Time-Series Forecasting

Ralf Becker

14 March 2019

```r
library(tidyverse)
library(ggplot2)
library(pdfetch)
library(xts)
library(AER)            # access to HS robust standard errors
library(stargazer)
source("stargazer_HC.r")  # includes the robust regression display
source("stargazer_HAC.r")  # includes the Newey-West standard errors
library(gridExtra)  # required for the combination of ggplots
library(forecast)   # to use for forecasting
library(tsbox)
```

Import data and look at ACF

Let's download female unemployment rates and monthly inflation rates from the ONS database. On both occasions we put the

```r
# Download: Female unemployment rate (YCPL in database LMS)
ur_female <- pdfetch_ONS("YCPL","LMS")
names(ur_female) <- "Unemp Rate (female)"
periodicity(ur_female)
```

```
## Monthly periodicity from 1992-04-30 to 2018-12-31
```

```r
# keep all the data including 2018-Dec
# this was the last observation available at the time this was written
# remove this line if you want to use updated data
ur_female <- ur_female["/2018-12"]

ur_female_l <- data.frame(index(ur_female),stack(as.data.frame(coredata(ur_female))))
names(ur_female_l)[1] <- "Date"
names(ur_female_l)[2] <- "Value"
names(ur_female_l)[3] <- "id"

# Download: Inflation rate (D7OE in database MM23)
infl <- pdfetch_ONS("D7OE","MM23")
names(infl) <- "CPI Inflation"
periodicity(infl)
```

```
## Monthly periodicity from 1988-02-29 to 2019-02-28
```

```r
# keep all the data including 2019-Feb
# this was the last observation available at the time this was written
# remove this line if you want to use updated data
infl <- infl["/2019-2"]

infl_l <- data.frame(index(infl),stack(as.data.frame(coredata(infl))))
```

```r
names(infl_l)[1] <- "Date"
names(infl_l)[2] <- "Value"
names(infl_l)[3] <- "id"
```

Now we put both series into one dataframe by attaching the individual dataframes. We can do this as we gave identical names to the three columns in both dataframes. We mainly do these to use the ggplot function.
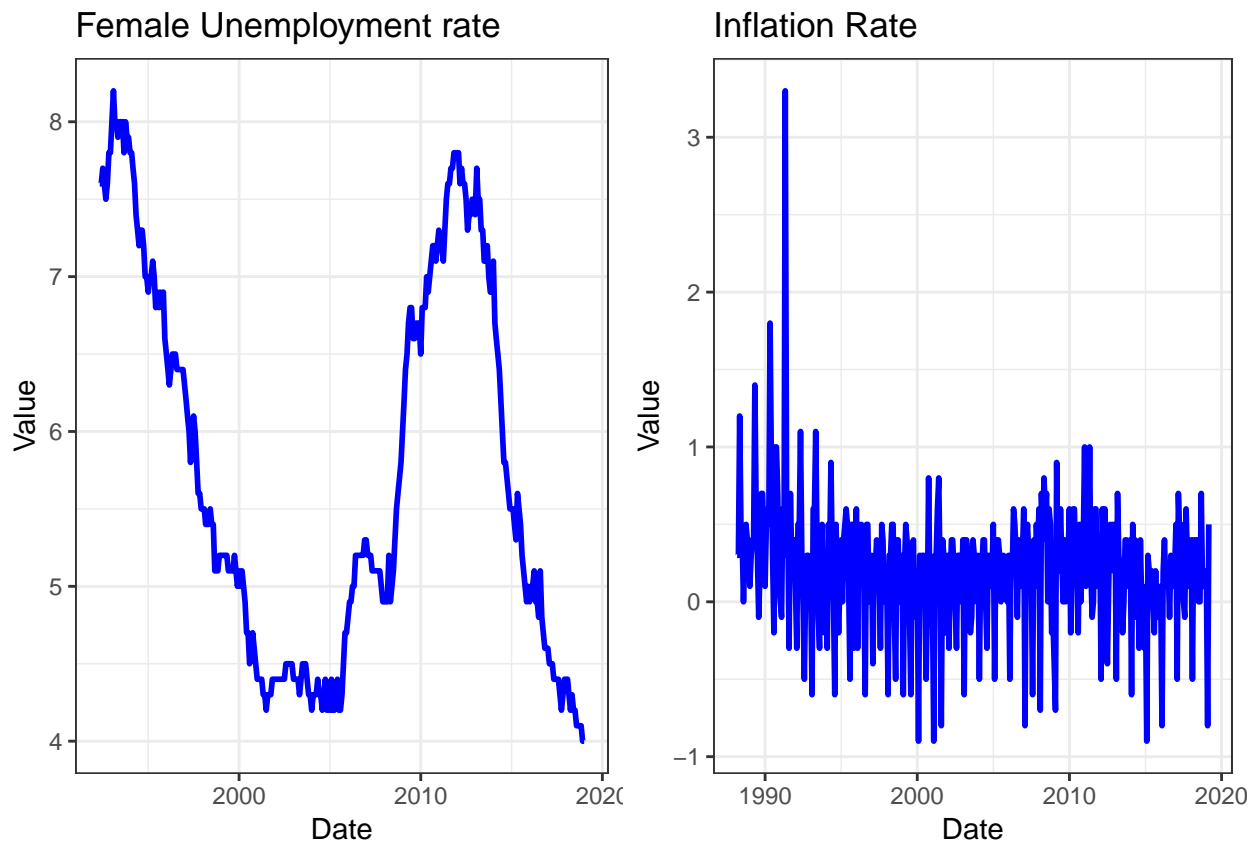
```r
data_l <- rbind(ur_female_l,infl_l)
```

Now we produce some time series plots

```r
p1 <- ggplot(data_l[data_l$id == "Unemp Rate (female)",],aes(x =Date, y=Value)) +
  geom_line(colour = "blue",size = 1.0) +
  ggtitle("Female Unemployment rate") +
  theme_bw()

p2 <- ggplot(data_l[data_l$id == "CPI Inflation",],aes(x =Date, y=Value)) +
  geom_line(colour = "blue",size = 1.0) +
  ggtitle("Inflation Rate") +
  theme_bw()

grid.arrange(p1, p2, nrow=1, ncol=2)
```

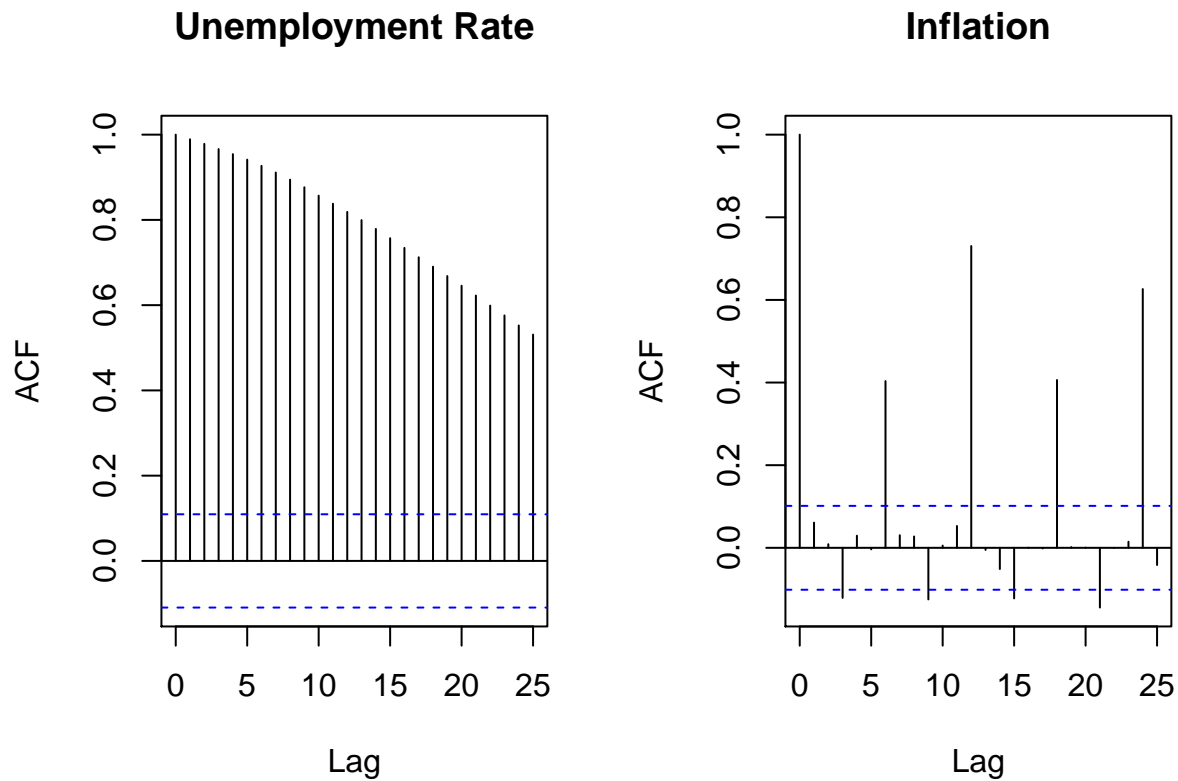

Let's look at the ACF.

```r
par(mfrow=c(1,2))

acf(ur_female,main = "Unemployment Rate")
```

```r
acf(infl, main = "Inflation")
```

Unemployment Rate   **Inflation**

When building forecasting models we will make use fo these ACF!

We know that the unemployment rate is nonstationary. Hence we shall (log) difference the series to produce a stationary series.

```r
g_ur_female <-diff(log(ur_female))
names(g_ur_female) <- "Growth in Unemp Rate (female)"
```

Data types

R has different data types which deal with time series. The `pdfetch` class of functions we used to obtain time-series delivered `xts` data types. This is a very flexible class of time-series data. However, for forecasting it turns out that it is more convenient to use the `ts` class. Fortunately it is fairly straightforward to translate from `xts` to `ts`.

```r
infl_ts <- ts(as.numeric(infl), start = c(1988,2),frequency = 12)
ur_ts <- ts(as.numeric(ur_female), start = c(1992,4), frequency = 12)
```

The `as.numeric()` function first strips the original series of its data info and then you re-introduce it via the `start` and `frequency` options of the `ts` function.

Seasonal pattern

The inflation rate has a seasonal component. We could see that from the time-series plot, but also from the ACF. It can be useful to make this more obvious with the `ggseasonalplot` function

```
ggseasonplot(infl_ts)
```
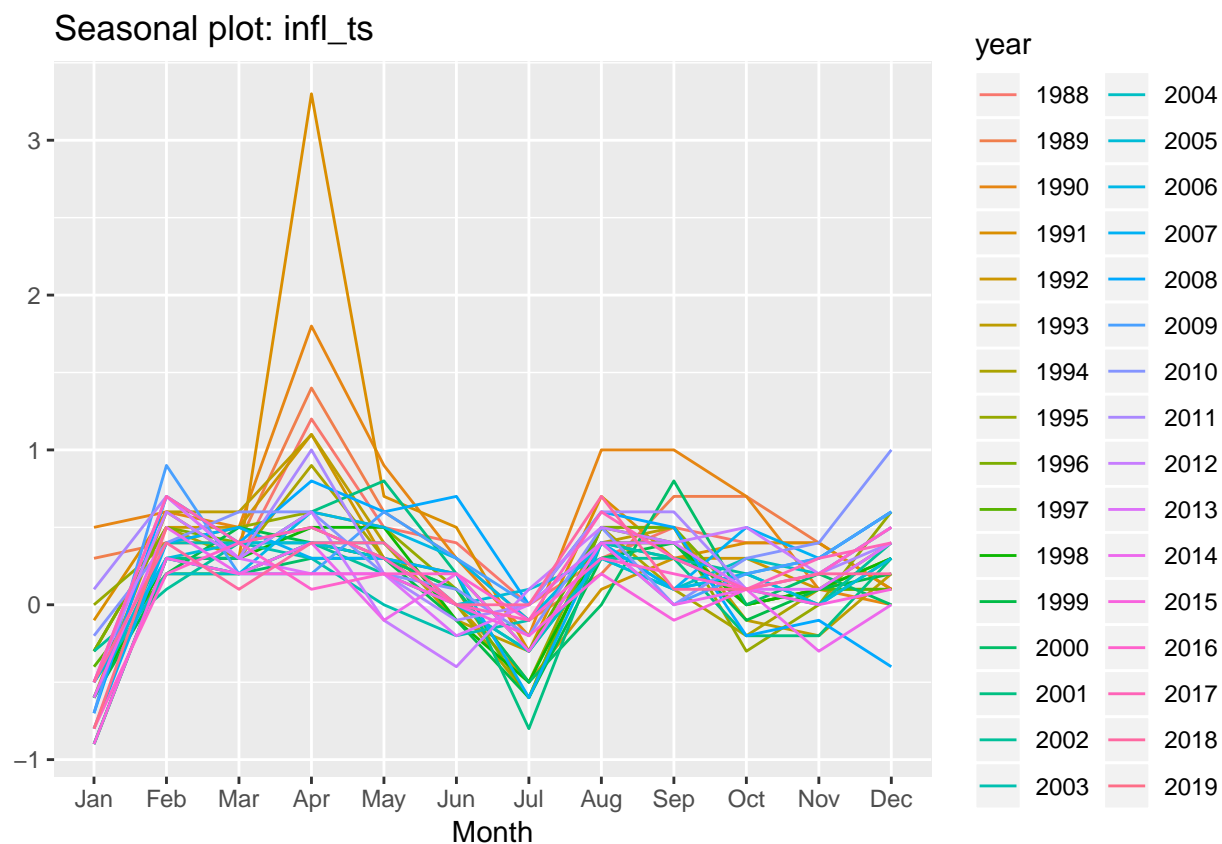

From here we can see that in the early years there was an April spike in the monthly inflation series, but that spike has been disappearing such that the only seasonal pattern remaining is a dip in inflation in January and July.

This information will lead us to excluding early years from our dataset when we estimate a model to forecast inflation.

For the unemployment rate there was no obvious seasonal pattern. Let's just confirm that seasonal plot does not reveal such a pattern.

```
ggseasonplot(ur_ts)
```

Seasonal plot: ur_ts
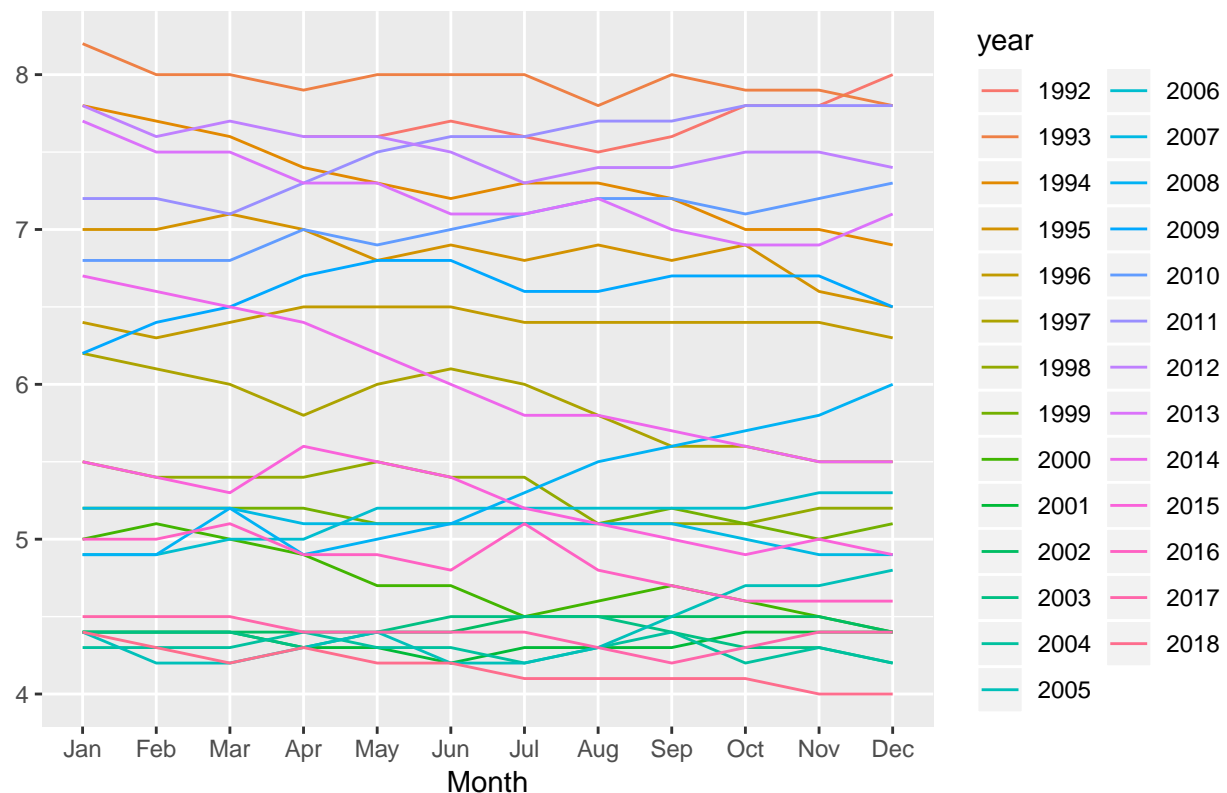
No, no seasonal pattern can be seen here.

Using AR models for forecasting

We will be using AR models for forecasting. We will rely on the `forecast` package authored by Rob Hyndman to do the heavy lifting.

Unemployment Rate

First we create a differenced series for the unemployment rate as we wish to model a stationary season.

```
dur_ts <- diff(ur_ts)
```

We could fit an AR(2) model. In order to estimate an AR(2) model we use the `arima` function as follows

```
fit_dur <- Arima(dur_ts,order = c(2,0,0))
```

It is via the `order` option that we determine the order of the forecast model. We feed in three values, the first one is the AR order (2), the second is the difference operator (here 0 as we don't need any further differencing as Δur_t is already stationary) and the third is the MA order (here 0). We didn't introduce any MA models and hence we set it to 0.

The object `fit_dur` now contains the information for the estimated model.

```
summary(fit_dur)
```

```
## Series: dur_ts
## ARIMA(2,0,0) with non-zero mean
##
## Coefficients:
##           ar1     ar2     mean
##       -0.0179  0.1715  -0.0112
## s.e.   0.0549  0.0550   0.0074
##
## sigma^2 estimated as 0.01279:  log likelihood=244.9
## AIC=-481.79   AICc=-481.67   BIC=-466.72
##
## Training set error measures:
##                        ME      RMSE        MAE MPE MAPE      MASE
## Training set -7.117335e-05 0.1125524 0.08556378 NaN  Inf 0.7259957
##                    ACF1
## Training set 0.002503311
```

Note that this estimates that an AR model which looks like this

$$(\Delta ur_t - m) = \alpha_1 \, (\Delta ur_t - m) + \alpha_2 \, (\Delta ur_t - m) + u_t$$

For all intends and purposes this is similar to this more traditional looking AR(2) model

$$\Delta ur_t = \beta_0 + \beta_1 \, \Delta ur_t + \beta_2 \, \Delta ur_t + u_t$$

The `mean` parameter in the model output is the sample estimate for (m). You can also see that it is the 2nd lag (the `ar2` coefficient) which is statistically significant.

If we want to forecast from this model we can do this as follows

```
for_dur <- forecast(fit_dur,h=10)
plot(for_dur)
```
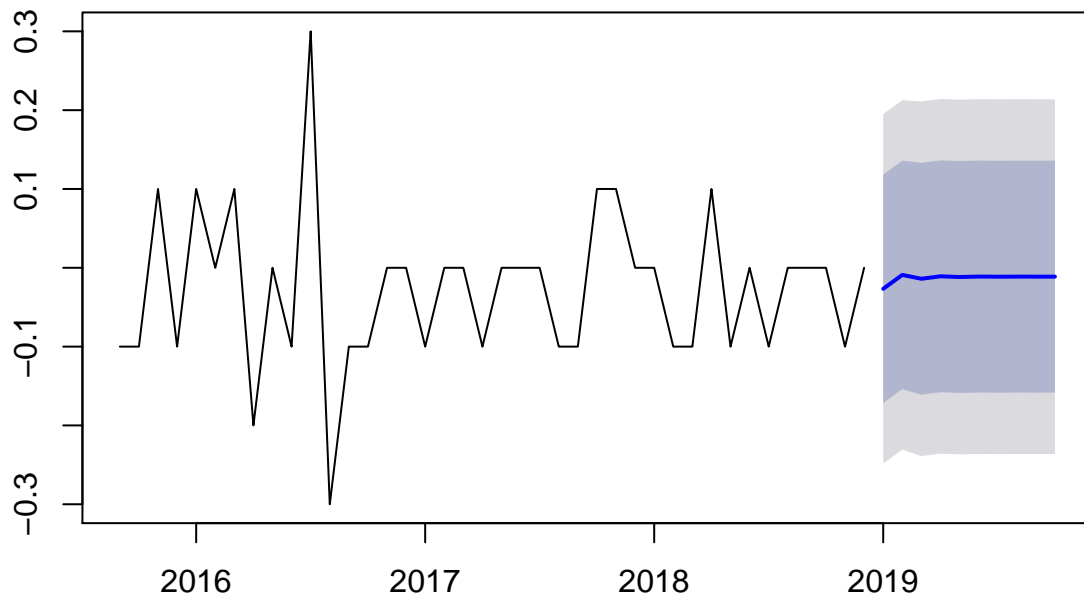
Forecasts from ARIMA(2,0,0) with non−zero mean
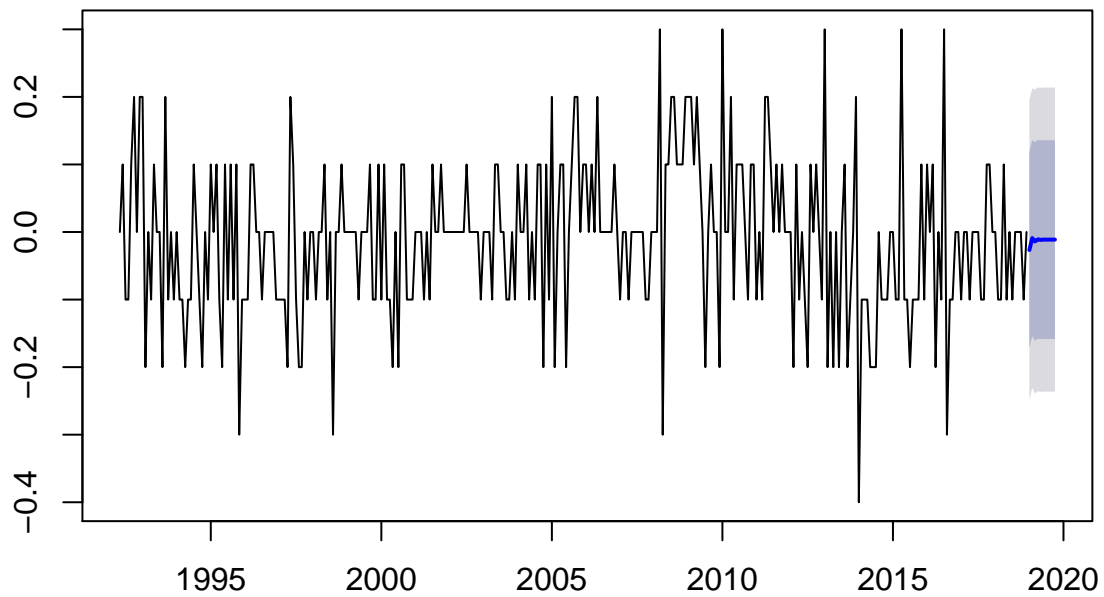
We would like to see a little more detail at the end of the data. We achieve this by using the `include` option in the plot function,

```
plot(for_dur, include = 40)
```

Forecasts from ARIMA(2,0,0) with non−zero mean

As we can see the forecast does not flatten out very quickly. Essentially it will move towards the estimated `mean` value, -0.011.

If you want to get the actual values which were forecast. Recall that the last observation available for the model estimation process was December 2018.

```
for_dur$mean
```

```
##              Jan         Feb         Mar         Apr         May
## 2019 -0.026646204 -0.009022347 -0.013906474 -0.010797111 -0.011690187
##              Jun         Jul         Aug         Sep         Oct
## 2019 -0.011141044 -0.011303999 -0.011206922 -0.011236599 -0.011219422
```

From Δur_t to ur_t

We estimated a model in differences (as we wanted to model a stationary series). But in the end we are interested in teh original series, the unemployment rate itself. We could now use the estimated changes to add them to the currrent level of the unemployment rate to get the forecast value for the next period.

Fortunately there is a slightly easier way to deal with this. Recall that when we called the estimation function we called `fit_dur <- Arima(dur_ts,order = c(2,d,0))`. The `d` value we set to 0 as we had handed in the differenced series. However we could have also handed in the `ur_ts` series and than told the `Arima` function to difference once. The advantage is that then R knows that we are actually interested in `ur_ts` the level of unemployment.

```
fit_ur <- Arima(ur_ts,order = c(2,1,0))
summary(fit_ur)
```

8

```
## Series: ur_ts
## ARIMA(2,1,0)
##
## Coefficients:
##          ar1     ar2
##       -0.0097  0.1797
## s.e.   0.0549  0.0549
##
## sigma^2 estimated as 0.01284:  log likelihood=243.78
## AIC=-481.56   AICc=-481.49   BIC=-470.26
##
## Training set error measures:
##                         ME      RMSE        MAE        MPE      MAPE
## Training set -0.009345469 0.1127695 0.08475281 -0.1830474 1.495837
##                   MASE          ACF1
## Training set 0.189498 -0.005455141
```

```
for_ur <- forecast(fit_ur, h =10)
plot(for_ur, include = 40)
```

Forecasts from ARIMA(2,1,0)
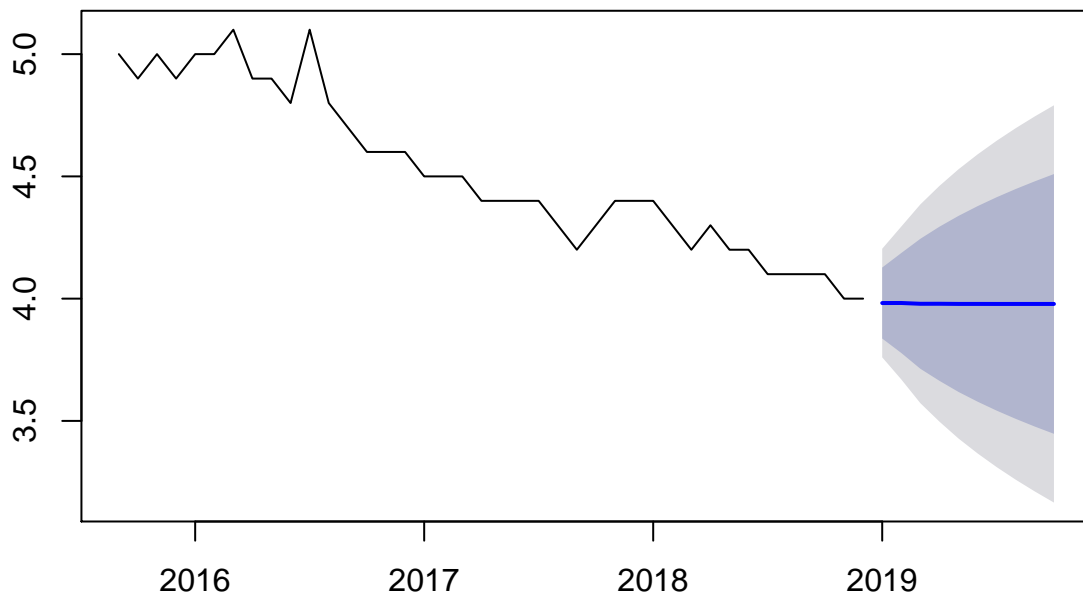
In essence you will get the same as if you were to aggregate up the forecast changes.
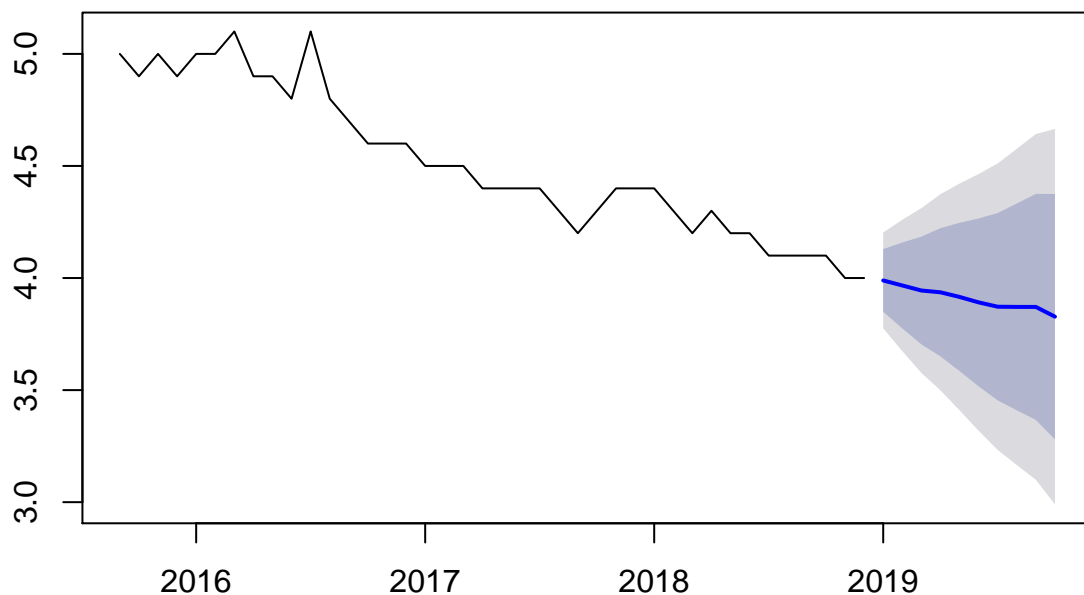
Using the `auto.arima` function

Above we decided from the outset that we wanted to estimate an AR(2) model for the changes. The `forecast` package allows you to let the software find the model which minimises an information criterion.

```
fit_ur_a <- auto.arima(ur_ts)
summary(fit_ur_a)
```

```
## Series: ur_ts
## ARIMA(2,1,1)(2,0,0)[12]
##
## Coefficients:
##          ar1     ar2      ma1     sar1     sar2
##       0.8127  0.1528  -0.8758  -0.0422  -0.2202
## s.e.  0.0695  0.0596   0.0464   0.0566   0.0563
##
## sigma^2 estimated as 0.01189:  log likelihood=256.83
## AIC=-501.66   AICc=-501.39   BIC=-479.05
##
## Training set error measures:
##                         ME      RMSE        MAE         MPE      MAPE
## Training set -0.004724813 0.1080236 0.08311991 -0.07270474 1.468983
##                  MASE         ACF1
## Training set 0.185847 0.003182505
```

```
for_ur_a <- forecast(fit_ur_a, h =10)
plot(for_ur_a, include = 40)
```

Forecasts from ARIMA(2,1,1)(2,0,0)[12]

It turns out that the `auto.arima` function picks a model which does have an AR(2) component, but also an MA(1) component. In addition to this it has also picked an extra seasonal component.

Forecast evaluation

If we waited for the unemployment rate in the next few periods to come in we could evaluate how well the forecasts fit the actual observations.

Instead we will turn back time by one year and pretend that it is the end of 2017 and we wanted to forecast the unemployment rate in 2018. Then we wil be able to compare our forecasts to the actual realisations.

So let's start by restricting the sample up to

```
ur_ts_17 <- window(ur_ts,end = c(2017,12))
```

Now we repeat the estimation process, both of the AR(2) model and then of the model automatically fitted by R.

```
fit_ur_17 <- Arima(ur_ts_17,order = c(2,1,0))
for_ur_17 <- forecast(fit_ur_17, h =10)
plot(for_ur_17, include = 40)
```

## Forecasts from ARIMA(2,1,0)
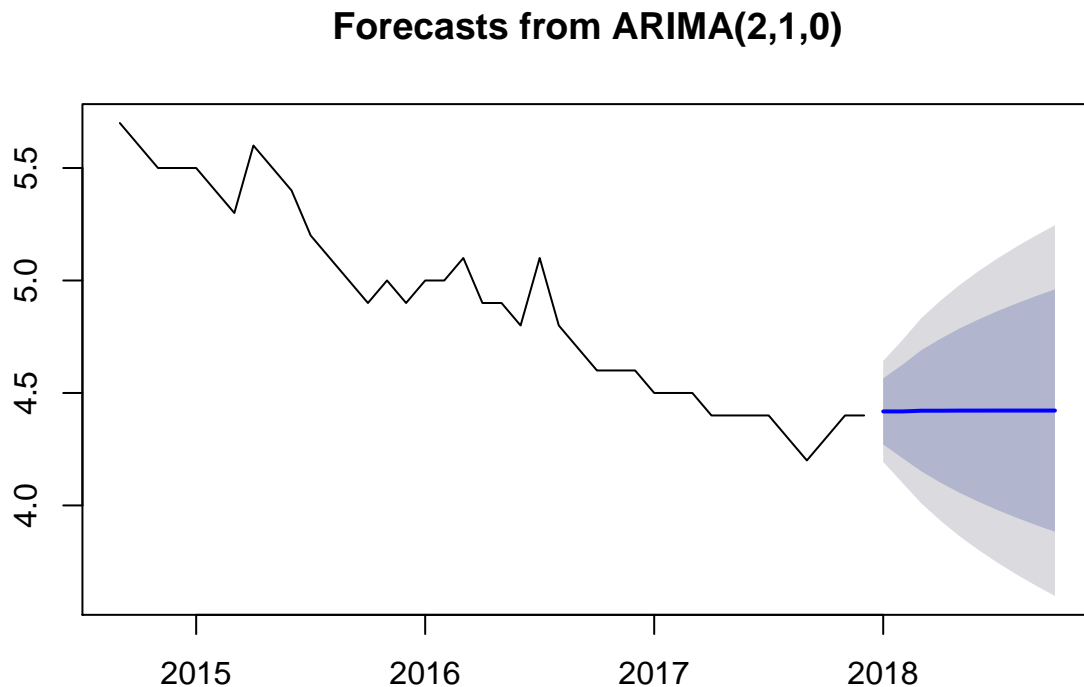

```
fit_ur_a_17 <- auto.arima(ur_ts_17)
summary(fit_ur_a_17)
```

```
## Series: ur_ts_17
## ARIMA(2,1,1)(2,0,0)[12]
```

```
## 
## Coefficients:
##          ar1     ar2      ma1     sar1     sar2
##        0.809  0.1526  -0.8700  -0.0358  -0.2152
## s.e.  0.073  0.0609   0.0508   0.0582   0.0592
## 
## sigma^2 estimated as 0.01227:  log likelihood=242.51
## AIC=-473.02   AICc=-472.74   BIC=-450.64
## 
## Training set error measures:
##                      ME      RMSE       MAE        MPE     MAPE
## Training set -0.004330286 0.1096765 0.08473188 -0.06255499 1.487238
##                   MASE       ACF1
## Training set 0.1857223 0.003385942
```

```r
for_ur_a_17 <- forecast(fit_ur_a_17, h =10)
plot(for_ur_a_17, include = 40)
```

Forecasts from ARIMA(2,1,1)(2,0,0)[12]
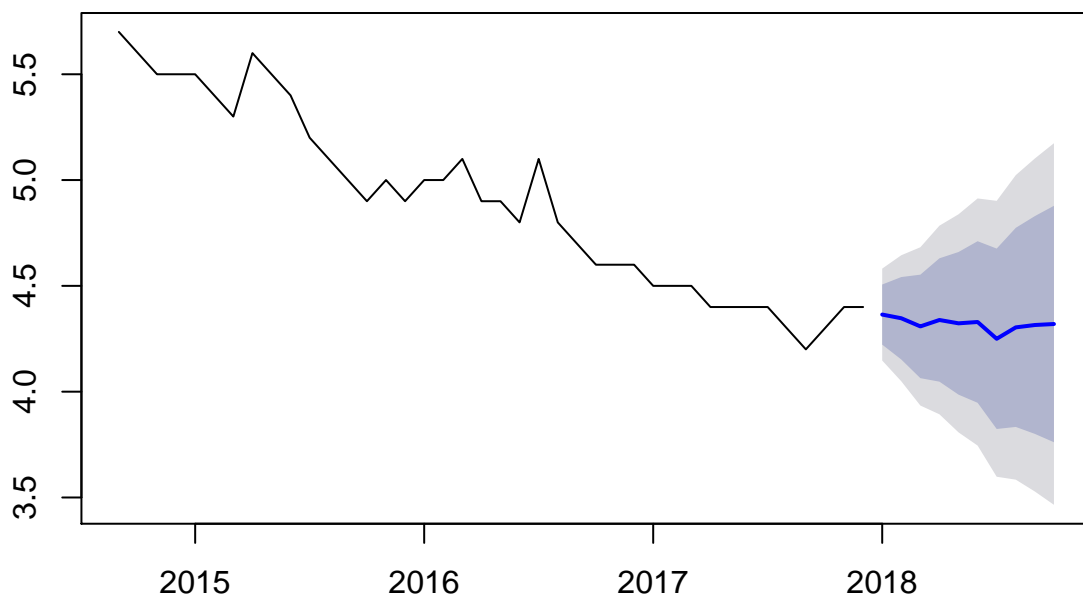
Now we use the `accuracy` function to evaluate how well these forecasts fit the actual observations in 2018.

```r
accuracy(for_ur_17,ur_ts)
```

```
##                      ME      RMSE       MAE        MPE     MAPE
## Training set -0.008496628 0.1141159 0.08595689 -0.1609257 1.504272
## Test set     -0.220806068 0.2428940 0.22080607 -5.3168441 5.316844
##                   MASE        ACF1 Theil's U
## Training set 0.1884074 -0.004431789        NA
## Test set     0.4839808  0.501587650  3.526255
```

The measures of fit are presented for the observations in the "in-sample" period (training set) and for the forecast period ("out-of-sample" or Test Set). You can see a very typical pattern, the model fits the in-sample data better than the out-of-sample data.

But how does this model compare to the model which is automatically chosen by the `auto.arima` function?

```
accuracy(for_ur_a_17,ur_ts)
```

```
##                          ME       RMSE        MAE         MPE     MAPE
## Training set -0.004330286 0.1096765 0.08473188 -0.06255499 1.487238
## Test set     -0.119951808 0.1439328 0.12706386 -2.90089657 3.062534
##                   MASE        ACF1 Theil's U
## Training set 0.1857223 0.003385942        NA
## Test set     0.2785090 0.524312856  2.093382
```

The automatically chosen model does improve on the forecast performance.