# Time Series Analysis - lab01

*Lennart Schilling (lensc874), Sridhar Adhikarla (sriad858)*

*2019-09-20*

## Contents

# Lab01

## Assignment 1. Computations with simulated data.

### 1a. Generating two time series. Applying smoothing filter.

Generate two time series $x_t = -0.8x_{t-2} + w_t$, where $x_0 = x_1 = 0$ and $x_t = cos(\frac{2\pi t}{5})$ with 100 observations each. Apply a smoothing filter $v_t = 0.2(x_t + x_{t-1} + x_{t-2} + x_{t-3} + x_{t-4})$ to these two series and compare how the filter has affected them.

#### Generating two time series.

First, a function for each given time series models will be implemented. Using these functions makes it possible to generate both time series afterwards. $w_t$ is expected to be normally distributed. From now on, the first given time series will be named as *ts1* and the second as *ts2*.

```
# Implementing function to generate t1.
generate_ts1 = function(n) {
  # Initializing x_0, x_1 and vector to store generated values for x_t.
  x_0 = 0
  x_1 = 0
  x_t = c(x_0, x_1)
  # Generating further n-2 values for x_t.
  for (i in 1:(n-2)) {
    t = length(x_t)
    x_t = c(x_t, -0.8*x_t[t-1] + rnorm(n = 1))
  }
  # Returning generated time series as data frame.
  return(data.frame(t = 0:(n-1), x_t = x_t))
}

# Generating with 100 observations.
ts1 = generate_ts1(n = 100)

# Implementing function to generate ts2.
generate_ts2 = function(n) {
  # Initializing vector to store generated values for x_t.
  x_t = c()
  # Generating n values for x_t.
  for (t in 0:(n-1)) {
    x_t = c(x_t, cos((2*pi*t)/5))
  }
  # Returning generated time series as data frame.
  return(data.frame(t = 0:(n-1), x_t = x_t))
}

# Generating with 100 observations.
ts2 = generate_ts2(n = 100)
```

#### Applying smoothing filter.

Again, a function will be implemented to apply the smoothing filter on the specified time series. Since $v_t$ will be calculated using the values from $x_t$ to $x_{t-4}$ and since both generated time series only consists of values for $t >= 0$, smoothing is only possible for $t >= 4$. The smoothed values will be added to the dataframes.

```
# Implementing function to apply smoothing filter on input time series.
add_smooth_ts = function(ts) {
  # Initializing vector to store smoothed values v_t.
  v_t = c()
  # Generating smoothed time series from input time series.
  for (obs in 5:nrow(ts)) {
    v_t = c(v_t, 0.2*(ts$x_t[obs] + ts$x_t[obs-1] + ts$x_t[obs-2] + ts$x_t[obs-3] + ts$x_t[obs-4]))
  }
  # Adding smoothed time series to data frame.
  ts_smoothed = data.frame(t = 4:(nrow(ts)-1),
                           v_t = v_t)
  ts_combined = merge(x = ts,
                      y = ts_smoothed,
                      by = "t",
                      all.x = TRUE)
  return(ts_combined)
}

# Smoothing ts1.
ts1 = add_smooth_ts(ts1)

# Smoothing ts2.
ts2 = add_smooth_ts(ts2)
```

**Plotting time series.**

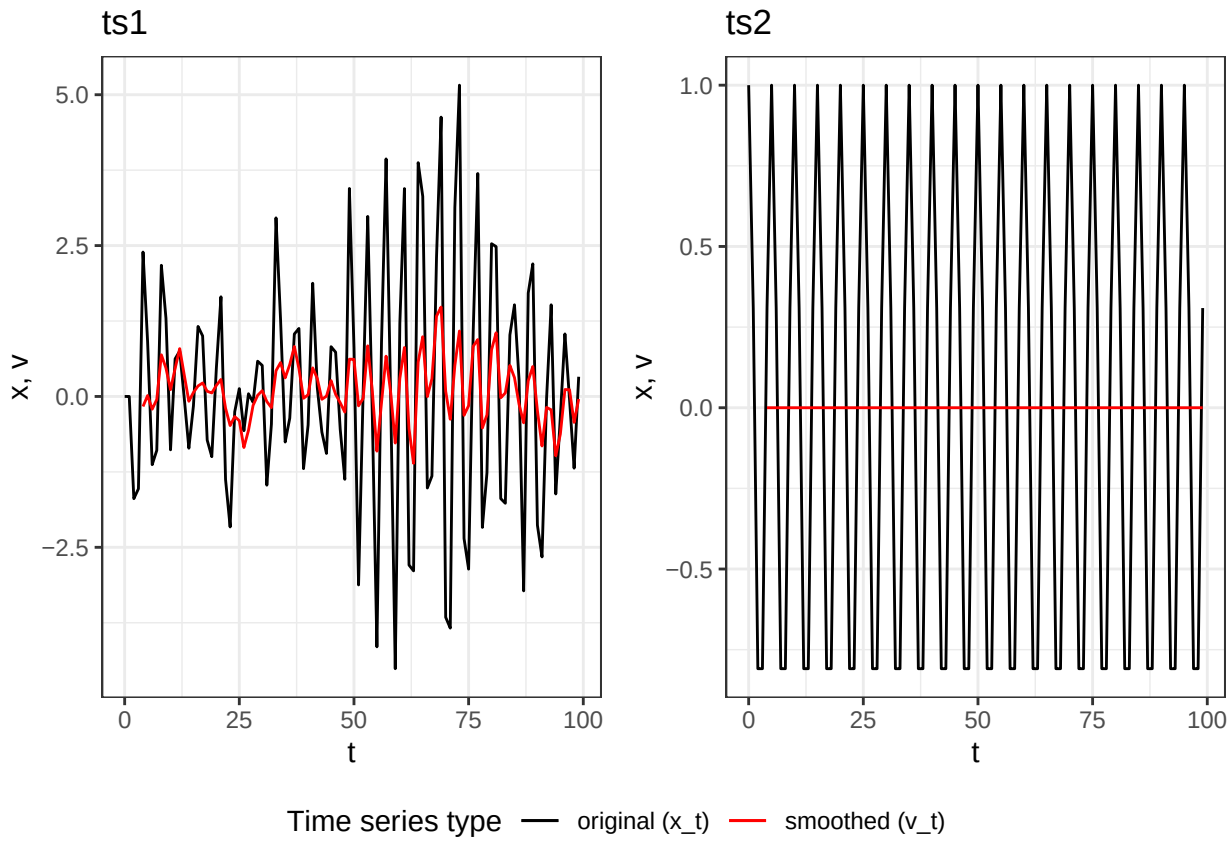
In the following, the affection of the filter will be graphically compared for both time series.

```
# Implementing function to create plot for time series.
library(ggplot2)
plot_ts = function(ts) {
  ts_plot =   ggplot(data = ts) +
    geom_line(aes(x = t,
                  y = x_t,
                  color = "original (x_t)")) +
    geom_line(aes(x = t,
                  y = v_t,
                  color = "smoothed (v_t)")) +
    scale_color_manual(values = c("original (x_t)" = "black",
                                  "smoothed (v_t)" = "red")) +
    theme_bw() +
    labs(title = as.character(substitute(ts)),
         x = "t",
         y = "x, v",
         colour = "Time series type")
  return(ts_plot)
}

# Plotting both time series next to each other.
library(ggpubr)
ggarrange(plot_ts(ts1), plot_ts(ts2), ncol = 2, common.legend = TRUE, legend="bottom")
```

```
Warning: Removed 4 rows containing missing values (geom_path).
```

```
Warning: Removed 4 rows containing missing values (geom_path).
```

```
Warning: Removed 4 rows containing missing values (geom_path).
```




It can be seen that while for *ts1* the smoothing process has delivered good results (values are smoothed but still vary over time t), for *ts2* the filter led to a nonsatisfying result by creating a new time series i almost exactly 0 for every t so that the value $v$ does not depend on t anymore.

**1b. Investigating causality and invertibility.**

Consider time series $x_t - 4x_{t-1} + 2x_{t-2} + x_{t-5} = w_t + 3w_{t-2} + w_{t-4} - 4w_{t-6}$. Write an appropriate R code to investigate whether this time series is causal and invertible.

Within the given time series expression, the left part of the equation represents the AR-part. Furthermore, the right part instead reflects the MA-part.

The equation can be transformed as follows:

$$(1 - 4B + 2B^2 + B^5)x_t = (1 + 3B^2 + B^4 - 4B^6)w_t$$

An ARMA-model is causal iff roots $\phi(z') = 0$ are outside unit circle.

```
# Checking causality.
polyroot_causality = polyroot(c(1, -4, 2, 0, 0, 1))
paste0("Causal? ", all(sqrt(Im(polyroot_causality)^2 + Re(polyroot_causality^2)) > 1))
```

```
[1] "Causal? FALSE"
```

An ARMA-model is invertible iff roots $\theta(z') = 0$ are outside unit circle.

```
# Checking invertibility.
polyroot_invertibility = polyroot(c(1, 0, 3, 0, 1, 0, -4))
paste0("Invertible? ", all(sqrt(Im(polyroot_invertibility)^2 + Re(polyroot_invertibility^2)) > 1))
```

```
[1] "Invertible? FALSE"
```

### 1c. Simulating from ARMA-process. Computing ACF using built-in R functions.

Use built-in R functions to simulate 100 observations from the process $x_t + \frac{3}{4}x_{t-1} = w_t - \frac{1}{9}w_{t-2}$, compute sample ACF and theoretical ACF, use seed 54321. Compare the ACF plots.

#### Simulating from ARMA-process.

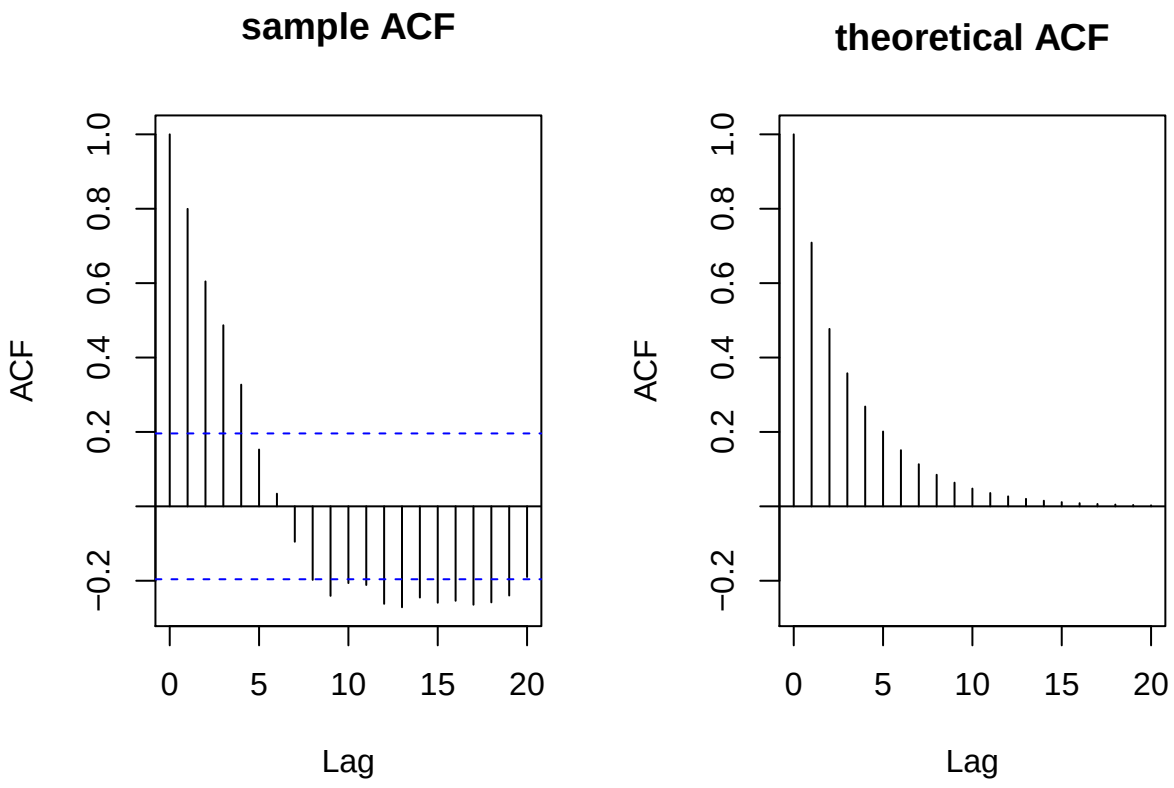
To simulate from the specified ARMA-process, we can use the built-in R function `arima.sim()`.

```
# Setting seed.
set.seed(54321)

# Simulating 100 observations from given ARMA-process.
ts = arima.sim(list(ar = 3/4, ma = c(0, -(1/9))),
               n = 100)
```

#### Computing and plotting sample ACF and theoretical ACF.

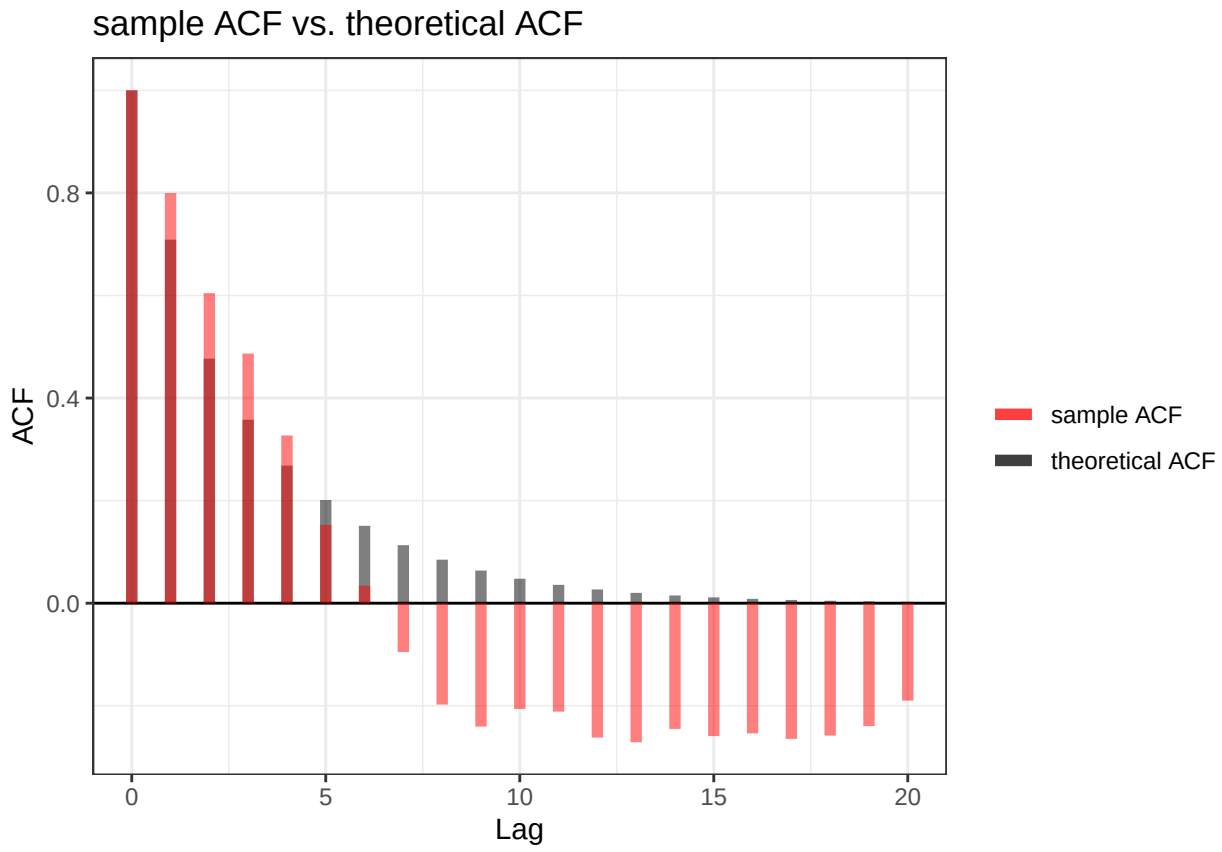
To compute sample ACF, the built-in R function `acf()` is used. To compute the theoretical ACF, the built-in R function `ARMAacf()` is used. To compare both ACF, we will calculate the theoretical autocorrelations up to the maximum lag of the sample ACF. To create a plot for the theoretical ACF, we do not use any built-in R function which directly creates the plot. Instead, we are creating the plot manually using the obtained values in `theoretical_acf`. Finally, we can compare both ACF plots next to each other.

```
# Plotting theoretical acf plot next to previously created plot sample_acf.
  # Defining following plots to be next to each other.
  par(mfrow = c(1,2))
  # Plotting sample ACF using simulated values.
  sample_acf = acf(x = ts,
                   type = "correlation",
                   plot = TRUE,
                   main = "sample ACF")
  # Storing theoretical acf.
  theoretical_acf = ARMAacf(ar = 3/4,
                            ma = c(0, -(1/9)),
                            lag.max = max(sample_acf$lag))
  # Plotting theoretial ACF.
  theoretical_acf_plot = plot(x = 0:max(sample_acf$lag),
                              y = theoretical_acf,
                              type = "h",
                              ylab = "ACF",
                              xlab = "Lag",
                              main = "theoretical ACF",
                              ylim = c(min(sample_acf$acf), 1))
  abline(h = 0)
```




We can also compare both together in one plot.

```
ggplot(mapping = aes(x = 0:max(sample_acf$lag))) +
  geom_hline(aes(yintercept = 0)) +
  geom_segment(mapping = aes(y = theoretical_acf,
                             yend = 0,
                             xend = 0:max(sample_acf$lag),
                             color = "theoretical ACF"),
               alpha = 0.5,
               size = 2) +
  geom_segment(mapping = aes(y = as.numeric(sample_acf$acf),
                             yend = 0,
                             xend = 0:max(sample_acf$lag),
                             color = "sample ACF"),
               alpha = 0.5,
               size = 2) +
  scale_color_manual(values = c("theoretical ACF" = "black",
                                "sample ACF" = "red")) +
  theme_bw() +
  labs(title = "sample ACF vs. theoretical ACF",
       x = "Lag",
       y = "ACF",
       colour = "")
```




It becomes obvious that fot the first lags (0 to 5), both ACF are very similar. However, for lags larger than 5, one can observe a certain difference. While the theoretical correlations are continously larger than zero, the sample ACF also recognizes a lot of values smaller than zero.

## Assignment 2. Visualization, detrending and residual analysis of Rhine data.

The data set Rhine.csv contains monthly concentrations of total nitrogen in the Rhine River in the period 1989-2002.

### 2a. Exploring time series.

Import the data to R, convert it appropriately to ts object (use function ts()) and explore it by plotting the time series, creating scatter plots of $x_t$ against $x_{t-1}, ..., x_{t-12}$. Analyze the time series plot and the scatter plots: Are there any trends, linear or seasonal, in the time series? When during the year is the concentration highest? Are there any special patterns in the data or scatterplots? Does the variance seem to change over time? Which variables in the scatterplots seem to have a significant relation to each other?

#### Importing data and converting it to time series object.

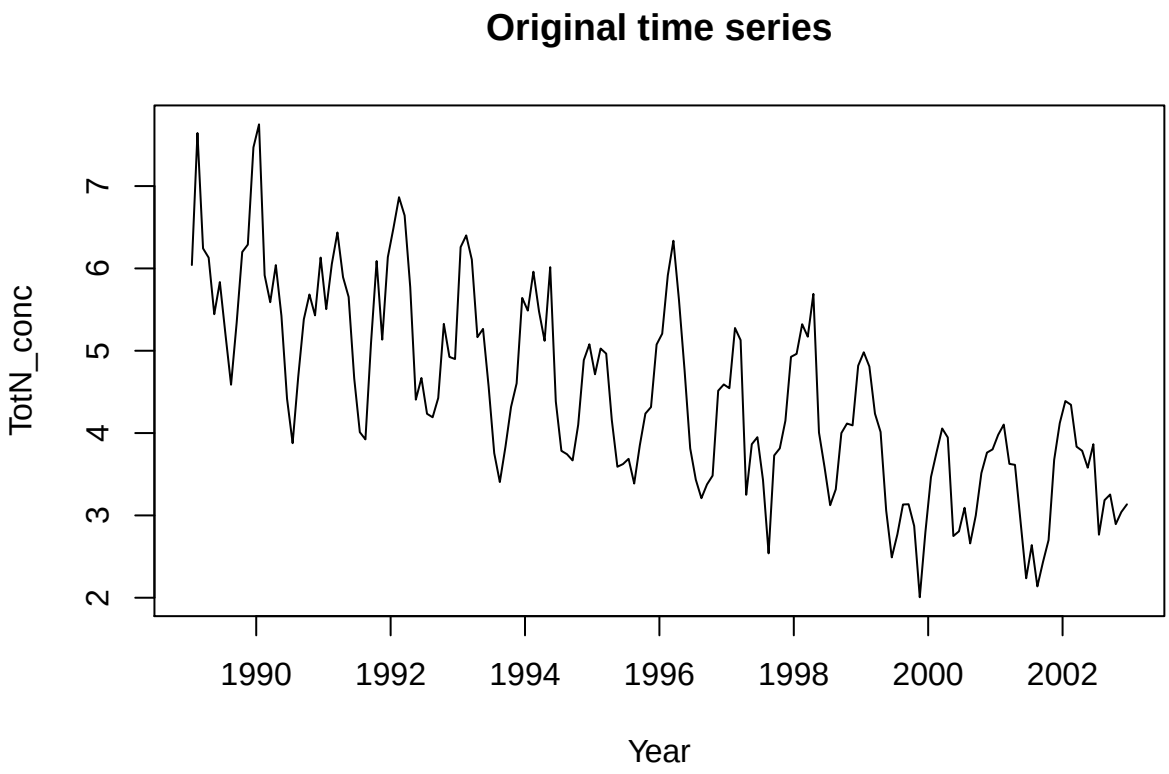
First, the data will be imported and converted.

```
# Importing data.
data_df = read.csv2("Rhine.csv", sep = ";")

# Converting dataframe to time series object.
data_ts = ts(data_df)
```

#### Plotting time series.

The time series will be plotted.

```
# Plotting time series.
plot.ts(x = data_ts[, 3],
        y = data_ts[, 4],
        type = "l",
        xlab = "Year",
        ylab = "TotN_conc",
        main = "Original time series")
```




```
# Other opportunity: ggplot.
# ggplot() +
#     geom_line(aes(x = data_ts[, 3],
#                   y = data_ts[, 4])) +
#     theme_bw() +
#     labs(title = "Original time series",
#          x = "Year",
#          y = "TotN_conc")
```

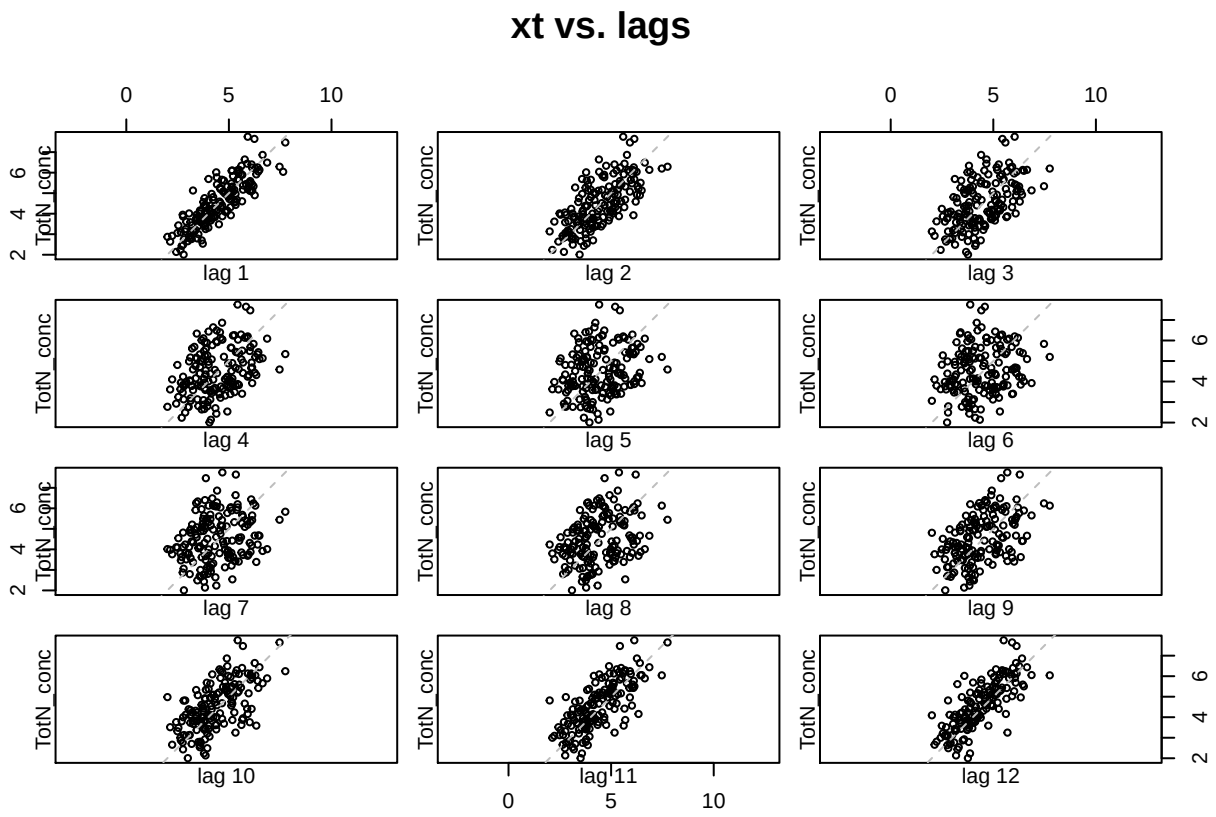
In general, the time series seems to be characterized by a negative linear trend. The mean is decreasing over time. Even if the variance does not stay exactly the same over time, one can say that it the variance of the time series does not increase or decrease too much and stays quite constant over time. Furthermore, it seems to follow a certain seasonality since on a high peak usually a low peak follows and the other way around. Thus, a similar pattern is repeating again and again.

**Plotting scatterplots to compare lags.**

Also, scatterplots will be created where $x_t$ is compared to the previous lags $x_{t-1}, ..., x_{t-12}$ by usage of the built-in R function `lag.plot()`. This implies that we compare a data point related to the twelve previous data points. It follows that we try to see a connection between a month's record and the records of the previous twelve months.

```
# Creating scatterplots for x_t vs. x_t-1, ..., x_t-12.
TotN_conc =  data_ts[, 4]
lag.plot(x = TotN_conc,
```

```
        lags = 12,
        main = "xt vs. lags")
```




The scatterplot confirms the seasonality. While the linear trend is clearly visible for lags close to zero or close to twelve, this pattern disappears the closer you get to lags like lag 6.

As a result, a value of a month seems to be very similar to the directly previously recorded values and also to the values recorded around a year (twelve months) ago. However, the value does not show that much similarity to values recorded around half a year ago. This implies that there is probably a seasonality over around half a year (e.g. winter-summer-seasonality) within the data.

### 2b. Fitting linear model. Analysing residuals.

Eliminate the trend by fitting a linear model with respect to t to the time series. Is there a significant time trend? Look at the residual pattern and the sample ACF of the residuals and comment how this pattern might be related to seasonality of the series.

#### Fitting linear model.

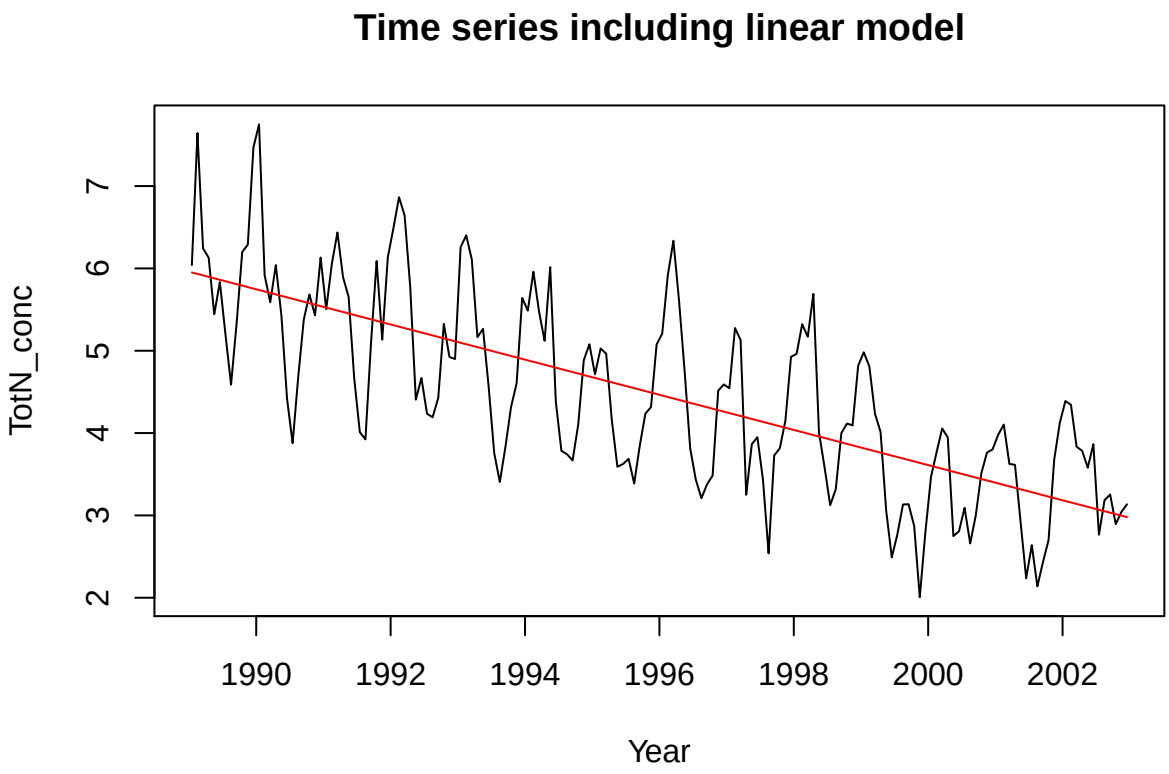
First, we fit a linear model w.r.t. t to the time series.

```
# Fitting linear mode w.r.t. t to time series.
linear_model = lm(formula = TotN_conc ~ Time,
                  data = data_ts)
# Printing information about coefficients of linear model.
knitr::kable(summary(linear_model)$coefficients[, c(1,4)])
```

| | Estimate | Pr(>\|t\|) |
|---|---|---|
| (Intercept) | 430.7072475 | 0 |
| Time | -0.2135483 | 0 |

The fitted linear model will be implemented in the plot of the time series.

```
# Plotting time series with linear model.
plot.ts(x = data_ts[, 3],
        y = data_ts[, 4],
        type = "l",
        xlab = "Year",
        ylab = "TotN_conc",
        main = "Time series including linear model")
lines(x = linear_model$model$Time,
      y = linear_model$fitted.values,
      col = "red")
```

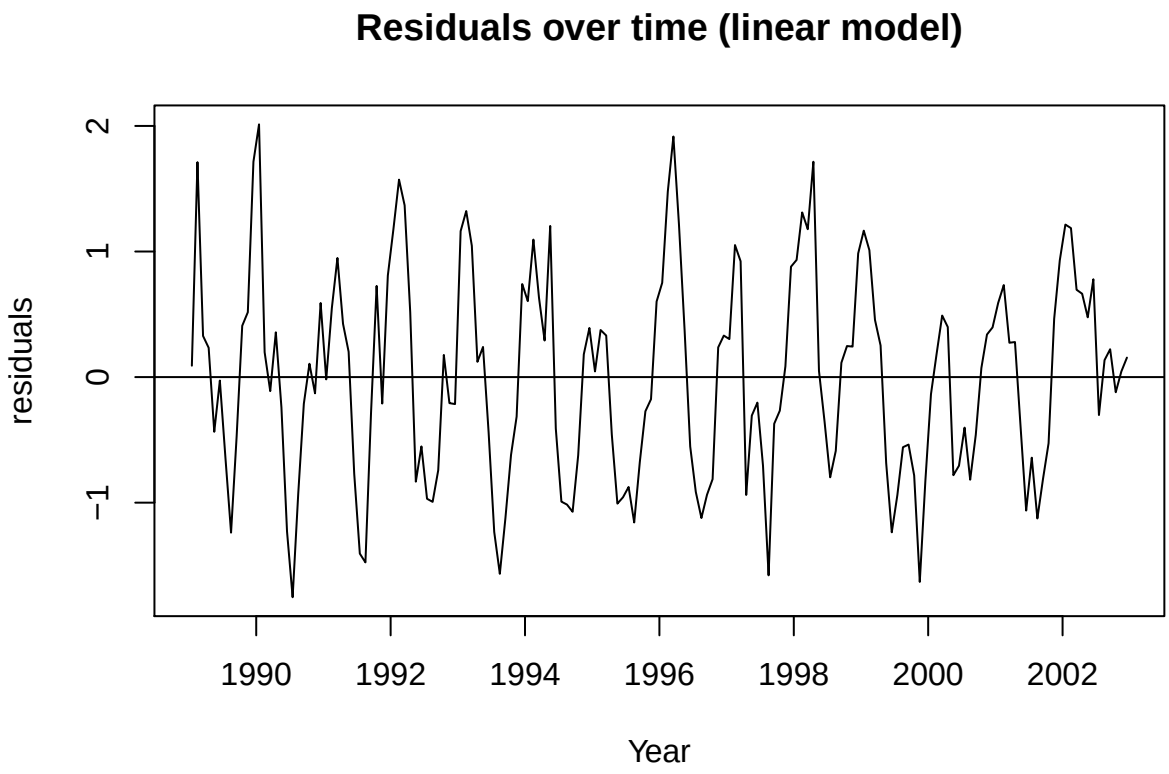



Both the graph and the given coeffients information (p-value for the $\beta$ of time smaller than 5%) confirm the guess about the significant negative linear trend.

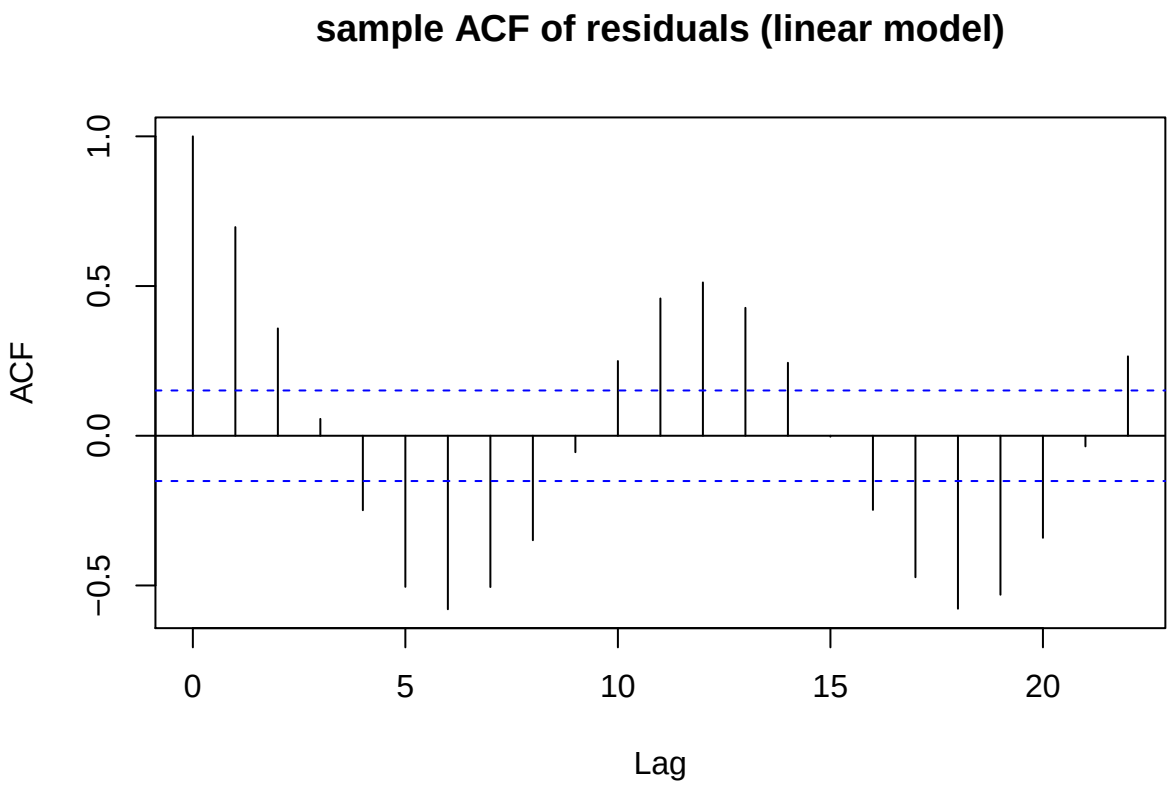
**Analyzing residuals.**

Comparing the actual time series to the fitted model, we are also able to analyze the residual pattern by plotting the residuals over time and analyzing the sample ACF of the residuals.

```
# Plotting residuals over time.
plot(x = linear_model$model$Time,
     y = linear_model$residuals,
     xlab = "Year",
     ylab = "residuals",
     type = "l",
     main = "Residuals over time (linear model)")
abline(h = 0)
```




```r
# Plotting sample ACF of residuals.
acf(x = linear_model$residuals,
    type = "correlation",
    plot = TRUE,
    main = "sample ACF of residuals (linear model)")
```



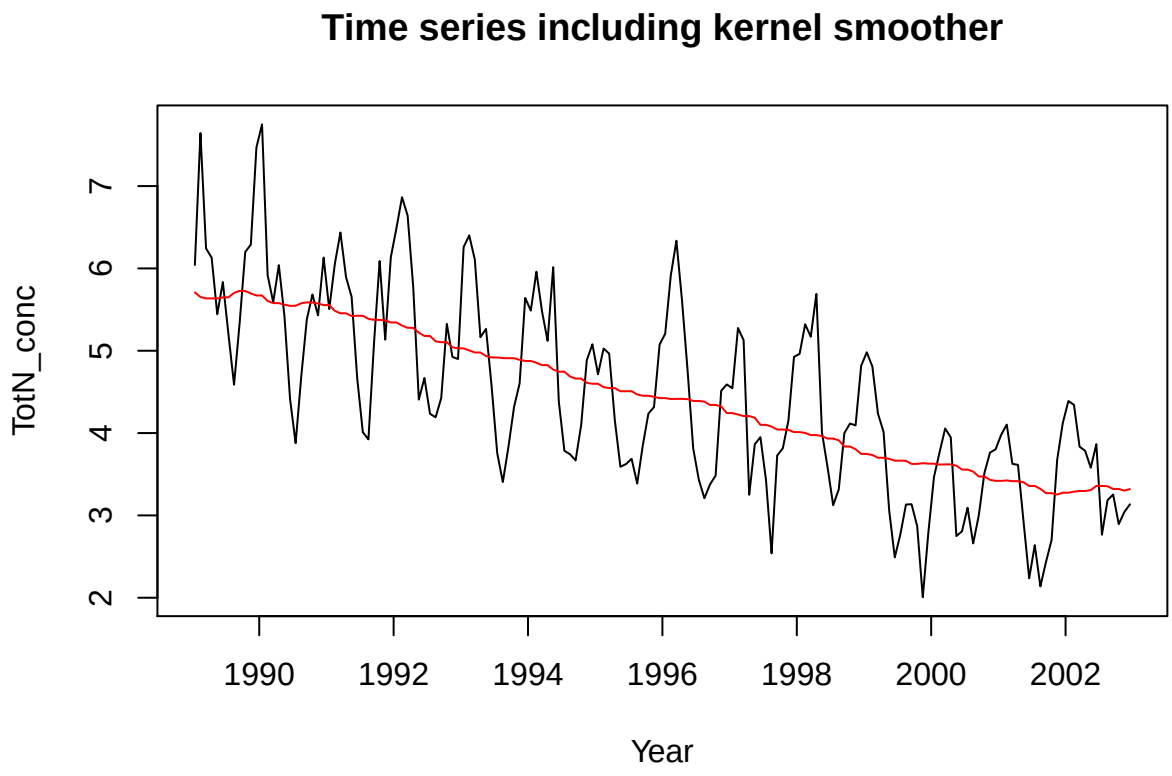

Looking at the plot which shows the residuals over time, a seasonal pattern seem to be visible again. Repetitively, positive peaks are followed by negative peaks and the other way around. Again, the time difference between the peaks is about half a year. The sample ACF plot confirms this pattern. A residual at time $x_t$ seems to have a high correlation to other residuals which are recorded diretly before ($x_{t-1}$ and $x_{t-2}$). The same correlation can be seen for residuals recorded about twelve months (one year) ago. In addition to that, a negative correlation can be seen for residuals recorded arond half a year ago. As a result, this plot also confirms the guess about the seasonality over around half a year (e.g. winter-summer-seasonality) within the data.

**2c. Fitting kernel smoother. Analysing residuals.**

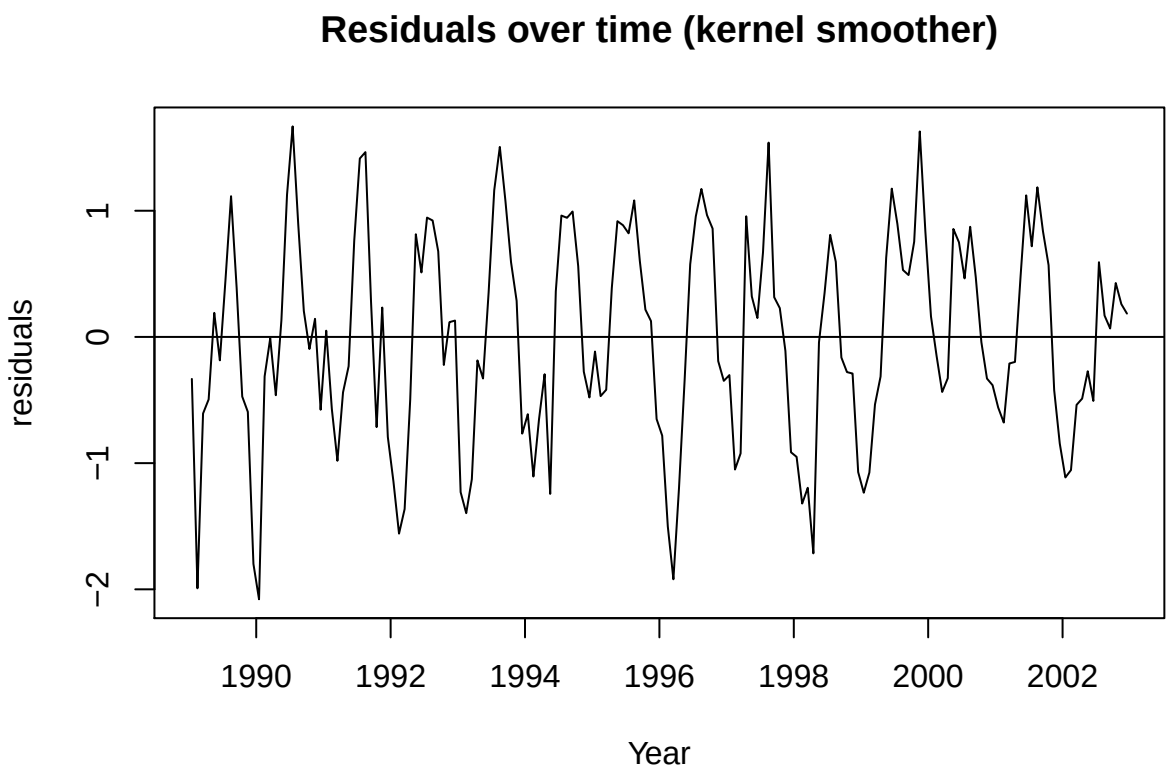
| |
|---|
| Eliminate the trend by fitting a kernel smoother with respect to t to the time series (choose a reasonable bandwidth yourself so the fit looks reasonable). Analyze the residual pattern and the sample ACF of the residuals and compare it to the ACF from step b). Conclusions? Do residuals seem to represent a stationary series? |

The general procedure from *2b* will be repeated. This time, the trend will not be eliminated by fitting a linear model but by fitting a kernel smoother. To do so, thee built-in R function `ksmooth()` will be used. By testing different values, a `bandwith` $= 5$ seems to be a reasonable choice.

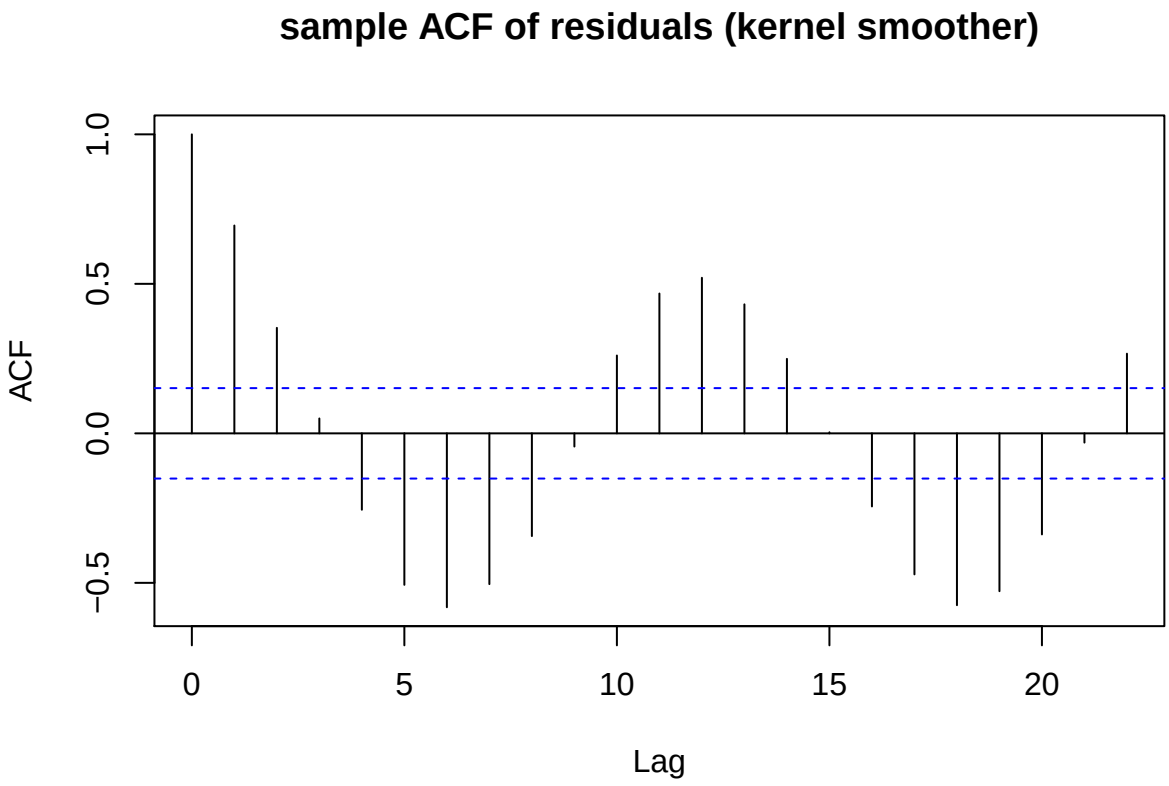
```
# Fitting kernel smoother w.r.t. t to time series.
kernel_model = ksmooth(x = data_ts[, 3],
                       y = data_ts[, 4],
                       bandwidth = 5)

# Plotting time series with fitted kernel smoother.
plot.ts(x = data_ts[, 3],
        y = data_ts[, 4],
        type = "l",
        xlab = "Year",
        ylab = "TotN_conc",
        main = "Time series including kernel smoother")
lines(x = kernel_model$x,
      y = kernel_model$y,
      col = "red")
```



```r
# Plotting residuals over time.
plot(x = linear_model$model$Time,
     y = kernel_model$y - data_ts[, 4],
     xlab = "Year",
     ylab = "residuals",
     type = "l",
     main = "Residuals over time (kernel smoother)")
abline(h = 0)
```




```r
# Plotting sample ACF of residuals.
acf(x =  kernel_model$y - data_ts[, 4],
    type = "correlation",
    plot = TRUE,
    main = "sample ACF of residuals (kernel smoother)")
```



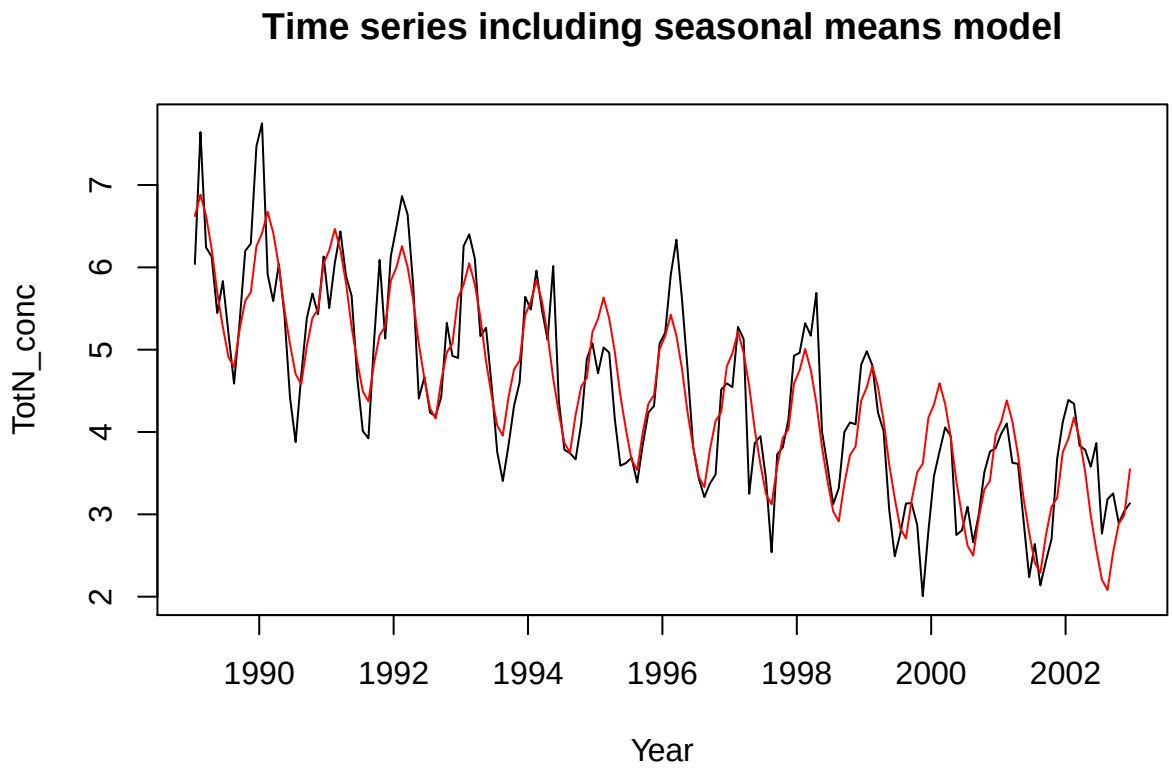

The knowledge we gain from these plots are the the same compared to the usage of the fitted linear model: Clearly, a seasonality is shown. The residuals over time do not represent a stationary time series. Instead, they follow as seasonal pattern.

**2d. Fitting seasonal means model. Analysing residuals.**

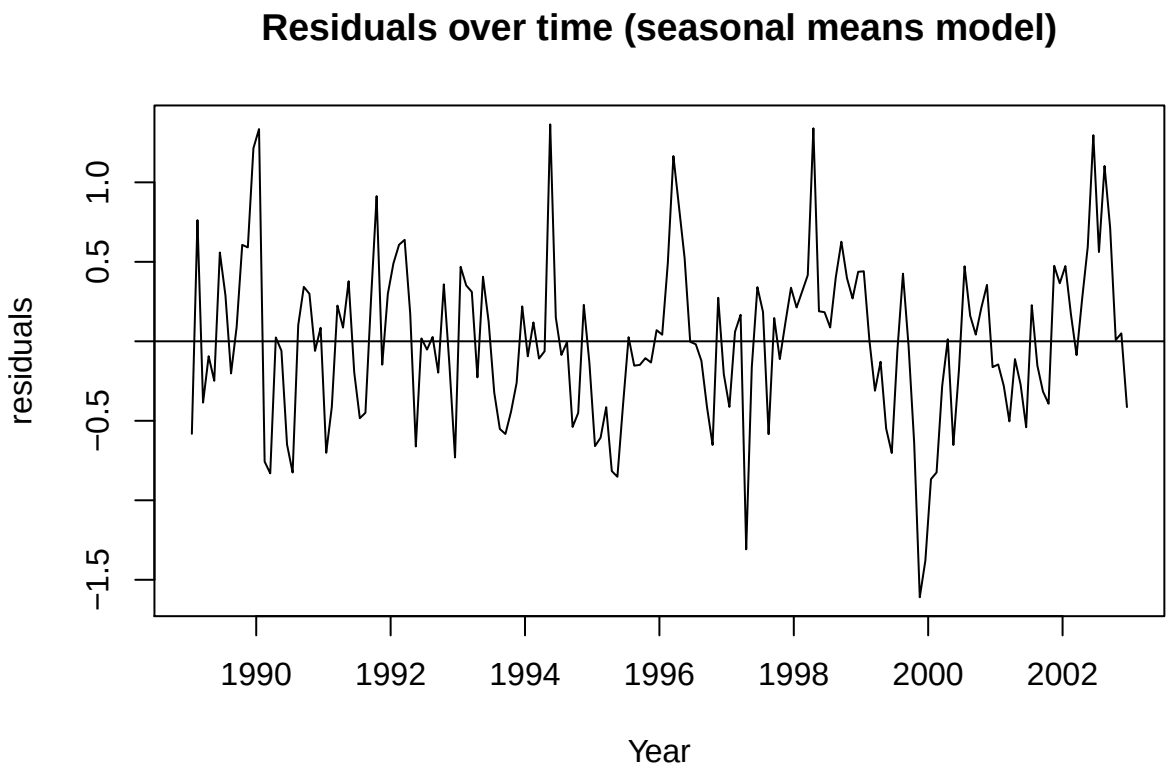
Eliminate the trend by fitting the following so-called seasonal means model: $x_t = \alpha_0 + \alpha_1 t + \beta_1 I(month = 2) + ... + \beta_1 2I(month = 12) + w_t$, where $I(x) = 1$ if x is true and 0 otherwise. Fitting of this model will require you to augment data with a categorical variable showing the current month, and then fitting a usual linear regression. Analyze the residual pattern and the ACF of residuals.

Again, the general procedure from *2b* and *2c* will be repeated. This time, the trend will be eliminated by fitting a seasonal means model.

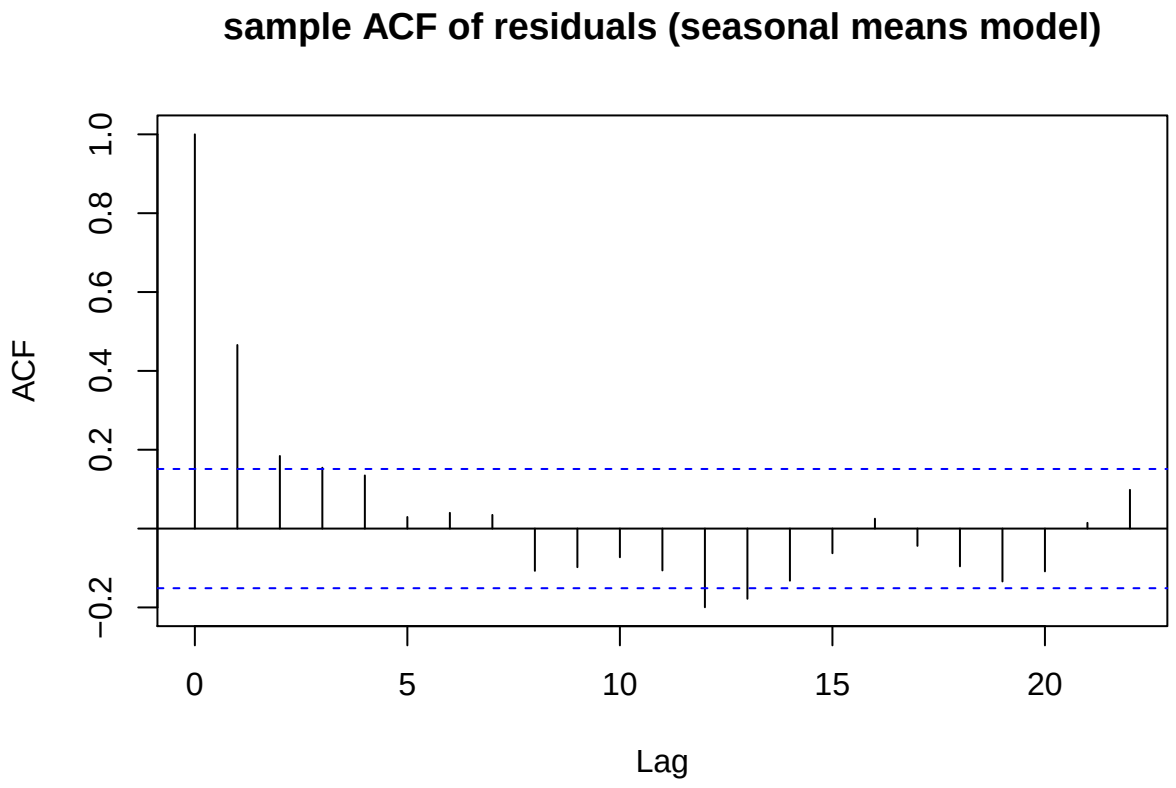
```
# Fitting seasonal means model.
seasonal_means_model = lm(formula = TotN_conc ~ Time + as.factor(Month),
                          data = data_ts)

# Plotting time series with fitted seasonal means model.
plot.ts(x = data_ts[, 3],
        y = data_ts[, 4],
        type = "l",
        xlab = "Year",
        ylab = "TotN_conc",
        main = "Time series including seasonal means model")
lines(x = seasonal_means_model$model$Time,
      y = seasonal_means_model$fitted.values,
      col = "red")
```



```r
# Plotting residuals over time.
plot(x = seasonal_means_model$model$Time,
     y = seasonal_means_model$residuals,
     xlab = "Year",
     ylab = "residuals",
     type = "l",
     main = "Residuals over time (seasonal means model)")
abline(h = 0)
```




```r
# Plotting sample ACF of residuals.
acf(x =  seasonal_means_model$residuals,
    type = "correlation",
    plot = TRUE,
    main = "sample ACF of residuals (seasonal means model)")
```




It is obvious that in this case, not only the trend but also the seasonality has been removed. By analyzing the plots which show the residuals over time and the ACF of the residuals, one can see that the data looks much more stationary than before using the other models. No seasonal, repetitively pattern is recognizable anymore.

**2e. Variable selection.**

| Perform stepwise variable selection in model from step d). Which model gives you the lowest AIC value? Which variables are left in the model? |
|---|

To perform variable selection, we use the function `step()` from the `MASS`-package. This function automatically performs stepwise model selection based on a comparison of the AIC.

```
library(MASS)

step(object = seasonal_means_model,
     direction = "both")
```

```
Start:  AIC=-202.02
TotN_conc ~ Time + as.factor(Month)

                   Df Sum of Sq     RSS      AIC
<none>                           43.237 -202.023
- as.factor(Month) 11    68.524 111.761  -64.477
- Time              1   118.387 161.624   17.499


Call:
lm(formula = TotN_conc ~ Time + as.factor(Month), data = data_ts)

Coefficients:
       (Intercept)                Time   as.factor(Month)2
         420.82746            -0.20824             0.27659
 as.factor(Month)3   as.factor(Month)4   as.factor(Month)5
           0.04006            -0.34643            -0.86165
 as.factor(Month)6   as.factor(Month)7   as.factor(Month)8
          -1.26114            -1.60808            -1.71242
 as.factor(Month)9  as.factor(Month)10  as.factor(Month)11
          -1.23669            -0.87446            -0.75127
as.factor(Month)12
          -0.17745
```

Since a lower AIC suggests a better model, the function analyzed that by removing variables from the model, the model only gets worse. That is why all independent variables (`Month`, `Time`) are kept.

## Assignment 3. Analysis of oil and gas time series.

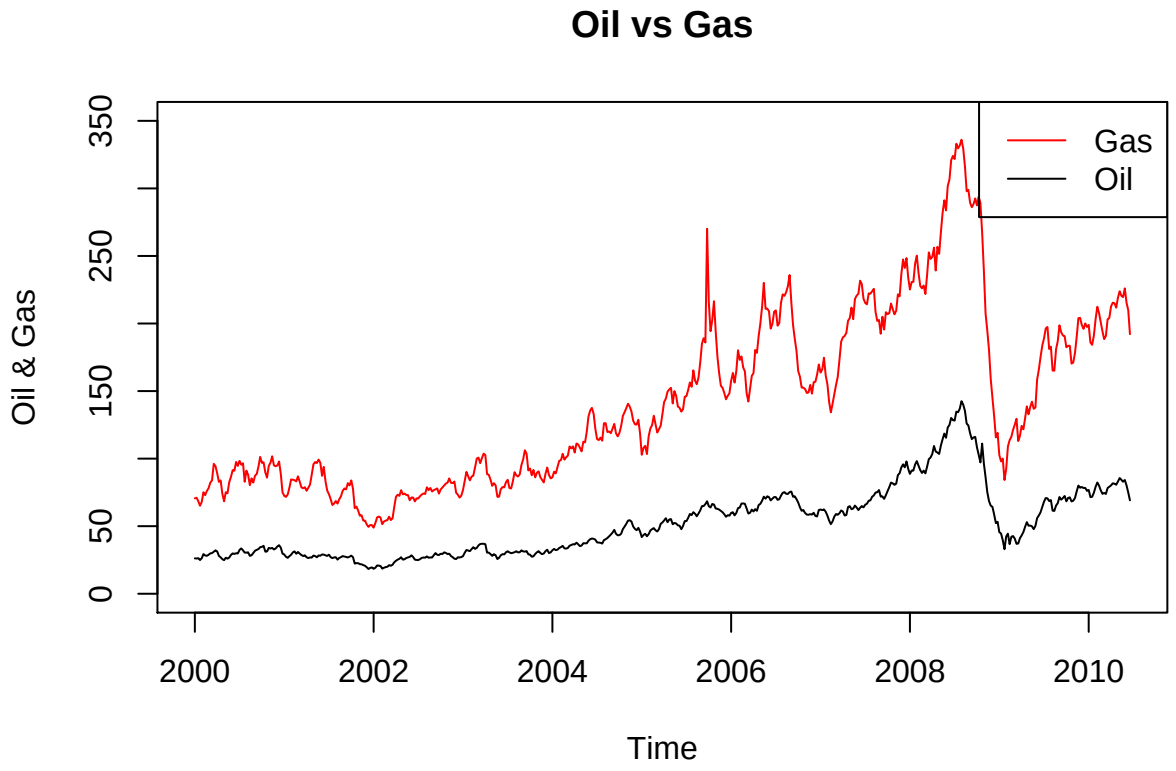
Weekly time series oil and gas present in the package astsa show the oil prices in dollars per barrel and gas prices in cents per dollar.

```
# Loading data.
library(astsa)
data(oil)
data(gas)
```

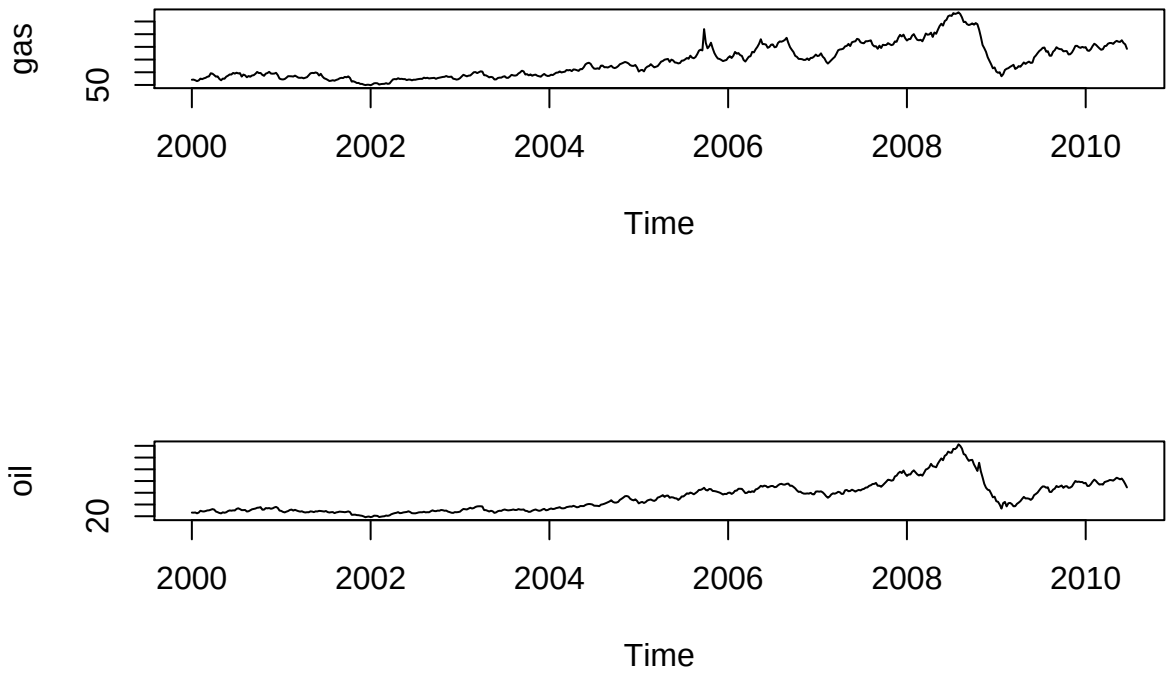
### 3a. Plotting original time series.

Plot the given time series in the same graph. Do they look like stationary series? Do the processes seem to be related to each other? Motivate your answer.

```
plot(oil, ylab = "Oil & Gas",
xlab = "Time", main = "Oil vs Gas",
ylim = c(0,350))
lines(gas, col = "red")
legend("topright", legend = c("Gas", "Oil"), lty = c(1,1), col = c("red", "black"))
```




```
par(mfrow=c(2,1))
plot(gas)
plot(oil)
```

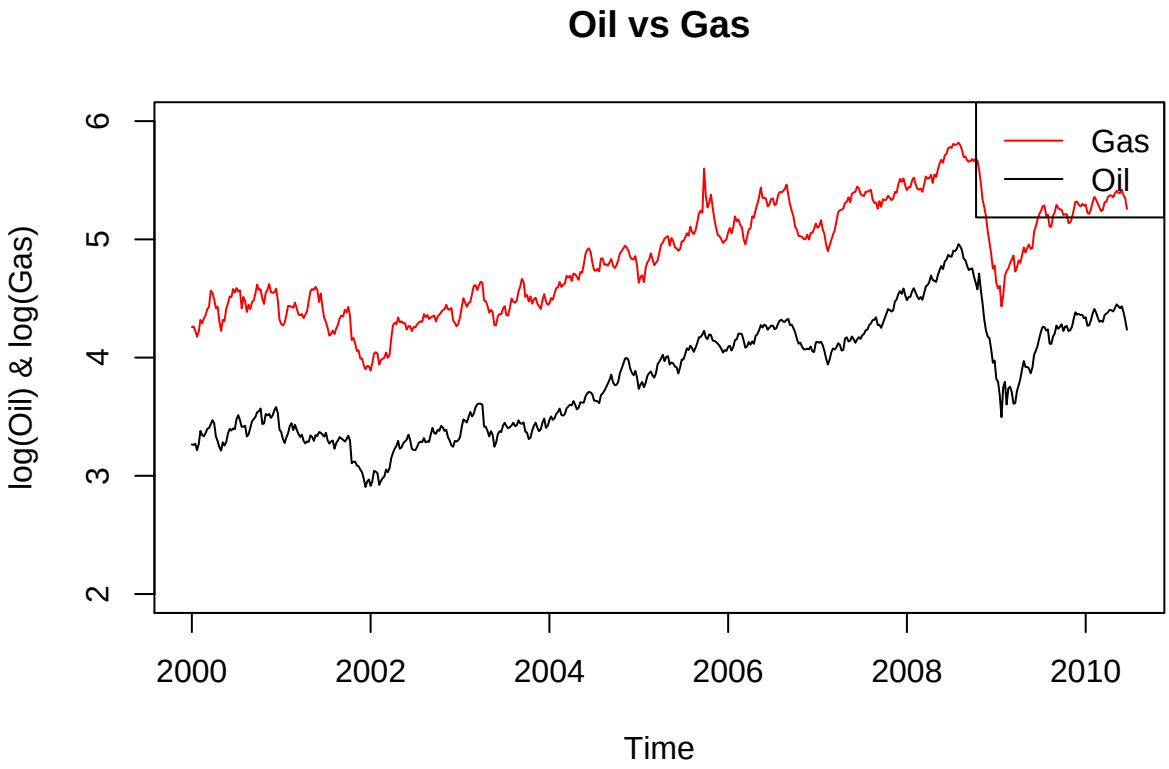



Both the time series for gas and oil are not stationary. They both follow a similar increasing trend, with a dip after 2008. Though gas seems to have a larger variance than oil, it is just because they both are not in the same units. Oil prices given are in Dollars per barrel and gas prices are in Cents per gallon. Both the time series have highs and lows during the same time periods in the time series data. This becoms clearer when we plot the two in different plots.

**3b. Plotting log-transformed time series.**

Apply log-transform to the time series and plot the transformed data. In what respect did this transformation made the data easier for the analysis?

```
plot(log(oil), ylab = "log(Oil) & log(Gas)",
xlab = "Time", main = "Oil vs Gas",
ylim = c(2,6))
lines(log(gas), col = "red")
legend("topright", legend = c("Gas", "Oil"), lty = c(1,1), col = c("red", "black"))
```



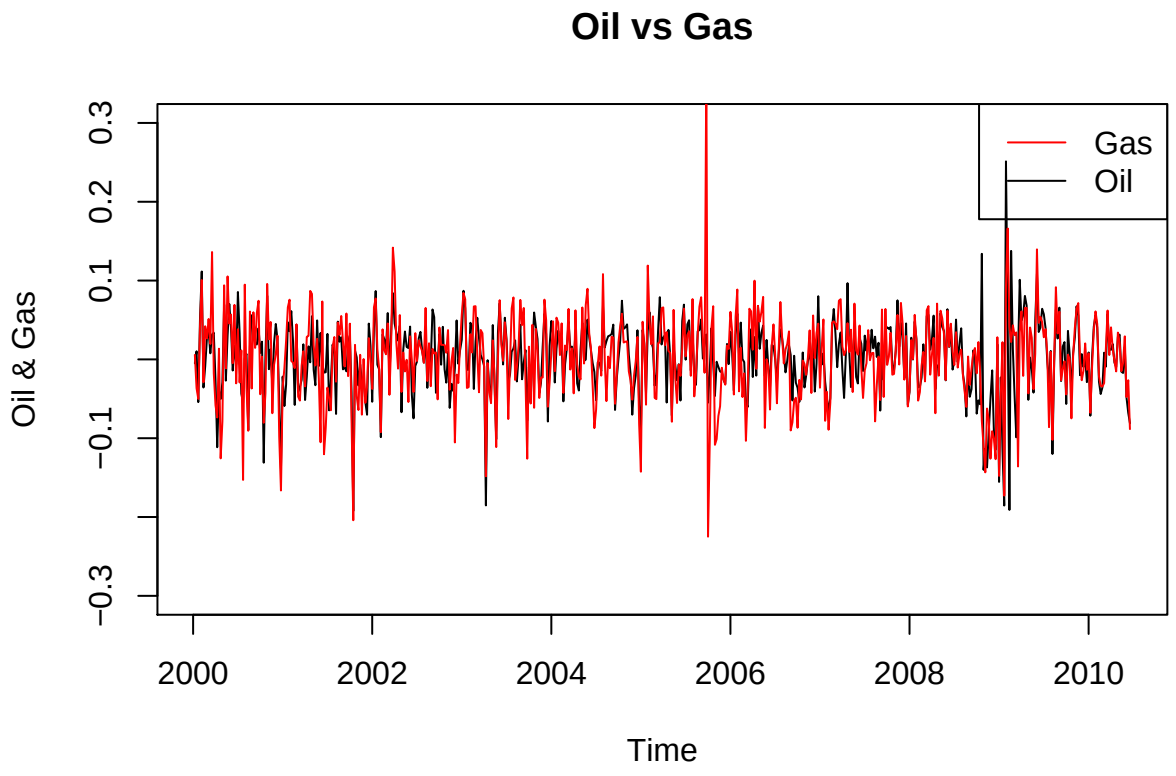

Log-transformations are often used to stabilize the variance. Comparing the origianl time series to the transformed ones, one can see that the variance is not close to constant yet. However, the differences of the variances over time have decreased. It becomes clearer that both the time series are following the same trend when we plot the log transformation of the data.

**3c. Computing first differences of log-transformed data. Plotting.**

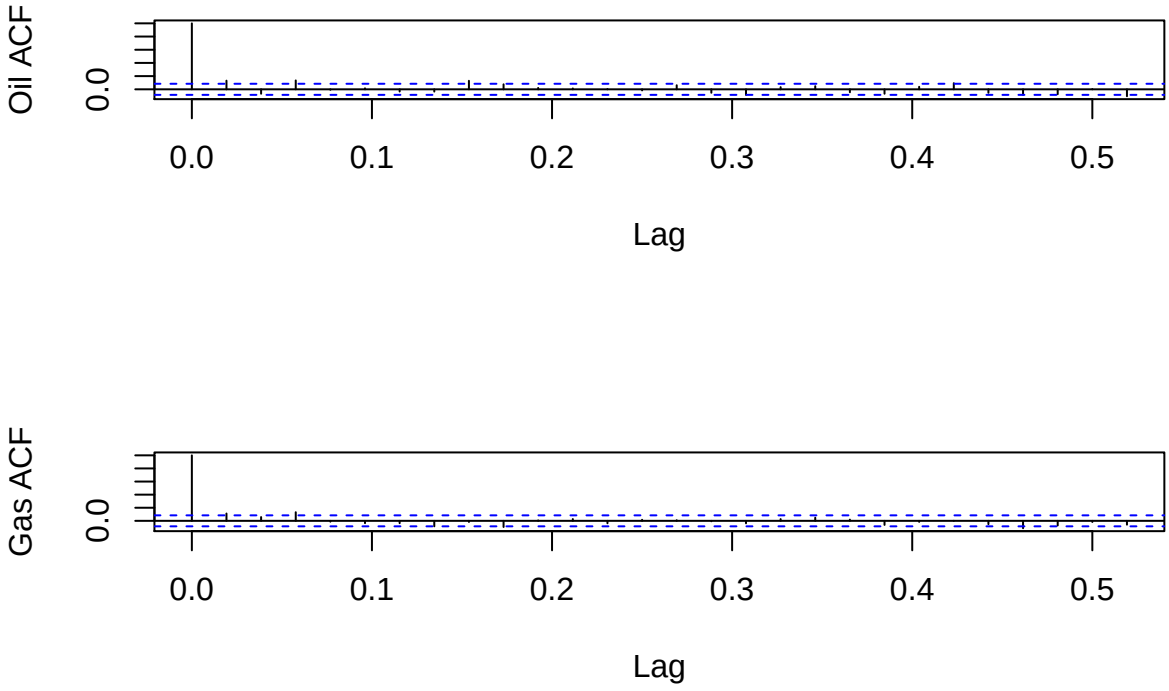
To eliminate trend, compute the first difference of the transformed data, plot the detrended series, check their ACFs and analyze the obtained plots. Denote the data obtained here as $x_t$(oil) and $y_t$ (gas).

The first difference is given by $\nabla x_t = x_t - x_{t-1}$. To compute the differences $x_t$(oil) and $y_t$(gas), we use the built-in R function `diff()`.

```
xt = diff(log(oil))
yt = diff(log(gas))

plot(xt, ylab = "Oil & Gas",
xlab = "Time", main = "Oil vs Gas",
ylim = c(-0.3,0.3))
lines(yt, col = "red")
legend("topright", legend = c("Gas", "Oil"), lty = c(1,1), col = c("red", "black"))
```




```
par(mfrow=c(2,1))
acf(xt, ylab="Oil ACF", main="")
acf(yt, ylab="Gas ACF", main="")
```





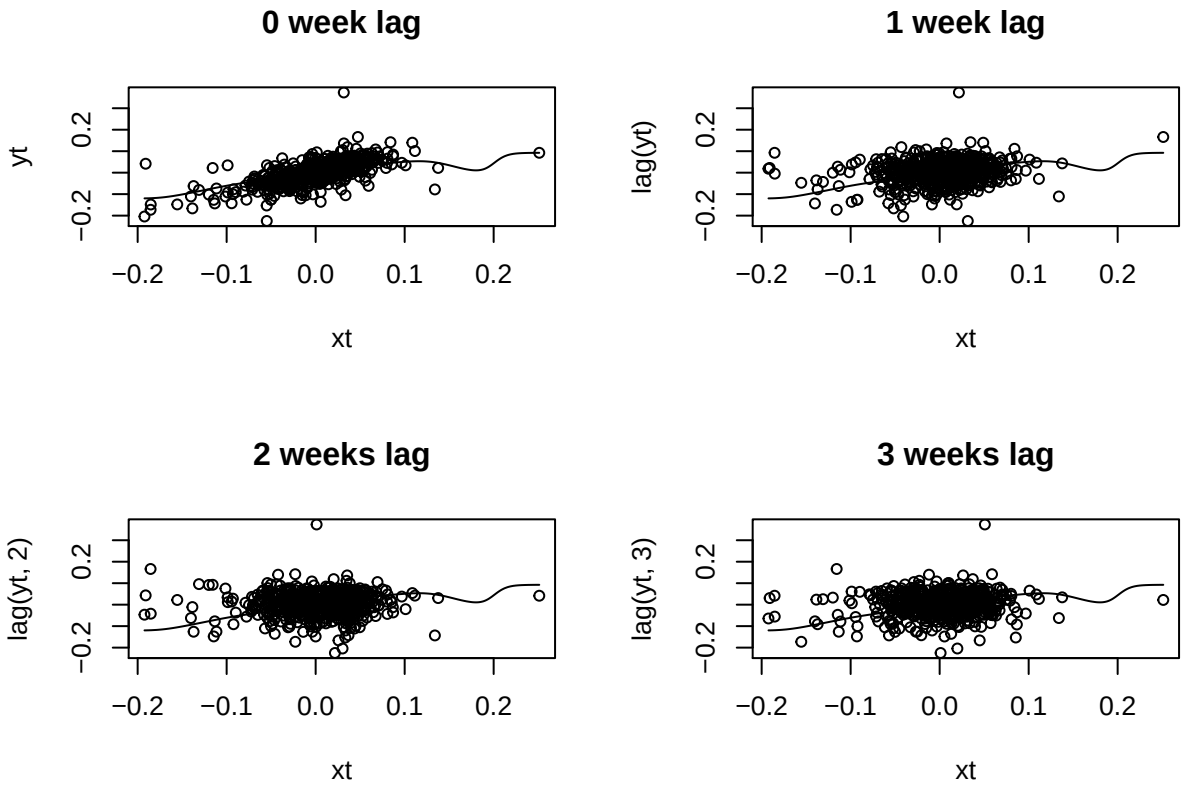


In general, excluding a few outliers, both time series seem to be stationary. The mean is very constant over time. Excluding the mentioned outliers, same could be said for the variance. Furthermore, looking at the ACF plots, it becomes obvious that both differenced time series are not characterized by a significant autocorrelation anymore.

**3d. Analyzing scatterplots for different lags.**

Exhibit scatterplots of $x_t$ and $y_t$ for up to three weeks of lead time of $x_t$; include a nonparametric smoother in each plot and comment the results: are there outliers? Are the relationships linear? Are there changes in the trend?

```
par(mfrow = c(2,2))
plot(x = xt , y = yt, main = "0 week lag")
lines(ksmooth(x = xt, y = yt, bandwidth = 0.08, kernel = "normal"))

plot(x = xt, y = lag(yt), main = "1 week lag")
lines(ksmooth(x = xt, y = lag(yt), bandwidth = 0.08, kernel = "normal"))

plot(x = xt, y = lag(yt, 2), main = "2 weeks lag")
lines(ksmooth(x = xt, y = lag(yt,2), bandwidth = 0.08, kernel = "normal"))

plot(x = xt, y = lag(yt, 3), main = "3 weeks lag")
lines(ksmooth(x = xt, y = lag(yt,3), bandwidth = 0.08, kernel = "normal"))
```




We can see from the plot that as we increase the lag, the linear relationship decreases and the number of outliers increase. There seems to be as linear relationship just lag zero. After some tries we found that bandwidth of 0.08 gave the best fit to the data with all the lags.

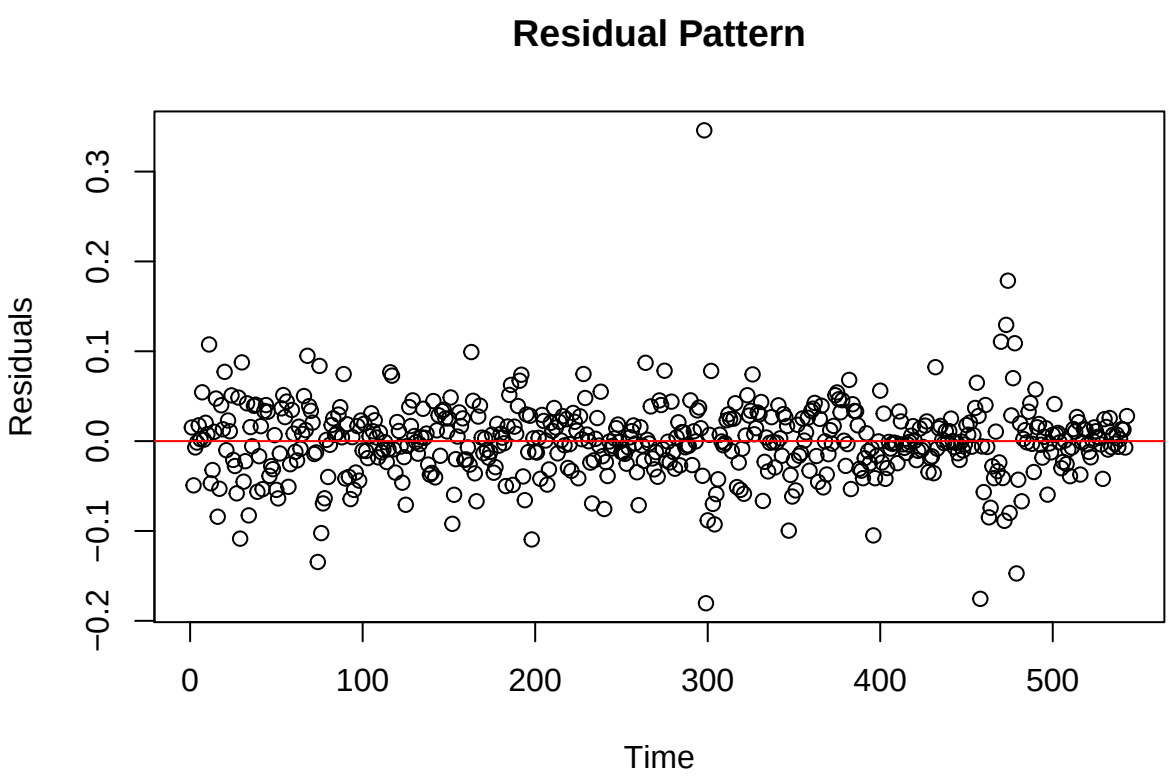
### 3e. Analyzing model with transformed features

Fit the following model: $y_t = \alpha_0 + \alpha_1 I(x_t > 0) + \beta_1 x_t + \beta_2 x_{t-1} + w_t$ and check which coefficients seem to be significant. How can this be interpreted? Analyze the residual pattern and the ACF of the residuals.

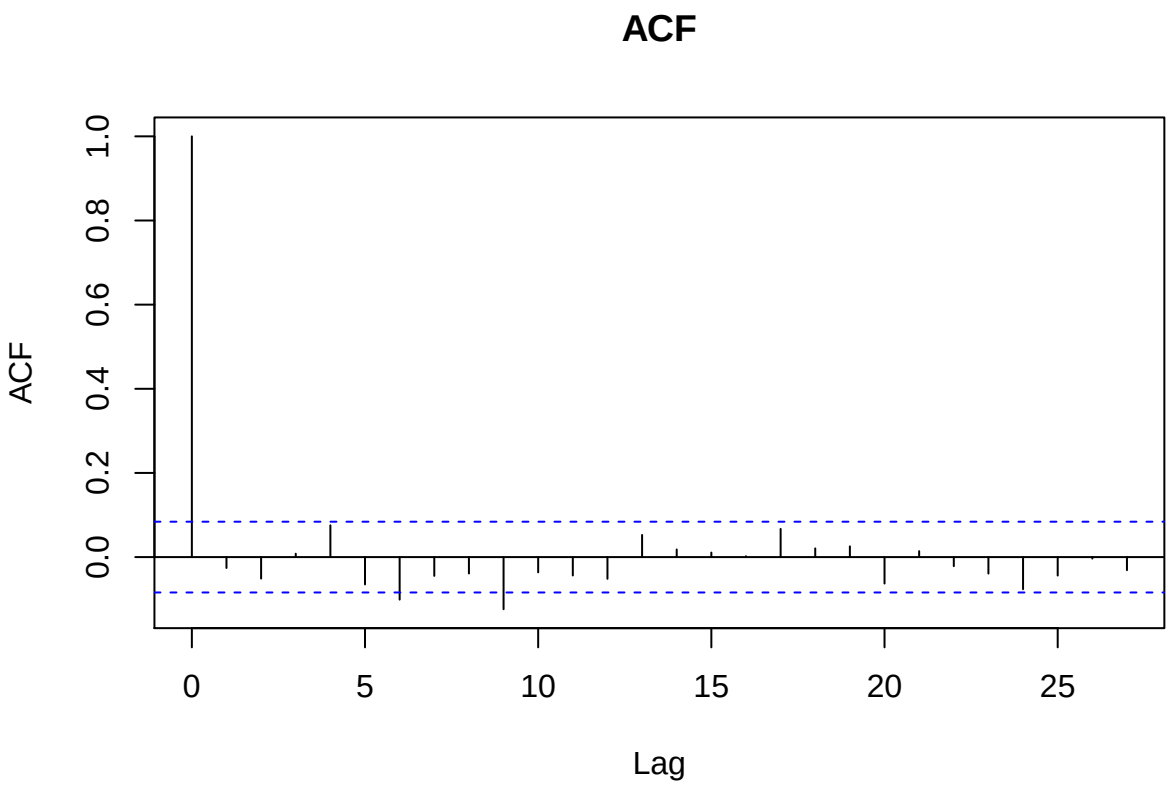
```
df <- ts.intersect(y = yt, xt = xt, xt1 = lag(xt,1), xtbin = xt>0)

lrMod = lm(y ~ xt + xt1 + xtbin, data = df)
summary(lrMod)
```

```
Call:
lm(formula = y ~ xt + xt1 + xtbin, data = df)

Residuals:
     Min       1Q   Median       3Q      Max
-0.18044 -0.02103  0.00003  0.02170  0.34592

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.006176   0.003470  -1.780   0.0757 .
xt           0.694200   0.058898  11.786   <2e-16 ***
xt1          0.012660   0.038729   0.327   0.7439
xtbin        0.012376   0.005542   2.233   0.0259 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.04202 on 539 degrees of freedom
Multiple R-squared:  0.445, Adjusted R-squared:  0.4419
F-statistic: 144.1 on 3 and 539 DF,  p-value: < 2.2e-16
```

```
plot(residuals(lrMod), ylab = "Residuals",
xlab = "Time", main = "Residual Pattern")
abline(h = 0, col="red")
```




```r
acf(residuals(lrMod), main = "ACF")
```




x_t was the most significant feature, which shows that the prediction of y_t can be done quiet accurately with just the values of x_t. It had a positive coefficient(0.69). The binary feature showing where x_t is greater than zero is the next most significant feature, with a positive coefficient as well(0.012).

The residual pattern shows that the model was a good fit to the data. The acf of the residuals shows that there is no significant autocorrelation in the residuals.

# Time Series Analysis - lab02

*Lennart Schilling (lensc874), Sridhar Adhikarla (sriad858)*

*2019-09-30*

## Contents

# Lab02

```
set.seed(12345)
```

## Assignment 1. Computations with simulated data.

### 1a. Computing partial autocorrelation in different ways.

Generate 1000 observations from AR(3) process with $\phi_1 = 0.8, \phi_2 = -0.2, \phi_3 = 0.1$. Use these data and the definition of PACF to compute $\phi_{33}$ from the sample, i.e. write your own code that performs linear regressions on necessarily lagged variables and then computes an appropriate correlation. Compare the result with the output of function pacf() and with the theoretical value of $\phi_{33}$.

#### Generating time series for AR(3)-process.

```
ts_original = arima.sim(list(ar = c(0.8,-0.2,0.1)),
                        n=1000)
```

#### Computing partial autocorrelation with own code.

Our goal is to get the partial correlation between the original series ($x$) and its third lag ($x_{t-3}$). We do this by the following steps:

- Creating lagged series ($x_{t-1}, x_{t-2} and x_{t-3}$) and binding all together with the original time series ($x$) in a data frame.
- Fitting two linear models to the data with the original time series ($x$) and the lagged time series of interest ($x_{t-3}$) as the dependent variables and with the other lagged time series in between ($x_{t-1}, x_{t-2}$) as the independent variables.
- Calculating the partial autocorrelation between $x$ and $x_{t-3}$ as the correlation between the residuals of the two linear models.

```
# Binding original and lagged time series in a data frame.
ts_all = ts.intersect(x = ts_original,
                      x1 = lag(ts_original, -1),
                      x2 = lag(ts_original, -2),
                      x3 = lag(ts_original, -3),
                      dframe = T)

# Fitting linear models.
lm1 = lm(formula = x ~ x1 + x2,
         data = ts_all)
lm2 = lm(formula = x3 ~ x1 + x2,
         data = ts_all)

# Calculating correlation between the residuals of the models.
knitr::kable(cor(cbind(x = lm1$residuals,
                       x3 = lm2$residuals)),
             caption = "Own-calculated partial autocorrelation between x and x3")
```

Table 1: Own-calculated partial autocorrelation between x and x3

| | x | x3 |
|---|---|---|
| x | 1.0000000 | 0.1146076 |
| x3 | 0.1146076 | 1.0000000 |

**Computing partial autocorrelation using pacf().**

```
print("Simulated partial autocorrelation between x and x3 using pacf()-function:")
```

```
[1] "Simulated partial autocorrelation between x and x3 using pacf()-function:"
```

```
pacf(ts_original, lag.max = 3, plot = F)$acf[3]
```

```
[1] 0.1170643
```

**Computing theoretical value.**

```
print("Computed theoretical value for the partial autocorrelation between x and x3:")
```

```
[1] "Computed theoretical value for the partial autocorrelation between x and x3:"
```

```
print(ARMAacf(ar = c(0.8,-0.2,0.1),
                      lag.max = 3,
                      pacf = TRUE)[3])
```

```
[1] 0.1
```

**Conclusions.**

Both approaches (own implementation and usage of the `pacf()`-function) have delivered reasonable results by returning a similar value for the partial autocorrelation between the original time series and its third lagged version compared to the theoretical value.

### 1b. Estimating parameter for an AR-model using different methods. Analysing results.

Simulate an AR(2) series with $\phi_1 = 0.8, \phi_2 = 0.1$ and $n = 100$. Compute the estimated parameters and their standand errors by using three methods: method of moments (Yule-Walker equations), conditional least squares and maximum likelihood (ML) and compare their results to the true values. Which method does seem to give the best result? Does theoretical value for $\phi_2$ fall within confidence interval for ML estimate?

#### Generating time series for AR(2)-process.

```
ts = arima.sim(list(ar = c(0.8, 0.1)),
               n = 100)
```

#### Computing estimated parameters for different methods.

To fit an AR-model to an univariate time series, we use the function `ar()`. Within this function, the parameter `order.max` is set to 2 so that an AR(2)-model will be fit to the data. Alternatively, `arima()` could also be used.

```
# Computing estimated parameters.
  # Method of moments (Yule-Walker equations).
  yule_walker_fit = ar(x = ts,
                       aic = FALSE,
                       order.max = 2,
                       method = "yule-walker")
  # Conditional least squares.
  cls_fit = ar(x = ts,
               aic = FALSE,
               order.max = 2,
               method = "ols")
  # Maximum likelihood.
  ml_fit = ar(x = ts,
              aic = FALSE,
              order.max = 2,
              method = "ml")
```

#### Calculating the standard errors of the estimated parameters.

The formula for the standard error is given by $\sqrt{\frac{\sigma^2}{n}}$. $\sigma^2$ refers to the variance of the estimated parameters. Therefore, we calculate the standard errors of the parameters for Yule-Walker and maximum likelihood in R as follows:

```
# Calculating standard errors of estimated parameters.
  # Method of moments (Yule-Walker equations).
  se_yule_walker = c(sqrt(yule_walker_fit$asy.var.coef[1,1]/yule_walker_fit$n.used),
                     sqrt(yule_walker_fit$asy.var.coef[2,2]/yule_walker_fit$n.used))
  # Maximum likelihood.
  se_ml = c(sqrt(ml_fit$asy.var.coef[1,1]/ml_fit$n.used),
            sqrt(ml_fit$asy.var.coef[2,2]/ml_fit$n.used))
```

For conditional least squares, the standard error is directly given within the `ar`-object (fitted model). Therefore, we can directly access the standard error in this case.

```
  # Conditional least squares.
  se_cls = cls_fit$asy.se.coef$ar
```

#### Calculating confidence interval for ML estimate.

We calculate the 95% confidence interval for the ML parameter estimate $\hat{\phi}_2$ by $\hat{\phi}_2 \pm 1.96 * \sqrt{\sigma^2_{\hat{\phi}_2}}$.

```
# Calculating 95% confidence interval for ml estimate phi_2.
ci = c(lower_limit = ml_fit$ar[2] - 1.96 * sqrt(ml_fit$asy.var.coef[2, 2]),
       upper_limit = ml_fit$ar[2] + 1.96 * sqrt(ml_fit$asy.var.coef[2, 2]))
```

**Results.**

```
knitr::kable(data.frame(true = c(0.8, 0.1),
                        yule_walker = yule_walker_fit$ar,
                        se_yule_walker = se_yule_walker,
                        cls = cls_fit$ar,
                        se_cls = se_cls,
                        ml = ml_fit$ar,
                        se_ml = se_ml),
             caption = "Result of parameter estimation for the AR(2) model")
```

Table 2: Result of parameter estimation for the AR(2) model

| | true | yule_walker | se_yule_walker | cls | se_cls | ml | se_ml |
|---|---|---|---|---|---|---|---|
| ar1 | 0.8 | 0.8571752 | 0.0101514 | 0.9386075 | 0.1013164 | 0.9015078 | 0.0093787 |
| ar2 | 0.1 | -0.0199902 | 0.0101514 | -0.0910831 | 0.0991170 | -0.0354404 | 0.0093787 |

```
knitr::kable(data.frame(theoretical = 0.1,
                        lower_limit = ci[1],
                        upper_limit = ci[2],
                        row.names = NULL),
             caption = "Theoretical value vs. confidence interval for ML estimate phi_2.")
```

Table 3: Theoretical value vs. confidence interval for ML estimate phi_2.

| theoretical | lower_limit | upper_limit |
|---|---|---|
| 0.1 | -0.2192624 | 0.1483816 |

**Conclusions.**

It follows that for all used methods, the estimations still differ quite a lot from the obtained theoretical values for the parameters $\phi_1$ and $\phi_2$. This is based on the small given size of the time series ($n = 100$). Increasing the size to a larger number of observations would lead to a closer approach of the estimates towards the theoretical values. However, in this case, the Yule-Walker method seems to be the most accurate method even if all methods return very similar values.

The second returnded table shows that the theoretical value for $\phi_2$ lies within the confidence interval for the estimate using the maximum likelihood method.

### 1c. Generating multiplicative seasonal ARMA. Analysing.

Generate 200 observations of a seasonal $ARIMA(0,0,1) \times (0,0,1)_{12}$ model with coefficients $\Theta = 0.6$ and $\theta = 0.3$ by using arima.sim(). Plot sample ACF and PACF and also theoretical ACF and PACF. Which patterns can you see at the theoretical ACF and PACF ? Are they repeated at the sample ACF and PACF ?

#### Rewriting multiplicative seasonal ARIMA.

In this case, we are requested to generate a time series from a multiplicative seasonal ARIMA. It consists of a standard ($ARIMA(0,0,1)$) and a seasonal ($(0,0,1)_{12}$) part. Since the function `arima.sim()` does not include any parameter to set up the seasonal part, we have to rewrite the multiplicative seasonal ARIMA to a normal ARIMA.

Following the slides from the second teaching session, we can rewrite it as follows:

$$ARIMA(0,0,1) \times (0,0,1)_{12}$$
$$ARIMA(0,0,1) : x_t = (1 + \theta B)w_t$$
$$(0,0,1)_{12} : x_t = (1 + \Theta B^{12})w_t$$

Mutiplying these two series leads to the follwing series:

$$x_t = (1 + \theta B)w_t(1 + \Theta B^{12})$$
$$\Rightarrow x_t = (1 + \theta B + \Theta B^{12} + \theta\Theta B^{13})w_t$$
$$\Rightarrow x_t = w_t + \theta w_{t-1} + \Theta w_{t-12} + \theta\Theta w_{t-13}$$

#### Generating time series for MA(13)-process.

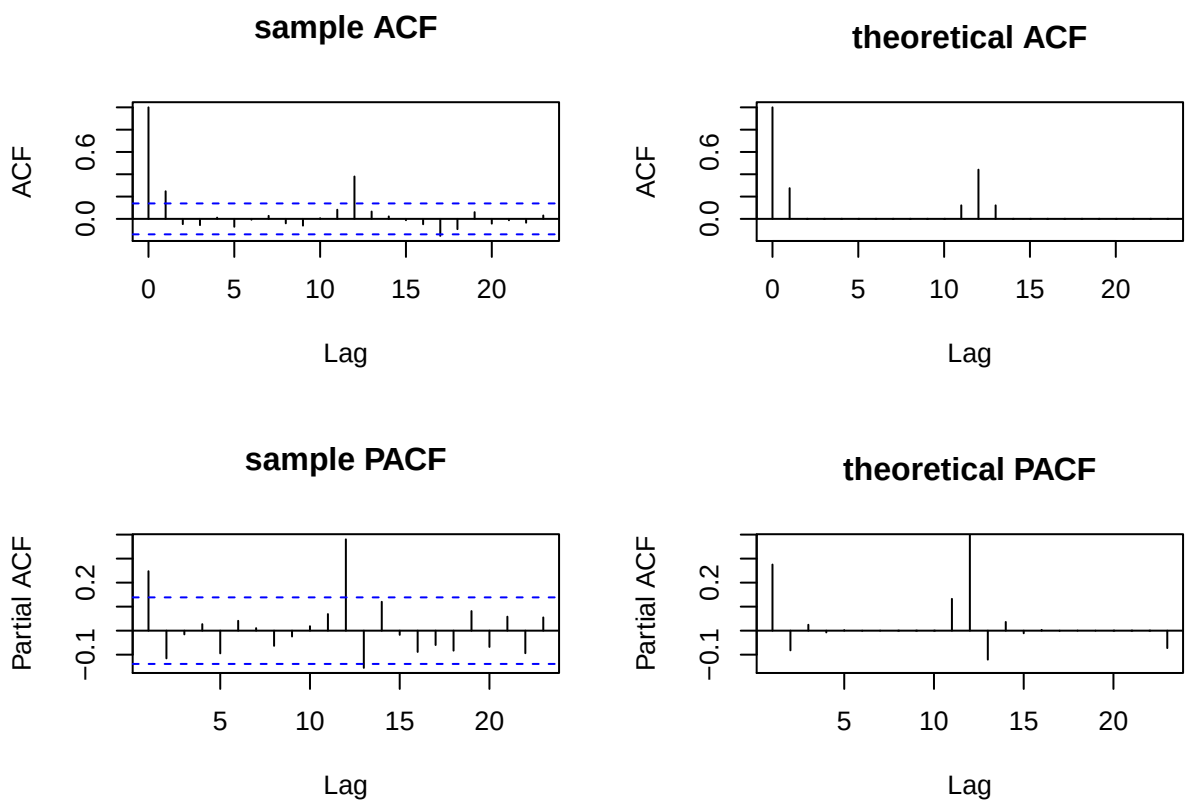
Using the derived MA(13)-model, we can generate 200 observations with coefficients $\Theta = 0.6$ and $\theta = 0.3$.

```
ts = arima.sim(model = list(ma = c(0.3, # t-1
                                   rep(0, 10), # t-2, ..., t-11
                                   0.6, # t-12
                                   0.3 * 0.6)), # t-13
               n = 200)
```

#### Plotting sample/theoretical ACF and PACF.

```
# Defining following plots to be next to each other.
par(mfrow = c(2,2))

# Plotting sample/theoretical ACF.
  # Plotting sample ACF using simulated values.
  sample_acf = acf(x = ts,
                   type = "correlation",
                   plot = TRUE,
                   main = "sample ACF")
  # Storing theoretical ACF.
  theoretical_acf = ARMAacf(ma = c(0.3, rep(0, 10), 0.6, 0.3 * 0.6),
                            lag.max = max(sample_acf$lag))
  # Plotting theoretial ACF.
  theoretical_acf_plot = plot(x = 0:max(sample_acf$lag),
                              y = theoretical_acf,
                              type = "h",
                              ylab = "ACF",
```

```
                              xlab = "Lag",
                              main = "theoretical ACF",
                              ylim = c(min(sample_acf$acf), 1))
  abline(h = 0)

# Plotting sample/theoretical PACF.
  # Plotting sample PACF using simulated values.
  sample_pacf = pacf(x = ts,
                     plot = TRUE,
                     main = "sample PACF")
  # Storing theoretical PACF.
  theoretical_pacf = ARMAacf(ma = c(0.3, rep(0, 10), 0.6, 0.3 * 0.6),
                             lag.max = max(sample_acf$lag),
                             pacf = TRUE)
  # Plotting theoretial ACF.
  theoretical_pacf_plot = plot(x = 1:max(sample_pacf$lag), # PACF plots from lag1 onwoards.
                               y = theoretical_pacf,
                               type = "h",
                               ylab = "Partial ACF",
                               xlab = "Lag",
                               main = "theoretical PACF",
                               ylim = c(min(sample_pacf$acf),
                                        max(sample_pacf$acf)))
  abline(h = 0)
```




**Conclusions.**

Analysing the created plots, one can see that as expected, both the autocorrelation and partial autocorrelation is significant for the lags 1 and 12 in all cases (theoretical values and sample values). However, the autocorrelation with lag 13 is only significantly visible in the PACF plots.

### 1d. Forecasting after fitting multiplicative seasonal ARIMA.

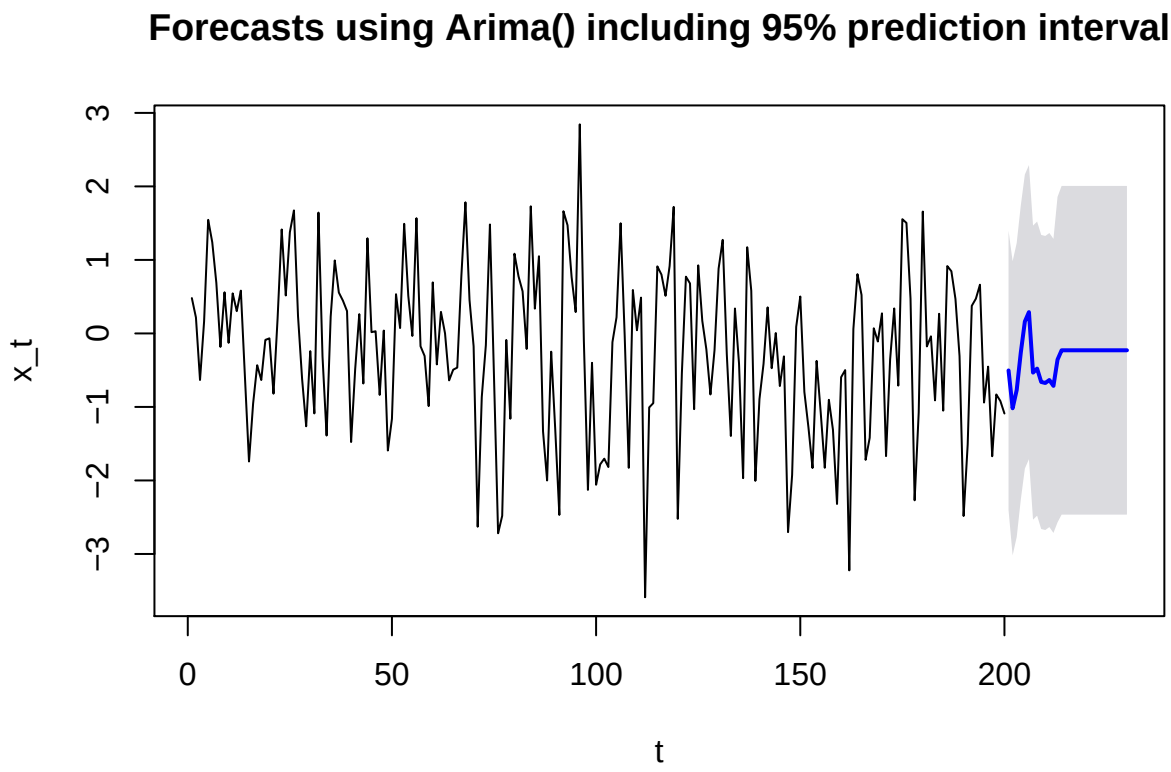
Generate 200 observations of a seasonal $ARIMA(0,0,1) \times (0,0,1)_{12}$ model with coefficients $\Theta = 0.6$ and $\theta = 0.3$ by using arima.sim(). Fit $ARIMA(0,0,1) \times (0,0,1)_{12}$ model to the data, compute forecasts and a prediction band 30 points ahead and plot the original data and the forecast with the prediction band. Fit the same data with function gausspr from package kernlab (use default settings). Plot the original data and predicted data from $t = 1$ to $t = 230$. Compare the two plots and make conclusions.

Since we should generate from the same model as in *1c*, we will use the same generated time series.

#### Fitting model and performing forecast using forecast-package.

In the first approach, we will use the package `forecast`.

```
# Loading forecast package.
library(forecast)

# Fitting model to time series using Arima() from forecast-package.
fit_Arima = Arima(y = ts,
                  order = c(0,0,1),
                  seasonal = list(order = c(0,0,1),
                                  period = 12))

# Performing forecast.
forecast_Arima = forecast(object = fit_Arima,
                          # Forecasting 30 points ahead.
                          h = 30,
                          # Setting confidence level for prediction intervals to 95%.
                          level = 95)

# Plotting results.
plot(forecast_Arima,
     ylab = "x_t",
     xlab = "t",
     main = "Forecasts using Arima() including 95% prediction interval")
```

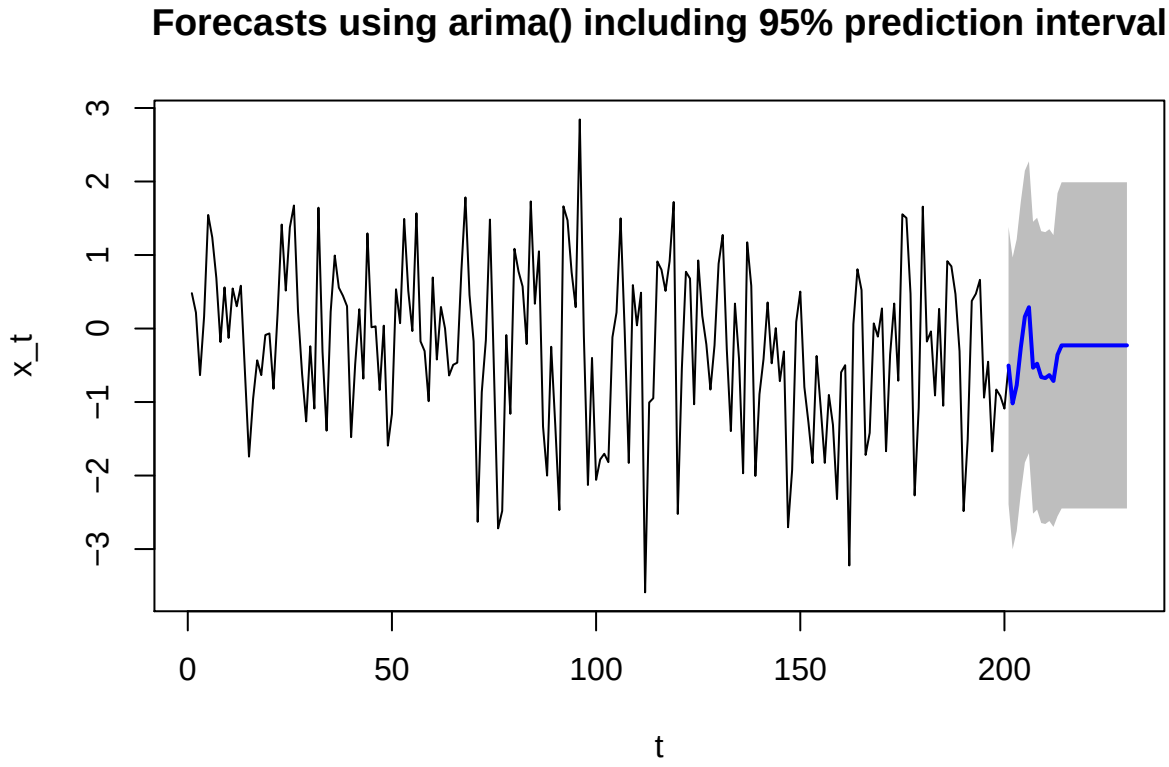



**Fitting model and performing forecast using built-in R functions.**

Another option is to fit the model and perform the foreacast using only built-in R functions. Since the function `predict()` only returns the values for the prediction and their standard errors, we need to create the plot on our own. Also, we need to calculate the prediction band on our own. For each predicted value $\hat{x}$, the prediction band is calculated by $\hat{x} \pm 1.96 * \sigma$. Since the function `predict()` returns the standard error which is defined as $se = \sqrt{\frac{\sigma^2}{n}}$, we get $\sigma = \sqrt{se^2 n}$. Since $n = 1$ for every single predicted value $\hat{x}$, $\sigma = se$ and therefore we get the prediction band by $\hat{x} \pm 1.96 * se$.

```
# Fitting model to time series using arima().
fit_arima = arima(x = ts,
                  order = c(0,0,1),
                  seasonal = list(order = c(0,0,1),
                                  period = 12))

# Performing forecast.
forecast_arima = predict(object = fit_arima,
                         # Forecasting 30 points ahead.
                         n.ahead = 30)

# Calculating prediction bands.
forecast_arima_upper_pb = forecast_arima$pred + 1.96*forecast_arima$se
forecast_arima_lower_pb = forecast_arima$pred - 1.96*forecast_arima$se

# Plotting results.
plot(c(ts, forecast_arima$pred),
```

```
     type = "l",
     ylab = "x_t",
     xlab = "t",
     main = "Forecasts using arima() including 95% prediction interval")
polygon(x = c(c(201:230), rev(c(201:230))),
        y = c(forecast_arima_lower_pb, rev(forecast_arima_upper_pb)),
        col = 'grey',
        border = NA)
lines(forecast_arima$pred,
      col = "blue",
      lwd = 2)
```



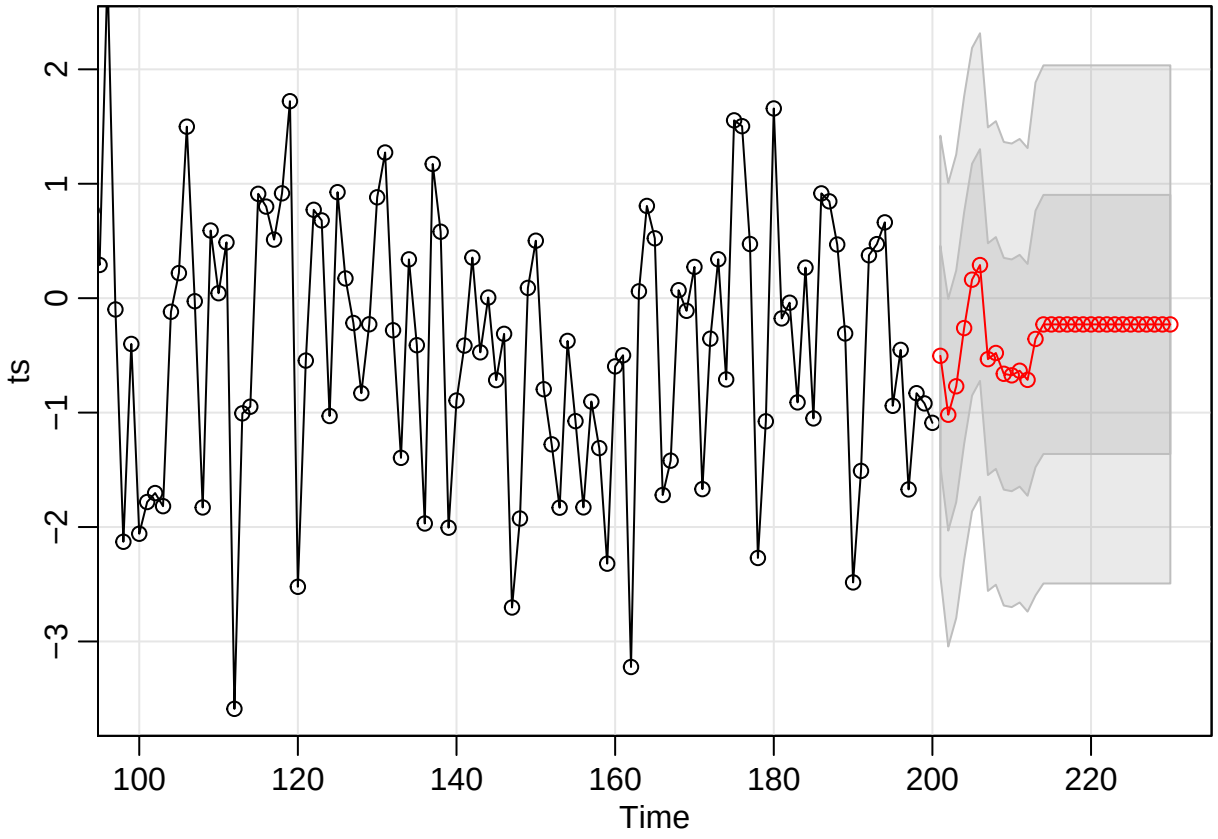

As it can be seen, the manual implementation using only built-in R functions returns the same plot as the one from the `forecast`-package.

**Fitting model and performing forecast using astsa-package.**

Another option could be to use the `astsa`-package. The output automatically prints the forecasts and the standard errors of the forecasts, and supplies a graphic of the forecast with +/- 1 and 2 prediction error bounds.

```
# Importing library.
library(astsa)

# Fitting model, performing forecast and plotting in once.
sarima.for(xdata = ts,
           n.ahead = 30,
```

```
          p = 0, d = 0, q = 1, # non-seasonal part
          Q = 1, S = 12) # seasonal part
```




```
$pred
Time Series:
Start = 201
End = 230
Frequency = 1
 [1] -0.5038231 -1.0185914 -0.7698338 -0.2613545  0.1618070  0.2895813
 [7] -0.5329064 -0.4787998 -0.6605170 -0.6743154 -0.6340145 -0.7137672
[13] -0.3562407 -0.2293420 -0.2293420 -0.2293420 -0.2293420 -0.2293420
[19] -0.2293420 -0.2293420 -0.2293420 -0.2293420 -0.2293420 -0.2293420
[25] -0.2293420 -0.2293420 -0.2293420 -0.2293420 -0.2293420 -0.2293420

$se
Time Series:
Start = 201
End = 230
Frequency = 1
 [1] 0.9619242 1.0127645 1.0127645 1.0127645 1.0127645 1.0127645 1.0127645
 [8] 1.0127645 1.0127645 1.0127645 1.0127645 1.0127645 1.1209171 1.1320303
[15] 1.1320303 1.1320303 1.1320303 1.1320303 1.1320303 1.1320303 1.1320303
[22] 1.1320303 1.1320303 1.1320303 1.1320303 1.1320303 1.1320303 1.1320303
[29] 1.1320303 1.1320303
```

Again, the function retuens the same predictions as before.

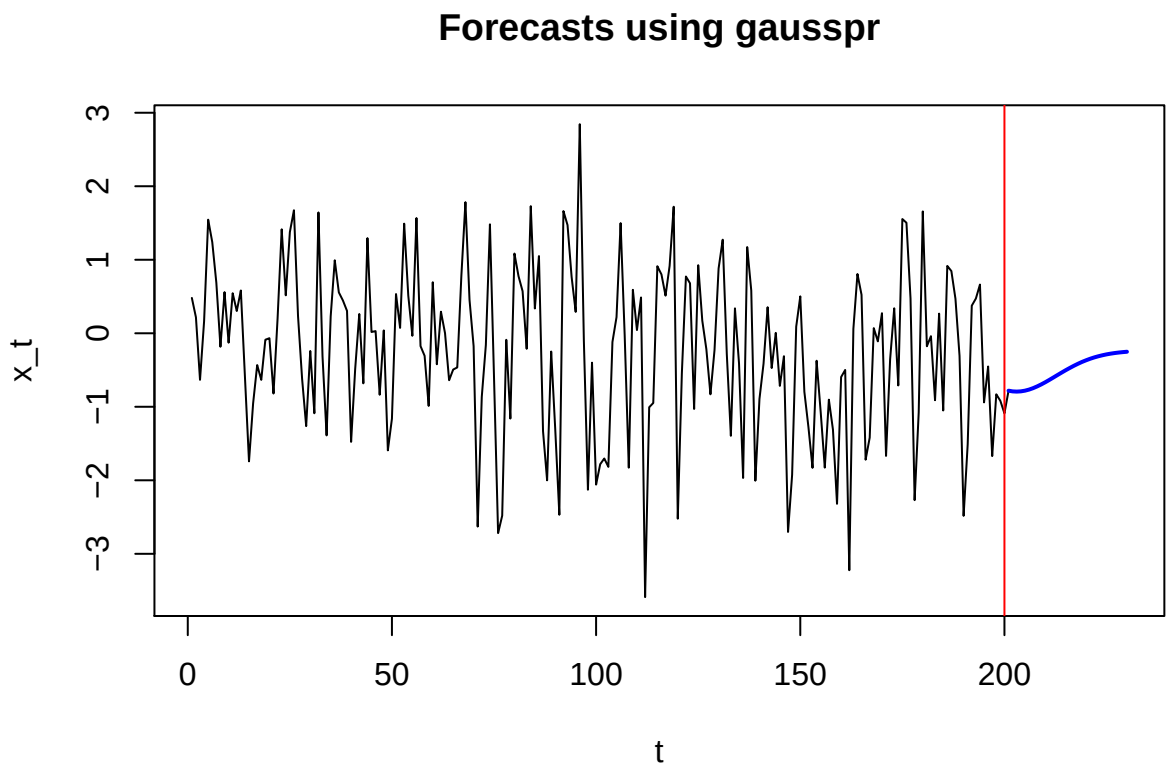
**Fitting model and performing forecast using kernlab-package.**

In contrast to the ARIMA models, we were also asked to fit the data using the `gausspr()` function from the kernlab-package.

```
# Loading kernlab package.
library(kernlab)

# Fitting model to time series using gausspr() from kernlab-package.
fit_gausspr = gausspr(x = c(1:200),
                      y = ts)
```

```
Using automatic sigma estimation (sigest) for RBF or laplace kernel
```

```
# Performing forecast.
forecast_gausspr = predict(fit_gausspr, c(201:230))

# Plotting results.
plot(c(ts, forecast_gausspr),
     type = "l",
     ylab = "x_t",
     xlab = "t",
     main = "Forecasts using gausspr")
lines(c(rep(NA, 200), forecast_gausspr),
      col = "blue",
      lwd = 2)
abline(v = 200, col = "red")
```



**Conclusions.**

First, as already concluded, the usage of the `forecast`-package return the same result as the own implementation using only built-in R functions. Comparing these arima-forecasts to the third approach (Gaussian process), it follows that in both cases the further the prediction time point goes the more the predictions approach the actual mean of the given time series. However, the prediction using ARIMA modelling clearly return a more realistic and varying prediction for the first time points.

**1e. Forecasting and comparing prediction interval with true values.**

Generate 50 observations from ARMA(1,1) process with $\phi = 0.7, \theta = 0.5$. Use first 40 values to fit an ARMA(1,1) model with $\mu = 0$. Plot the data, the 95% prediction band and plot also the true 10 values that you initially dropped. How many of them are outside the prediction band? How can this be interpreted?

**Generating time series for AR(2)-process.**

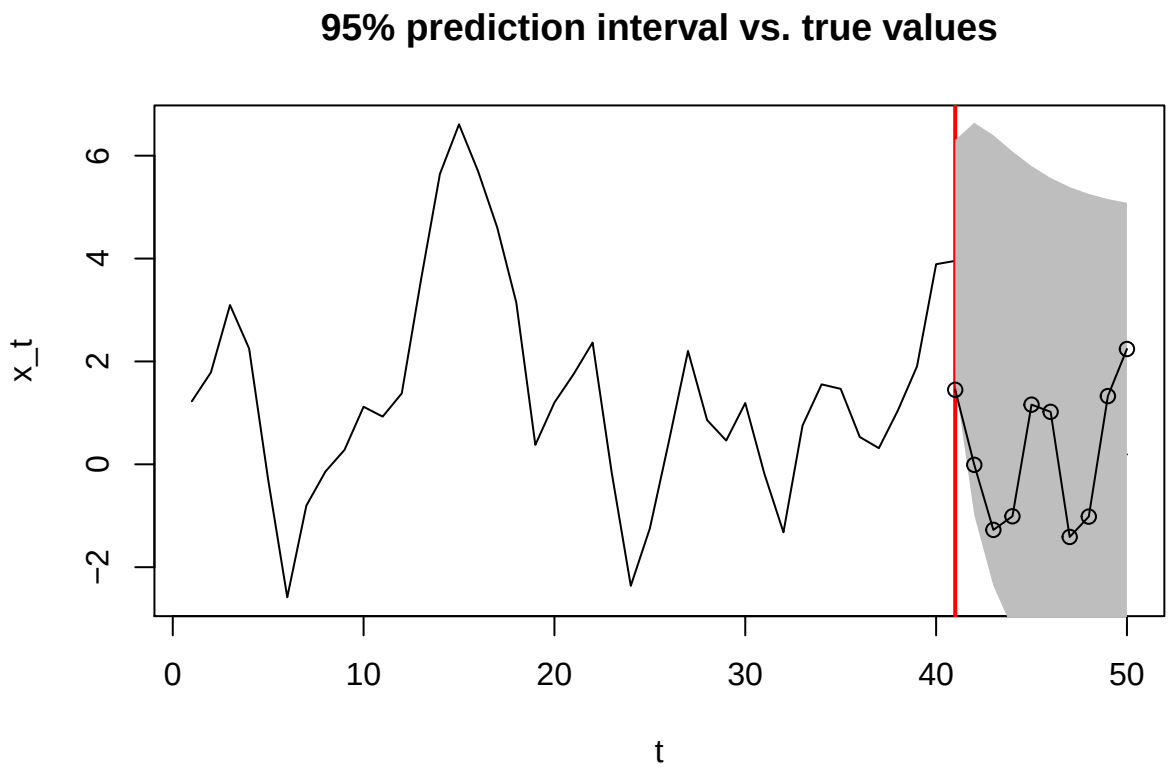
```
set.seed(12345)
ts = arima.sim(list(ar = 0.7,
                    ma = 0.5),
               n = 50)
```

**Fitting model with specified mean.**

```
fit_arima = arima(x = ts[1:40],
                  order = c(1,0,1),
                  include.mean = 0)
```

**Plotting data with prediction band.**

```
# Performing forecast.
forecast_arima = predict(object = fit_arima,
                         # Forecasting 10 points ahead.
                         n.ahead = 10)

# Calculating prediction bands.
forecast_arima_upper_pb = forecast_arima$pred + 1.96*forecast_arima$se
forecast_arima_lower_pb = forecast_arima$pred - 1.96*forecast_arima$se

# Plotting results.
plot(c(ts[1:40], forecast_arima$pred),
     type = "l",
     ylab = "x_t",
     xlab = "t",
     main = "95% prediction interval vs. true values")
abline(v = 41, col = "red", lwd = 2)
polygon(x = c(c(41:50), rev(c(41:50))),
        y = c(forecast_arima_lower_pb, rev(forecast_arima_upper_pb)),
        col = 'grey',
        border = NA)
points(c(rep(NA, 40), ts[41:50]))
lines(c(rep(NA, 40), ts[41:50]))
```




**Conclusions.**

The 95% prediction interval implies that 95% of all predictions should lie within the grey area. In our case, 10 out of 10 of the true values lie within the area which is a reasonable result.

## Assignment 2. ACF and PACF diagnostics.

### 2a|2b. Suggesting models based on ACF and PACF diagnostics.

a) For data series chicken in package astsa (denote it by $x_t$, plot 4 following graphs up to 40 lags: $\text{ACF}(x_t), \text{PACF}(x_t), \text{ACF}(\nabla x_t), \text{PACF}(\nabla x_t)$ (group them in one graph). Which $ARIMA(p,d,q)$ or $ARIMA(p,d,q) \times (P,D,Q)_s$ models can be suggested based on this information only? Motivate your choice.
b) Repeat step 1 for the following datasets: so2, EQcount, HCT in package astsa.

```
genPlots = function(x_t, dataset){
  par(mfrow=c(2,1),oma = c(0, 0, 2, 0))
  plot(x_t)
  plot(diff(x_t))
  mtext(paste(dataset, " dataset"), outer = TRUE, cex = 1.5)

  par(mfrow=c(2,2),oma = c(0, 0, 2, 0))
  acf(x_t, lag.max = 40, main="")
  pacf(x_t, lag.max = 40, main="")
  acf(diff(x_t), lag.max = 40, main="")
  pacf(diff(x_t), lag.max = 40, main="")
  mtext(paste(dataset, " dataset ACF and PACF plots"), outer = TRUE, cex = 1.5)
}
```

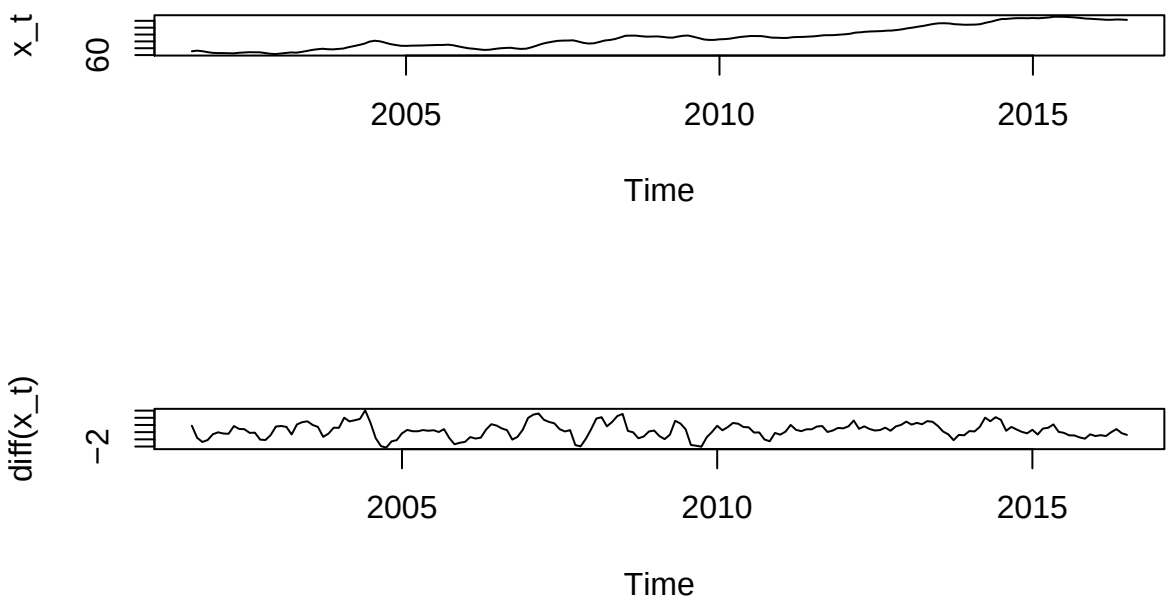
#### Datasets

#### Chicken

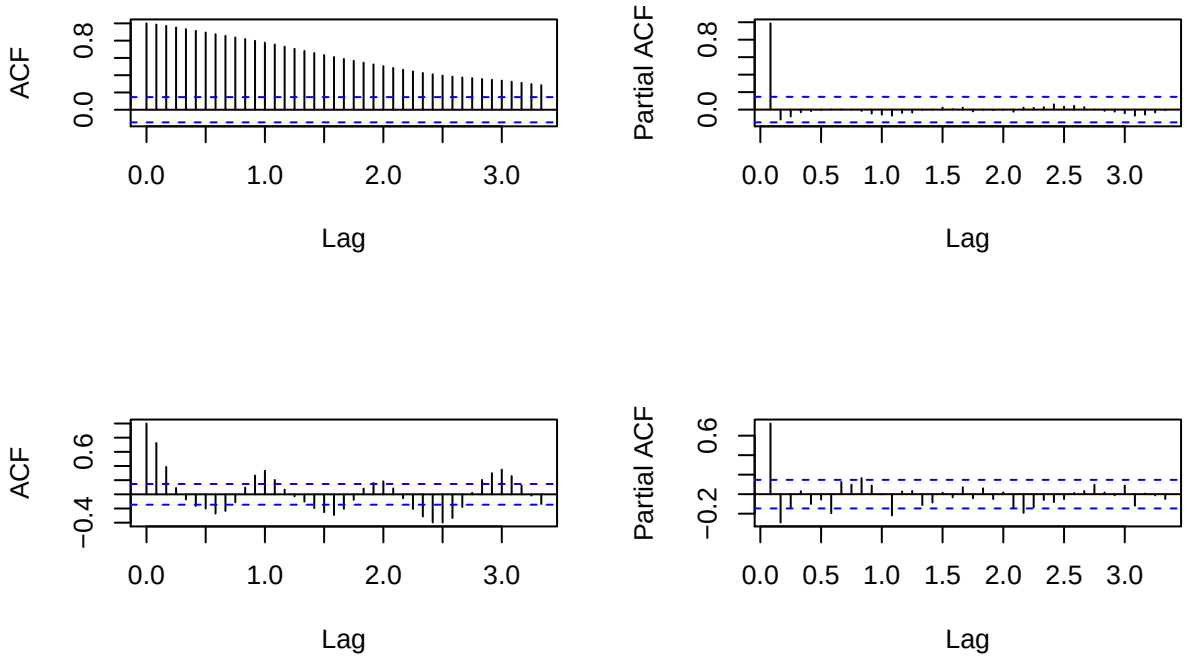
The decreasing trend in the ACF plot of chicken and the cutoff in the PCAF plot suggests that this could be an AR process. The positive ACF at lag 1 for the differenced data confirms that this is a AR process. The PCAF of the differenced data suggests that it is a AR(3) process, since there is a cutoff after lag 3 in the plot. We can see seasonality in the differenced dataset. The arima model $ARIMA(3,1,0)(3,1,0)_{12}$. would be good for this dataset.

```
genPlots(chicken, "Chicken")
```

# Chicken dataset



# Chicken  dataset ACF and PACF plots

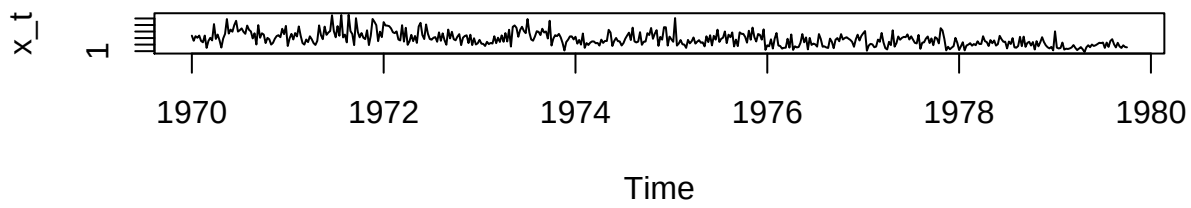


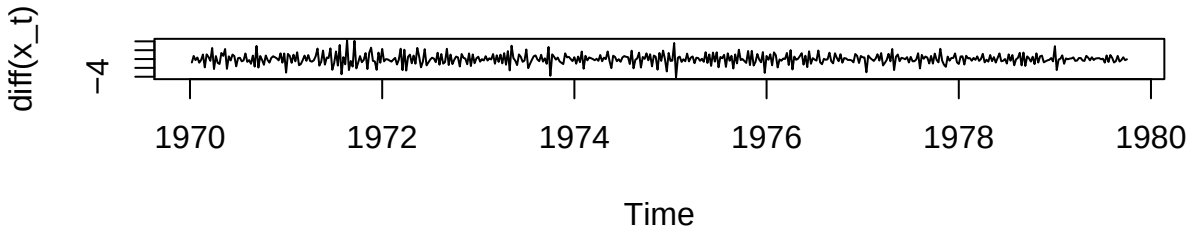

**so2**

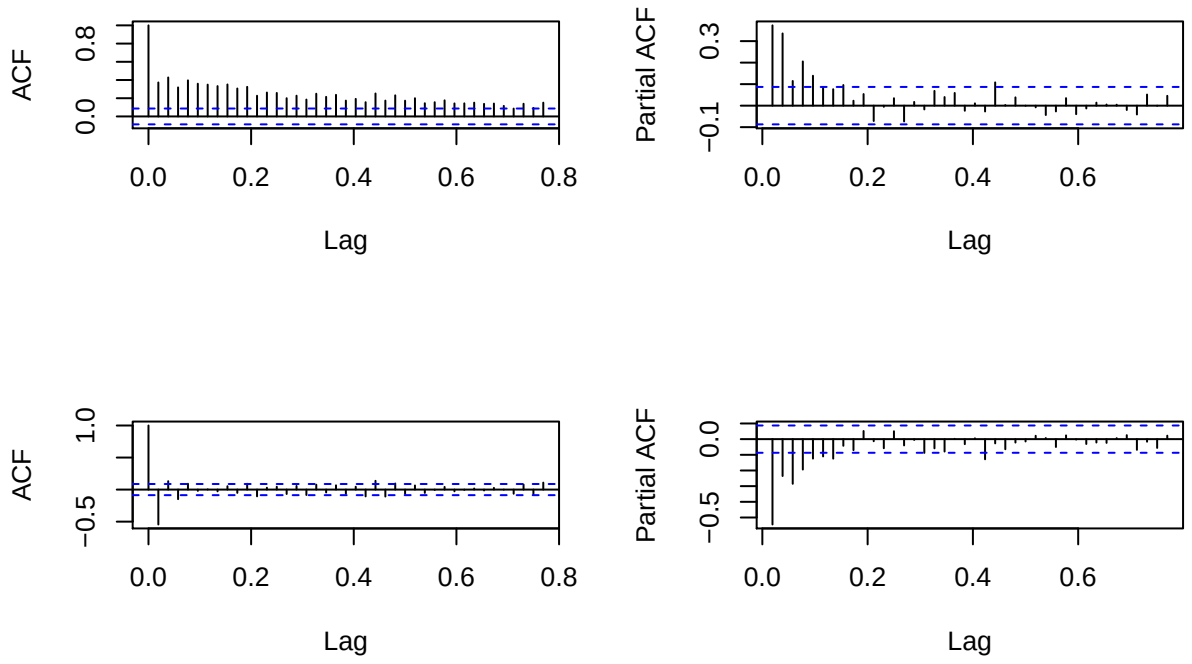
The decreasing trend in the ACF plot of the dataset suggests an ARIMA model could be a good fit for it. The negative ACF at lag 1 for the differenced dataset suggests to use an MA model. The pcaf plot of differenced dataset tells us that MA(7) model would be a good fit for the dataset. $ARIMA(0, 1, 7)$ would be a good model for this data.

```
genPlots(so2, "so2")
```

# so2 dataset


x_t
1
1970
1972
1974
1976
1978
1980
Time


diff(x_t)
−4
1970
1972
1974
1976
1978
1980
Time

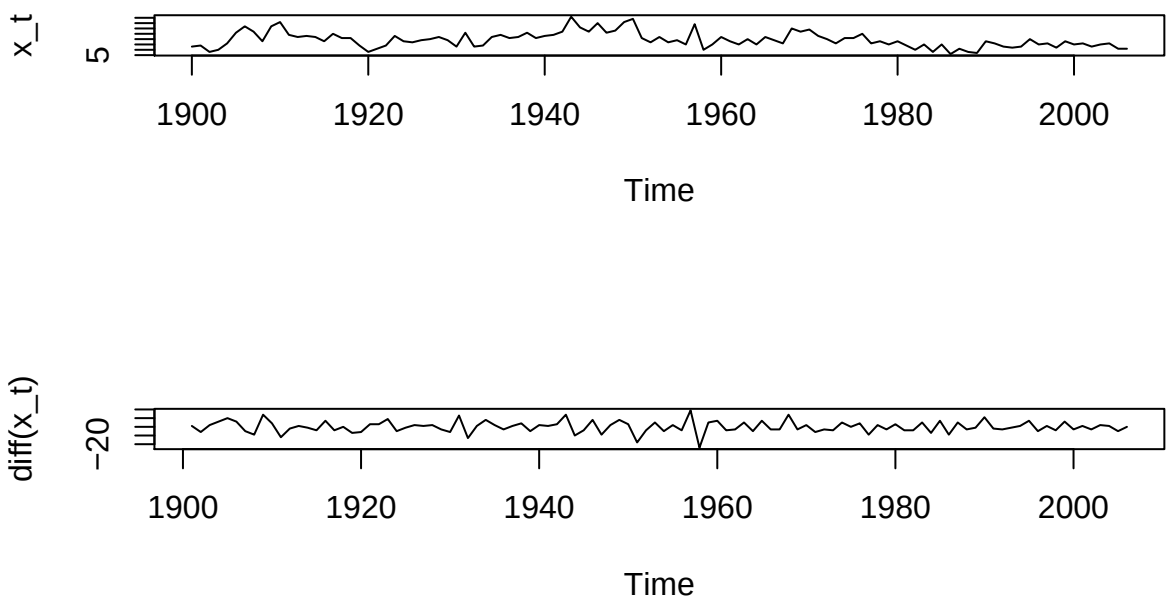


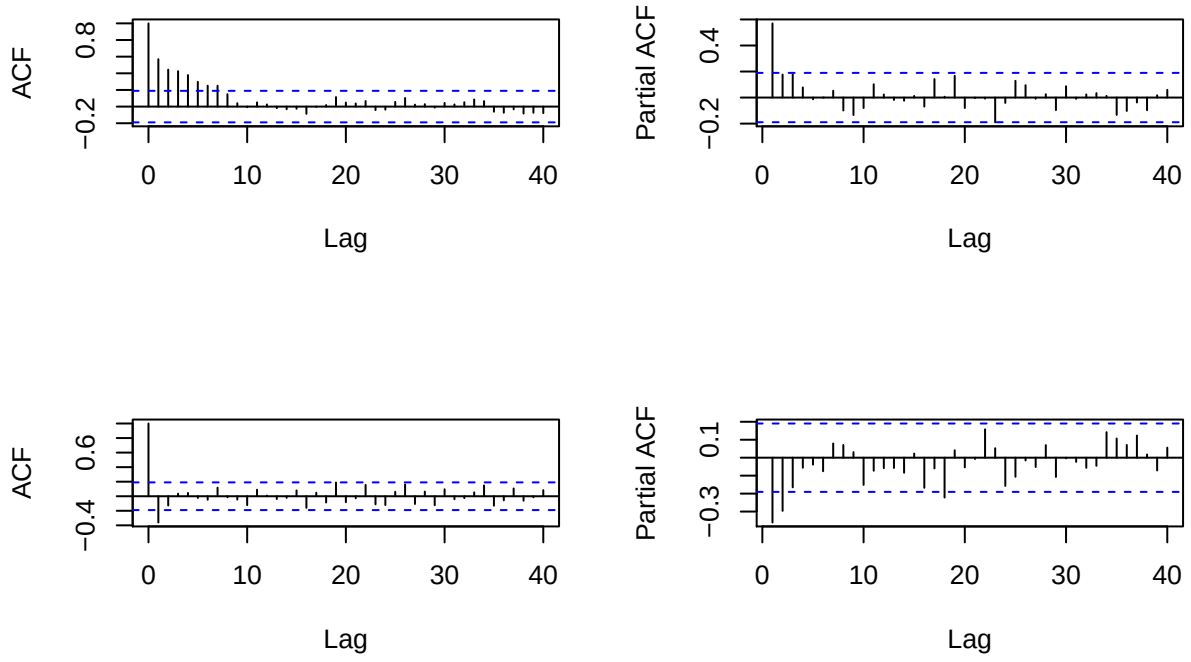

**EQcount**

The decreasing trend in the ACF plot of the dataset suggests an ARIMA model could be a good fit for it. The negative ACF at lag 1 for the differenced dataset suggests to use an MA model. The pcaf plot of differenced dataset tells us that MA(2) model would be a good fit for the dataset. $ARIMA(0, 1, 2)$ would be a good model for this data.

```
genPlots(EQcount, "EQCount")
```

# EQCount dataset

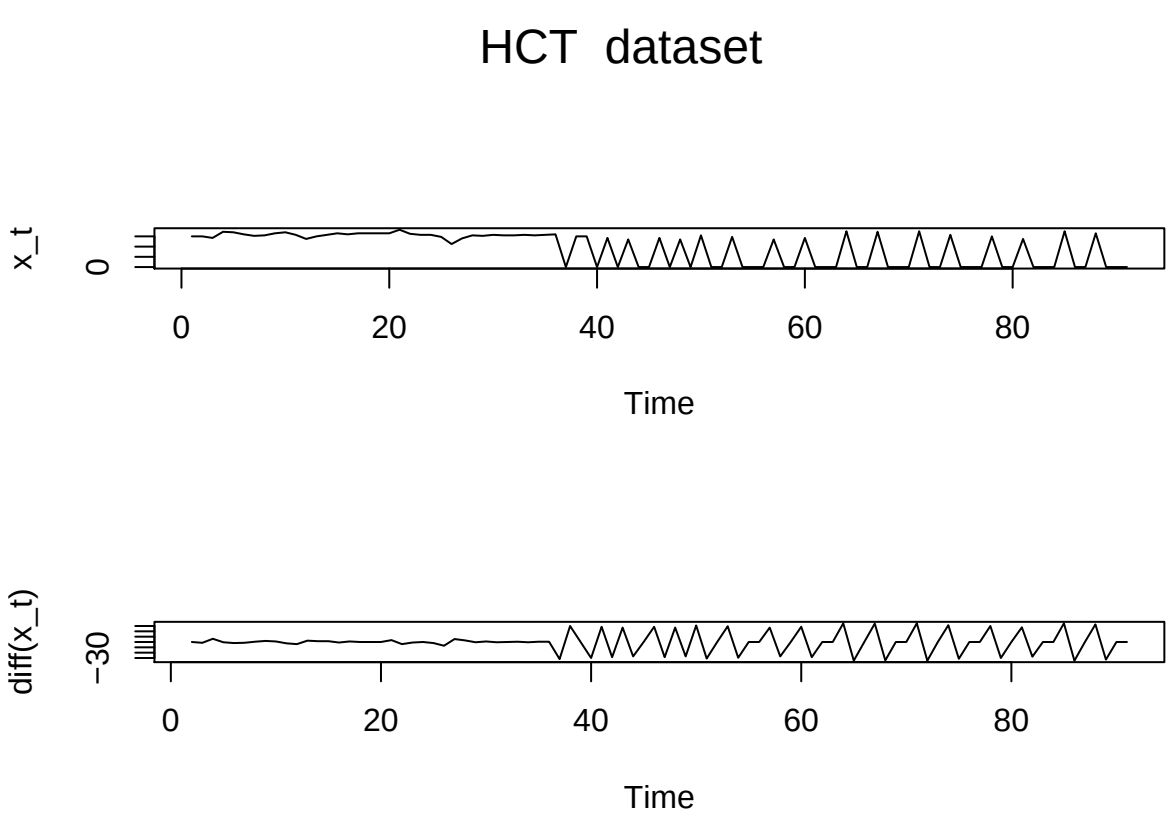




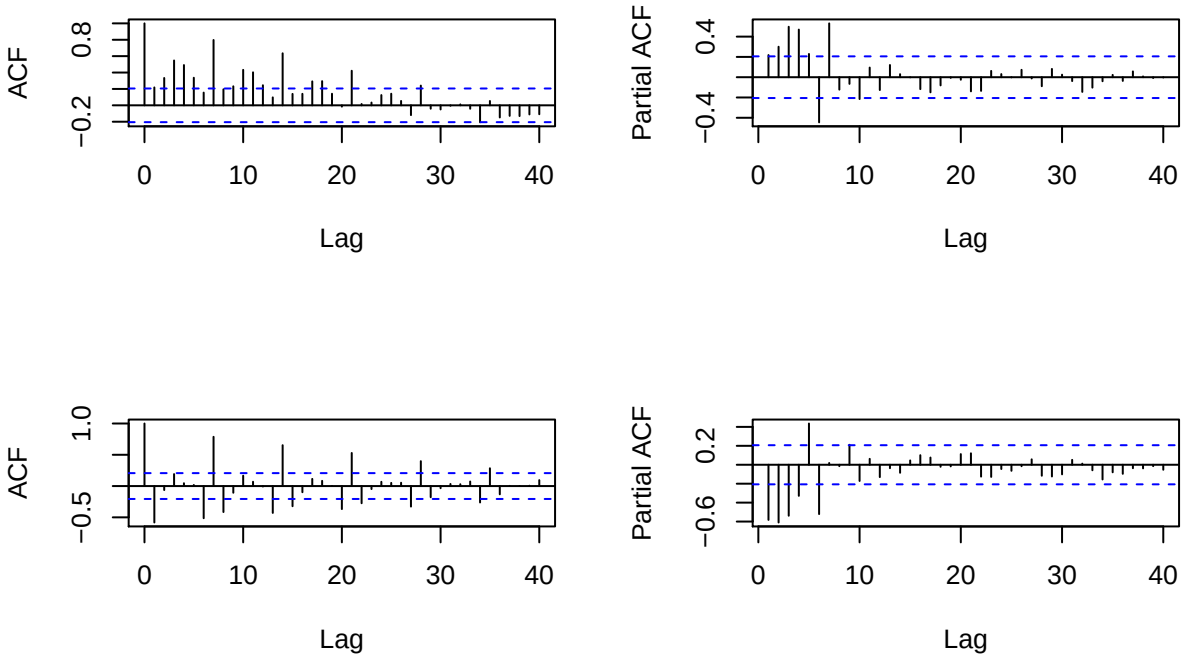

**HCT**

The decreasing trend in the ACF plot of the dataset suggests an ARIMA model could be a good fit for it. The negative ACF at lag 1 for the differenced dataset suggests to use an MA model. The pcaf plot of differenced dataset tells us that MA(7) model would be a good fit for the dataset. $ARIMA(0, 1, 7)$ would be a good model for this data. From the difference of order 1 we can see some seasonality after seven lags(seven days).

```
genPlots(HCT, "HCT")
```


HCT dataset
x_t
0
0
20
40
60
80
Time
diff(x_t)
−30
0
20
40
60
80
Time

# HCT  dataset ACF and PACF plots



## Assignment 3. ARIMA modeling cycle.

In this assignment, you are assumed to apply a complete ARIMA modeling cycle starting from visualization and detrending and ending up with a forecasting.

### 3a. Finding suitable non-seasonal ARIMA(p,d,q).

Find a suitable $ARIMA(p, d, q)$ model for the data set oil present in the library astsa. Your modeling should include the following steps in an appropriate order: visualization, unit root test, detrending by differencing (if necessary), transformations (if necessary), ACF and PACF plots when needed, EACF analysis, Q-Q plots, Box-Ljung test, ARIMA fit analysis, control of the parameter redundancy in the fitted model. When performing these steps, always have 2 tentative models at hand and select one of them in the end. Validate your choice by AIC and BIC and write down the equation of the selected model. Finally, perform forecasting of the model 20 observations ahead and provide a suitable plot showing the forecast and its uncertainty.

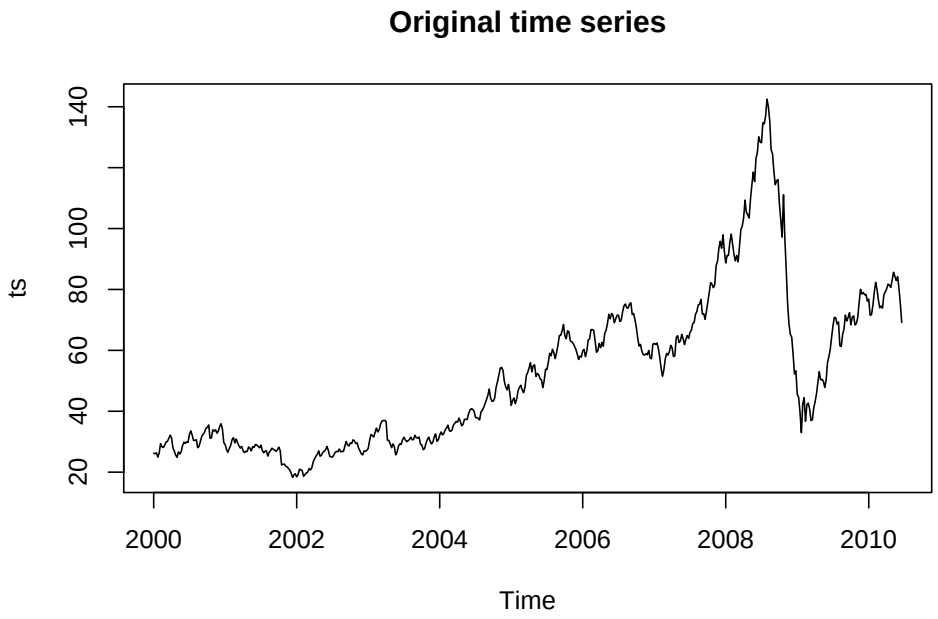
The follwoing complete ARIMA modeling cycle is performed by the following steps:

1. Analyzing original time series
2. Making time series stationary
3. Defining tentative models
4. Fitting models
5. Selecting model
6. Performing forecast using selected model

#### Analyzing original time series.

First, we will have a look at the original time series `oil`. It gives first answers to questions about stationarity or seasonality.

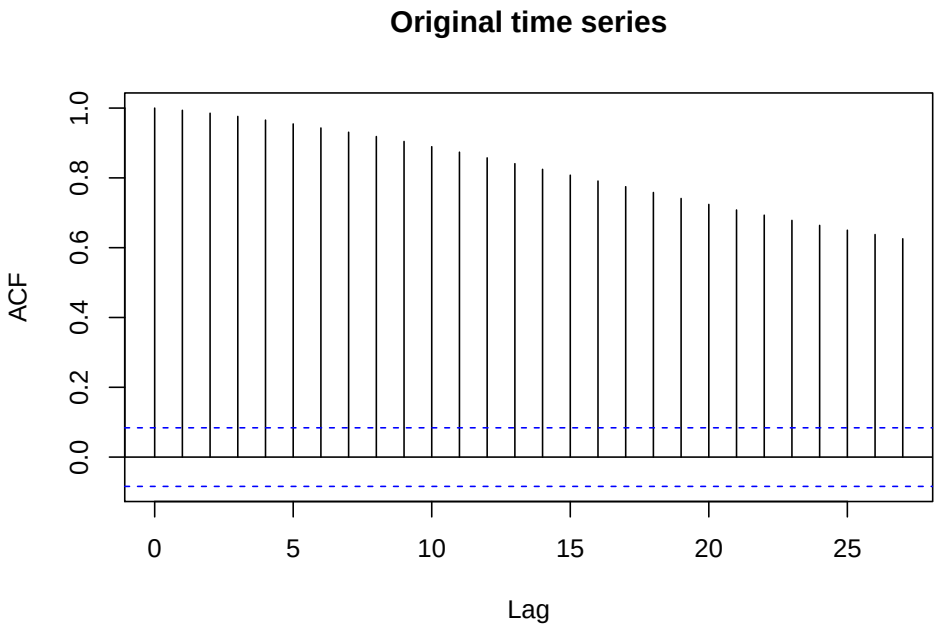
```
# Loading original time series.
library(astsa)
data(oil)
ts = oil

# Plotting original time series.
ts.plot(x = ts, main = "Original time series")
```




ARIMA-models require stationarity of the fitted time series. Since the mean seems to increase over time and is therefore not constant, it is obvious that the time series might not be not stationary. Furthermore, the variance of the time series differ a lot over time.

To check for more evidence of the stationarity, we can plot the sample ACF of the time series. Again, as in assignment 2, wrap the `ts()`-function around the time series before plotting the sample ACF. This will not change the time series itself, but somehow the follwing plot shows the correct values for the x-axis (lags up to 40 by steps of 1).

```
# Plotting sample ACF.
acf(ts(ts), main = "Original time series")
```




The approximate linear decay of the sample ACF as it can be seen here is often taken as a symptom that the underlying time series is nonstationary.

Next to the performed visible analysis of the staionarity, we can also quantify the evidence of nonstationarity using the *Dickey-Fuller Unit-Root Test*. Within this test, the alternative hypothesis is that the process is stationary.

```
# Peforming Dickey-Fuller Unit-Root Test.
library(tseries)
adf.test(x = ts,
         alternative = "stationary")
```

```
    Augmented Dickey-Fuller Test

data:  ts
Dickey-Fuller = -3.4217, Lag order = 8, p-value = 0.04983
alternative hypothesis: stationary
```

The test returns that with an $\alpha = 0.05$, we would assume that the data is stationary. However, combining the previous visual analysis and the fact that the p-value is very close to $\alpha$, we will analyze how detrending and transformation might lead to an even more stationary time series.

**Making time series stationary.**

We learned some tools which might help making a time series more stationary. Within this assignment, we are allowed to perform detrending by differencing and transformation. Using the original time series, we will test three other versions of it using these tools.

- first difference of the time series ($x'_t = \nabla x_t = x_t - x_{t-1}$)
- log of the time series ($x''_t = log(x_t)$)
- first difference of the log of the time series ($x'''_t = \nabla log(x_t) = log(x_t) - log(x_{t-1})$)

For each version, we will check for stationarity again with the help of the *Dickey-Fuller Unit-Root Test*.

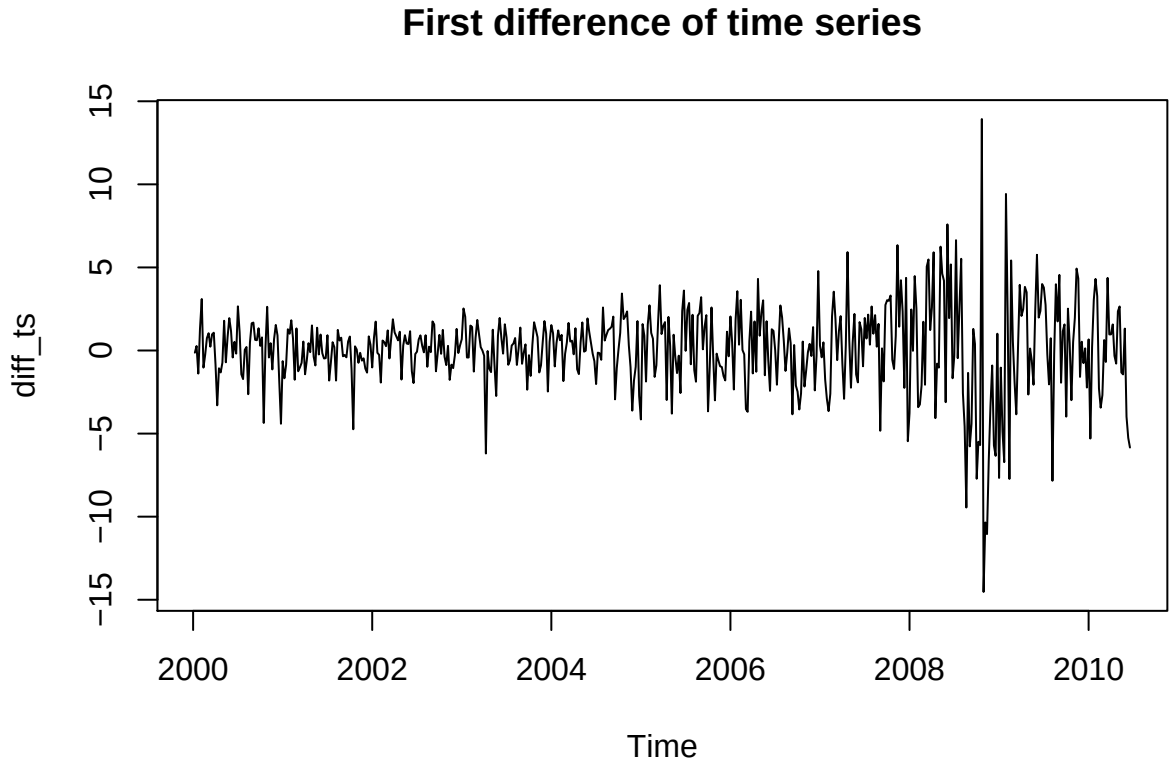
```r
# Differencing / Transforming original ts.
diff_ts = diff(ts)
log_ts = log(ts)
diff_log_ts = diff(log(ts))

# Peforming Dickey-Fuller Unit-Root tests for differenced/transformed ts.
knitr::kable(data.frame(diff_ts = adf.test(x = diff_ts,
                                           alternative = "stationary")$p.value,
                        log_ts = adf.test(x = log_ts,
                                          alternative = "stationary")$p.value,
                        diff_log_ts = adf.test(x = diff_log_ts,
                                               alternative = "stationary")$p.value),
             caption = "p-values after performing Dickey-Fuller Unit-Root tests")
```

Table 4: p-values after performing Dickey-Fuller Unit-Root tests

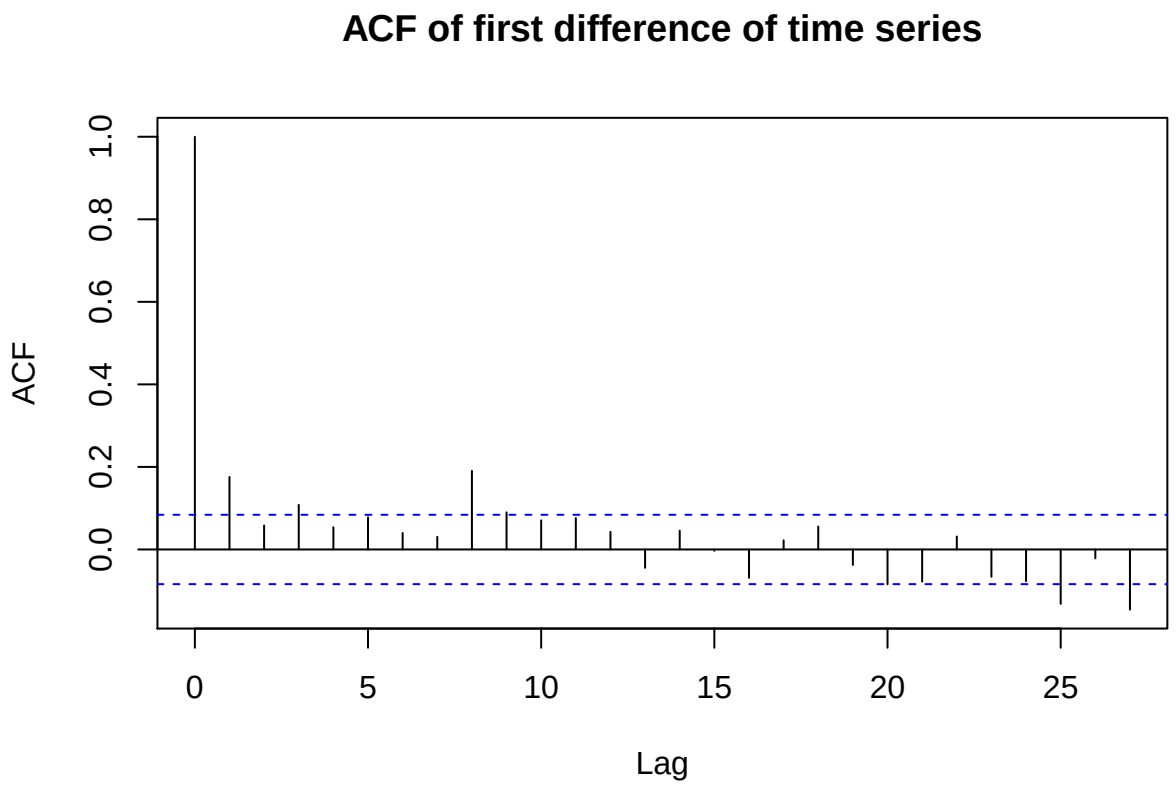
| diff_ts | log_ts | diff_log_ts |
|---|---|---|
| 0.01 | 0.2502782 | 0.01 |

It follows that the first difference and the first difference of the log led to the lowest p-values. Since the first difference (`diff_ts`) is closer to the original time series, we will visually analyze the resulting time series.

```r
# Plotting first difference of ts.
ts.plot(x = diff_ts, main = "First difference of time series")
```



```r
# Plotting sample ACF of first difference of ts.
acf(ts(diff_ts), main = "ACF of first difference of time series")
```

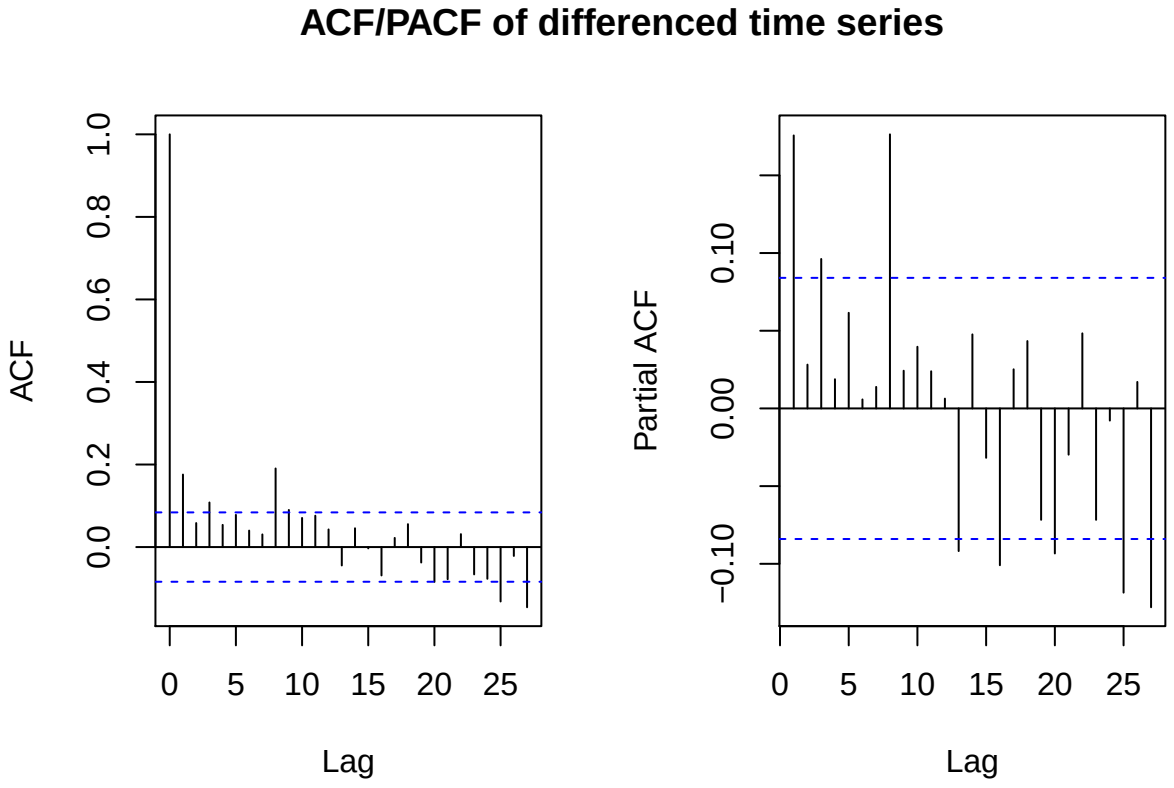



Both plots confirm that the time series looks much more stationary now. The AFC plot supports the suggestion about staionarity, because most of the peaks are inside the significance limits. For the further analysis, we will use this first difference of the original time series to perform ARIMA-modelling.

**Defining tentative models.**

To be able to define tentative ARIMA(p,d,q)-models, we use ACF and PACF plots and perform EACF (extended autocorrelation function) analysis. All can be used to specify the order of the ARMA part.

```r
# Plotting ACF and PACF.
  # Defining following plots to be next to each other.
  par(mfrow = c(1, 2))
  # Plotting.
  acf(x = ts(diff_ts), main = NA)
  pacf(x = ts(diff_ts), main = NA)
  # Defining shared title.
  title("ACF/PACF of differenced time series", line = -2, outer = T)
```




```
# Computing empirical ACF (EACF) to find reasonable AR-/MA-parameter.
library(TSA)
eacf(diff_ts)
```

```
AR/MA
  0 1 2 3 4 5 6 7 8 9 10 11 12 13
0 x o x o o o o x x o o  o  o  o
1 x o o o o o o x o o o  o  o  o
2 x x o o o o o x o o o  o  o  o
3 x x x o o o o x o o o  o  o  o
4 x x x o o o o x o o o  o  o  o
5 x x o o o o o x o o o  o  o  o
6 x o x o x o o x o o o  o  o  o
7 o o x o x o x x o o o  o  o  o
```

The EACF suggests an ARMA(0,1) model as a good option. Also, the ARMA(1,1) seem to be a reasonable choice according to the EACF. Since both the ACF and PACF plot did not give us any better idea for another model, we will follow these two suggestions. Combined with the fact that we are working with the first difference of the original time series ($d = 1$), we will investigate the follwing two non-seasonal ARIMA-models:

$$ARIMA(0, 1, 1)$$
$$ARIMA(1, 1, 1)$$

**Fitting models.**

To fit the models, we will use the `forecast`-package. The other options (e.g. using only built-in R functions) are presented in assignment 1. We learned about the different methods to fit the time series (Method of

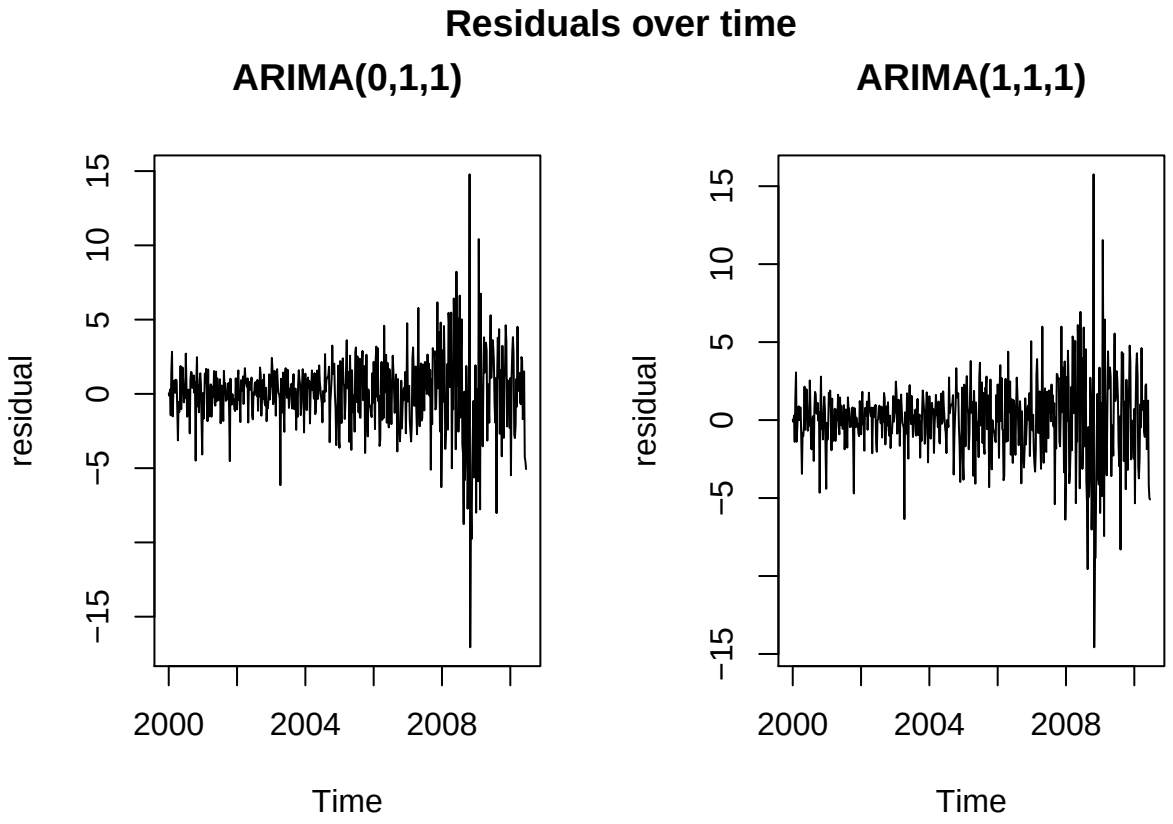
moments, conditional least squares, maximum likelihood). Since no information is given, we will use the Maximum likelihood method. We will fit the models on the original time series. The parameter $d$ will be set to 1, so that differencing the data will be considered.

```
# Loading forecast package.
library(forecast)

# Fitting models.
  # ARIMA(0,1,1).
  arima_0_1_1 = Arima(y = ts,
                      order = c(0,1,1),
                      method = "ML")
  # ARIMA(1,1,1).
  arima_1_1_1 = Arima(y = ts,
                      order = c(1,1,1),
                      method = "ML")
```

**Selecting model.**

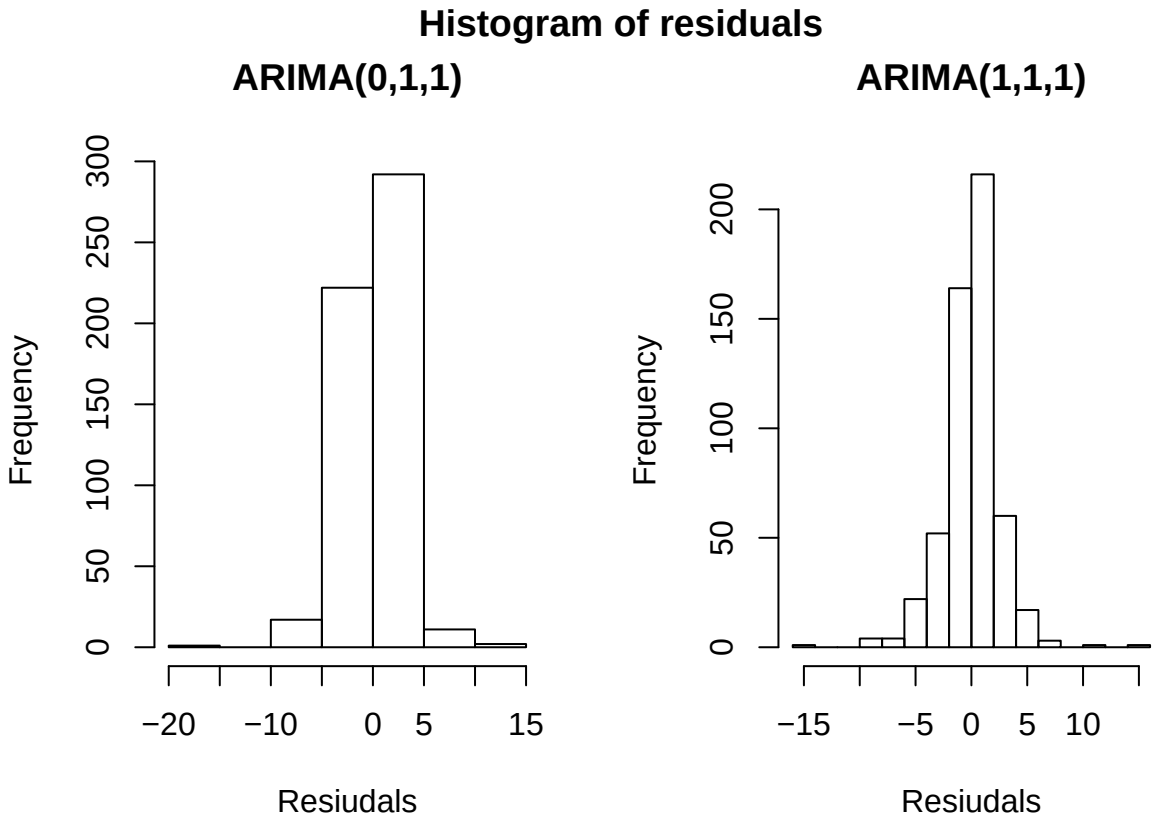
Since our goal is to decide for one of the two tentative models, we will perform residual analysis which will give us and idea about which model seems to be more promising.

```
# Plotting residuals.
  # Defining following plots to be next to each other.
  par(mfrow = c(1, 2))
  # Plotting.
  ts.plot(arima_0_1_1$residuals, ylab = "residual", main = "ARIMA(0,1,1)")
  ts.plot(arima_1_1_1$residuals, ylab = "residual", main = "ARIMA(1,1,1)")
  # Defining shared title.
  title("Residuals over time", line = -1, outer = T)
```



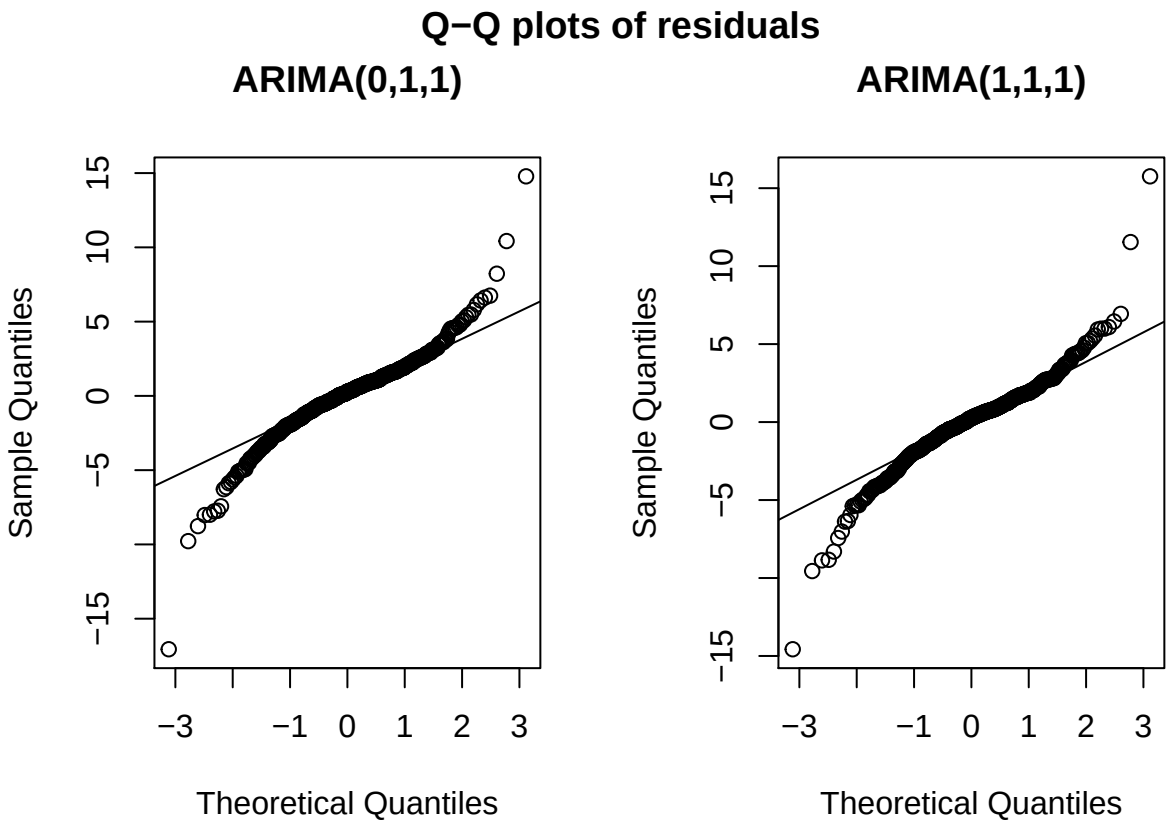

Since the residuals over time look random (white noise) for both models, it is an indication that both fitted models seem to be a promising choice. However, the plots do not return a better idea about which model seems to be the better choice.

```
# Plotting histogram of residuals.
  # Defining following plots to be next to each other.
  par(mfrow = c(1, 2))
  # Plotting.
  hist(arima_0_1_1$residuals, main = "ARIMA(0,1,1)", xlab = "Resiudals")
  hist(arima_1_1_1$residuals, main = "ARIMA(1,1,1)", xlab = "Resiudals")
  # Defining shared title.
  title("Histogram of residuals", line = -1, outer = T)
```



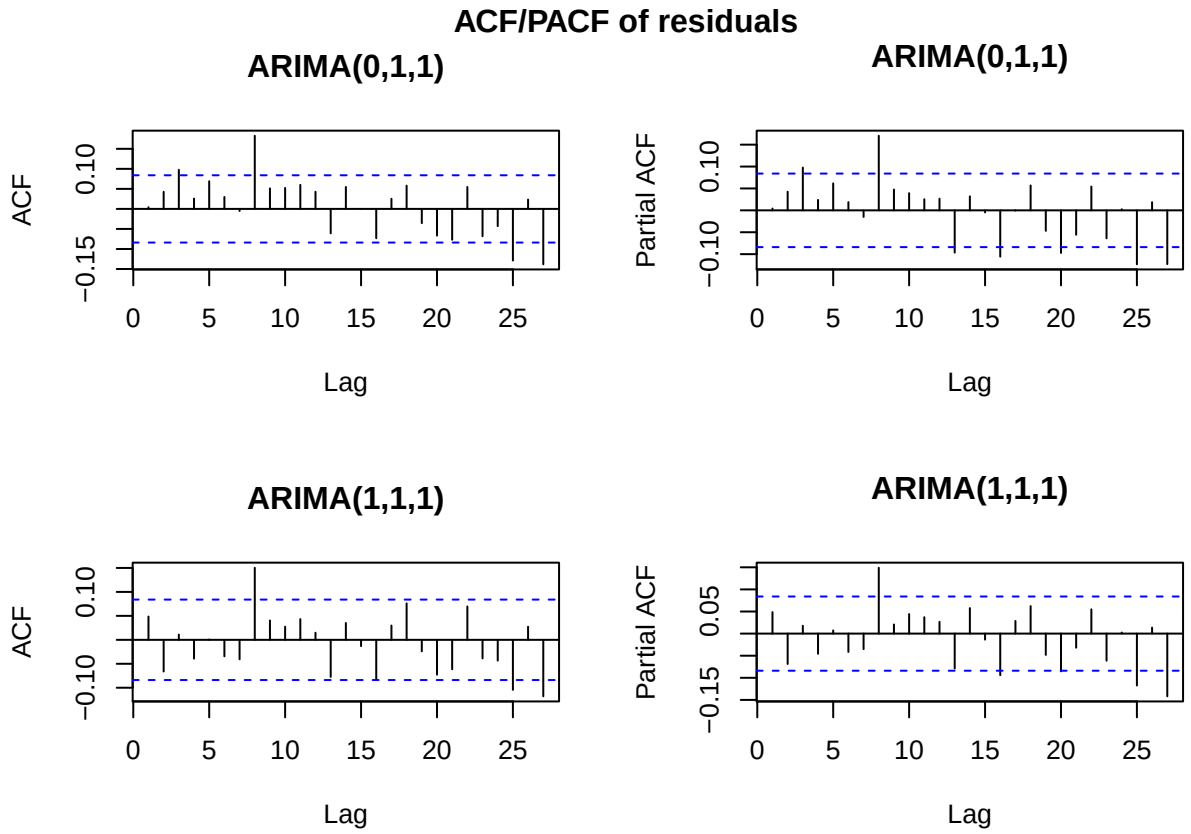

Both histograms reprent at least approximately a normal distribution.

```
# Plotting Q-Q plots of residuals.
  # Defining following plots to be next to each other.
  par(mfrow = c(1, 2))
  # Plotting.
  qqnorm(arima_0_1_1$residuals, main = "ARIMA(0,1,1)")
  qqline(arima_0_1_1$residuals)
  qqnorm(arima_1_1_1$residuals, main = "ARIMA(1,1,1)")
  qqline(arima_1_1_1$residuals)
  # Defining shared title.
  title("Q-Q plots of residuals", line = -1, outer = T)
```




Again, we analyze the nornmal distribution by usage of the QQ-normal plots. If the data is normally distributed, the points in the QQ-normal plot lie on the straight diagonal line. Comparing these plots for both models, in both cases a normal distribution of the residuals is not clearly identifiable since the sample quantiles differ sometimes quite a lot from the theoretical quantiles even if most of the points are at least close to the diagonal line for both plots. Since they do not differ that much, it is hard to say which model seems to be a bette choice according the the plots.

```
# Plotting ACF/PACF of residuals.
  # Defining following plots to be next to each other.
  par(mfrow = c(2, 2))
  # Plotting.
  acf(ts(arima_0_1_1$residuals), main = "ARIMA(0,1,1)")
  pacf(ts(arima_0_1_1$residuals), main = "ARIMA(0,1,1)")
  acf(ts(arima_1_1_1$residuals), main = "ARIMA(1,1,1)")
  pacf(ts(arima_1_1_1$residuals), main = "ARIMA(1,1,1)")
  # Defining shared title.
  title("ACF/PACF of residuals", line = -1, outer = T)
```




For all shown plots (ACF and PACF for both models), almost all values of the (partial) autocorrelation lies within the blue range and can be therefore seen as non-significant. Except from the 8th lag, the residuals therefore seem to independent of each other.

To test this independence between the lags of the residuals quantitatively, we will perform the statistical *Runs test*. Since the alternative hypothesis implies that the values are not i.i.d, the residuals can be assumed to be independent for a resulting small p-value ($< 0.05$ if we assume $\alpha = 0.05$).

```
# Computing runs test for randomness of residuals.
library(TSA)
knitr::kable(data.frame(model = c("ARIMA_0_1_1", "ARIMA_1_1_1"),
                        p_value = c(runs(arima_0_1_1$residuals)$pvalue,
                                    runs(arima_1_1_1$residuals)$pvalue)),
             caption = "Results of Runs test")
```

Table 5: Results of Runs test

| model | p_value |
|---|---|
| ARIMA_0_1_1 | 0.870 |
| ARIMA_1_1_1 | 0.206 |

According to the runs test, the residuals of the $ARIMA(1,1,1)$-model are much more independent than the residals of the $ARIMA(0,1,1)$-model.

Box-Ljung test is a type of statistical test of whether any of a group of autocorrelations of a time series are different from zero. Instead of testing randomness at each distinct lag, it tests the "overall" randomness

based on a number of lags. The alternative hypothesis is the same as for the runs test (values are not i.i.d), so again we assume independece for a resulting small p-value ($< 0.05$ if we assume $\alpha = 0.05$).

```
# Performing Ljung-Box test.
knitr::kable(data.frame(model = c("ARIMA_0_1_1", "ARIMA_1_1_1"),
                        p_value = c(Box.test(x = arima_0_1_1$residuals,
                                             type = "Ljung-Box")$p.value,
                                    Box.test(x = arima_1_1_1$residuals,
                                             type = "Ljung-Box")$p.value)),
             caption = "Results of Ljung-Box test")
```

Table 6: Results of Ljung-Box test

| model | p_value |
|---|---|
| ARIMA_0_1_1 | 0.9221396 |
| ARIMA_1_1_1 | 0.2550042 |

The result confirms the results of the runs test and the residuals of the $ARIMA(1,1,1)$-model are assumed to be more independent.

Next to the performed residuals analysis, another way to suggest which fitted model might be better is given by looking at the scores: AIC, BIC. Both scores refer to the likelihood of the time series given the fitted model, but penalize each model according to its number of parameters.

```
# Comparing AIC, BIC between the models.
knitr::kable(data.frame(model = c("ARIMA_0_1_1", "ARIMA_1_1_1"),
                        AIC = c(arima_0_1_1$aic, arima_1_1_1$aic),
                        BIC = c(arima_0_1_1$bic, arima_1_1_1$bic)),
             caption = "Comparison of AIC, BIC")
```

Table 7: Comparison of AIC, BIC

| model | AIC | BIC |
|---|---|---|
| ARIMA_0_1_1 | 2567.297 | 2575.895 |
| ARIMA_1_1_1 | 2561.818 | 2574.715 |

For both cases (AIC, BIC), the $ARIMA(1,1,1)$ returns a better score.

All information combined lead us to the assumption that the $ARIMA(1,1,1)$-model might be a better fit to our differenced time series. Since we can extract the coefficient of the fitted model

```
arima_1_1_1$coef
```

```
       ar1        ma1
 0.8750497 -0.7711252
```

we can write down the equation of the model like this:

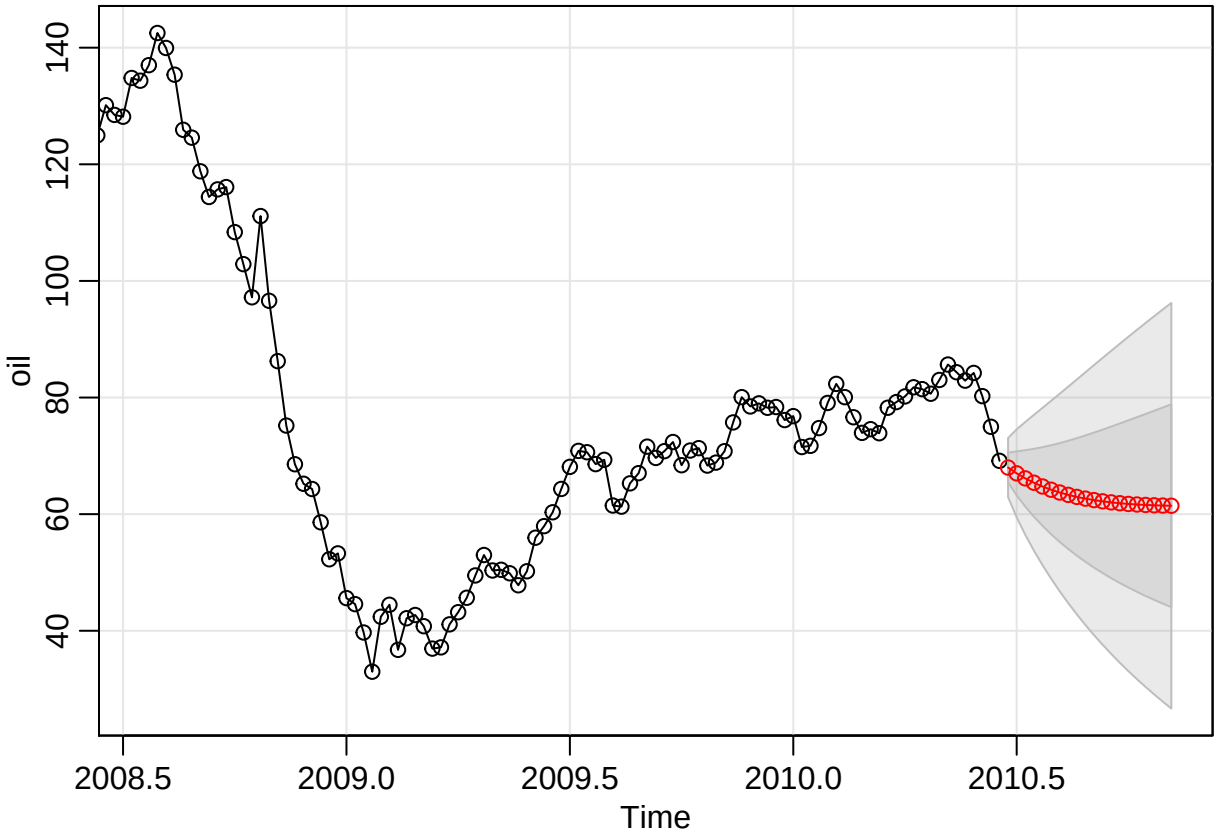
$$(1 - 0.8750497B)x_t = (1 - 0.7711252B)w_t$$

We will select this model and use it for the following forecast.

**Performing forecast using selected model.**

The goal is to forecast 20 observations ahead using our selected $ARIMA(0,1,1)$-model. Furthermore, we will implement a plot showing the forecast and its uncertainty by including a 95% prediction interval. We fitted the model using the `forecast`-package. However, we will make use of the `astsa`-package to perform the forecast. The reason is that we need to perfom the forecast not for the differenced time series, but for the original time series. The `sarima.for()` function provides the possibility to set the data which should be used for the forecast.

```
# NOT SURE ABOUT THIS. BECAUSE IT WILL FIT THE MODEL AGAIN ON ORIGINAL DATA, NOT DIFFERENED DATA.
# Performing forecast using selected model.
sarima.for(oil,
           n.ahead = 20,
           1,1,1)
```




```
$pred
Time Series:
Start = c(2010, 26)
End = c(2010, 45)
Frequency = 52
 [1] 67.99167 66.99612 66.13406 65.38866 64.74522 64.19086 63.71436
 [8] 63.30589 62.95688 62.65982 62.40817 62.19619 62.01888 61.87188
[15] 61.75135 61.65396 61.57679 61.51729 61.47323 61.44267

$se
Time Series:
Start = c(2010, 26)
End = c(2010, 45)
Frequency = 52
```

```
 [1]  2.534435  3.774577  4.838242  5.815883  6.738162  7.618877  8.465334
 [8]  9.281896 10.071456 10.836115 11.577523 12.297059 12.995927 13.675212
[15] 14.335915 14.978967 15.605242 16.215568 16.810725 17.391452
```

### 3b. Finding suitable multiplicative ARIMA $(p, d, q) \times (P, D, Q)_s$

Find a suitable $ARIMA(p, d, q) \times (P, D, Q)_s$ model for the data set unemp present in the library astsa. Your modeling should include the following steps in an appropriate order: visualization, detrending by differencing (if necessary), transformations (if necessary), ACF and PACF plots when needed, EACF analysis, Q-Q plots, Box-Ljung test, ARIMA fit analysis, control of the parameter redundancy in the fitted model. When performing these steps, always have 2 tentative models at hand and select one of them in the end. Validate your choice by AIC and BIC and write down the equation of the selected model (write in the backshift operator notation without expanding the brackets). Finally, perform forecasting of the model 20 observations ahead and provide a suitable plot showing the forecast and its uncertainty.

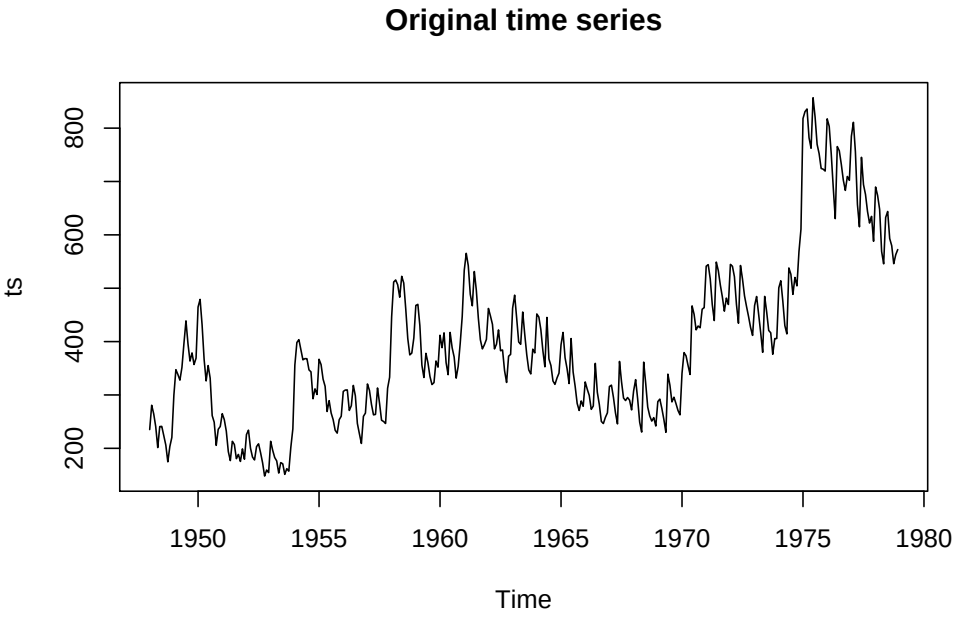
Like in assignment *3a*, we will repeat the same process. However, this time we need to find a suitable multicplicative seasonal ARIMA. The explanations for each step will be shortened this time, because we have explained the single steps already in assignment *3a* already.

The follwoing complete ARIMA modeling cycle is performed by the following steps:

1. Analyzing original time series
2. Making time series stationary
3. Defining tentative models
4. Fitting models
5. Selecting model
6. Performing forecast using selected model

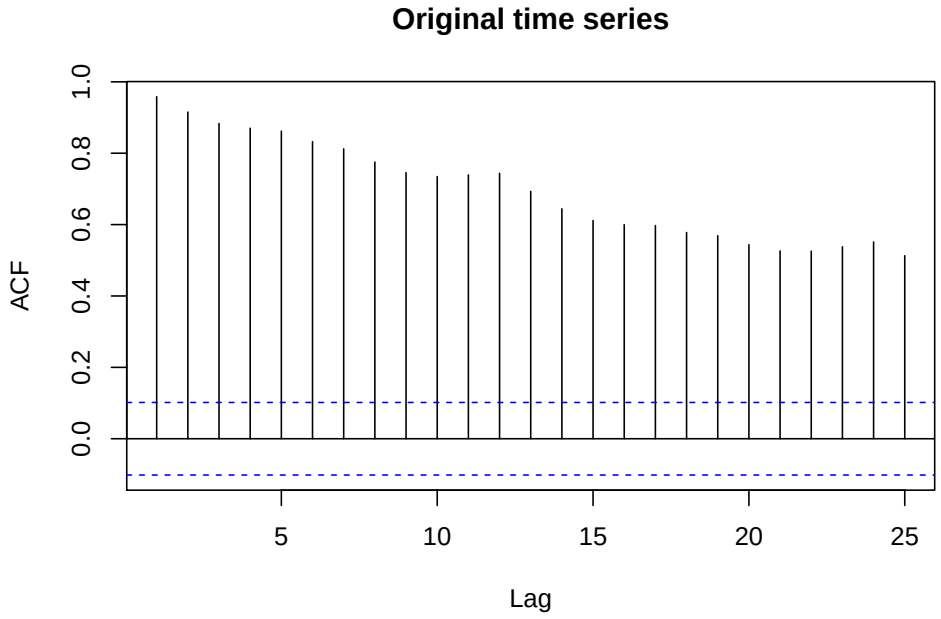
**Analyzing original time series.**

```
# Loading original time series.
library(astsa)
data(unemp)
ts = unemp

# Plotting original time series.
ts.plot(x = ts, main = "Original time series")
```



- Data clearly not stationary, since the mean is varying over time
- Seasonal pattern is visible

```
# Plotting sample ACF.
acf(ts(ts), main = "Original time series")
```




- ACF plot confirms lack of stationarity within the original time series

```
# Peforming Dickey-Fuller Unit-Root Test.
library(tseries)
adf.test(x = ts,
         alternative = "stationary")
```

```
    Augmented Dickey-Fuller Test

data:  ts
Dickey-Fuller = -3.5175, Lag order = 7, p-value = 0.04112
alternative hypothesis: stationary
```

The test returns that with an $\alpha = 0.05$, we would assume that the data is stationary. However, combining the previous visual analysis and the fact that the p-value is very close to $\alpha$, we will analyze how detrending and transformation might lead to an even more stationary time series.

**Making time series stationary.**

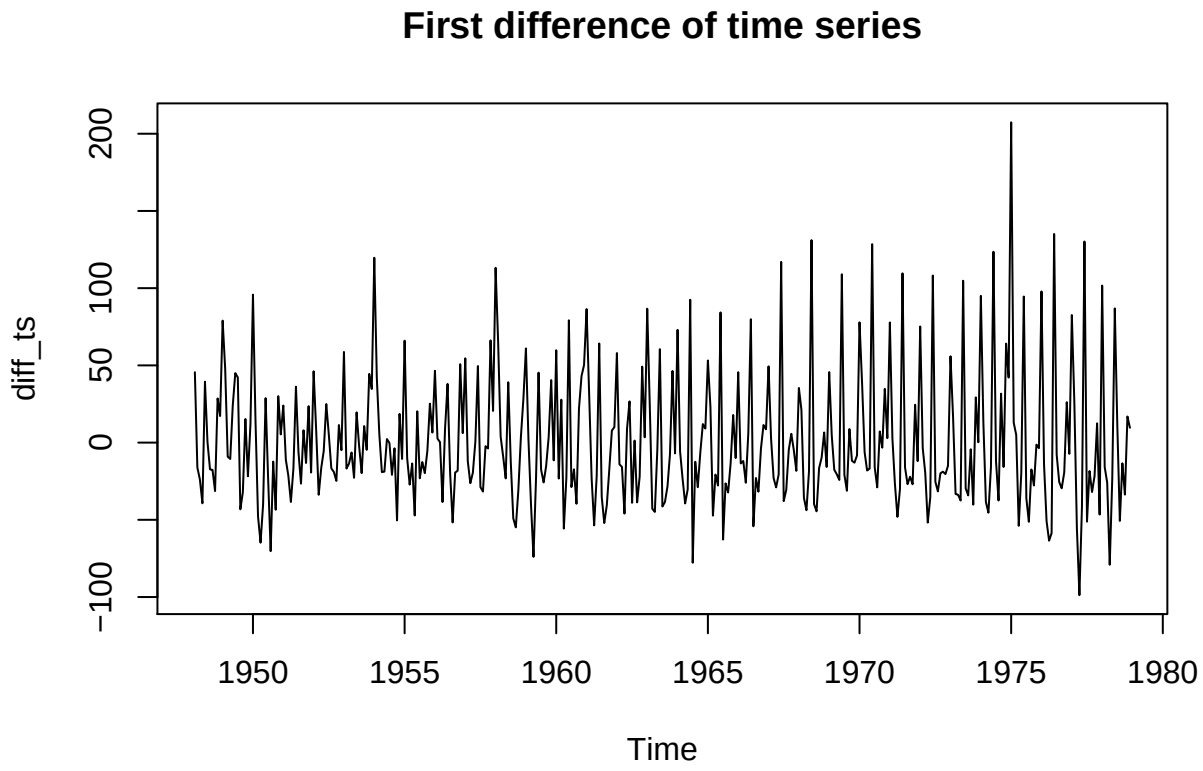
```
# Differencing / Transforming original ts.
diff_ts = diff(ts)
log_ts = log(ts)
diff_log_ts = diff(log(ts))

# Peforming Dickey-Fuller Unit-Root tests for differenced/transformed ts.
knitr::kable(data.frame(diff_ts = adf.test(x = diff_ts,
                                           alternative = "stationary")$p.value,
```

```
                    log_ts = adf.test(x = log_ts,
                                      alternative = "stationary")$p.value,
                    diff_log_ts = adf.test(x = diff_log_ts,
                                           alternative = "stationary")$p.value),
          caption = "p-values after performing Dickey-Fuller Unit-Root tests")
```

Table 8: p-values after performing Dickey-Fuller Unit-Root tests

| diff_ts | log_ts | diff_log_ts |
|---|---|---|
| 0.01 | 0.0231535 | 0.01 |

```
# Plotting first difference of ts.
ts.plot(x = diff_ts, main = "First difference of time series")
```



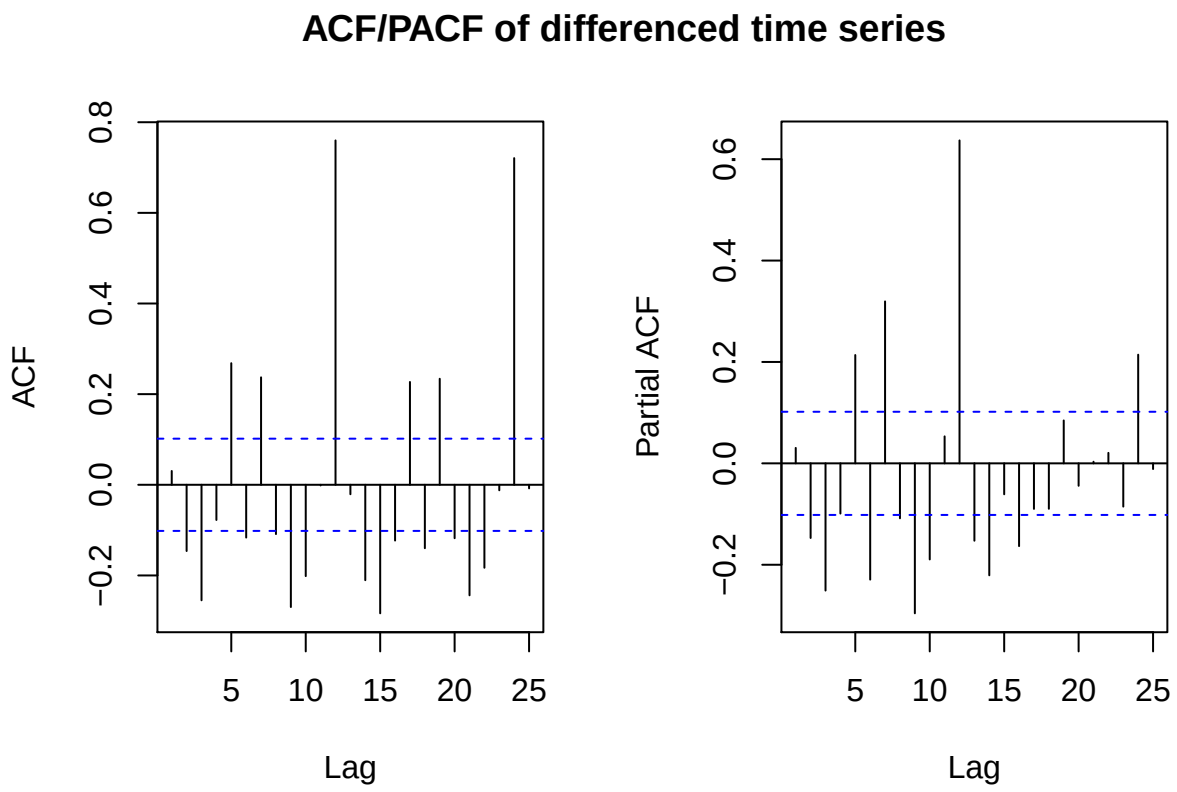

Again, `diff_ts` seems to be a good choice to work with.

**Defining tentative models.**

Unfortunately, the EACF can not be applied to series that show a seasonal pattern.

```
# Plotting ACF and PACF.
  # Defining following plots to be next to each other.
  par(mfrow = c(1, 2))
  # Plotting.
  acf(x = ts(diff_ts), main = NA)
```

```r
  pacf(x = ts(diff_ts), main = NA)
  # Defining shared title.
  title("ACF/PACF of differenced time series", line = -2, outer = T)
```




The decaying significant spikes at lag 12 and 24 in the ACF suggests a seasonal AR(1) component. It seems to be a yearly seasonality ($s = 12$). Since the ACF and PACF do not return more interpretable results related to the non-seasonal part of the model, a bunch of different model variations will be fitted and compared. The models with the lowest AIC, BIC will be analyzed more precisely regarding their residuals.

**Fitting models.**

As explained, different models of the form

$$(p, 1, q) \times (1, 1, 0)_{12}$$

will be tested. The resulting BIC and AIC scores will be compared.

```r
# Comparing different models regarding their AIC/BIC.
model_test = data.frame(p = numeric(), q = numeric(), AIC = numeric(), BIC = numeric())

for (p in (c(0,1,2))) {
  for (q in c(0,1,2)) {
    # Fitting model.
    model = Arima(y = ts,
                  order = c(p,1,q),
                  seasonal = list(order = c(1,1,0), period = 12),
                  method = "ML")
```

```
    model_test = rbind(model_test,
                       cbind(p = p,
                             q = q,
                             aic = model$aic,
                             bic = model$bic))
  }
}

knitr::kable(model_test, caption = "Comparison of different models regarding their AIC/BIC")
```

Table 9: Comparison of different models regarding their AIC/BIC

| p | q | aic | bic |
|---|---|---|---|
| 0 | 0 | 3320.679 | 3328.446 |
| 0 | 1 | 3314.444 | 3326.094 |
| 0 | 2 | 3286.786 | 3302.319 |
| 1 | 0 | 3309.448 | 3321.098 |
| 1 | 1 | 3289.832 | 3305.365 |
| 1 | 2 | 3282.936 | 3302.353 |
| 2 | 0 | 3278.749 | 3294.282 |
| 2 | 1 | 3280.373 | 3299.790 |
| 2 | 2 | 3281.390 | 3304.690 |

Regarding the AIC and BIC, the models $(2,1,0) \times (1,1,0)_{12}$ and $(2,1,1) \times (1,1,0)_{12}$ seem to be the most promising. That is why we will focus on these two models in the follow.

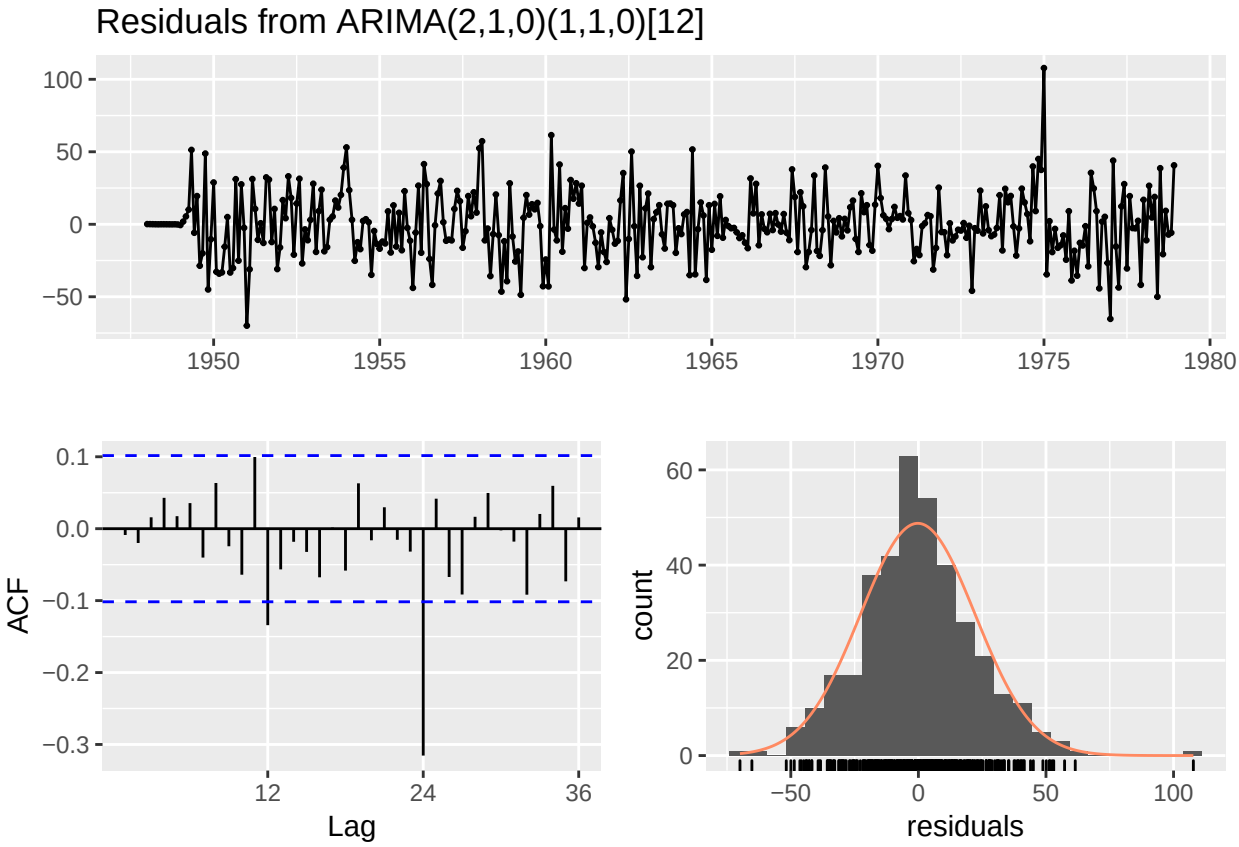
We will name the following fitted models after their non-seasonal part (e.g. `arima_2_1_0` for the $(2,1,0) \times (1,1,0)_{12}$).

```
# Fitting models.
arima_2_1_0 = Arima(y = ts,
                    order = c(2,1,0),
                    seasonal = list(order = c(1,1,0), period = 12),
                    method = "ML")

arima_2_1_1 = Arima(y = ts,
                    order = c(2,1,1),
                    seasonal = list(order = c(1,1,0), period = 12),
                    method = "ML")
```

**Selecting model.**

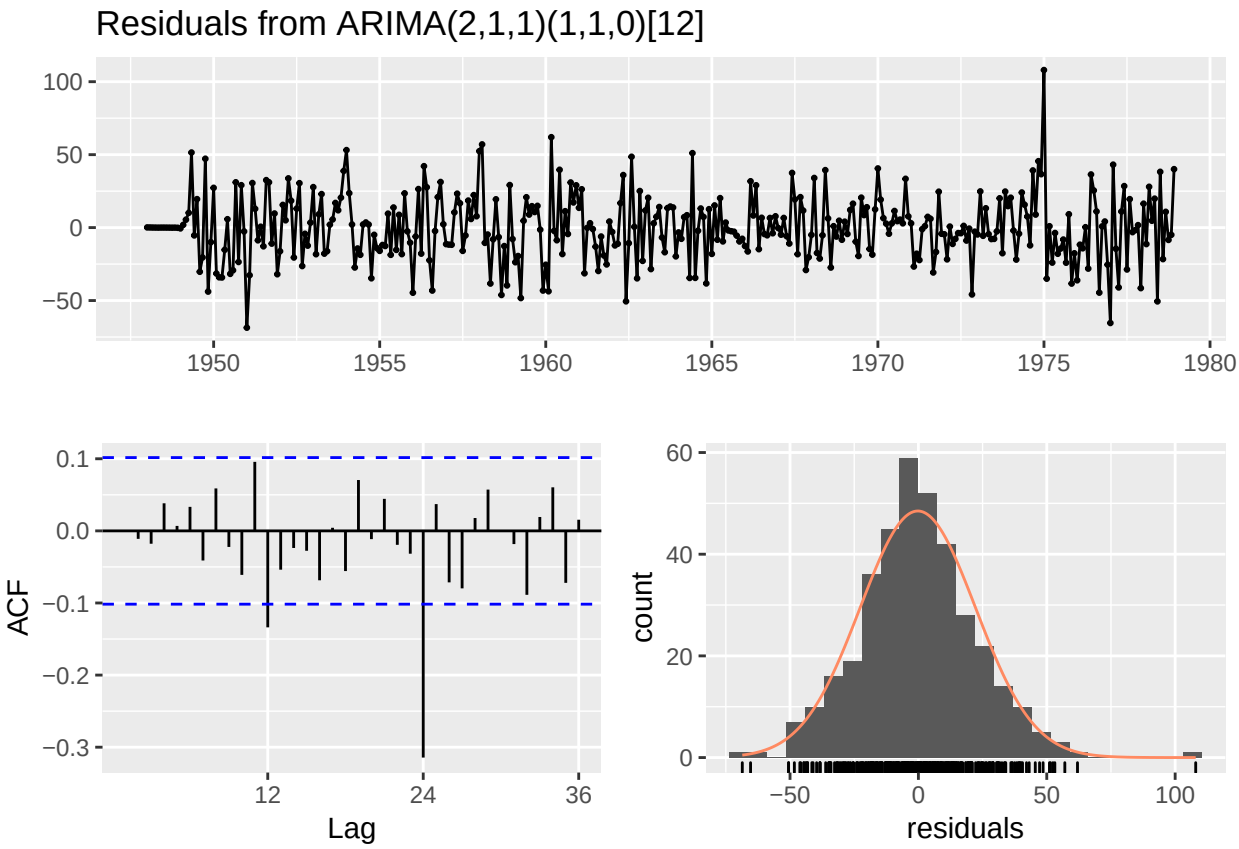
We will perform the residual analysis for both models. The AIC and BIC is already known (in both cases, the $(2,1,0) \times (1,1,0)_{12}$ is characterized by the smaller and therefore better values.)

```
checkresiduals(arima_2_1_0)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(2,1,0)(1,1,0)[12]
Q* = 63.495, df = 21, p-value = 3.725e-06

Model df: 3.   Total lags used: 24
```

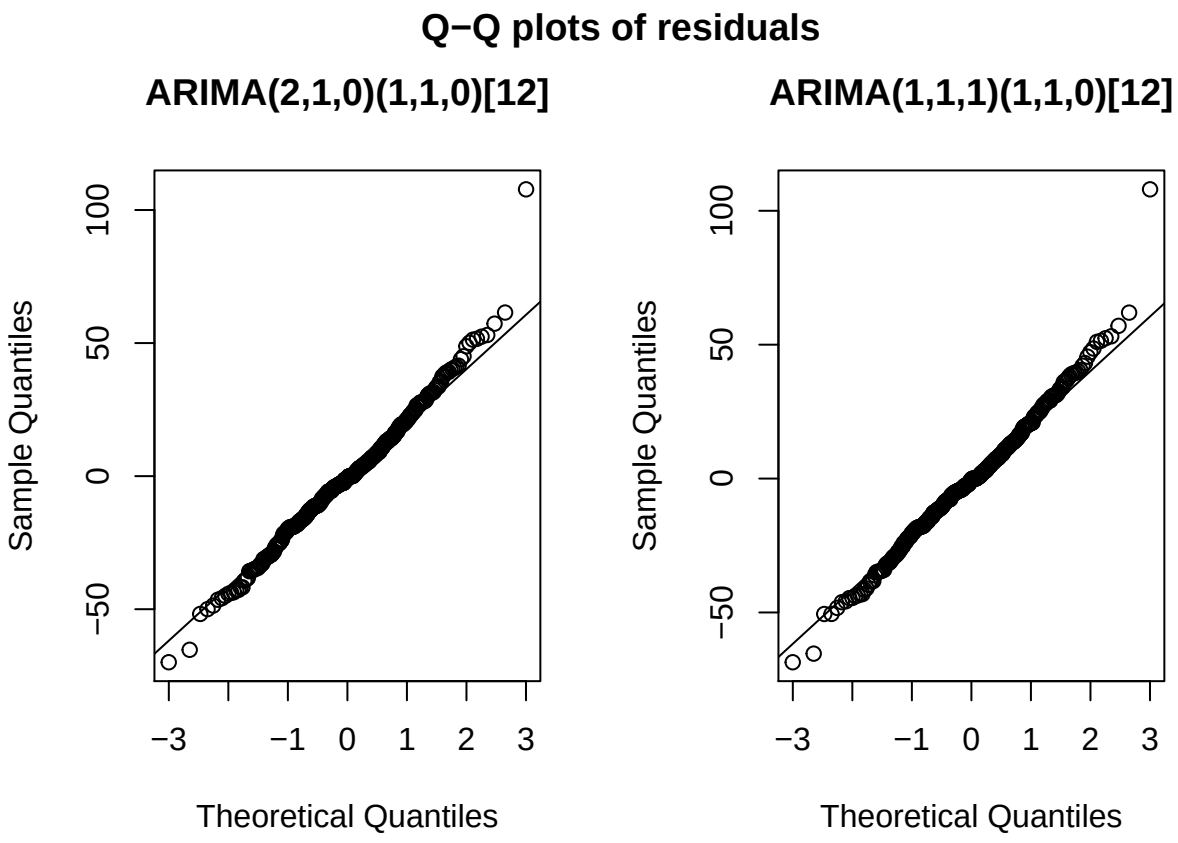
```
checkresiduals(arima_2_1_1)
```




```
    Ljung-Box test

data:  Residuals from ARIMA(2,1,1)(1,1,0)[12]
Q* = 62.752, df = 20, p-value = 2.649e-06

Model df: 4.   Total lags used: 24
```

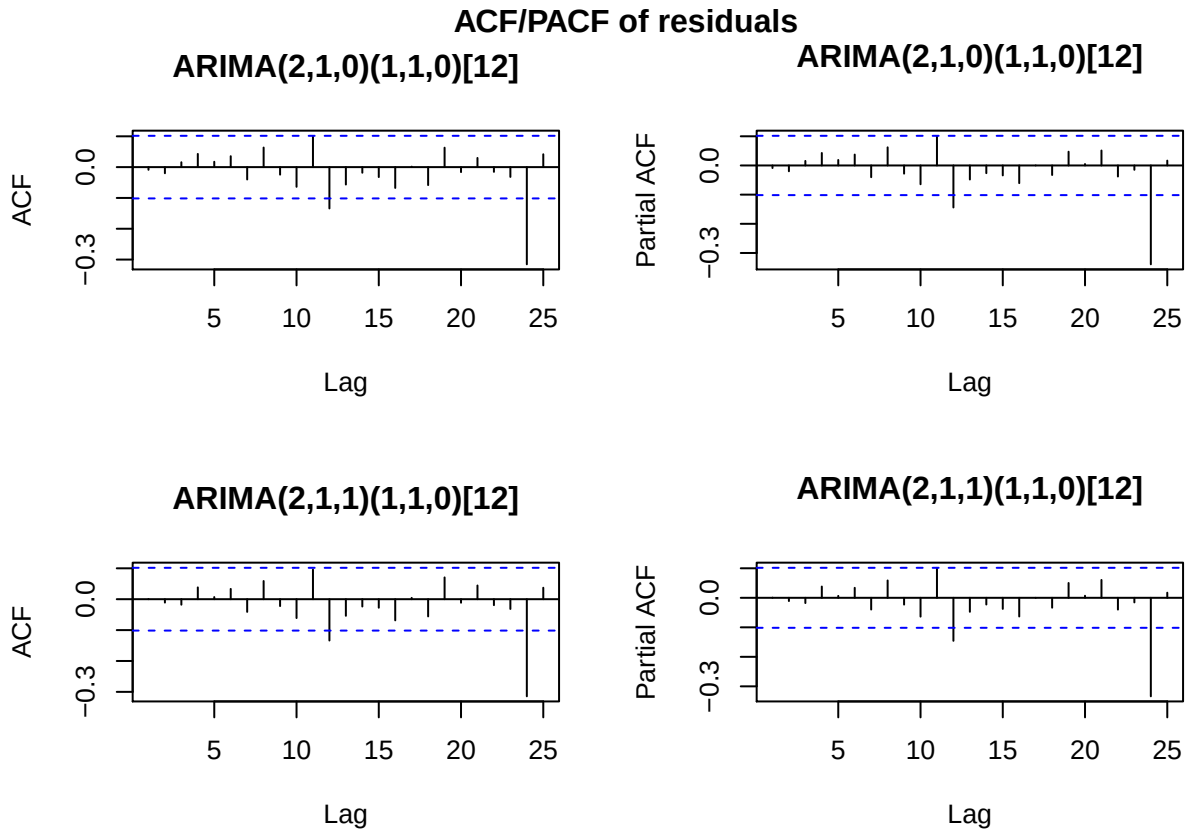
Since the residuals over time look random (white noise) for both models, it is an indication that both fitted models seem to be a promising choice. Looking at the ACF, only the seasonal lag (24) seems to have a significant autocorrelation value. Furthermore, both histograms are approximately normal. However, the plots do not return a better idea about which model seems to be the better choice.

```
# Plotting Q-Q plots of residuals.
# Defining following plots to be next to each other.
par(mfrow = c(1, 2))
# Plotting.
qqnorm(arima_2_1_0$residuals, main = "ARIMA(2,1,0)(1,1,0)[12]")
qqline(arima_2_1_0$residuals)
qqnorm(arima_2_1_1$residuals, main = "ARIMA(1,1,1)(1,1,0)[12]")
qqline(arima_2_1_1$residuals)
# Defining shared title.
title("Q-Q plots of residuals", line = -0.69, outer = T)
```




The Q-Q plots confirm that in both cases the residuals seem to follow a normal distribution.

```r
# Plotting ACF/PACF of residuals.
# Defining following plots to be next to each other.
par(mfrow = c(2, 2))
# Plotting.
acf(ts(arima_2_1_0$residuals), main = "ARIMA(2,1,0)(1,1,0)[12]")
pacf(ts(arima_2_1_0$residuals), main = "ARIMA(2,1,0)(1,1,0)[12]")
acf(ts(arima_2_1_1$residuals), main = "ARIMA(2,1,1)(1,1,0)[12]")
pacf(ts(arima_2_1_1$residuals), main = "ARIMA(2,1,1)(1,1,0)[12]")
# Defining shared title.
title("ACF/PACF of residuals", line = -1, outer = T)
```




For all shown plots (ACF and PACF for both models), all values, except for the seasonal lags, lie within the blue ranges and can be therefore seen as non-significantly correlated.

To test this independence between the lags of the residuals quantitatively, we will perform the statistical Runs test. Since the alternative hypothesis implies that the values are not i.i.d, the residuals can be assumed to be independent for a resulting small p-value ($< 0.05$ if we assume $\alpha = 0.05$).

```
# Computing runs test for randomness of residuals.
#library(TSA)
knitr::kable(data.frame(model = c("ARIMA(2,1,0)(1,1,0)[12]", "ARIMA(2,1,1)(1,1,0)[12]"),
                        p_value = c(runs(arima_2_1_0$residuals)$pvalue,
                                    runs(arima_2_1_1$residuals)$pvalue)),
             caption = "Results of Runs test")
```

Table 10: Results of Runs test

| model | p_value |
|---|---|
| ARIMA(2,1,0)(1,1,0)[12] | 0.345 |
| ARIMA(2,1,1)(1,1,0)[12] | 0.457 |

Based on the seasonal lags, the p-values are quite high in both cases. However, the $ARIMA(2,1,0)(1,1,0)_{12}$ performs better.

Box-Ljung test is a type of statistical test of whether any of a group of autocorrelations of a time series are different from zero. Instead of testing randomness at each distinct lag, it tests the "overall" randomness based on a number of lags. The alternative hypothesis is the same as for the runs test (values are not i.i.d), so again we assume independece for a resulting small p-value ($< 0.05$ if we assume $\alpha = 0.05$).

```
# Performing Ljung-Box test.
knitr::kable(data.frame(model = c("ARIMA(2,1,0)(1,1,0)[12]",
                                  "ARIMA(2,1,1)(1,1,0)[12]"),
                        p_value = c(Box.test(x = arima_2_1_0$residuals,
                                             type = "Ljung-Box")$p.value,
                                    Box.test(x = arima_2_1_1$residuals,
                                             type = "Ljung-Box")$p.value)),
             caption = "Results of Ljung-Box test")
```

Table 11: Results of Ljung-Box test

| model | p_value |
|---|---|
| ARIMA(2,1,0)(1,1,0)[12] | 0.8655168 |
| ARIMA(2,1,1)(1,1,0)[12] | 0.9919148 |

Again, based on the seasonal lags, dependence between the lags can not be rejected. However, at least the result confirms the results of the runs test that the residuals of the$ARIMA(2,1,0)(1,1,0)_{12}$ are assumed to be more independent.

Combining these information about the residuals with the AIC/BIC-information, we come to the conclusion that the $ARIMA(2,1,0)(1,1,0)_{12}$ might be the better choice and therefore select it.

The coefficients of the model are as follows:

```
arima_2_1_0$coef
```

```
       ar1        ar2       sar1
 0.1341472  0.2954506 -0.5035342
```

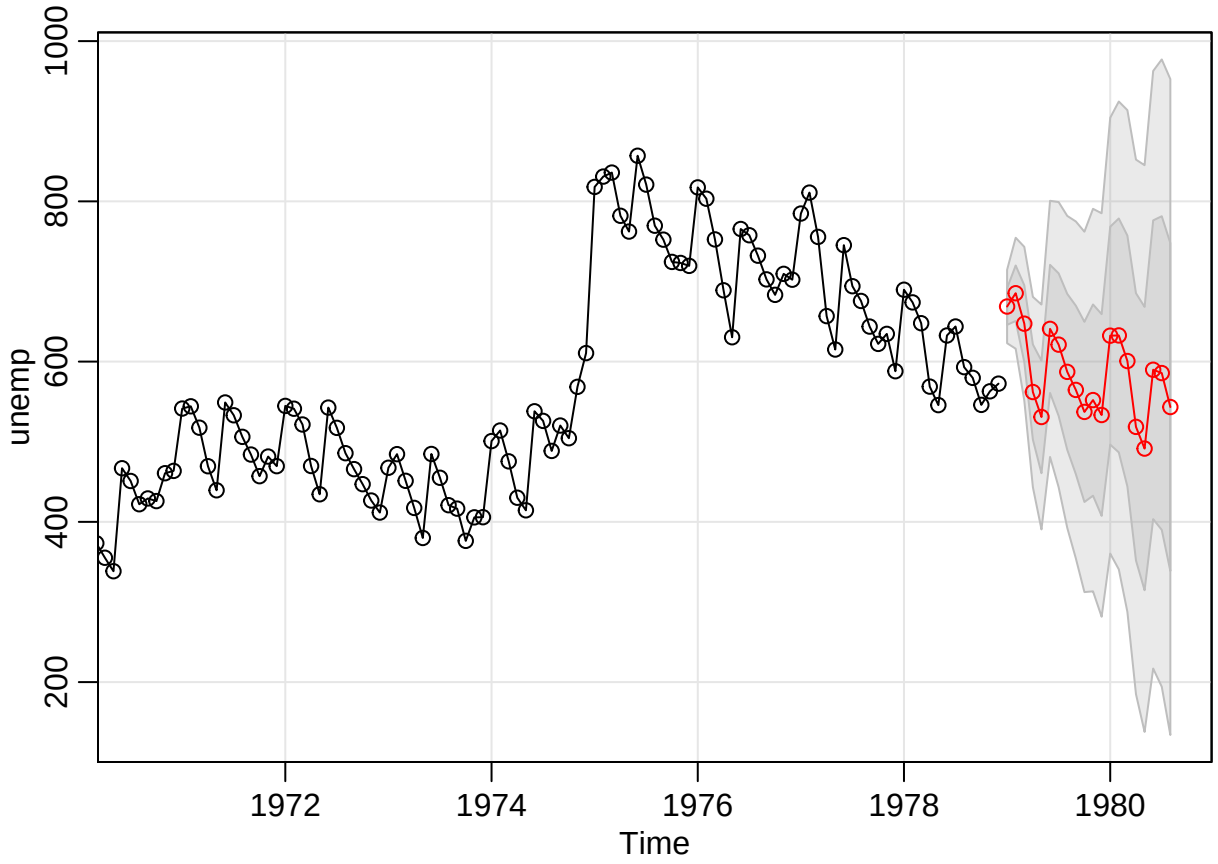
Therefore, we can write down the equation of the model like this:

$$(1-B)x_t = (1+0.1341472B+0.2954506B^2)(1-0.5035342B^{12})w_t$$

We will select this model and use it for the following forecast.

**Performing forecast using selected model.**

```
sarima.for(unemp,
           n.ahead = 20,
           p = 2,
           d = 1,
           q = 0,
           P = 1,
           D = 1,
           Q = 0,
           S = 12)
```




```
$pred
          Jan      Feb      Mar      Apr      May      Jun      Jul
1979 668.7292 685.3414 647.3309 562.0219 530.8644 640.8394 621.1759
1980 632.3583 632.6721 600.7224 518.5482 491.4530 589.8120 585.6913
          Aug      Sep      Oct      Nov      Dec
1979 587.1906 564.6499 537.2060 551.9788 533.3402
1980 543.2907

$se
           Jan       Feb       Mar       Apr       May       Jun       Jul
1979  22.90220  34.62949  47.94117  59.37226  70.16803  79.92460  88.96953
1980 135.99694 145.96330 156.54902 166.78393 176.81535 186.44517 195.72522
           Aug       Sep       Oct       Nov       Dec
1979  97.32440 105.12626 112.44006 119.34172 125.88386
1980 204.63823
```

# Time Series Analysis - lab03

*Lennart Schilling (lensc874), Sridhar Adhikarla (sriad858)*

*2019-10-12*

## Contents

```
library(astsa)
library(forecast)
library(ggplot2)
```

## Assignment 1

**1 Write down the expression for the state space model that is being simulated.**

The state space model being simulated is -

$$
\begin{aligned}
z_t &= (A_{t-1} * z_{t-1}) + e_t \quad - Transition\ equation \\
x_t &= (C_t * z_t) + v_t \quad - Observation\ equation \\
v_t &= Normal(0, R_t) \\
e_t &= Normal(0, Q_t)
\end{aligned}
$$

The parameter values for this state space model are -

$$
\begin{aligned}
A &= 1 \\
C &= 1 \\
R &= 1 \\
Q &= 1
\end{aligned}
$$

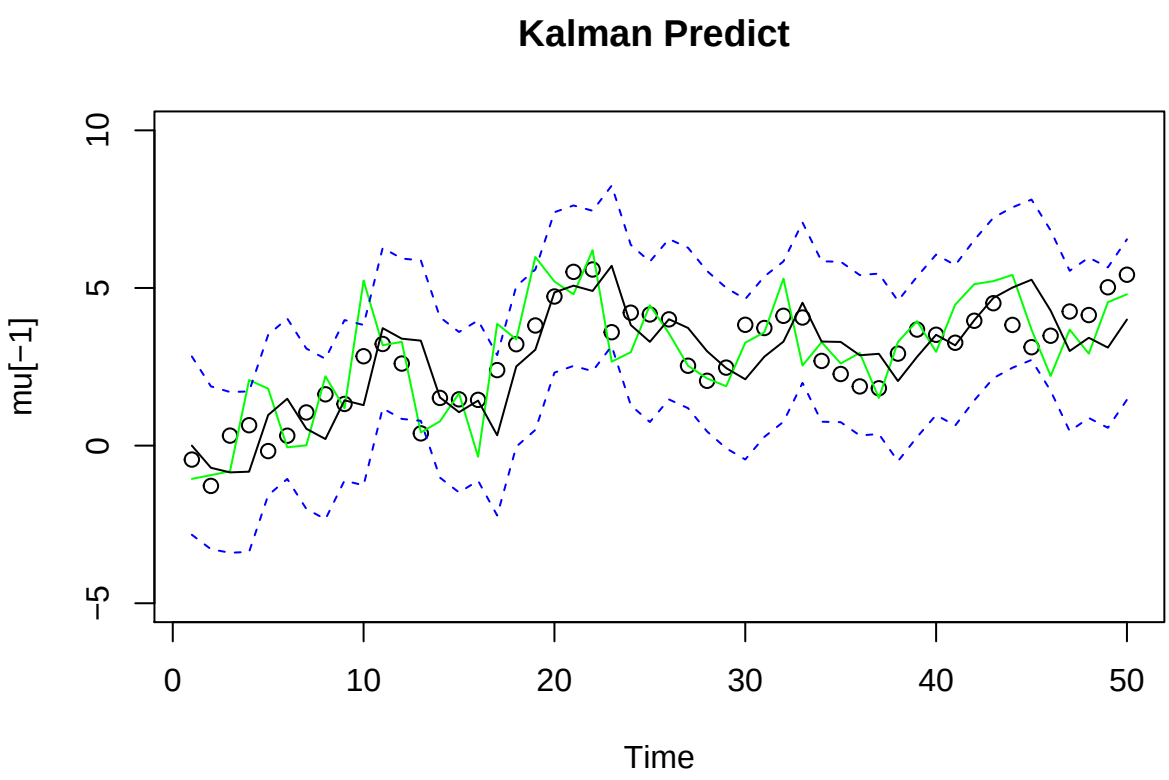
The state space model with these parameter values is -

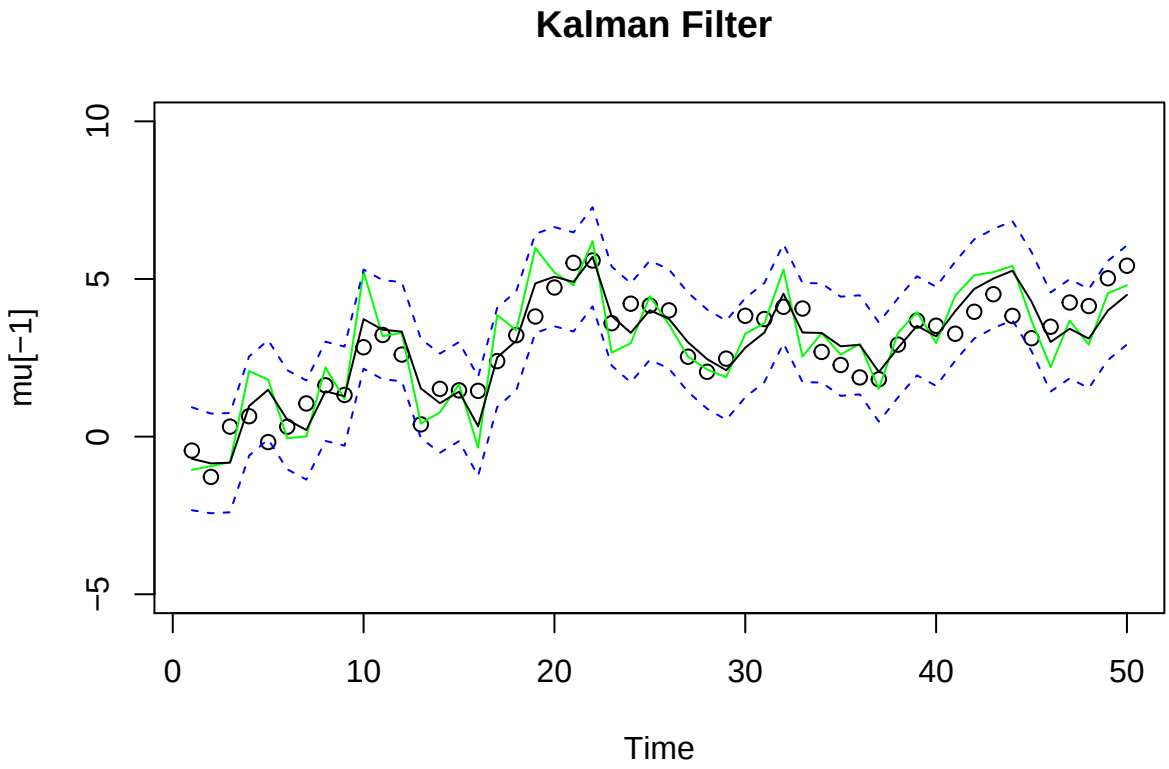
$$
\begin{aligned}
z_t &= z_{t-1} + e_t \quad - Transition\ equation \\
x_t &= z_t + v_t \quad - Observation\ equation \\
v_t &= Normal(0, 1) \\
e_t &= Normal(0, 1)
\end{aligned}
$$

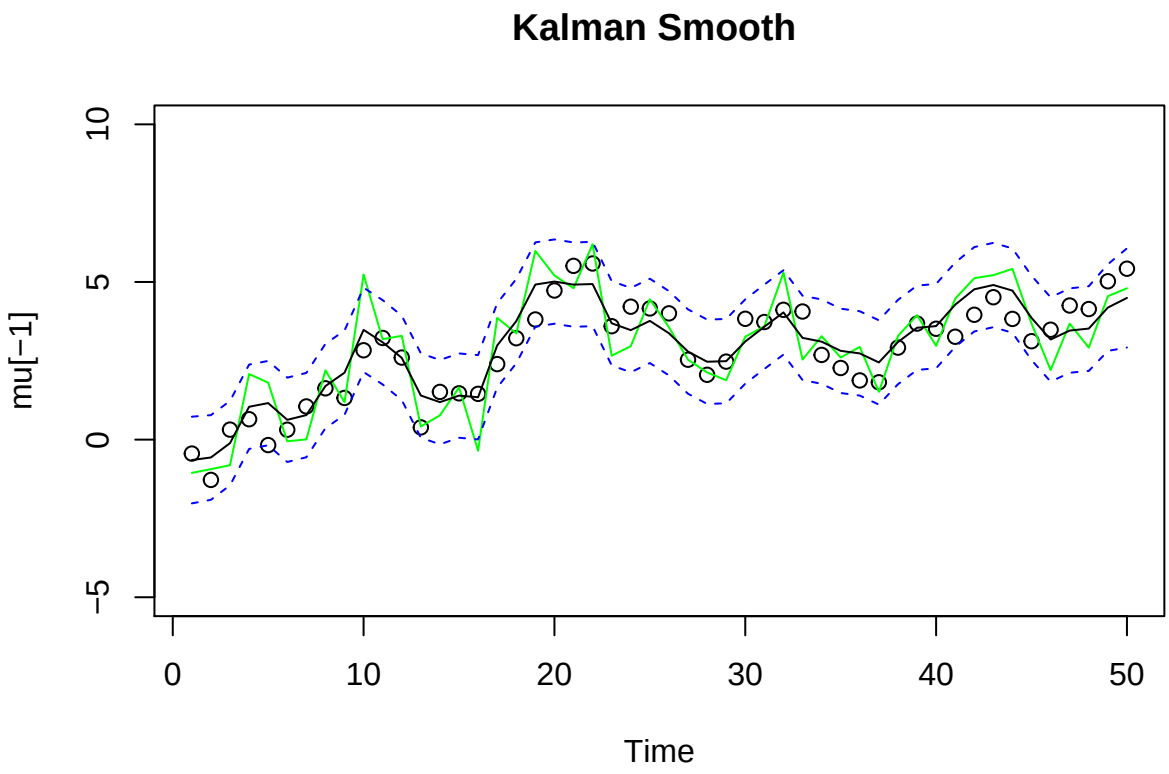
**2 Run this script and compare the filtering results with a moving average smoother of order 5.**

Results from running the kalman filter

```
#utiliy functions

#used to generate data simulated from the SSM
gen_Data = function(num=50, Q=1, R=1){
  set.seed(1)
  w = rnorm(num+1,0,R)
  v = rnorm(num ,0,Q)
  mu = cumsum(w) # state : mu[0], mu[1] ,... , mu[50]
  y = mu[-1] + v # obs: y[1] ,... , y[50]
  return(list(Y = y, MU = mu))
}

#fits a kalman filter on
fit_Kalman = function(y, Q=1, R=1){
  # filter and smooth ( Ksmooth 0 does both )
  ks = Ksmooth0(length(y) , y, A=1, mu0=0, Sigma0=1, Phi=1, cQ=Q, cR=R)
  return(ks)
}

gen_Plot = function(ks, mu, y){
  num = length(y)
  Time = 1:num
  plot (Time , mu[-1], main ='Kalman Predict ', ylim =c(-5,10))
  lines (Time ,y,col=" green ")
  lines (ks$xp)
  lines (ks$xp+2* sqrt (ks$Pp), lty =2, col=4)
  lines (ks$xp -2* sqrt (ks$Pp), lty =2, col=4)
  plot (Time , mu[-1], main ='Kalman Filter ', ylim =c(-5,10))
  lines (Time ,y,col=" green ")
  lines (ks$xf)
  lines (ks$xf+2* sqrt (ks$Pf), lty =2, col=4)
  lines (ks$xf -2* sqrt (ks$Pf), lty =2, col=4)
  plot (Time , mu[-1], main ='Kalman Smooth ', ylim =c(-5,10))
  lines (Time ,y,col=" green ")
  lines (ks$xs)
  lines (ks$xs+2* sqrt (ks$Ps), lty =2, col=4)
  lines (ks$xs -2* sqrt (ks$Ps), lty =2, col=4)
}

dat = gen_Data()
ks = fit_Kalman(dat$Y)
gen_Plot(ks, dat$MU, dat$Y)
```

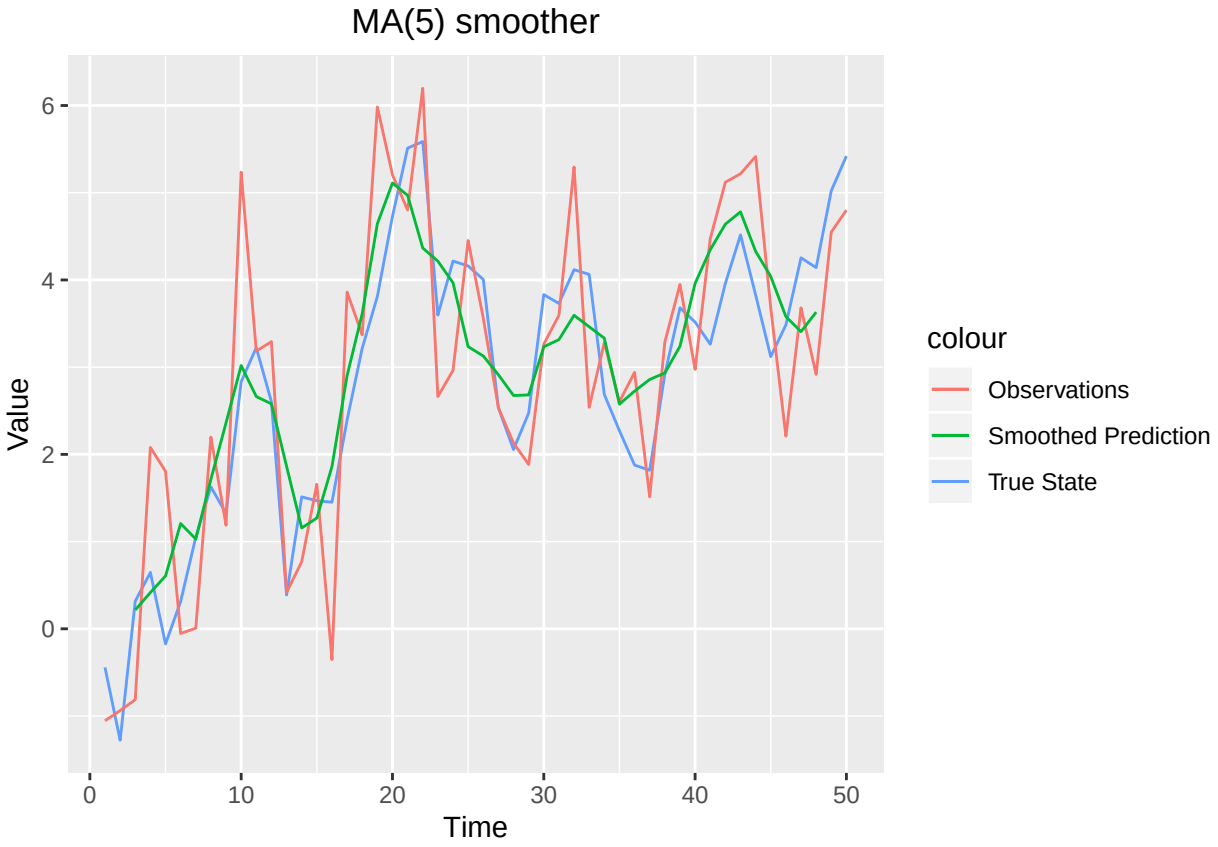

Kalman Predict
mu[−1]
10
5
0
−5
0
10
20
30
40
50
Time


Kalman Filter
mu[−1]
10
5
0
−5
0
10
20
30
40
50
Time




Results on running a moving average smoother on the data

```
ma5_smooth = ma(dat$Y, order = 5)
num = length(dat$Y)

ggplot() +
  geom_line(aes(x=1:num, y=dat$MU[-1], col="True State")) +
  geom_line(aes(x=1:num, y=dat$Y, col="Observations")) +
  geom_line(aes(x=1:num, y=ma5_smooth, col="Smoothed Prediction")) +
  ggtitle("MA(5) smoother") + xlab("Time") + ylab("Value") +
  theme(plot.title = element_text(hjust = 0.5))
```




As we can see from the plots the Kalman Filter fit to the data was better than the one with MA(5) smoother. If followed the true data closely where as the MA(5) model, the predictions are not as close.

```
t1 = 4:49
print("MA(5) smoother SSE")
```

```
## [1] "MA(5) smoother SSE"
```

```
sum((dat$MU[t1] - ma5_smooth[t1-1])^2)
```

```
## [1] 18.75434
```

```
print("Kalman smoother SSE")
```

```
## [1] "Kalman smoother SSE"
```

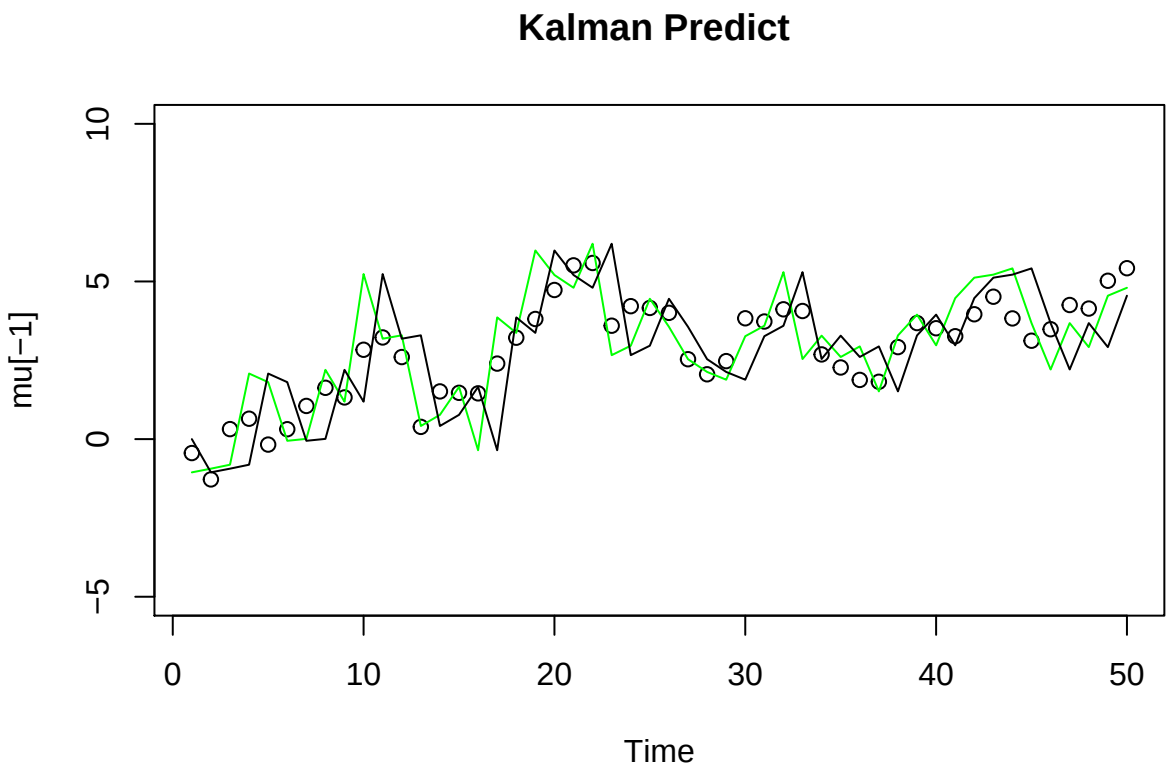
```
sum((dat$MU[t1] - ks$xs[t1-1])^2)
```

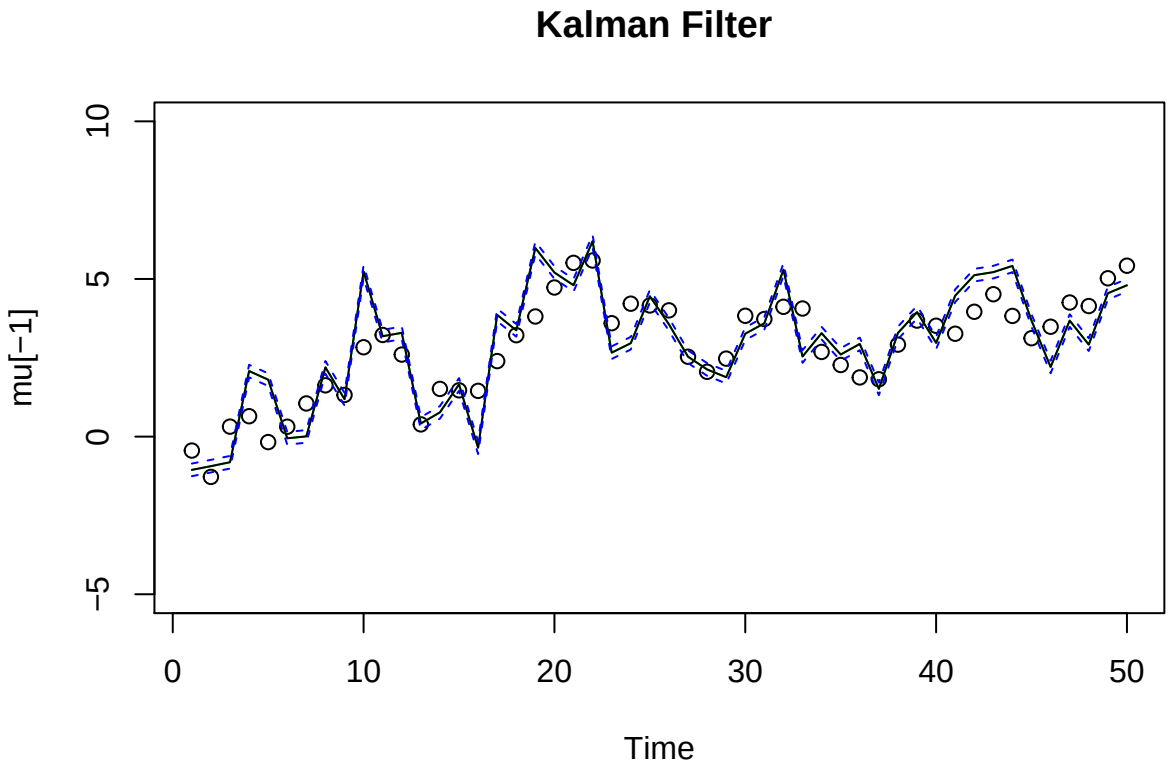
```
## [1] 15.81689
```

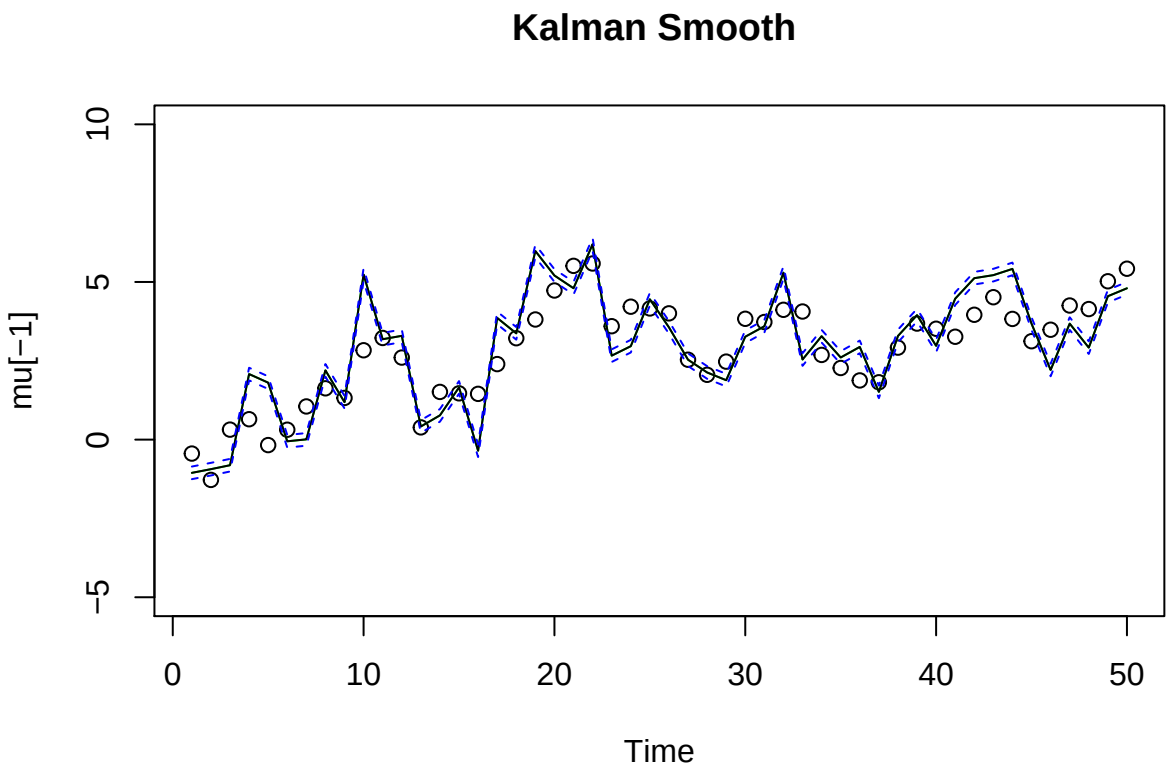
The SSE values for the two smoothers show clearly that the kalman smoother is doing a better job, though the difference is not much.

**3 Also, compare the filtering outcome when R in the filter is 10 times smaller than its actual value and while Q in the filter is 10 times larger than its actual value. How does the filtering outcome varies?**

```
ks3 = fit_Kalman(dat$Y, Q = 10, R = 1/10)
gen_Plot(ks3, dat$MU, dat$Y)
```



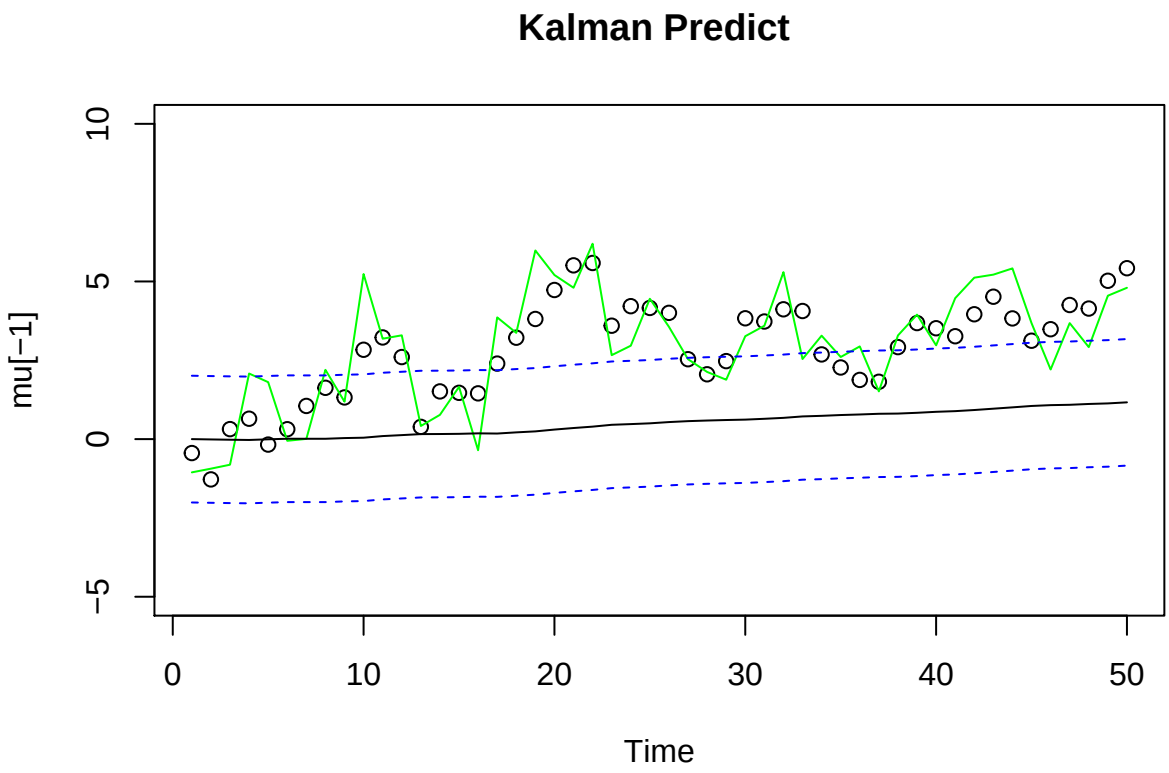

Kalman Filter
mu[−1]
10
5
0
−5
0
10
20
30
40
50
Time

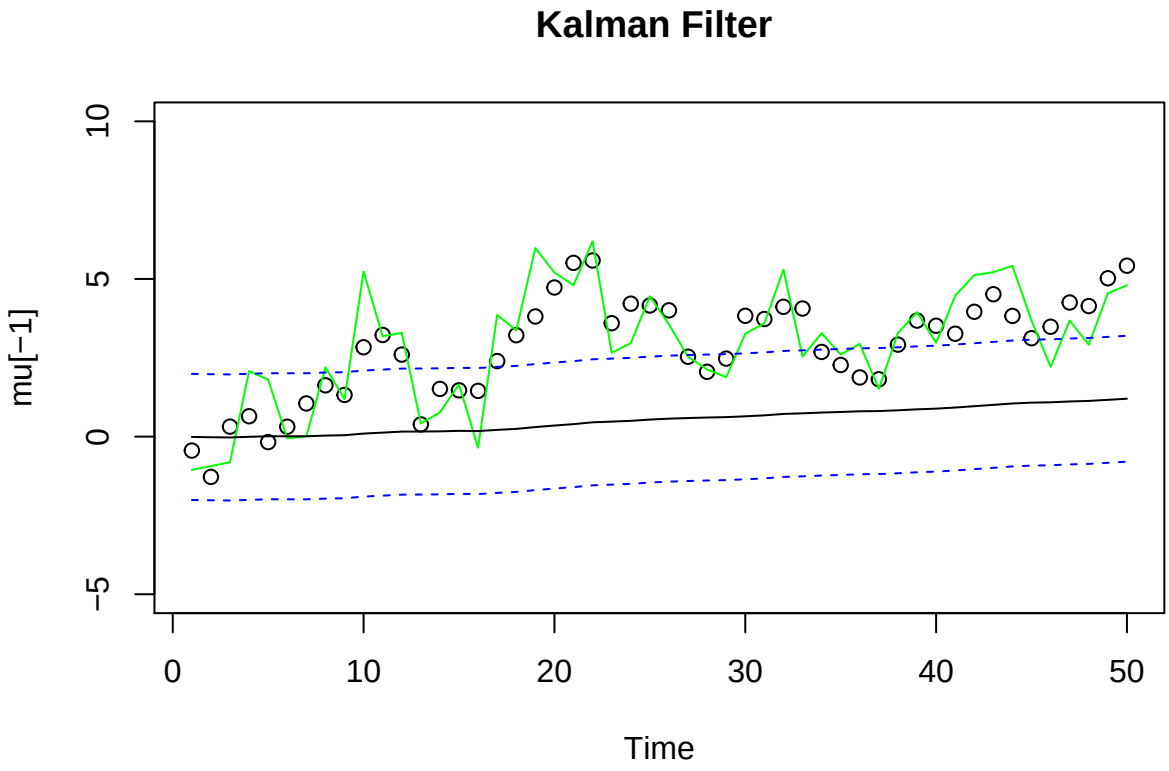


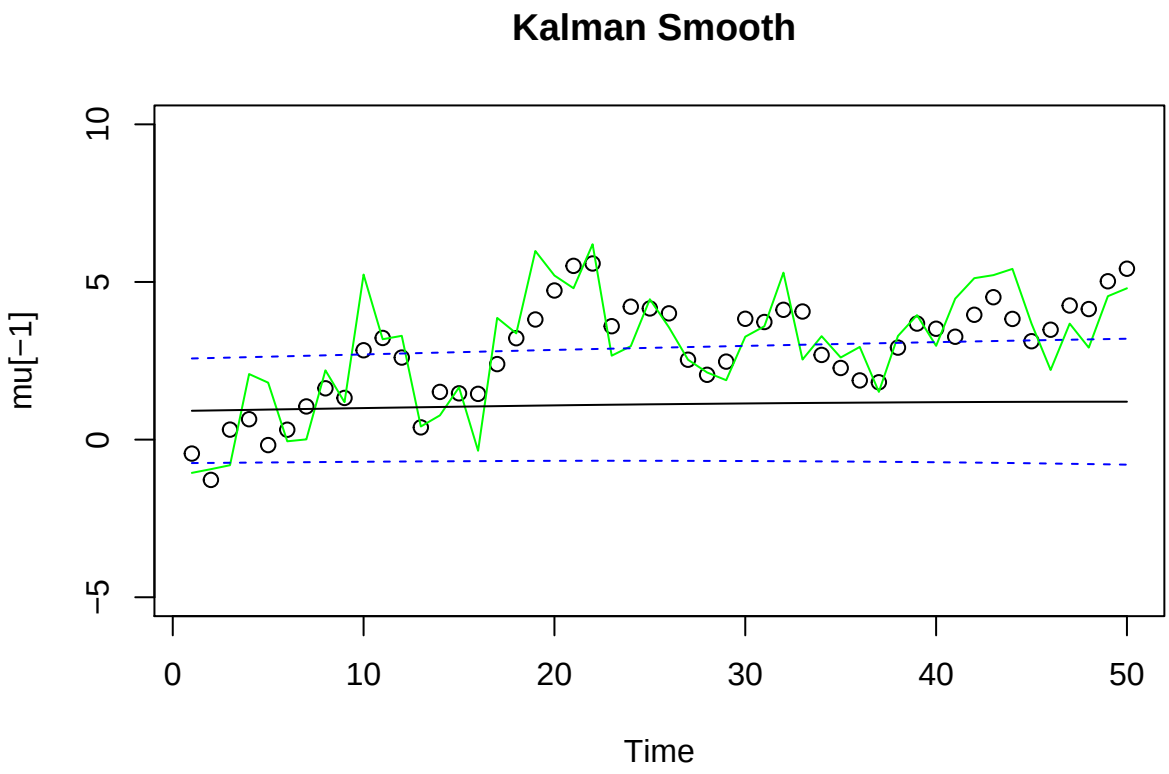

The parameter R is for the varience of the error in measurements. Making it 10 times smaller means that the measurements are very accurate and the estimates should be taken care of. This is why the confidence bands for the prediction are so close to the prediction. For the kalman smooth and filter fit the line overlaps the observation completely. Since it assumes that the measurements are accurate it just uses that to make the get the next point. Also having a large value for Q means that the prediction relies more on the observations than the previous estimate. So, it follows the observations closely.

**4 Now compare the filtering outcome when R in the filter is 10 times larger than its actual value while Q in the filter is 10 times smaller than its actual value. How does the filtering outcome varies?**

```
ks4 = fit_Kalman(dat$Y, Q = 1/10, R = 10)
gen_Plot(ks4, dat$MU, dat$Y)
```



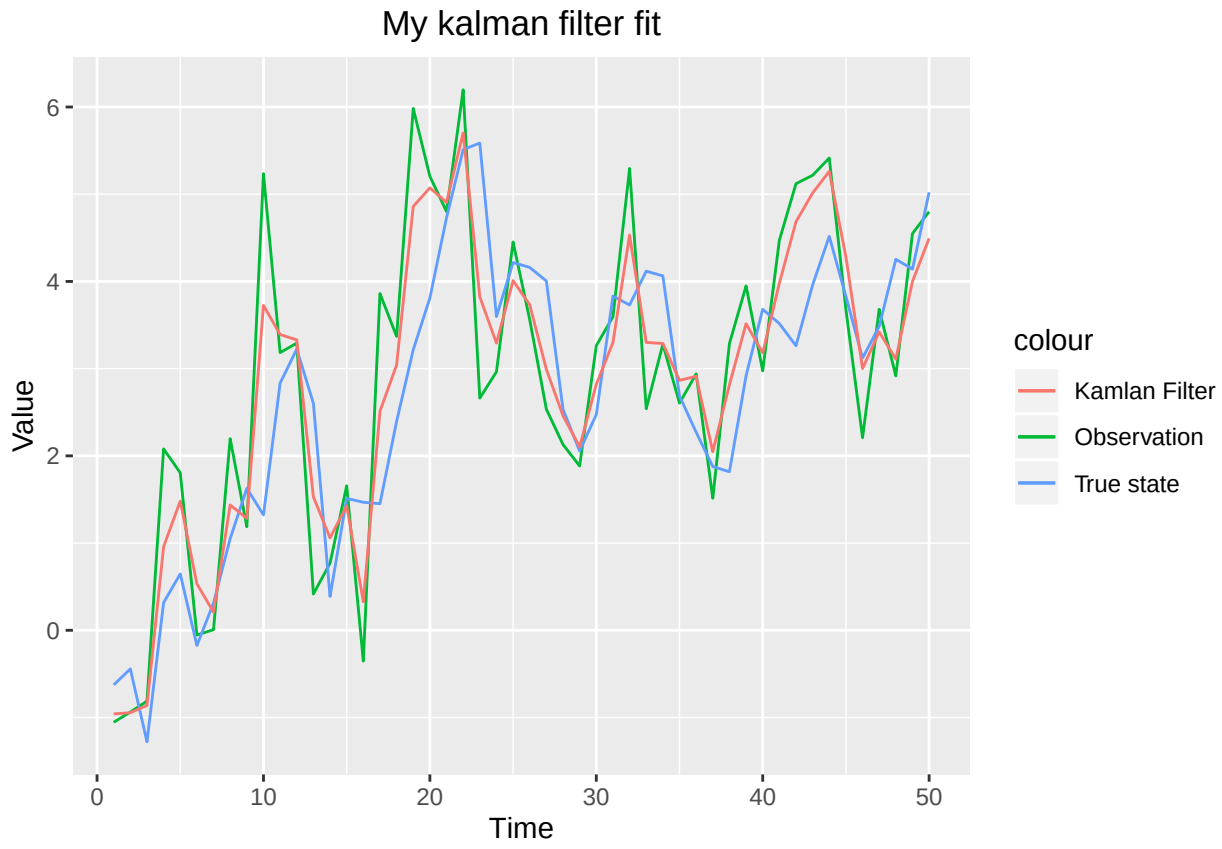

Kalman Filter
mu[−1]
10
5
0
−5
0
10
20
30
40
50
Time



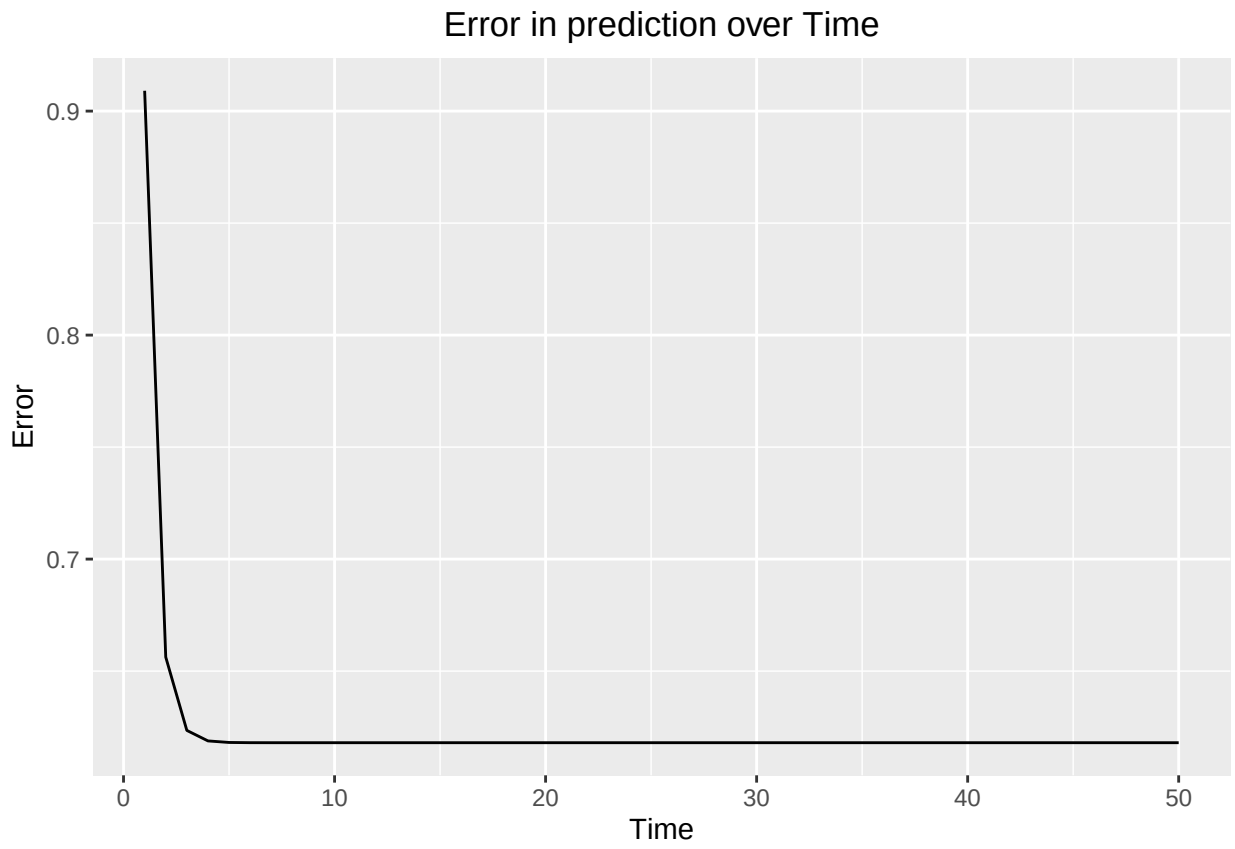

Having a large value for R means that the observations are not reliable and it just goes along with the prediction it had in the previous step. This is the reason we get almost straight line as a fit to the data. It does not deviate much from its initial prediction and since it assumes that the predictions are accurate the confidence interval bands are much further away from the prediction. Having a small value for Q supports this behaivous as it does not rely much on the observations not and instead relies on the previous estimate.

**5 Implement your own Kalman filter and replace ksmooth0 function with your script.**

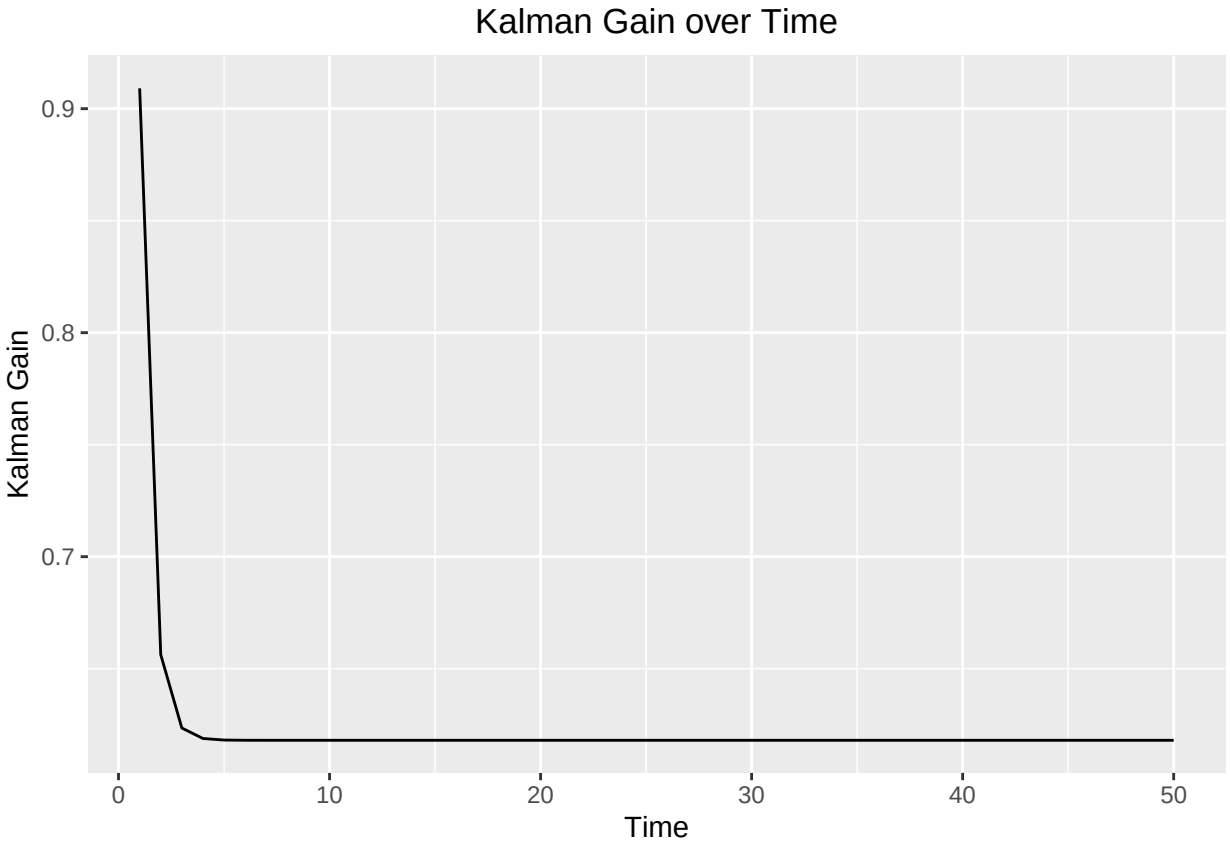
```
my_kalman = function( y, mea_0, err_p0, A=1, C=1, Q=1, R=1){
  n = length(y); kal_gain = c(); mea_t = mea_0; err_pt = err_p0

  for (t in 1:n) {
    kal_gain[t] = err_pt[t]/(err_pt[t] + R)
    mea_t[t] = mea_t[t] + kal_gain[t] * (y[t] - mea_t[t]*C)
    err_pt[t] = (1 - kal_gain[t]*C) * err_pt[t]
    mea_t[t+1] = A*mea_t[t]
    err_pt[t+1] = A*err_pt[t] + Q
  }
  return(list(klf = mea_t[1:n], err_pred = err_pt[1:n], kg = kal_gain[1:n]))
}


my_kal = my_kalman(dat$Y, 0, 10)

ggplot()+
  geom_line(aes(x=1:50, y=dat$Y, col="Observation")) +
  geom_line(aes(x=1:50, y=dat$MU[1:50], col="True state")) +
  geom_line(aes(x=1:50, y=my_kal$klf, col="Kamlan Filter")) +
  ggtitle("My kalman filter fit") + xlab("Time") + ylab("Value") +
  theme(plot.title = element_text(hjust = 0.5))
```



```
ggplot() +
  geom_line(aes(x=1:50, my_kal$err_pred)) +
  ggtitle("Error in prediction over Time") + xlab("Time") + ylab("Error") +
  theme(plot.title = element_text(hjust = 0.5))
```




```
ggplot() +
  geom_line(aes(x=1:50, my_kal$kg)) +
  ggtitle("Kalman Gain over Time") + xlab("Time") + ylab("Kalman Gain") +
  theme(plot.title = element_text(hjust = 0.5))
```




As we can see from the plot the kalman gain starts off with a large value close to 1 and it gets rid of the error in prediction in very few steps as the kalman gain gets to a stable value of approximately 0.5 in about 5 steps. This shows the fast convergence of the kalman filter.

## 6 How do you interpret the Kalman gain?

Kalman gain can take up a value between 0 and 1. It controls how much should our next prediction be affected by the observation. A large value of kalman gain implies that the measurements are accurate and the estimates are unstable. This leads to a large deviation in the prediction from the previous step. A small value of kalman gain implies that the measurements are inacurate and the estimates are stable with small errors. So it uses up most of the previous estimate to make the next prediction without much contribution from the observation. The Kalman gain is the relative weight given to the measurements and current state estimate, and can be "tuned" to achieve a particular performance. With a high gain, the filter places more weight on the most recent measurements, and thus follows them more responsively.

Teaching Session

Sridhar Adhikarla

Assignment #2:

Let $\{x_t\}$ be a zero-mean, unit-variance stationary process with autocorrelation function $\rho_h$. Suppose that $\mu_t$ is a non-constant function and that $\sigma_t$ is a positive-valued non-constant function. The observed series is formed as

$$Y_t = \mu_t + \sigma_t \cdot X_t$$

(a) Find the mean and covariance function for the process $\{y_t\}$

Mean:

$$E[Y_t] = E[\mu_t] + E[\sigma_t \cdot X_t]$$

$$= \mu_t + \sigma_t \cdot E[X_t]$$

$$= \mu_t + 0$$

$$= \mu_t$$

Auto covariance:

$$\text{cov}(Y_s, Y_t) = \text{cov}((\mu_s + \sigma_s X_s), (\mu_t + \sigma_t \cdot X_t))$$

$$= E[(\mu_s + \sigma_s X_s - E[\mu_s + \sigma_s X_s]) \cdot (\mu_t + \sigma_t \cdot X_t) - E[\mu_t + \sigma_t \cdot X_t]]$$

We know that $E[\mu_t + \sigma_t \cdot X_t] + \mu_t = E[(\mu_s + \sigma_s \cdot X_s - \mu_s)(\mu_t + \sigma_t X_t) - \mu_t]$

$$= (\sigma_s \cdot X_s)(\sigma_t \cdot X_t)$$

$$= \sigma_s \cdot \sigma_t \cdot E[X_s \cdot X_t]$$

(2)

We know that $E[X_s] = 0$ so we can add that in the above eq.

$$= \sigma_s \cdot \sigma_t \cdot [E[(X_s - E[X_s])(X_t - E[X_t])]$$

$$= \sigma_s \cdot \sigma_t \cdot \text{cov}(X_s, X_t) \quad \text{—— (1)}$$

Now, given auto correlation function $\rho_h$

and assume $s = t-h$

$\therefore$ auto covariance $= \sigma_t \cdot \sigma_{t-h} \cdot \text{cov}(X_t, X_{t-h})$ —— (2)

We know: $$\rho(t, t-h) = \frac{r(t, t-h)}{\sqrt{(r(t,t) \cdot r(t-h, t-h))}}$$

Given:

$$r(t,t) = \text{var}(X_t) = 1$$

$$\therefore \quad \rho(t, t-h) = r(t, t-h)$$

using this in equation (2) we get

auto covariance $= \sigma_t \cdot \sigma_{t-h} \cdot \rho_h$

(3)

(b) Show that the auto correlation function for the $\{y_t\}$ process depends only on the time lag. Is the $\{y_t\}$ process stationary?

Autocorrelation function

$$\rho(s,t) = \frac{r(s,t)}{\sqrt{r(s,s) \cdot r(t,t)}}$$

$$\rho(s,t) = \frac{r(Y_s, Y_t)}{\sqrt{\text{var}(Y_s)\,\text{var}(Y_t)}}$$

From solution (a) we know that $r(s,t) = \text{cov}(Y_s, Y_t) = \sigma_s \sigma_t \rho_h$

assume $s = t-h$

$$r(t, t-h) = \text{cov}(Y_t, Y_{t-h}) = \sigma_t \cdot \sigma_{t-h} \cdot \rho_h$$

$$\text{var}(Y_t) = \text{var}(\mu_t + \sigma_t \cdot X_t)$$

$$= \sigma_t^2 \cdot \text{var}(X_t)$$

$$= \sigma_t^2$$

$$\therefore \text{var}(Y_{t-h}) = \sigma_{t-h}^2$$

$$\therefore \rho(t, t-h) = \frac{\sigma_t \cdot \sigma_{t-h} \cdot \rho_h}{\sqrt{\sigma_t^2 \cdot \sigma_{t-h}^2}} = \rho_h$$

with this, we can say that the auto correlation function depends only on the lag.

(4)

(c) Is it possible to have a time series with a constant mean and with $\text{cov}(Y_t, Y_{t+h})$ free of $t$ but with $\{Y_t\}$ not stationary?

Assuming constant mean $E[Y_t] = E[\mu + \sigma_t \cdot X_t] = \mu$

$$\text{cov}(Y_s, Y_t) = \text{cov}((\mu + \sigma_s \cdot X_s), (\mu + \sigma_t \cdot X_t))$$

$$= E[(\mu + \sigma_s \cdot X_s - \mu)(\mu + \sigma_t \cdot X_t - \mu)]$$

Assuming $s = t-h$

$$= E[(\sigma_t X_t)(\sigma_{t-h} \cdot X_{t-h})]$$

$$= \sigma_t \cdot \sigma_{t-h} \, \text{cov}(X_t \cdot X_{t-h})$$

This is same as equation ① from exercise (a)

hence from the above proof.

$$\gamma(h) = \sigma_t \cdot \sigma_{t-h} \cdot \rho_h \quad \text{——} \quad (3)$$

Even though the mean is constant, and auto correlation depends only on lag, the auto covariance depends on time 't' and hence the process is not stationary.

(5)

Assignment 18

For each of the following ARMA models, find the roots of the AR and MA polynomials, identify the values of p and q for which they are ARMA(p,q) (be careful of parameter redundancy), determine wether they are causal and determine whether they are invertible. In each case, $w_t \sim wn(0,1)$

(a) $x_t - 3x_{t-1} = w_t + 2w_{t-1} - 8w_{t-2}$

$\Rightarrow x_t - 3Bx_t = w_t + 2Bw_t - 8B^2 w_t$

$\Rightarrow x_t(1-3B) = w_t(1+2B-8B^2)$

$\Rightarrow \phi(B)x_t = \theta(B)w_t$

roots of $\phi(B) \Rightarrow 1-3B=0$

$\Rightarrow B = \frac{1}{3}$

The root is not greater than 1 hence not causal

roots of $\theta(B) \Rightarrow 1+2B-8B^2=0$

using $\Rightarrow \frac{-b \pm \sqrt{b^2-4ac}}{2a}$ to find roots.

where $a=-8, b=2, c=1$

$\Rightarrow \frac{-2 \pm \sqrt{4+32}}{16} \Rightarrow = \frac{(-2 \pm 6)}{16} = 0.25, -0.5$

The roots do not lie outside the unit circle, $\therefore$ not invertible.

(6)

(b) $x_t - 2x_{t-1} + 2x_{t-2} = w_t - \frac{8}{9} \cdot w_{t-1}$

$x_t - 2Bx_t + 2B^2 \cdot x_t = w_t - \frac{8}{9} B \cdot w_t$

$x_t(1 - 2B + 2B^2) = w_t(1 - \frac{8}{9} B)$

$\phi(B) x_t = \theta(B) w_t$

roots of $\phi(B) \Rightarrow 1 - 2B + 2B^2 = 0$

using $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ where $a = 2, b = -2, c = 1$

$= \frac{2 \pm \sqrt{4 - 8}}{4} = \frac{(2 \pm 2i)}{4} = (0.5 + 0.5i), (0.5 - 0.5i)$

The roots are not greater than 1, hence not causal.

roots of $\theta(B) \Rightarrow 1 - \frac{8}{9} B = 0$

$B = \frac{9}{8}$

The root lies outside unit circle, $\therefore$ invertible.

(7)

(c)

$$x_t - 4x_{t-2} = w_t - w_{t-1} + 0.5 w_{t-2}$$

$$x_t - 4B^2 x_t = w_t - B w_t + 0.5 B^2 w_t$$

$$x_t (1 - 4B^2) = w_t (1 - B + 0.5B^2)$$

$$\phi(B) x_t = \theta(B) \cdot w_t$$

roots of $\phi_R(B) \Rightarrow 1 - 4B^2 = 0$

$$B = \pm \frac{1}{2}$$

Roots are not greater than 1, hence not causal

roots of $\theta(B)$ using $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

where $a = 0.5$, $b = -1$, $c = 1$

$$= \frac{1 \pm \sqrt{1-2}}{1} = \frac{1 \pm i}{1} = (1+i), (1-i)$$

The roots lie outside unit circle Hence invertible.

(8)

(d)

$$x_t - \frac{9}{4} x_{t-1} - \frac{9}{4} x_{t-2} = w_t$$

$$x_t - \frac{9}{4} B x_t - \frac{9}{4} B^2 x_t = w_t$$

$$x_t \left(1 - \frac{9}{4} B - \frac{9}{4} B^2\right) = w_t$$

$$\phi(B)\, x_t = \theta(B)\, w_t$$

roots of $\phi(B)$ using $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

where $a = -\frac{9}{4}$, $b = -\frac{9}{4}$, $c = 1$

$$\frac{\frac{9}{4} \pm \sqrt{\left(-\frac{9}{4}\right)^2 + 9}}{2 \times -\frac{9}{4}} = 0.33 \; ; \; -1.33$$

The roots are not greater than 1, hence not causal.

roots of $\theta(B) = 1$

The root lie on unit circle. Hence invertible.

# Teaching session Assignment 2:-

Sridhar Adhikarla.

## Assignment 1:-

Using the example solved during lecture 6, find the state space model representation of a multiplicative seasonal arima model.

$$ARIMA\ (p,d,q) \times (P,D,Q)_s$$

This process can be written using the backshift operator as...

$$\Phi^P(B^s)\,\phi^p(B)(1-B^s)^D(1-B)^d \cdot x_t = \Theta^Q(B^s)\cdot\theta^q(B)\,\omega_t$$

$$\rightarrow \Phi^P(B^s)\cdot\phi^p(B)(1-B^s)^D(1-B)^d \cdot x_t \cdot [\Theta^Q(B^s)\cdot\theta^q(B)]^{-1} = \omega_t$$

$$\text{let } z_t = [\Theta^Q(B^s)\cdot\theta^q(B)]^{-1}\cdot x_t$$

$$\rightarrow \begin{cases} x_t = \Theta^Q(B^s)\cdot\theta^q(B)\cdot z_t & (AR) \\ \omega_t = \Phi^P(B^s)\cdot\phi^p(B)\cdot(1-B^s)^D(1-B)^d\cdot z_t & (MA) \end{cases} \rightarrow ①$$

Now the state space model is given by..

$$\begin{cases} z_t = A\cdot z_{t-1} + e_t \\ x_t = C\cdot z_t + \gamma_t \end{cases} \rightarrow ②$$

$$e_t \sim N(0,R)$$
$$\gamma_t \sim N(0,Q)$$

Comparing equation ① and ② we get that the first term is AR and the second term is for MA.

We now need to find matrix A and C in order to determine the state space equivalence.

(2)

From Harvey representation of ARIMA

$$F = \begin{pmatrix} \phi_1 & 1 & 0 & 0 & \cdots & 0 \\ \vdots & 0 & 1 & 0 & \cdots & 0 \\ & & \ddots & & & \vdots \\ \phi_d & 0 & 0 & 0 & \cdots & 0 \end{pmatrix}$$

$$\therefore \left\{ Z_t = \underbrace{F \cdot Z_{t-1}}_{AR} + \underbrace{\omega_t \cdot \begin{pmatrix} 1 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{ds-1} \end{pmatrix}}_{MA} \right\} \rightarrow (3)$$

$$d = \max(p, q+1)$$

$$\left\{ X_t = (1 \; 0 \; 0 \cdots 0)^T \cdot Z_t \right\} \rightarrow (4)$$

We can rewrite the expression such that :

$$C \text{ matrix} = [1 \; \theta_1 \; \theta_2 \cdots \theta_{ds-1}]$$

$$A \text{ matrix} = \begin{bmatrix} \phi_1 & 1 & 0 & 0 & \cdots & 0 \\ \phi_2 & 0 & 1 & \cdots & & \vdots \\ \vdots & & & & & \\ \phi_d & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

The dimension of matrix A will be :

$$\max(p + Sp + d + S \cdot D, \; S \cdot Q + q)$$

# Time Series: Teaching session III

Sridhar Adhikarla.

## Assignment 1

Prove the Kalman filtering recursion for the following state space model with initial prior on the state

$f(z_1) = N(z_1; m_0, P_0)$ where $\epsilon_t \sim N(0, Q_t)$ and $v_t \sim N(0, R_t)$

$$z_t = A_{t-1} \cdot z_{t-1} + \epsilon_t$$

$$x_t = C_t z_t + v_t$$

① Particularly show that given $f(z_t | x_{1:t}) = N(z_t; m_{t|t}, P_{t|t})$, the predicted density $f(z_{t+1} | x_{1:t})$ is given by

$$f(z_{t+1} | x_{1:t}) = N(z_{t+1}; A_t m_{t|t}, A_t P_{t|t} A_t^T + Q_{t+1})$$

② Also show that given $f(z_t | x_{1:t-1}) = N(z_t; m_{t|t-1}, P_{t|t-1})$, the observation updated density $f(z_t | x_{1:t})$ is given by

$$f(z_t | x_{1:t}) = N(z_t; m_{t|t}, P_{t|t})$$

where:

$$m_{t|t} = m_{t|t-1} + K_t (x_t - C_t \cdot m_{t|t-1})$$

$$P_{t|t} = (I - K_t \cdot C_t) P_{t|t-1}$$

$$K_t = P_{t|t-1} \cdot C_t^T (C_t \cdot P_{t|t-1} \cdot C_t^T + R_t)^{-1}$$

(2)

Given

Q1

$$f(z_t \mid x_{1:t}) = N(z_t; m_{t|t}, P_{t|t})$$

Also from the transition model of the state space model we have..

$$z_t = A_{t-1} \cdot z_{t-1} + e_t \qquad e_t \sim N(0, Q_t)$$

$$\therefore f(z_{t+1} \mid z_t) = N(z_{t+1}; A_t \cdot z_t, Q_{t+1})$$

Joint distribution:

$$f(z_{t+1}, z_t \mid x_{1:t}) = f(z_t \mid x_{1:t}) \cdot f(z_{t+1} \mid z_t)$$

$$= N(z_t; m_{t|t}, P_{t|t}) \times N(z_{t+1}; A_t \cdot z_t, Q_{t+1})$$

$$= N\left( \begin{bmatrix} z_t \\ z_{t+1} \end{bmatrix}; \begin{bmatrix} m_{t|t} \\ A_t \cdot m_{t|t} \end{bmatrix}; \begin{bmatrix} P_{t|t} & \cdots \\ \cdots & A_t P_{t|t} A_t^T + Q_{t+1} \end{bmatrix} \right)$$

Marginalizing for $z_{t+1}$ we get..

$$\left\{ f(z_{t+1} \mid x_{1:t}) = N(z_{t+1}; A_t \cdot m_{t|t}, A_t \cdot P_{t|t} \cdot A_t^T + Q_{t+1}) \right\}$$

③

**Q-2** Given

$$f(Z_t \mid X_{1:t-1}) = N(Z_t;\, m_{t|t-1},\, P_{t|t-1})$$

$$f(Z_t \mid X_{1:t}) = \frac{f(Z_t, X_t \mid X_{1:t-1})}{f(X_t \mid X_{1:t-1})}$$

$$= \frac{f(Z_t \mid X_{1:t-1}) \cdot f(X_t \mid Z_t, X_{1:t-1})}{f(X_t \mid X_{1:t-1})}$$

$$= \frac{f(Z_t \mid X_{1:t-1}) \cdot f(X_t \mid Z_t)}{f(X_t \mid X_{1:t-1})}$$

$$\propto f(Z_t \mid X_{1:t-1}) \cdot f(X_t \mid Z_t)$$

likelihood function..

$$\propto N(Z_t;\, m_{t|t-1},\, P_{t|t-1}) \cdot N(X_t;\, C_t \cdot Z_t,\, R_t)$$

$$\propto N\left(\begin{bmatrix} Z_t \\ X_t \end{bmatrix};\, \begin{bmatrix} m_{t|t-1} \\ C_t \cdot m_{t|t-1} \end{bmatrix},\, \begin{bmatrix} P_{t|t-1} & P_{t|t-1} \cdot C_t^T \\ C_t \cdot P_{t|t-1} & C_t \cdot P_{t|t-1} \cdot C_t^T + R_t \end{bmatrix}\right)$$

using conditioning rules of normal distribution

$$f(Z_t \mid X_{1:t}) = N\Big(Z_t;\, \underbrace{m_{t|t-1} + P_{t|t-1} \cdot C_t^T (C_t \cdot P_{t|t-1} \cdot C_t^T + R_t)^{-1} \cdot (X_t - C_t \cdot m_{t|t-1})}_{m_{t|t}},\, \underbrace{P_{t|t-1} - P_{t|t-1} \cdot C_t^T (C_t \cdot P_{t|t-1} \cdot C_t^T + R_t)^{-1} \cdot (C_t \cdot P_{t|t-1})}_{P_{t|t}}\Big)$$

$K_t$

$$\therefore \boxed{f(Z_t \mid X_{1:t}) = N(Z_t;\, m_{t|t},\, P_{t|t})}$$

# 732A62 Lab C Group 12

*Gustav Lundberg (guslu389), Raymond Sseguya (rayss753)*

*2019-10-10*

## Contents

# 1 Kalman Filter implementation

## 1.a Model expression

*Write down the expression for the state space model that is being simulated.*

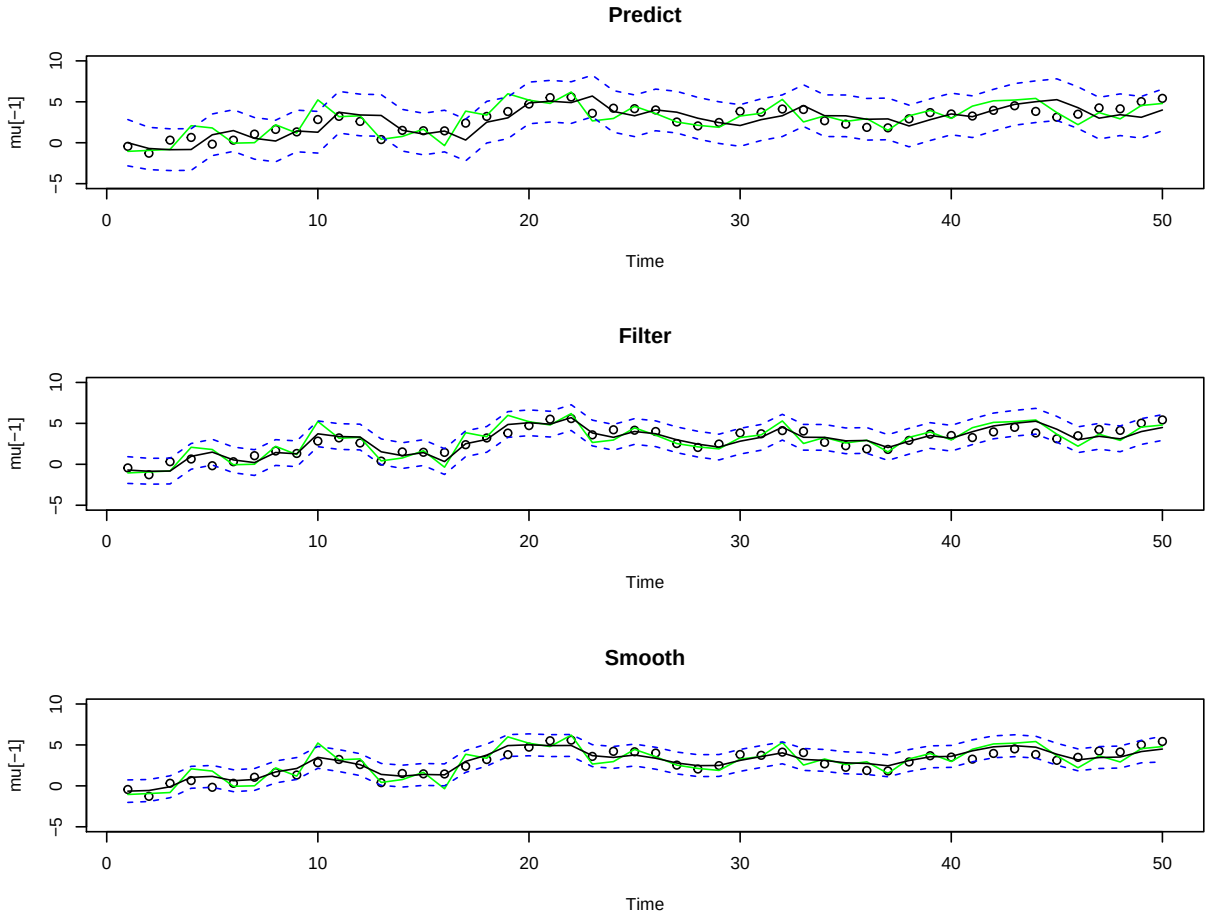
In the simulated model, both the transition matrix $A_{t-1}$ and the emission matrix $C_1$ are equal to 1. This can be seen by the line that `mu = cumsum(w)` which means that any mu is the sum of all the previous noises, or the previous `mu` plus the current noise, which gives $z_t = 1 \cdot z_{t-1} + e_t$. The same reasoning goes for $x_t = 1 \cdot z_t + \nu_t$,

$R_t$ and $Q_t$ are also equal to 1 since both $e_t$ and $\nu_t$ are generated by `rnorm(0,1)`.

## 1.b Script vs MA-smoother

*Run this scrip and compare the filtering results with a moving average smoother of order 5.*

```
library(astsa)
# generate data
Q <- R <- 1
set.seed(1); num = 50
w = Q * rnorm(num+1,0,1); v = R * rnorm(num,0,1)
mu = cumsum(w) # state: mu[0], mu[1],..., mu[50]
y = mu[-1] + v # obs: y[1],..., y[50]
# filter and smooth (Ksmooth0 does both)
ks = Ksmooth0(num, y, A=1, mu0=0, Sigma0=1, Phi=1, cQ=1, cR=1) # start figure
```




```
mu[1]; ks$x0n; sqrt(ks$P0n) # initial value info
```

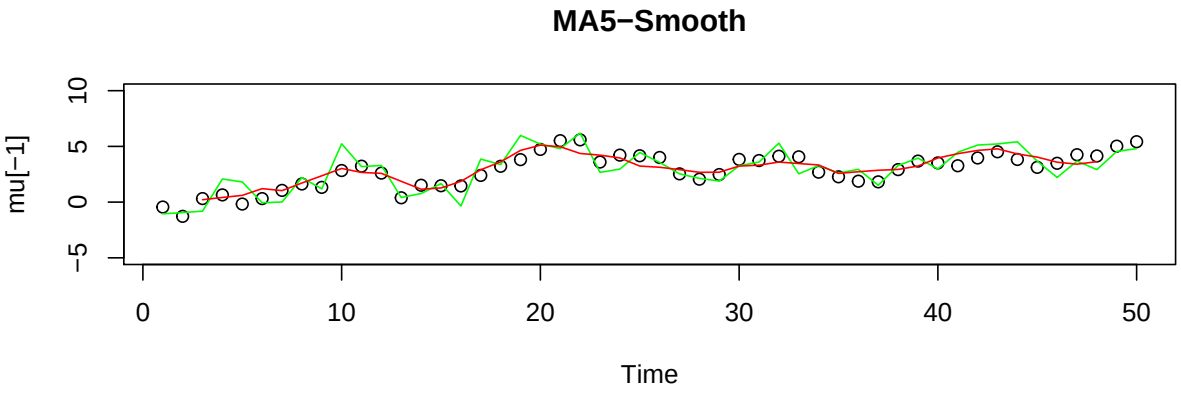
```
## [1] -0.6264538
```

```
##             [,1]
## [1,] -0.3241541
```

```
##           [,1]
## [1,] 0.7861514
```

The moving average-smoother is done with the `filter()`-function.

```r
par(mfrow=c(1,1))
ma5 <- filter(y, filter = rep(0.2,5), method = "conv")
plot(Time, mu[-1], main='MA5-Smooth', ylim=c(-5,10))
lines(Time,y,col="green")
lines(ma5, col = 2)
```



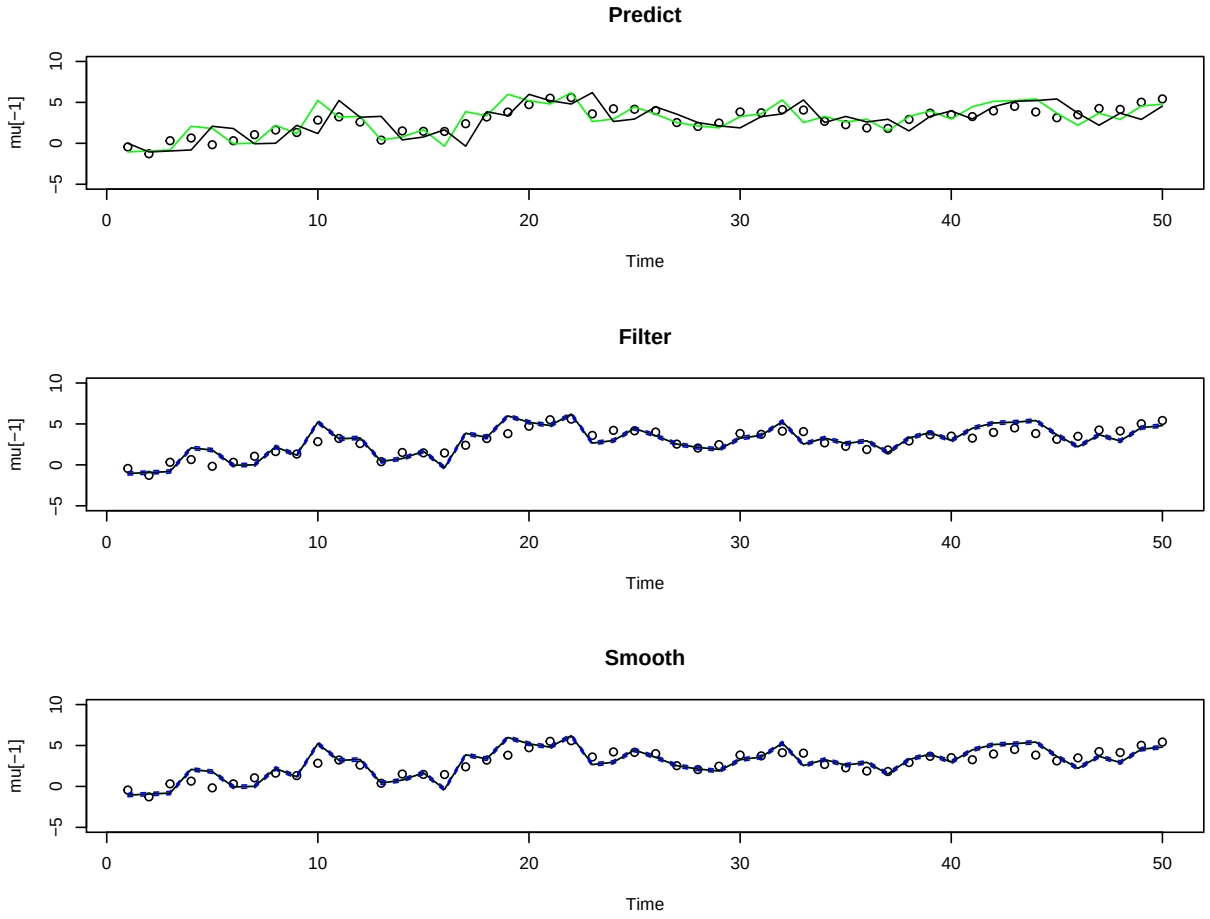

The moving average smoother is quite close to the Kalman smoother, only that the moving average has to drop two points in the beginning and end of the series while the Kalman smoother can actually produce an estimate for these points.

## 1.c Comparison with different covariances 1

*Also, compare the filtering outcome when R in the filter is 10 times smaller than its actual value while Q in the filter is 10 times larger than its actual value. How does the filtering outcome varies?*

A small R means that the observation error is very small, i.e. the observations are very close to the real signal.

```
# generate data
set.seed(1); num = 50
w = rnorm(num+1,0,1); v = rnorm(num,0,1)
mu = cumsum(w) # state: mu[0], mu[1],..., mu[50]
y = mu[-1] + v # obs: y[1],..., y[50]
# filter and smooth (Ksmooth0 does both)
ks = Ksmooth0(num, y, A=1, mu0=0, Sigma0=1, Phi=1, cQ=10, cR=0.1) # start figure
```




```
mu[1]; ks$x0n; sqrt(ks$P0n) # initial value info
```

```
## [1] -0.6264538
```

```
##              [,1]
## [1,] -0.01044278
```

```
##           [,1]
## [1,] 0.9950377
```

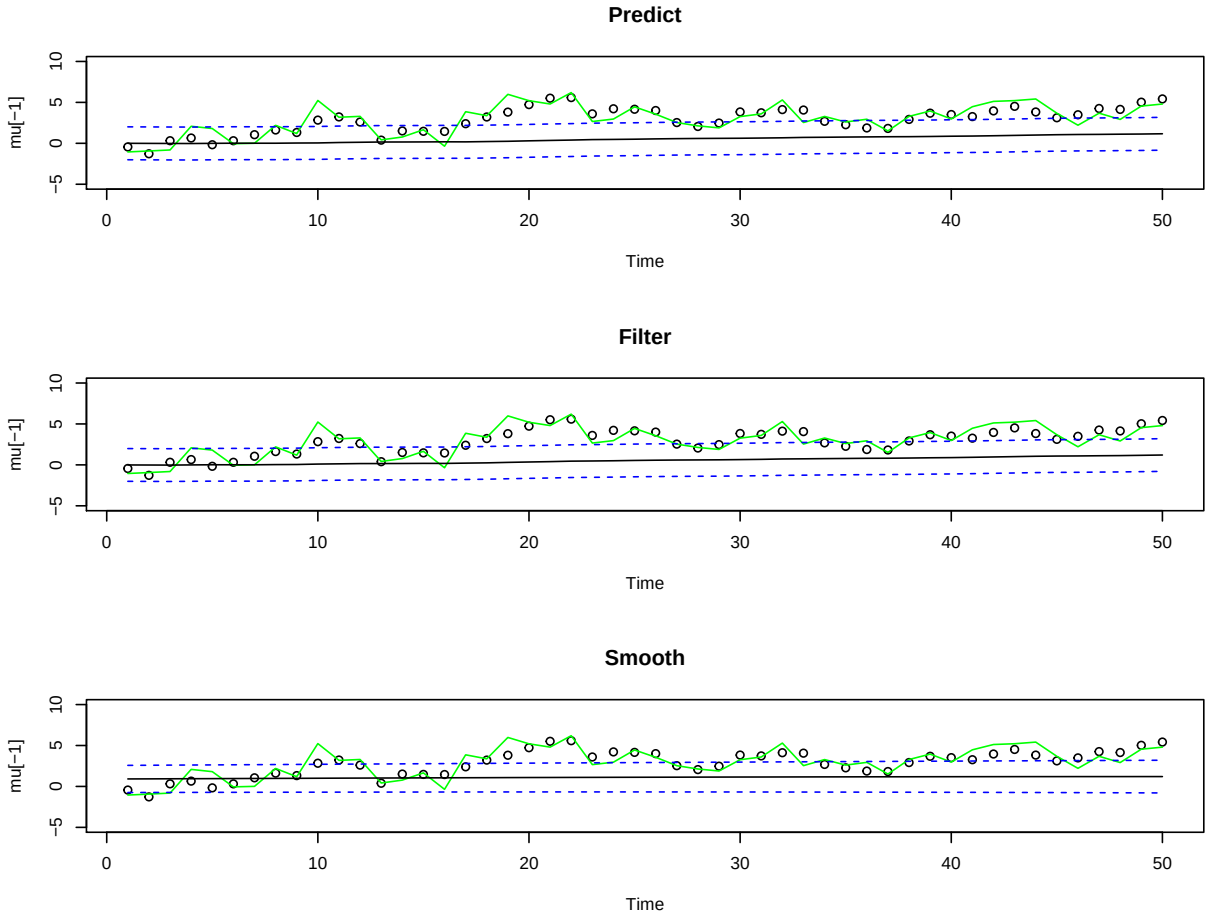
Since we now imply that the observations are very close to the original signal, the fits are forced to follow the observations closely and thus miss a lot of the actual signal. We basically overfit using such a small $R_t$.

## 1.d Comparison with different covariances 2

> *Now compare the filtering outcome when R in the filter is 10 times larger than its actual value while Q in the filter is 10 times smaller than its actual value. How does the filtering outcome varies?*

In this case, the observation noise is very large, i.e. we have a rather constant process, but the sensors are bad at reading it.

```
# generate data
set.seed(1); num = 50
w = rnorm(num+1,0,1); v = rnorm(num,0,1)
mu = cumsum(w) # state: mu[0], mu[1],..., mu[50]
y = mu[-1] + v # obs: y[1],..., y[50]
# filter and smooth (Ksmooth0 does both)
ks = Ksmooth0(num, y, A=1, mu0=0, Sigma0=1, Phi=1, cQ=0.1, cR=10) # start figure
```




```
mu[1]; ks$x0n; sqrt(ks$P0n) # initial value info
```

```
## [1] -0.6264538
```

```
##           [,1]
## [1,] 0.905755
```

```
##          [,1]
## [1,] 0.82731
```

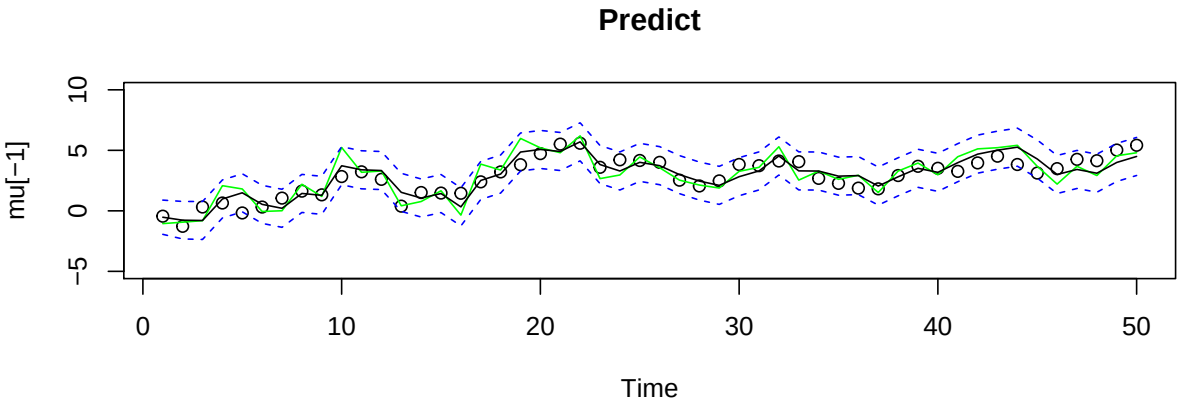
Here we reason that the observations are rather poor, but the signal is almost constant since the $Q_t$ is small, i.e the difference from the previous state is small.

## 1.e Custom Kalman

*Implement your own Kalman filter and replace ksmooth0 function with your script.*

Since the formulas are given with the time-related indexes, we might as well build a filtering function that handles time-varying covariance- and transformation matrices for the transition and emission.

```
my_kalman <- function(x, R, Q, A, C, m0, P0){

  if (!is.matrix(x)){ x <- matrix(x) }
  nsteps <- nrow(x)

  same_A <- !is.list(A); if(same_A) { A <- list(A) }
  same_Q <- !is.list(Q); if(same_Q) { Q <- list(Q) }
  same_C <- !is.list(C); if(same_C) { C <- list(C) }
  same_R <- !is.list(R); if(same_R) { R <- list(R) }

  m <- matrix(NA, nrow = nsteps, ncol = ncol(x))
  P <- vector("list", length = nsteps)
  mpred <- m0
  Ppred <- P0

  for (i in seq(1:nsteps)){
    # To enable varying C, R, Q and A per timestep
    ci = ifelse(same_C, 1, i)
    ri = ifelse(same_R, 1, i)
    qi = ifelse(same_Q, 1, i)
    ai = ifelse(same_A, 1, i)
    # Observation update step
    K <- Ppred %*% t(C[[ci]]) %*% solve(C[[ci]] %*% Ppred %*% t(C[[ci]]) + R[[ri]])
    m[i, ] <- mpred + K %*% (x[i, ] - C[[ci]] %*% mpred)
    P[[i]] <- (diag(ncol(x)) - K %*% C[[ci]]) %*% Ppred

    # Prediction step
    mpred <- A[[ai]] %*% m[i, ]
    Ppred <- A[[ai]] %*% P[[i]] %*% t(A[[ai]]) + Q[[qi]]
  }
  list(means = m, covs = P)
}

my_kf <- my_kalman(y, A=1, Q=1, C=1, R=1, m0=0, P0=1)
```




The custom built filter actually does a fairly good job of estimating the series. All of the actual signal is within the confidence bands.

## 1.f Kalman gain interpretation

*How do you interpret the Kalman gain?*

The Kalman gain controls the level of trust you have in the observed value compared to the predicted value at each timestep as seen in the line `m[i, ] <- mpred + K %*% (x[i, ] - C[[ci]] %*% mpred)`. The higher the gain is, the more influence the difference between observed value and predicted value will have on the updated mean value(s).

On the same page, it also controls the amount of influence that the emission models covariance has in the updated covariance estimate of each time step.