Air quality project : Time Series Analysis

Yousra Cherkaoui Tangi - Lou Peltier - Guilhem Bois

12 janvier, 2020

Contents

Preliminary

Our project consists of time series analysis using R programming language. Our data set contains the responses of a gas mutisensor device deployed on the field in an Italian city, in a significantly polluted area, at road level. The data set contains 9358 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. Data were recorded from March 2004 to February 2005.

```
dim(data)
```

```
## [1] 9471    17
```

```
str(data)
```

```
## 'data.frame':    9471 obs. of  17 variables:
##  $ Date        : chr  "10/03/2004" "10/03/2004" "10/03/2004" "10/03/2004" ...
##  $ Time        : chr  "18.00.00" "19.00.00" "20.00.00" "21.00.00" ...
##  $ CO.GT.      : chr  "2,6" "2" "2,2" "2,2" ...
##  $ PT08.S1.CO. : int  1360 1292 1402 1376 1272 1197 1185 1136 1094 1010 ...
##  $ NMHC.GT.    : int  150 112 88 80 51 38 31 31 24 19 ...
##  $ C6H6.GT.    : chr  "11,9" "9,4" "9,0" "9,2" ...
##  $ PT08.S2.NMHC.: int  1046 955 939 948 836 750 690 672 609 561 ...
##  $ NOx.GT.     : int  166 103 131 172 131 89 62 62 45 -200 ...
##  $ PT08.S3.NOx. : int  1056 1174 1140 1092 1205 1337 1462 1453 1579 1705 ...
##  $ NO2.GT.     : int  113 92 114 122 116 96 77 76 60 -200 ...
##  $ PT08.S4.NO2. : int  1692 1559 1555 1584 1490 1393 1333 1333 1276 1235 ...
##  $ PT08.S5.O3. : int  1268 972 1074 1203 1110 949 733 730 620 501 ...
##  $ T           : chr  "13,6" "13,3" "11,9" "11,0" ...
##  $ RH          : chr  "48,9" "47,7" "54,0" "60,0" ...
##  $ AH          : chr  "0,7578" "0,7255" "0,7502" "0,7867" ...
##  $ X           : logi  NA NA NA NA NA NA ...
##  $ X.1         : logi  NA NA NA NA NA NA ...
```

Attribute	Description
Date	Date (DD/MM/YYYY)
Time	Time (HH.MM.SS)
CO(GT)	True hourly averaged concentration CO in mg/mˆ3 (reference analyzer)
PT08.S1(CO)	PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted)
NMHC(GT)	True hourly averaged overall Non Metanic Hydro Carbons concentration in microg/mˆ3(reference analyzer
C6H6(GT)	True hourly averaged Benzene concentration in microg/mˆ3 (reference analyzer)
PT08.S2(NMHC)	PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted)
NOx(GT)	True hourly averaged NOx concentration in ppb (reference analyzer)
PT08.S3(NOx)	PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted)
NO2(GT)	True hourly averaged NO2 concentration in microg/mˆ3 (reference analyzer)
PT08.S4(NO2)	PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO2 targeted)
PT08.S5(O3)	PT08.S5 (indium oxide) hourly averaged sensor response (nominally O3 targeted)
T	Temperature in °C
RH	Relative Humidity (%)
AH	AH Absolute Humidity

I)-Cleaning and preprocessing

1) Data cleaning

As we can see in the str(dim) output, our data contains missing values, some are directly reported as NA and others have been assigned to -200. It also contains a certain number of numeric variables that are characters which makes it not directly exploitable and some numbers have commas within them. For all these reasons, we had to clean our data set first so that we can exploit it afterwards.

```
## Loading required package: Rcpp
```

```
## ##
## ## Amelia II: Multiple Imputation
## ## (Version 1.7.6, built: 2019-11-24)
## ## Copyright (C) 2005-2020 James Honaker, Gary King and Matthew Blackwell
## ## Refer to http://gking.harvard.edu/amelia/ for more information
## ##
```


Missing values vs observed
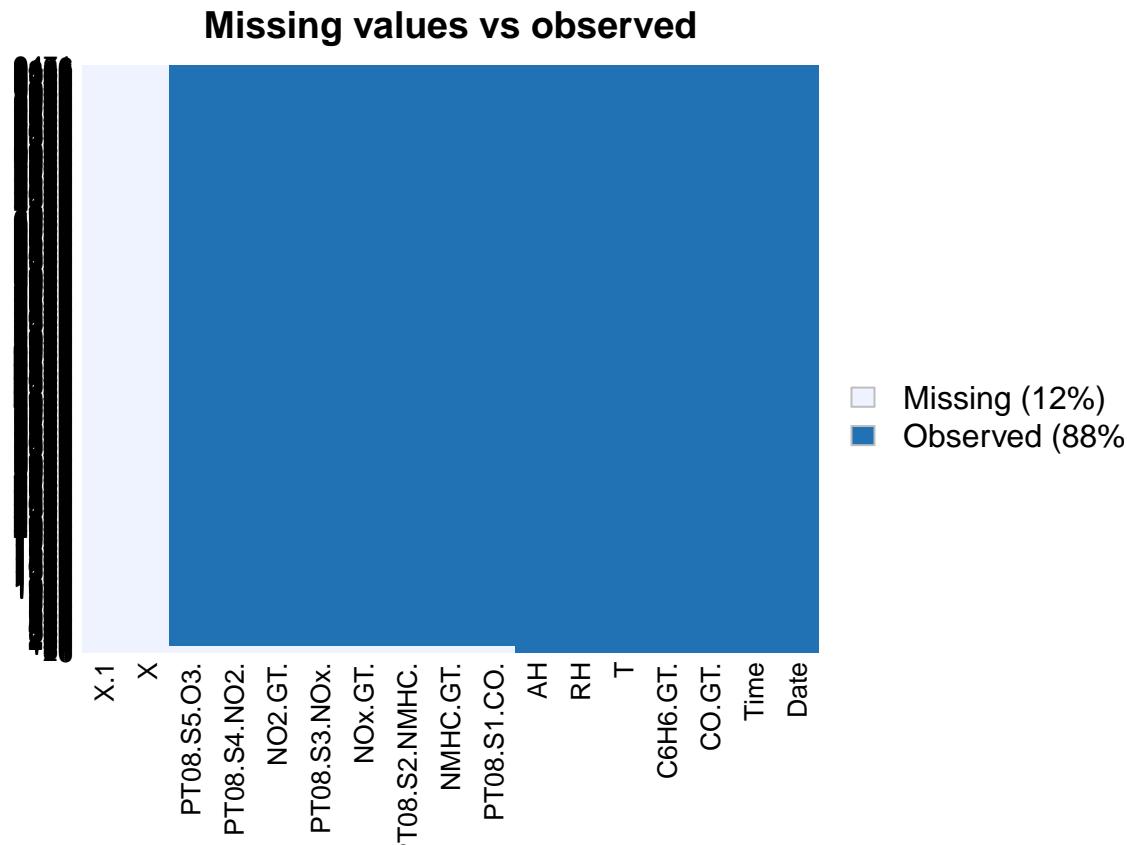
We remove X1 and X beacause they have no data within them, also we compute the necessary changes so that our data is exploitable afterwards.

We noticed that NHMC contains a lot of -200 values, so we prefered to delet it along with all the lines that contain -200.

```r
summary(data$NMHC.GT.)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -200.0  -200.0  -200.0  -159.1  -200.0  1189.0
```

```
str(data)
```

```
## 'data.frame':    6941 obs. of  14 variables:
##  $ Date        : Date, format: "2004-03-10" "2004-03-10" ...
##  $ Time        : chr  "18.00.00" "19.00.00" "20.00.00" "21.00.00" ...
##  $ CO.GT.      : num  2.6 2 2.2 2.2 1.6 1.2 1.2 1 0.9 0.7 ...
##  $ PT08.S1.CO. : int  1360 1292 1402 1376 1272 1197 1185 1136 1094 1066 ...
##  $ C6H6.GT.    : num  11.9 9.4 9 9.2 6.5 4.7 3.6 3.3 2.3 1.1 ...
##  $ PT08.S2.NMHC.: int  1046 955 939 948 836 750 690 672 609 512 ...
##  $ NOx.GT.     : int  166 103 131 172 131 89 62 62 45 16 ...
##  $ PT08.S3.NOx.: int  1056 1174 1140 1092 1205 1337 1462 1453 1579 1918 ...
##  $ NO2.GT.     : int  113 92 114 122 116 96 77 76 60 28 ...
##  $ PT08.S4.NO2.: int  1692 1559 1555 1584 1490 1393 1333 1333 1276 1182 ...
##  $ PT08.S5.O3. : int  1268 972 1074 1203 1110 949 733 730 620 422 ...
##  $ T           : num  13.6 13.3 11.9 11 11.2 11.2 11.3 10.7 10.7 11 ...
##  $ RH          : num  48.9 47.7 54 60 59.6 59.2 56.8 60 59.7 56.2 ...
##  $ AH          : num  0.758 0.726 0.75 0.787 0.789 ...
```

Now that our data is nearly clean, we chose to consider the daily concentration of the different gases our main focus of study by taking the average of the concentrations observed daily.

We obtain 341 observations of 13 variables.

```
str(daily_data)
```

```
## 'data.frame':    341 obs. of  13 variables:
##  $ Date        : Date, format: "2004-03-10" "2004-03-11" ...
##  $ CO.GT.      : num  1.97 2.31 2.9 2.74 2.47 ...
##  $ PT08.S1.CO. : num  1316 1265 1309 1346 1372 ...
##  $ C6H6.GT.    : num  8.45 8.57 12.67 11.38 9.84 ...
##  $ PT08.S2.NMHC.: num  912 880 1036 1010 951 ...
##  $ NOx.GT.     : num  132 150 181 188 150 ...
##  $ PT08.S3.NOx.: num  1167 1233 1053 978 999 ...
##  $ NO2.GT.     : num  109 103 120 120 112 ...
##  $ PT08.S4.NO2.: num  1546 1551 1651 1614 1608 ...
##  $ PT08.S5.O3. : num  1096 923 1121 1269 1241 ...
##  $ T           : num  12 9.8 11.8 13.4 16.4 ...
##  $ RH          : num  54.9 64.4 49.6 49.9 47.6 ...
##  $ AH          : num  0.766 0.778 0.667 0.734 0.848 ...
```
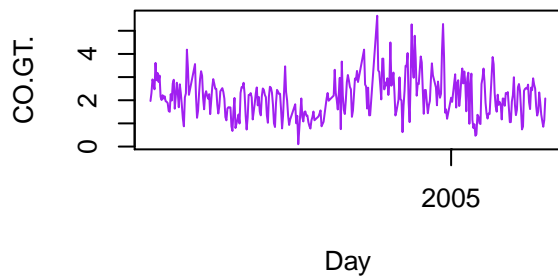
```
missmap(daily_data, main = "Missing values vs observed")
```
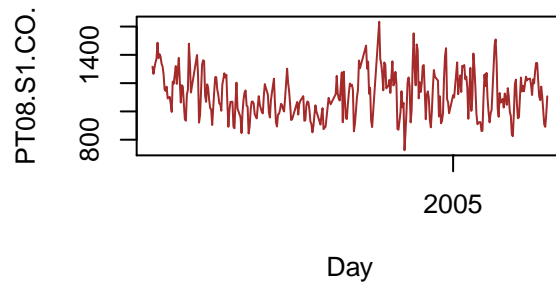
Missing values vs observed
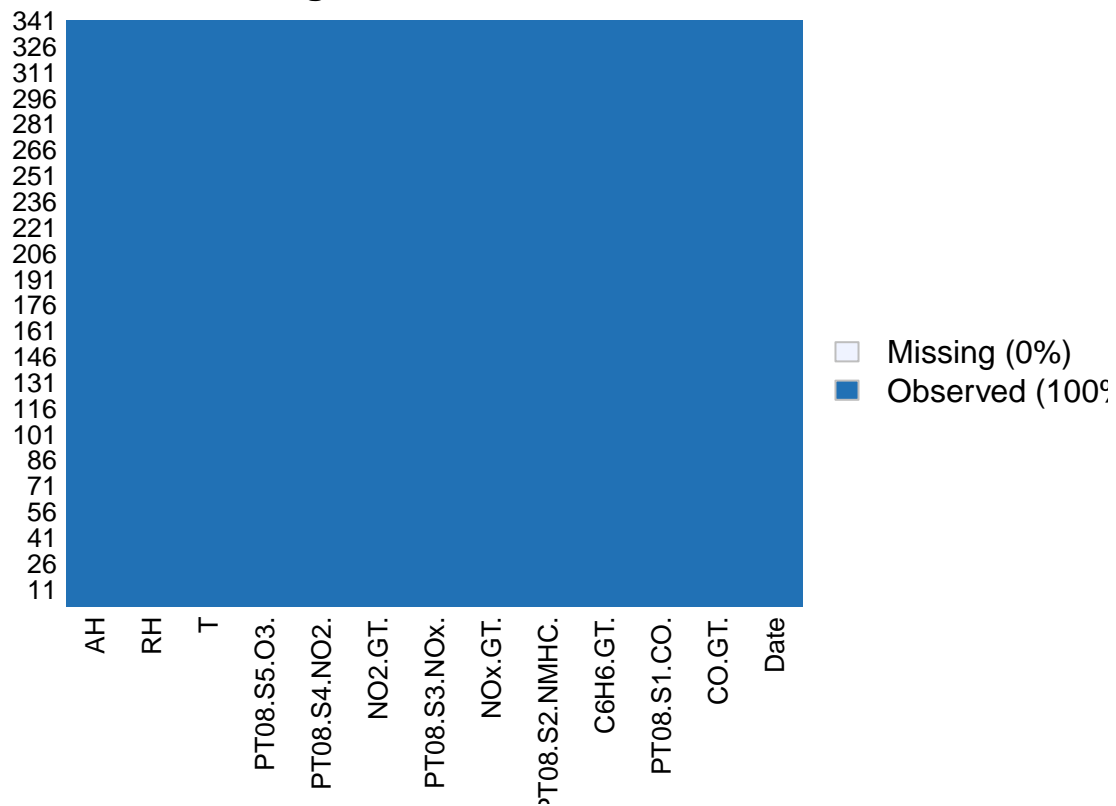
Since our data is all observed now, we can start plotting it to visualize what our data set looks like.
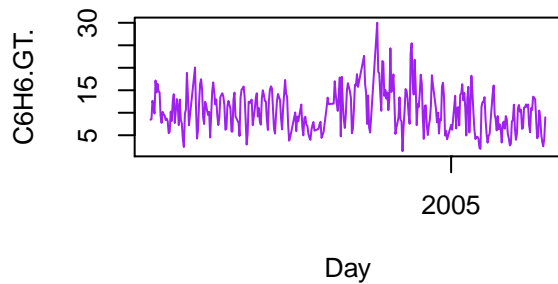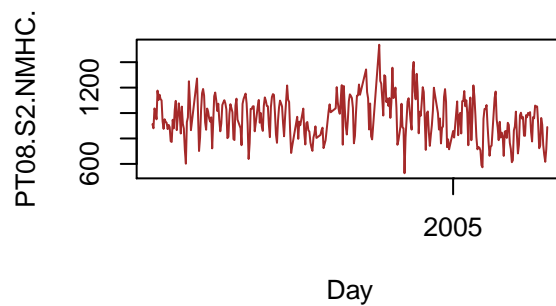

5

NOx.GT. concentration
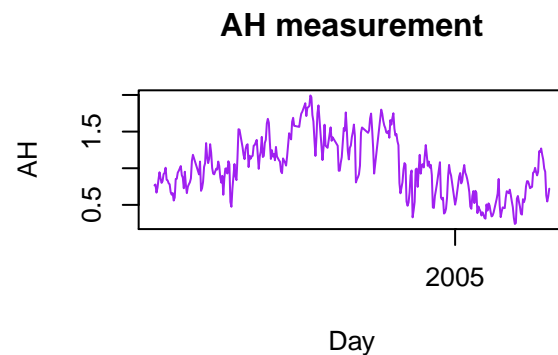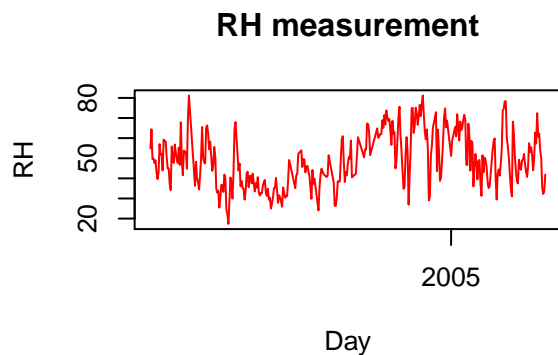
PT08.S3.NOx. concentration

NO2.GT. concentration

PT08.S4.NO2. concentration

PT08.S5.O3. concentration

T measurement

RH measurement

AH measurement
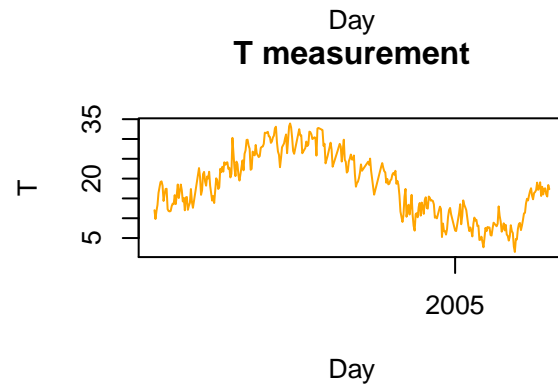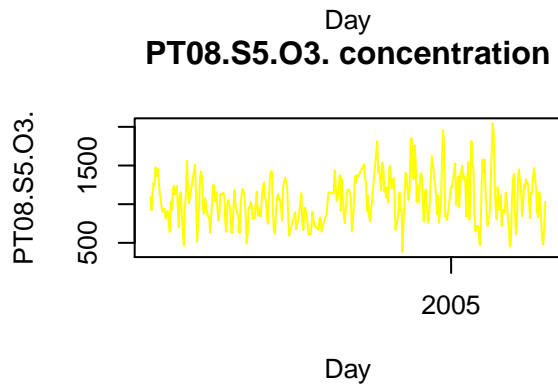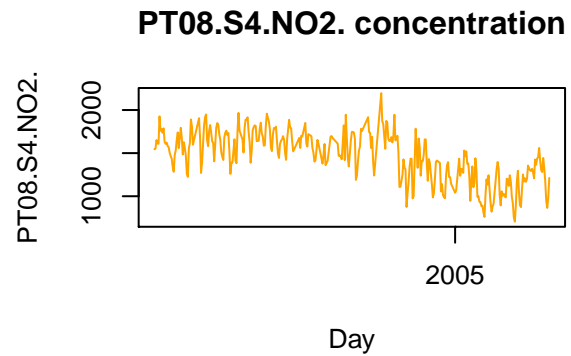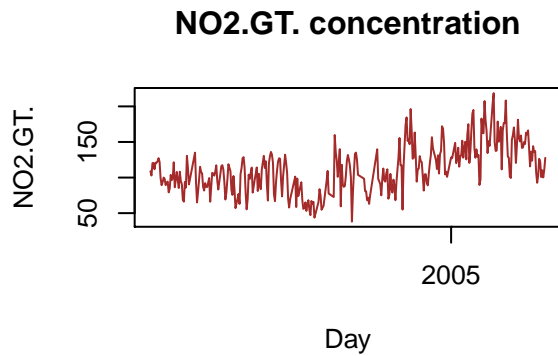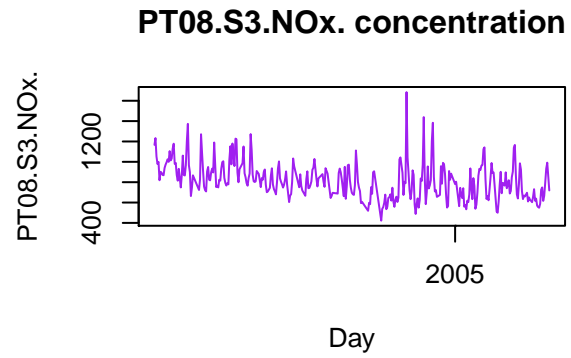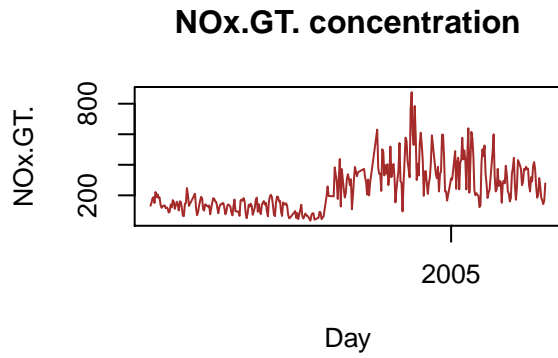
The variables starting with PT are the responses of the sensors measuring the concentration of the gas concerned. When investigating our data and by doing some research on air pollution, we found that the main air pollutants belong to the nitrogen oxides family (NOx). Thus, we wanted to see the relation between this gas concentration and the other ones. A linear regression was run between the NOx concentration and the

other gases.

```
summary(reg)
```

```
##
## Call:
## lm(formula = daily_data$PT08.S3.NOx. ~ +daily_data$PT08.S1.CO. +
##     daily_data$PT08.S2.NMHC. + daily_data$PT08.S4.NO2. + daily_data$PT08.S5.O3. +
##     daily_data$T + daily_data$RH + daily_data$AH)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -182.43  -60.67  -12.99   32.68  736.42
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)              1499.58349   73.78298  20.324  < 2e-16 ***
## daily_data$PT08.S1.CO.     -0.30643    0.09141  -3.352 0.000894 ***
## daily_data$PT08.S2.NMHC.   -0.46111    0.10894  -4.233 2.99e-05 ***
## daily_data$PT08.S4.NO2.     0.51459    0.05633   9.135  < 2e-16 ***
## daily_data$PT08.S5.O3.     -0.26128    0.06138  -4.256 2.70e-05 ***
## daily_data$T               -4.07044    3.05090  -1.334 0.183057
## daily_data$RH              -0.70990    1.08830  -0.652 0.514660
## daily_data$AH            -263.09360   48.56193  -5.418 1.16e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 95.89 on 333 degrees of freedom
## Multiple R-squared:  0.714,  Adjusted R-squared:  0.708
## F-statistic: 118.8 on 7 and 333 DF,  p-value: < 2.2e-16
```

We will build the NOx concentration as a time series beacause it comes from sensor measurements, thus noise will be modeled accordingly. Since it is linearly related to the other gases (except for the temperature, relative humidity and absolute humidity that we will exclude from our study). We will focus on the NOx model.
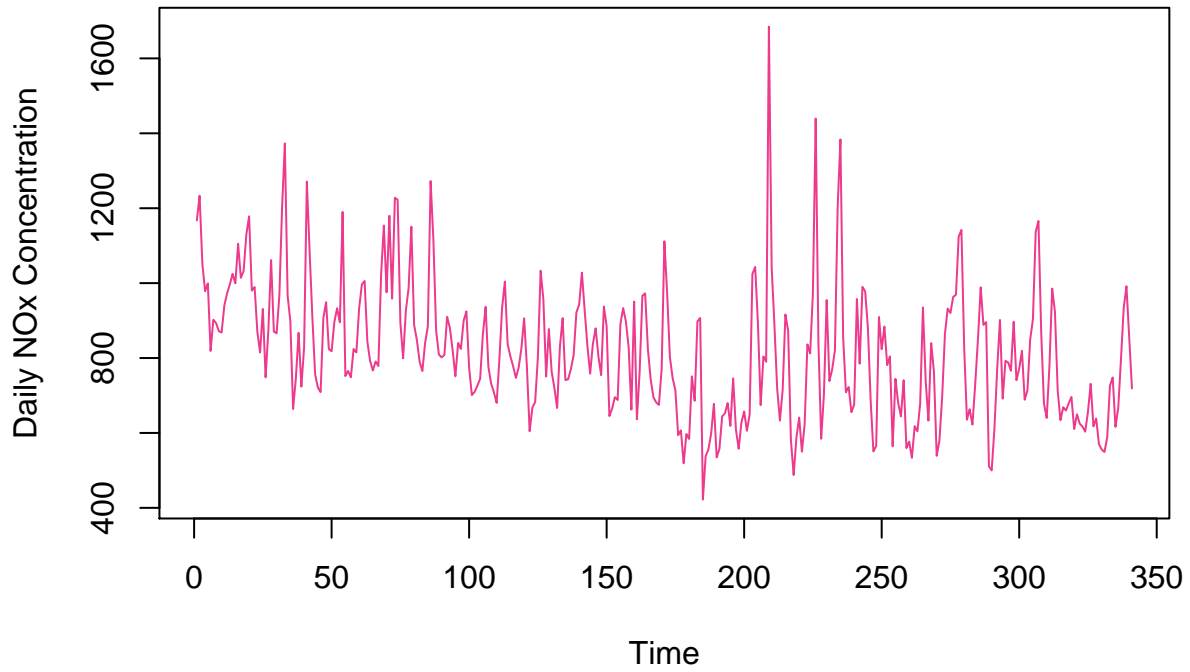
2) Data preprocessing

We count one PT08.S3.NOX observation per day for 341 days. We suppose that our time series follows an additive model :

$$D_t = S_t + T_t + X_t$$

Where $(S_t)_t$ is the seasonality, $(T_t)_t$ the trend and $(X_t)_t$ is assumed to be stationary.

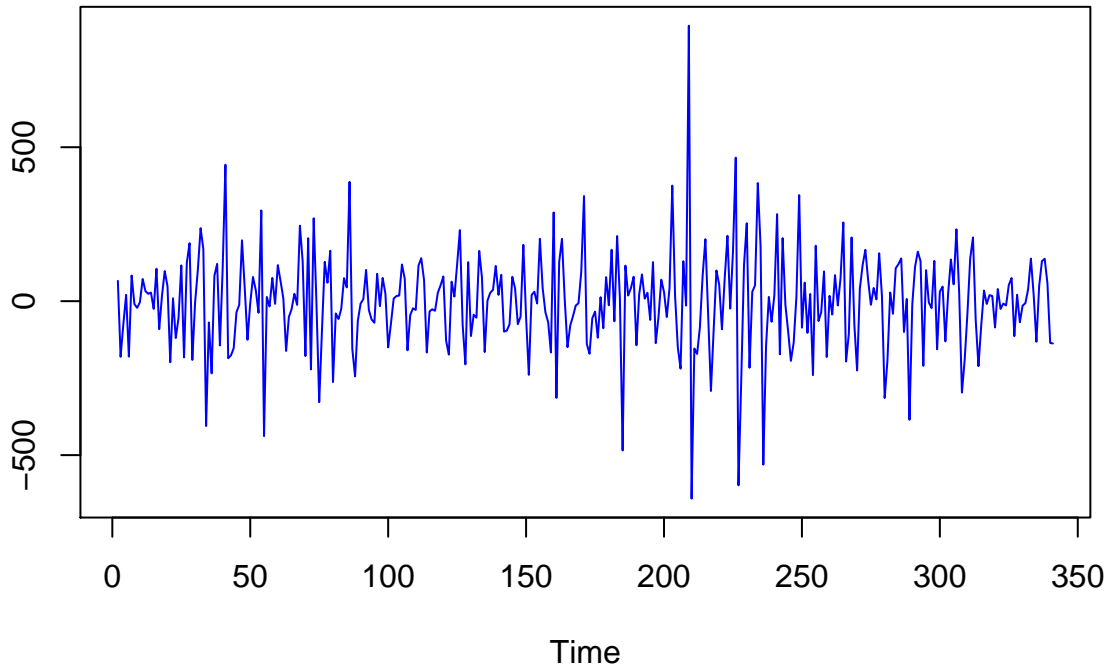We obtain this representation :

Before building any model, we have to stationarise it first by removing the seasonal and trend components.

$$X_t = D_t - T_t - S_t$$

So we use differencing :

$$D_t - D_{t-1}$$

We want to make sure of the stationarity of our series, so we use the Augmented Dickey-Fuller Test that tests the null hypothesis that a unit root is present in a time series sample.

We obtain these results :

```
## Warning in adf.test(ts_stat_PT08.S3.NOx.): p-value smaller than printed p-value
```

```
##
##  Augmented Dickey-Fuller Test
##
## data:  ts_stat_PT08.S3.NOx.
## Dickey-Fuller = -11.962, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```

Since our p-value is smalled than 0.01, we reject the null hypothesis with level of confidence of 99%

II)-Model fitting on the time series of interest

In order to test different models, we take out the last 10 most recent data that will be used for testing, the other observations will be used for the training.

Let $(X_t)_t$ be a centered second order stationary process. For $h \in Z$, we define :

- The autocovariance function :

$$\gamma_X(h) = Cov(X_t, X_{t+h}) = Cov(X_0, X_h) = E[X_0 X_h]$$

- The autorrelation function (ACF):

$$\rho_X(h) = \rho(X_t, X_{t+h}) = \frac{\gamma_X(h)}{\gamma_X(0)}$$

- The partial autocorrelation function (PACF):

$$\tilde{\rho}_X(h) = \rho_X(X_0 - \pi_{h-1}(X_0), X_h - \pi_{h-1}(X_h))$$

with the convention $\pi_0(X_1) = 0$ where $\pi_{h-1}(X_0)$ is the projection of X_0 on the linear span of $(X_1, X_2, ..., X_{h-1})$

1) MA model
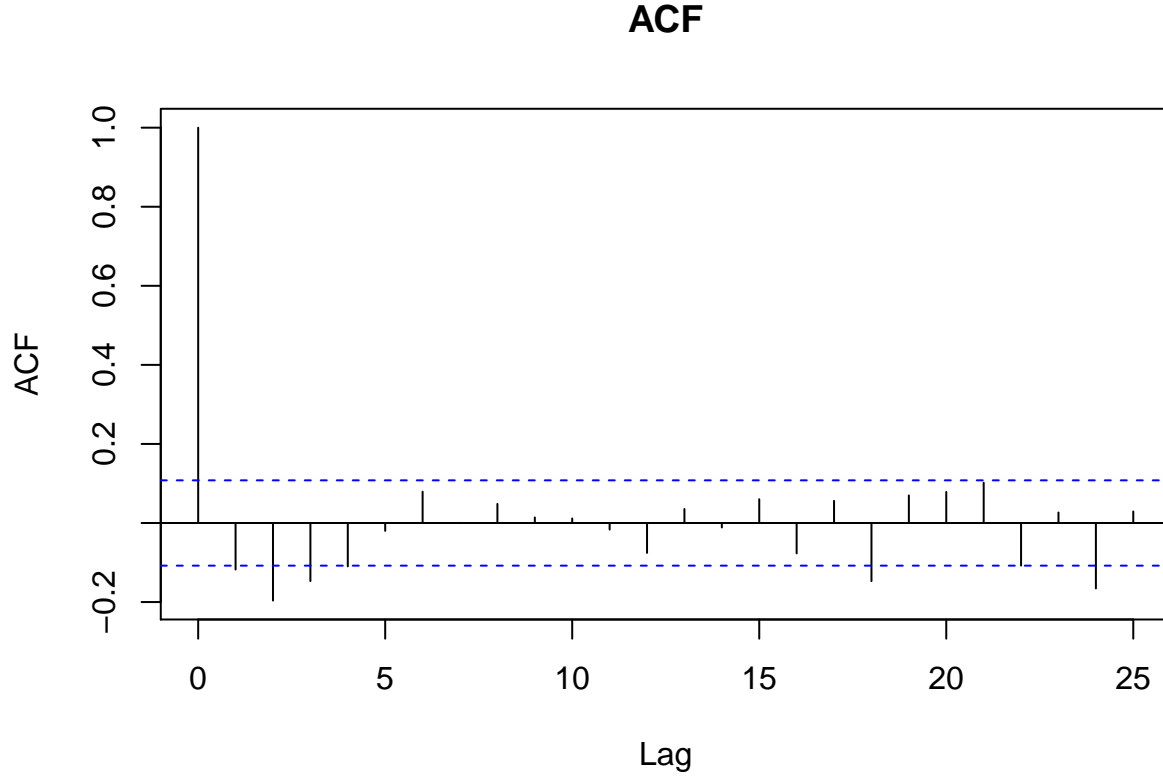
A MA(q) process, with $q \in N$, is a solution to the equation :

$$X_t = Z_t + \gamma_1 Z_{t-1} + ... + \gamma_q Z_{t-q}$$

with $t \in Z$ and $(Z_t)_t$ a white noise.

In order to find q, we use this MA(q) property :

$$\gamma_X(h) = 0 \ \forall h \geq p$$

We choose the q parameter accordingly to the last lag in the acf that is significantly non-null, outside the blue confident band.

ACF

We notice that we can not fit our data into a MA model, which is quite unexpected, because the data comes from sensor measurements so we have expected a strong noise presence. Thus, we will try other models.

2) AR model

A second model is the AR(p).
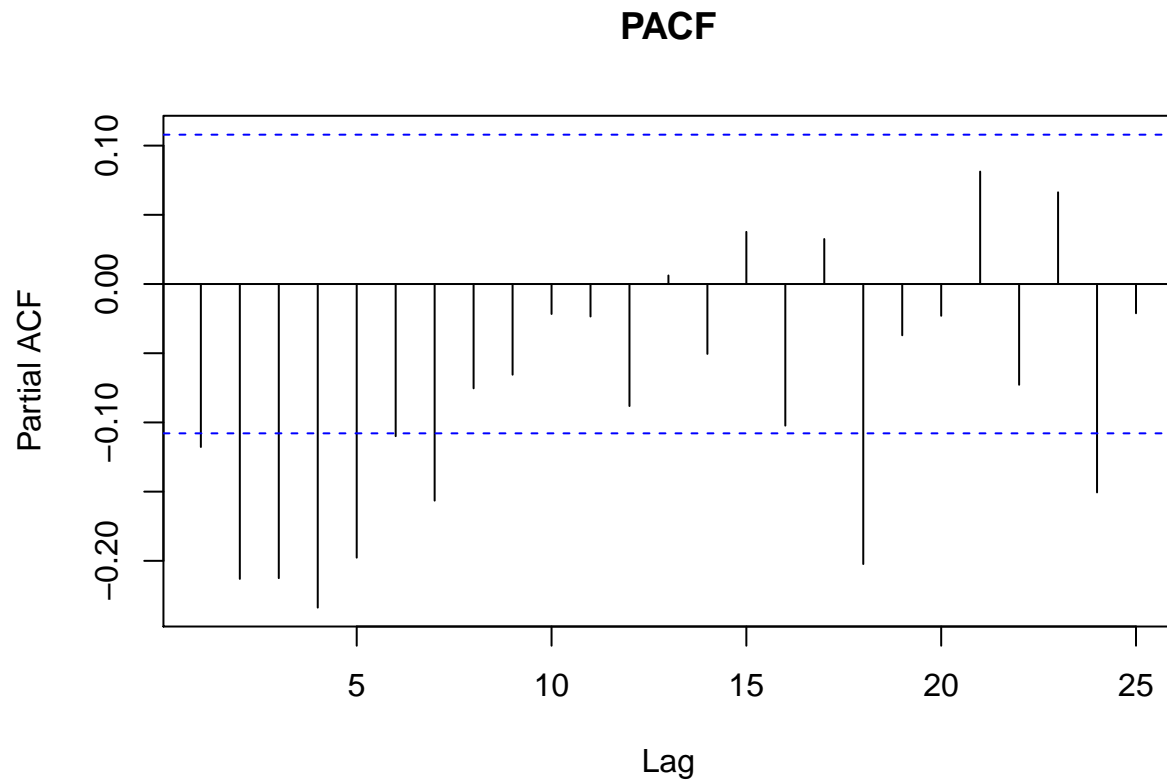
An AR(p) process, with $p \in N$, is a solution of the equation :

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + ... + \phi_p X_{t-p} + Z_t$$

We will use a pacf AR(p) proprety, equivalent to the acf MA(q) property :

$$\tilde{\rho}_x(h) = 0 \ \forall h > p$$

We obtain this pacf plot :

PACF

The PACF plot indicates a significant value at lag 5. Thus, we choose an AR(5) model.

```
##
## Call:
## arima(x = ts_stat_PT08.S3.NOx._train_set, order = c(5, 0, 0), include.mean = FALSE)
##
## Coefficients:
##           ar1      ar2      ar3      ar4      ar5
##       -0.2842  -0.3523  -0.3167  -0.2801  -0.1998
## s.e.   0.0539   0.0541   0.0546   0.0538   0.0539
##
## sigma^2 estimated as 20679:  log likelihood = -2108.17,  aic = 4228.34
```

So we have :

$$X_t = -0.2842X_{t-1} - 0.3523X_{t-2} - 0.3167X_{t-3} - 0.2801X_{t-4} - 0.1998X_{t-5}$$

Now we check that the residuals are likely white noise.

Standardized Residuals

ACF of Residuals

p values for Ljung–Box statistic
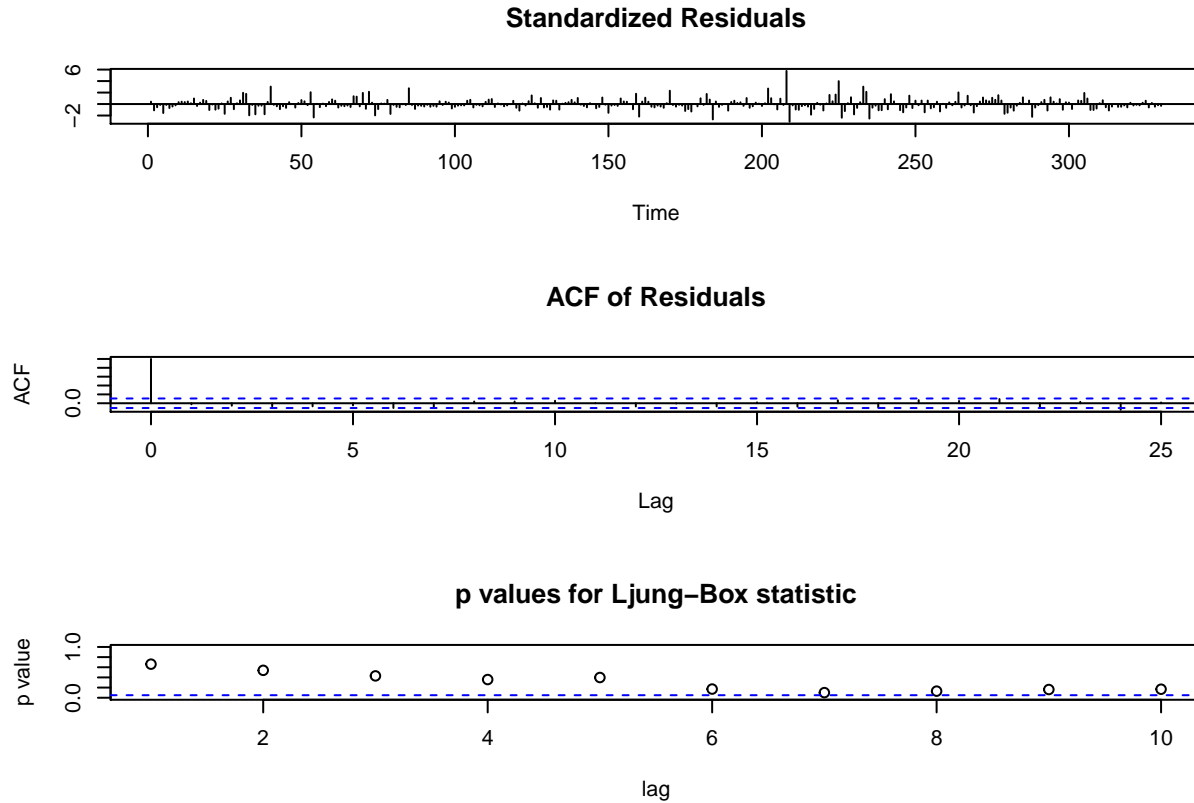
The ACF plot of residuals show no significant lags, so the AR(5) is likely a good representation of the series. Also, the p-values for Ljung-Box statistic are all greater than 0.05, so we cannot reject the hypothesis that the autocorrelation is different from 0. Therefore, the AR(5) model is an appropriate one.

3) ARMA model

Then, we're going to fit an ARMA model. An ARMA(p,q) time series is a process solution of the following model :

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + ... + \phi_p X_{t-p} + Z_t + \gamma_1 Z_{t-1} + ... + \gamma_q Z_{t-q}$$

with

$$\theta = (\phi_1, ..., \phi_p, \gamma_1, ..., \gamma_q)'$$

the parameters of the model and Z_t a white noise.

We want to find the ARMA's orders p and q, so we need to use some information criterion. An information criterion is an estimator of the relative quality of statistical models for a given set of data. Indeed, given a collection of models for the data, the information criterion estimates the quality of each model relative to each of the other models. So, we will use the results of the AIC to dertermine the orders of our ARMA model.

The AIC (Akaike Information Criterion) offers an estimate of the relative information lost when a given model is used to represent the process that generated the data. So the model with the minimum AIC is considered as the best model for the given data.

To observe the AIC values obtained we create a matrix with all AIC for p,q<6.

```
## Warning in arima(ts_stat_PT08.S3.NOx._train_set, order = c(i, 0, j),
## include.mean = FALSE): possible convergence problem: optim gave code = 1
```

12

```
## Warning in arima(ts_stat_PT08.S3.NOx._train_set, order = c(i, 0, j),
## include.mean = FALSE): possible convergence problem: optim gave code = 1

##           MA1       MA2       MA3       MA4       MA5
## AR1 4211.483 4206.408 4207.209 4208.772 4210.647
## AR2 4204.466 4205.355 4206.963 4208.054 4199.441
## AR3 4205.208 4207.144 4205.510 4210.136 4210.828
## AR4 4206.998 4208.875 4209.925 4206.425 4208.000
## AR5 4208.125 4210.959 4206.094 4213.566 4200.198
```

We can see that the minimum of AIC is obtained for the ARMA(5,5) model, so the best model based on the AIC is an ARMA(5,5).

We can now fit our ARMA(5,5) model.

```
## Warning in arima(ts_stat_PT08.S3.NOx._train_set, order = c(5, 0, 5),
## include.mean = FALSE): possible convergence problem: optim gave code = 1

##
## Call:
## arima(x = ts_stat_PT08.S3.NOx._train_set, order = c(5, 0, 5), include.mean = FALSE)
##
## Coefficients:
##          ar1     ar2      ar3      ar4     ar5      ma1      ma2     ma3
##       0.1440  1.2499  -0.6517  -0.5872  0.4571  -0.4865  -1.5923  0.9387
## s.e.  0.1565  0.1168   0.2250   0.0881  0.0736   0.1687   0.1037  0.3199
##          ma4      ma5
##       0.8175  -0.6513
## s.e.  0.0905   0.1880
##
## sigma^2 estimated as 18107:  log likelihood = -2089.1,  aic = 4200.2
```

We have

$$X_t = 0.144X_{t-1} + 1.2499X_{t-2} - 0.6517X_{t-3} - 0.5872X_{t-4} + 0.4571X_{t-5} +$$

$$Z_t - 0.4865Z_{t-1} - 1.5923Z_{t-2} + 0.9387Z_{t-3} + 0.8175Z_{t-4} - 0.6513Z_{t-5}$$

Our model is now fitted so we have to check if the residuals are likely white noise.

Standardized Residuals

ACF of Residuals

p values for Ljung–Box statistic
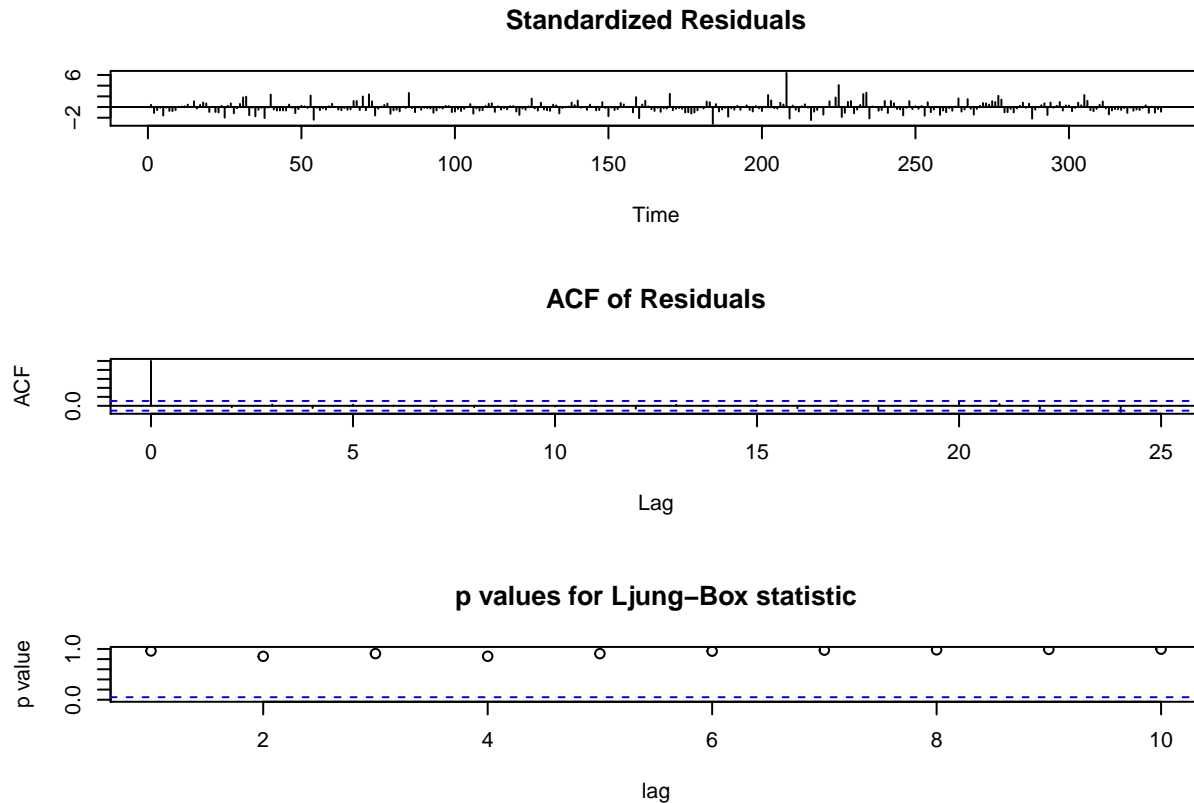
By looking at the ACF plot of residuals we can notice there is not significant lag outside the blue confident band which means that the selected model ARMA(5,5) is a good model to represent the serie. Moreover, the p-values of the Ljung-Box statistic test are all much greater than 5%, so we can't reject the null hypothesis of the autocorrelation different from 0. The ARMA(5,5) model is an appropriate model for our time series.
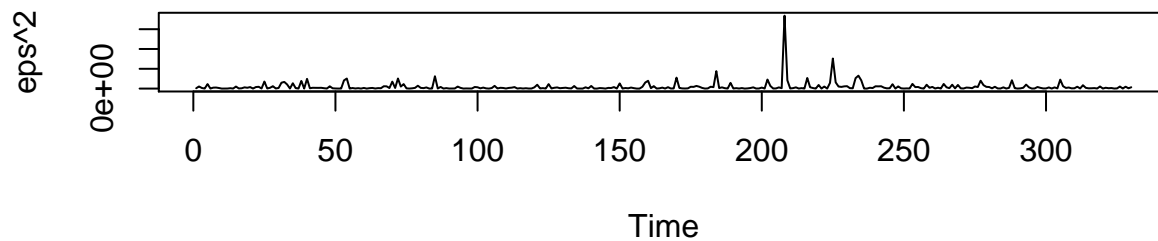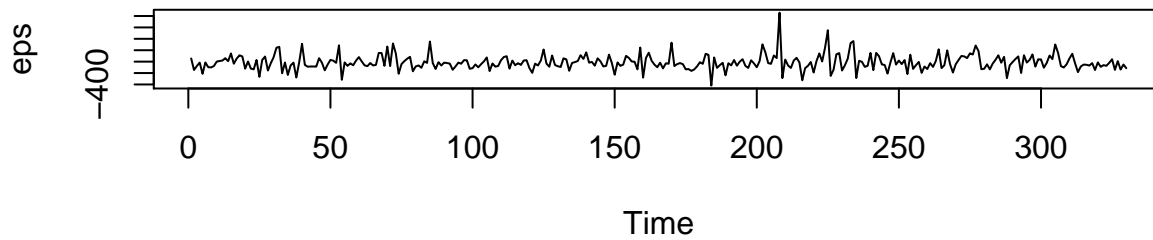
4) Residuals

We have fitted two models, and then e have to choose between these two models. In order to make this choice, we will look at two information criterions : the AIC previously used and the BIC a cirterion similar to the AIC but with a different penalty for the numbers of parameters.

```
##                df      AIC
## fit_ar_model    6 4228.336
## fit_arma_model 11 4200.198

##                df      BIC
## fit_ar_model    6 4251.131
## fit_arma_model 11 4241.988
```

The smallest AIC and the smallest BIC are bith given for the ARMA(5,5) model, so we are going to choose this one for the rest of the project.
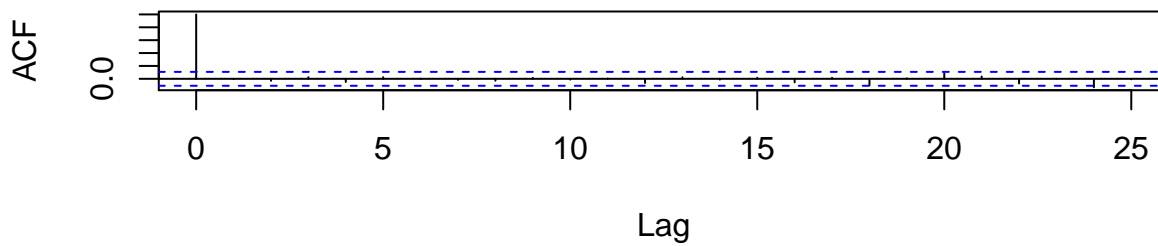
We want to check the normality of the residuals. We begin our analysis by plotting the residuals and the suqarred residuals :
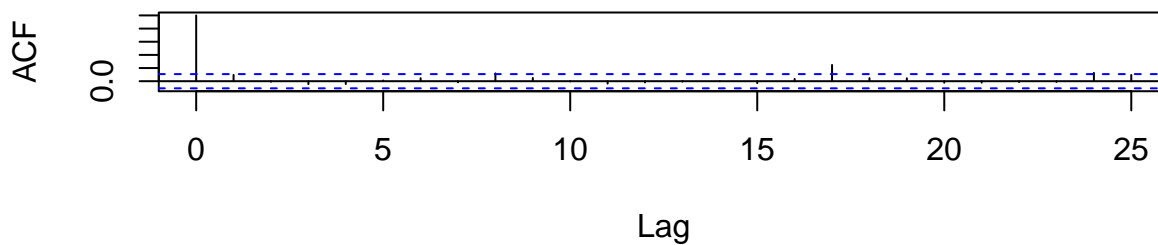
We notice an outlier at time 208 which means something unusual has may happened this day.
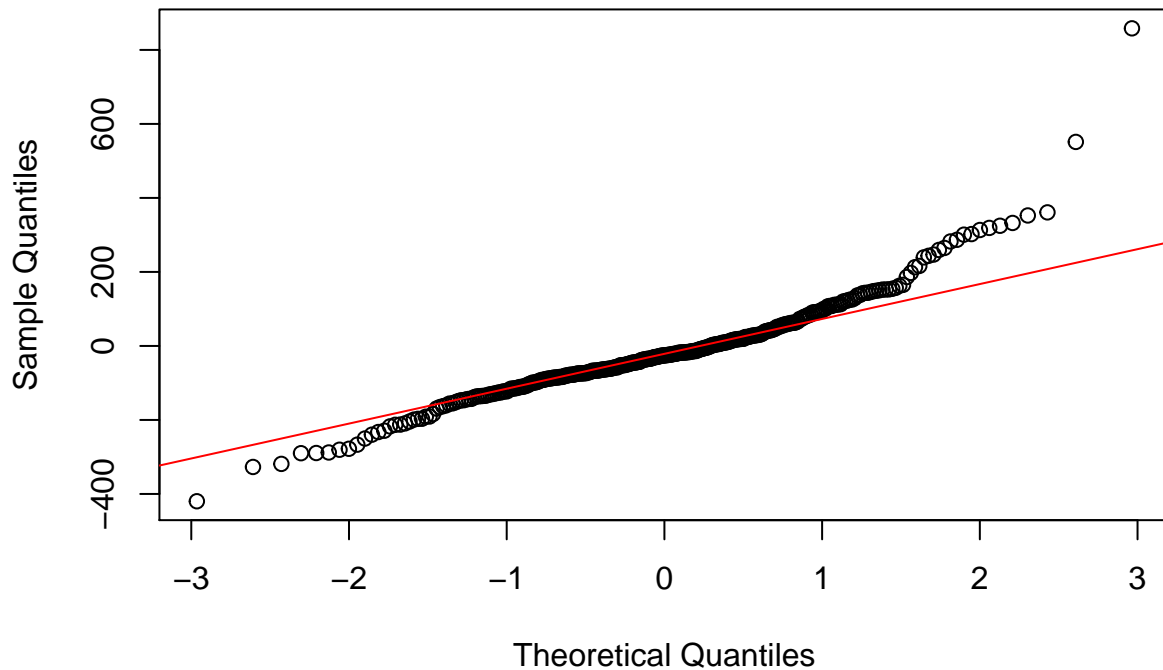
Then we plot the ACF graphs :

Series eps

Series eps^2

These plots show us there is no autocorrelation remaining in the residuals which means that the chosen model captures the pattern in the data effectively.

finally we plot the QQ plot of the residuals to observe their behavior :

Normal Q–Q Plot

By looking at this graph we can conlude that the residuals don't seem gaussian as there are too many extreme values, and thus fat tail. The relationship between sample quantiles and theoretical quantiles is not linear.

We can validate our hypothesis of the non normality of the residuals by performing a Jarque-Bera test. This test verify whether sample data have the skewness and kurtosis corresponding to a normal distribution.

```
##
##  Jarque Bera Test
##
## data:  eps
## X-squared = 604.13, df = 2, p-value < 2.2e-16
```

The p-value is smaller than 5% so we can reject the null hypothesis of the data normally distributed.

So we can reasonably conclude that the residuals are not gaussians and so the condition of the normal distribution of the error terms is not met.

5) GARCH model

6) Prediction intervals for the 10 most recent data

III)-Training on the times series of interest using explanatory times series

Yet, we will introduce other components, that we didn'tuse in the models previously. We do it in order to find better predictions and better confidence intervals.

1) Preprocessing

first, we have to stationarise our data with the same method as before. This means that we remove seasonal and trend components by using differencing.

```
library(data.table)
#On choisit le fichier qui contient les données

path = "/home/yousra/3A/cours/SeriesTemp/Air-Quality-Project/src/cleaningData_bis.R"
source(path)
```

```
library(KFAS)
library(ggplot2)
ggplot(daily_data,  aes(x=Date, y=PT08.S4.NO2.)) + geom_line()
```
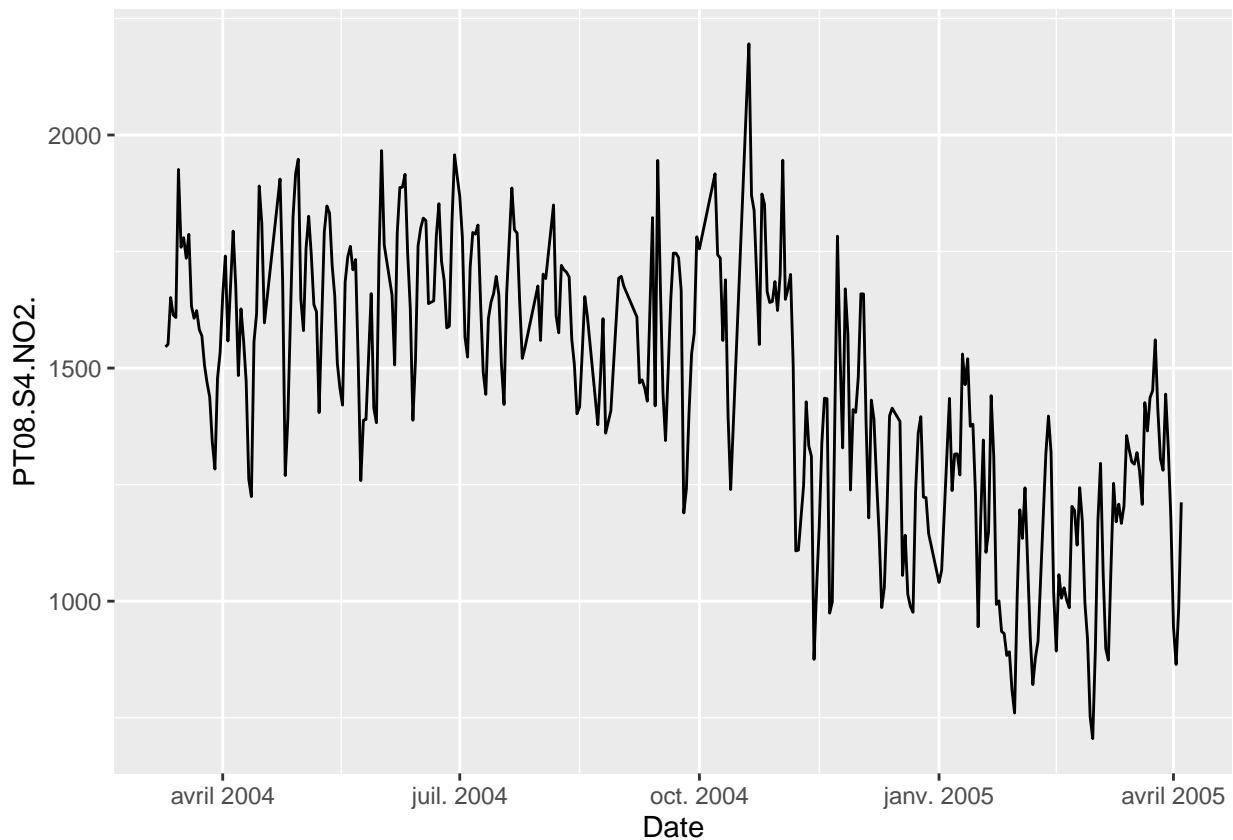

We see indeed that this data need to be stationarised

```
#we stationarise the dataset
ts_PT08.S4.NO2. = ts(daily_data$PT08.S4.NO2., start=1, frequency =1)
ts_stat_PT08.S4.NO2. = diff(ts_PT08.S4.NO2.)
daily_data$PT08.S4.NO2._stat <- c(0,diff(daily_data$PT08.S4.NO2.))
ggplot(data=daily_data, aes(x=Date, y=PT08.S4.NO2._stat)) + geom_line()
```
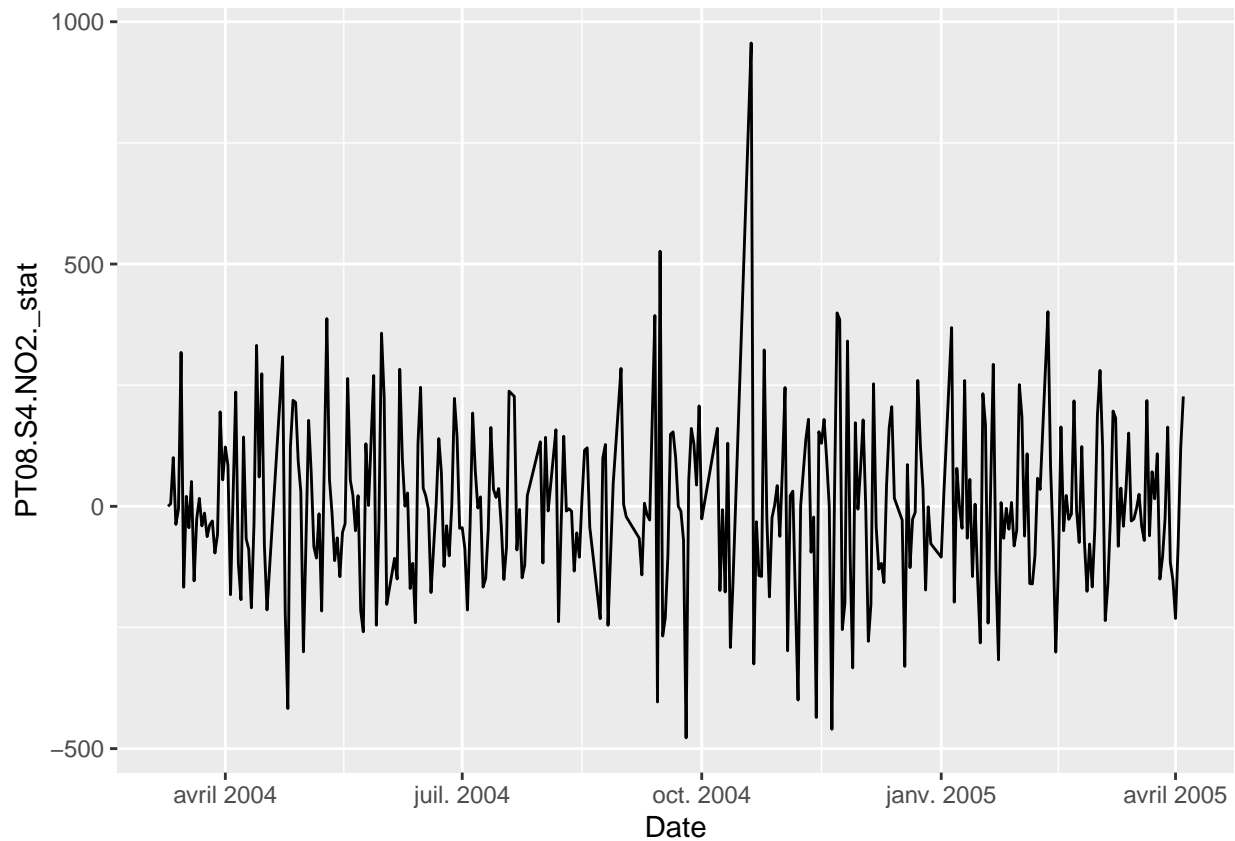
As before, we apply the Augmented Dickey-Fuller Test

```
library(tseries)
test_stationnarity = adf.test(ts_stat_PT08.S4.NO2.)
```
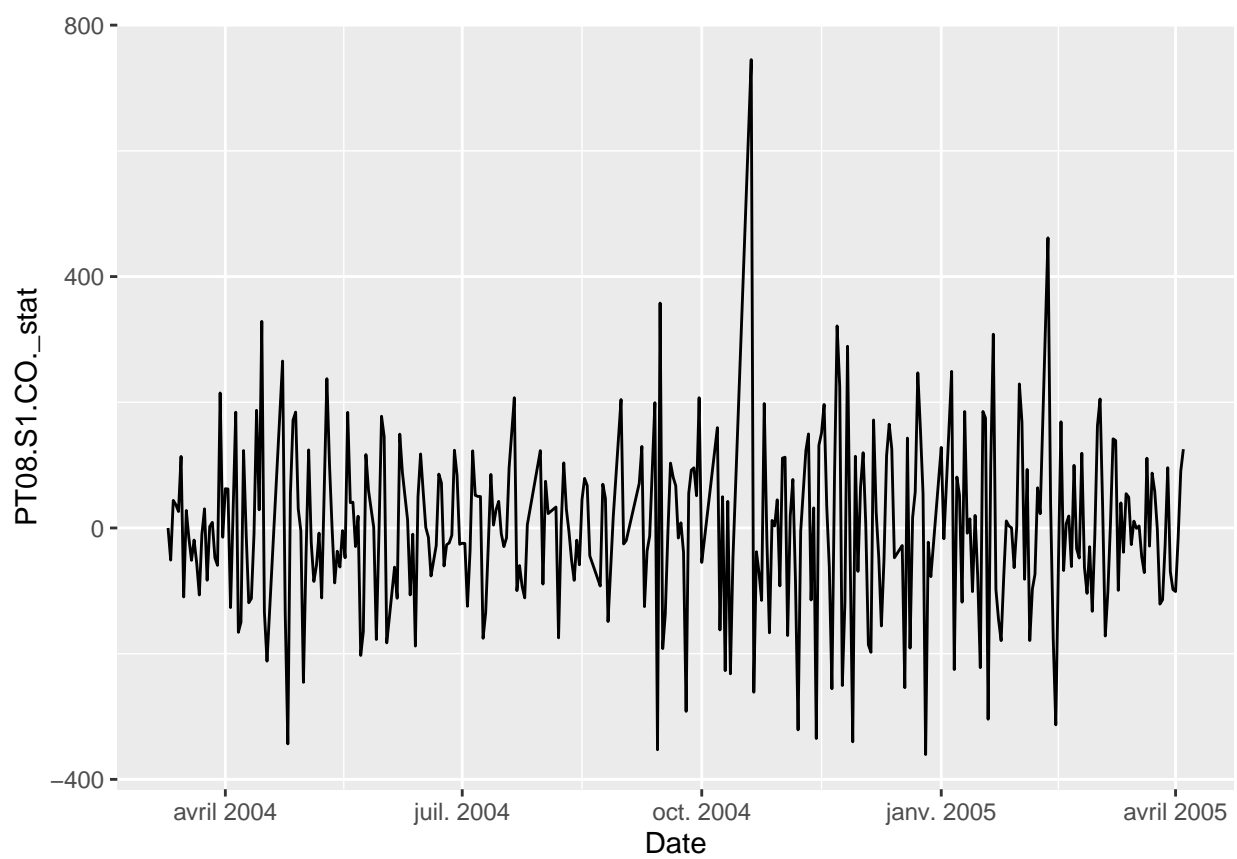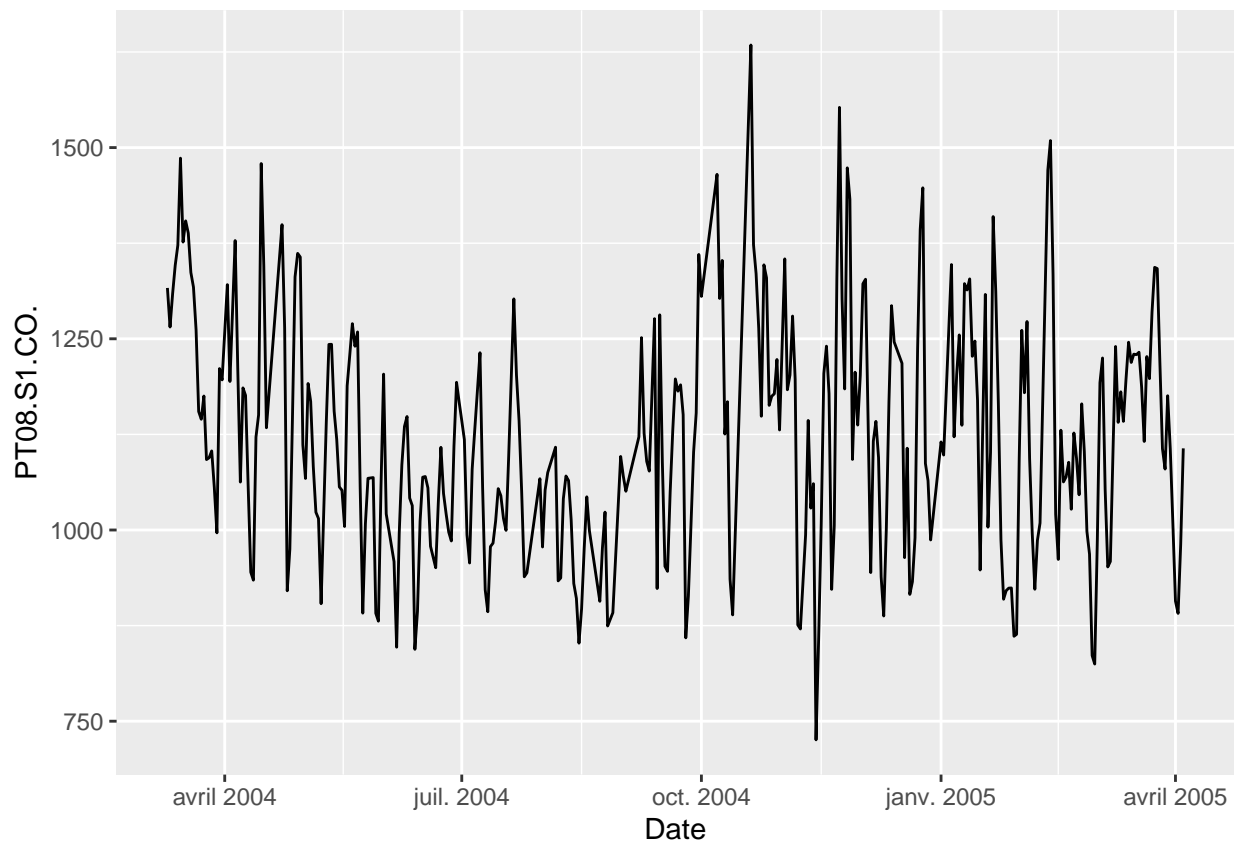
```
## Warning in adf.test(ts_stat_PT08.S4.NO2.): p-value smaller than printed p-value
```

```
print(test_stationnarity)
```

```
##
##   Augmented Dickey-Fuller Test
##
## data:  ts_stat_PT08.S4.NO2.
## Dickey-Fuller = -11.191, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```

We do it for each component with a name starting with PT.

To be sure that everything is fine, we use the Augmented Dickey-Fuller Test.

Here are the results of this tests after the stationarisation for each component.

```
library(tseries)
test_stationnarity = adf.test(ts_stat_PT08.S1.CO.)
```

```
## Warning in adf.test(ts_stat_PT08.S1.CO.): p-value smaller than printed p-value
```

```
print(test_stationnarity)
```

```
##
##   Augmented Dickey-Fuller Test
##
## data:  ts_stat_PT08.S1.CO.
## Dickey-Fuller = -11.123, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```
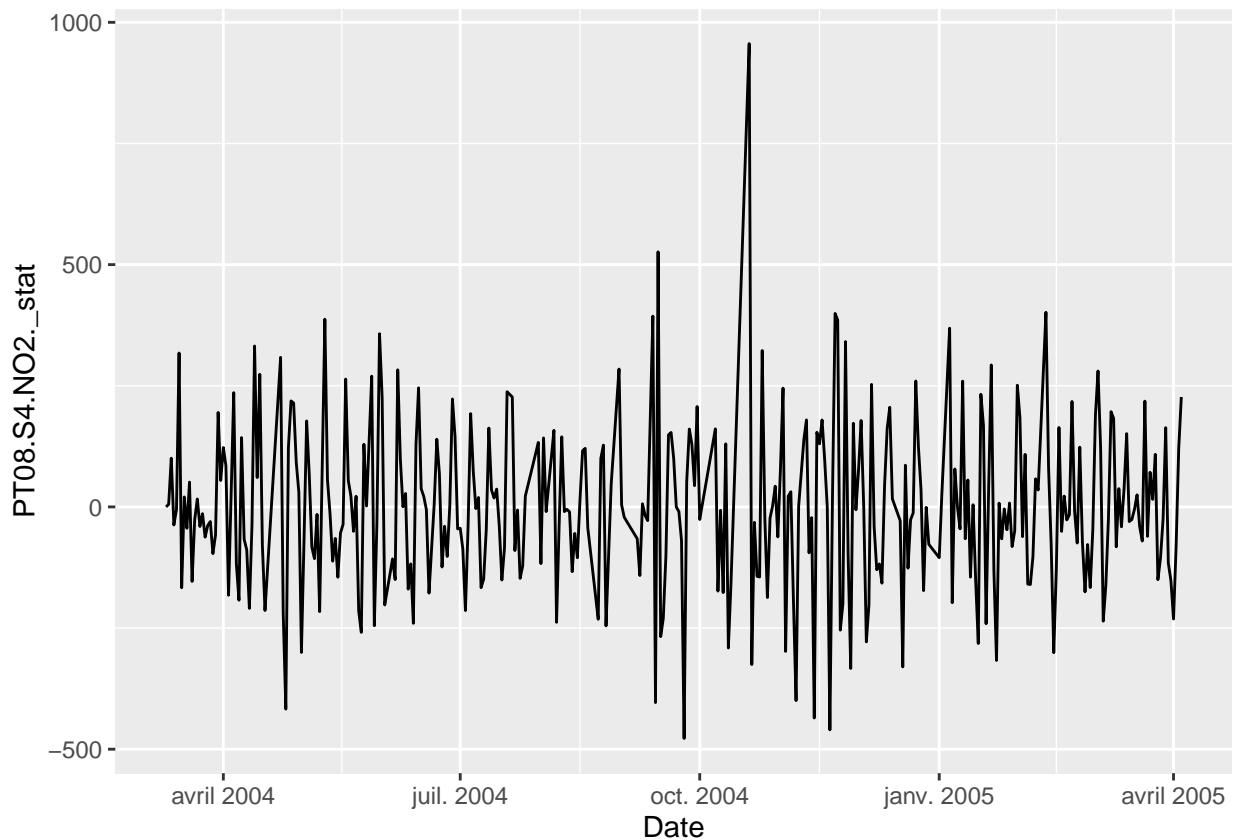


```
#
library(tseries)
test_stationnarity = adf.test(ts_stat_PT08.S2.NMHC.)
```

```
## Warning in adf.test(ts_stat_PT08.S2.NMHC.): p-value smaller than printed p-value
```

```
print(test_stationnarity)
```
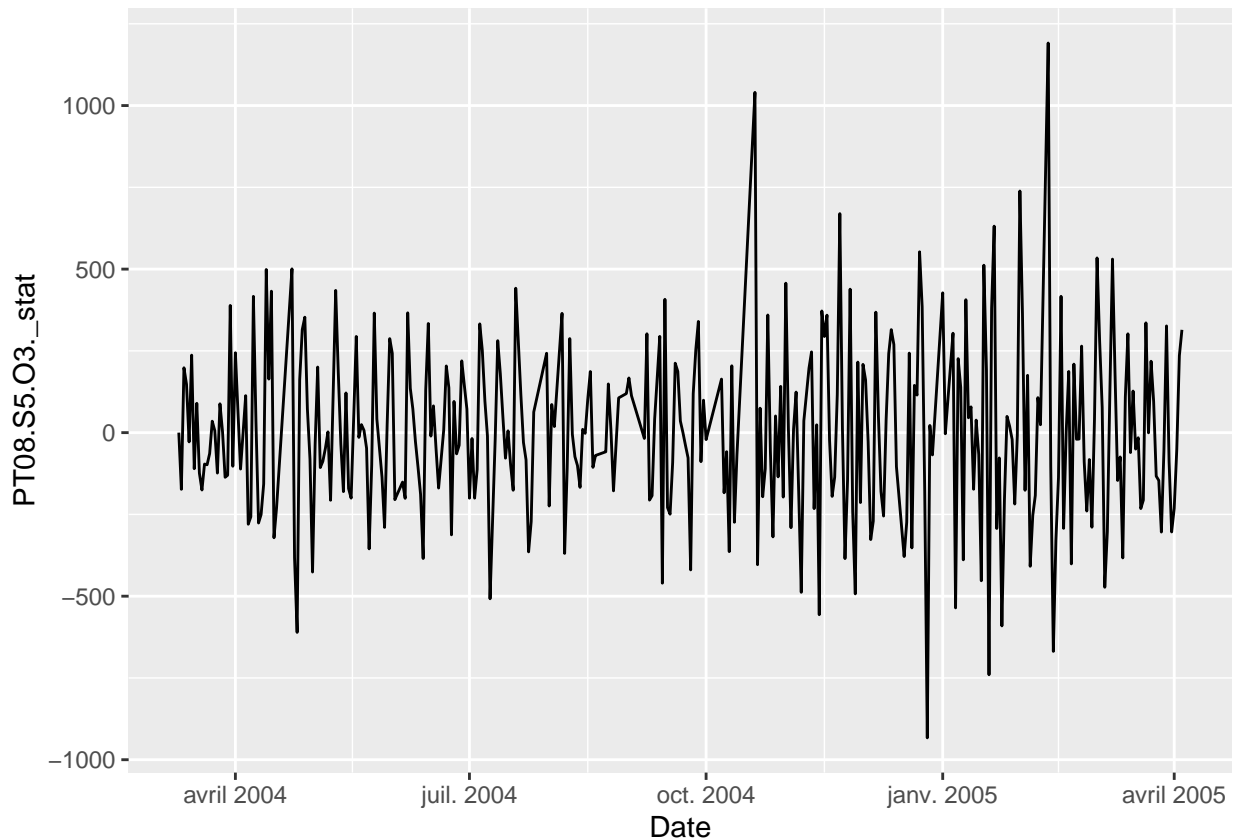
```
##
##   Augmented Dickey-Fuller Test
##
## data:  ts_stat_PT08.S2.NMHC.
## Dickey-Fuller = -11.266, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```

```
#
library(tseries)
test_stationnarity = adf.test(ts_stat_PT08.S5.O3.)
```

Warning in adf.test(ts_stat_PT08.S5.O3.): p-value smaller than printed p-value

```
print(test_stationnarity)
```

```
##
##   Augmented Dickey-Fuller Test
##
## data:  ts_stat_PT08.S5.O3.
## Dickey-Fuller = -11.178, Lag order = 6, p-value = 0.01
## alternative hypothesis: stationary
```

2) Time varying coefficients

Now we want to build a dynamical model thanks to the explanatory time series. We know that the order of the AR model was 5, so we will use 5 past values of the time series of interest for predicting the present value of the time series of interest too.

```
##           ytraining lag(ytraining, 1) lag(ytraining, 2) lag(ytraining, 3)
## [334,]  -14.251812         -6.708333          38.74275         138.21558
## [335,]   -6.708333         38.742754         138.21558          22.08333
## [336,]   38.742754        138.215580          22.08333        -131.90761
## [337,]  138.215580         22.083333        -131.90761               NA
## [338,]   22.083333       -131.907609               NA               NA
## [339,] -131.907609               NA               NA               NA
```

```
##         lag(ytraining, 4) lag(ytraining, 5) lag(ytraining, 6) lag(ytraining, 7)
## [334,]          22.08333         -131.9076                NA                NA
## [335,]        -131.90761                NA                NA                NA
## [336,]                NA                NA                NA                NA
## [337,]                NA                NA                NA                NA
## [338,]                NA                NA                NA                NA
## [339,]                NA                NA                NA                NA
##         lag(ytraining, 8) lag(ytraining, 9) lag(ytraining, 10)
## [334,]                NA                NA                 NA
## [335,]                NA                NA                 NA
## [336,]                NA                NA                 NA
## [337,]                NA                NA                 NA
## [338,]                NA                NA                 NA
## [339,]                NA                NA                 NA
```
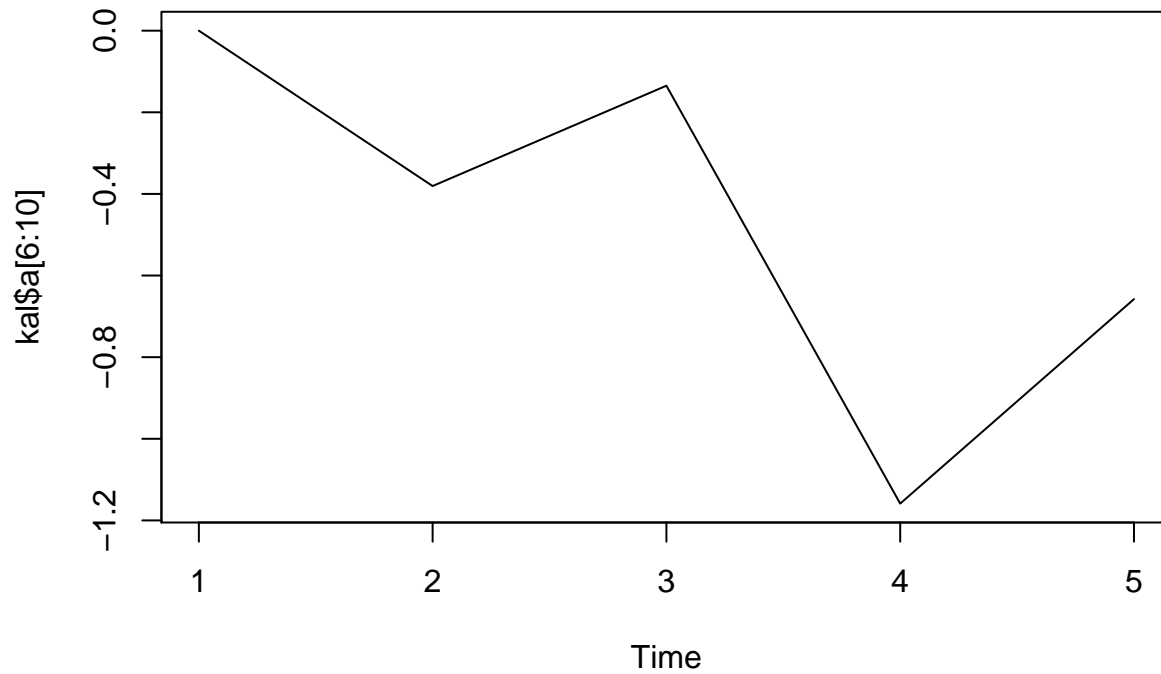
We use the SSModel function. Our target variable is ts_PTO8.S3.NOx. and our covariates are the other components starting with PT.

```
## [1] "data.frame"
```
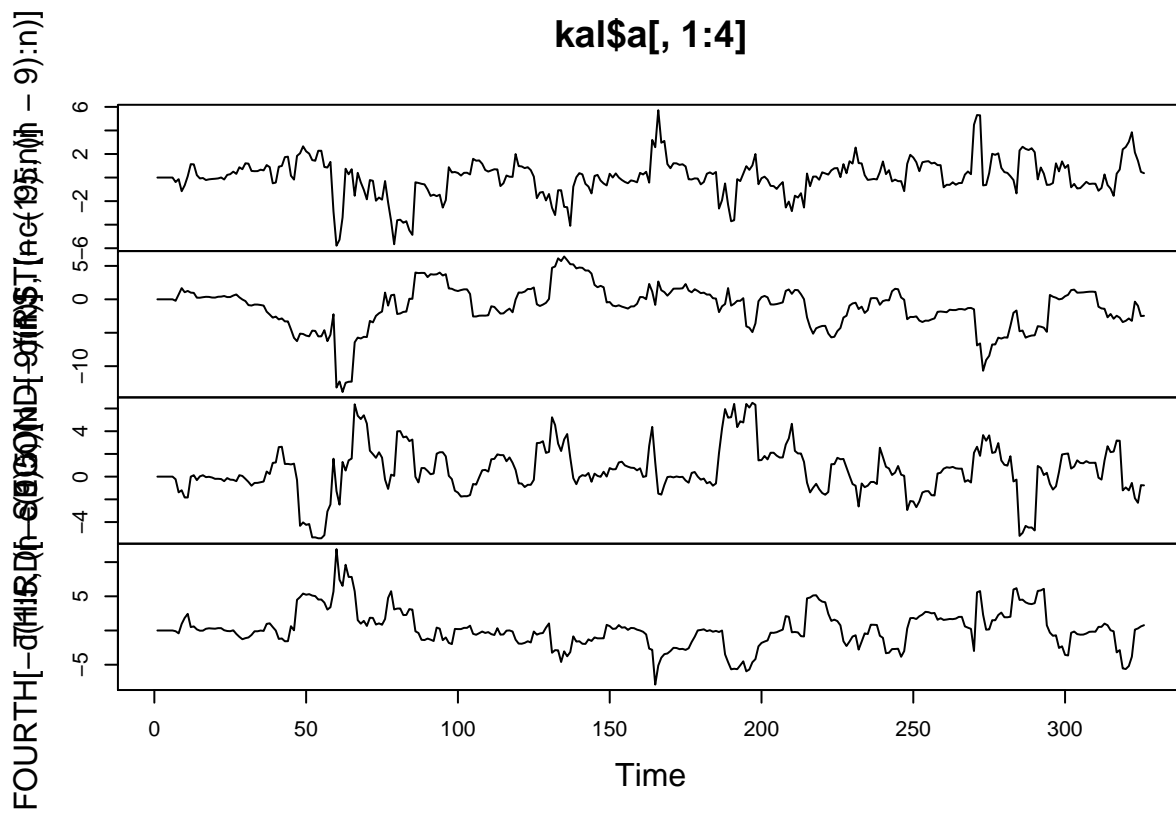
3) QLIK

We have our model so we can use it with a QFAS in order to tune the hyperparameter.

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1

## , , 1
##
##      [,1]
## [1,]    1
```
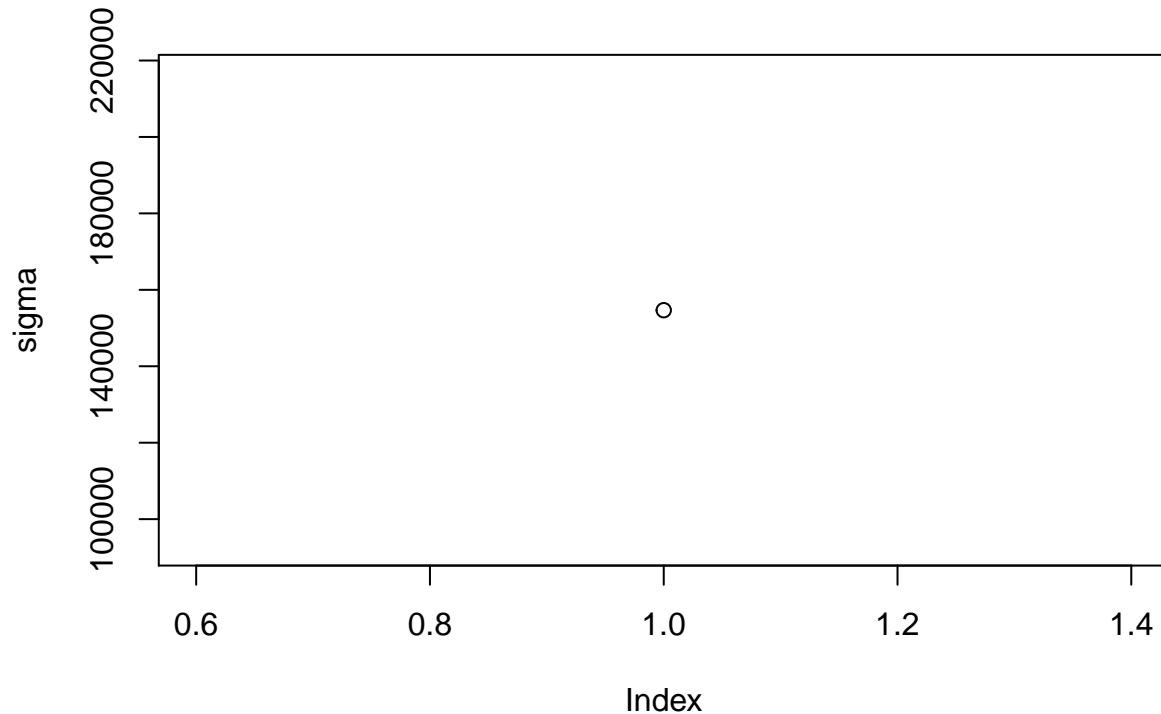
We have the covariance matrix of disturbance terms that is the identity. So we have for each component this result :

```
# the parameters for the 1 step prediction
plot.ts(kal$a[,1:4] )
```

kal$a[, 1:4]

and we find the variance

```
# the variances estimation as mean square errors for 10 lags
sigma <- colMeans((kal$m-Y[(n-5):(n-1),1])^2,na.rm = TRUE)    # remove a burn-in period to estimate the
plot(sigma)
```


It
seems that Variances can be seen as constant through time here.

4) Prediction

We have everything yet in order to do the prediction. The intervals of prediction can indeed be produced thanks to the use of the Kalman's recursion on the tuned dynamical model.

Let's explain why we use this : by contrast with the AR models, it is much more difficult to find the best possible (linear) prediction of an ARMA mode. Indeed, as soon as the MA part is non degenerate, the filter can have infinitely many non null coefficients.

One way to solve the problem is to consider ARMA model as a more general linear model called state space models. Those models have been introduced in signal processing and the best linear prediction can be computed recursively by the Kalman's recursion. How does this work ? A state space linear model of dimension r with constant coefficient is given by a system of space euqation and state equations of the form : $X_t = G^T Y_t + Z_T$ $Y_t = FY_{t-1} + V_t$ which are respectively the Space equation and the State equation, where (Z_t) and (V_t) are uncorrelated white noise with variance R and Q, $G \in Rr$, $F \in M(r,r)$ and $Y \in Rr$ is the random state of the system. The Kalman theorem says : In a state-space model with constant coefficients, if $\widehat{Y_0}$ and Ω_0 are well chosen, one can compute recursively

$$\widehat{X_n} = \pi_{n-1}(X_n)$$

$$R_n^L = E[(X_n - \widehat{X_n})^2]$$

$$\widehat{Y_n} = \pi_{n-1}(Y_n)$$

$$and \ \Omega_n = E[(Y_n - \widehat{Y_n})(Y_n - \widehat{Y_n})^T]$$

24

$$by\ the\ following\ recursion\ : \widehat{Y_{n+1}} = F\widehat{Y_n} + \frac{F\Omega_n G}{R_n^L} * (X_n - G^T \widehat{Y_n})$$
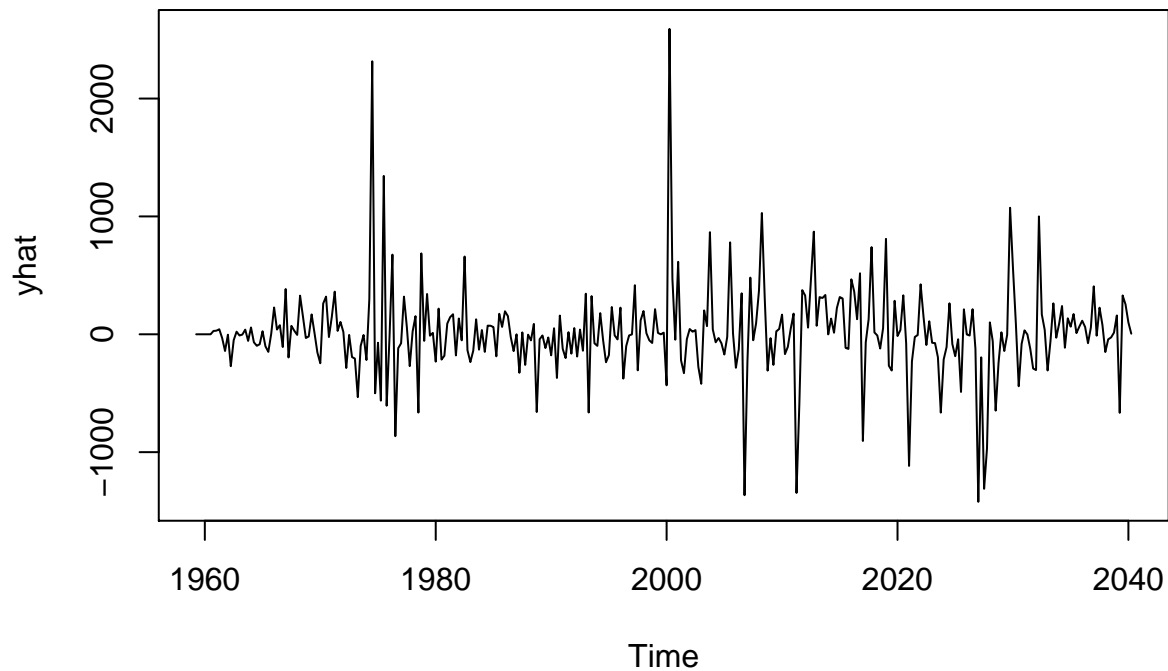
$$\widehat{X_{n+1}} = (G)^T * \widehat{Y_{n+1}}$$

$$\Omega_{n+1} = F\Omega_n F^T + Q - \frac{F\Omega_n G}{R_n} * G^T \Omega_n F^T$$

$$R_{n+1}^L = G^T \Omega_{n+1} G + R$$

Our target variable is ts_PTO8.S3.NOx. and our covariates are the other components starting with PT.

```
#the predictions in sample
yhat<-ts(kal$m[,1],frequency = 4, start = c(1959,2))
ts.plot(yhat)
```



```
# the estimation of the volatility, i.e. the condiitonal variance of the 1-step prediction

vol<-ts(kal$F[1,]-rep(1,n-10)+rep(sigma[1],n-10),frequency = 4, start = c(1959,2))
```

```
## Warning in kal$F[1, ] - rep(1, n - 10): la taille d'un objet plus long n'est pas
## multiple de la taille d'un objet plus court
```
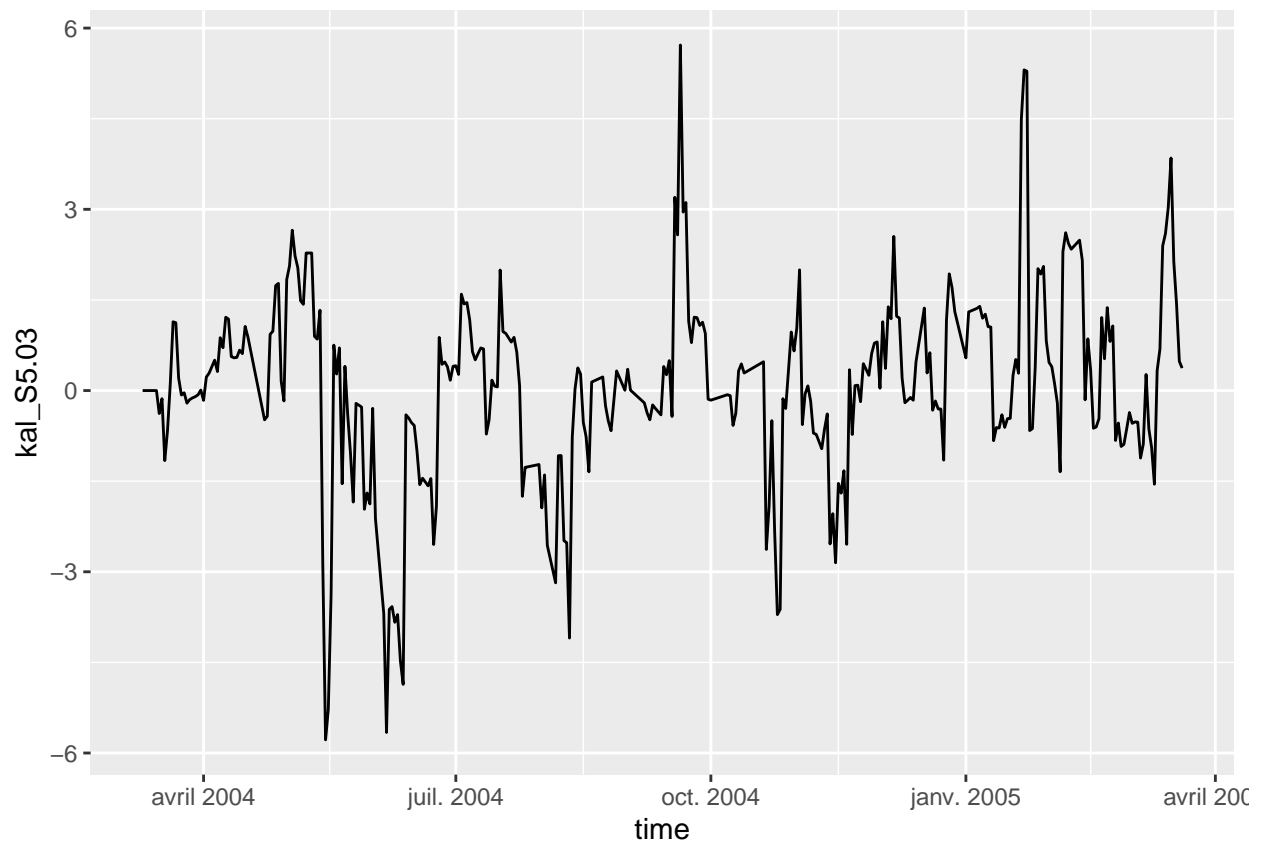
```
ts.plot(vol)
```
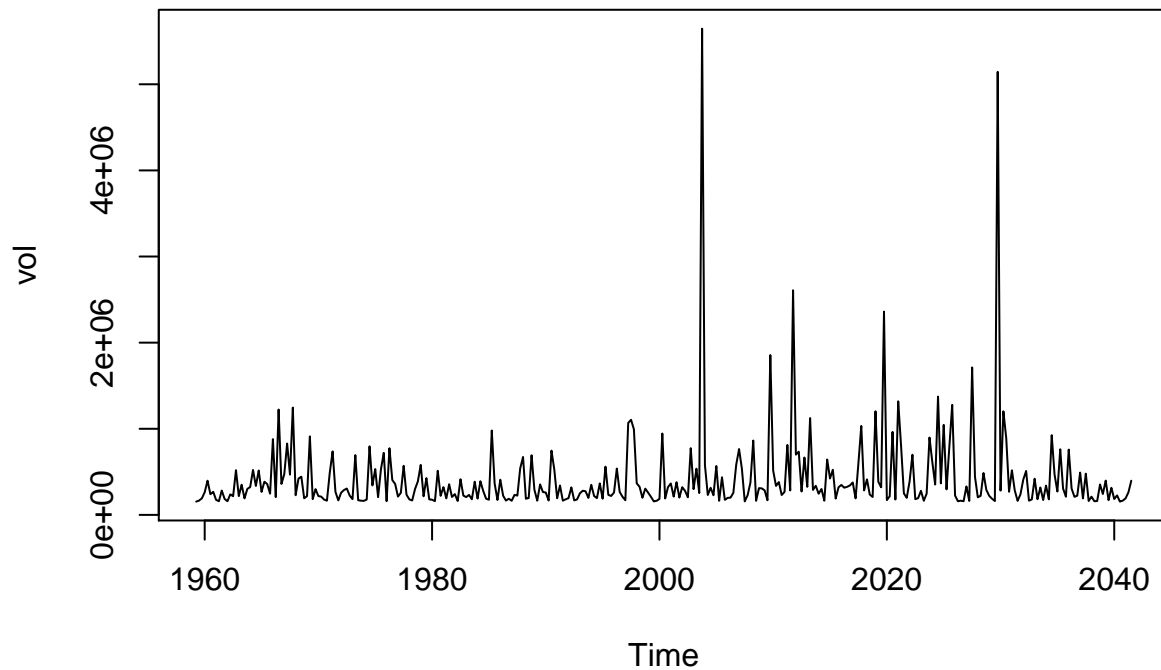
So we have a volatility that fluctuates a lot. Yet, we can focus on our intervals

```
kalman_frame <- data.frame(kal_S5.03 <- as.vector(kal$a[,1]),
                           kal_CO <- as.vector(kal$a[,2]),
                           kal_NMHC <- as.vector(kal$a[,3]),
                           kal_NO2 <- as.vector(kal$a[,4]),
                           time <- daily_data$Date[1:(n-14)])

# The parameters for the 1 step prediction
ggplot(data=kalman_frame, aes(x=time, y=kal_S5.03)) + geom_line()
```
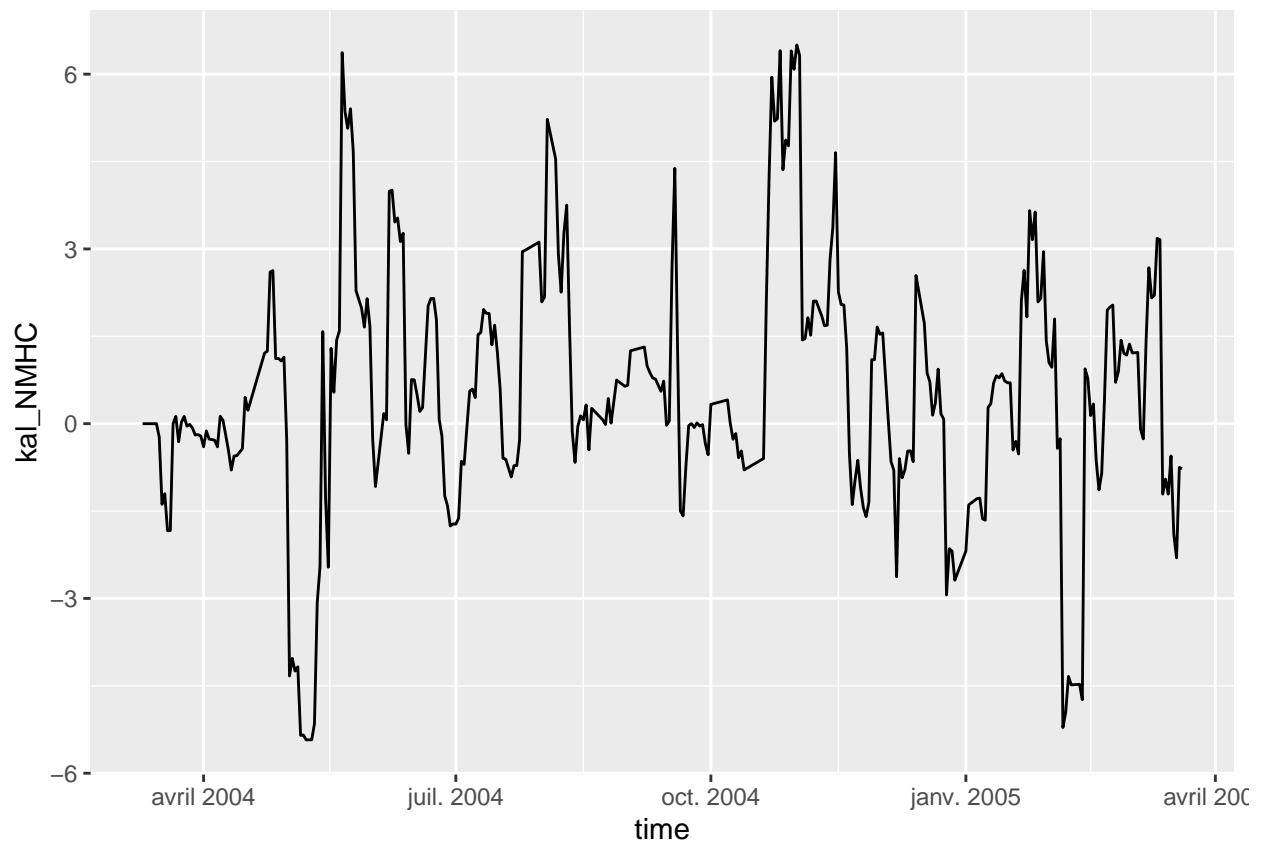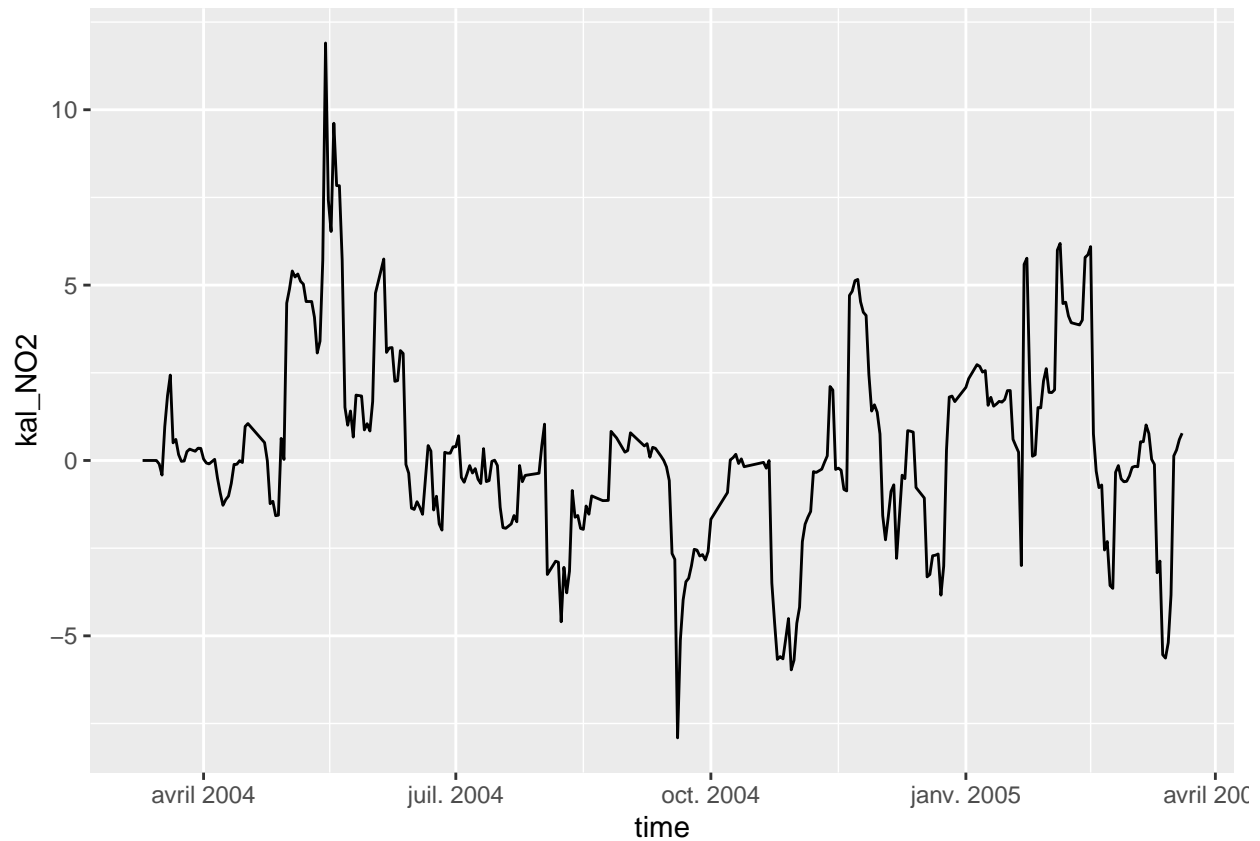
```
ggplot(data=kalman_frame, aes(x=time, y=kal_CO)) + geom_line()
```
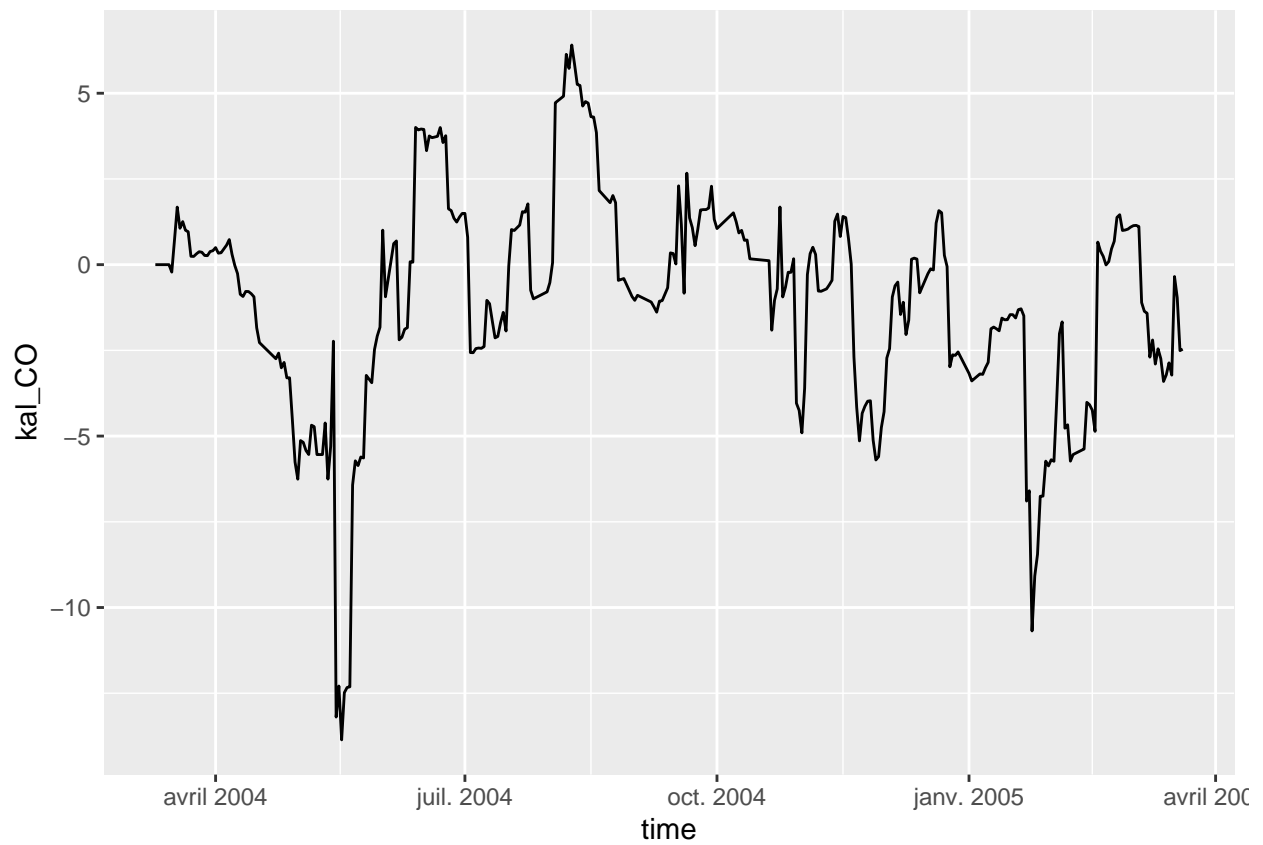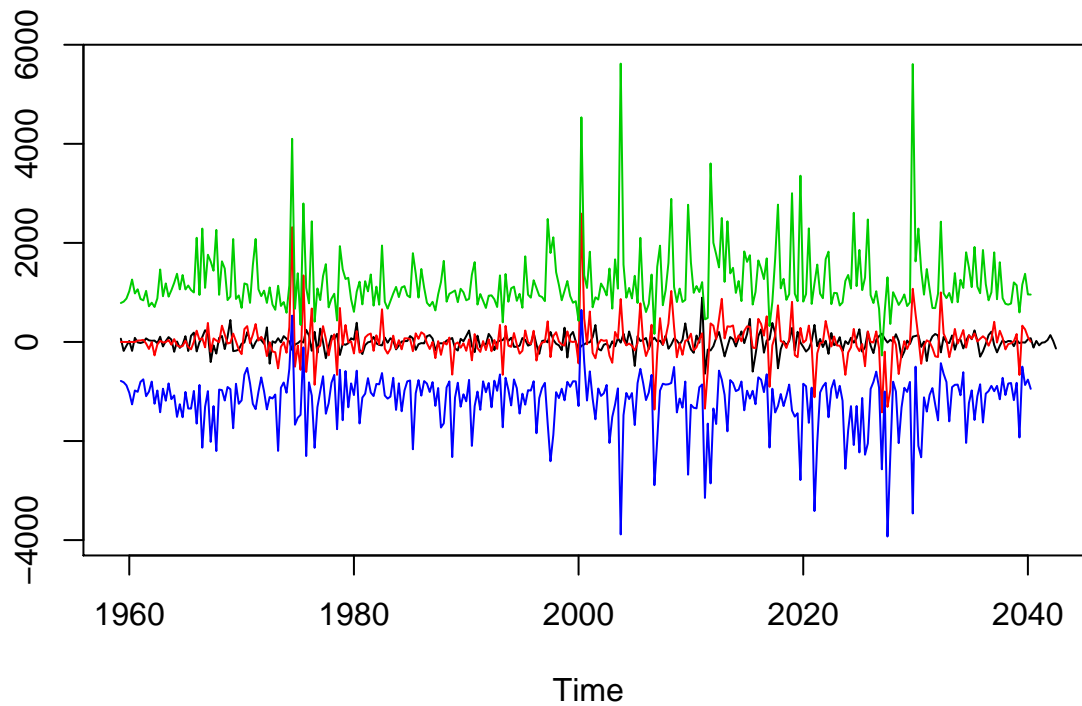
```
ggplot(data=kalman_frame, aes(x=time, y=kal_NMHC)) + geom_line()
```

```
ggplot(data=kalman_frame, aes(x=time, y=kal_NO2)) + geom_line()
```

```
# intervals of predictions in sample
upperCI<-ts(yhat+2*sqrt(vol),frequency = 4, start = c(1959,2))
lowerCI<-ts(yhat-2*sqrt(vol),frequency = 4, start = c(1959,2))
ts.plot(ytraining,yhat,upperCI,lowerCI,col=1:4)
```

```r
ts.plot(ts(cbind(ytraining[-(1:10)],yhat[-(1:10)],upperCI[-(1:10)],lowerCI[-(1:10)])),col=1:4)
```

```
## Warning in cbind(ytraining[-(1:10)], yhat[-(1:10)], upperCI[-(1:10)], lowerCI[-
## (1:10)]): number of rows of result is not a multiple of vector length (arg 2)
```
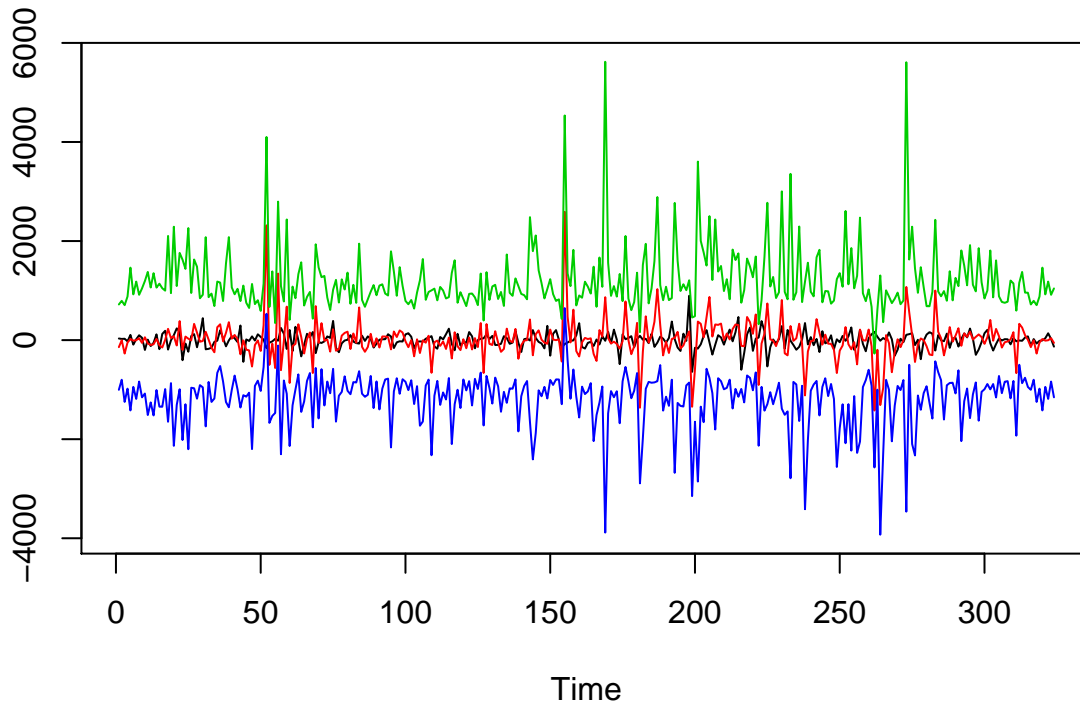

The predictions (in black) are quite good. Beside the upper and lower bound of the interval of confidence has big margins. And finally we have can product the prediction out of sample
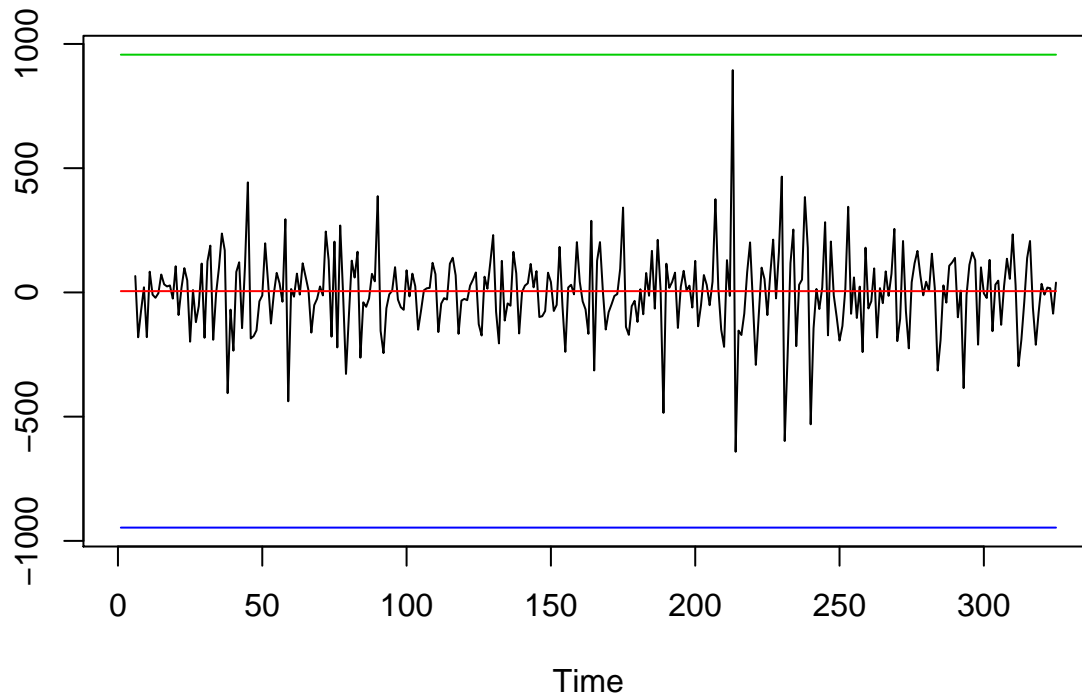
```r
# Prediction out of sample
ypred<-kal$m[n-15,]   # here we use different lags
volpred<-kal$F[1,(n-15)]-rep(1,10)+sigma # here we use the variances for different lags
ts.plot(cbind(Y[1:(n-15),1],as.numeric(ypred),as.numeric(ypred+2*sqrt(volpred)),as.numeric(ypred-2*sqrt
```

```
## Warning in cbind(Y[1:(n - 15), 1], as.numeric(ypred), as.numeric(ypred + :
## number of rows of result is not a multiple of vector length (arg 3)
```

Time

This last one is well in the interval of confidence. So we can say that this prediction is efficient

The Kalman's recursion has several advantages : - It is a recursve procedures - Each step requires the inversion of a scalar R_n and not the entire covariance matrix - The recursion can handle missing values nicely

However, there are also some drawbacks : - Tuning hyperparameters requires a non explicit minimization - The recursion can be instable

IV) Conclusion

In conclusion, this project allowed us to implement various time series models on R, namely AR, MA, ARMA and GARCH and also to get an insight on the Kalman recursion. It also allowed us to learn about versioning tools such as Github in order to easilty share our code among us and finally, to practice latex and markdown.