Time Series Analysis - lab02

Lennart Schilling (lensc874)

2019-09-29

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 ϕ_{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 ϕ_{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().

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

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

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

```
[1] 0.1170643
```

Computing theoretical value.

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

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

```r
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 ϕ_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 $\sqrt{\frac{\sigma^2}{n}}$. σ^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 $\hat{\phi}_2 \pm 1.96 * \sqrt{\sigma^2_{\hat{\phi}_2}}$.

```r
# 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.

```r
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

```r
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 ϕ_1 and ϕ_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 ϕ_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.

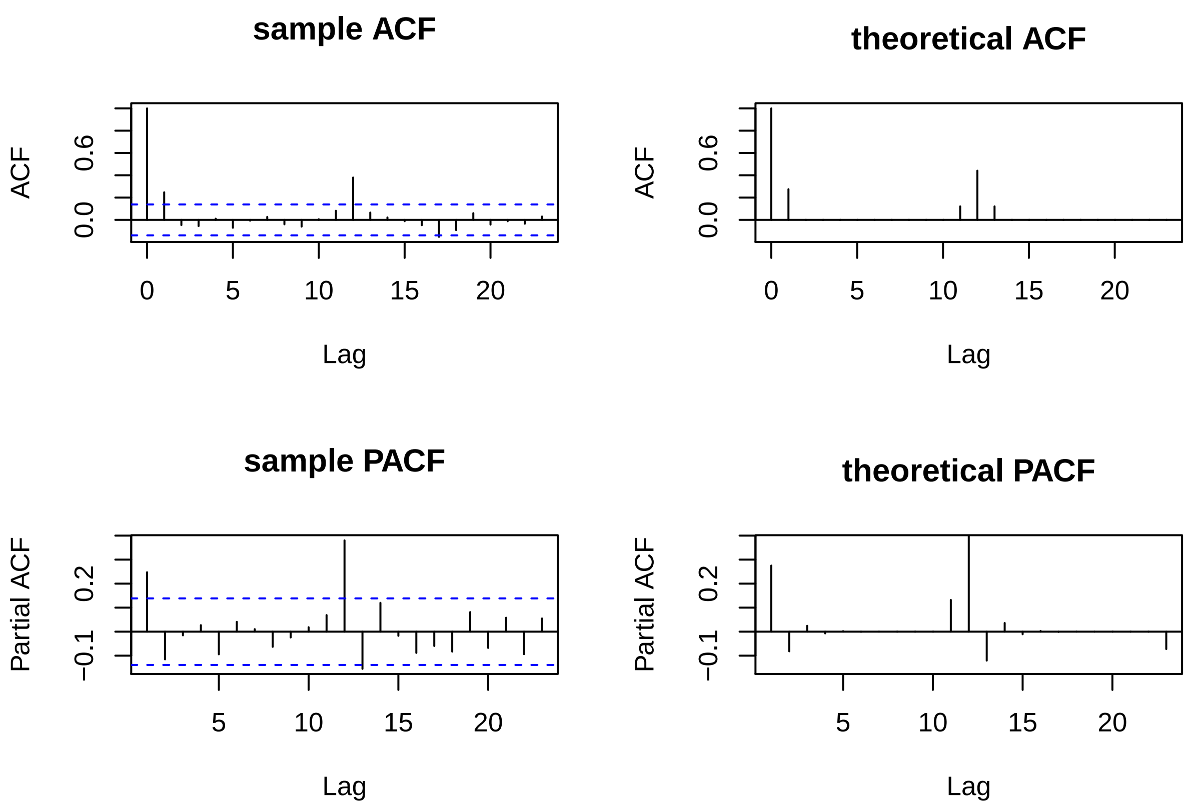
Using the derived MA(13)-model, we can generate 200 observations with coefficients $\Theta = 0.6$ and $\theta = 0.3$.

```
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.

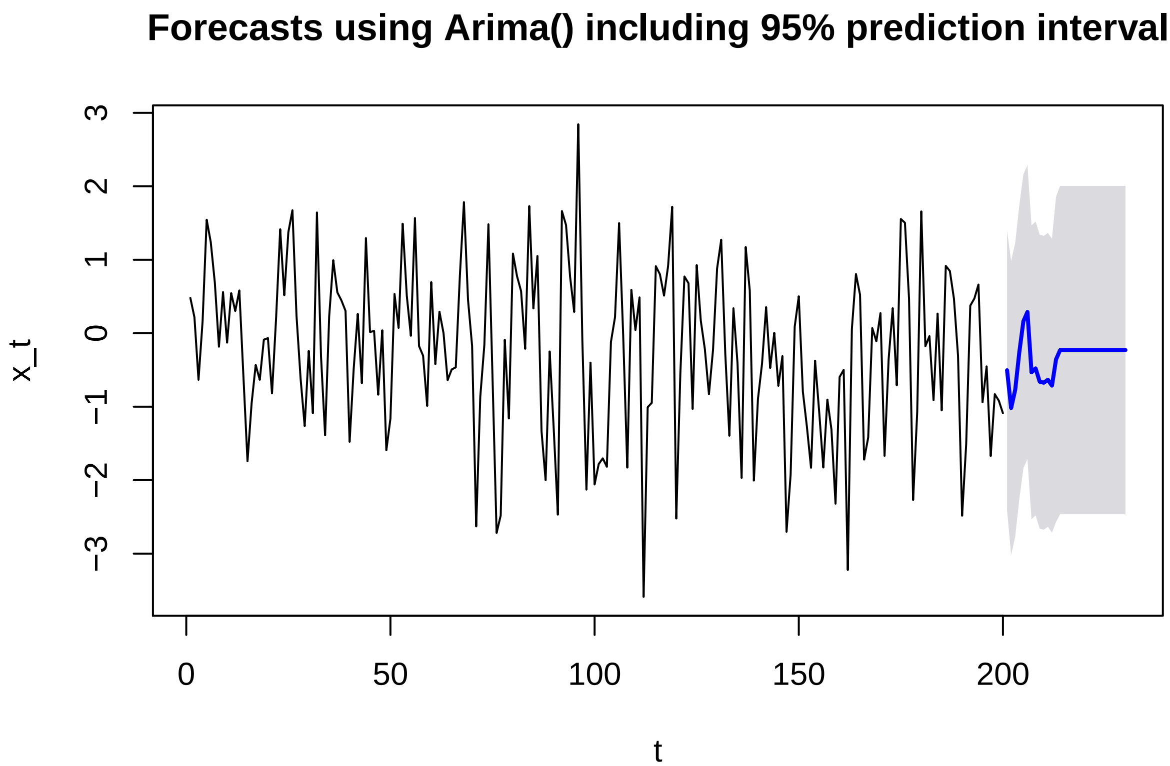
Generate 200 observations of a seasonal $ARIMA(0,0,1) \times (0,0,1)_{12}$ model with coefficients $\Theta = 0.6$ and $\theta = 0.3$ by using arima.sim(). Fit $ARIMA(0,0,1) \times (0,0,1)_{12}$ model to the data, compute forecasts and a prediction band 30 points ahead and plot the original data and the 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")
```

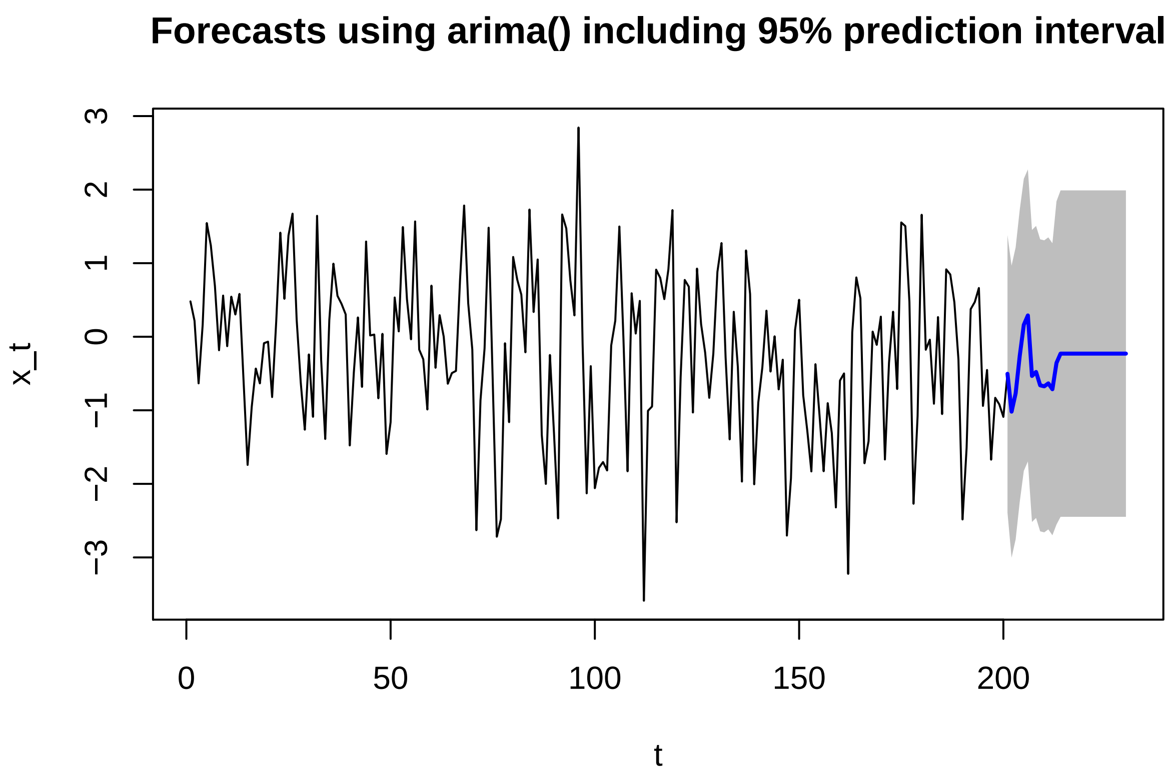


Fitting model and performing forecast using built-in R functions.

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 $\hat{x} \pm 1.96 * se$.

```r
# 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),
```

```r
      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)
```



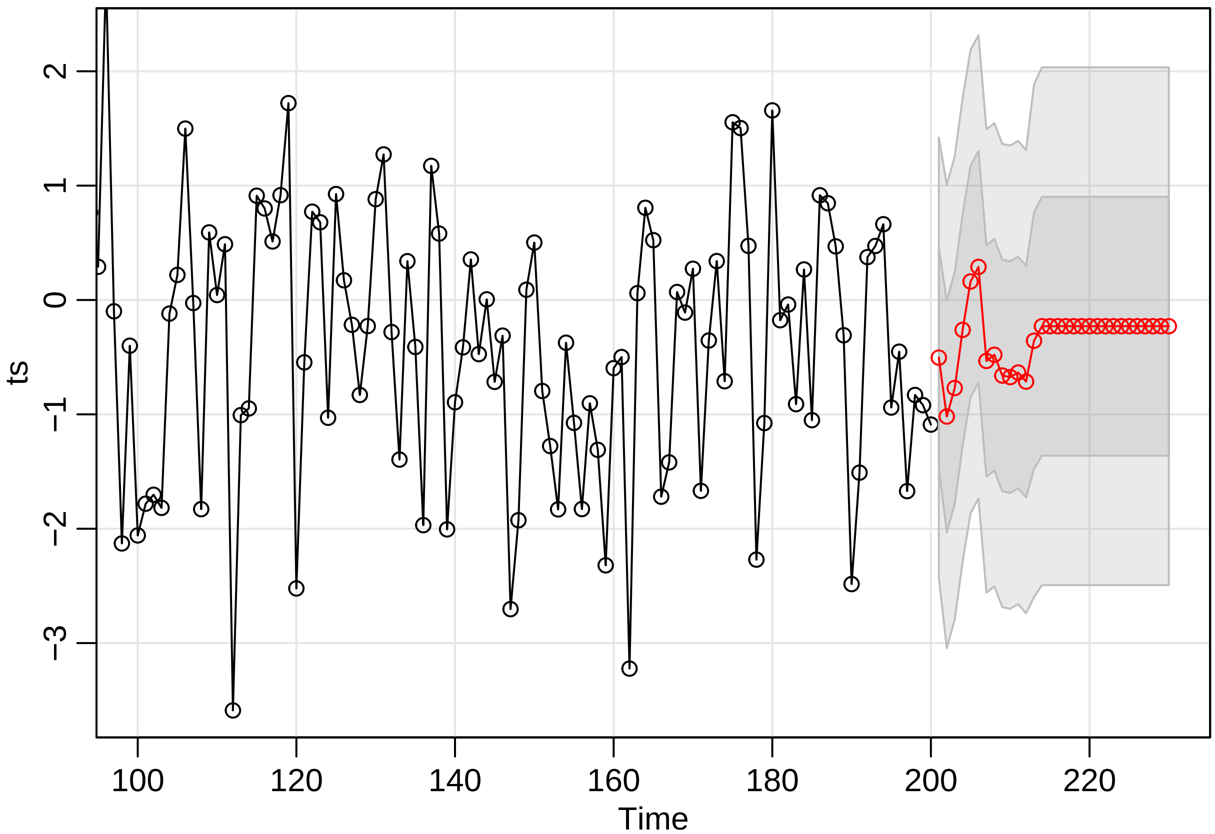
As it can be seen, the manual implementation using only built-in R functions returns the same plot as the one from the `forecast`-package.

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.

```r
# Importing library.
library(astsa)

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

```r
        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.

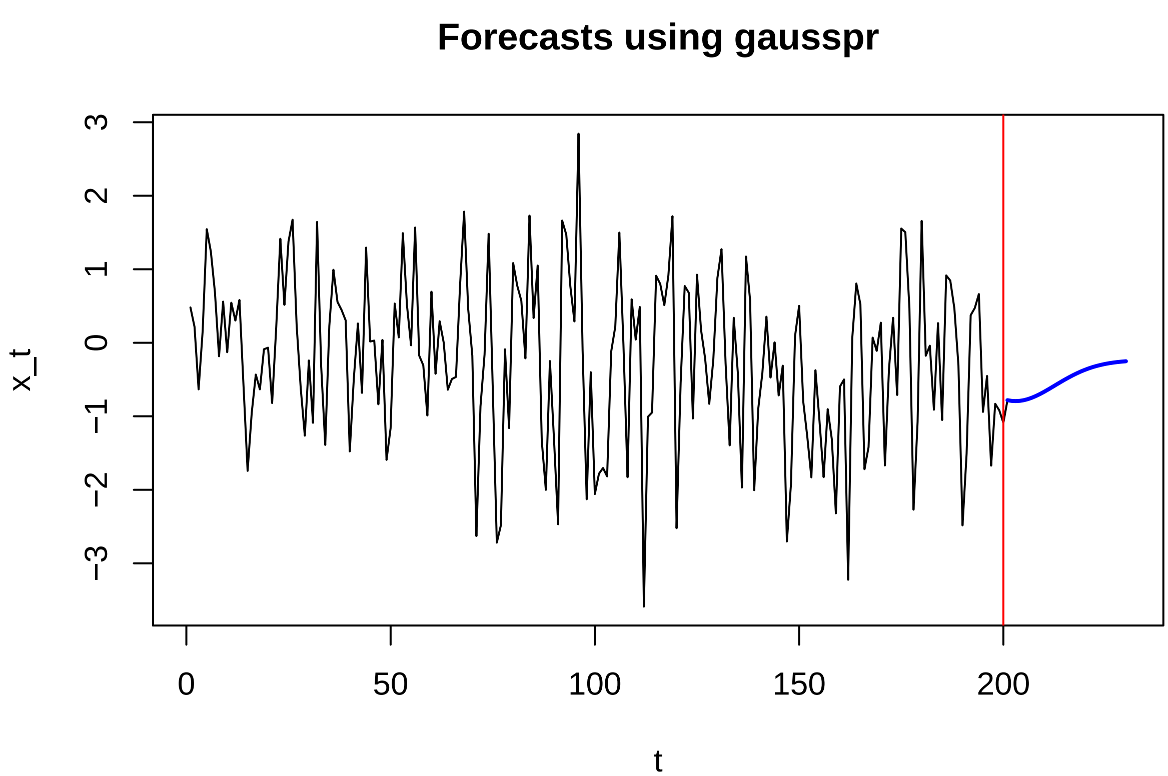
Fitting model and performing forecast using kernlab-package.

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.

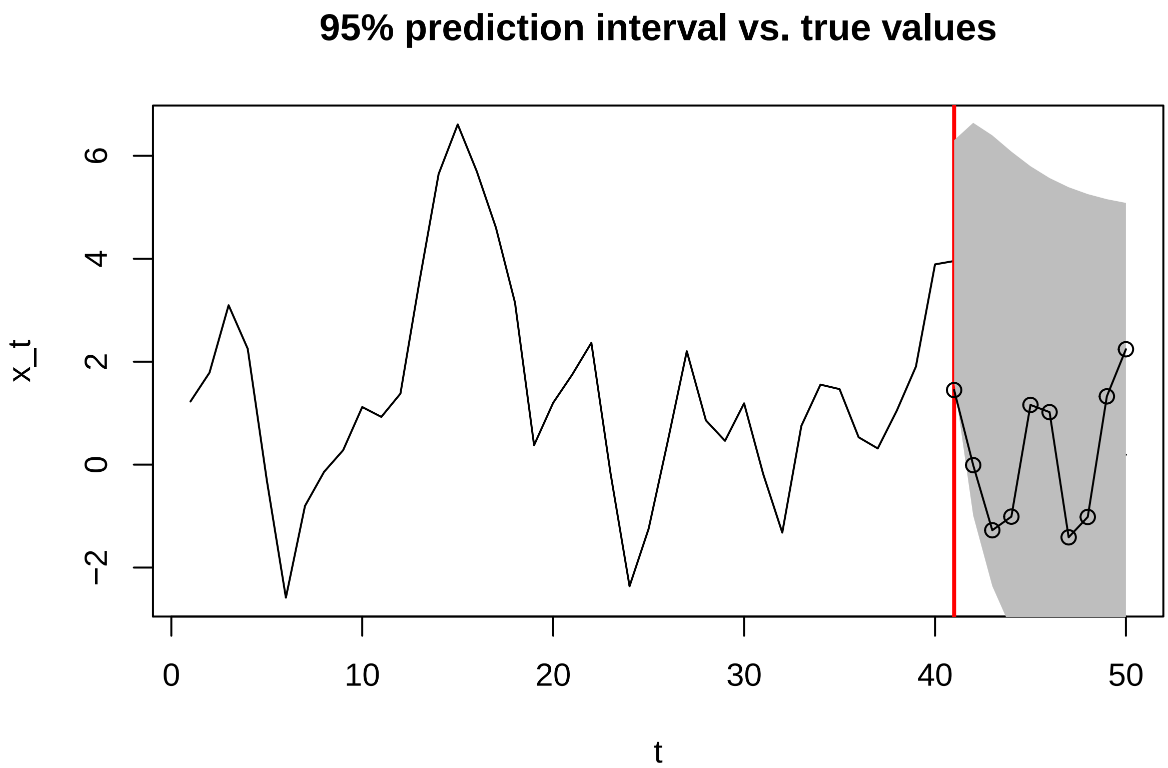
```
set.seed(12345)
ts = arima.sim(list(ar = 0.7,
                    ma = 0.5),
               n = 50)
```

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.

Including helpful tables from slides

From the lecture slides, we can include the following tables which help finding a suitable model by looking at the ACF/PACF.

```r
knitr::kable(x = matrix(data = c("Tails off", "Cuts off after lag p",
                                 "Cuts off after lag q", "Tails off",
                                 "Tails off", "Tails off"),
                        nrow = 2,
                        dimnames = list(c("ACF", "PACF"),
                                        c("AR(p)", "MA(q)", "ARMA(p,q)"))),
             caption = "Analysing ACF/PACF for non-seasonal model")
```

Table 4: Analysing ACF/PACF for non-seasonal model

	AR(p)	MA(q)	ARMA(p,q)
ACF	Tails off	Cuts off after lag q	Tails off
PACF	Cuts off after lag p	Tails off	Tails off

```r
knitr::kable(x = matrix(data = c("Tails off at lags ks", "Cuts off after lag P_s",
                                 "Cuts off after lag Q_s","Tails off at lags ks",
                                 "Tails off at lags ks", "Tails off at lags ks"),
                        nrow = 2,
                        dimnames = list(c("ACF*", "PACF*"),
                                        c("AR(P)_s", "MA(Q)_s", "ARMA(P,Q)_s"))),
             caption = "Analysing ACF/PACF for seasonal model")
```

Table 5: Analysing ACF/PACF for seasonal model

	AR(P)_s	MA(Q)_s	ARMA(P,Q)_s
ACF*	Tails off at lags ks	Cuts off after lag Q_s	Tails off at lags ks
PACF*	Cuts off after lag P_s	Tails off at lags ks	Tails off at lags ks

The values at nonseasonal lags $h \neq$ ks for $k = 1, 2, ...$ are zero.

Importing astsa package and reading data.

```r
library(astsa)
data(chicken)
data(so2)
data(EQcount)
data(HCT)
```

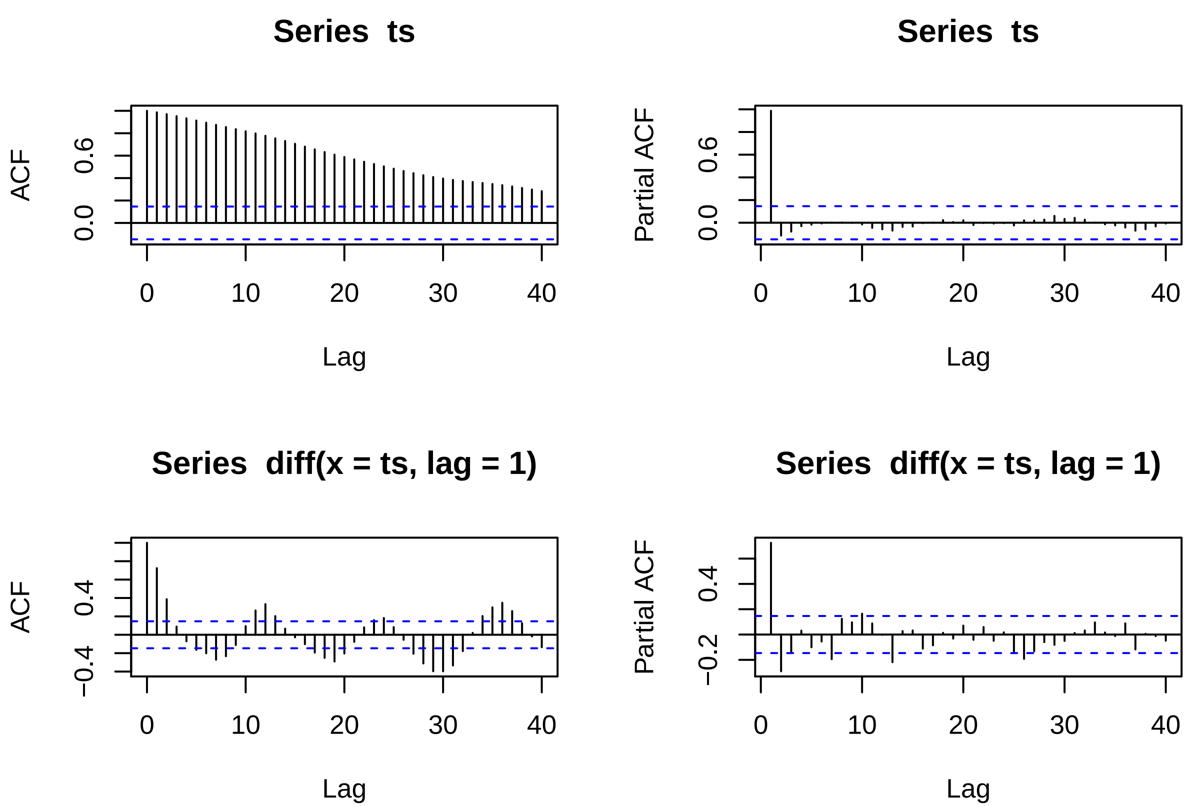
Implementing function to plot ACF, PACF all together.

```
plot_all_p_acf = function(ts, n_lags) {
  # Defining following plots to be next to / on top of each other.
  par(mfrow = c(2, 2))
  # Plotting.
  acf(x = ts, lag.max = n_lags)
  pacf(x = ts, lag.max = n_lags)
  acf(diff(x = ts, lag = 1), lag.max = n_lags)
  pacf(diff(x = ts, lag = 1), lag.max = n_lags)
}
```

Plotting and analysing regarding chicken time series.

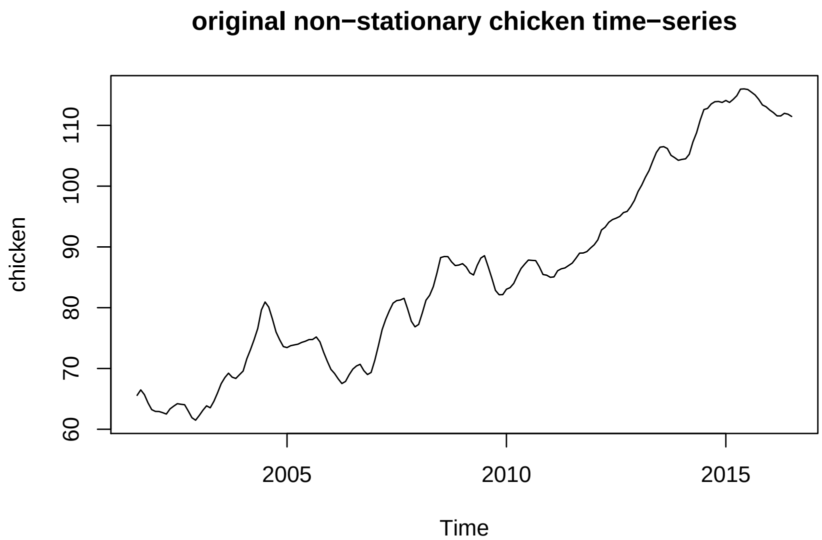
To show the correct values for the lags on the x-axis, we need to wrap the `ts()`-function around the data. This will not change the time series, but somehow the plot shows the correct values for the x-axis (lags up to 40 by steps of 1).

```
plot_all_p_acf(ts = ts(chicken),
               n_lags = 40)
```



By looking first at the plots (ACF, PACF) for the original time series `ts`, we see that the ACF tails off and the PACF cuts off after lag 1. Thus, we may conclude that an AR(1) model might describe the data well. However, we can see the ACF tails off very slow, no exponential decaying is visible. Furthermore, the partial autocorrelation for the first lag is almost 1. This indicates that the original `chicken` time-series is not stationary. Plotting the time series `ts` confirms that:

```r
# Plotting original chicken time series.
ts.plot(chicken, main = "original non-stationary chicken time-series")
```



Therefore, we conclude that differencing should be done. We will concentrate on the two plots at the bottom for the differenced ($\mathbf{d = 1}$) time series. The ACF and PACF for the non-stationary original time series do not return any value for us anymore related to the suggestion of a good model.

First, we will analyze the possibility of a non-seasonal $ARIMA(p, d, q)$-model. The ACF shows that the autocorrelation decays exponentially. Lag 1 is still characterized by a very high value, the autocorrelation with lag 2 has already decreased a lot. For the other lags larger than 2, the autocorrelation values varies roughly between +0.2 and -0.2. This exponential decay indicates that there must be an AR(p)-part, because it "tails off" (referring to table 4). The PACF confirms that assumption, because we can see a cut off after lag 2. Thus, $\mathbf{p = 2}$. An MA(q)-part can not be identified, because neither does the ACF cut off after any lag q nor does the PACF taill off. Therefore, $\mathbf{q = 0}$. Combining the derived parameter, a non-seasonal $ARIMA(2, 1, 0)$ might be a reasonable guess.

However, we can identify a repeating pattern within the ACF and therefore a seasonal part as well. That is why we will analyze the possibility of a seasonal $(P, D, Q)_s$-model as well. Within the ACF, the repeating pattern is visible over every 12 month. Thus, $\mathbf{S = 12}$. Since the plots refer to the first differenced data, $\mathbf{D = 1}$. The ACF does not cut off after lag Q_s so that we conclude that there is no seasonal $MA(Q_s)$-part. Again, the PACF also does not tail off which supports the assumption that there is no $MA(Q_s)$-part. Therefore, $\mathbf{Q = 0}$. Combining the derived parameter, a seasonal model $(0, 1, 0)_{12}$ might be a reasonable guess.

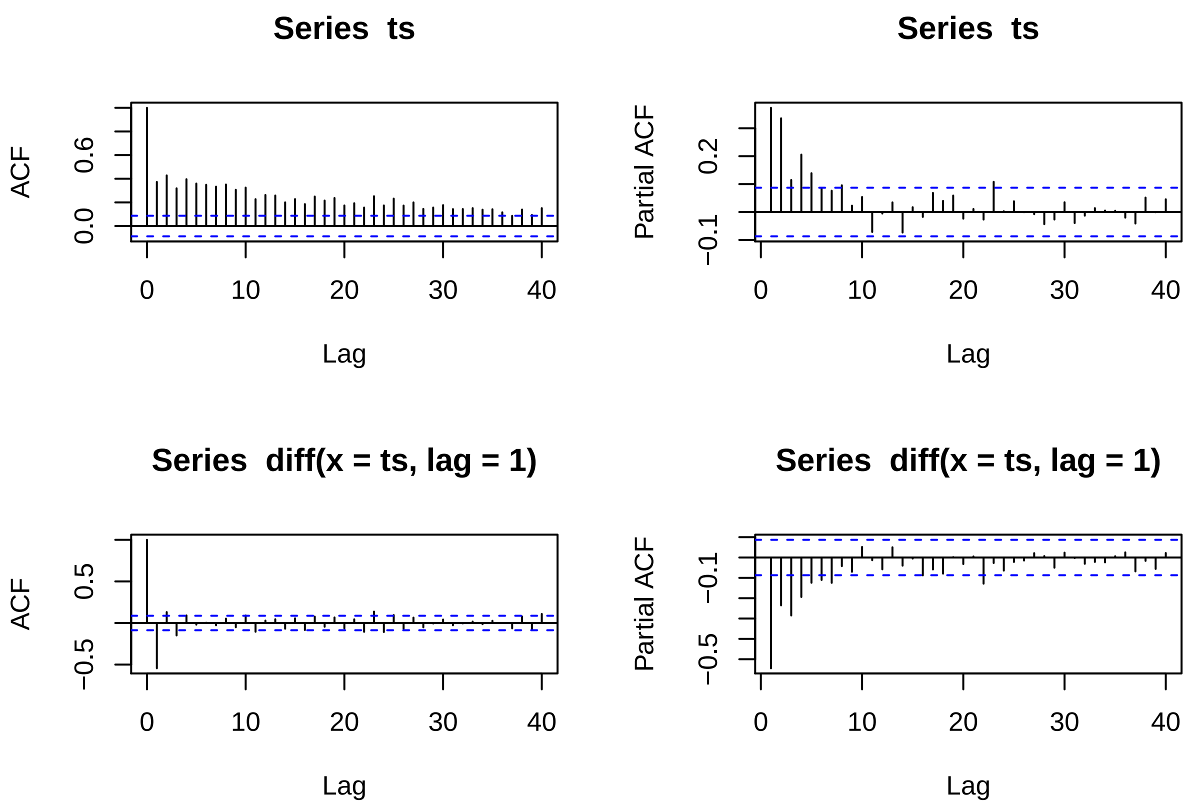
Combining the non-seasonal and seasonal part, we have identified the following model as a reasonable guess:

$$ARIMA(2, 1, 0) \times (0, 1, 0)_{12}$$

Plotting and analysing regarding so2 time series.

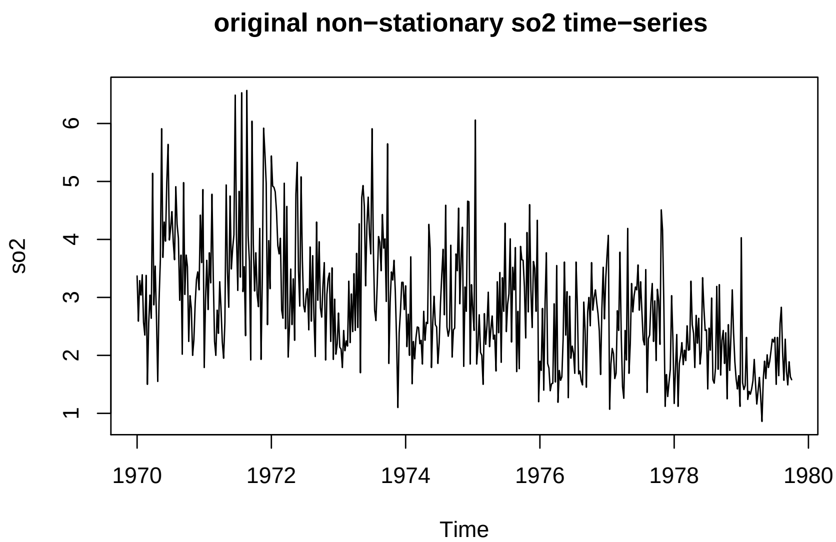
To show the correct values for the lags on the x-axis, we need to wrap the `ts()`-function around the data. This will not change the time series, but somehow the plot shows the correct values for the x-axis (lags up to 40 by steps of 1).

```r
plot_all_p_acf(ts = ts(so2),
               n_lags = 40)
```

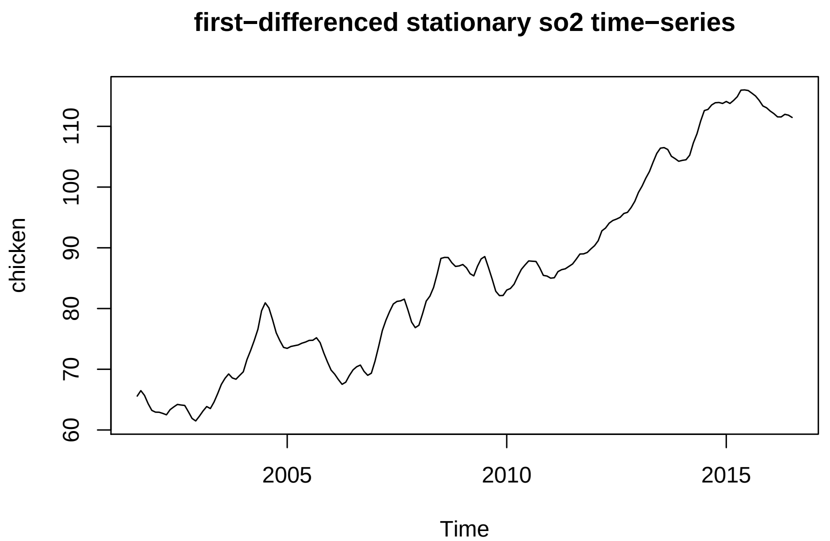


By first looking at the plots for the original time series, we might conclude that a non-seasonal ARMA(p,q) might be a good fit. This assumption is based on the analysis that both the ACF and PACF tails off. The PACF tails off relatively quickly and exponentially. However, similar to the `chicken` time series, the ACF does not tail off exponentially, but very slowly. Again, this is an indicator for a non-stationary time series. Plotting both the original `so2` time series and the first-differenced time series confirms that.

```
# Plotting original so2 time series.
ts.plot(so2, main = "original non-stationary so2 time-series ")
```




```r
# Plotting first-differenced so2 time series.
ts.plot(chicken, main = "first-differenced stationary so2 time-series")
```

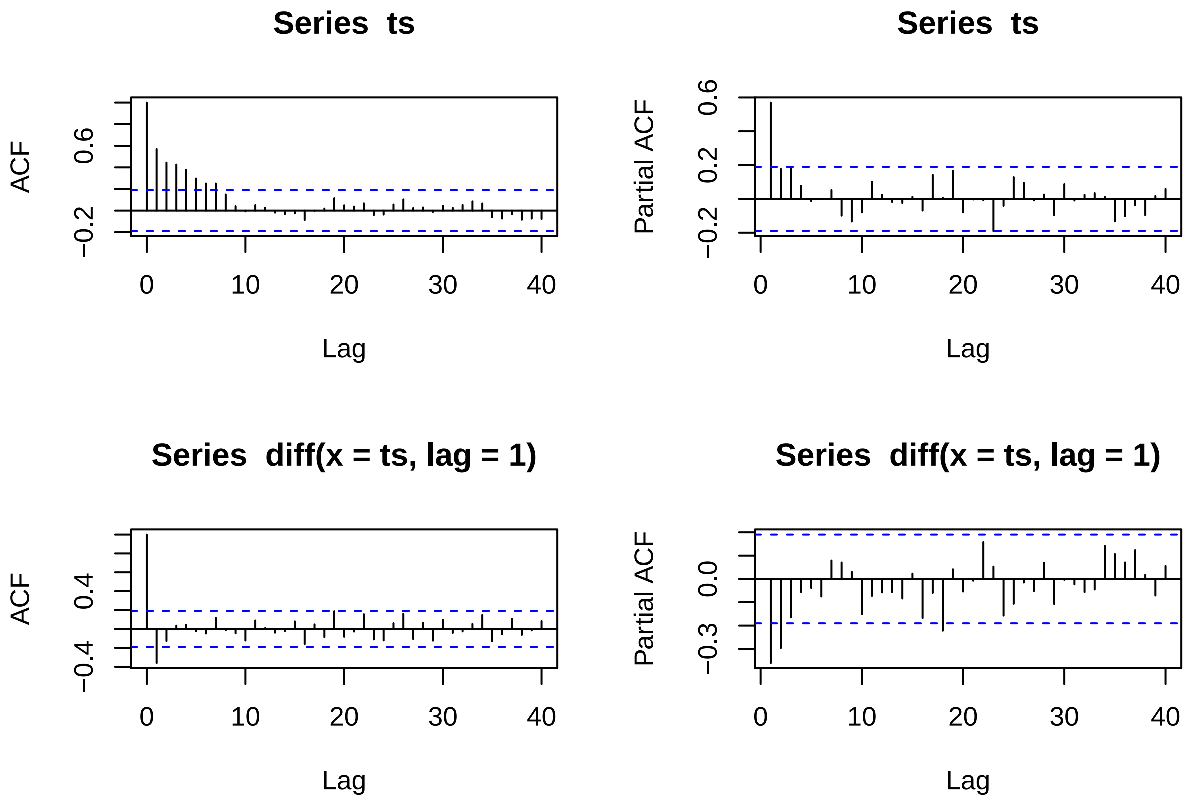


We clearly can see that the time first-differenced time series is more stationary than the original time series. Thus, again we will concentrate on the ACF and PACF for the first-differenced time series.

This time the analysis is more straight: In the ACF we can see that it cuts off after lag 1. Furthermore, it tails off exponentially in the PACF. A seasonal pattern is not visible, thus a non-seasonal $ARIMA(0, 1, 1)$-model seems to be a reasonable fit.

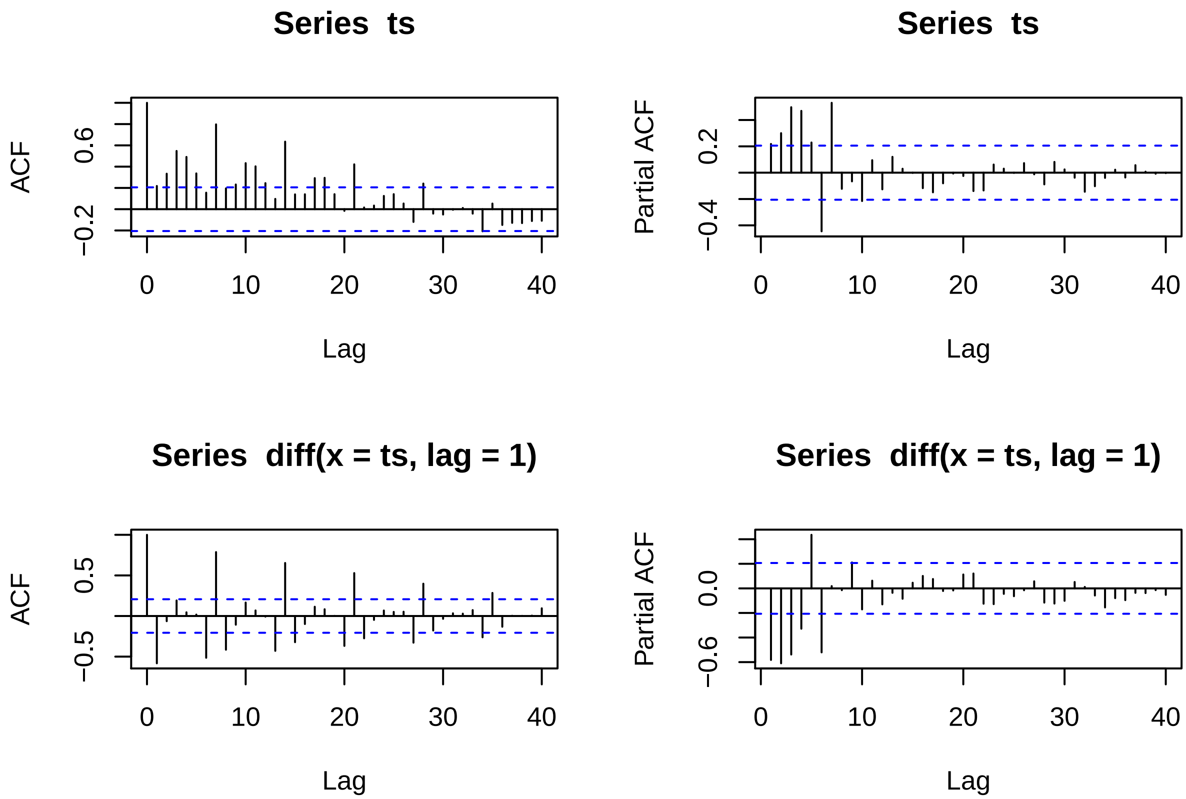
Plotting and analysing regarding EQcount time series.

```r
plot_all_p_acf(ts = EQcount,
               n_lags = 40)
```



Plotting and analysing regarding HCT time series.

```
plot_all_p_acf(ts = HCT,
               n_lags = 40)
```


Series ts
ACF
0.6
-0.2
0
10
20
30
40
Lag
Series ts
Partial ACF
0.2
-0.4
0
10
20
30
40
Lag
Series diff(x = ts, lag = 1)
ACF
0.5
-0.5
0
10
20
30
40
Lag
Series diff(x = ts, lag = 1)
Partial ACF
0.0
-0.6
0
10
20
30
40
Lag

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.

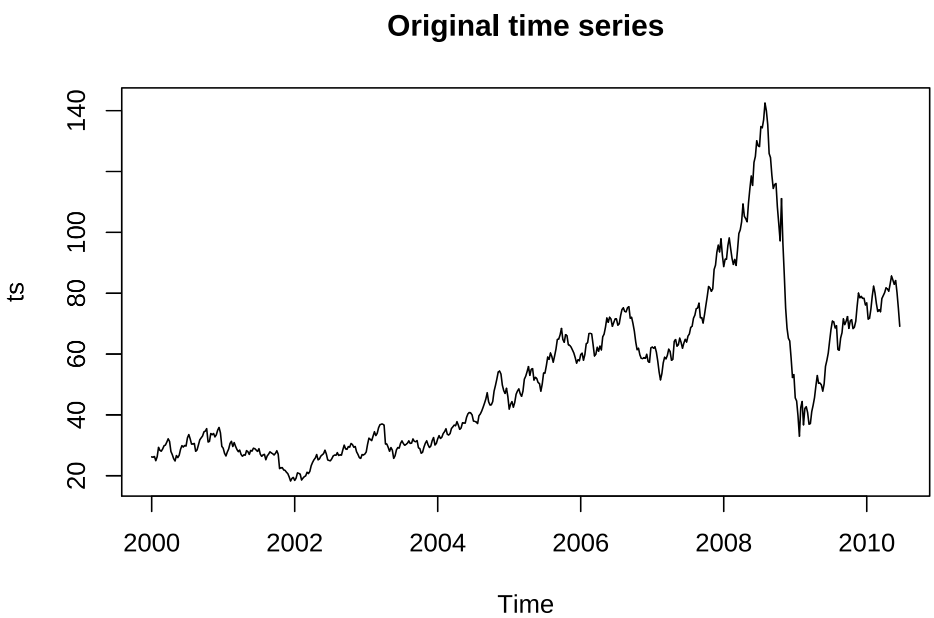
The follwoing complete ARIMA modeling cycle is performed by the following steps:

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.

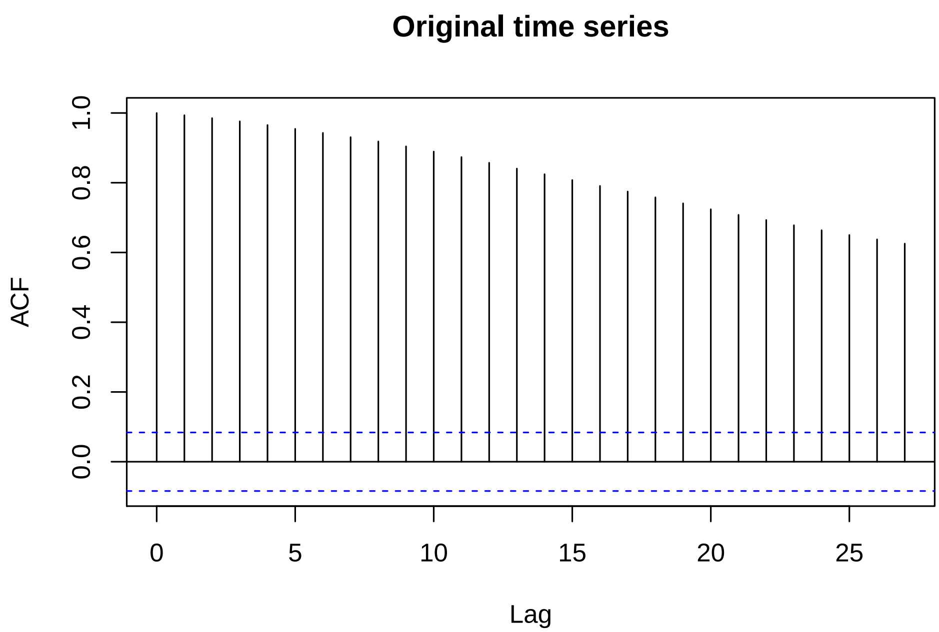
```r
# Loading original time series.
library(astsa)
data(oil)
ts = oil

# 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 α, 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*.

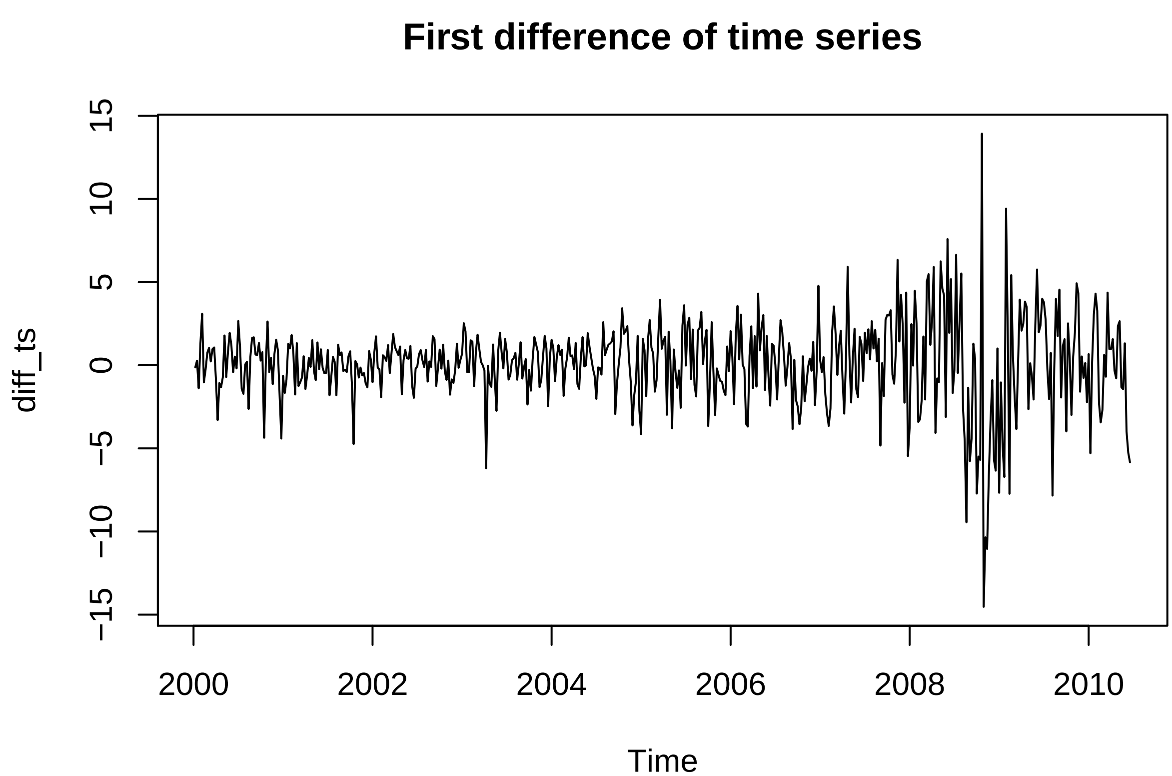
```r
# Differencing / Transforming original ts.
diff_ts = diff(ts)
log_ts = log(ts)
diff_log_ts = diff(log(ts))

# 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 6: p-values after performing Dickey-Fuller Unit-Root tests

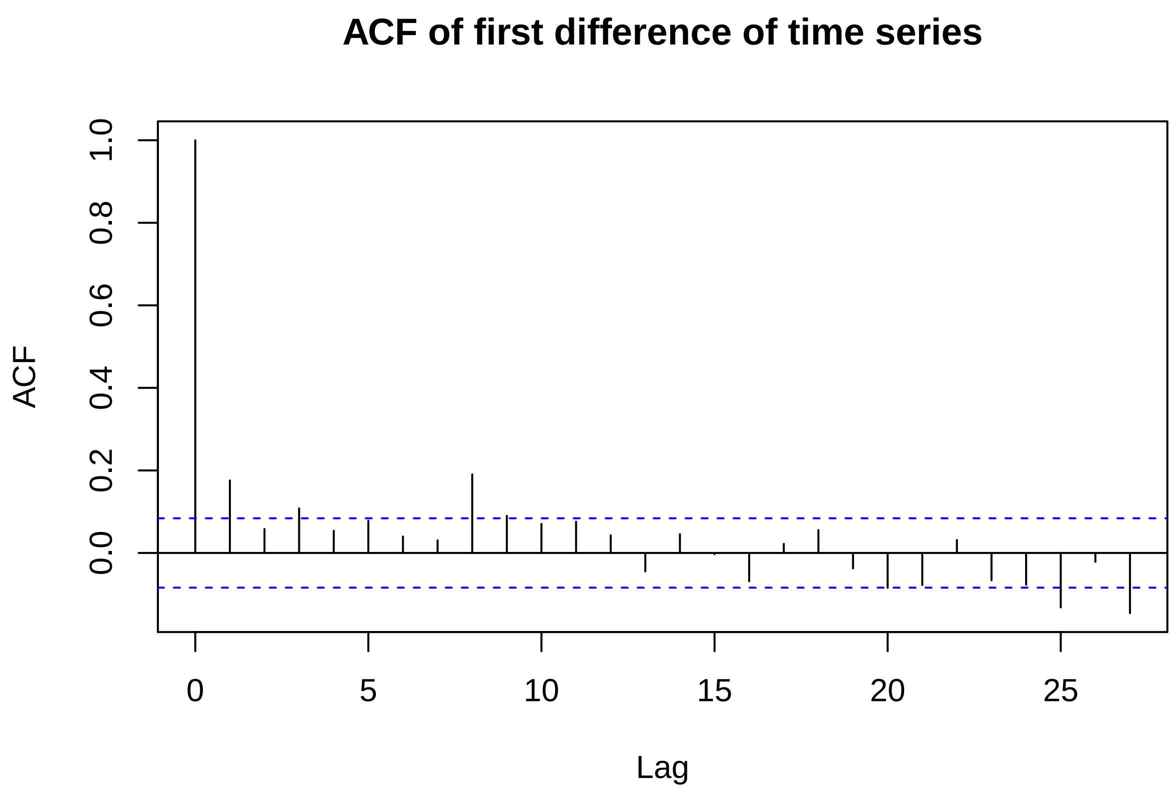
diff_ts	log_ts	diff_log_ts
0.01	0.2502782	0.01

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")
```



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

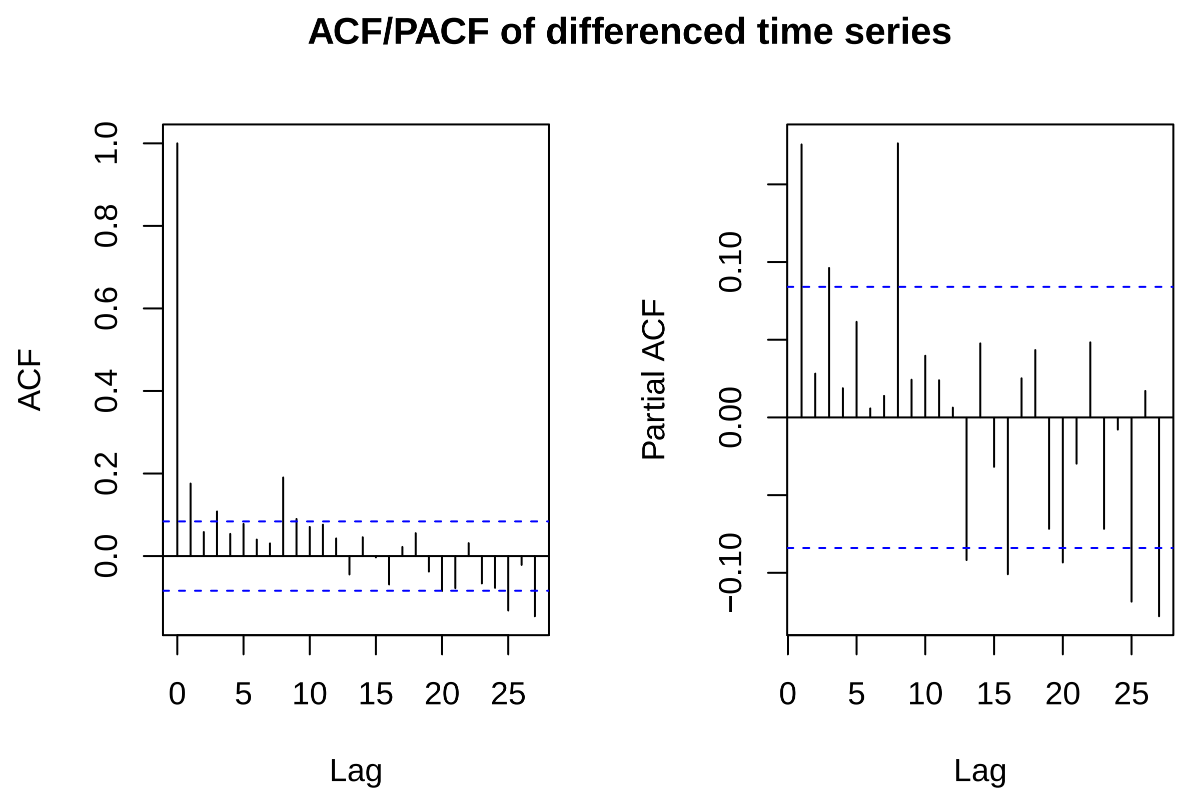


Both plots confirm that the time series looks much more stationary now. The AFC plot supports the suggestion about staionarity, because most of the peaks are inside the significance limits. For the further analysis, we will use this first difference of the original time series to perform ARIMA-modelling.

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.

```
# 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)
```




```r
# 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.

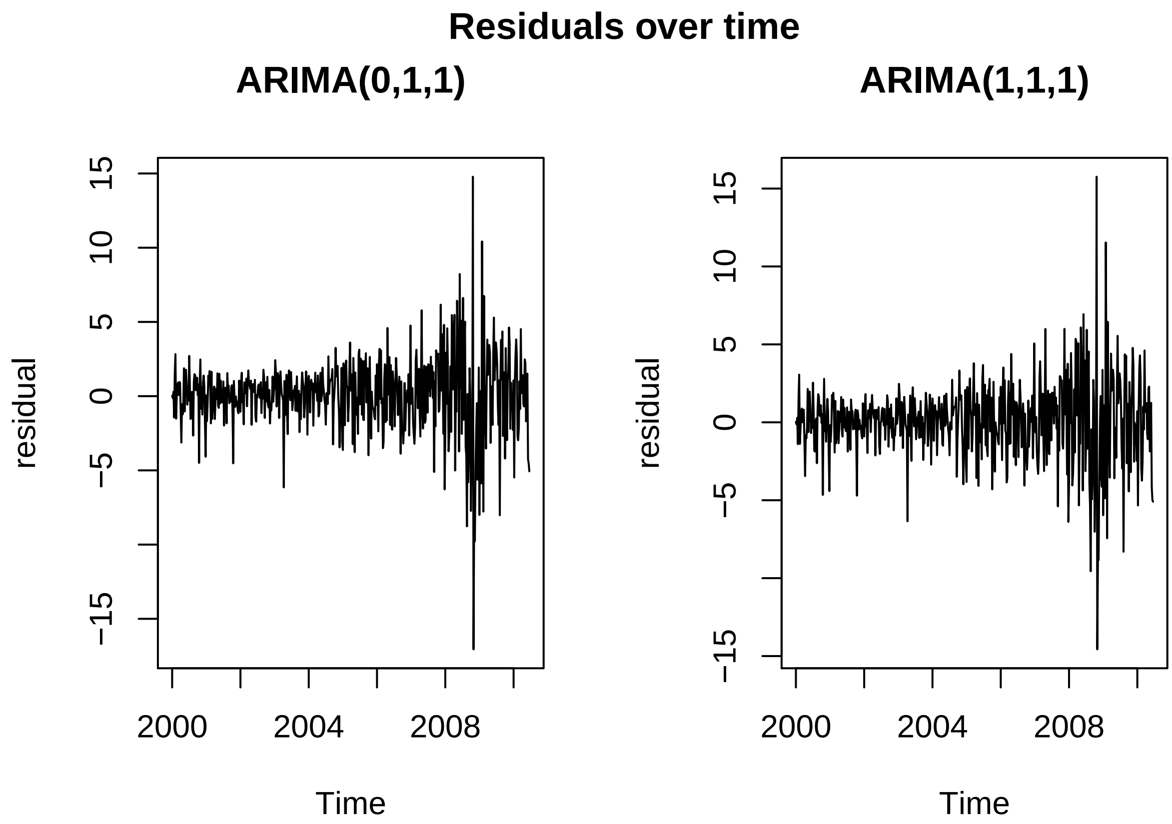
To fit the models, we will use the `forecast`-package. The other options (e.g. using only built-in R functions) are presented in assignment 1. We learned about the different methods to fit the time series (Method of moments, conditional least squares, maximum likelihood). Since no information is given, 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.

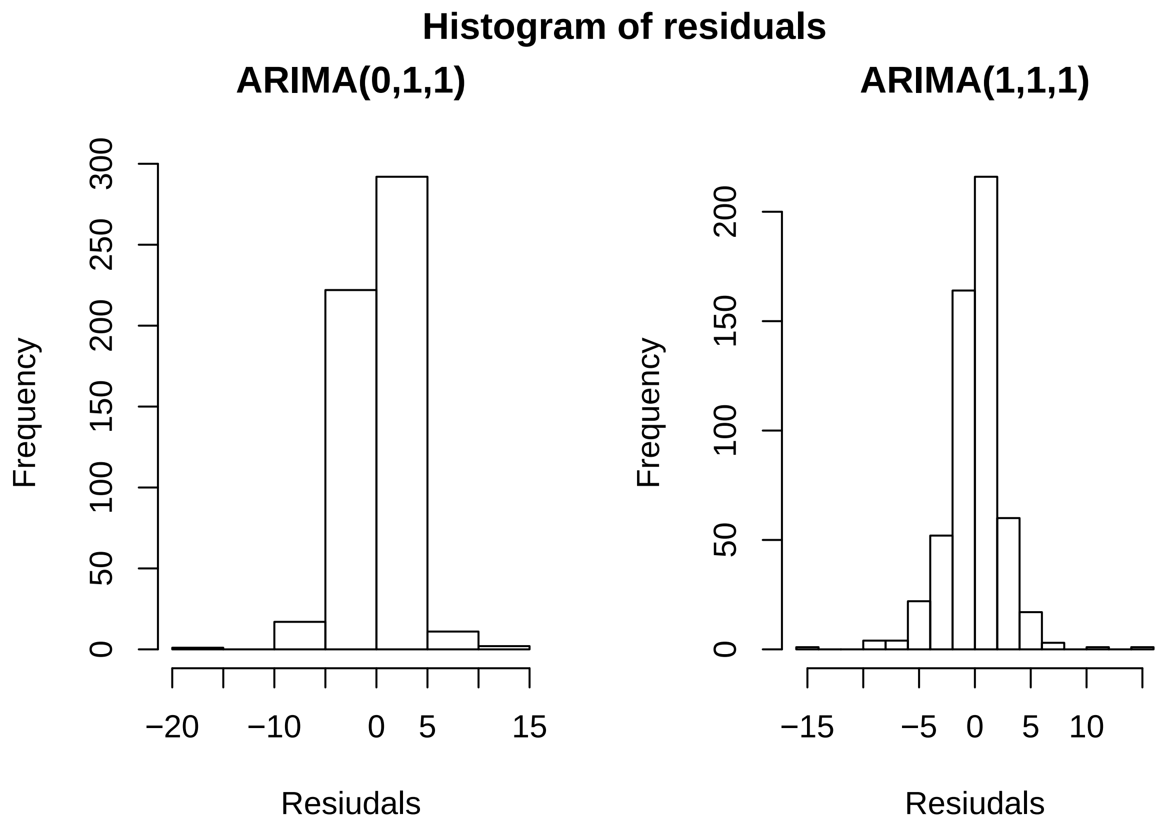
Since our goal is to decide for one of the two tentative models, we will perform residual analysis which will give us and idea about which model seems to be more promising.

```
# 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)
```



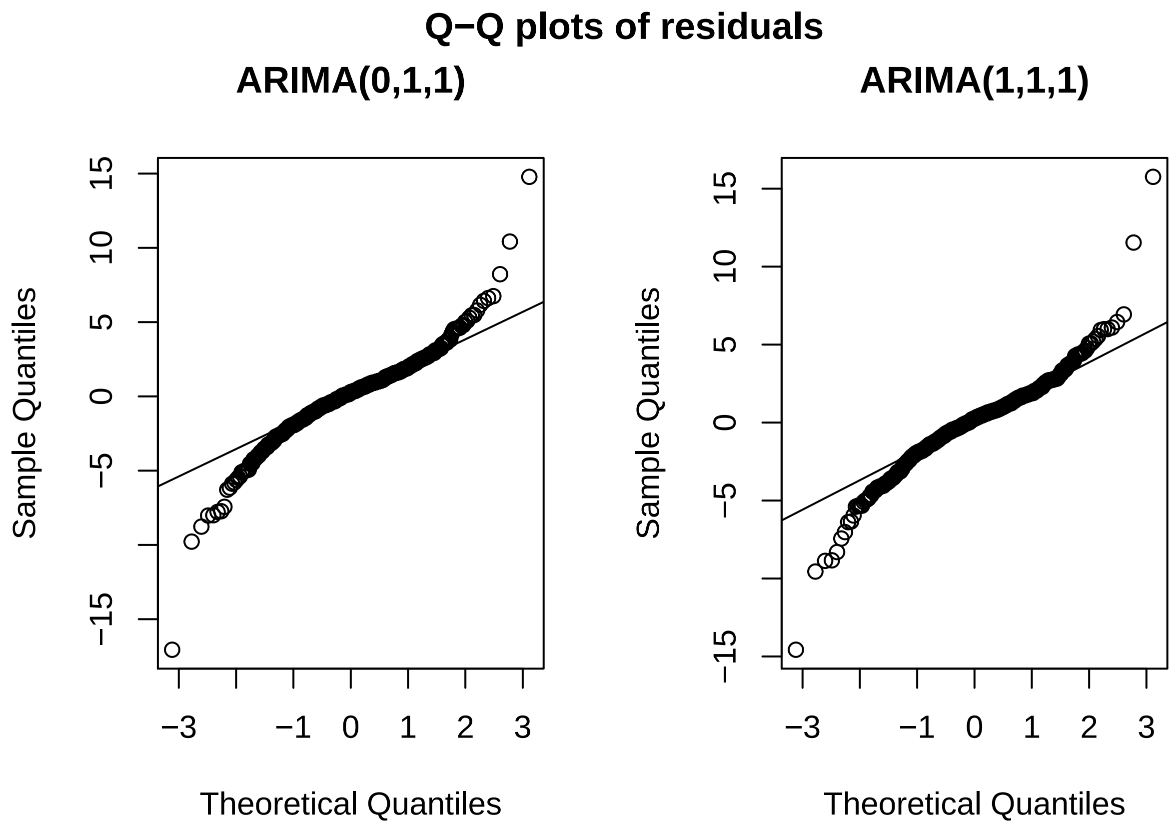
Since the residuals over time look random (white noise) for both models, it is an indication that both fitted models seem to be a promising choice. However, the plots do not return a better idea about which model seems to be the better choice.

```
# 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)
```



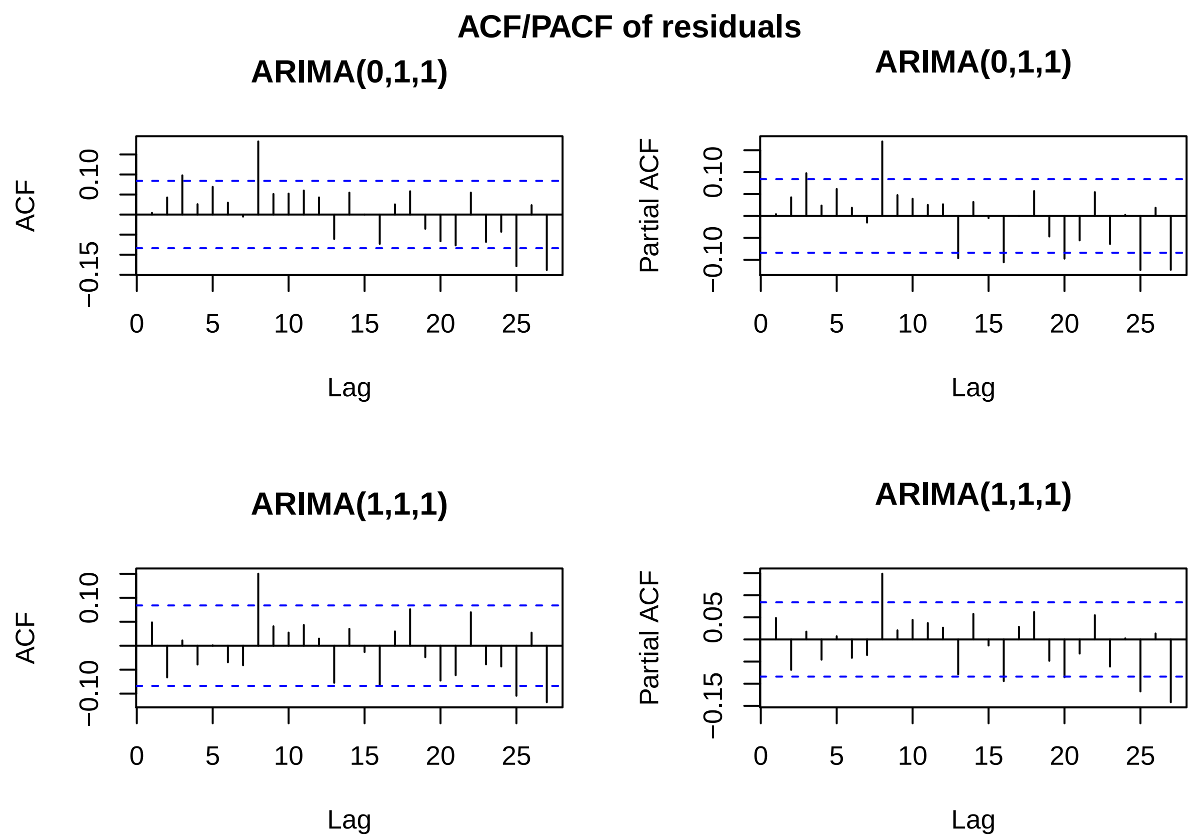
Both histograms reprent at least approximately a normal distribution.

```
# 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 7: 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 8: 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 9: 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:

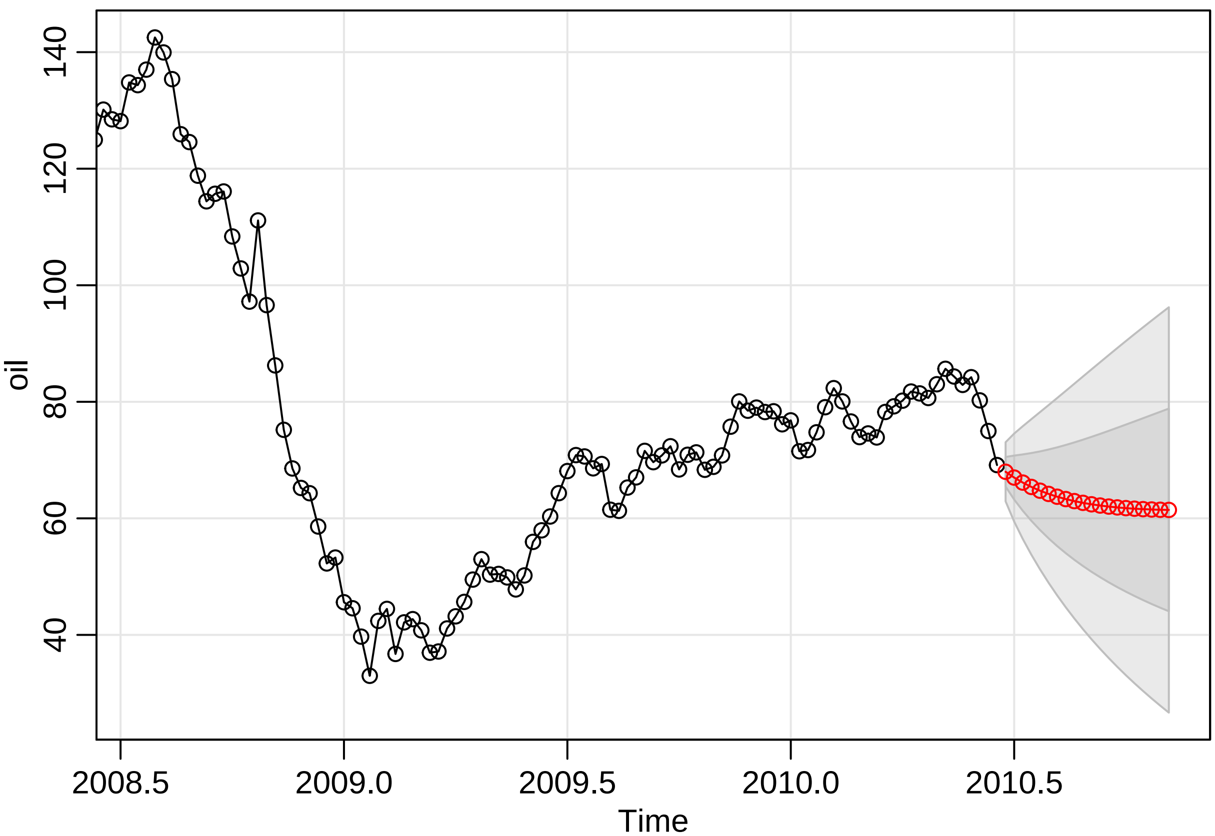
$$(1 - 0.8750497B)x_t = (1 - 0.7711252B)w_t$$

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.

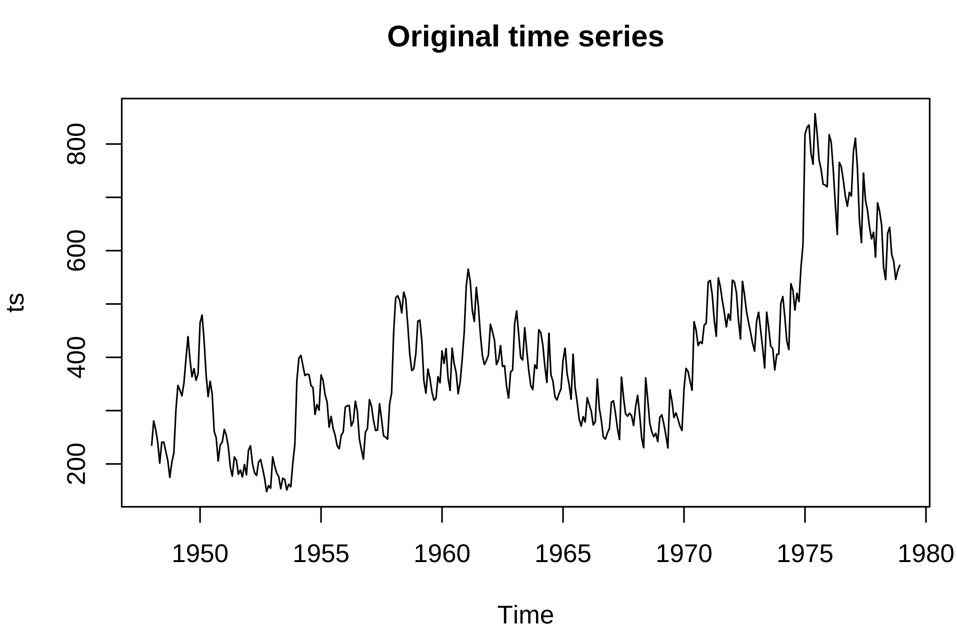
Like in assignment *3a*, we will repeat the same process. However, this time we need to find a suitable multicplicative seasonal ARIMA. The explanations for each step will be shortened this time, because we have explained the single steps already in assignment *3a* already.

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

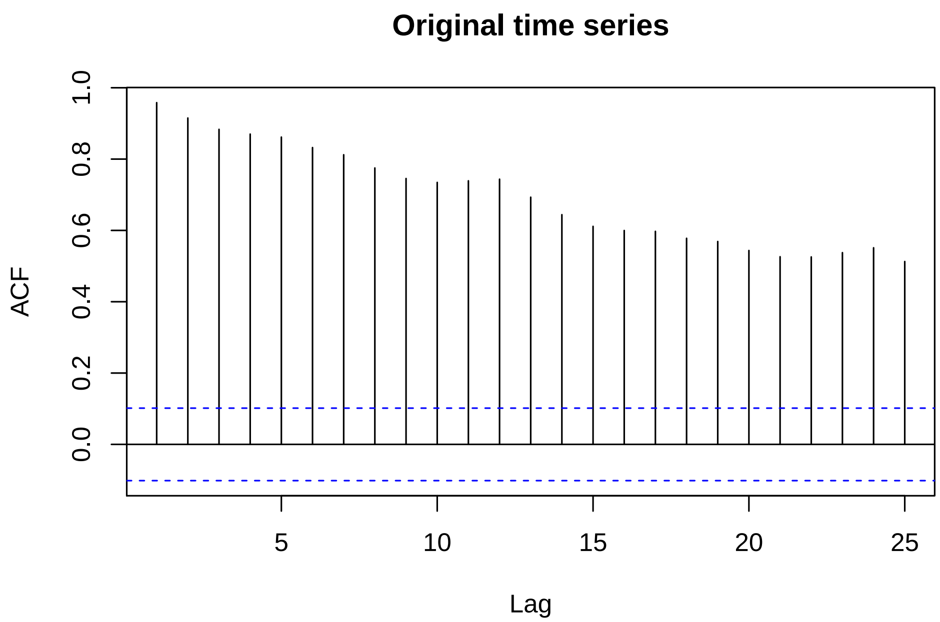
Analyzing original time series.

```
# 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

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



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

```r
# 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 α, we will analyze how detrending and transformation might lead to an even more stationary time series.

Making time series stationary.

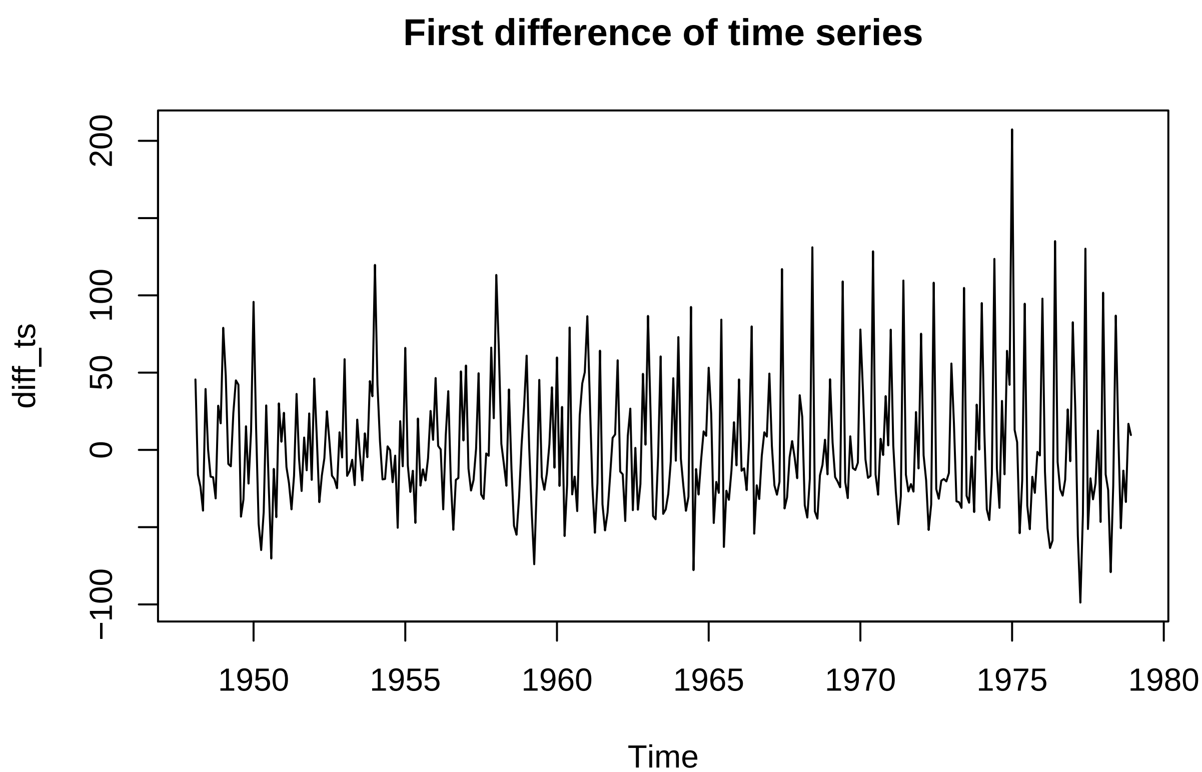
```r
# Differencing / Transforming original ts.
diff_ts = diff(ts)
log_ts = log(ts)
diff_log_ts = diff(log(ts))

# 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 10: 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")
```



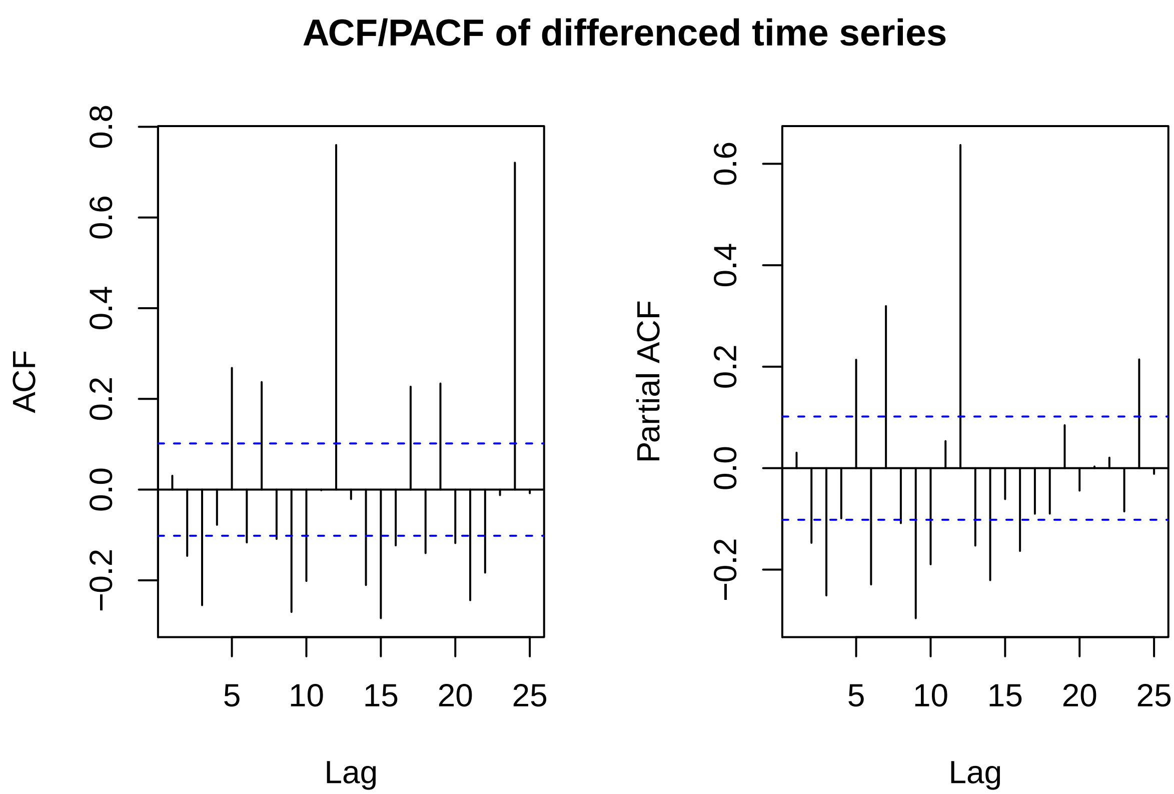
Again, `diff_ts` seems to be a good choice to work with.

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)
```

```
  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.

```
# 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 11: 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.

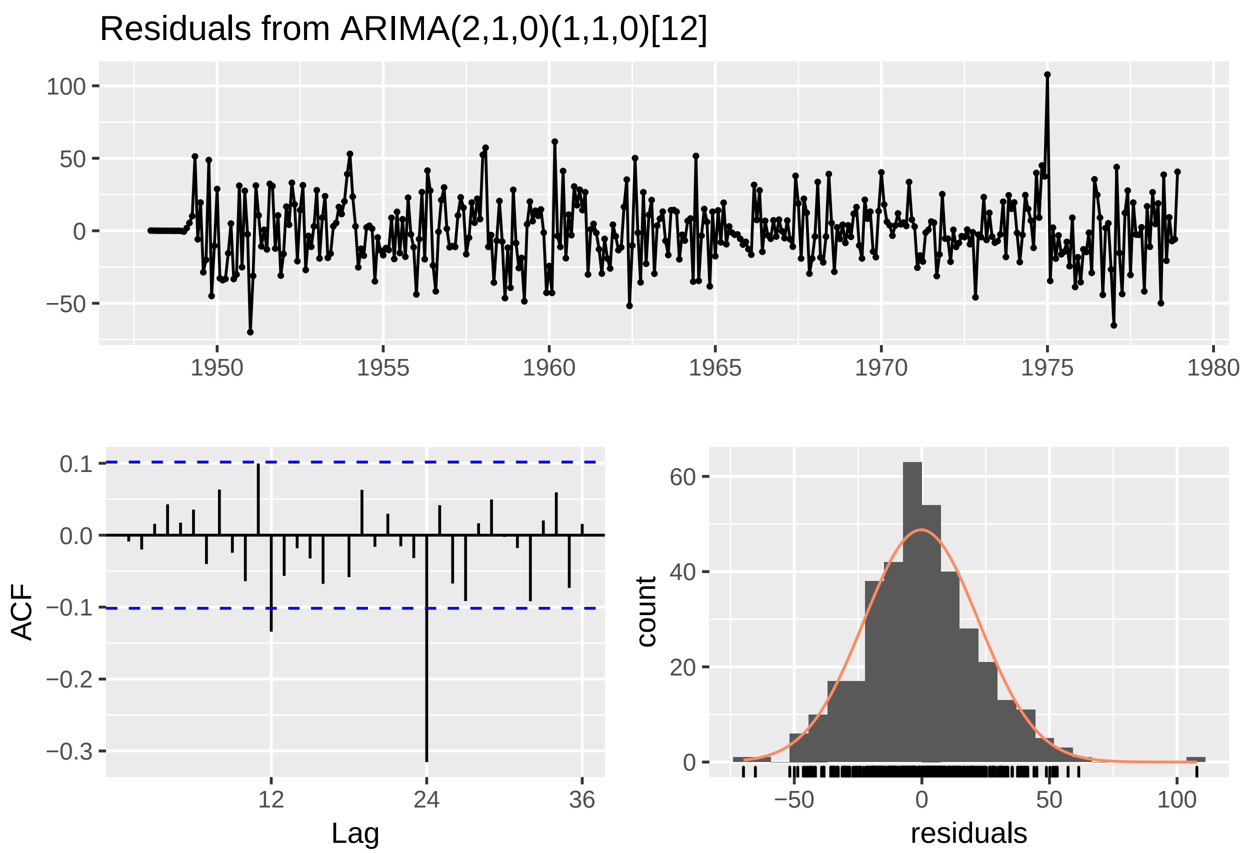
We will name the following fitted models after their non-seasonal part (e.g. `arima_2_1_0` for the $(2,1,0) \times (1,1,0)_{12}$).

```
# 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.

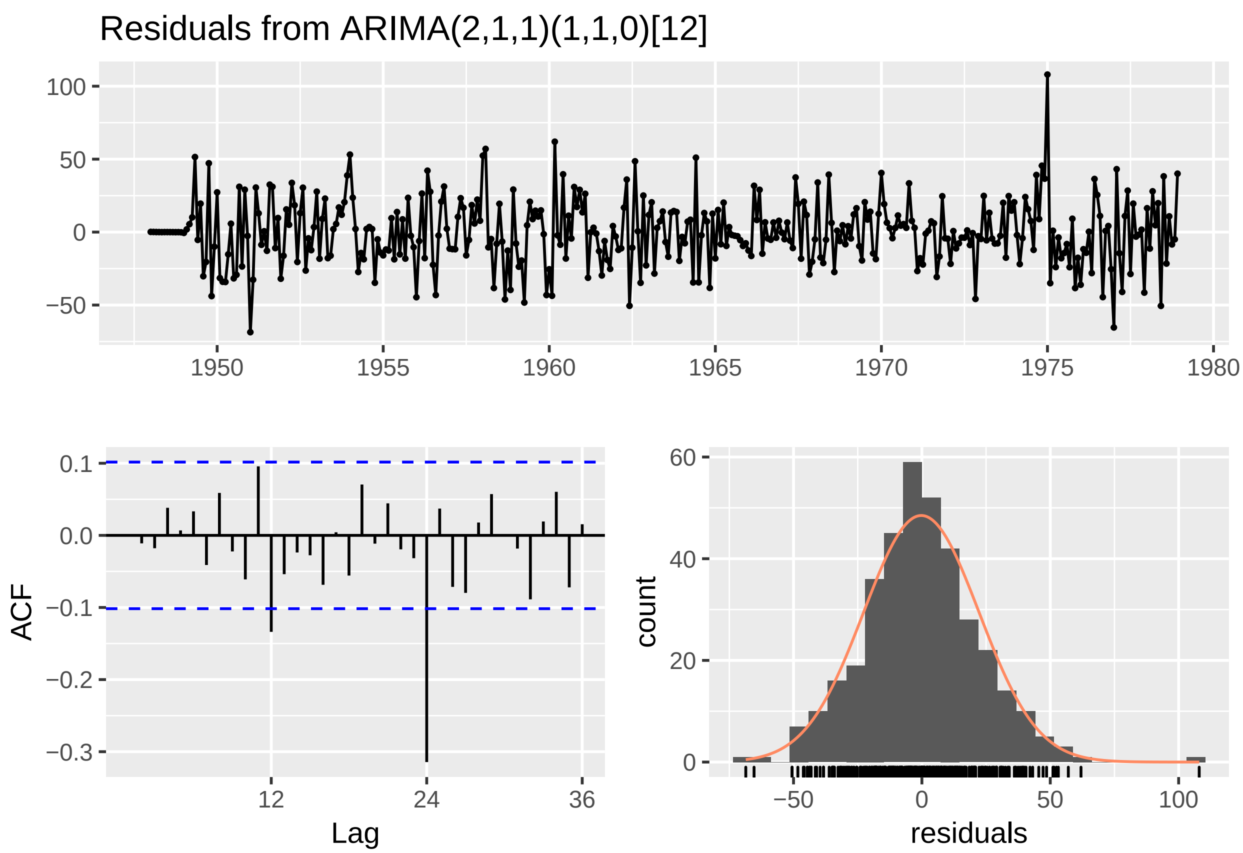
We will perform the residual analysis for both models. The AIC and BIC is already known (in both cases, the $(2,1,0) \times (1,1,0)_{12}$ is characterized by the smaller and therefore better values.)

```
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
```

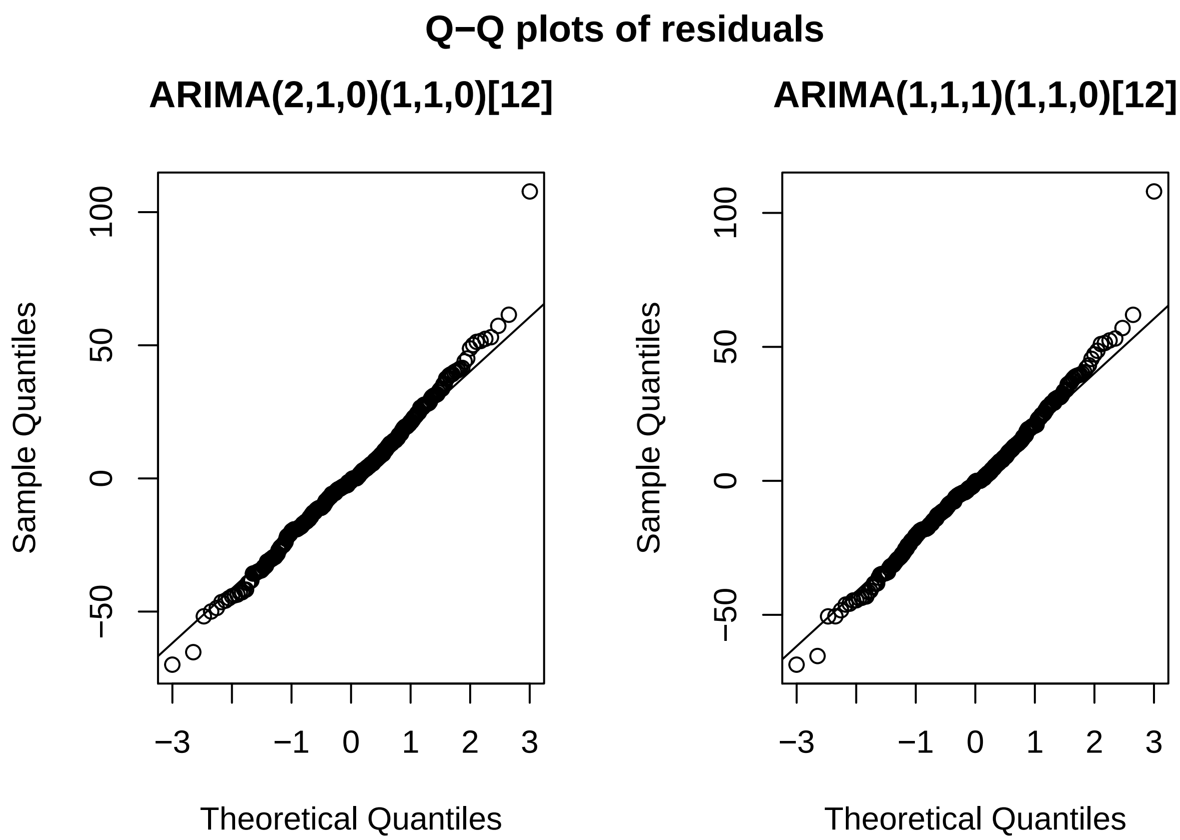
```r
checkresiduals(arima_2_1_1)
```




```
    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
```

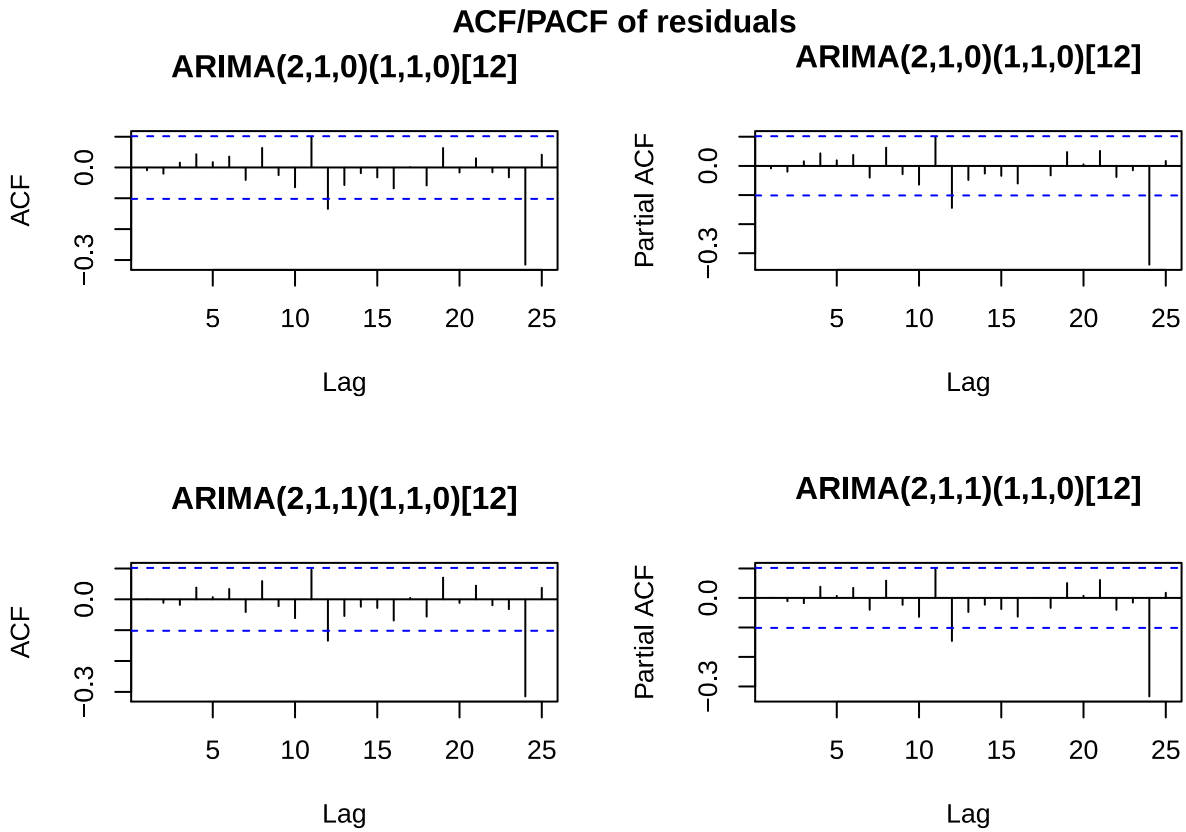
Since the residuals over time look random (white noise) for both models, it is an indication that both fitted models seem to be a promising choice. Looking at the ACF, only the seasonal lag (24) seems to have a significant autocorrelation value. Furthermore, both histograms are approximately normal. 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.

```
# 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 12: 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 $\alpha = 0.05$).

```r
# 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 13: 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:

```r
arima_2_1_0$coef
```

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

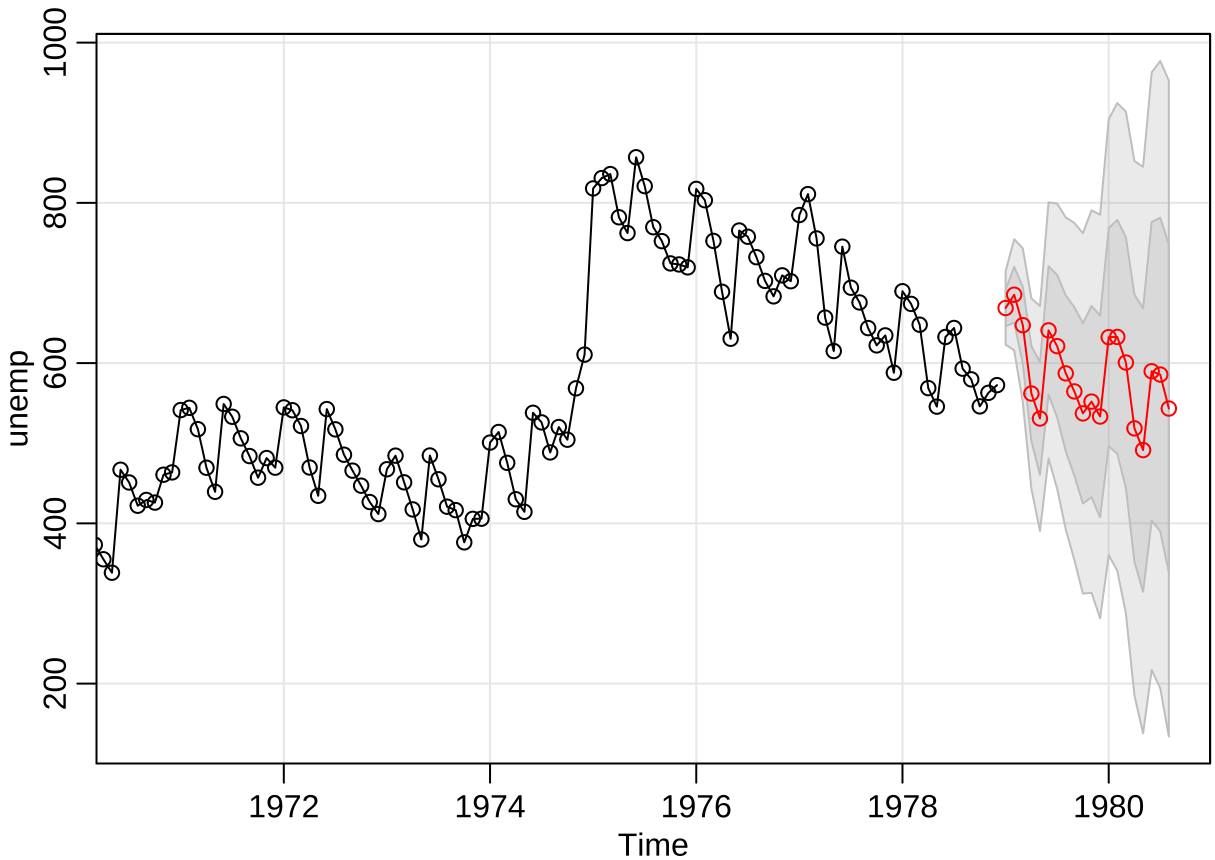
Therefore, we can write down the equation of the model like this:

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

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

Performing forecast using selected model.

```r
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
```