Mauna Kea CO2 levels

Michal Odler

13.10.2021

Introduction

In this project, we are going to analyze a time series containing monthly measurements of CO2 levels atop the Mauna Kea mountain in Hawaii. The data available here stretches from January 1959 till November 2020 and contains the value of ppm concentration of CO2 levels for each month, coupled by a column of deseasonalized values, as well as the number of days during which the level was measured, std. deviations of days and uncertainty of the monthly mean. Some of these values are also said to have been interpolated, but for the sake of the project we will consider these observations as unedited.

Firstly, let us load the data and look at the first couple of rows of our dataset:

```
##                                                                                 V1
## 1  1959    1   1959.0411      315.58      315.55     -1   -9.99   -0.99
## 2  1959    2   1959.1260      316.48      315.86     -1   -9.99   -0.99
## 3  1959    3   1959.2027      316.65      315.38     -1   -9.99   -0.99
## 4  1959    4   1959.2877      317.72      315.41     -1   -9.99   -0.99
## 5  1959    5   1959.3699      318.29      315.49     -1   -9.99   -0.99
## 6  1959    6   1959.4548      318.15      316.03     -1   -9.99   -0.99
```

As we can see, the dataset is broken up, as the individual rows are just long strings belonging to a single text column. We will fix this by using a for cycle with a regex functionality:

```
data <- data.frame()
for (i in 1:dim(mauna)[1]) {
  data <- rbind(data, unlist(strsplit(mauna[i, 1], '\\s+')))
}
data[, c(1, 7, 8, 9)] <- NULL # delete columns we won't use
names(data) <- c('year', 'month', 'year.frac', 'co2avg', 'deseasoned')
data <- as.data.frame(sapply(data, as.numeric))
head(data)
```

```
##   year month year.frac co2avg deseasoned
## 1 1959     1  1959.041 315.58     315.55
## 2 1959     2  1959.126 316.48     315.86
## 3 1959     3  1959.203 316.65     315.38
## 4 1959     4  1959.288 317.72     315.41
## 5 1959     5  1959.370 318.29     315.49
## 6 1959     6  1959.455 318.15     316.03
```

At this moment our data is in the form of a clean data frame, so we can proceed to the next step, which is EDA.

Exploratory Data Analysis

Firstly, let us see if we are dealing with some missing data:

```
sum(is.na(data))
```

```
## [1] 0
```

Our data contains no NA values, so we may look at the time series graph, shown on Fig. 1:

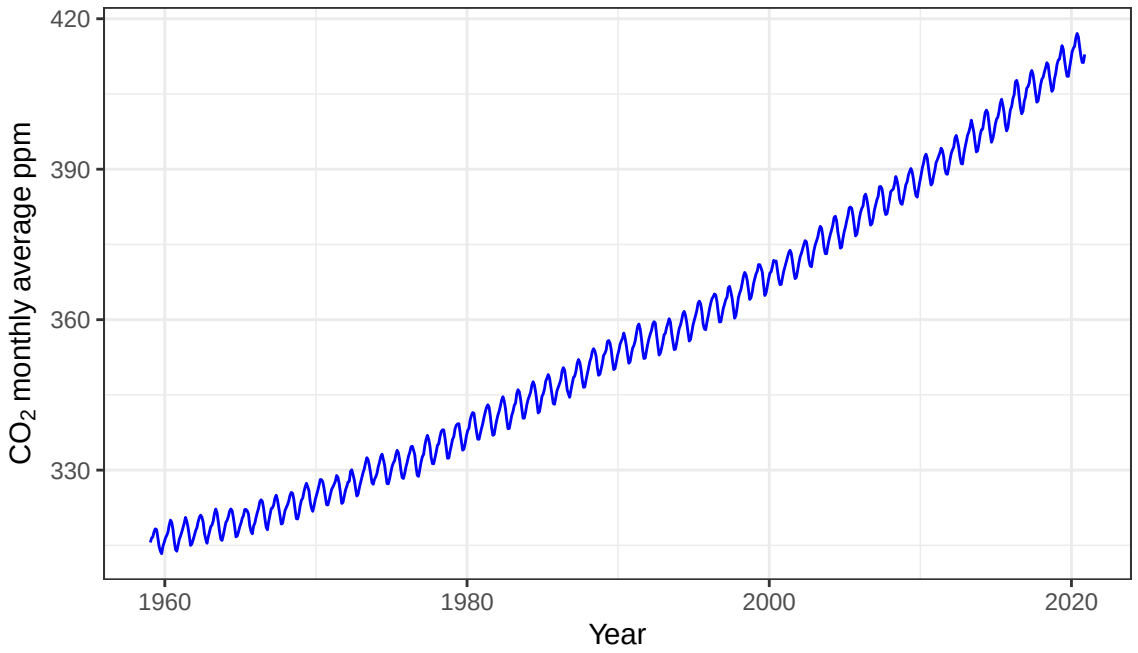

Figure 1: Monthly CO2 at Mauna Kea [ppm]

We can conclude that the CO2 concentration is steadily rising from the start of our observations and throughout the current century. A closer look even tells us that this rise is faster than linear. There is also an apparent seasonal factor that affects the carbon dioxide concentrations, meaning that the levels fluctuate throughout the year.

R has some great tools for working with time series, including a special class for TS objects (called just *ts*). We should therefore convert our data column into this object and continue working with it:

```
# manually setting the start, end and freq parameters
ts.vect <- ts(data$co2avg, start = 1959, end = 2020 + 10/12, freq = 12)
head(ts.vect, 12)
```

```
##         Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct
## 1959 315.58 316.48 316.65 317.72 318.29 318.15 316.54 314.80 313.84 313.33
##         Nov    Dec
## 1959 314.81 315.58
```

We can consider this vector just as a numerical vector. However, it also contains the time information (year+month in our case), and we will use it as the key input for useful time series functions implemented in R.

Decomposition

Time series are usually made up of several components, such as level, trend, seasonality, cycles and noise. Looking at our previous plot, we clearly see that our series has an upward trend and seasonally fluctuates throughout the individual years. We may try a simple additive decompostion, which breaks up the series as follows:

$$Y_t = T_t + S_t + E_t,$$

i.e. each observation is a sum a trend component T_t, seasonal component S_t and a noise component E_t, each of them obviously dependent on time. This decompostion is the simplest one and it allows us to see how the trend component deaseasonalized series changes over time. However, it would not be able to decompose the series correctly if the seasonal effects were not constant; in this case it would be correct to use another form of decompostion, e.g. multiplicative decomposition, or STL.

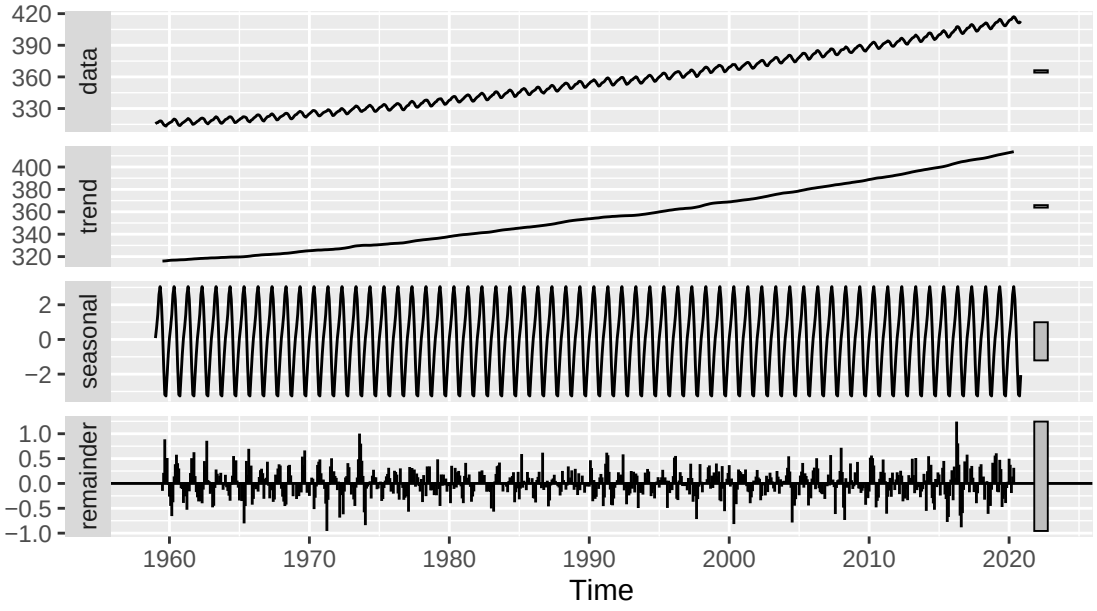

Figure 2: Additive decomposition

By looking at Fig. 2, it looks like the trend is indeed just a bit curved, almost linear. The resudials (noise) seem to be normally distributed around 0, which is a desirable outcome. They do show a bit higher variability in the first and the last decades, which might indicate stronger seasonal effects in those timeframes - we may also try a more robust type of decomposition, the STL (*Seasonal and Trend decomposition using Loess*).

The resisdual part on Fig. 3 looks a bit different, but the differences seem negligible. Apart from visual inspection of the residuals, we can also look at their autocorrelation graph (Fig. 4), which might uncover some things we might miss with the naked eye:

```
# The s.window parameter is chosen empirically based on the seasonal variations
acf(stl(ts.vect, s.window = 51)$time.series[, 3], main = '')
```

Indeed, the residuals still show some autocorrelation (especially 3 to 6 months later). We will use these plots and their insight again, while constructing more complex models on our dataset.

Prediction

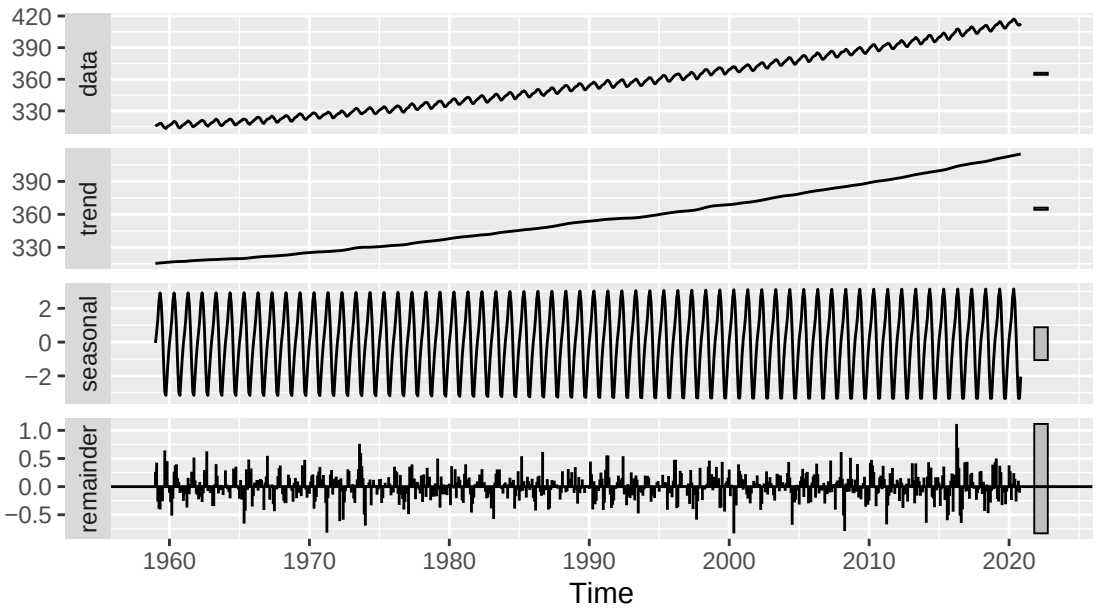

Figure 3: STL decomposition

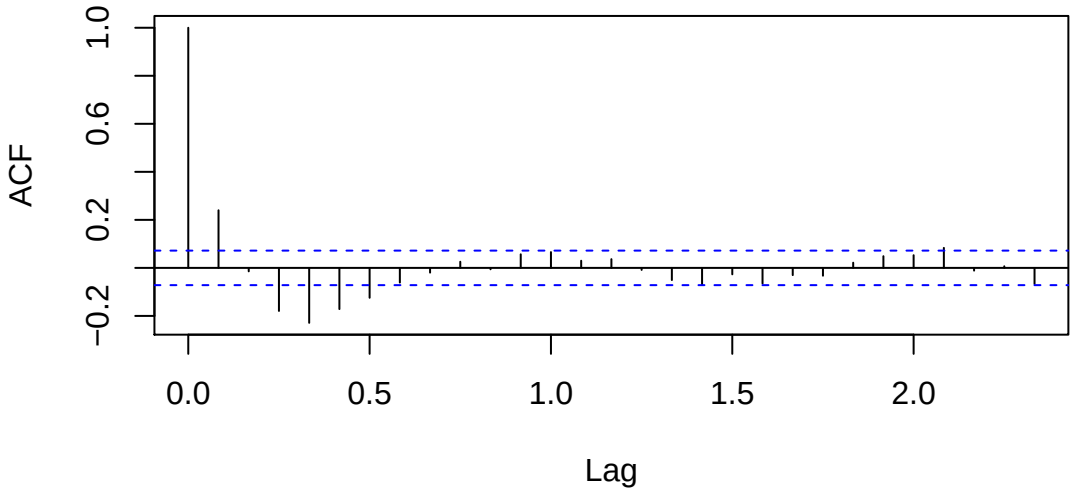

Figure 4: Autocorrelation function of the residual component

Holt-Winters method

Let us move on to exponential smoothing methods. These methods work by taking both newer and older observations into account, while putting smaller weights on the older observations to lessen their effect. These methods can be also used for efficient and fast forecasts. The first one we will use is the Holt-Winters method, which also works with series that have a trend and seasonality. It is a bit more robust than simpler methods such as basic exponential smoothing or the Holt's method. A great article about how the Holt-Winters method works can be found here.

We will divide our time series into 2 parts: The first one will be up to the year 2015 and will serve as our training set, on which we will build our models and tune the parameters. The remaining observations will be our test set; we will use them to compare the accuracy of our final predictions.

```
ts.train <- window(ts.vect, end = 2014.99)
ts.test <- window(ts.vect, start = 2015)
```

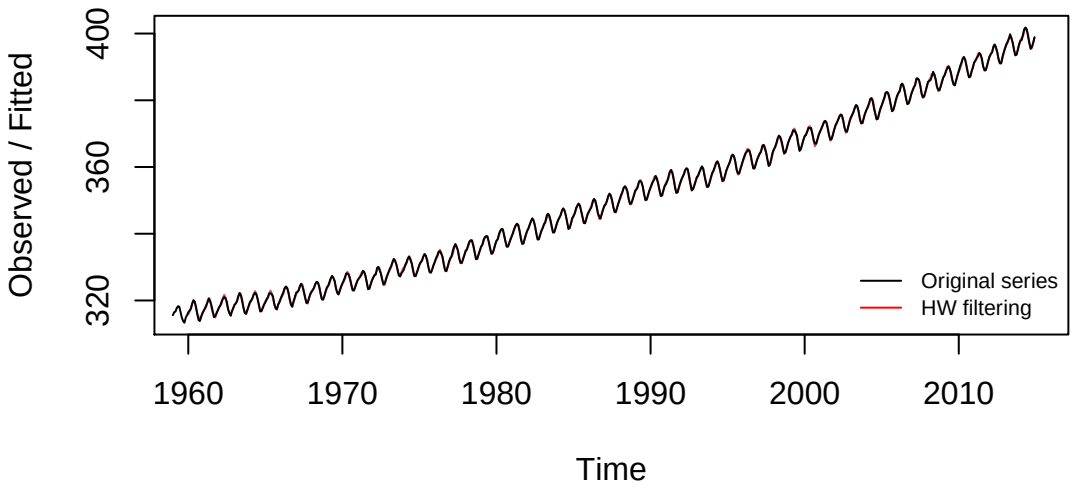

Figure 5: Holt-Winters filtering

The Holt-Winters filtering is shown on Fig. 5. The smoothed time series on the training part is almost identical to the real one. Now we will compute the prediction for the next (almost) 6 years and compare it with the test series, shown on Fig. 6:

The 95% confidence interval is pretty narrow, which indicates that the series behaves well and both the trend and seasonality seem very predictable. However, the test set data is *barely* contained in the confidence interval, even rising beyond it sometime during spring 2016. This indicates that a rather simple Holt-Winters prediction was pretty accurate, yet the real levels of CO2 at Mauna Kea during the years 2015-2020 were higher than we would predict by the Holt-Winters prediction.

SARIMA models

Another family of time series models are the SARIMA models. This abbreviation stands for the intimidating name *Seasonal Autoreggresive Intergrated Moving Average* models. ARIMA models, unlike smoothing models, are build to describe autocorrelations in the data. We shall omit the details of how exactly these models work, since the theory contains enough math to fill half of a graduate course on time series.

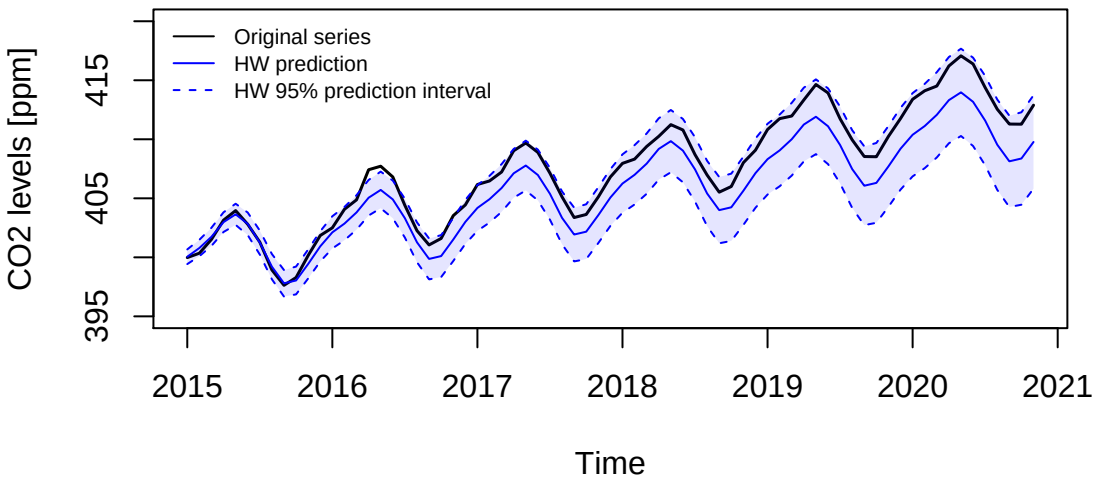

Figure 6: Holt-Winters filtering prediction

A lot of the modelling we do on time series only works for what we call *stationary* time series, which is a very imporant definition in this topic. In a nutshell, a stationary time series is one that has *constant trend*, and its autocovariance function has to be a function of *time difference* only, not time itself. It also must have a finite second moment, but that is fulfilled for basically any real time series we wish to analyze.

This definition clarifies that any time series with seasonality or a non-constant trend is not stationary. Our main goal will be finding the steps needed to make our series a stationary one and then estimating the autoregression and moving average coefficients so that we can describe our time series with an ARMA model. Luckily, R has functions for this parameter estimation and the only weight on our shoulders is to correctly set the degree of AR/MA polynomials.

Firstly, we wish to remove the trend and seasonality from our time series. The way we do this is by differencing (more info here), which just means taking the difference of two consecutive observations. This should remove the trend:

Fig. 7 shows that the trend is successfully removed, but the seasonality is still visible in thedata. We can remove this component by *seasonal* differencing, which works on the same principle, but instead of differencing consecutive observations, we are taking differences of observations m indices away (where $m = 12$ is the length of our season). Fig. 8 shows the deseasonalized and detrended series, which already looks stationary and therefore we may fit our ARMA model on it.

The basic approach of determining AR/MA coefficients is by looking at the ACF & PACF (which are the autocorrelation and parital autocorrelation functions) graphs of our stationary time series (Fig. 8). The cutoff points in sample ACF and exponential decay in sample PACF graphs (Fig. 9) indicate both MA and seasonal MA coefficients to be 1, while integration coefficients are given by differencing - so both non-seasonal and seasonal will have value 1 (since we used 1st order difference and one seasonal difference). More info on how to determine the AR/MA coefficients by hand from looking at these graphs can be found here.

There is also an R function *auto.arima* that does the coefficient choice for us - however, it is not always smart to rely on this automatic decision. Let us compare our hand-picked ARIMA model with the one R chose for our training time series.

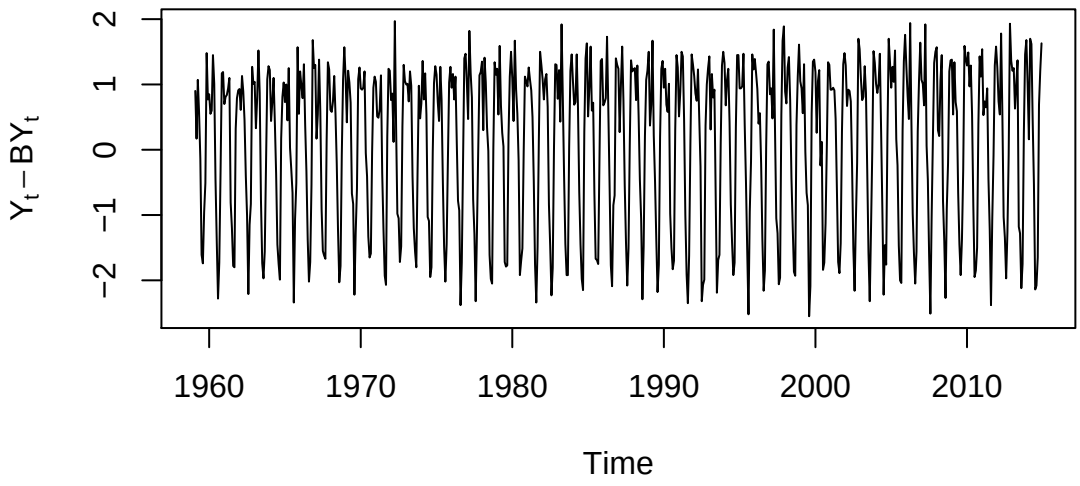

Figure 7: 1st order differenced time series

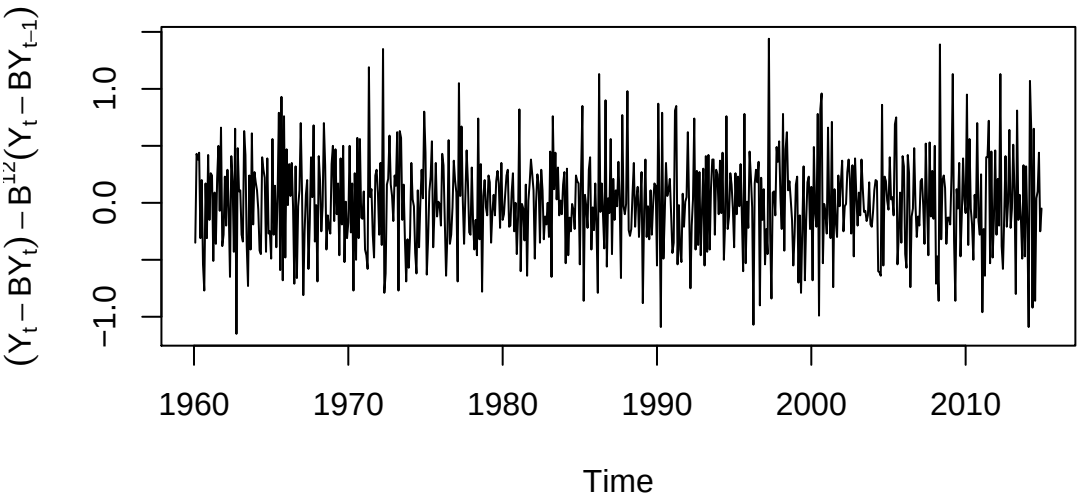

Figure 8: 1st order and seasonally differenced time series

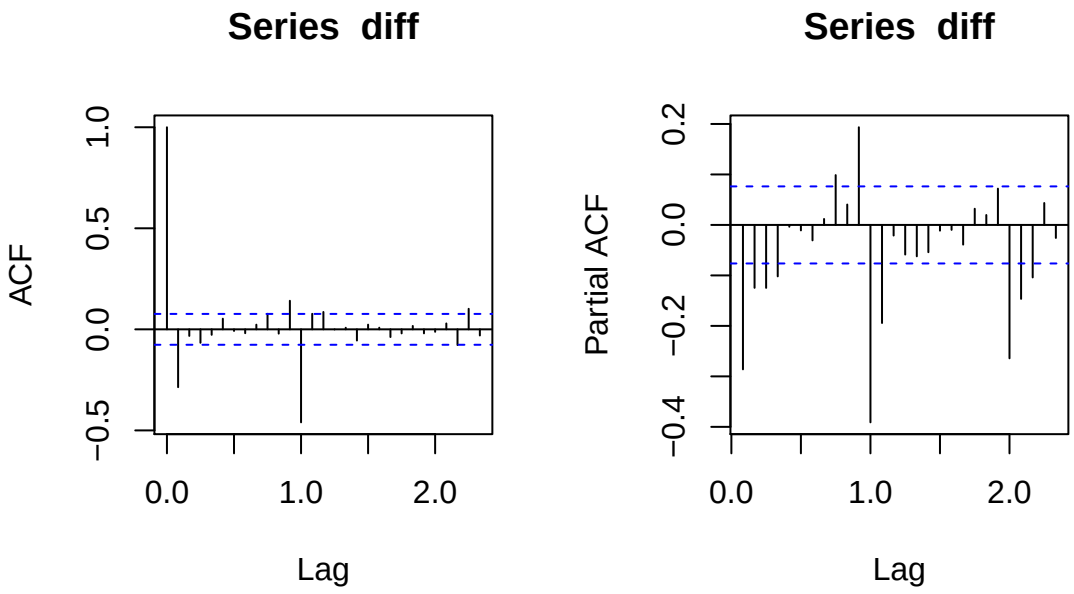

Figure 9: Sample autocorrelation and sample partial autocorrelation of our differenced time series

```
mod1 <- Arima(ts.train, order = c(0, 1, 1), seasonal = c(0, 1, 1))
mod1
```

```
## Series: ts.train
## ARIMA(0,1,1)(0,1,1)[12]
##
## Coefficients:
##           ma1     sma1
##       -0.3806  -0.8780
## s.e.   0.0403   0.0188
##
## sigma^2 estimated as 0.09179:  log likelihood=-148.95
## AIC=303.9   AICc=303.94   BIC=317.37
```

```
mod.auto <- auto.arima(ts.train)
mod.auto
```

```
## Series: ts.train
## ARIMA(1,1,1)(1,1,1)[12]
##
## Coefficients:
##          ar1      ma1     sar1     sma1
##       0.1777  -0.5396  -0.0178  -0.8733
## s.e.  0.1044   0.0904   0.0445   0.0213
##
## sigma^2 estimated as 0.09169:  log likelihood=-147.53
## AIC=305.06   AICc=305.15   BIC=327.52
```

As we can see from the summaries, the ARIMA model R chose for us differs from the one found by graph inspection - it also included and estimated both seasonal and non-seasonal 1st degree AR coefficient. However,

if we look at the AIC/BIC criteria, we conclude that our model was better than R's; so if we go by the principle of parsimony, we found a simpler model that seems do describe the training series better than the more complex one.

We may try other combinations of AR/MA coefficients, but after checking this I failed to find a better model in terms of Akaike or Bayesian criterion. We may continue our analysis and try to predict the CO2 level for the next 6 years and check which one of our 3 predictions was the most accurate (prediction by Holt-Winters filtering, our ARIMA model, R's model chosen by auto.arima).

We will use *Mean Absolute Error* and *Mean Squared Error* as our model metrics (described here). Let us predict the CO2 level values for the test set timeframe, i.e. from January 2015 until November 2020 with our 3 models. Afterwards we will calculate the *MAE* and *MSE* values and evaluate the prediction quality of these models.

```
# Holt-Winters prediction is already calcualted in predict.hw
predict.arima <- forecast(mod1, h = length(ts.test))
predict.auto <- forecast(mod.auto, h = length(ts.test))
# Create data.frame with MAE's and MSE's
df <- data.frame('MAE' = c(sum(abs(ts.test - predict.arima$mean)) / length(ts.test),
                           sum(abs(ts.test - predict.auto$mean)) / length(ts.test),
                           sum(abs(ts.test - predict.hw[, 1])) / length(ts.test)),
                 'MSE' = c(sum((ts.test - predict.arima$mean)^2) / length(ts.test),
                           sum((ts.test - predict.auto$mean)^2) / length(ts.test),
                           sum((ts.test - predict.hw[, 1])^2) / length(ts.test)))
rownames(df) <- c('Our SARIMA model', "R's SARIMA model",
                  'Holt-Winters prediction')
print(df)
```

```
##                              MAE      MSE
## Our SARIMA model        1.741682 3.839532
## R's SARIMA model        1.731733 3.789299
## Holt-Winters prediction 1.720675 3.749293
```

Fig. 10 shows the prediction of both SARIMA models, including the actual observed values and 95% confidence intervals. ARIMA(0,1,1)(0,1,1)[12] is our model, while ARIMA(1,1,1)(1,1,1)[12] is auto.arima's choice. The models look almost indistinguishable and visual inspection will not tell us much in this case.

However, the results of our metrics are showing some interesting results. The best model, both by *MAE* and *MSE* is the Holt-Winters prediction, although the results were close. We would probably expect that the more complex and robust SARIMA models would perform better, but we have to take into account that we are dealing with a rather well-behaved and predictable time series, which means that even a simple algorithm might lead to good results. Our model's performance was the worst from the three, but the results were pretty close. We see that even a model with worse AIC/BIC metrics might perform better on the test set.

Conclusion

The aim of this project was to showcase basic time-series analysis in the R software. We showed some decomposition techniques as a part of exploratory analysis, the simple Holt-Winters prediction method based on exponential smoothing and talked about Seasonal ARIMA models, their implementation and parameter setting in R. The results concluded that simple prediction methods are on par with SARIMA models, if we're dealing with simple enough time series.

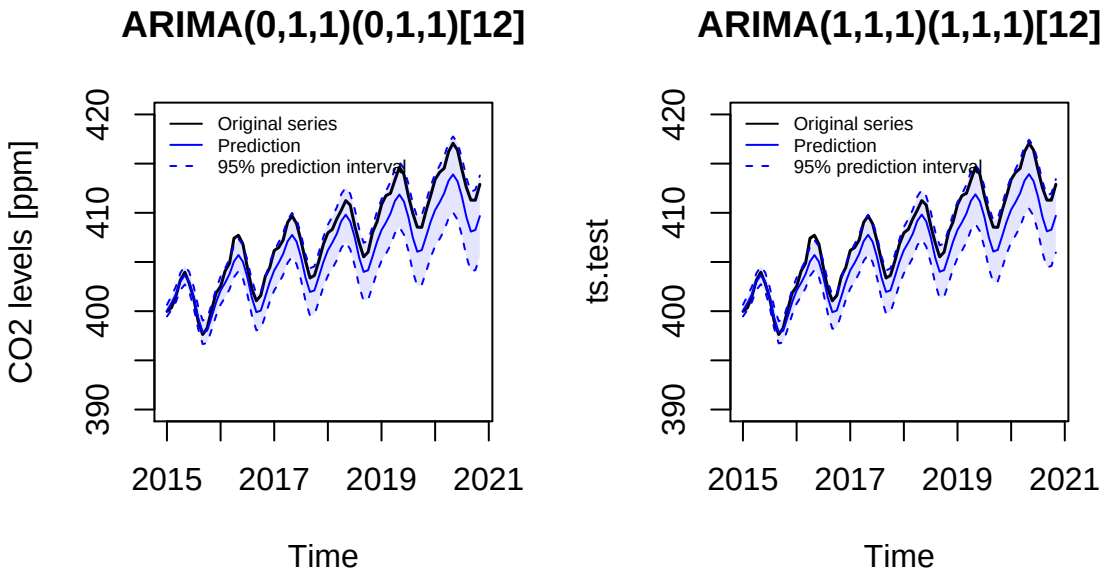

Figure 10: Predictions and 95% confindence intervals for ours and R's SARIMA models