")]
```

We now form the weekly aggrgated time series to use for data exploration! Please note that we will analyze the weekly aggregated data not the original (daily) data.

```
agg <- aggregate(x = df$jpy, by = list(df$week), FUN = mean)
colnames(agg) <- c("week", "jpy")

jpy.ts <- ts(agg$jpy, start = 2000, freq = 52)
```

Please use the `jpy` series to code and answer the following questions.

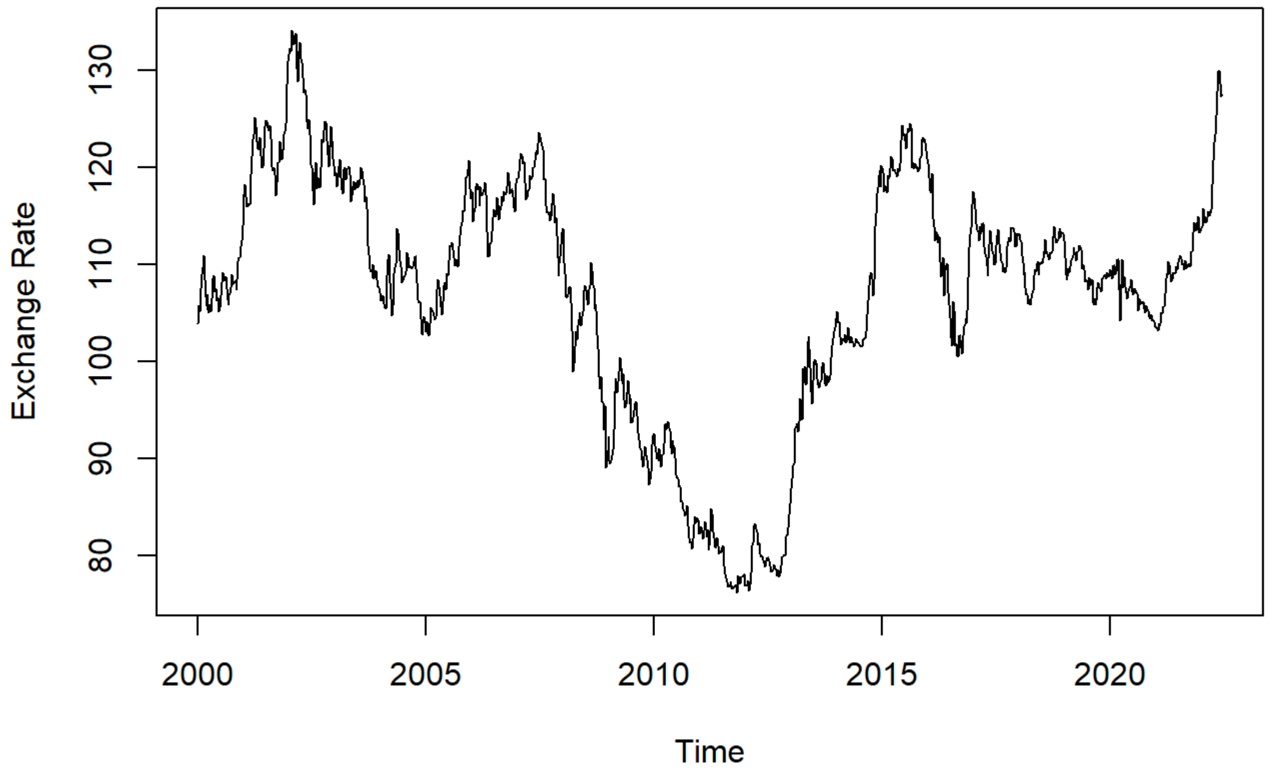
Question 1a: Exploratory Data Analysis

Before exploring the data, can you infer the data features from what you know about the USD-JPY currency exchange? Next plot the Time Series and ACF plots of the weekly data. Comment on the main features, and identify what (if any) assumptions of stationarity are violated.

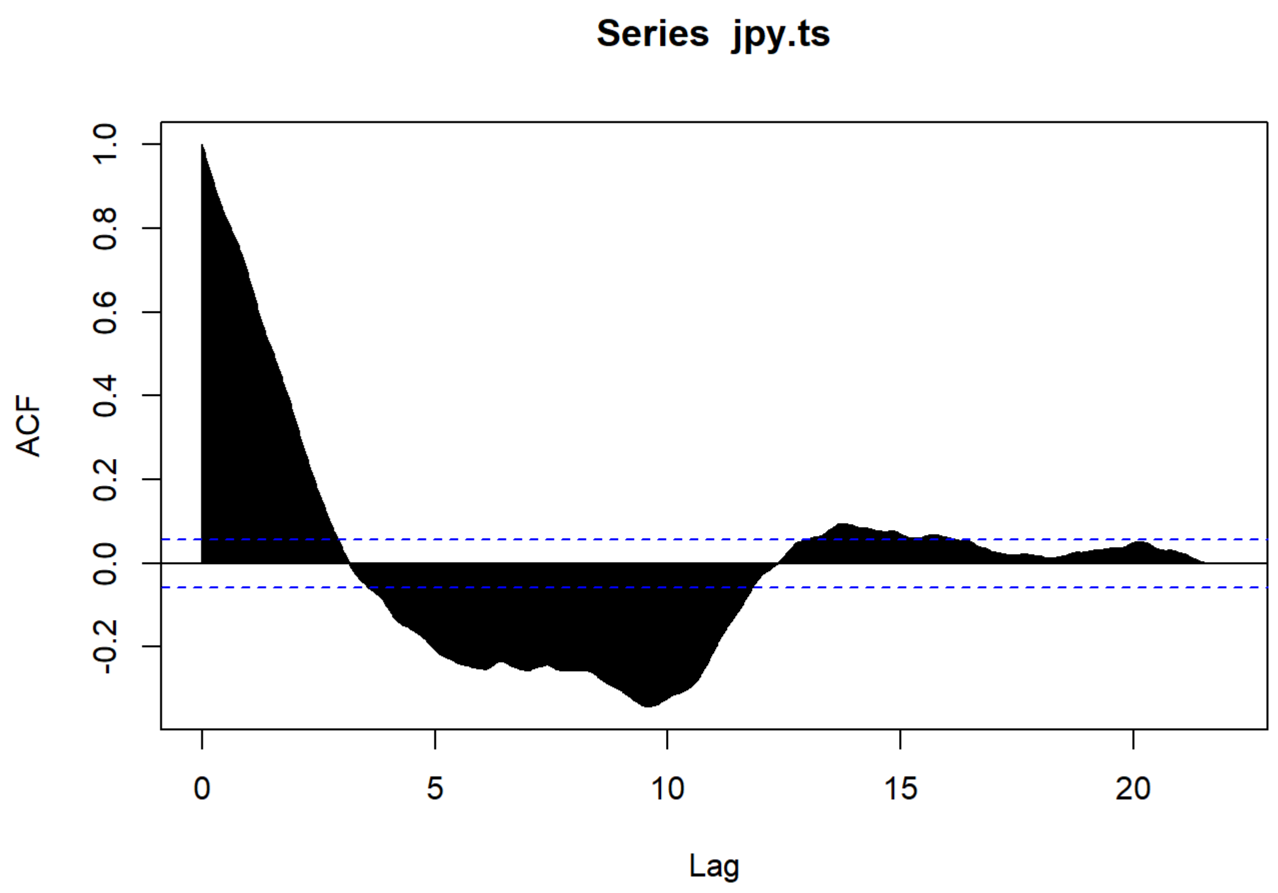
Which type of model do you think will fit the data better: the trend or seasonality fitting model? Provide details for your response.

Response: General Insights on the USD-JPY Currency Rate I am assuming that the data should follow similarly to the US stock market (i.e. decreases in 2008, 2012, and 2020). Based on the time series plot, the assumptions of stationarity that are violated are constant mean and constant variance. There is also significant autocorrelation present.

```
ts.plot(jpy.ts,ylab="Exchange Rate")
```



```
acf(jpy.ts, lag.max = 52*22)
```



Response: General Insights from the Graphical Analysis Based on the ACF plot, seasonality could play a role if you expand the lag.max parameter to 52 weeks * 22 years. Therefore, a seasonality plot would probably be the most realistic.

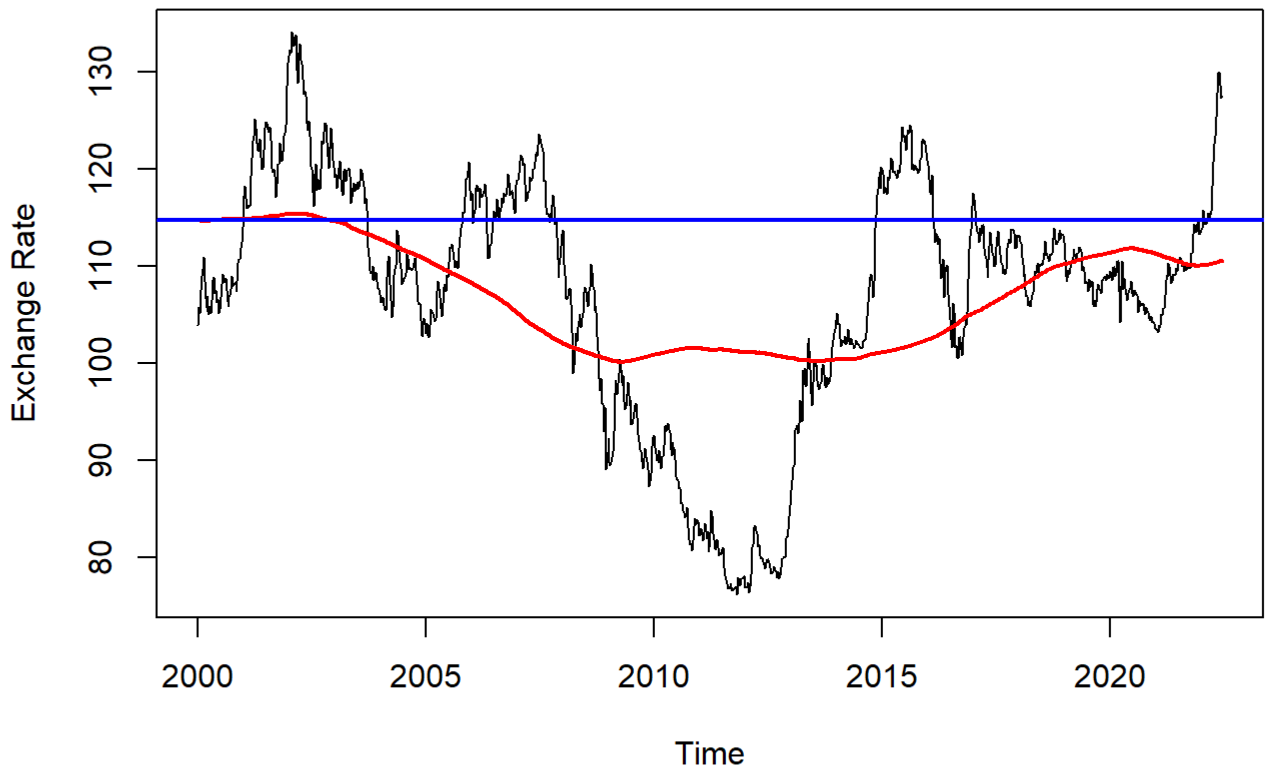
Question 1b: Trend Estimation

Fit the following trend estimation models:

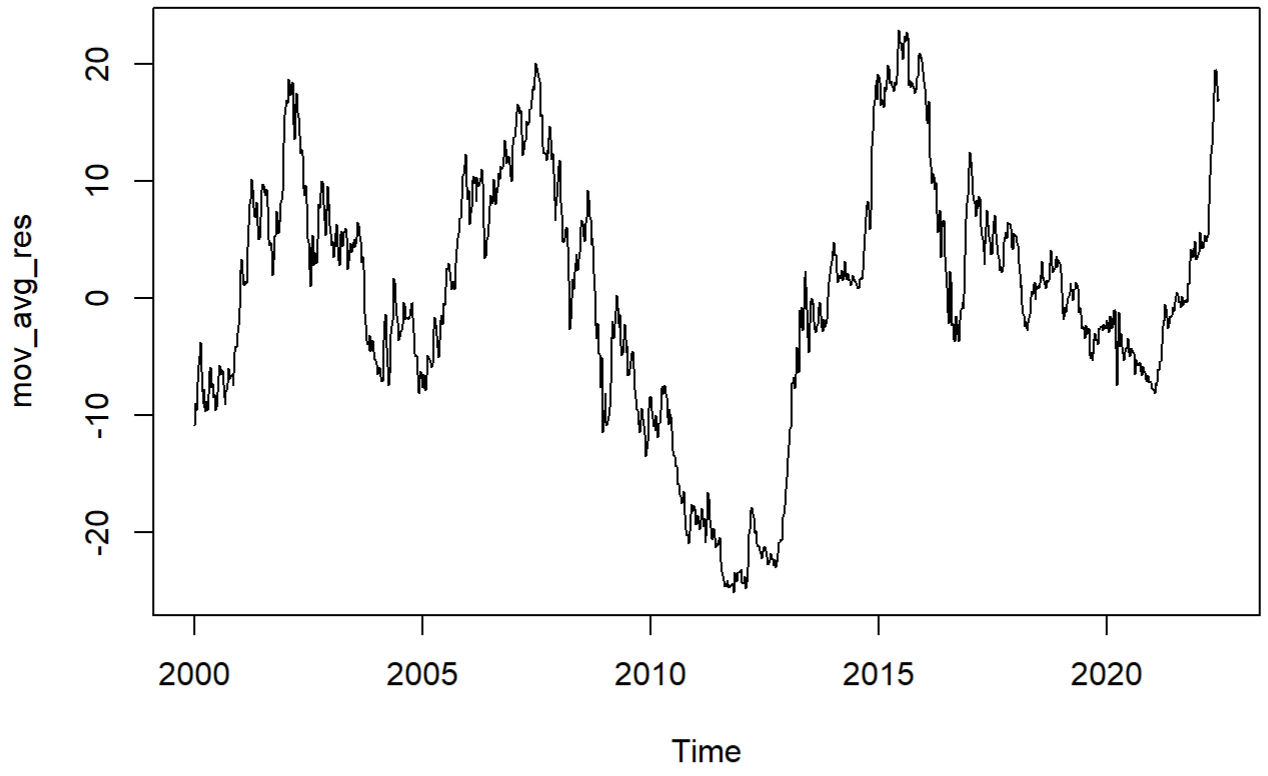
- Moving Average
- Parametric Quadratic Polynomial
- Local Polynomial
- Splines Smoothing

Overlay the fitted values on the original time series. Plot the residuals with respect to time for each model. Plot the ACF of the residuals for each model also. Comment on the four models fit and on the appropriateness of the stationarity assumption of the residuals.

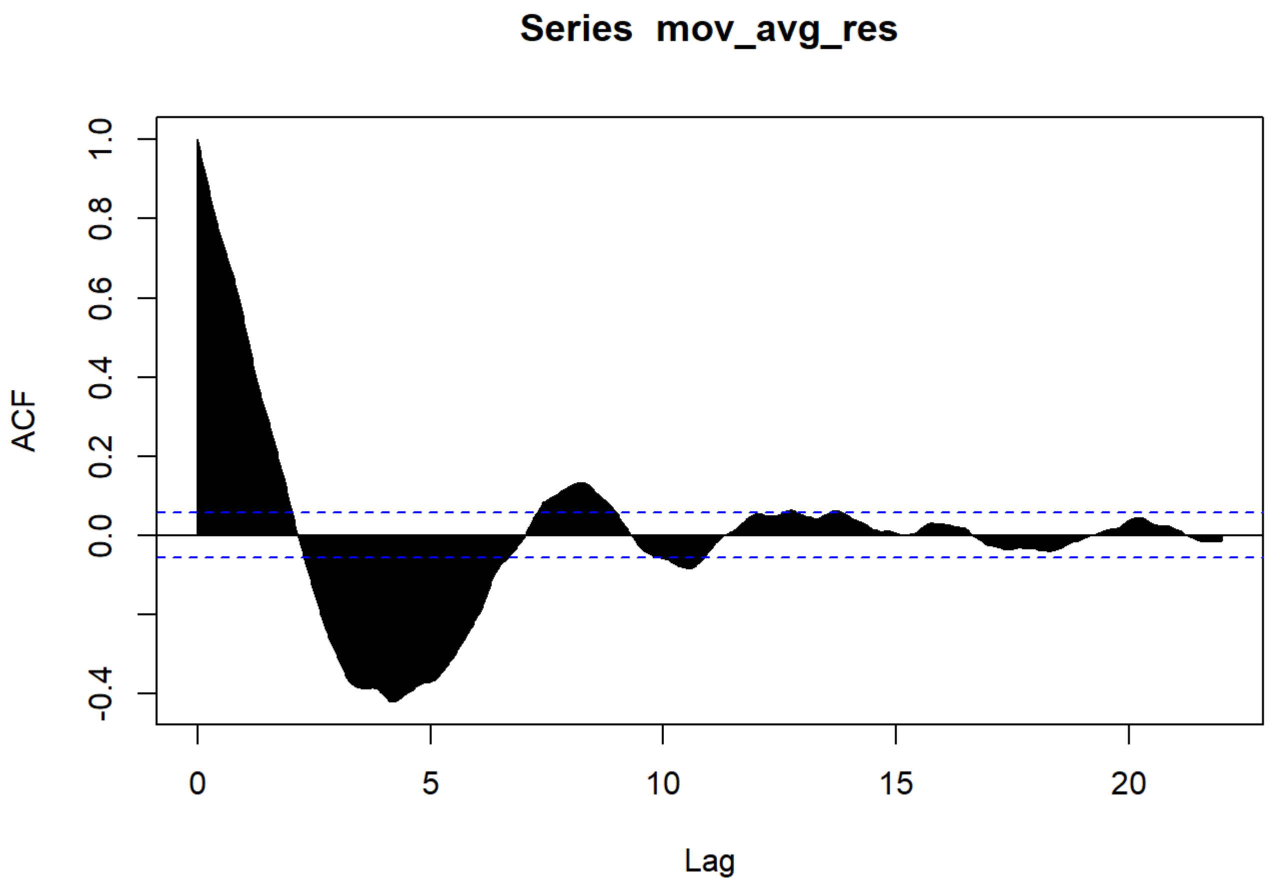
```
time_pts = c(1:length(jpy.ts))
time_pts = c(time_pts-min(time_pts))/max(time_pts)

#moving average
mov_avg = ksmooth(time_pts,jpy.ts, kernel = "box")
mov_avg_fit = ts(mov_avg$y, start = 2000, frequency = 52)

plot(jpy.ts, ylab="Exchange Rate")
lines(mov_avg_fit, lwd=2, col="red")
abline(mov_avg_fit[1], 0, lwd=2, col="blue")
```



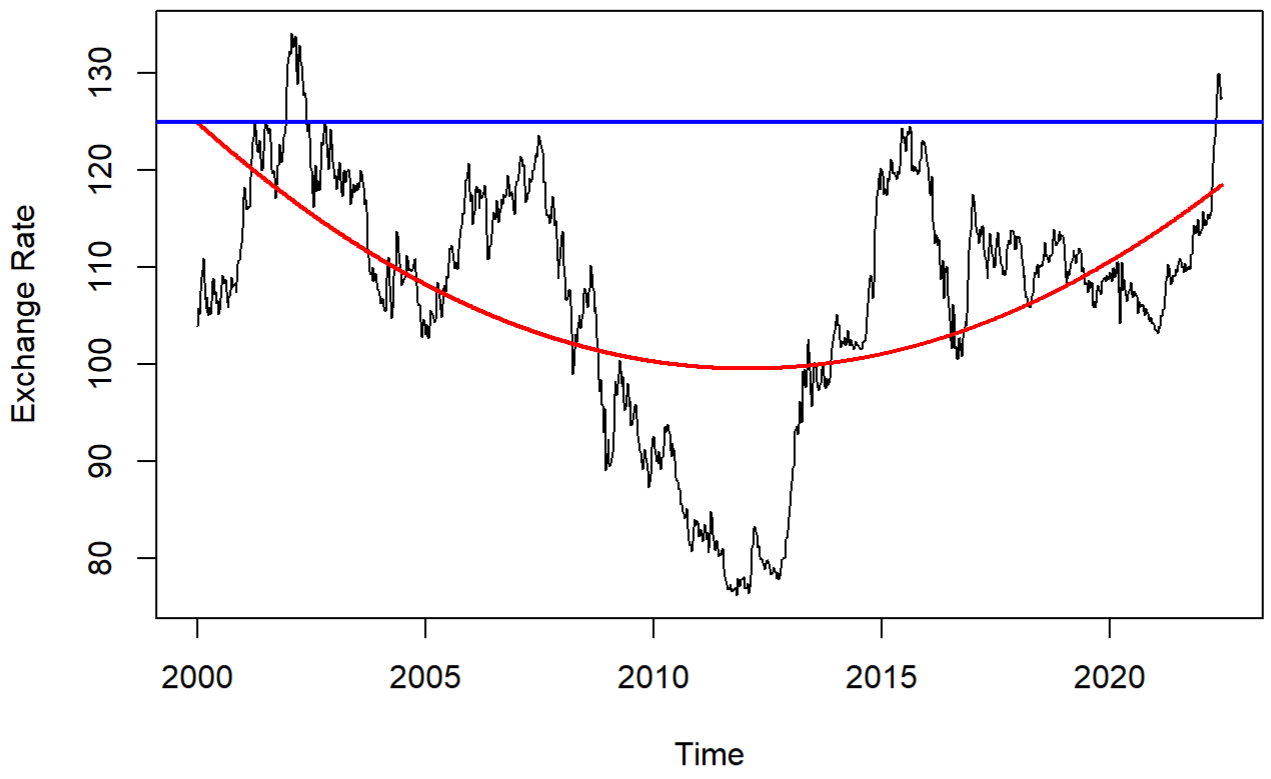
```
#residuals and ACF
mov_avg_res = ts((jpy.ts-mov_avg$y),start=2000,frequency=52)
plot(mov_avg_res)
```



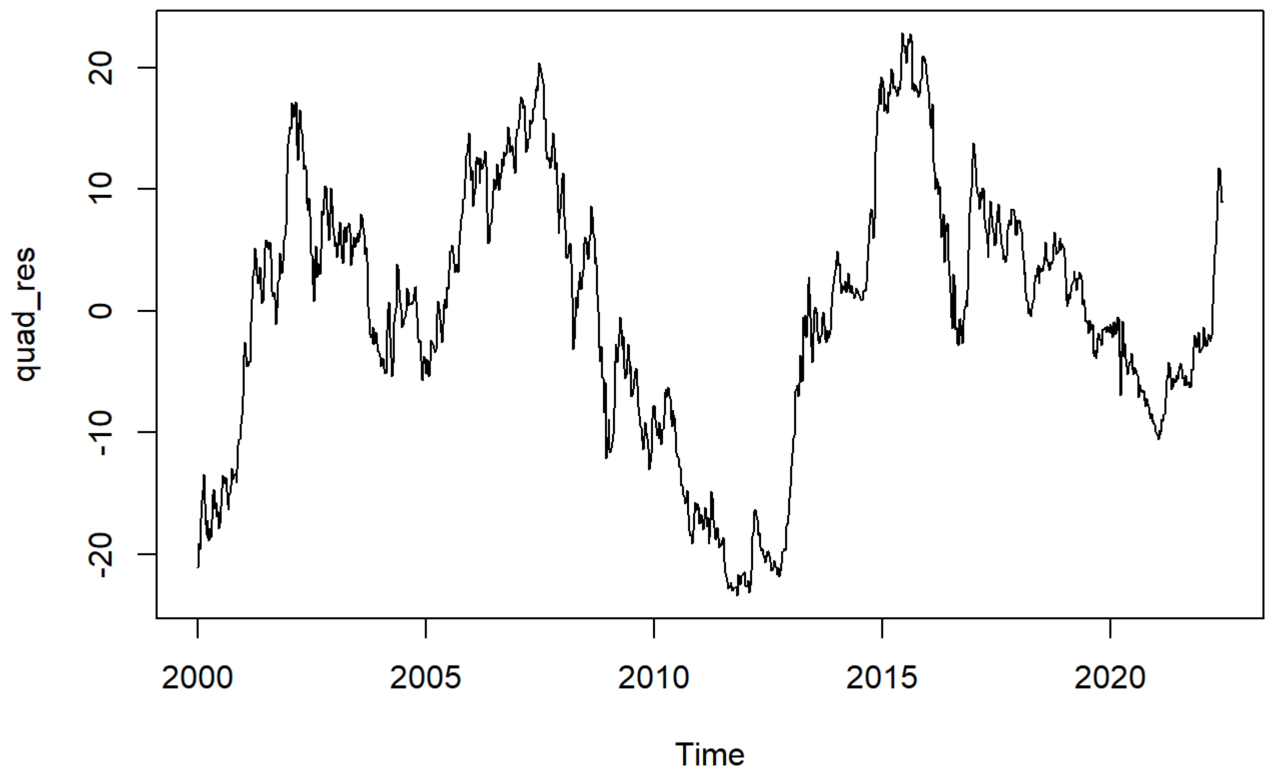
```
acf(mov_avg_res, lag.max = 52*22)
```



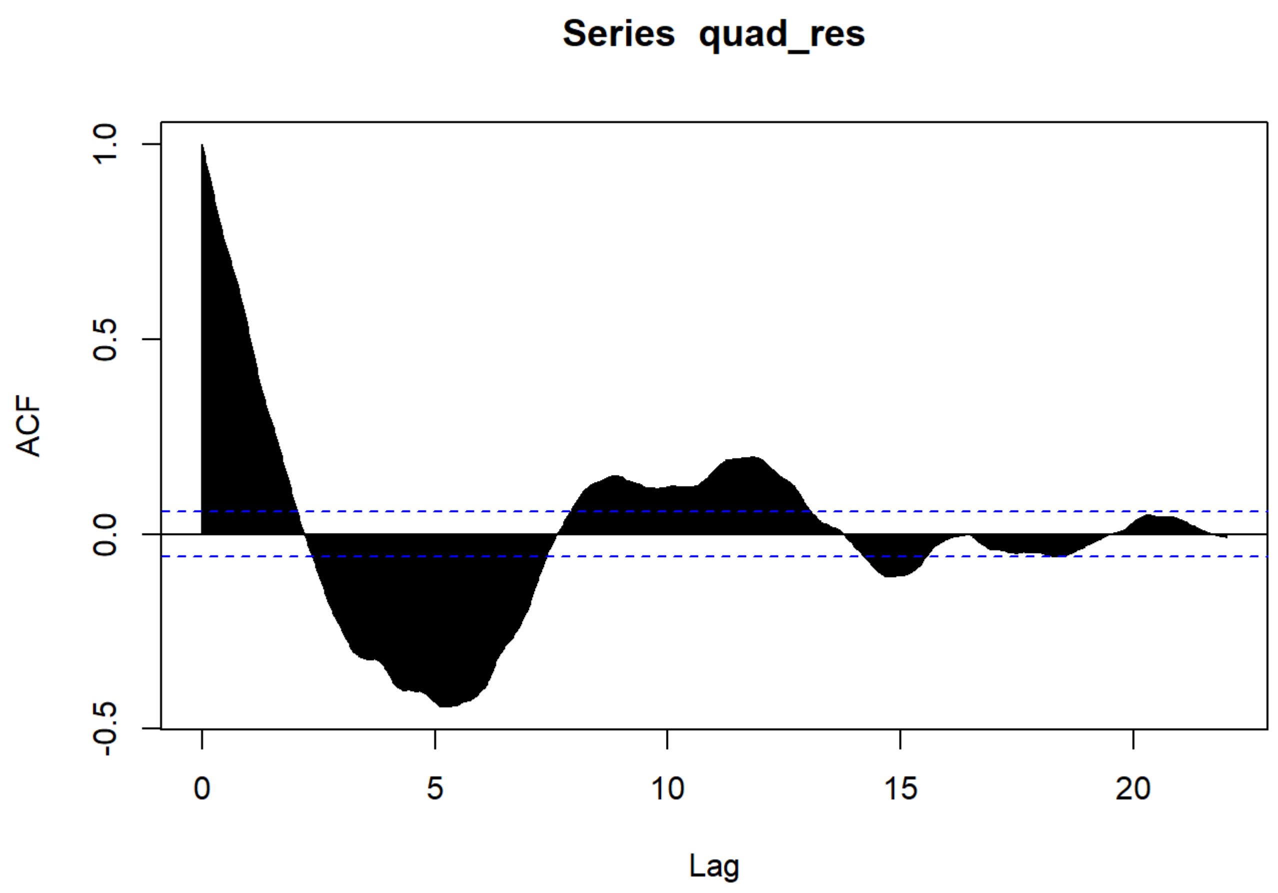
```
#Parametric Quadratic Polynomial
x1 = time_pts
x2 = time_pts^2
quad= lm(jpy.ts ~ x1+x2)
quad_fit = ts(fitted(quad),start = 2000, frequency = 52)
plot(jpy.ts, ylab="Exchange Rate")
lines(quad_fit, lwd=2, col="red")
abline(quad_fit[1], 0, lwd=2, col="blue")
```



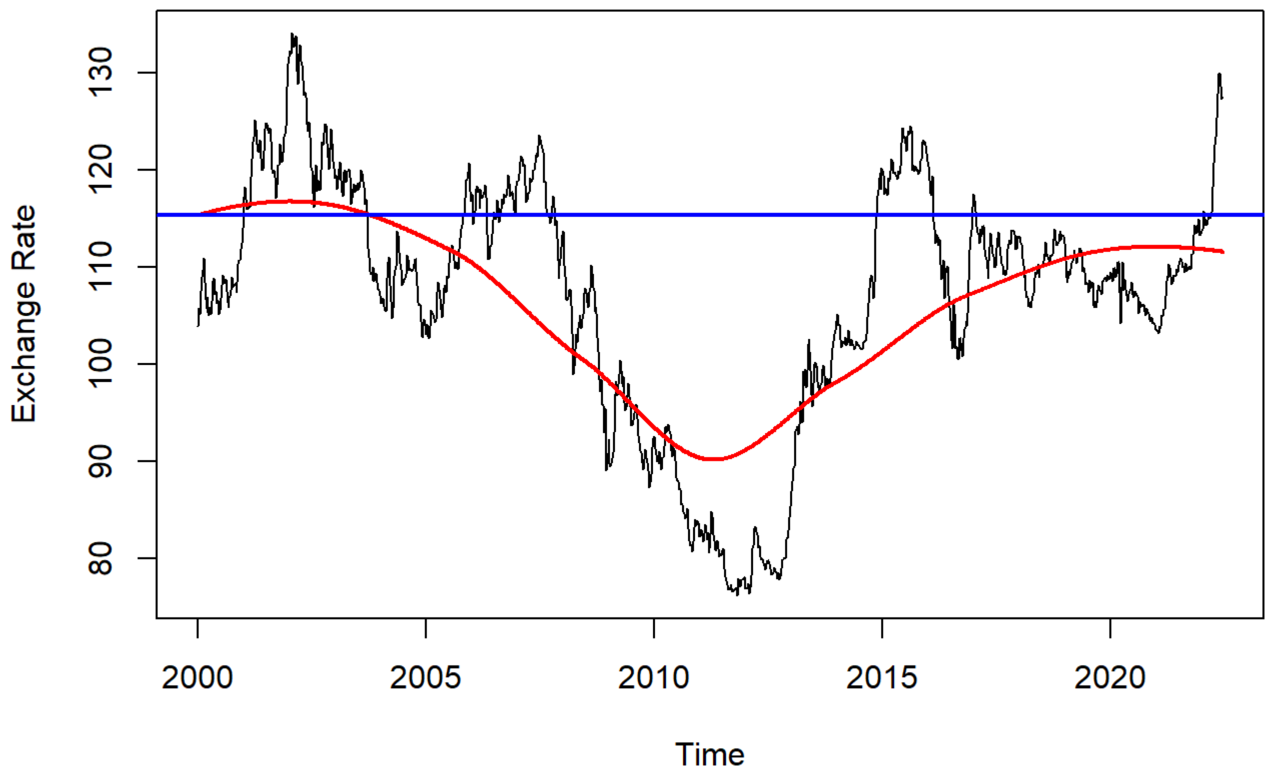
```
#residuals and ACF
quad_res = ts((jpy.ts-fitted(quad)),start=2000,frequency=52)
plot(quad_res)
```



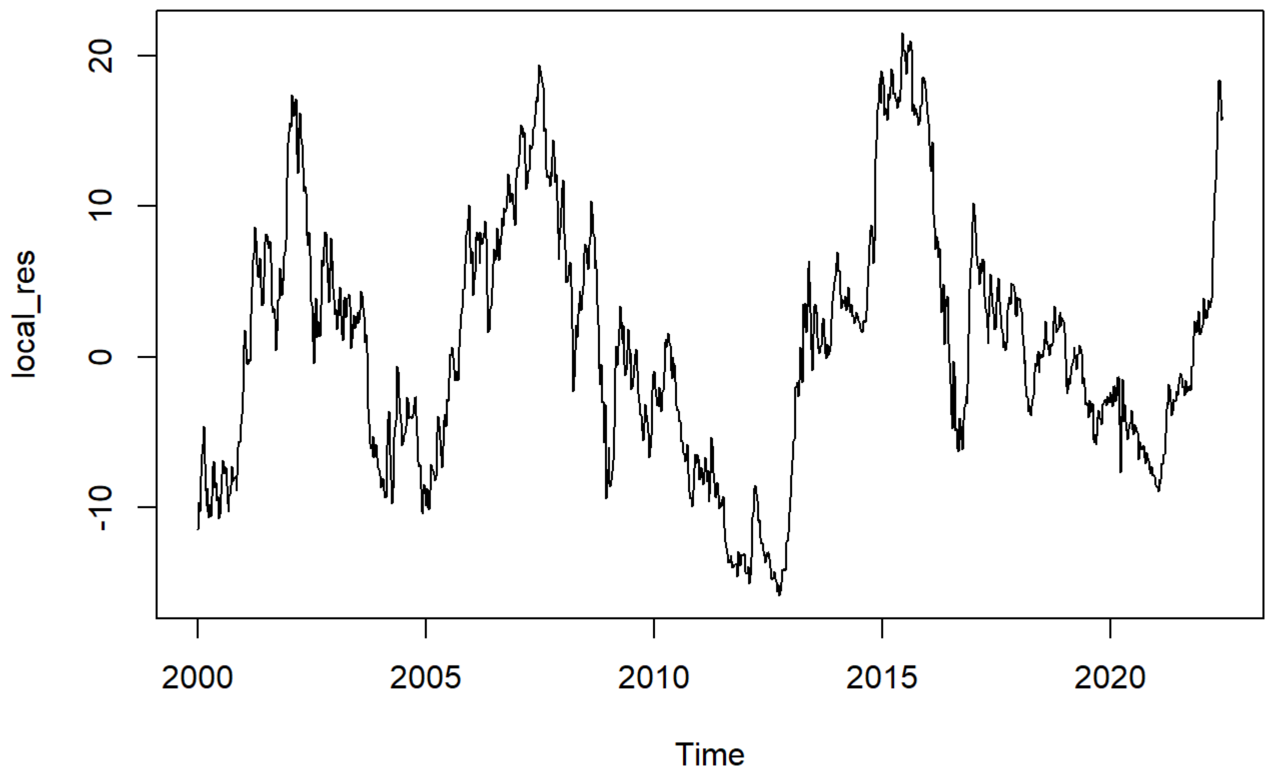
```
acf(quad_res, lag.max = 52*22)
```



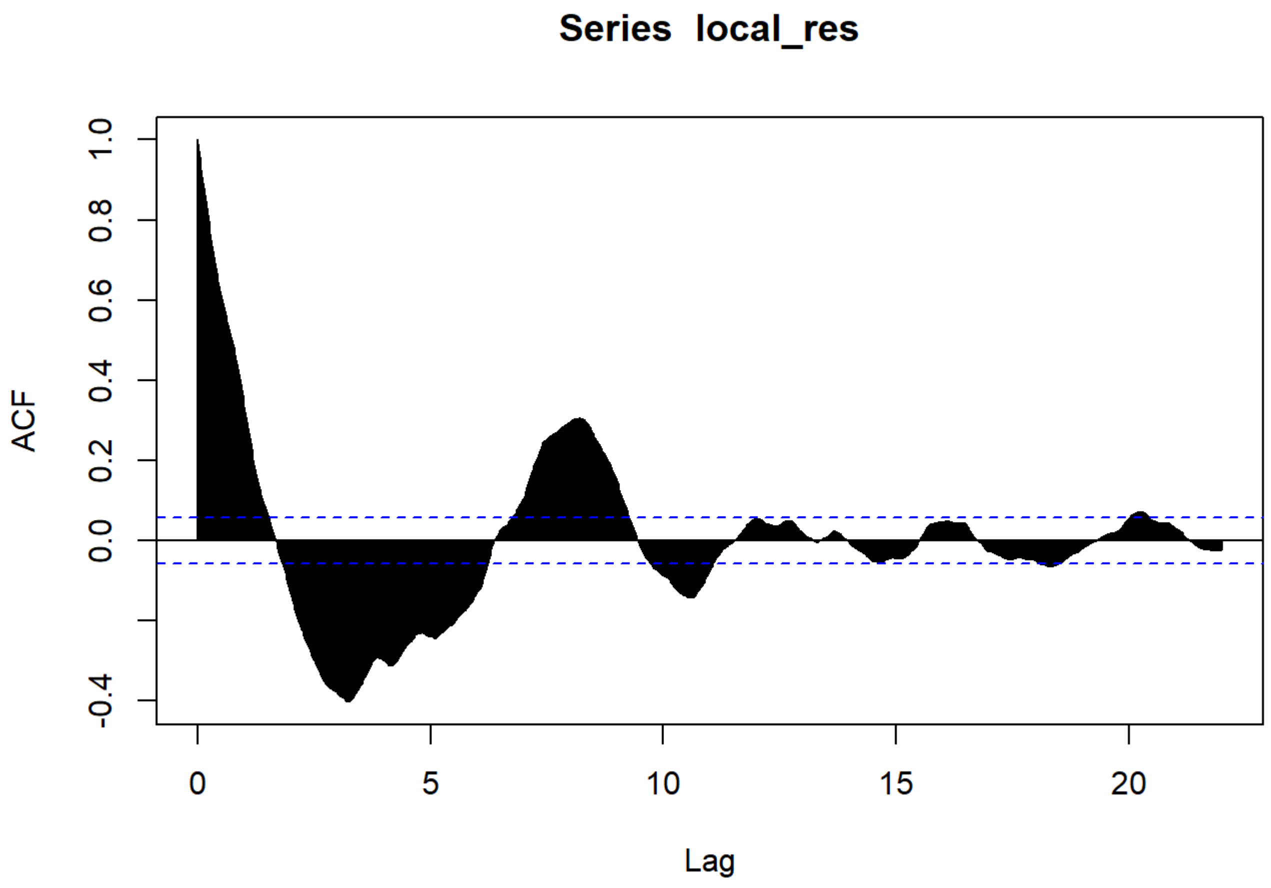
```
#Local Polynomial
local = loess(jpy.ts~time_pts)
local_fit = ts(fitted(local), start = 2000, frequency = 52)
plot(jpy.ts, ylab="Exchange Rate")
lines(local_fit, lwd=2, col="red")
abline(local_fit[1], 0, lwd=2, col="blue")
```



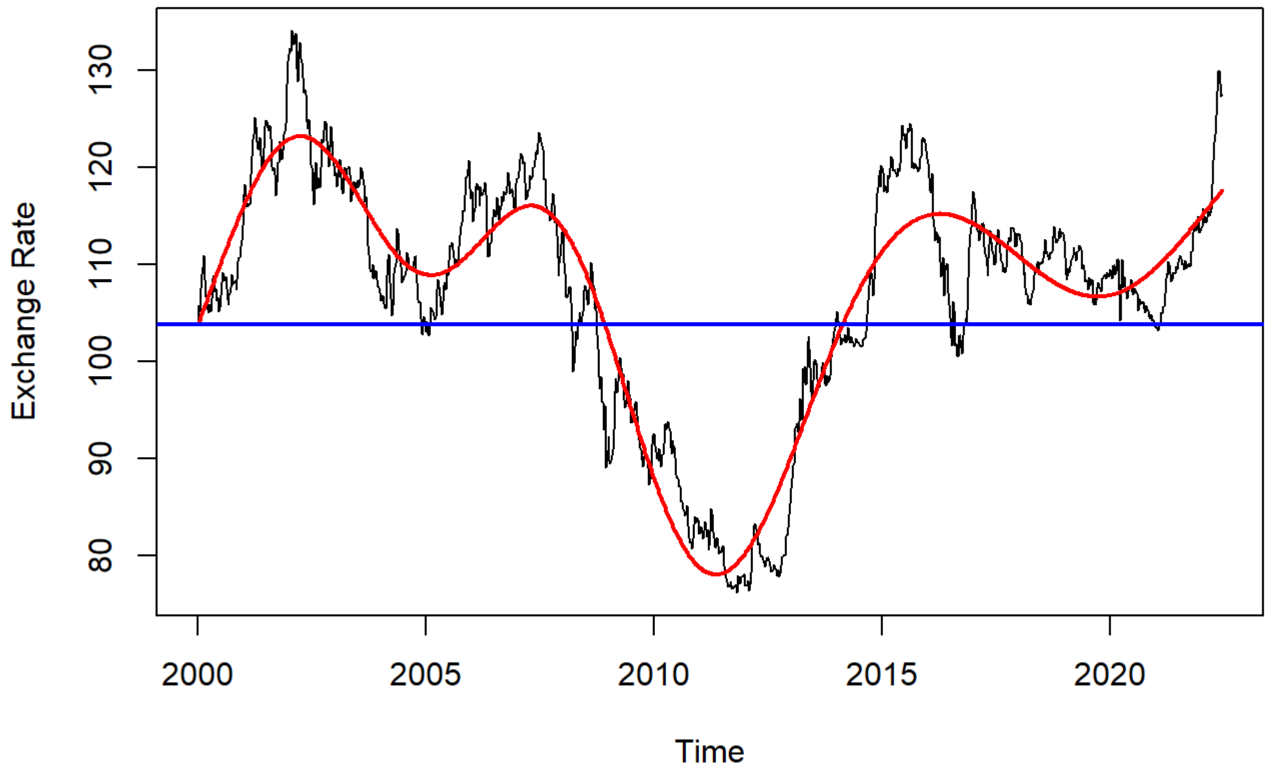
```
#residuals and ACF
local_res = ts((jpy.ts-fitted(local)),start=2000,frequency=52)
plot(local_res)
```



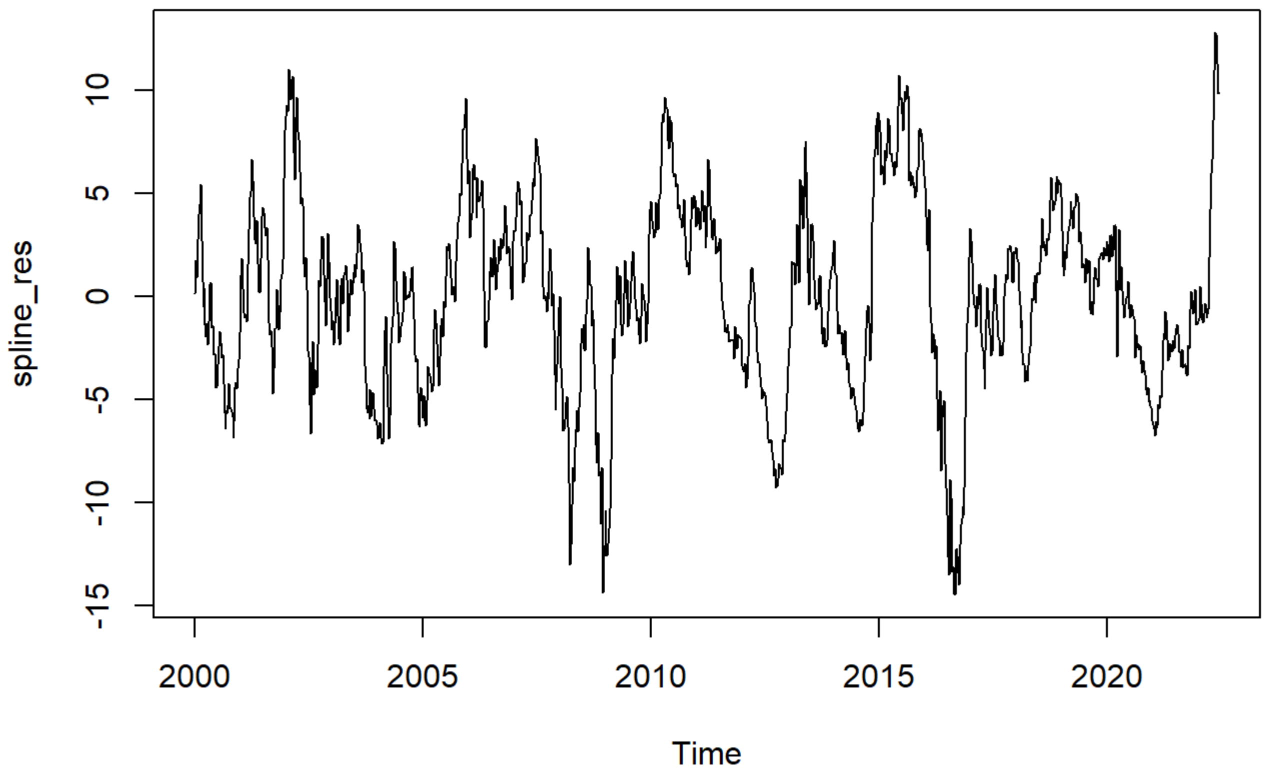
```
acf(local_res, lag.max = 52*22)
```



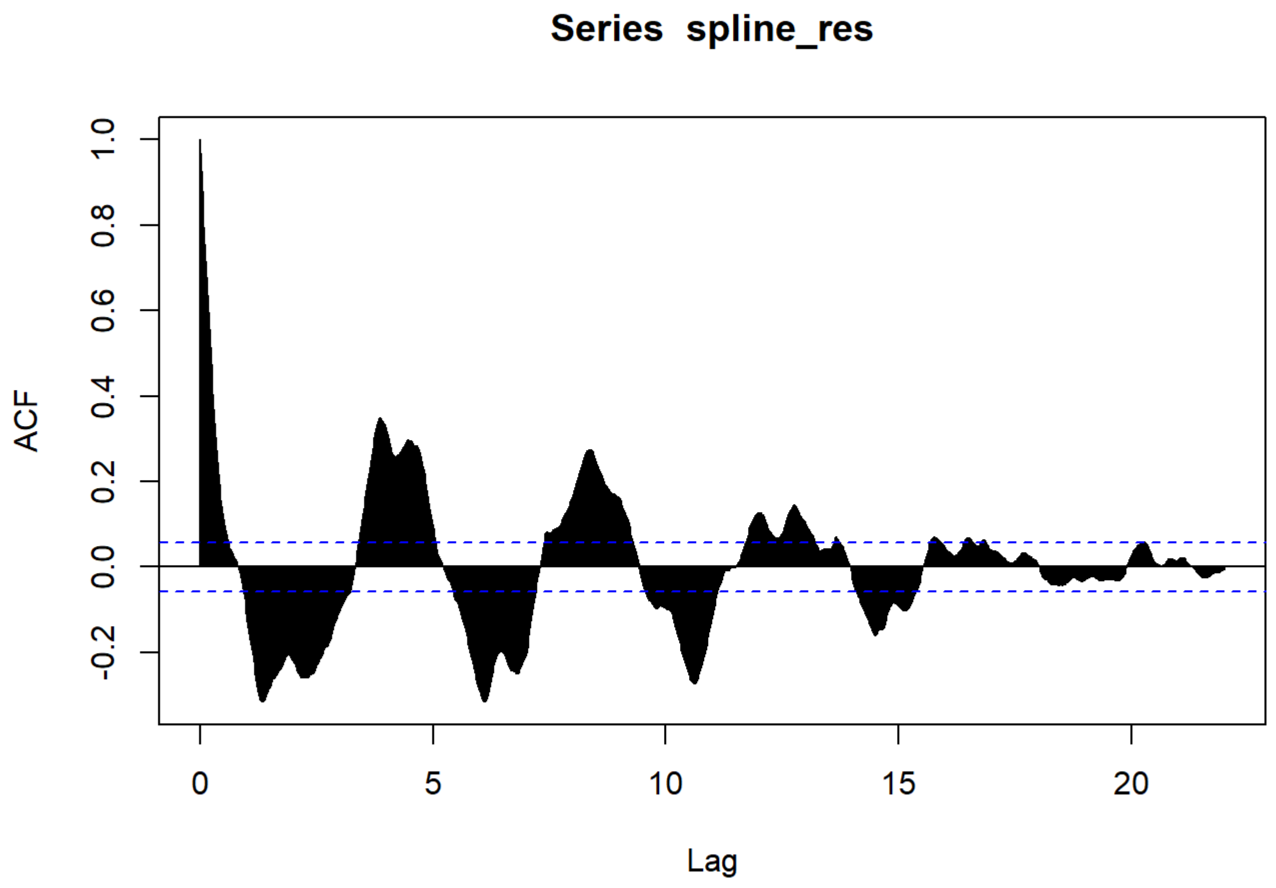
```
#Spline
spl = gam(jpy.ts~s(time_pts))
spl_fit = ts(fitted(spl), start = 2000, frequency = 52)
plot(jpy.ts, ylab="Exchange Rate")
lines(spl_fit, lwd=2, col="red")
abline(spl_fit[1], 0, lwd=2, col="blue")
```



```
#residuals and ACF
spline_res = ts((jpy.ts-fitted(spl)),start=2000,frequency=52)
plot(spline_res)
```

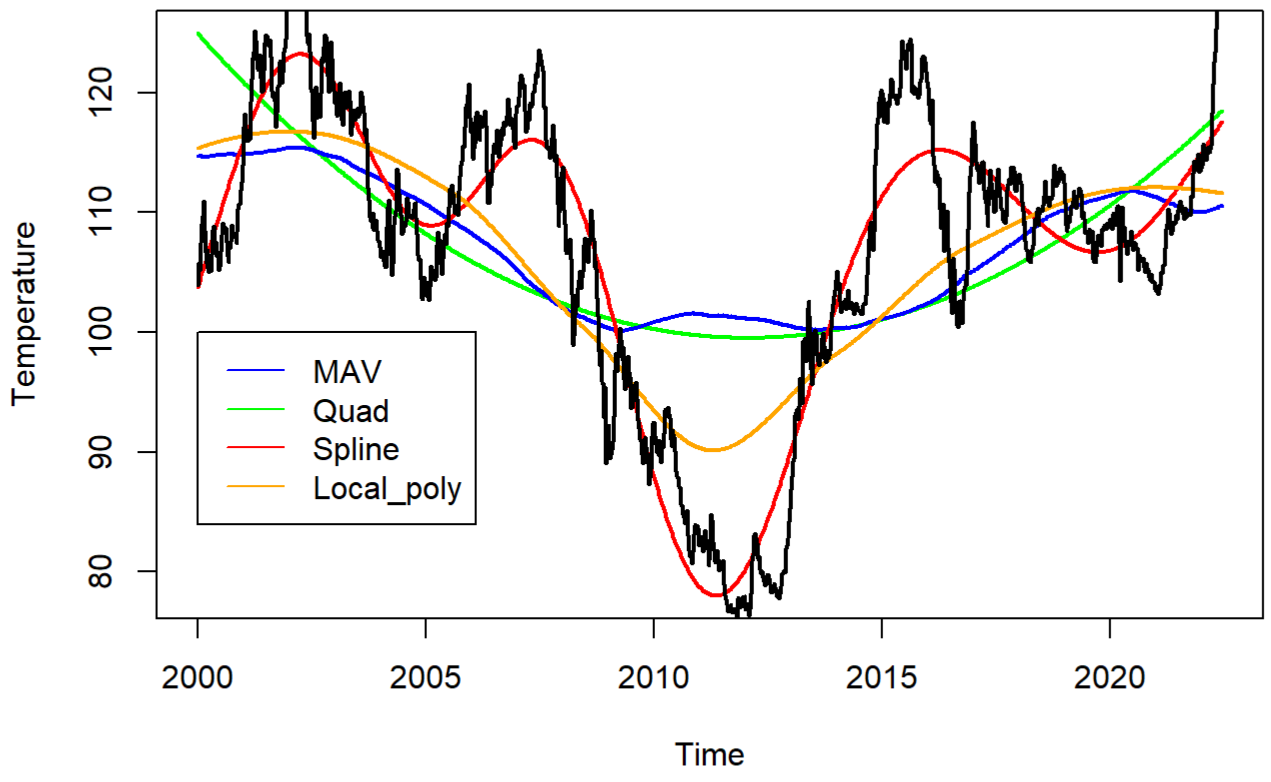


```
acf(spline_res, lag.max = 52*22)
```



Response: Comparison of the fitted trend models: Below is a graph of all the fitted trend models overlaid on top of each other. The black line is the original data. The spline gets the closest on matching the jpy.ts behavior. The Parametric Quadratic Polynomial is the least closest to the original behavior. The local polynomial and moving average have a similar behavior which is to be expected. However, the polynomial decreases more around 2011-2012 time points.

```
## Compare all estimated trends
all_val = c(mov_avg_fit, quad_fit, local_fit, spl_fit)
ylim= c(min(all_val),max(all_val))
ts.plot(quad_fit,lwd=2,col="green",ylim=ylim,ylab="Temperature")
lines(mov_avg_fit,lwd=2,col="blue")
lines(spl_fit,lwd=2,col="red")
lines(local_fit,lwd=2,col="orange1")
lines(jpy.ts,lwd=2,col="black")
legend(x=2000,y=100,legend=c("MAV","Quad","Spline","Local_poly"),lty = 1, col=c("blue","green",
"red","orange1"))
```



Response: Appropriateness of the trend model for stationarity The fitted trends models violate the assumptions for stationarity. There is not a constant mean based on the fitted trend lines as all the plots have a big decrease around 2012. There is not constant variance based on the residual plots. Autocorrelation is present in all the trend models based on the ACF plots. Therefore, stationarity cannot be assumed for these trend models.

Question 1c: Differenced Data Modeling

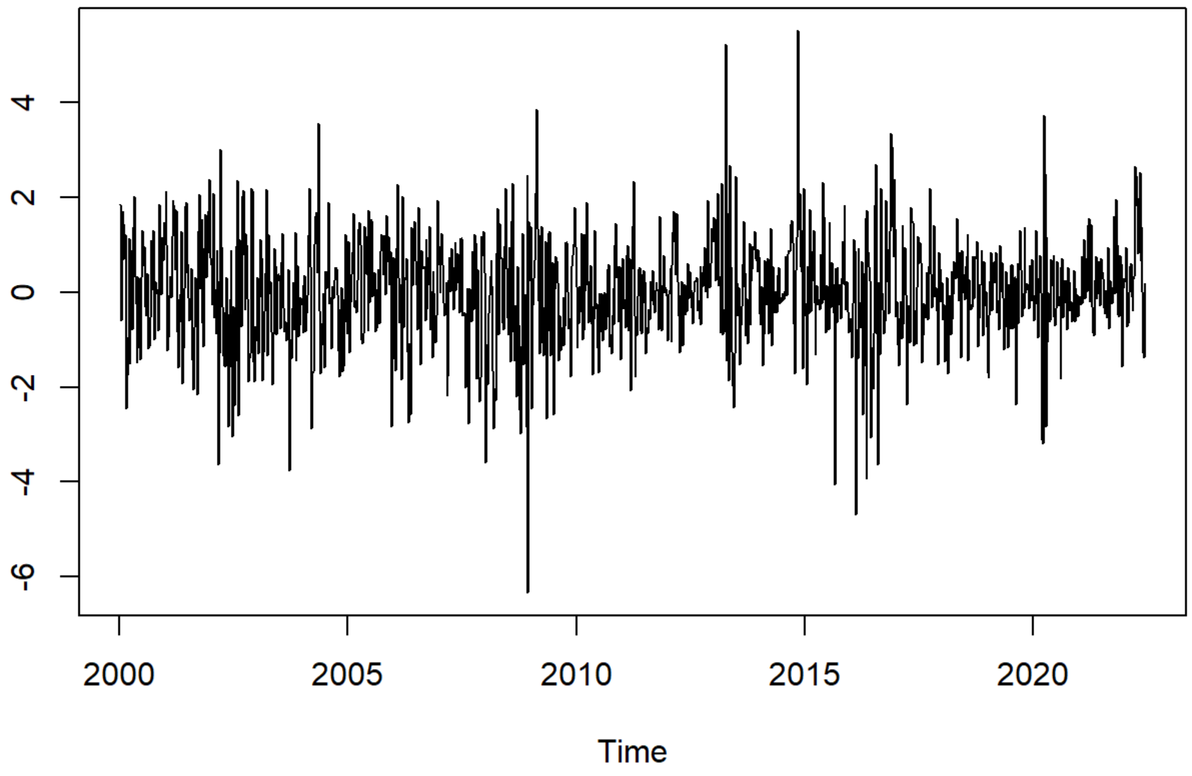
Now plot the difference time series and its ACF plot. Apply the four trend models in Question 1b to the differenced time series. What can you conclude about the difference data in terms of stationarity? Which model would you recommend to apply (trend removal via fitting trend vs differencing) such that to obtain a stationary process?

Hint: When TS data are differenced, the resulting data set will have an NA in the first data element due to the differencing.

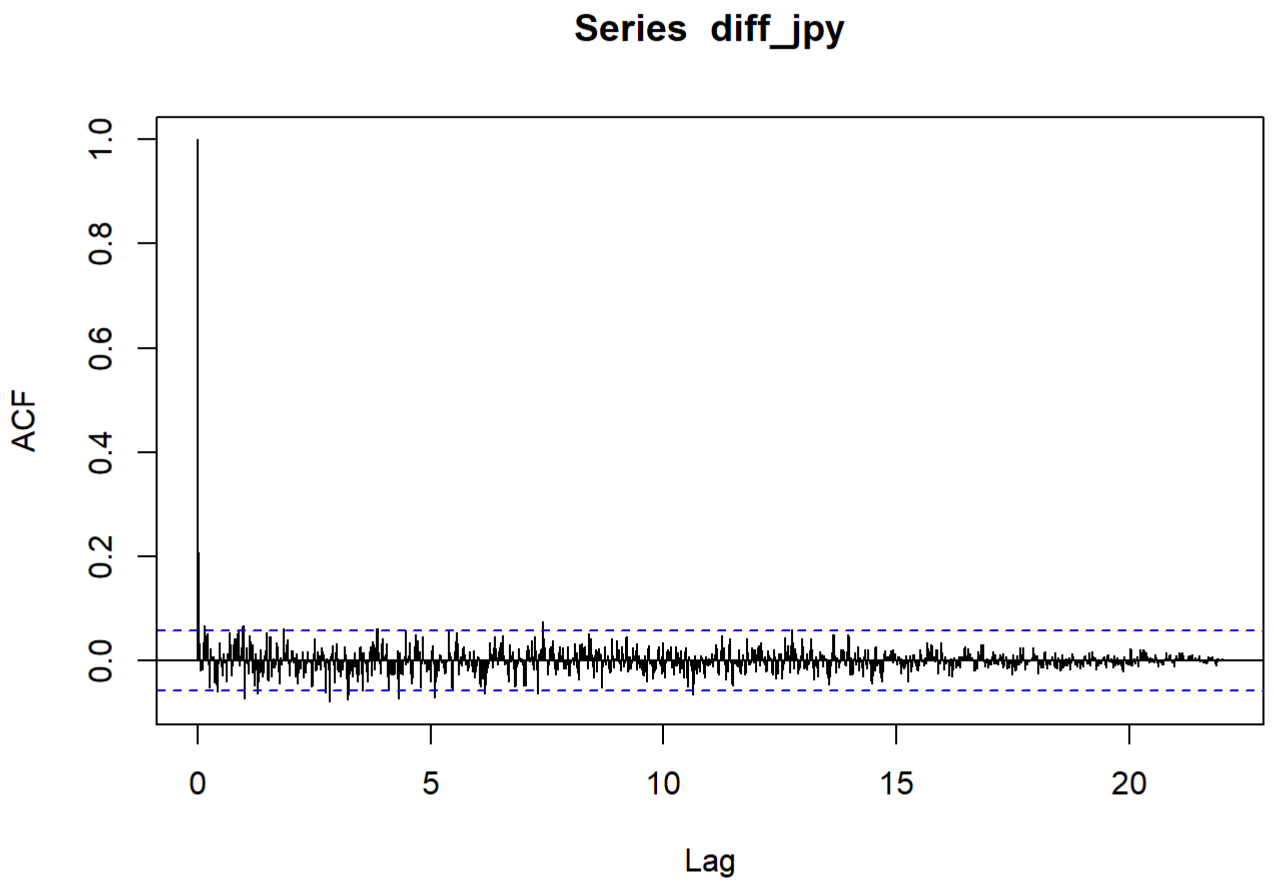
```
diff_jpy = diff(jpy.ts)
ts.plot(diff_jpy, "Exchange Rate")
```

```
## Warning in xy.coords(x = matrix(rep.int(tx, k), ncol = k), y = x, log = log, :
## NAs introduced by coercion
```

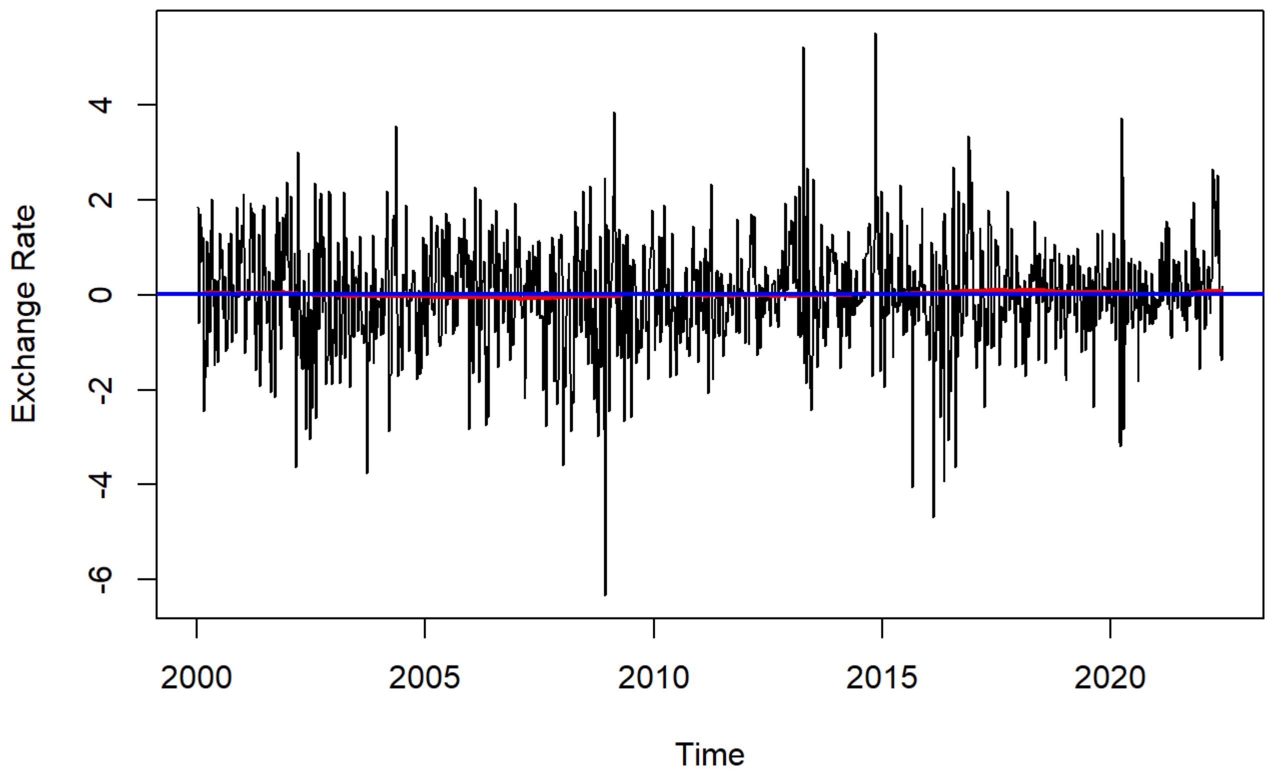
```
## Warning in xy.coords(x, y): NAs introduced by coercion
```



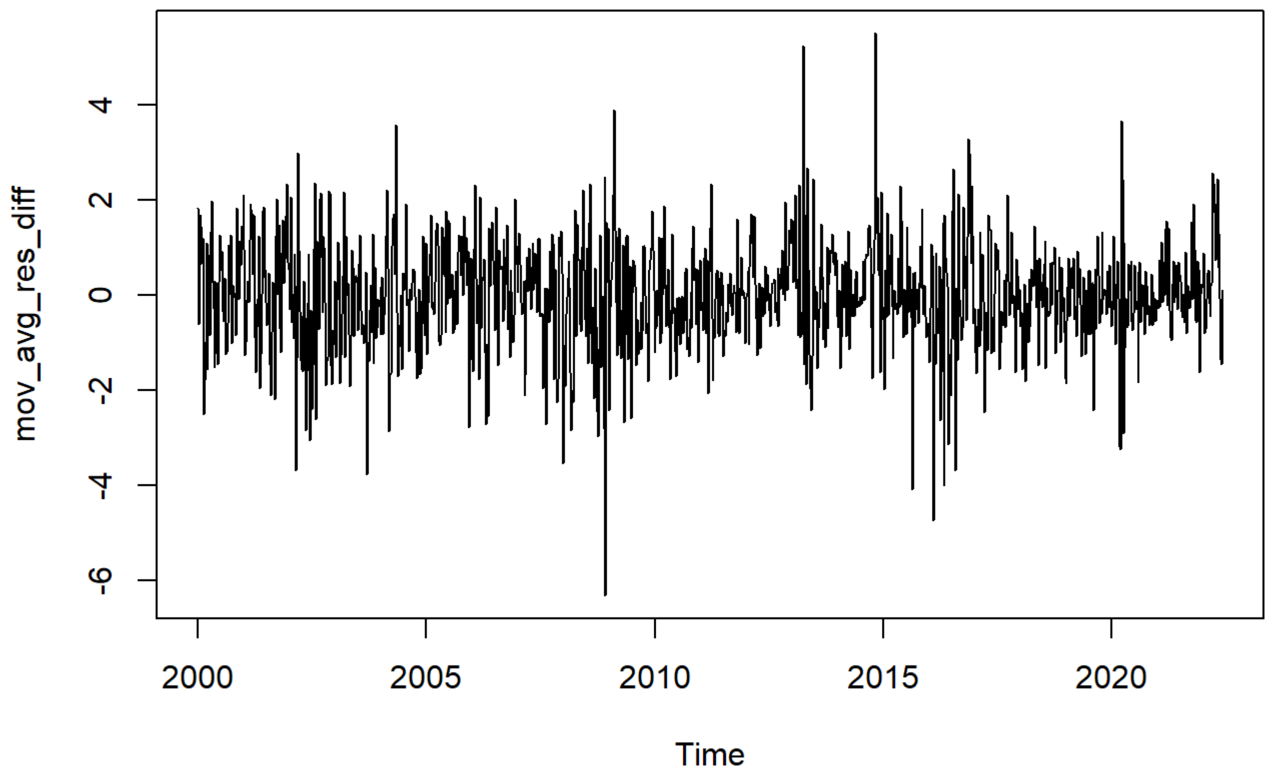
```
acf(diff_jpy, lag.max = 52*22)
```



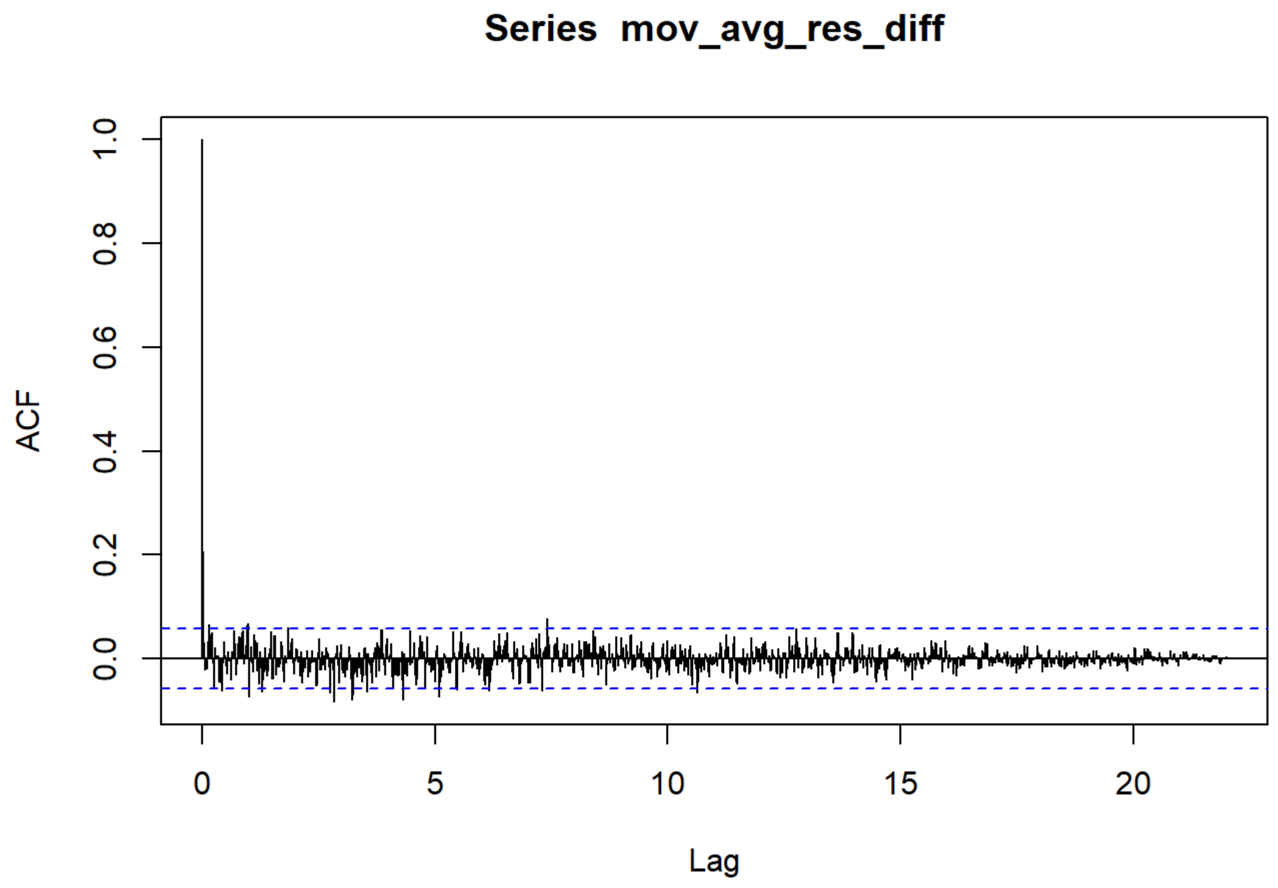
```
time_pts_diff = c(1:length(diff_jpy))
time_pts_diff = c(time_pts_diff-min(time_pts_diff))/max(time_pts_diff)

#moving average
mov_avg_diff = ksmooth(time_pts_diff,diff_jpy, kernel = "box")
mov_avg_fit_diff = ts(mov_avg_diff$y, start = 2000, frequency = 52)

plot(diff_jpy, ylab="Exchange Rate")
lines(mov_avg_fit_diff, lwd=2, col="red")
abline(mov_avg_fit_diff[1], 0, lwd=2, col="blue")
```



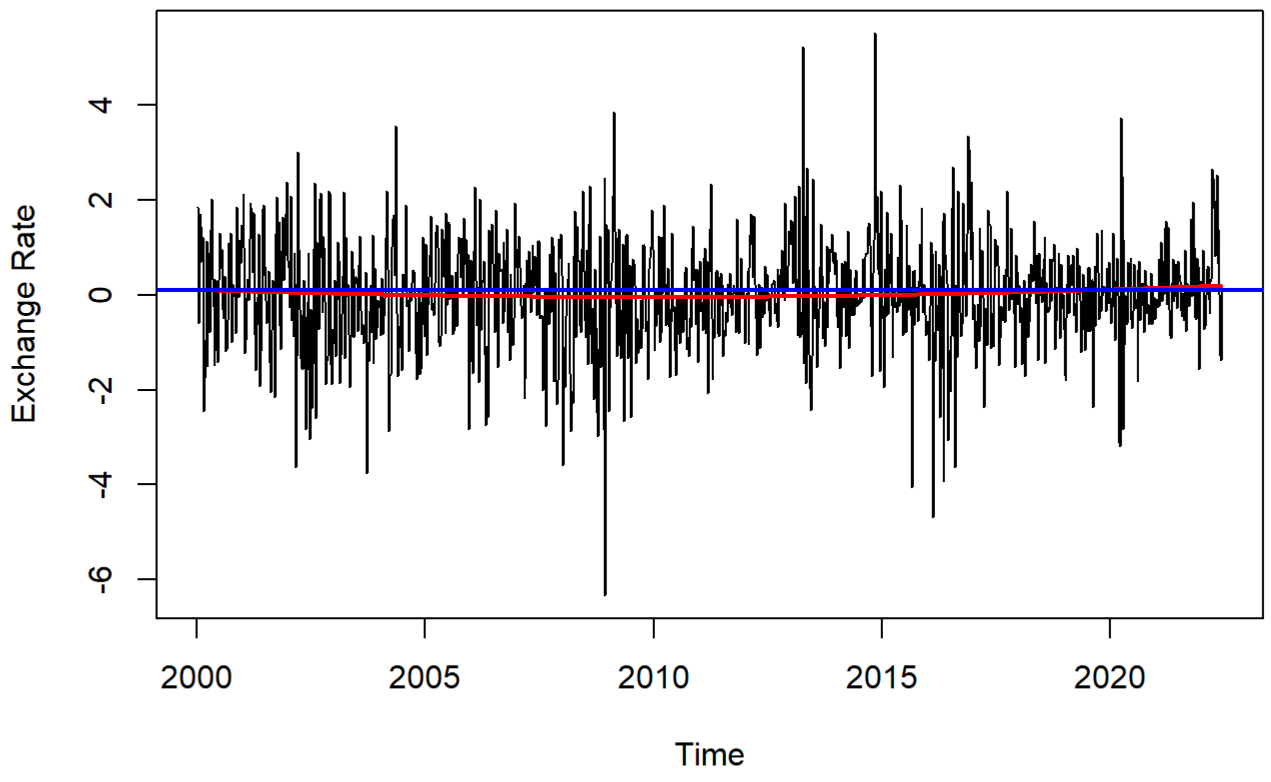
```
#residuals and ACF
mov_avg_res_diff = ts((diff_jpy-mov_avg_diff$y),start=2000,frequency=52)
plot(mov_avg_res_diff)
```



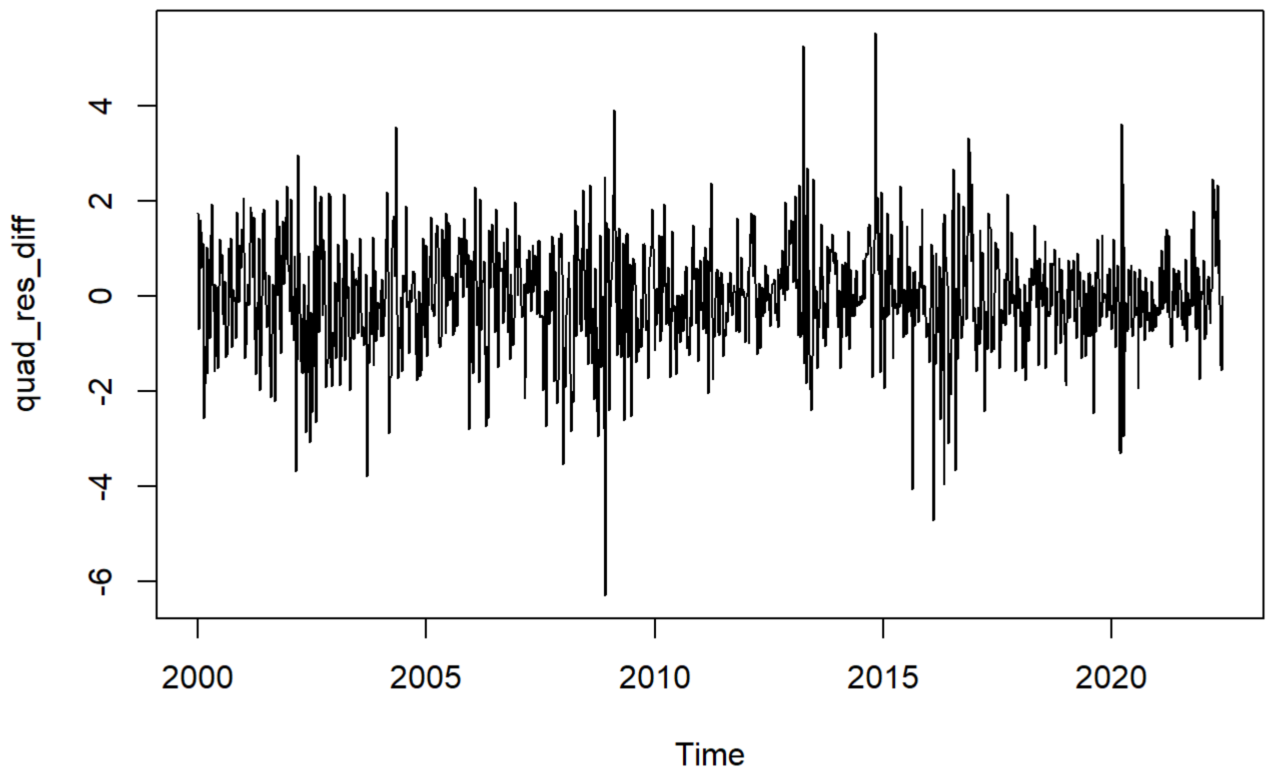
```
acf(mov_avg_res_diff, lag.max = 52*22)
```



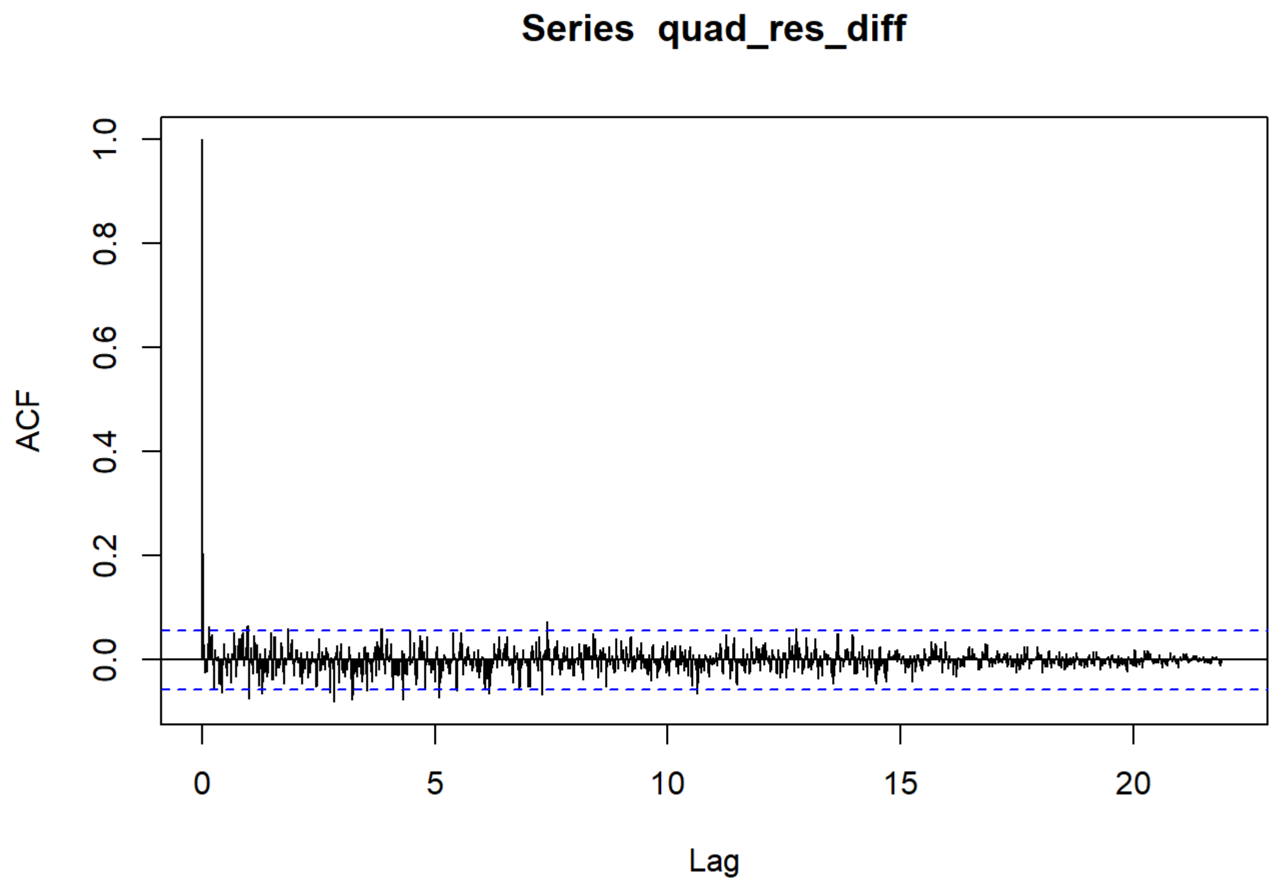
```
#Parametric Quadratic Polynomial
x1_diff = time_pts_diff
x2_diff = time_pts_diff^2
quad_diff = lm(diff_jpy ~ x1_diff+x2_diff)
quad_fit_diff = ts(fitted(quad_diff),start = 2000, frequency = 52)
plot(diff_jpy, ylab="Exchange Rate")
lines(quad_fit_diff, lwd=2, col="red")
abline(quad_fit_diff[1], 0, lwd=2, col="blue")
```



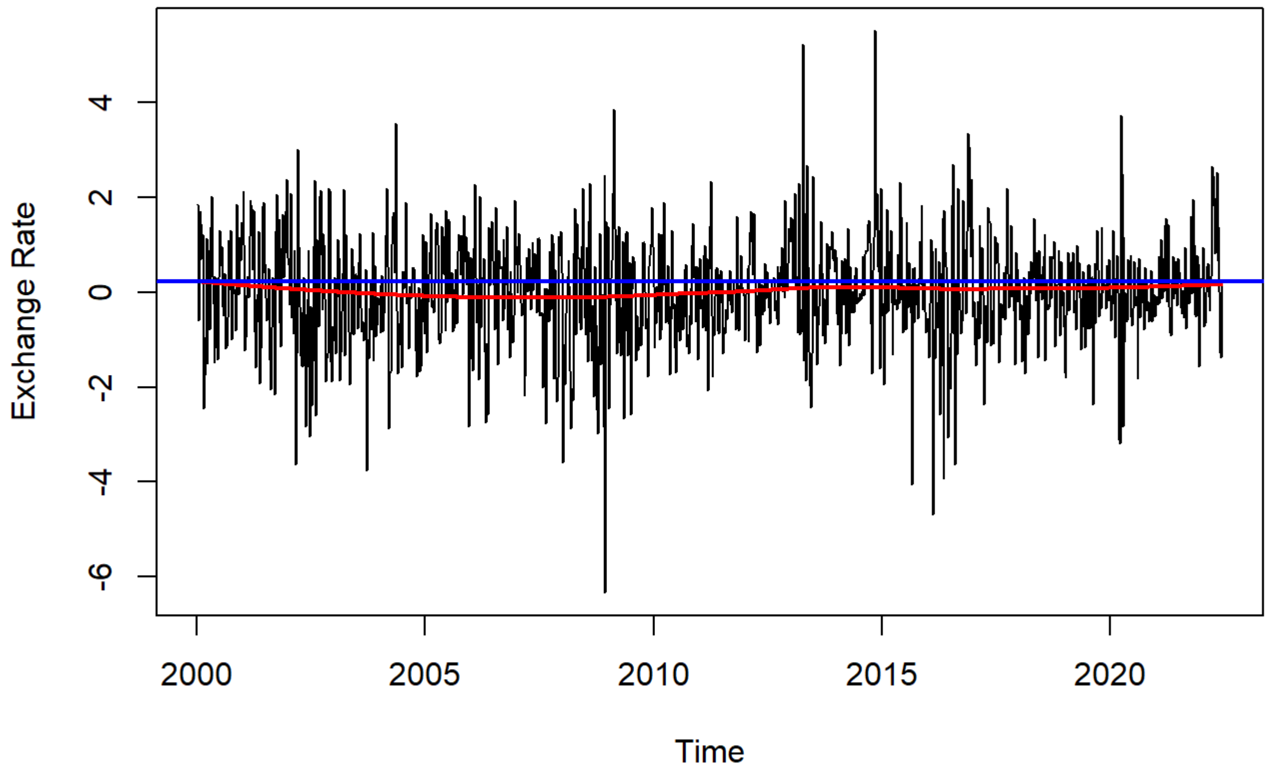
```
#residuals and ACF
quad_res_diff = ts((diff_jpy-fitted(quad_diff)),start=2000,frequency=52)
plot(quad_res_diff)
```



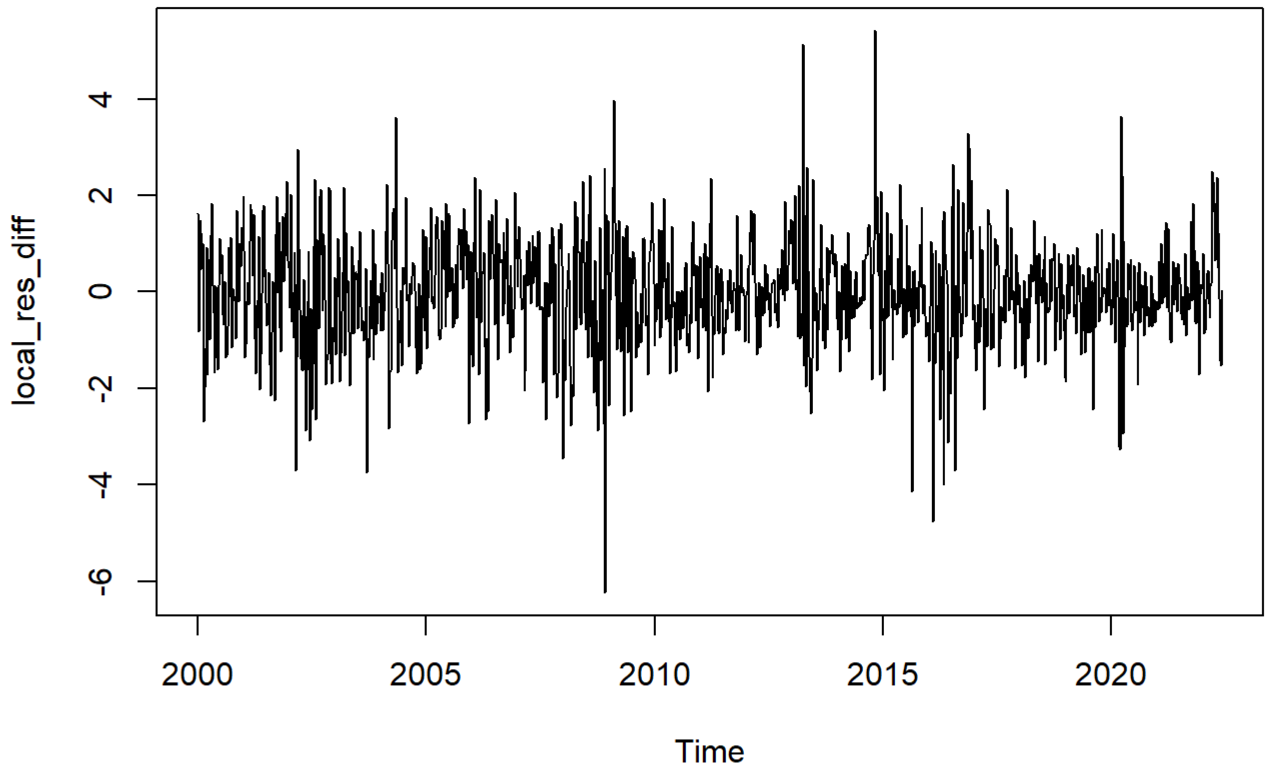
```
acf(quad_res_diff, lag.max = 52*22)
```



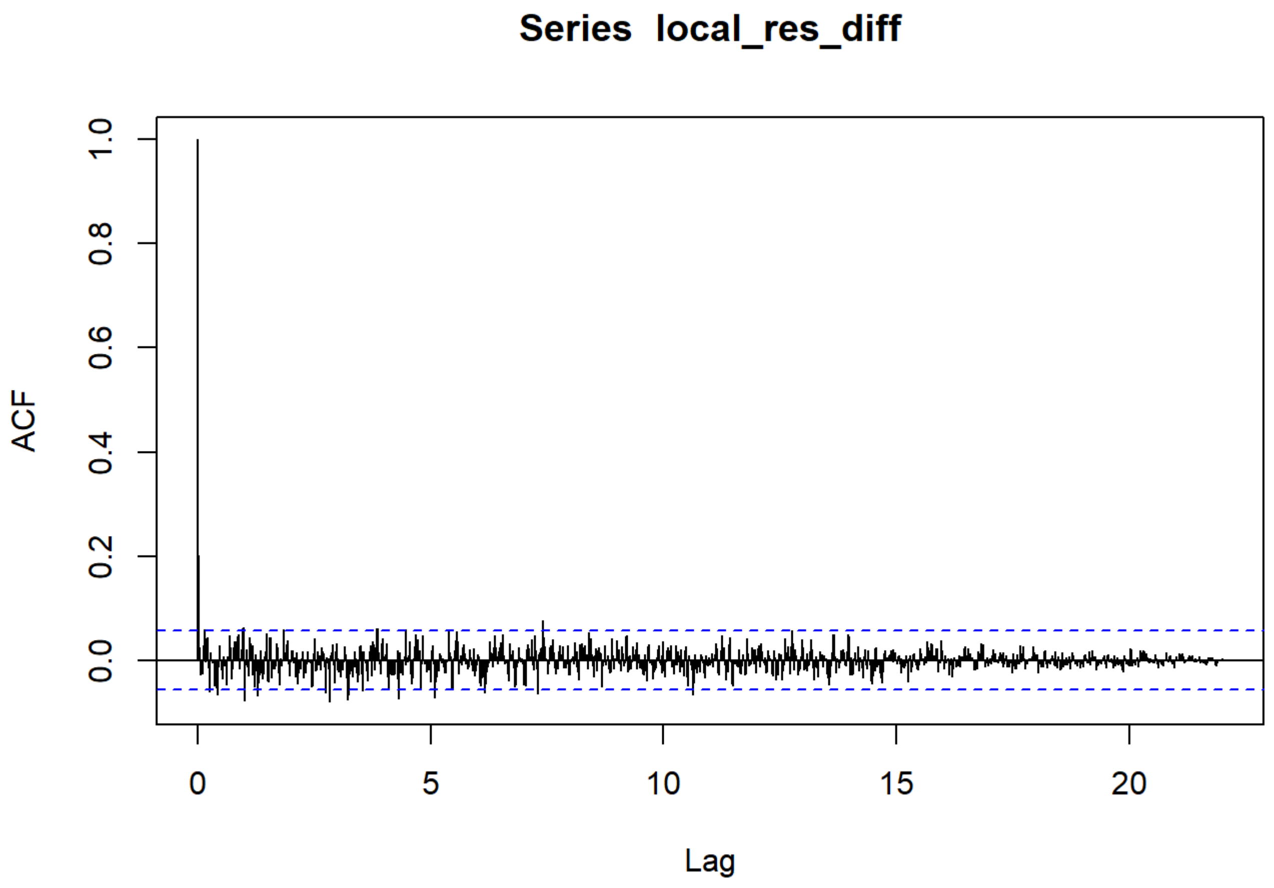
```
#Local Polynomial
local_diff = loess(diff_jpy~time_pts_diff)
local_fit_diff = ts(fitted(local_diff), start = 2000, frequency = 52)
plot(diff_jpy, ylab="Exchange Rate")
lines(local_fit_diff, lwd=2, col="red")
abline(local_fit_diff[1], 0, lwd=2, col="blue")
```



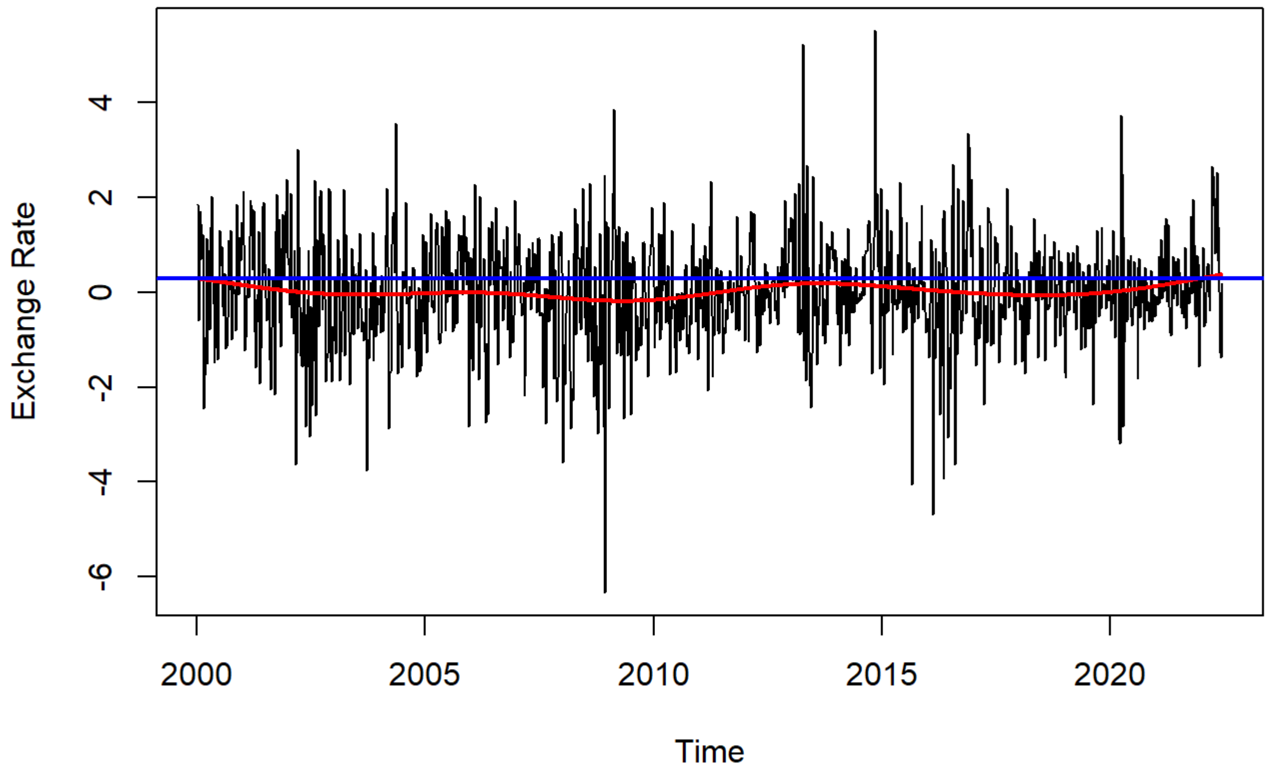
```
#residuals and ACF
local_res_diff = ts((diff_jpy-fitted(local_diff)),start=2000,frequency=52)
plot(local_res_diff)
```



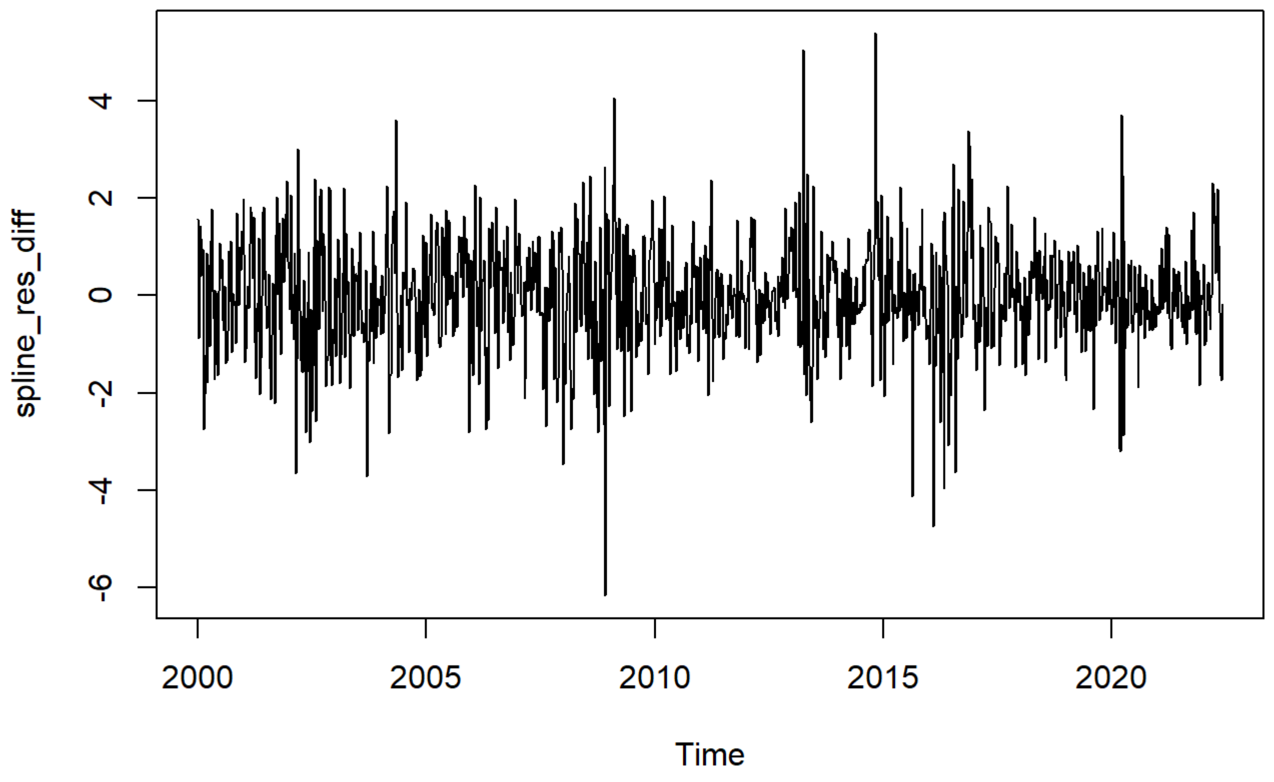
```
acf(local_res_diff, lag.max = 52*22)
```



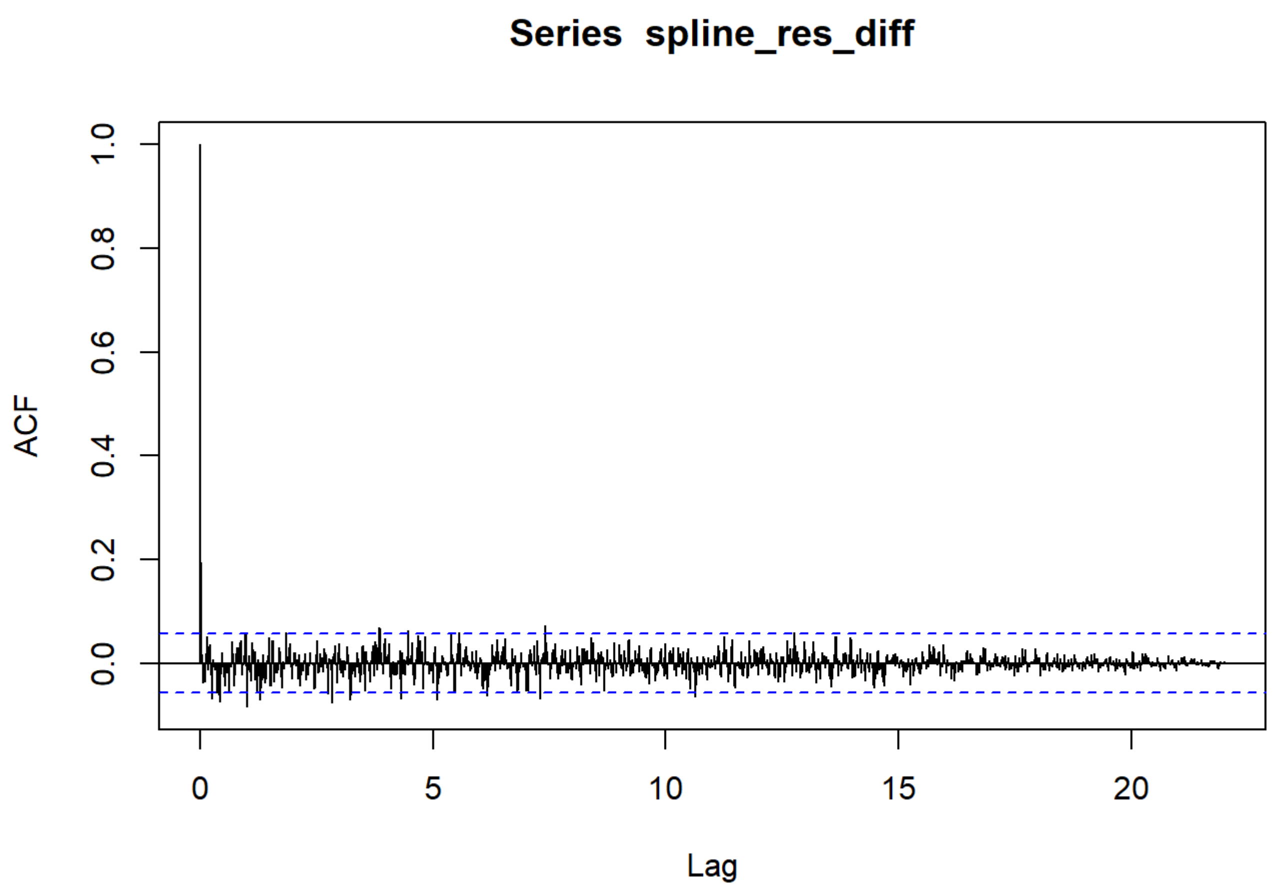
```r
#Spline
spl_diff = gam(diff_jpy~s(time_pts_diff))
spl_fit_diff = ts(fitted(spl_diff), start = 2000, frequency = 52)
plot(diff_jpy, ylab="Exchange Rate")
lines(spl_fit_diff, lwd=2, col="red")
abline(spl_fit_diff[1], 0, lwd=2, col="blue")
```



```
#residuals and ACF
spline_res_diff = ts((diff_jpy-fitted(spl_diff)),start=2000,frequency=52)
plot(spline_res_diff)
```



```
acf(spline_res_diff, lag.max = 52*22)
```



Response: Comments about the stationarity of the difference data: Differencing the data improved the stationarity assumptions. The model I recommend would be the trend removal with differencing. The ACF plots show that there is little autocorrelation. The mean and variance also got closer to constant.

Part 2: Temperature Analysis

Background

In this problem, we will analyze aggregated temperature data.

Data *Everest Temp Jan-Mar 2021.csv* contains the hourly average temperature at the Mount Everest Base Camp for the months of January to March 2021. Run the following code to prepare the data for analysis:

Instructions on reading the data

To read the data in `R`, save the file in your working directory (make sure you have changed the directory if different from the R working directory) and read the data using the `R` function `read.csv()`

You will perform the analysis and modelling on the `Temp` data column.

```
fpath <- "Everest Temp Jan-Mar 2021.csv"
df <- read.csv(fpath, head = TRUE)
```

Here are the libraries you will need:

```
library(mgcv)
library(TSA)
```

```
## Warning: package 'TSA' was built under R version 4.1.3
```

```
##
## Attaching package: 'TSA'
```

```
## The following objects are masked from 'package:stats':
##
##     acf, arima
```

```
## The following object is masked from 'package:utils':
##
##     tar
```

```
library(dynlm)
```

```
## Warning: package 'dynlm' was built under R version 4.1.3
```

```
## Loading required package: zoo
```

```
## Warning: package 'zoo' was built under R version 4.1.3
```

```
##
## Attaching package: 'zoo'
```

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

Run the following code to prepare the data for analysis:

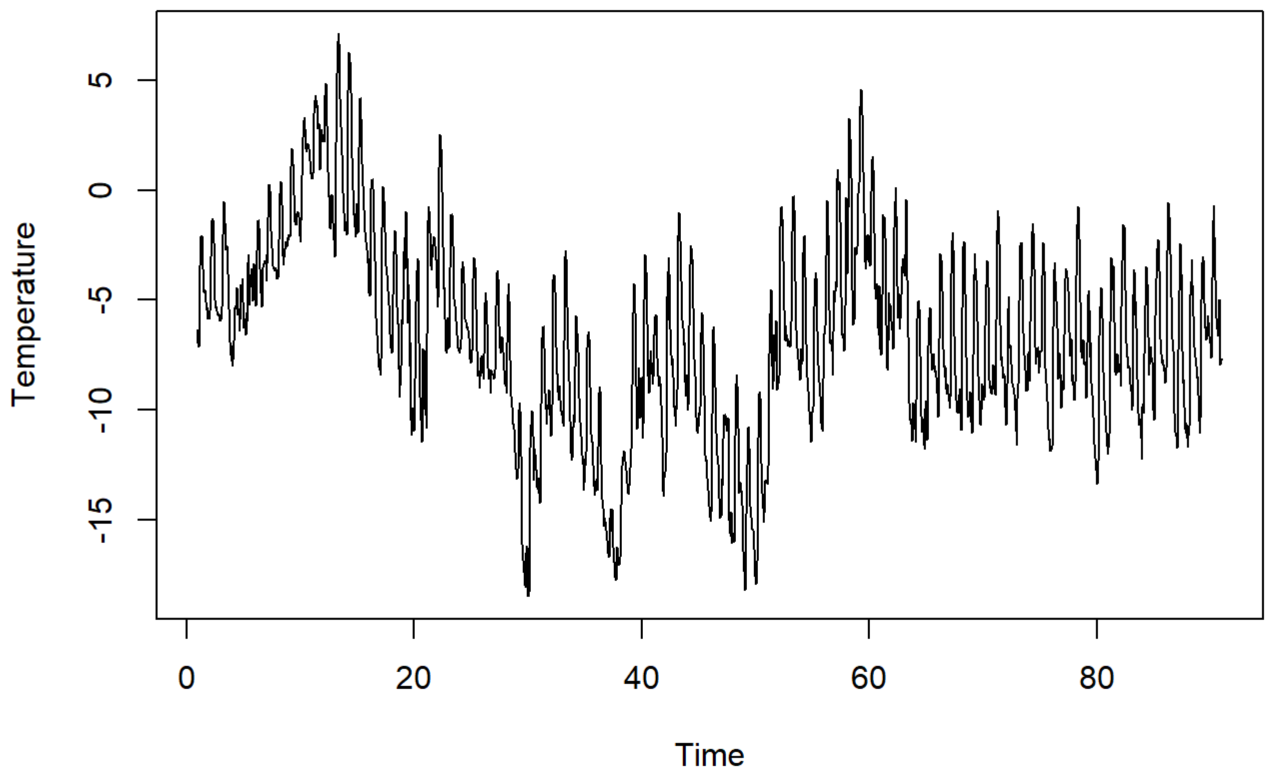
```
df$timestamp<-ymd_hms(df$timestamp)
temp <- ts(df$temp, freq = 24)

datetime<-ts(df$timestamp)
```

Question 2a: Exploratory Data Analysis

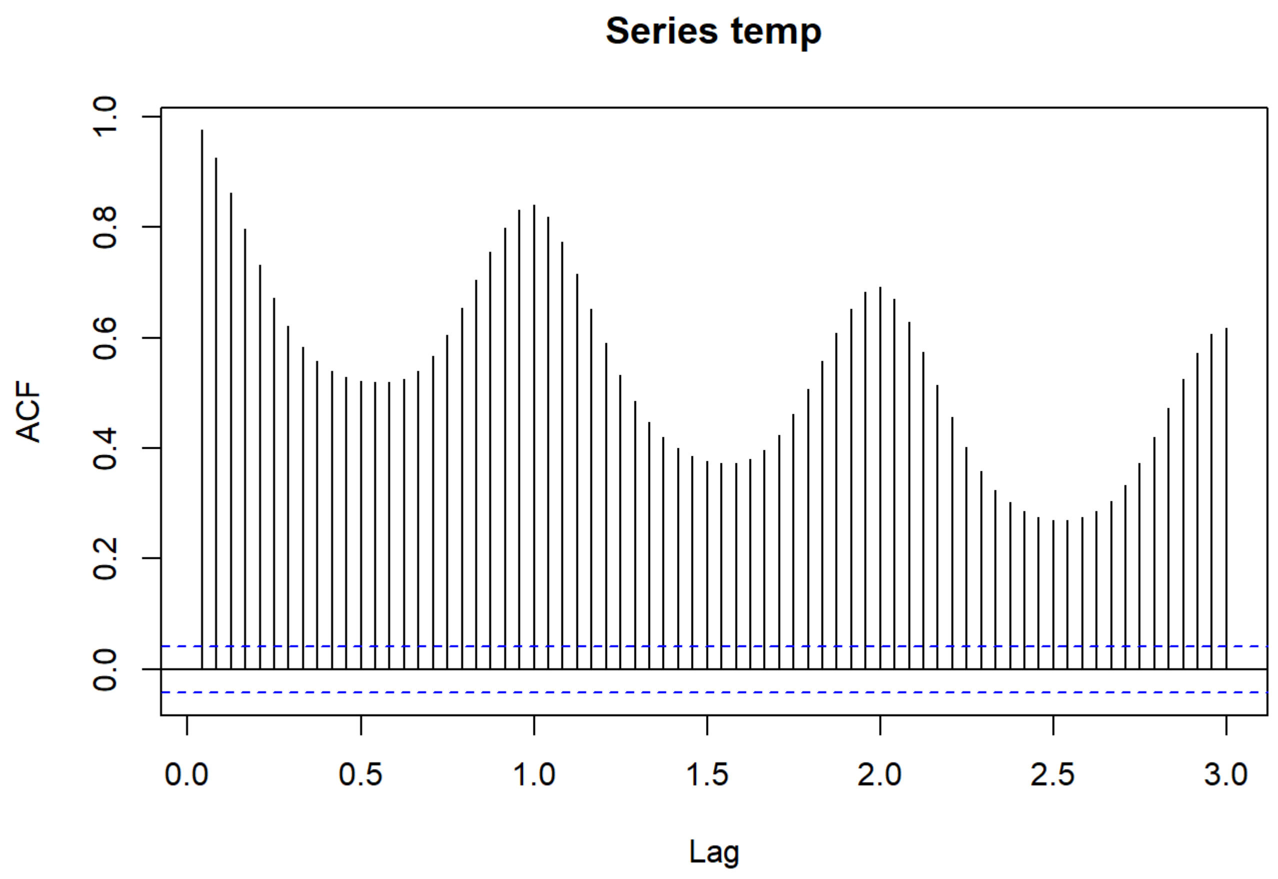
Plot both the Time Series and ACF plots. Comment on the main features, and identify what (if any) assumptions of stationarity are violated. Additionally, comment if you believe the differenced data is more appropriate for use in fitting the data. Support your response with a graphical analysis.

Hint: Make sure to use the appropriate differenced data.

```
ts.plot(temp,ylab="Temperature")
```



```
acf(temp, lag.max = 24*3)
```

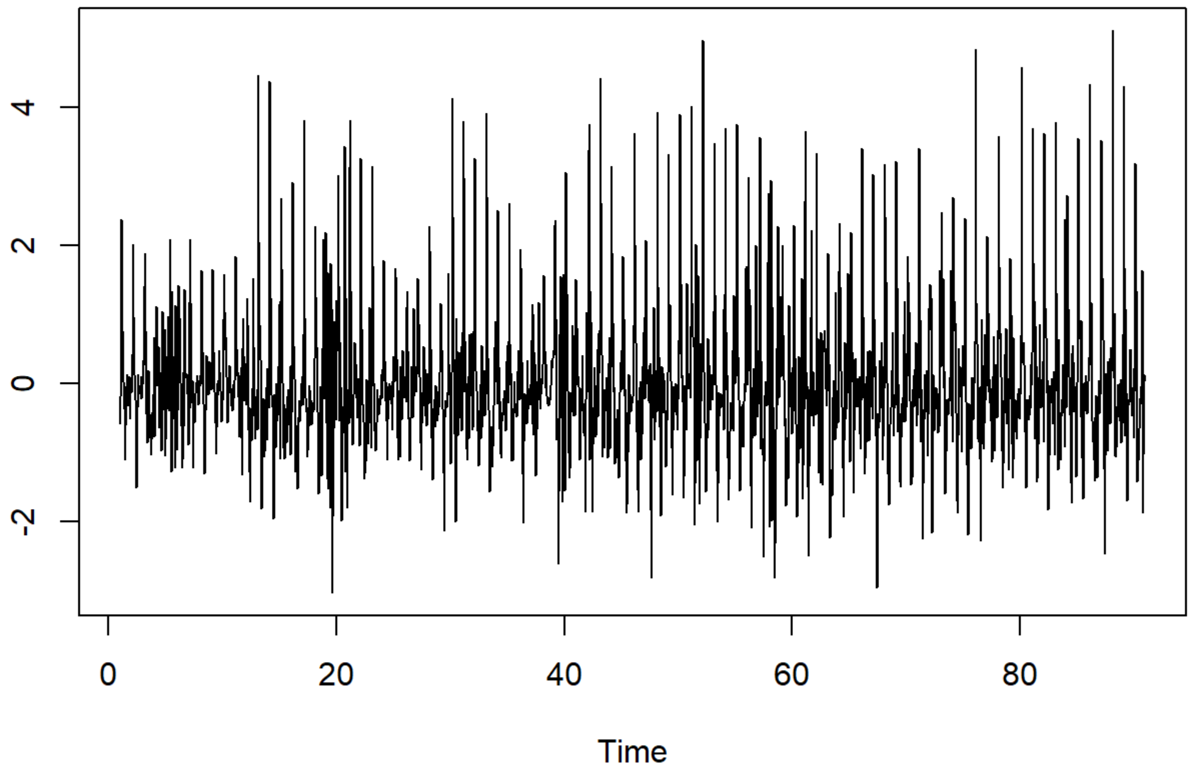


Response: Comments about the time series and ACF plots of the original time series Time series and ACF plots show that the series has seasonality. The constant mean and constant variation assumptions for stationarity are violated. Autocorrelation is also present.

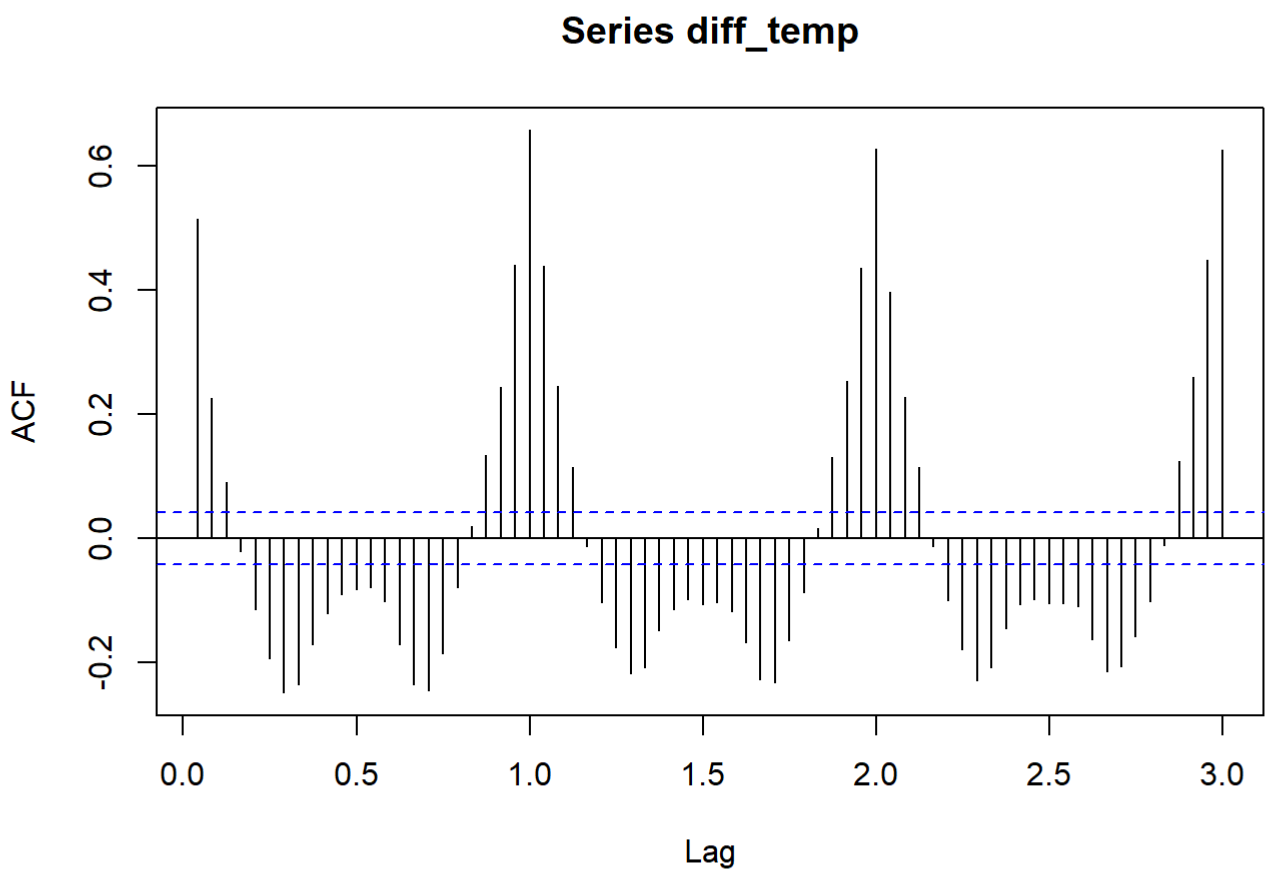
```
diff_temp = diff(temp)
ts.plot(diff_temp, "Temperature")
```

```
## Warning in xy.coords(x = matrix(rep.int(tx, k), ncol = k), y = x, log = log, :
## NAs introduced by coercion
```

```
## Warning in xy.coords(x, y): NAs introduced by coercion
```



```
acf(diff_temp, lag.max = 24*3)
```



Response: Comments about the time series and ACF plots of the difference time series The difference time series is more appropriate to use for fitting the data. Constant mean seems to be achieved. Constant variance is a still a problem. There is still autocorrelation present.

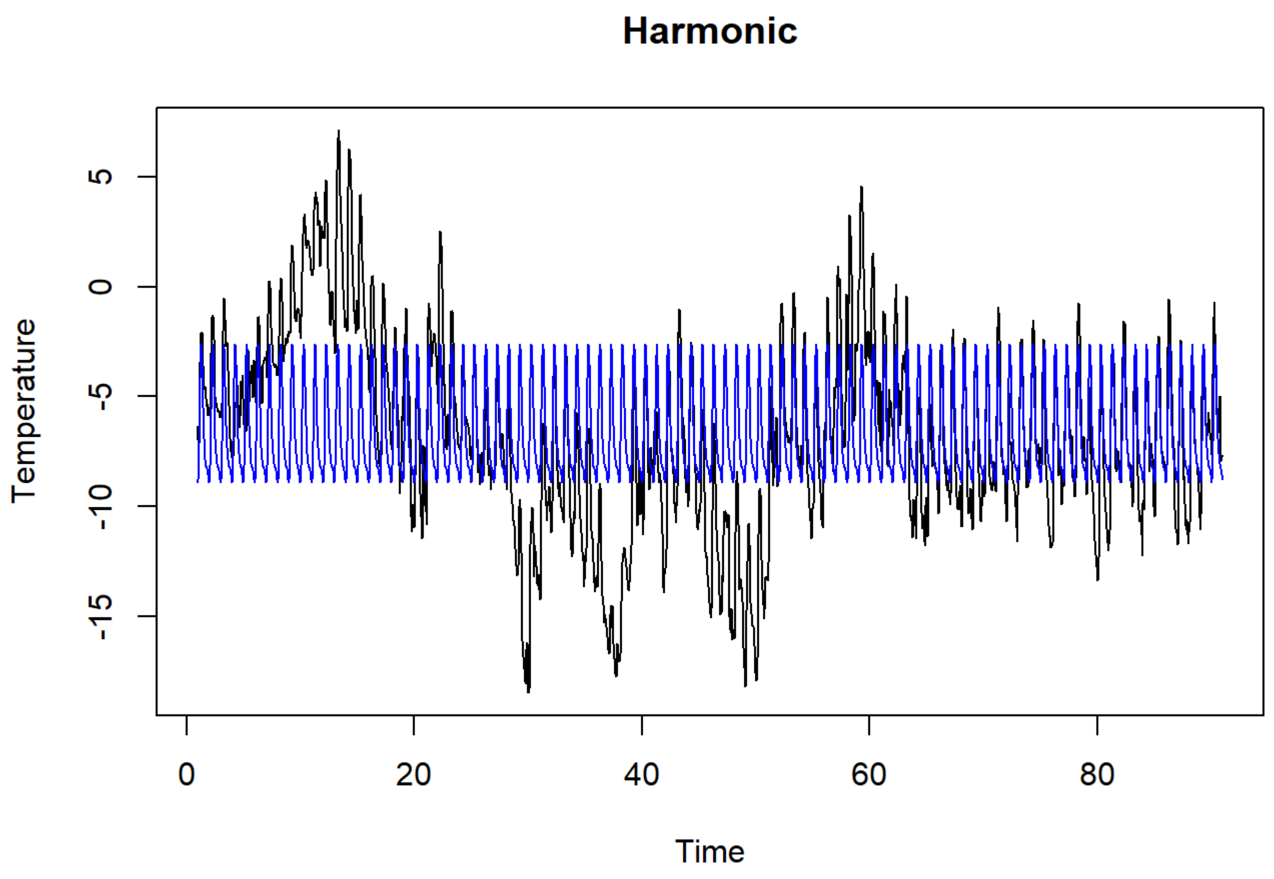
Question 2b: Seasonality Estimation

Separately fit a seasonality harmonic model and the ANOVA seasonality model to the temperature data. Evaluate the quality of each fit with residual analysis. Does one model perform better than the other? Which model would you select to fit the seasonality in the data?

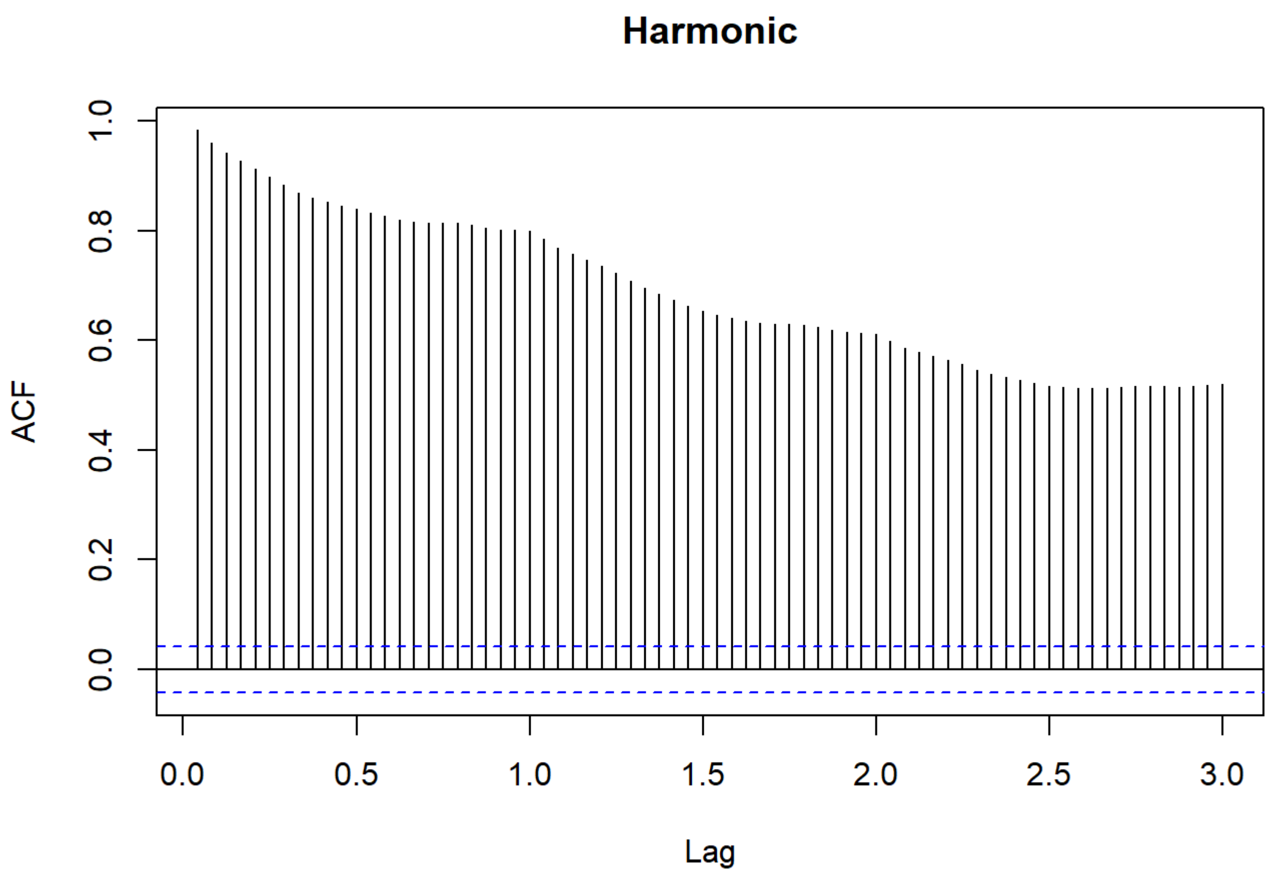
```
#harmonic model
harmon_model = dynlm(temp~harmon(temp,3))
summary(harmon_model)
```

```
##
## Time series regression with "ts" data:
## Start = 1(1), End = 90(24)
##
## Call:
## dynlm(formula = temp ~ harmon(temp, 3))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -12.454  -2.185  -0.051   2.008  11.282
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)          -6.62044    0.08545 -77.478  < 2e-16 ***
## harmon(temp, 3)cos1 -1.40540    0.12084 -11.630  < 2e-16 ***
## harmon(temp, 3)cos2 -0.99104    0.12084  -8.201 4.05e-16 ***
## harmon(temp, 3)cos3  0.09684    0.12084   0.801  0.42300
## harmon(temp, 3)sin1  2.22315    0.12084  18.397  < 2e-16 ***
## harmon(temp, 3)sin2 -0.68552    0.12084  -5.673 1.59e-08 ***
## harmon(temp, 3)sin3 -0.31471    0.12084  -2.604  0.00927 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.971 on 2153 degrees of freedom
## Multiple R-squared:  0.2124, Adjusted R-squared:  0.2102
## F-statistic: 96.76 on 6 and 2153 DF,  p-value: < 2.2e-16
```

```
ts.plot(temp,ylab="Temperature", main = "Harmonic")
lines(harmon_model$fitted,col="blue")
```



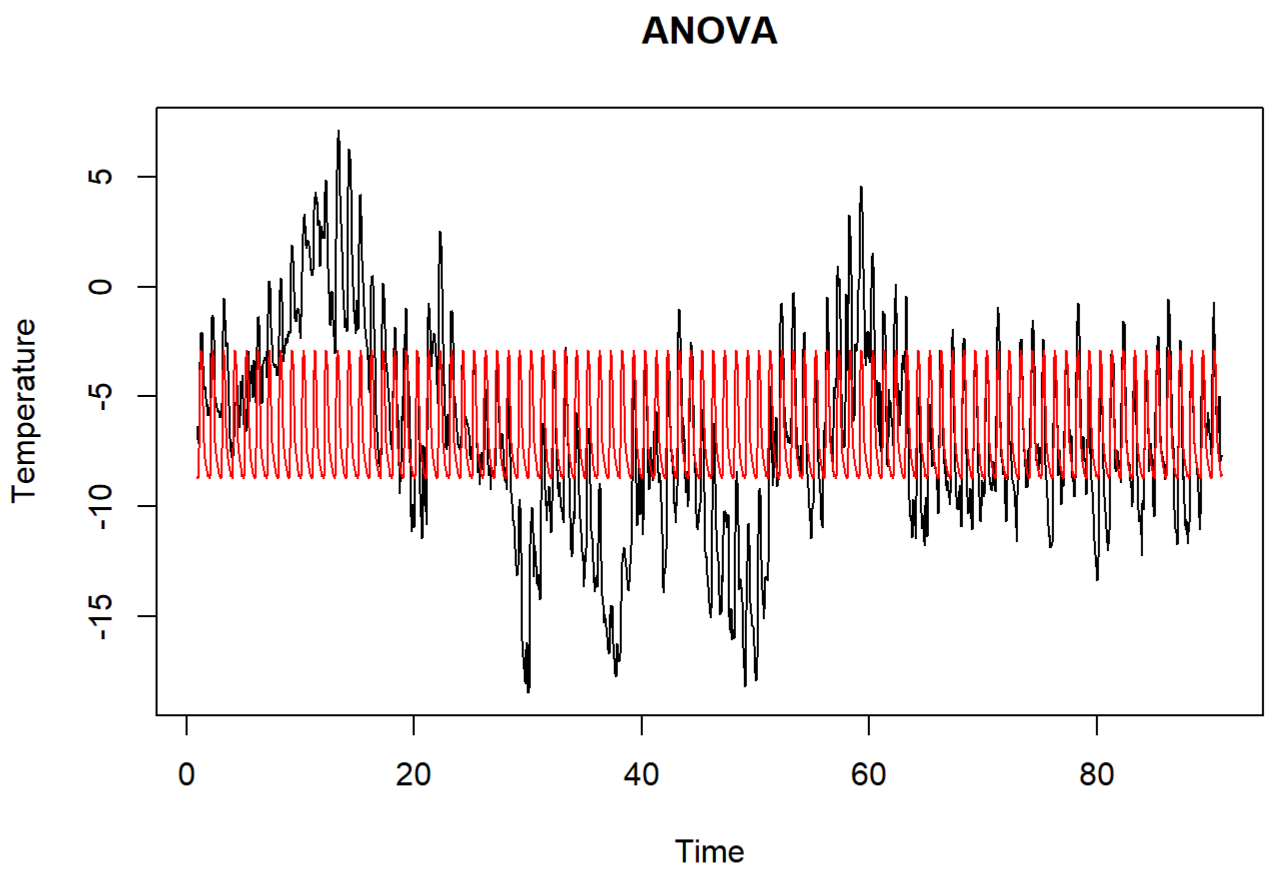
```
res_harmon = residuals(harmon_model)
acf(res_harmon, main = "Harmonic", lag.max = 24*3)
```



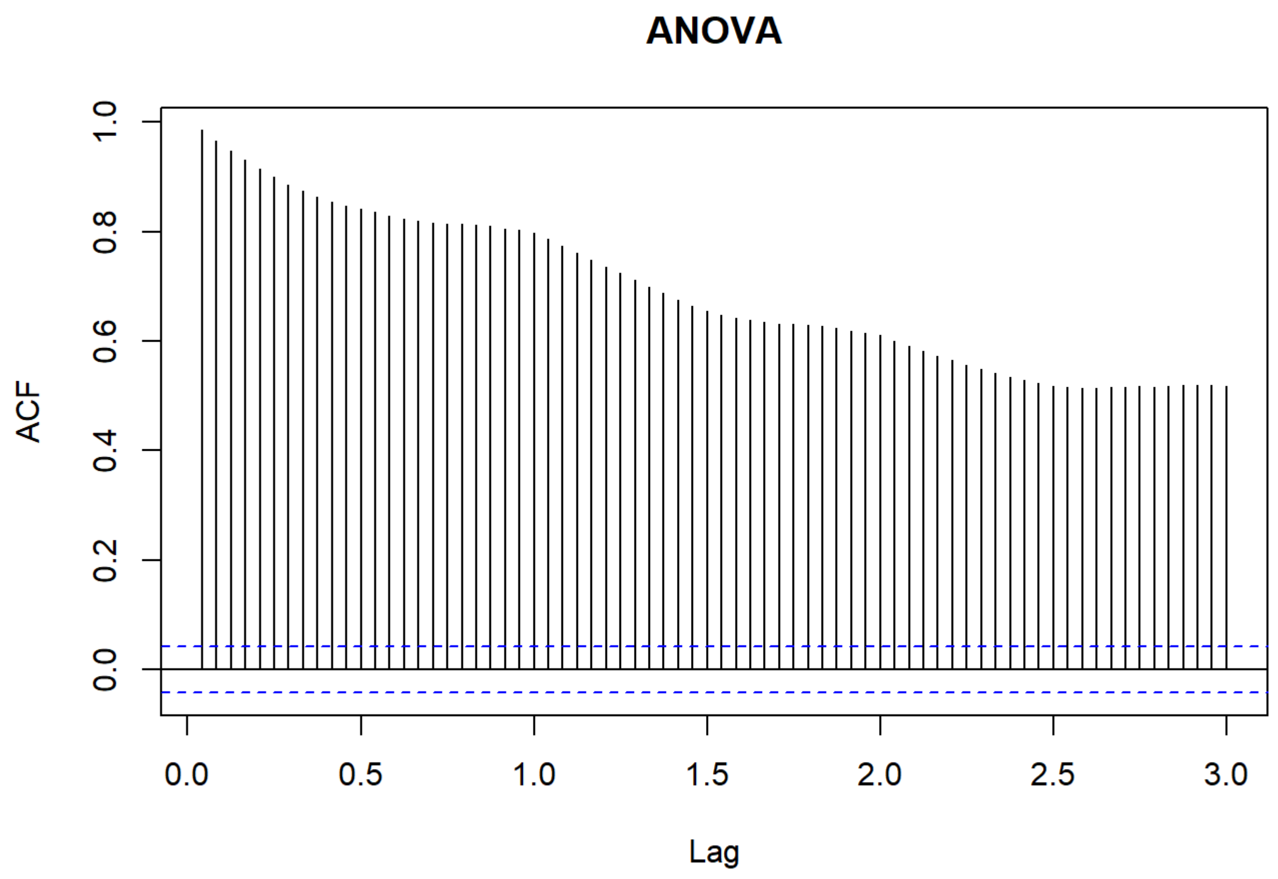
```
#ANOVA seasonality model
anova_model = dynlm(temp~season(temp)-1)
summary(anova_model)
```

```
##
## Time series regression with "ts" data:
## Start = 1(1), End = 90(24)
##
## Call:
## dynlm(formula = temp ~ season(temp) - 1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.3218  -2.1758  -0.0539   2.0218  11.2616
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## season(temp)1   -8.6377     0.4196 -20.586  < 2e-16 ***
## season(temp)2   -8.7277     0.4196 -20.800  < 2e-16 ***
## season(temp)3   -8.6304     0.4196 -20.569  < 2e-16 ***
## season(temp)4   -7.3675     0.4196 -17.559  < 2e-16 ***
## season(temp)5   -5.0811     0.4196 -12.110  < 2e-16 ***
## season(temp)6   -3.7974     0.4196  -9.050  < 2e-16 ***
## season(temp)7   -3.1982     0.4196  -7.622 3.73e-14 ***
## season(temp)8   -2.8833     0.4196  -6.872 8.30e-12 ***
## season(temp)9   -3.0570     0.4196  -7.286 4.48e-13 ***
## season(temp)10  -3.4577     0.4196  -8.241 2.95e-16 ***
## season(temp)11  -4.2223     0.4196 -10.063  < 2e-16 ***
## season(temp)12  -5.3755     0.4196 -12.811  < 2e-16 ***
## season(temp)13  -6.3816     0.4196 -15.209  < 2e-16 ***
## season(temp)14  -7.0679     0.4196 -16.845  < 2e-16 ***
## season(temp)15  -7.3277     0.4196 -17.464  < 2e-16 ***
## season(temp)16  -7.5673     0.4196 -18.035  < 2e-16 ***
## season(temp)17  -7.8190     0.4196 -18.635  < 2e-16 ***
## season(temp)18  -7.9359     0.4196 -18.914  < 2e-16 ***
## season(temp)19  -8.1156     0.4196 -19.342  < 2e-16 ***
## season(temp)20  -8.2290     0.4196 -19.612  < 2e-16 ***
## season(temp)21  -8.3712     0.4196 -19.951  < 2e-16 ***
## season(temp)22  -8.4796     0.4196 -20.209  < 2e-16 ***
## season(temp)23  -8.6059     0.4196 -20.510  < 2e-16 ***
## season(temp)24  -8.5544     0.4196 -20.387  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.981 on 2136 degrees of freedom
## Multiple R-squared:  0.7544, Adjusted R-squared:  0.7516
## F-statistic: 273.3 on 24 and 2136 DF,  p-value: < 2.2e-16
```

```
ts.plot(temp,ylab="Temperature", main = "ANOVA")
lines(ts(anova_model$fitted,start=1,frequency=24),col="red")
```



```
res_anova = residuals(anova_model)
acf(res_anova, main = "ANOVA", lag.max = 24*3)
```



Response: Compare Seasonality Models Based on the summaries, the ANOVA model performs better with a higher Adjusted R-squared. The ACF of the residuals has similar behavior. Therefore, I would pick the ANOVA model.

Question 2c: Trend-Seasonality Estimation

Using the time series data, fit the following models to estimate the trend with seasonality fitted using ANOVA:

- Parametric Polynomial Regression
- Non-parametric model

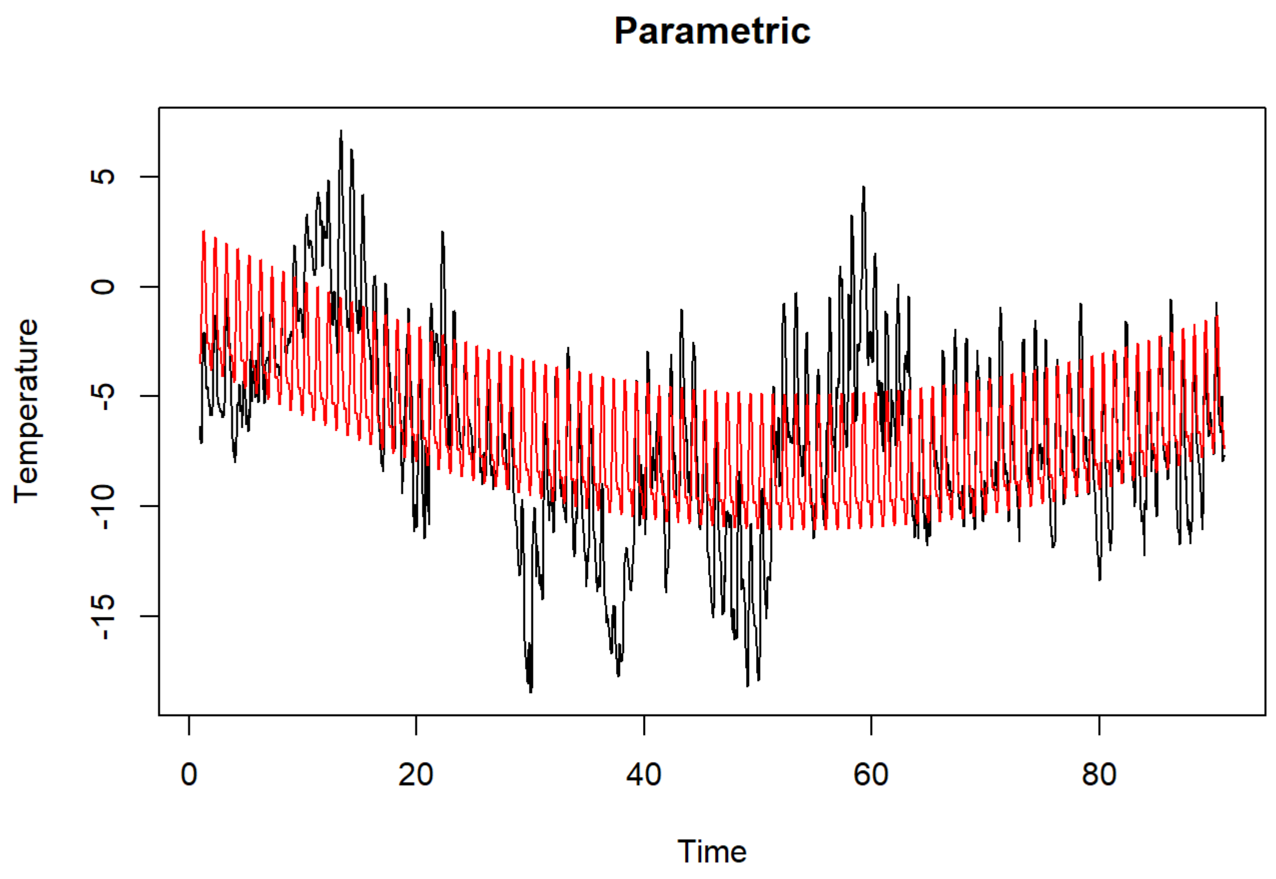
Overlay the fitted values on the original time series. Plot the residuals with respect to time. Plot the ACF of the residuals. Comment on how the two models fit and on the appropriateness of the stationarity assumption of the residuals.

What form of modelling seems most appropriate and what implications might this have for how one might expect long term temperature data to behave? Provide explicit conclusions based on the data analysis.

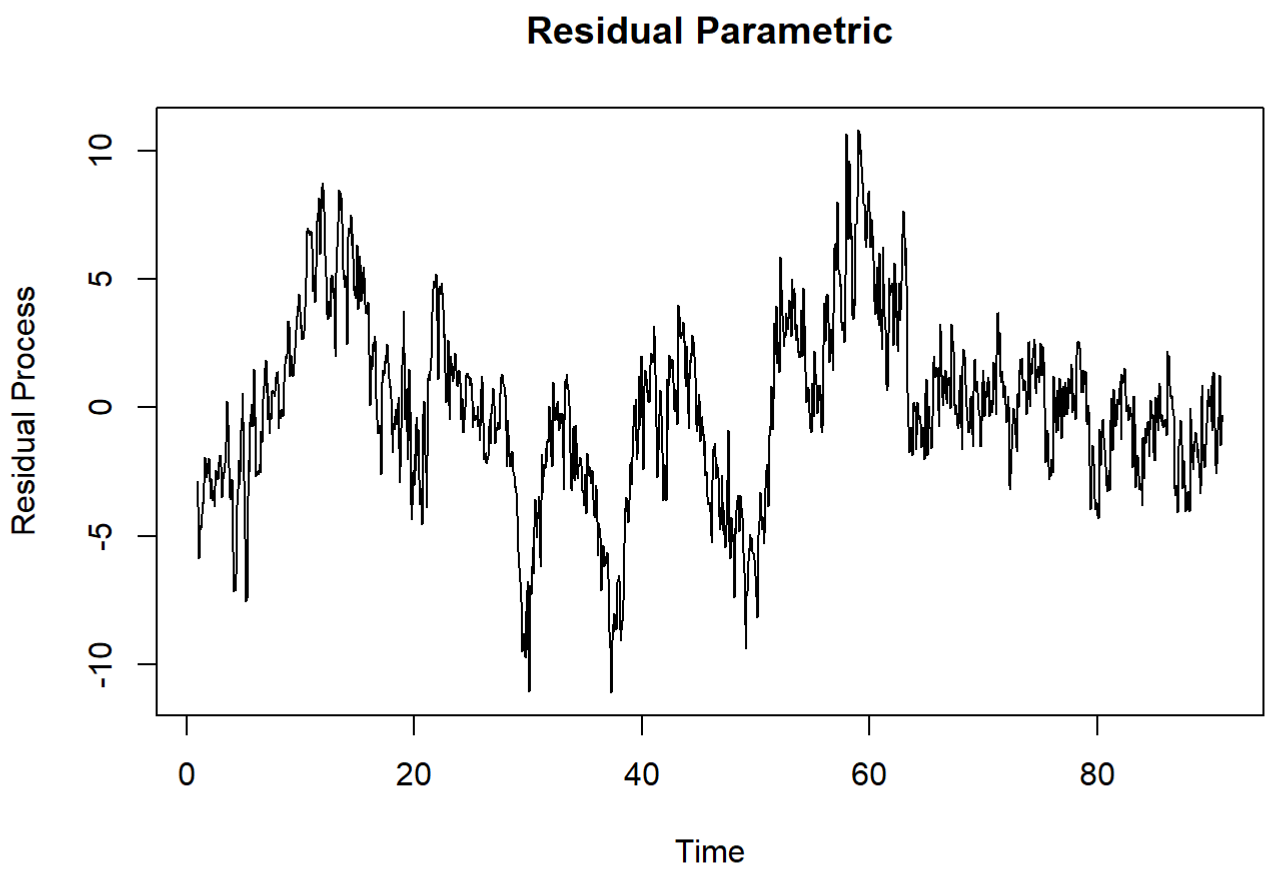
```
#parametric
time_pts2 = c(1:length(temp))
x1 = time_pts2
x2 = time_pts2^2
lm.fit = dynlm(temp~x1+x2+harmon(temp,2))
summary(lm.fit)
```

```
##
## Time series regression with "ts" data:
## Start = 1(1), End = 90(24)
##
## Call:
## dynlm(formula = temp ~ x1 + x2 + harmon(temp, 2))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.1092  -2.0899  -0.0632   1.8877  10.7950
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)         -1.059e+00  2.233e-01  -4.743 2.24e-06 ***
## x1                  -1.187e-02  4.772e-04 -24.873  < 2e-16 ***
## x2                   4.667e-06  2.138e-07  21.828  < 2e-16 ***
## harmon(temp, 2)cos1 -1.407e+00  1.052e-01 -13.384  < 2e-16 ***
## harmon(temp, 2)cos2 -9.929e-01  1.052e-01  -9.442  < 2e-16 ***
## harmon(temp, 2)sin1  2.210e+00  1.052e-01  21.012  < 2e-16 ***
## harmon(temp, 2)sin2 -6.922e-01  1.052e-01  -6.582 5.81e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.456 on 2153 degrees of freedom
## Multiple R-squared:  0.4036, Adjusted R-squared:  0.4019
## F-statistic: 242.8 on 6 and 2153 DF,  p-value: < 2.2e-16
```

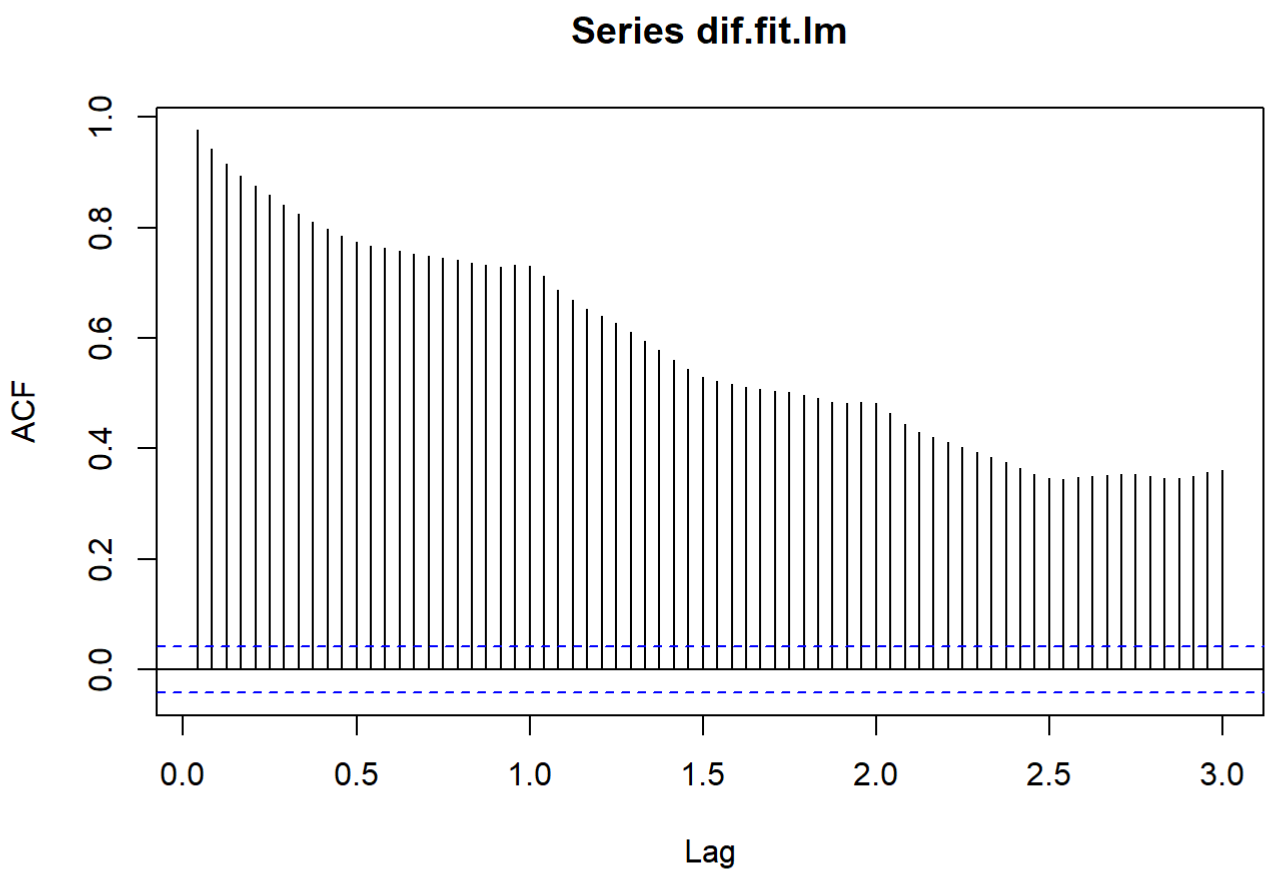
```
ts.plot(temp,ylab="Temperature", main = "Parametric")
lines(ts(lm.fit$fitted,start=1,frequency=24),col="red")
```



```
dif.fit.lm = ts((temp-fitted(lm.fit)),start=1,frequency=24)
ts.plot(dif.fit.lm,ylab="Residual Process", main = "Residual Parametric")
```



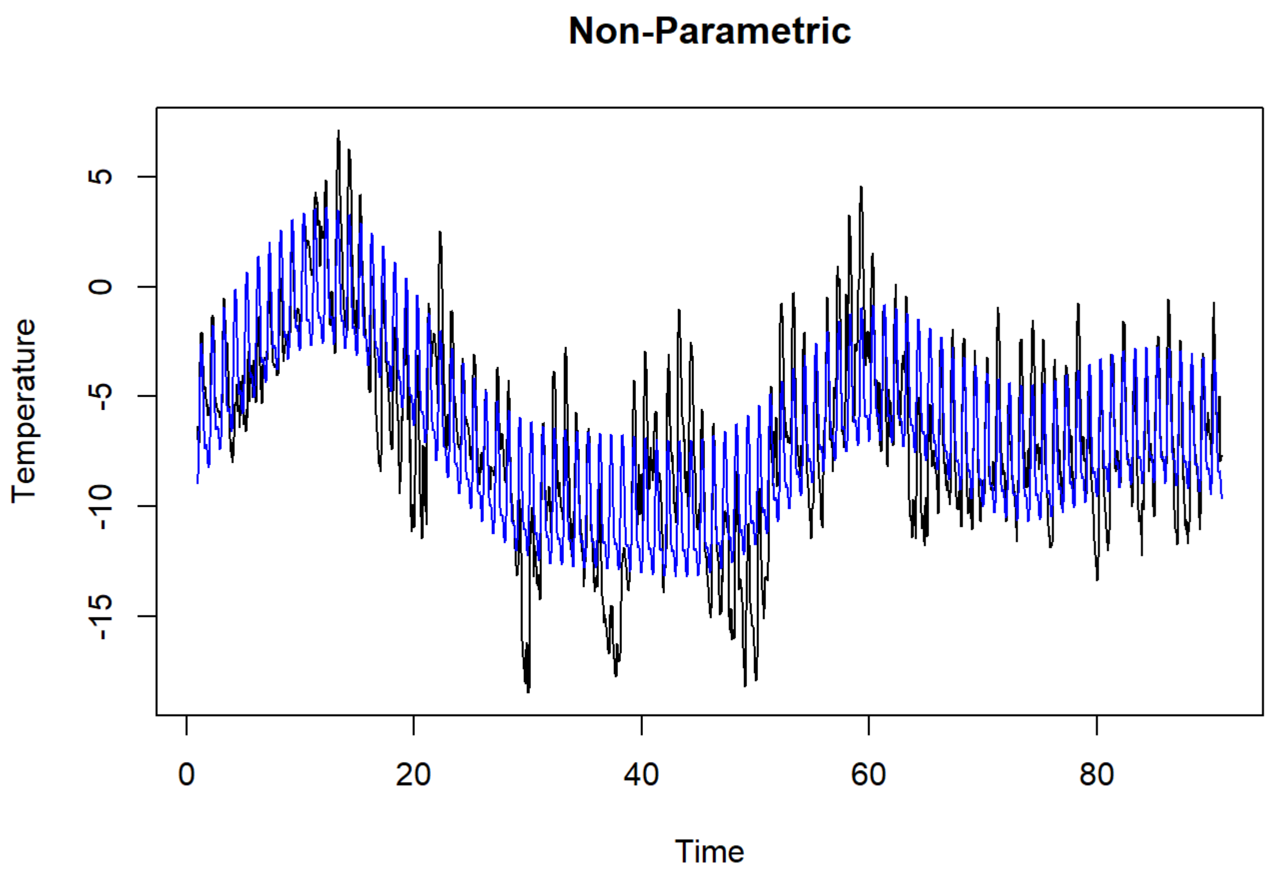
```
acf(dif.fit.lm, lag.max = 24*3)
```



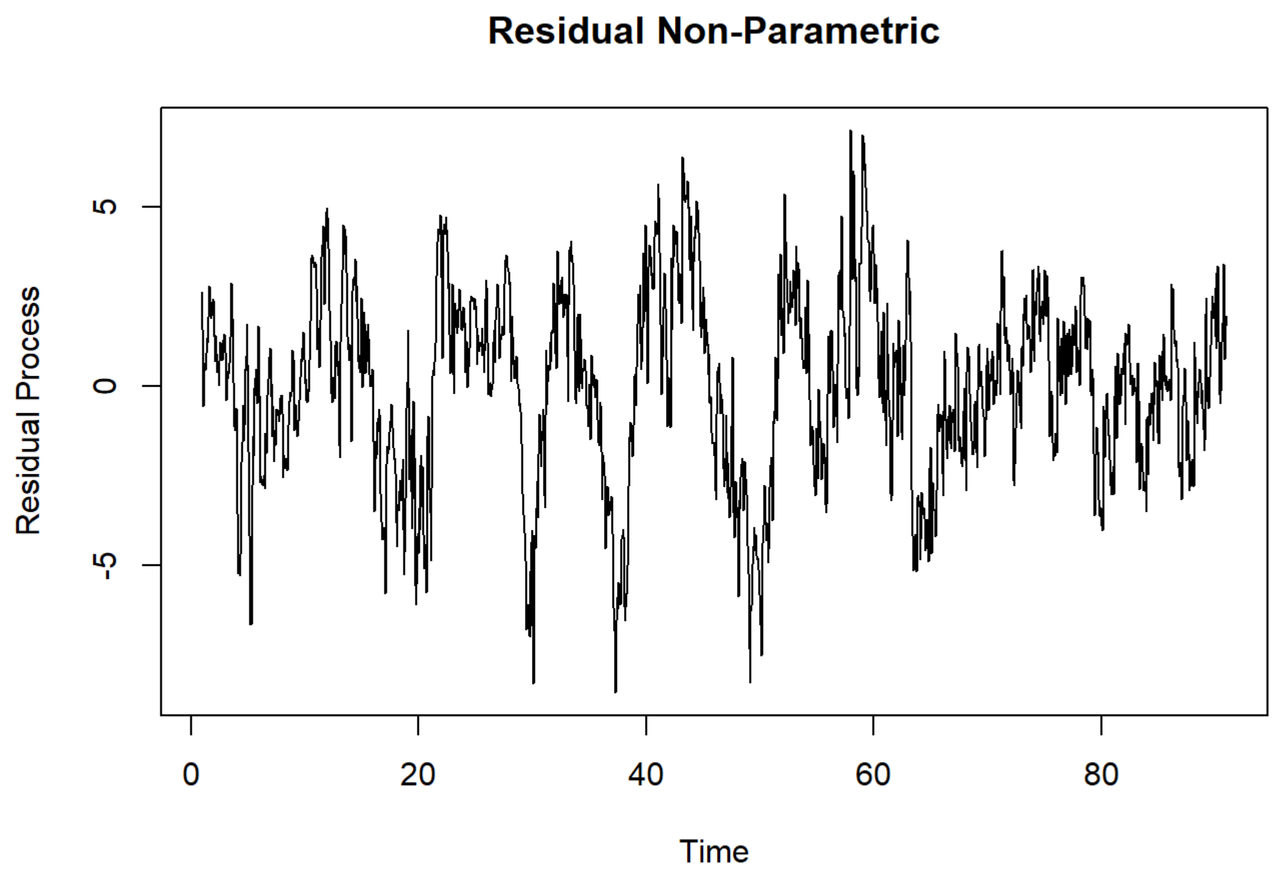
```
#non-parametric
har2 = harmonic(temp,2)
gam.fit = gam(temp~s(time_pts2)+har2)
summary(gam.fit)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## temp ~ s(time_pts2) + har2
##
## Parametric coefficients:
##                 Estimate Std. Error  t value Pr(>|t|)
## (Intercept)     -6.62044    0.05534 -119.632   <2e-16 ***
## har2cos(2*pi*t) -1.40512    0.07826  -17.954   <2e-16 ***
## har2cos(4*pi*t) -0.99119    0.07826  -12.665   <2e-16 ***
## har2sin(2*pi*t)  2.22097    0.07828   28.372   <2e-16 ***
## har2sin(4*pi*t) -0.68659    0.07827   -8.772   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##               edf Ref.df   F p-value
## s(time_pts2) 8.946  8.999 333  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.669   Deviance explained = 67.1%
## GCV = 6.6581  Scale est. = 6.6151    n = 2160
```

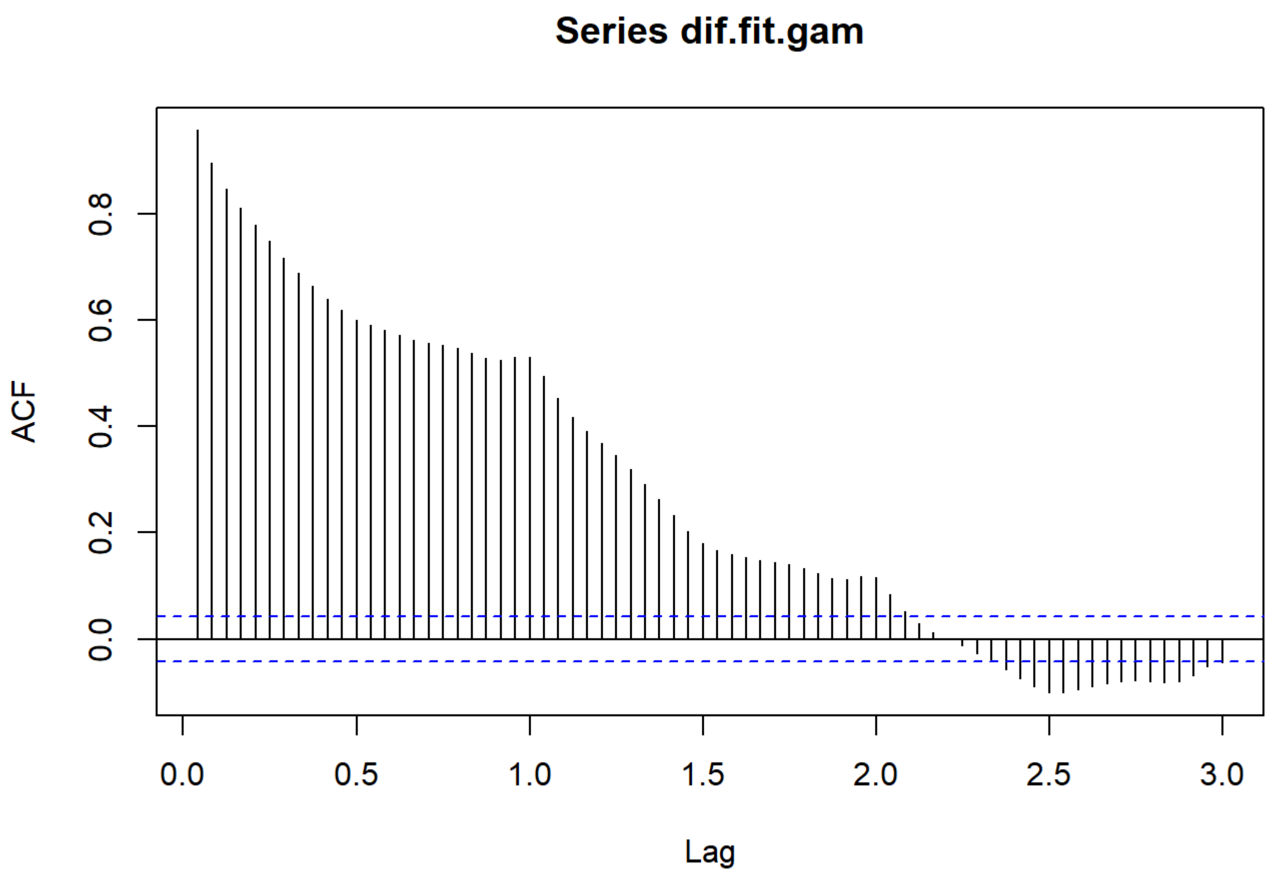
```
ts.plot(temp,ylab="Temperature", main = "Non-Parametric")
lines(ts(gam.fit$fitted,start=1,frequency=24),col="blue")
```



```
dif.fit.gam = ts((temp-fitted(gam.fit)),start=1,frequency=24)
ts.plot(dif.fit.gam,ylab="Residual Process", main = "Residual Non-Parametric")
```



```
acf(dif.fit.gam, lag.max = 24*3)
```



Response: Model Comparison Based on the fits and ACF plots, the non-parametric model is a better fit. The adjusted R squared is higher. The residual process for the non-parametric looks like it is closer for constant variance. The ACF plots also have autocorrelation but the non-parametric seems like it is getting closer to the confidence intervals. Constant mean is violated for both models. As for using the non-parametric model for long term temperatures, it models the general trend but it will not capture minimums/maximums like the true data would.