Time-Series Forecasting

Ralf Becker

01 May 2022

```
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. `pdfetch_ONS` requires an id for the series and the database where to find that series. You will have to identify these two pieces of information from the ONS website. Type "Female Unemployment Rate" into the search window.

On both occasions we put the data into a data frame with three columns ("Date", "Value" and "id") where the latter tells us what the variable in question is.

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

```
## Monthly periodicity from 1992-04-30 to 2022-01-31
```

```
# keep all the data including 2022-Jan
# 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["/2022-1"]

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 2022-03-31
```

```
# keep all the data including 2022-Mar
# this was the last observation available at the time this was written
# remove this line if you want to use updated data
infl <- infl["/2022-3"]

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

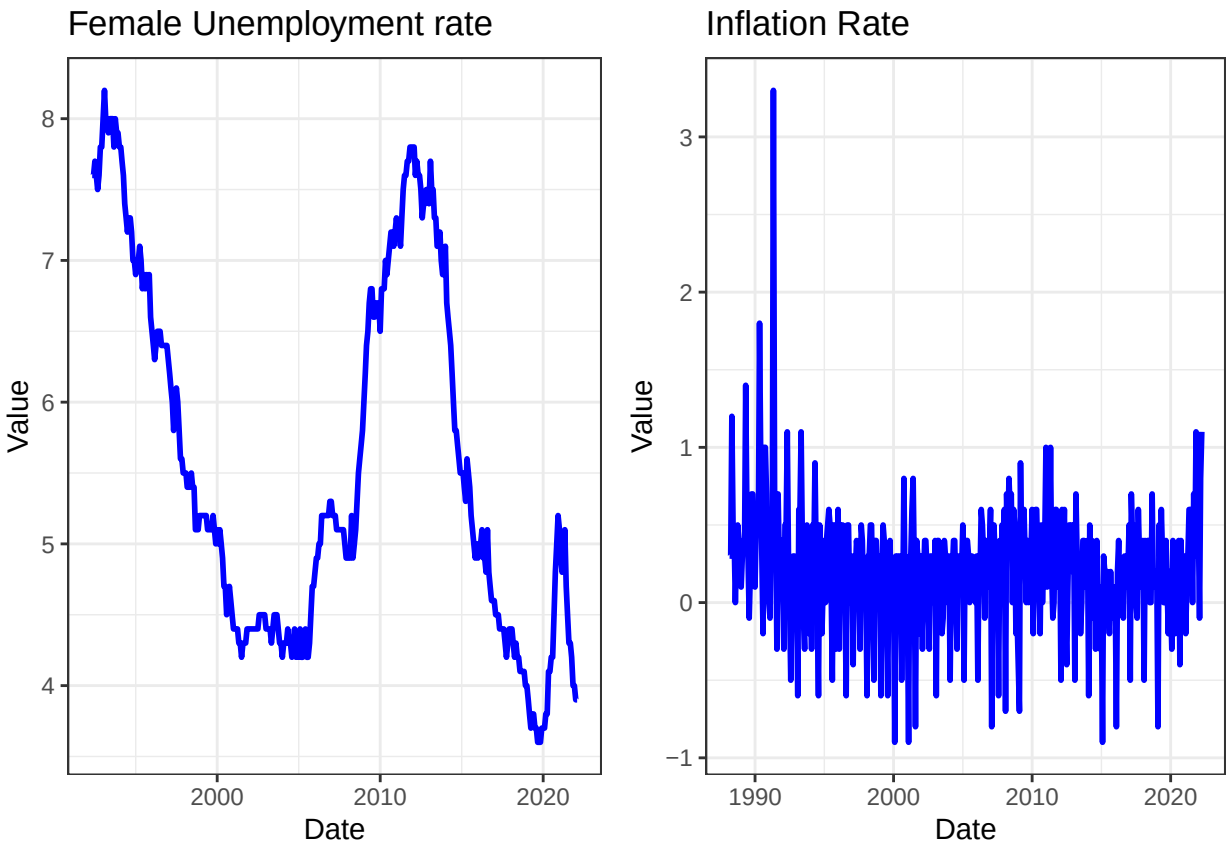
Now we put both series into one dataframe by attaching the individual dataframes. We can do this as we gave identical names to the three columns in both dataframes. We mainly do these to use the ggplot function.

```
data_l <- rbind(ur_female_l,infl_l)
```

Have a look at `data_l` to understand the structure of this dataframe. All the actual data are in the "Value" column (`data_l$Value`) while the "id column" (`data_l$id`) identifies the variable. We call this a long data structure (as opposed to the wide version which would have one column for each variable). This is mainly useful for using the `ggplot` function to produce nice plots.

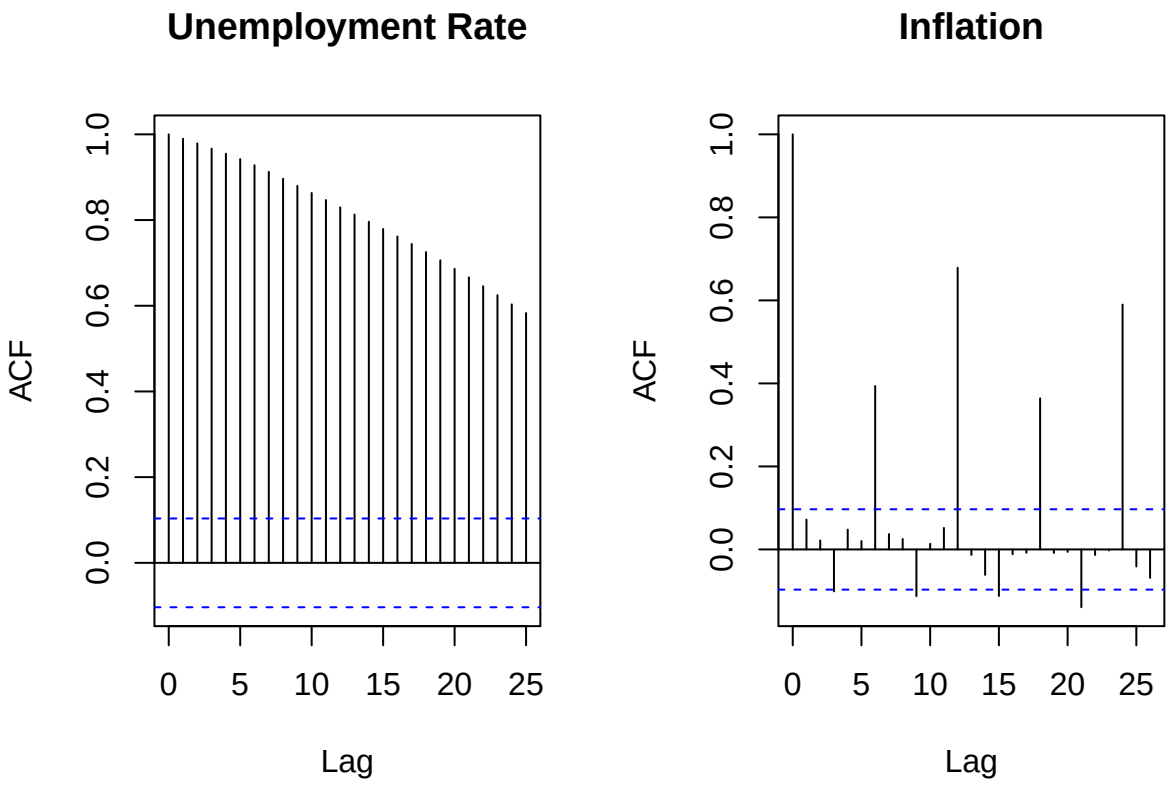
Now we produce some time series plots

```
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.

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

acf(ur_female,main = "Unemployment Rate")
acf(infl, main = "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.

```
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`.

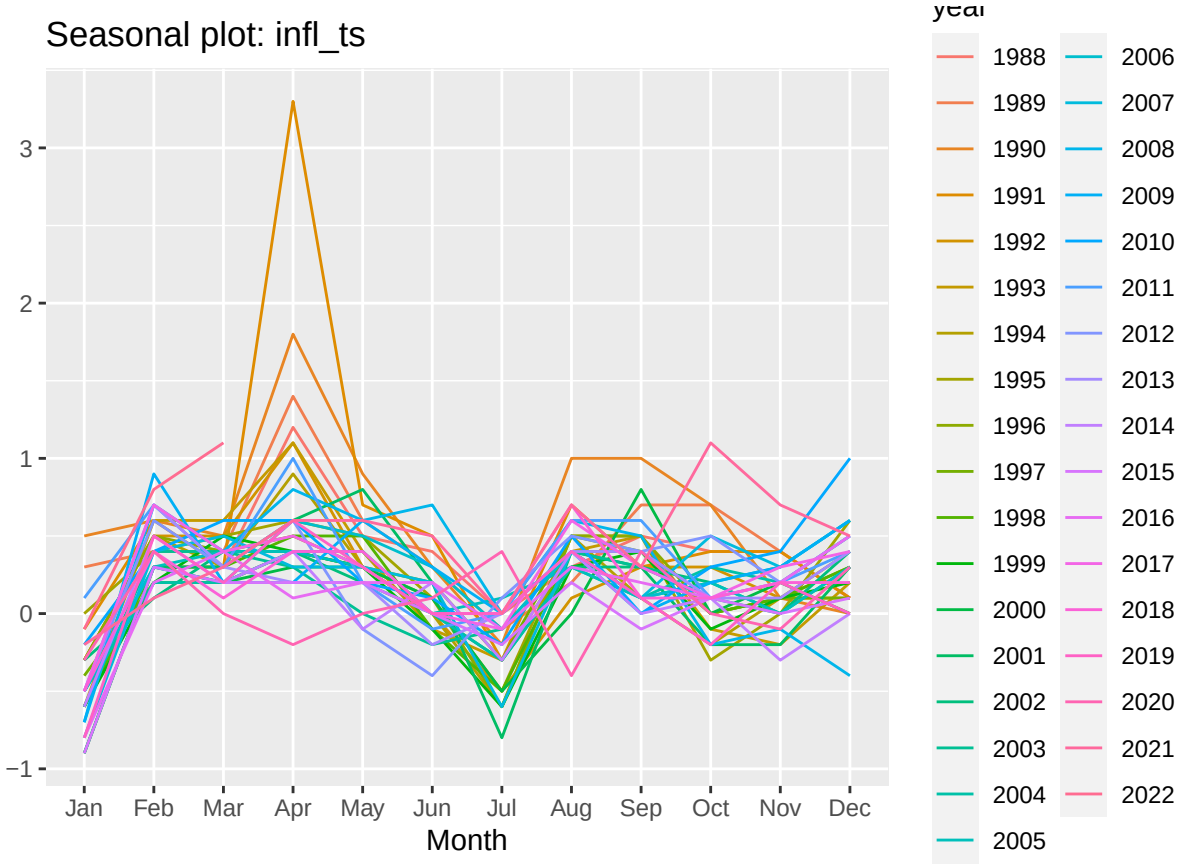
```
infl_ts <- ts(as.numeric(infl), start = c(1988,2),frequency = 12)
ur_ts <- ts(as.numeric(ur_female), start = c(1992,4), frequency = 12)
```

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)
```

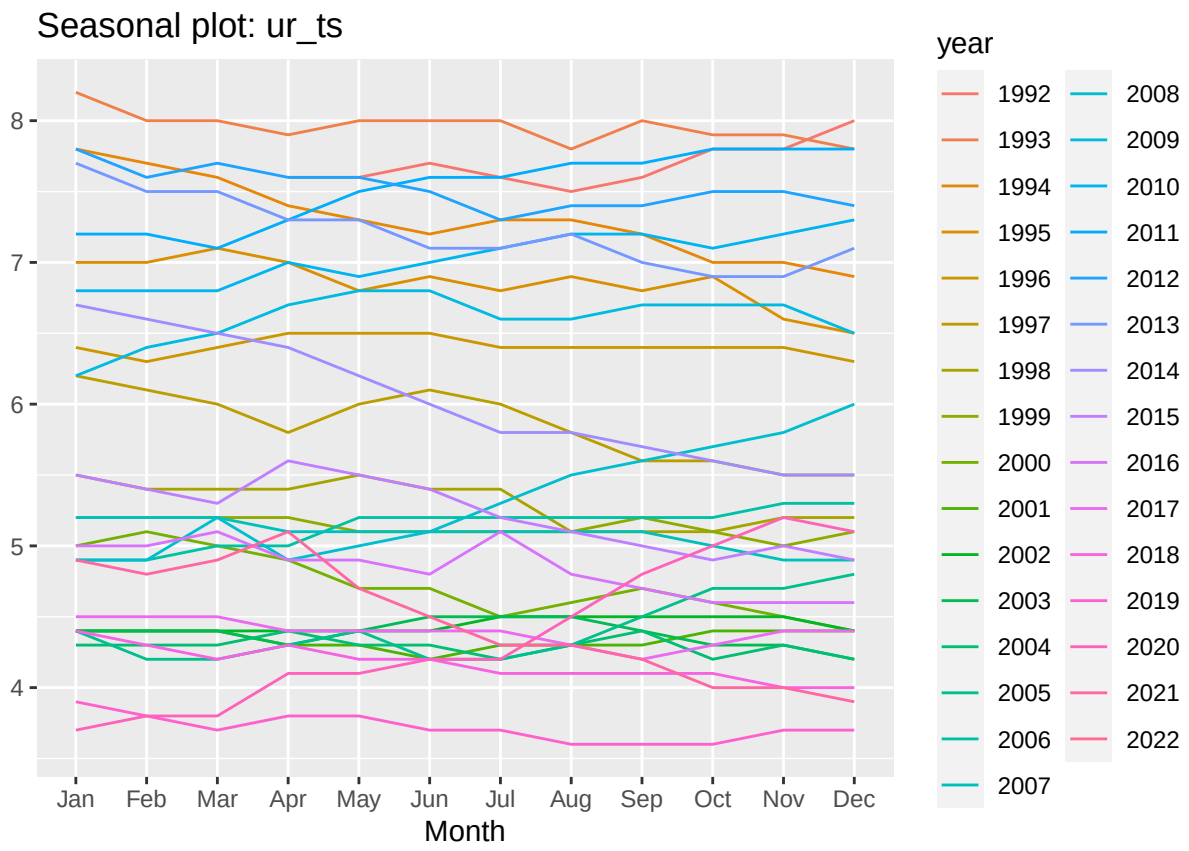


From here we can see that in the early years there was an April spike in the monthly inflation series, but that spike has been disappearing such that the only seasonal pattern remaining is a dip in inflation in January and July.

On the basis of this information we decide to exclude early years from our dataset when we use this to forecast inflation. If didn't do this the model would partially attempt to emulate this April spike, but we know that this is an old phenomenon.

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)
```



Indeed, 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
##       0.0329  0.1756  -0.0103
## s.e.  0.0520  0.0520   0.0078
##
## sigma^2 = 0.01375:  log likelihood = 260.09
## AIC=-512.18   AICc=-512.06   BIC=-496.66
##
## Training set error measures:
##                         ME     RMSE        MAE MPE MAPE      MASE        ACF1
## Training set -6.670941e-05 0.116768 0.08790611 NaN  Inf 0.7238093 0.005235681
```

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

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

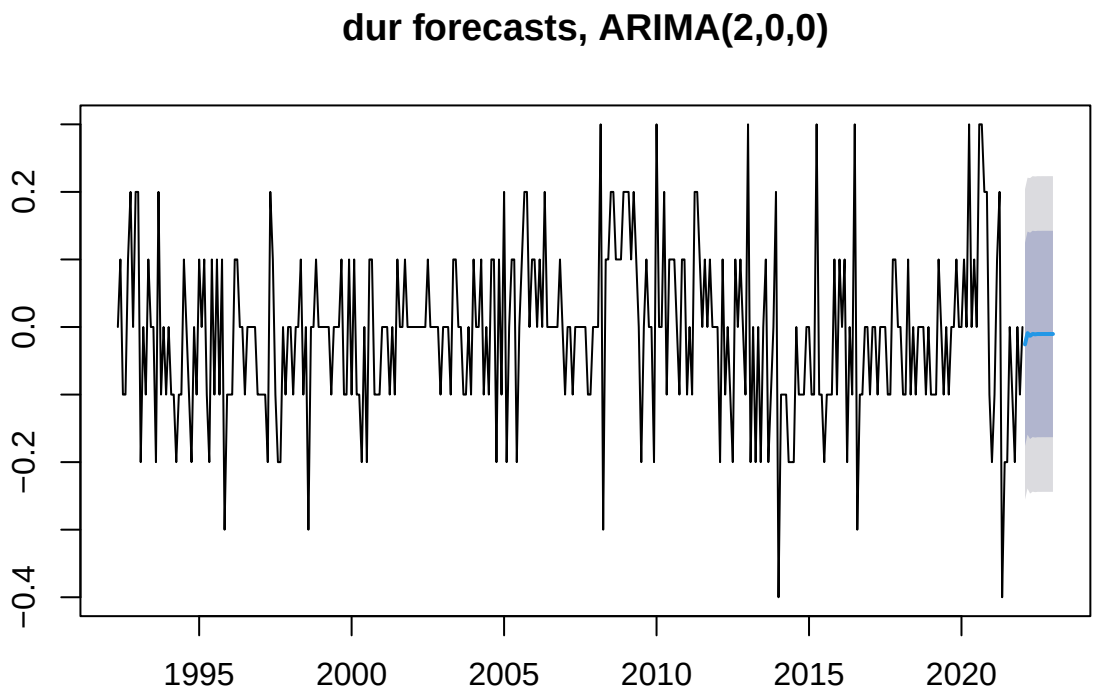
For all intends and purposes this is similar to this more traditional looking AR(2) model

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

The `mean` parameter in the model output is the sample estimate for m and it represents the value to which the series is predicted to return in the long-run (recall we are assuming a stationary model where such a value makes sense). You can also see that it is the 2nd lag (the `ar2` coefficient) which is statistically significant.

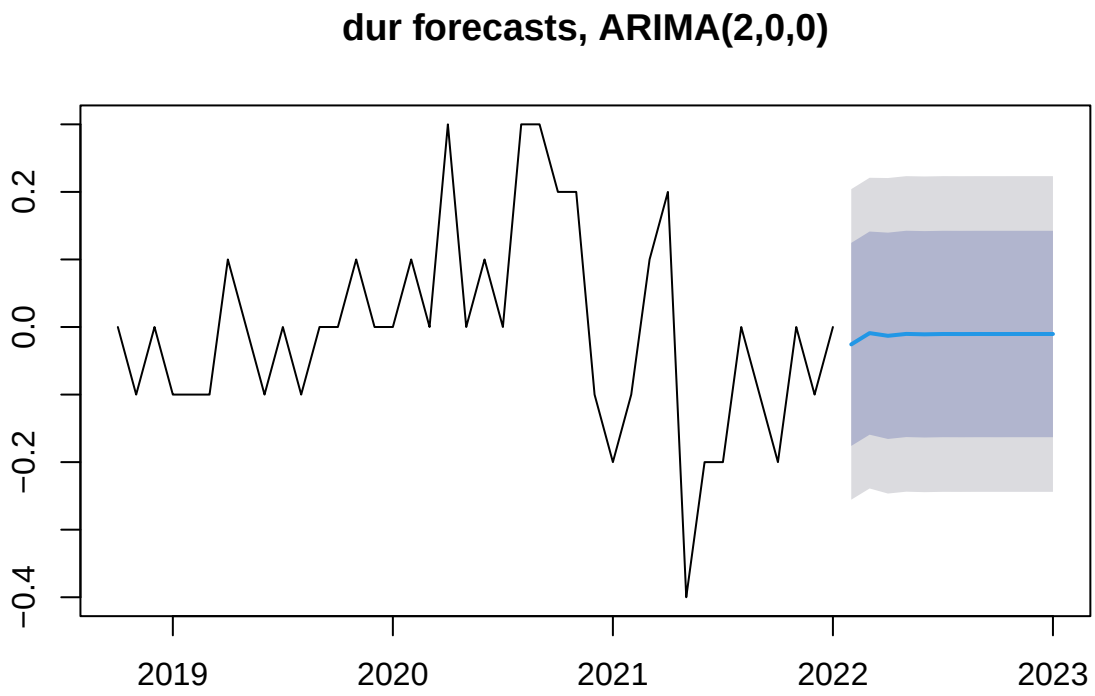
If we want to forecast from this model we can do this as follows

```
for_dur <- forecast(fit_dur,h=12)
plot(for_dur, main = "dur forecasts, ARIMA(2,0,0)")
```



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, main = "dur forecasts, ARIMA(2,0,0)")
```



As we can see the forecast does flatten out very quickly. Essentially it will move towards the estimated `mean` value, -0.011. The shaded areas around the forecast are forecast or prediction intervals. They recognise that the model contains an error term and hence we acknowledge that there is uncertainty about our prediction. The two different shades indicate 80 and 95% prediction intervals. They do, however, **assume that the estimated model is correct**.

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
## 2022              -0.025737759 -0.009027248 -0.012996195 -0.010192858
## 2023 -0.010335044
##               Jun          Jul          Aug          Sep          Oct
## 2022 -0.010797445 -0.010325144 -0.010415751 -0.010335808 -0.010349085
## 2023
##               Nov          Dec
## 2022 -0.010335486 -0.010337370
## 2023
```

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 may be interested in the 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.

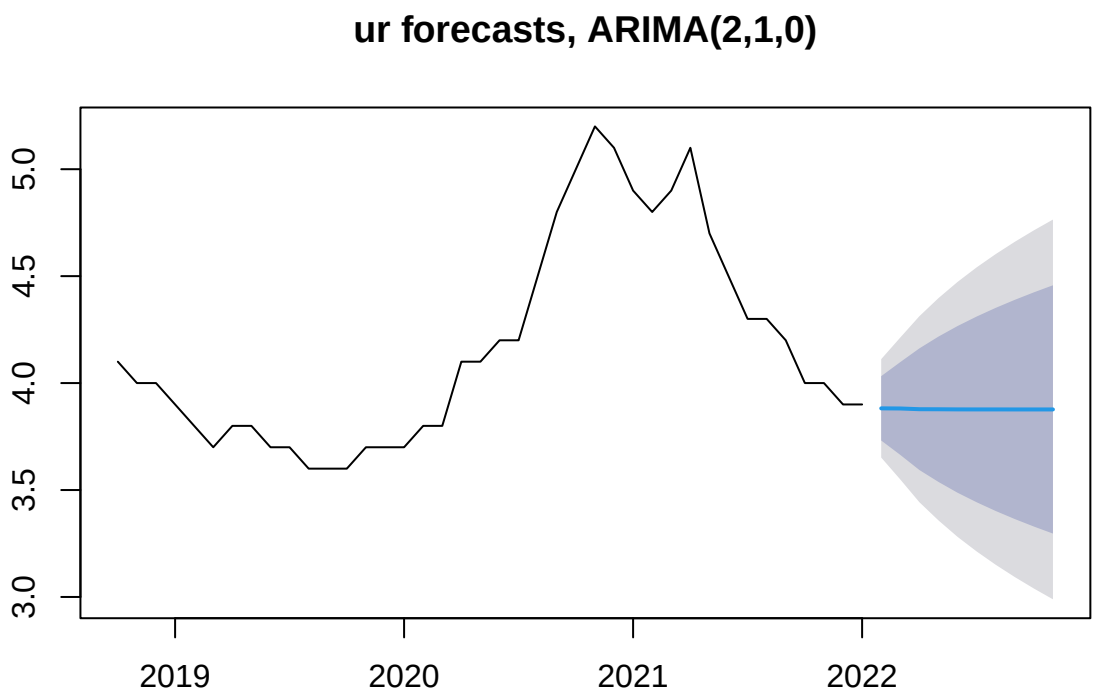
Fortunately there is a slightly easier way to deal with this. Recall that when we called the estimation function

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)
```

```
## Series: ur_ts
## ARIMA(2,1,0)
##
## Coefficients:
##          ar1     ar2
##       0.0386  0.1813
## s.e.  0.0519  0.0520
##
## sigma^2 = 0.01378:  log likelihood = 259.22
## AIC=-512.45   AICc=-512.38   BIC=-500.82
##
## Training set error measures:
##                       ME      RMSE        MAE        MPE     MAPE    MASE
## Training set -0.00809654 0.1168872 0.08729233 -0.1631667 1.599218 0.18598
##                       ACF1
## Training set -0.0002417907
```

```
for_ur <- forecast(fit_ur, h =10)
plot(for_ur, include = 40, main = "ur forecasts, ARIMA(2,1,0)")
```



In essence you will get the same as if you were to aggregate up the forecast changes.

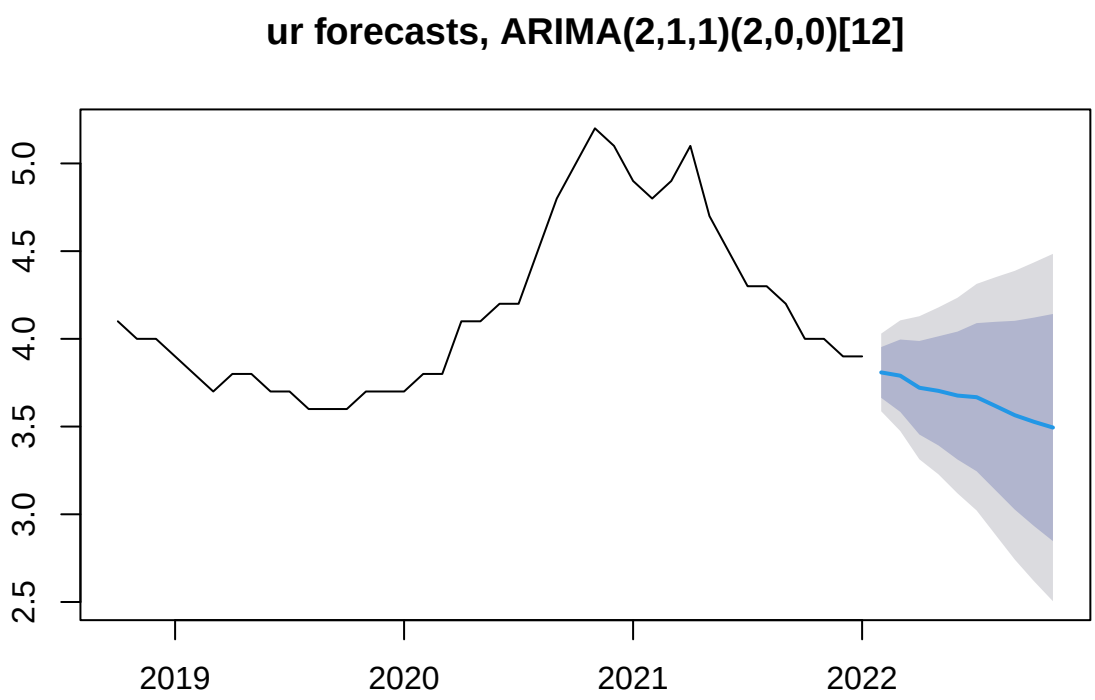
Using the `auto.arima` function

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(1,1,4)(2,0,0)[12]
##
## Coefficients:
##          ar1      ma1     ma2      ma3     ma4    sar1     sar2
##       0.7430  -0.7290  0.1425  -0.1746  0.2541  0.0204  -0.1563
## s.e.  0.1044   0.1078  0.0625   0.0671  0.0588  0.0560   0.0577
##
## sigma^2 = 0.01277:  log likelihood = 274.88
## AIC=-533.76   AICc=-533.34   BIC=-502.74
##
## Training set error measures:
##                        ME      RMSE        MAE        MPE     MAPE      MASE
## Training set -0.006423143 0.1117218 0.08497161 -0.1247938 1.559444 0.1810356
##                      ACF1
## Training set -0.003802818
```

```
for_ur_a <- forecast(fit_ur_a, h =10)
plot(for_ur_a, include = 40, main = "ur forecasts, 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.

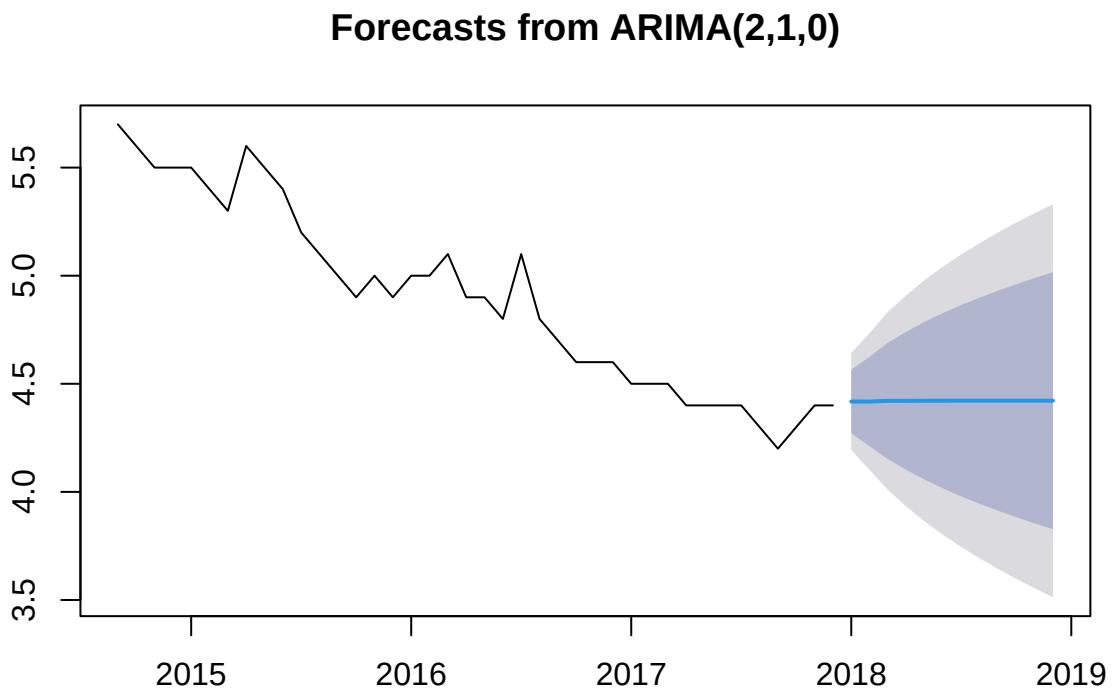
Instead we will turn back time by one year and pretend that it is the end of 2017 and we wanted to forecast the unemployment rate in 2018. Then we wil be able to compare our forecasts to the actual realisations.

So let's start by restricting the sample up to and including december 2017 using the `window` function.

```
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.

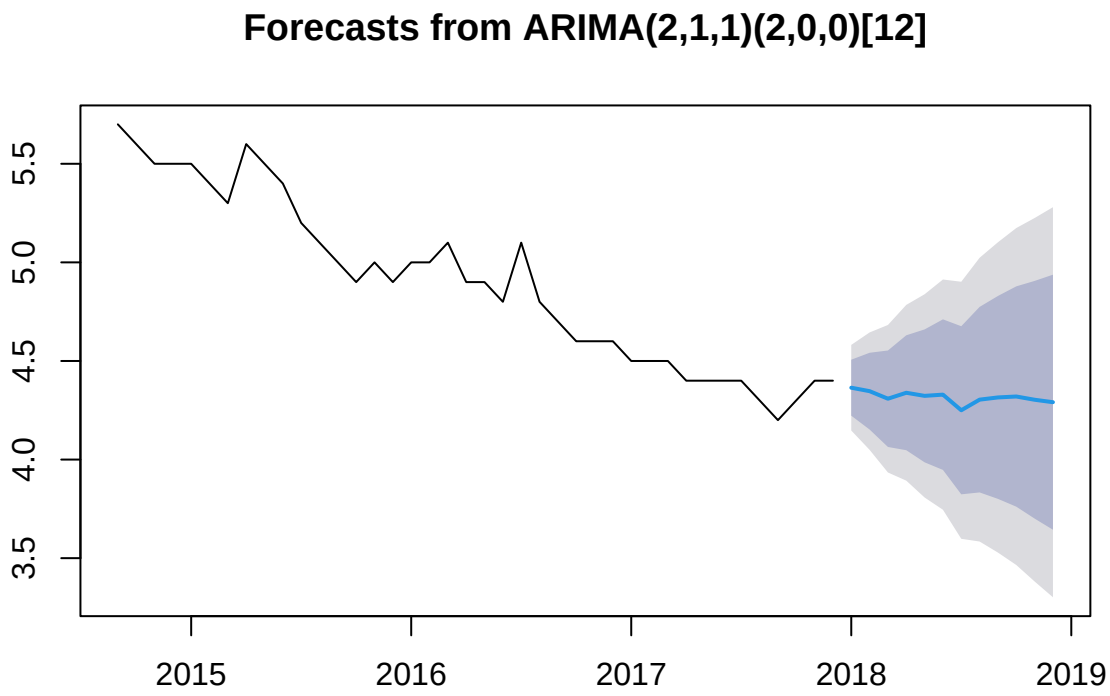
```
fit_ur_17 <- Arima(ur_ts_17,order = c(2,1,0))
for_ur_17 <- forecast(fit_ur_17, h =12)
plot(for_ur_17, include = 40)
```




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

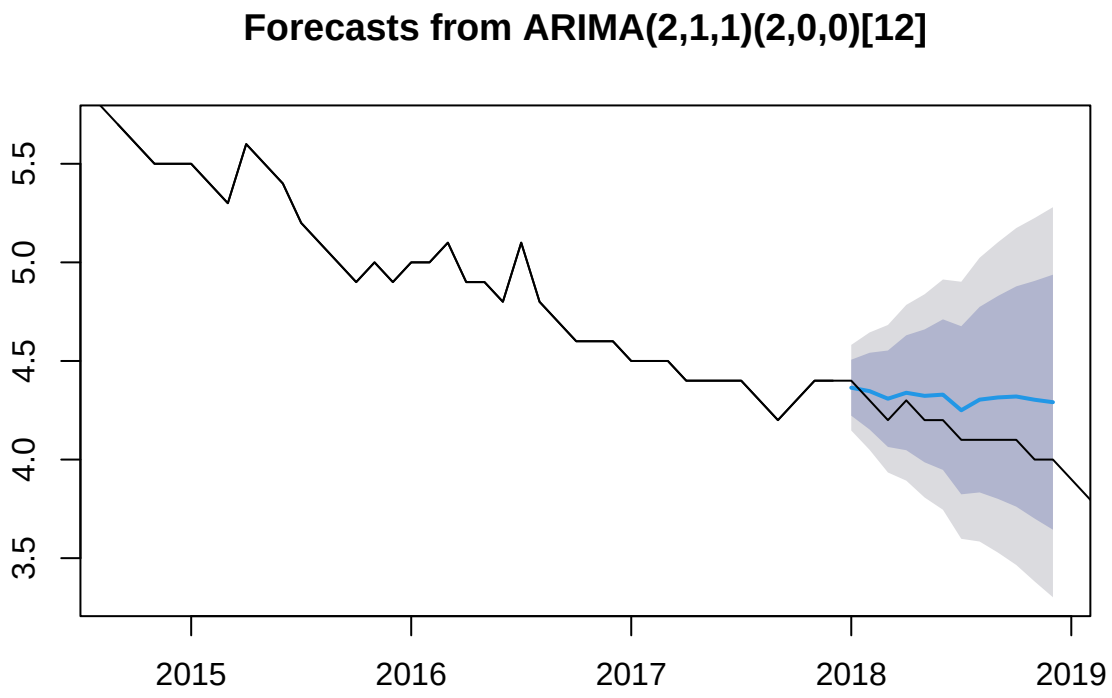
```
## Series: ur_ts_17
## ARIMA(2,1,1)(2,0,0)[12]
##
## Coefficients:
##          ar1     ar2      ma1     sar1     sar2
##       0.809  0.1526  -0.8700  -0.0358  -0.2152
## s.e.  0.073  0.0609   0.0508   0.0582   0.0592
##
## sigma^2 = 0.01227:  log likelihood = 242.51
## AIC=-473.02   AICc=-472.74   BIC=-450.64
##
## Training set error measures:
##                       ME      RMSE       MAE         MPE     MAPE      MASE
## Training set -0.004330286 0.1096765 0.08473188 -0.06255499 1.487238 0.1857223
##                      ACF1
## Training set 0.003385942
```

```
for_ur_a_17 <- forecast(fit_ur_a_17, h =12)
plot(for_ur_a_17, include = 40)
```



You can actually plot the observations next to the forecasts.

```
plot(for_ur_a_17, include = 40)
lines(ur_ts) # this adds a line to the previous plot
```



This is where we are reaping all the advantages of using data with time-series formats. Both `for_ur_a_17` and `ur_ts` contain date information and R will automatically align the data correctly.

Now we use the `accuracy` function to evaluate how well these forecasts fit the actual observations in 2018.

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

```
##                       ME      RMSE        MAE        MPE     MAPE      MASE
## Training set -0.008496628 0.1141159 0.08595689 -0.1609257 1.504272 0.1884074
## Test set     -0.254293230 0.2807261 0.25429323 -6.1879077 6.187908 0.5573807
##                      ACF1 Theil's U
## Training set -0.004431789        NA
## Test set      0.573813816  4.090548
```

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      MASE
## Training set -0.004330286 0.1096765 0.08473188 -0.06255499 1.487238 0.1857223
## Test set     -0.149464958 0.1788153 0.15539167 -3.65504176 3.789740 0.3406002
##                      ACF1 Theil's U
## Training set 0.003385942        NA
## Test set     0.622004661  2.613773
```

The automatically chosen model does improve on the forecast performance.

When performing forecasts with such models there is a significant note of caution in order. When we use an estimated model for forecasting we assume that the model will remian to be the correct model for the forecast period (in addition to assuming that it was correct for the in-sample period). This is a major assumption. Recall that, basically, a forecast model for stationary data contains information on a) a level to which the series is predited to converge in the long-run and b) a speed and pattern of convergance to that level in a).

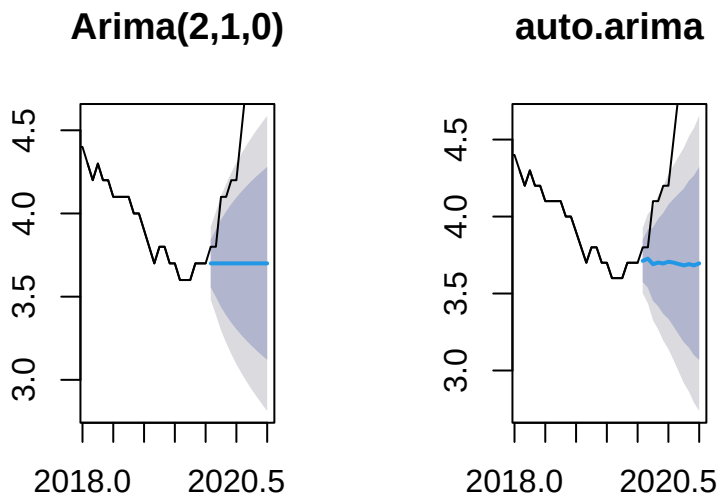
Any misspecification, in particular with respect to a) will cause forecasts to be off. And impotantly any substantial change in the economy which changes the information in a) or b), but particularly in a), during the forecast period will also ensure that the forecasts from such a model are problematic and potentially significantly off.

It is for this reason that any forecasts produced from such models (and much more complex models) should always be accompanied with this health warning.

Let's illustrate this: Imagine you had data up to January 2020 and you wanted to produce forecasts for the 12 months after.

```
ur_ts_20 <- window(ur_ts,end = c(2020,1))

fit_ur_20 <- Arima(ur_ts_20,order = c(2,1,0))
for_ur_20 <- forecast(fit_ur_20, h =12)

fit_ur_a_20 <- auto.arima(ur_ts_20)
for_ur_a_20 <- forecast(fit_ur_a_20, h =12)
```

We can now look at these graphically and compare to what actually happened in 2020.



Clearly the forecasts did not capture what actually happened. The realisations are also outside the confidence intervals. But, I guess we got to be realistic here. A simple model like ours, in fact even a more complicated model, cannot be expected to anticipate an extraordinary event like Covid-19.