Wind energy time series analysis

# Time Series Analysis Project

Team Members:

- Bombini Nicola, nicola.bombini01@icatt.it  
- Donelli Francesco, francesco.donelli01@icatt.it  
- Giadone Giuseppe, giuseppe.giadone01@icatt.it  
- Iania Leonardo, leonardo.iania01@icatt.it

# 1. Exploratory data analysis

In this project we will study a dataset which consists of 1094 daily observations related to the wind energy production of a German plant from 01/01/2017 to 30/12/2019. As first thing, we read our dataset from Excel. Then, we make sure that there are not missing values or NA and transform our values as numeric.

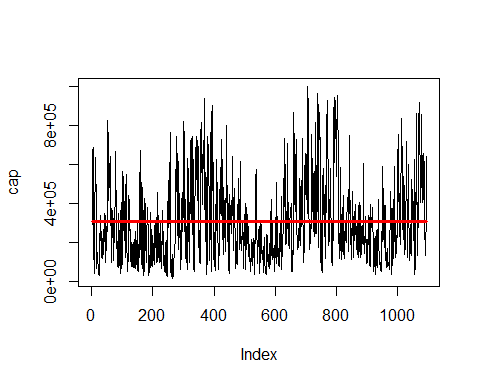
#import the dataset  
w <- read.csv("germany-wind-energy.csv")  
cap <- w$wind\_generation\_actual  
cap <- as.numeric(cap)  
n <- length(cap)  
range(cap)

## [1] 16482 998899

#We set a seed for our analysis  
set.seed(2023)

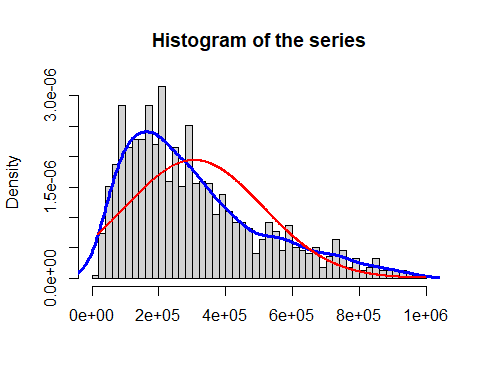
As you can see from above, the range is very large (16482, 998899) and we will have to take this into consideration later. First, we start our explanatory analysis by plotting the series. We notice that the series could be stationary since the function mean seems constant over time and the variance is independent of time, and we don’t notice any particular seasonal pattern

plot(cap,type="l")  
lines(1:n,rep(mean(cap),n),col="red",lwd=3)



We plot the histogram of the series to check for normality. We see that our distribution does not follow a normal distribution, since we see that it is skewed to the right, so we are not in an ideal situation, but we will go on with our analysis.

hist(cap,main="Histogram of the series",xlab="",freq=F,breaks=50)  
lines(density(cap),col="blue",lwd=3)  
zz=seq(min(cap),max(cap),length=n)  
f.zz=dnorm(zz,mean(cap),sd(cap))  
lines(zz,f.zz,col="red",lwd=2)

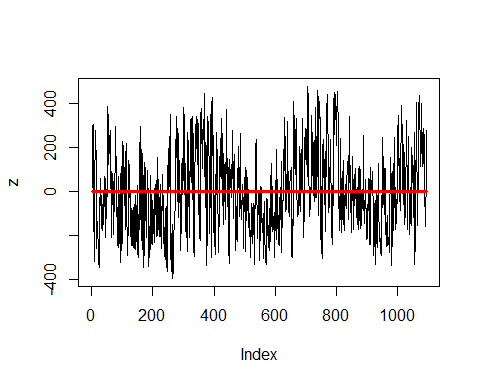


We then check again for normality of the observations in our dataset by comparing our distribution to the theoretical distribution of a normal random variable with the mean and standard deviation of our observations, after taking the square root of our values, to have a smaller range of observations, and we also take the mean adjusted values to have the series centered around 0.

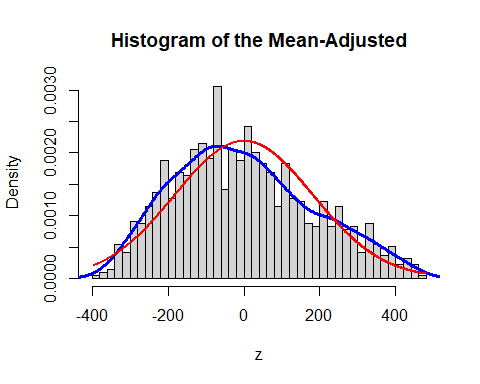
cap1=sqrt(cap)  
z=cap1-mean(cap1)  
n1=length(z)  
range(z)

## [1] -393.7540 477.3131

plot(z,type="l")  
lines(1:n1,rep(mean(z),n),col="red",lwd=3)



hist(z,main="Histogram of the Mean-Adjusted",breaks=50,freq = F)  
lines(density(z),col="blue",lwd=3)  
zz1=seq(min(z),max(z),length=n1)  
f.zz1=dnorm(zz1,mean(z),sd(z))  
lines(zz1,f.zz1,col="red",lwd=2)



We can now see from the series that the range is way smaller (-393.7540, 477.3131). From the histogram we see that the distribution does not seem normal, since it is again skewed to the right, even if less than before the transformation, but to be sure we check the normality through the Shapiro-Wilk test. (NB: this test is designed for non-dependent data, but our observations are recorded over time, hence there is dependence in time)

shapiro.test(z)

##   
## Shapiro-Wilk normality test  
##   
## data: z  
## W = 0.98057, p-value = 6.317e-11

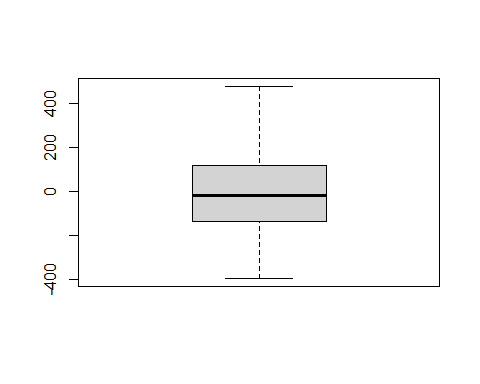
According to the p-value (way under 0.05), we reject the null hypothesis of normality. We then try with another test (again, designed for non-dependent data), the Jarque-Bera test:

jarque.bera.test(z)

##   
## Jarque Bera Test  
##   
## data: z  
## X-squared = 36.494, df = 2, p-value = 1.19e-08

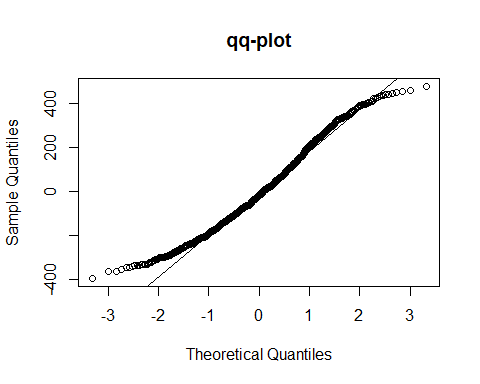
Again, according to the p-value (under 0.05) we reject the null hypothesis of normality. Taking a look at the boxplot, we can see that our distribution is very weakly asymmetric, but overall it’s not very bad.

boxplot(z)



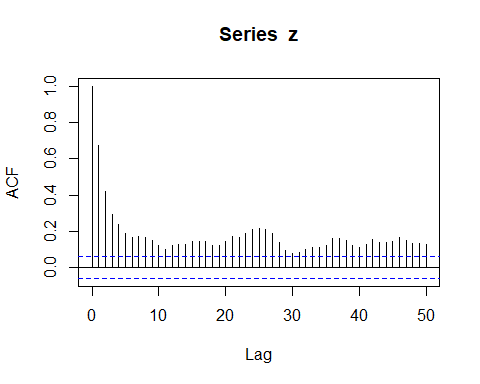
We then make one last check of normality, by looking at the qq-plot of our distribution against a normal one, and we can see that while in the middle of the distribution data seem to coincide, we see that the tails are far from normality

qqnorm(z,main="qq-plot")  
qqline(z)



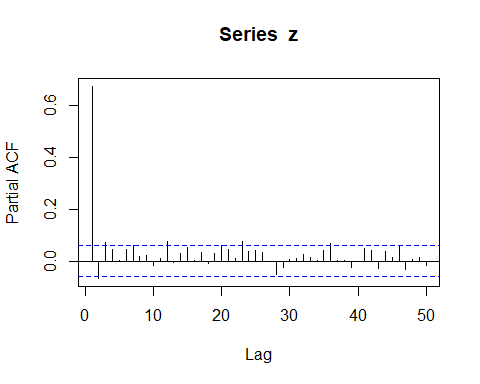
We then look at the autocorrelation function to see how data are correlated in time, and to see if there is an indication on the parameter to choose for a potential Moving Average model. We can’t have great indications about seasonality since we have daily observations, hence it’s difficult to spot potential monthly/quarterly… cycles. Looking at the plot, with a lag of 50 we can see that no tick goes under the blue band, hence no useful suggestion for a potential MA(q) model, and furthermore we can say that the observations have a relevant dependence at least on the previous 50 ones.

acf(z,lag=50)



We now take a look at the partial autocorrelation function, which checks the correlation between 2 observations located at a certain lag, given the observations between them as fixed. Moreover, it can be used as a tool to find the optimal parameter for an auto-regressive model. From the plot, we can see that he PACF value goes under the blue bands at lag 4, which means that the optimal number of parameters for an AR(p) model is lag-1, so 3.

pacf(z,lag=50)



# 2. AR model

## AR model specification

Let’s start our analysis by using the maximum likelihood estimation technique, considering an auto regressive model with 3 parameters (NB: we fit our model until n-10, since we will use the last observation as a comparison for our forecasted values):

AR3 <- arima(z[1:(n-10)],order=c(3,0,0),method='ML')  
  
(est.phi1=AR3$coef[1])

## ar1   
## 0.7244523

(est.phi2=AR3$coef[2])

## ar2   
## -0.1156331

(est.phi3=AR3$coef[3])

## ar3   
## 0.07106301

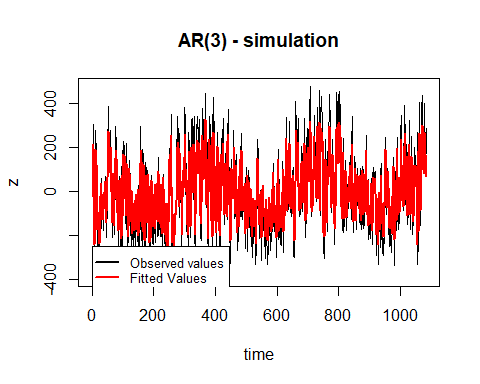
(est.sigma2=AR3$sigma2)

## [1] 17886.17

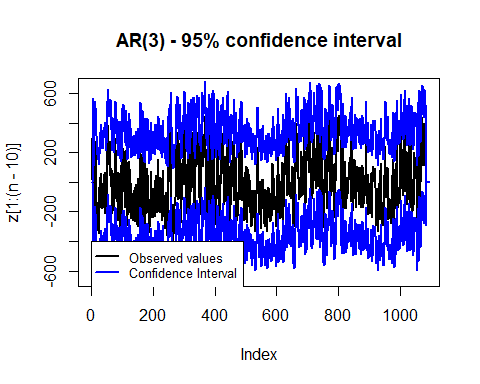
## AR(3) model fitting

With the parameters estimated above, we will obtain our fitted values as a weighted linear combination of parameters and observations:

z.fit=rep(NA,times=n-10)  
for (t in 4:(n-10)) {  
 z.fit[t]=est.phi1\*z[t-1]+est.phi2\*z[t-2]+est.phi3\*z[t-3]  
}  
  
plot(z[1:(n-10)],main="AR(3) - simulation",xlab="time",ylab="z",xlim=c(0,n-10),type="l",lwd=1)  
lines(z.fit,col="red",lwd=2)  
legend(0, -250, legend=c("Observed values", "Fitted Values"),  
 col=c("black", "red"), lty=c(1,1,2,1),lwd=c(2,2,2,4),pch=c(-1,-1,-1), cex=0.8)



left=rep(0,times=n)  
for (t in 4:(n-10)) {  
 left[t]=z.fit[t]-1.960\*sd(z[1:(n-10)])   
}  
  
right=rep(0,times=n)  
  
for (t in 4:(n-10)) {  
 right[t]=z.fit[t]+1.960\*sd(z[1:(n-10)])   
}  
  
plot(z[1:(n-10)],main="AR(3) - 95% confidence interval", type="l",lwd=2, ylim=c(-650,650))  
lines(left,col="blue",lwd=2)  
lines(right,col="blue",lwd=2)  
legend(0, -400, legend=c("Observed values","Confidence Interval"),  
 col=c("black", "blue"), lty=c(1,1,2,1),lwd=c(2,2,2,4),pch=c(-1,-1,-1), cex=0.8)



Looking at the plot of the fitted values against the true observations, we can say that an auto regressive model of order 3 fits quite well our data, given that it follows nearly perfectly our original series, but for some peaks. Looking at the second plot, which shows the confidence intervals of our fitted values, we can see that all the original observations are inside the interval, hence we can say once again that our model fits well the data. To check the stationarity of the model, we need to solve the characteristic polynomial of the model through a function called “polyroot”, which finds the roots of the polynomial, where, for the process to be stationary, the absolute value of the roots of the equation must be greater than 1.

xs=polyroot(c(1,-est.phi1,-est.phi2,-est.phi3))  
  
abs(xs[1])

## [1] 1.42117

abs(xs[2])

## [1] 3.146699

abs(xs[3])

## [1] 3.146699

y=1-est.phi1\*xs[1]-est.phi2\*xs[1]^2-est.phi3\*xs[1]^3

Thus, from the results obtained, we can say that the series is stationary. For future model comparison, we calculate the AIC and the BIC.

AIC(AR3)

## [1] 13701.19

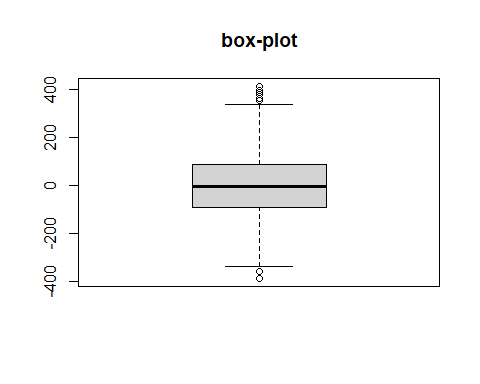
BIC(AR3)

## [1] 13726.13

## AR(3) model diagnostic

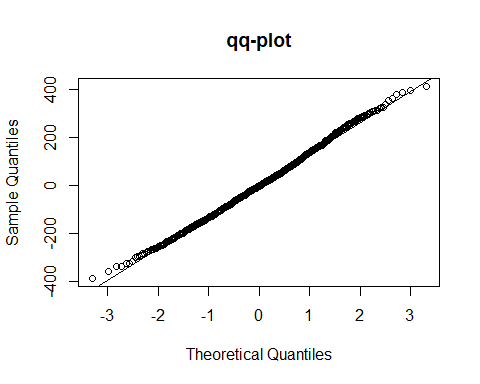
We now pass to model diagnostic. In order for our residuals to be correct, they need to be independent, identically distributed, and possibly normal. Hence, we start first from the boxplot, to check for symmetry and therefore indications of normality:

z.res <- AR3$residuals  
boxplot(z.res,main="box-plot")



We see that our residuals seem to be symmetric. We then go on to the qq-plot to check for normality, comparing our residuals with the normal distribution, to see if they follow it:

qqnorm(z.res,main="qq-plot")  
qqline(z.res)



It seems from the plot that, but for a couple of observations in the tails, our residuals follow a normal distribution. To confirm this, we also check 2 analytical tests of normality, the Shapiro-Wilk and the Jarque-Bera, which area as already said above designed for independent data, which our residulas shoould be, if the model is okay:

shapiro.test(z.res)

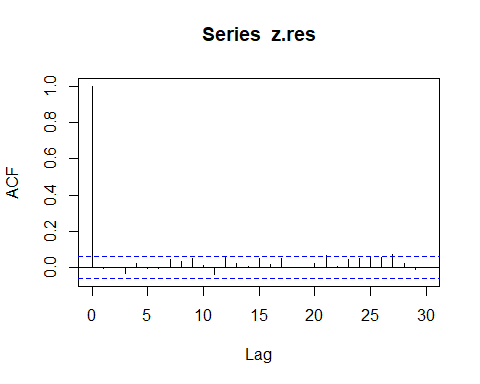
##   
## Shapiro-Wilk normality test  
##   
## data: z.res  
## W = 0.99753, p-value = 0.09908

jarque.bera.test(z.res)

##   
## Jarque Bera Test  
##   
## data: z.res  
## X-squared = 5.1153, df = 2, p-value = 0.07749

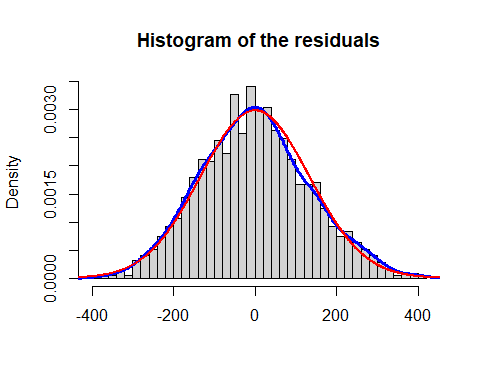
From both the tests, we can see that we fail to reject the null hypothesis of normality, hence our residuals can be said to be normal. We then check the ACF plot to see if our residuals are actually independent:

acf(z.res)



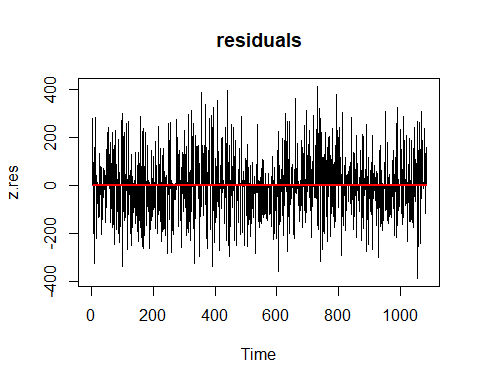
As you can see from above, there seems to be independence, since all the ticks are between the blue bands. We than plot a histogram, to see once again the distribution of our residuals and compare it with a normal distribution, to see if they behave similarly:

hist(z.res,main="Histogram of the residuals",xlab="",freq=F,breaks=50)  
lines(density(z.res),col="blue",lwd=3)  
zz=seq(-500,500,length=n)  
f.zz=dnorm(zz,mean(z.res),sd(z.res))  
lines(zz,f.zz,col="red",lwd=2)



Once again our residuals seem to be symmetric and normal. As last check, we inspect the magnitude of the residuals, and potential patterns:

plot(z.res,main="residuals", type="h",lwd=1)  
lines(rep(0,times=n-10),type="l",col="red",lwd=2)



We don’t see any particular pattern, and we can say that the magnitudex is reasonable, given the range of our observations. For future model comparison, and to evaluate the accuracy of our model, we check the MAE and RMSE, measures that tell us how far the fitted values are from the actual ones:

RMSE=sqrt(mean(z.res^2))  
  
RMSE

## [1] 133.7392

MAE=mean(abs(z.res))  
  
MAE

## [1] 106.4929

To have a more reliable metric, we divide the mean errors by the range of our response, to cancel the effect of the magnitude of our data:

#normalized RMSE  
RMSE/(abs(range(z[1:(n-10)])[1]-range(z[1:(n-10)])[2]))

## [1] 0.1535349

#normalized MAE  
MAE/(abs(range(z[1:(n-10)])[1]-range(z[1:(n-10)])[2]))

## [1] 0.1222557

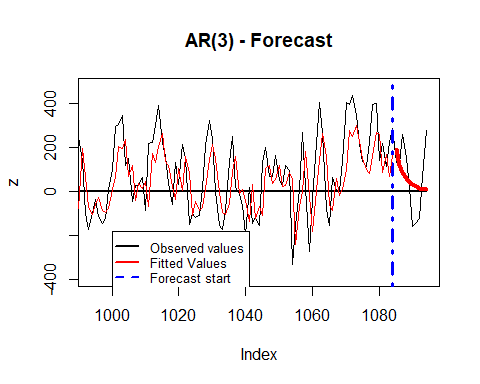
Given that our metrics vary between 0 and 1, we can say that our results are quite good.

## AR(3) model forecasting

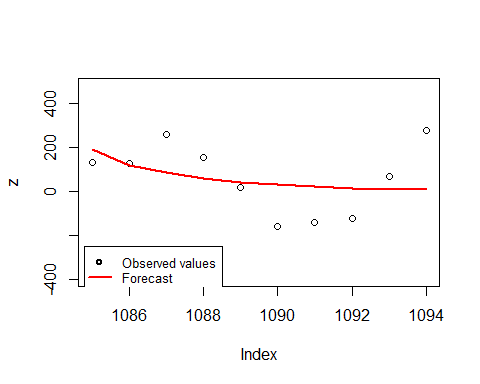
As last thing, we see how our model behaves in forecasting, so in the prediction of future values.

z.for=rep(NA,times=n) # empty vector to be filled with our forecast  
  
z.for[n-9]=est.phi1\*z[n-10]+est.phi2\*z[n-11]+est.phi3\*z[n-12] # the one-step ahead forecast is entirely based only on the observed values  
  
z.for[n-8]=est.phi1\*z.for[n-9]+est.phi2\*z[n-10]+est.phi3\*z[n-11] # now the value at t-1 is a forecasted one, while the other 2 are still observed  
  
z.for[n-7]=est.phi1\*z.for[n-8]+est.phi2\*z.for[n-9]+est.phi3\*z[n-10] # now just the observation as t-3 is observed, the other 2 are forecasted  
  
for (t in (n-6):n) {   
 z.for[t]=est.phi1\*z.for[t-1]+est.phi2\*z.for[t-2]+est.phi3\*z.for[t-3] # now that we are working just with forecasted values, we can set a for cycle that computes forecasts from n-6 until the end  
}

plot(z,main="AR(3) - Forecast",xlim=c(n-100,n),type="l")  
lines(rep(mean(z,na.rm=TRUE),times=n),col="black",lwd=2)  
abline(v=n-10,col="blue",lwd=3,lty=4)  
lines(z.for,type="l",col="red",lwd=4)  
lines(z.fit[1:(n-10)],type="l",col="red",lwd=1)  
legend(1000, -180, legend=c("Observed values", "Fitted Values","Forecast start","Forecasts"),  
 col=c("black", "red","blue","red"), lty=c(1,1,2,1),lwd=c(2,2,2,4),pch=c(-1,-1,-1), cex=0.8)



plot(z,xlim=c(n-9,n),type="p")  
lines(z.for,type="l",col="red",lwd=2)  
legend(1084.8, -250, legend=c("Observed values", "Forecast"),  
 col=c("black", "red"), lty=c(-1,1,1),lwd=c(2,2,2),pch=c(1,-1,-1), cex=0.8)



Now that we have our forecasted values, we can compute then errors we have in our forecasts:

er.for=rep(NA,times=n) # one-step-ahead forecast error  
  
for (t in (n-10):n) {  
 er.for[t]=z[t]-z.for[t]   
}  
  
RMSE.for=sqrt(mean(er.for^2,na.rm=TRUE))  
  
RMSE.for

## [1] 141.1461

MAE.for=mean(abs(er.for),na.rm=TRUE)  
  
MAE.for

## [1] 116.6984

To have a better representation for future comparison, we decided to standardize errors of the forecast

#normalized RMSE  
RMSE.for/(abs(range(z[1:(n-10)])[1]-range(z[1:(n-10)])[2]))

## [1] 0.1620382

#normalized MAE  
MAE.for/(abs(range(z[1:(n-10)])[1]-range(z[1:(n-10)])[2]))

## [1] 0.1339718

At this point we need to compute the psi coefficients to estimate the variance of the forecast errors:

psi=rep(0,times=10)  
psi.0=1  
psi[1]=est.phi1  
psi[2]=est.phi2+est.phi1\*psi[1]  
psi[3]=est.phi3+est.phi2\*psi[1]+est.phi1\*psi[2]  
  
for (j in 4:10) {  
 psi[j]=est.phi3\*psi[j-3]+est.phi2\*psi[j-2]+est.phi1\*psi[j-1]  
}  
psi

## [1] 0.72445232 0.40919810 0.28373688 0.20971877 0.14820073 0.10327714  
## [7] 0.07258570 0.05117422 0.03601916 0.02533489

var.er=rep(0,times=10)  
  
var.er[1]=est.sigma2  
  
for (j in 2:10) {  
 var.er[j]=est.sigma2\*(psi.0+sum(psi[1:(j-1)]^2))  
}  
  
var.er

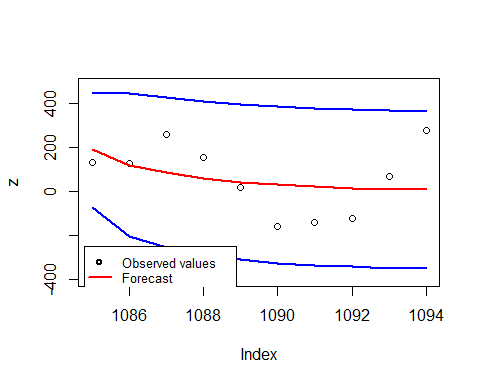
## [1] 17886.17 27273.39 30268.31 31708.27 32494.93 32887.78 33078.55 33172.79  
## [9] 33219.63 33242.84

We calculate confidence intervals of the values forecasted by the model

left.er=rep(NA,times=n)  
  
for (t in (n-9):n) {  
 left.er[t]=z.for[t]-1.960\*sqrt(var.er[t-n+10])   
}  
  
right.er=rep(NA,times=n)  
  
for (t in (n-9):n) {  
 right.er[t]=z.for[t]+1.960\*sqrt(var.er[t-n+10])  
}

And plot the forecast with the corresponding confidence intervals

plot(z,xlim=c(n-9,n),type="p")  
lines(z.for,type="l",col="red",lwd=2)  
lines(left.er,type="l",col="blue",lwd=2)  
lines(right.er,type="l",col="blue",lwd=2)  
legend(1084.8, -250, legend=c("Observed values", "Forecast","Confidence Interval"),  
 col=c("black", "red","blue"), lty=c(-1,1,1),lwd=c(2,2,2),pch=c(1,-1,-1), cex=0.8)



# 3. ARMA model

## ARMA model specification

To decide the parameters p,q of an ARMA model we can’t look at acf or pacf, we look at AIC and BIC of some models of this class and will fit on data the one with the lowest scores

#ARMA(1,1)  
dim=3 #number of parameters  
  
ARMA11=arima(z,order=c(1,0,1),method='ML') #ARMA(1,1) model  
  
est.phi1=ARMA11$coef[1] #values of the estimates of parameters  
  
est.theta1=ARMA11$coef[2]  
  
est.sigma2=ARMA11$sigma2  
  
a.obs=rep(0,times=n)  
  
LogLik=-0.5\*(n-1)\*log(2\*pi\*est.sigma2) #log likelihood  
for (t in 2:n) {  
 a.obs[t]=z[t]-est.phi1\*z[t-1]-est.theta1\*a.obs[t-1]  
 LogLik=LogLik-0.5\*(a.obs[t]^2)/est.sigma2  
}  
  
Deviance11=-2\*LogLik  
AIC11=Deviance11+2\*dim #value of AIC  
BIC11=Deviance11+dim\*log(n) #value of BIC

#ARMA(2,1)  
  
dim=4 #number of parameters  
  
ARMA21=arima(z,order=c(2,0,1),method='ML') #ARMA(2,1) model  
  
ARMA21est.phi1=ARMA21$coef[1] #estimates of parameters  
  
ARMA21est.phi2=ARMA21$coef[2]  
  
ARMA21est.theta1=ARMA21$coef[3]  
  
ARMA21est.sigma2=ARMA21$sigma2  
  
a.obs=rep(0,times=n)  
  
LogLik=-0.5\*(n-2)\*log(2\*pi\*est.sigma2) #log likelihood  
for (t in 3:n) {  
 a.obs[t]=z[t]-ARMA21est.phi1\*z[t-1]-ARMA21est.phi2\*z[t-2]-ARMA21est.theta1\*a.obs[t-1]  
 LogLik=LogLik-0.5\*(a.obs[t]^2)/est.sigma2  
}  
  
Deviance21=-2\*LogLik  
  
AIC21=Deviance21+2\*dim #value of AIC  
  
BIC21=Deviance21+dim\*log(n) #value of BIC

# ARMA(1,2)   
  
dim=4 #number of parameters  
  
ARMA12=arima(z,order=c(1,0,2),method='ML') #ARMA(1,2) model  
  
ARMA12est.phi1=ARMA12$coef[1] #estimates of parameters  
  
ARMA12est.theta1=ARMA12$coef[2]  
  
ARMA12est.theta2=ARMA12$coef[3]  
  
ARMA12est.sigma2=ARMA12$sigma2  
  
a.obs=rep(0,times=n)  
  
LogLik=-0.5\*(n-2)\*log(2\*pi\*est.sigma2) #log likelihood  
for (t in 3:n) {  
 a.obs[t]=z[t]-ARMA12est.phi1\*z[t-1]-ARMA12est.theta1\*a.obs[t-1]-ARMA12est.theta2\*a.obs[t-2]  
 LogLik=LogLik-0.5\*(a.obs[t]^2)/est.sigma2  
}  
  
Deviance12=-2\*LogLik  
  
AIC12=Deviance12+2\*dim #value of AIC  
  
BIC12=Deviance12+dim\*log(n) #value of BIC

#ARMA(2,2)  
  
dim=5 #number of parameters  
  
ARMA22=arima(z,order=c(2,0,2),method='ML') #ARMA(2,2) model  
  
ARMA22est.phi1=ARMA22$coef[1] #estimates of parameters  
  
ARMA22est.phi2=ARMA22$coef[2]  
  
ARMA22est.theta1=ARMA22$coef[3]  
  
ARMA22est.theta2=ARMA22$coef[4]  
  
ARMA22est.sigma2=ARMA22$sigma2  
  
a.obs=rep(0,times=n)  
  
LogLik=-0.5\*(n-2)\*log(2\*pi\*est.sigma2) #log likelihood  
for (t in 3:n) {  
 a.obs[t]=z[t]-ARMA22est.phi1\*z[t-1]-ARMA22est.phi2\*z[t-2]  
 -ARMA22est.theta1\*a.obs[t-1]-ARMA22est.theta2\*a.obs[t-2]  
 LogLik=LogLik-0.5\*(a.obs[t]^2)/ARMA22est.sigma2  
}  
  
Deviance22=-2\*LogLik  
  
AIC22=Deviance22+2\*dim #AIC score  
  
BIC22=Deviance22+dim\*log(n) #BIC score

Now that we have tried some models, we can compare AIC and BIC values for selecting the best one

AICS=c(AIC11, AIC12, AIC21, AIC22 )  
BICS=c(BIC11, BIC12, BIC21, BIC22)  
AICS

## [1] 13813.58 13800.39 13802.97 14487.25

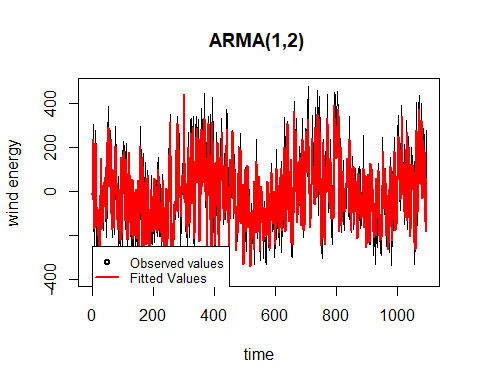
BICS

## [1] 13828.57 13820.38 13822.96 14512.24

The model with the lowest AIC and BIC scores is the one with parameters p=1, q=2, so we fit this model on data for further analysis and plot the fitted values with the corresponding confidence intervals

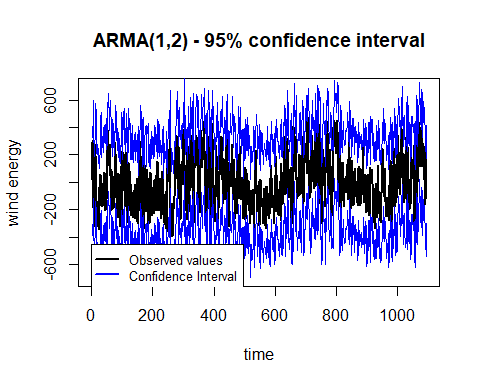
## ARMA(1,2) model fitting

#ARMA(1,2) fitting  
a=rnorm(n,mean=0,sd=sqrt(ARMA12est.sigma2)) #white noise vector  
z.fit=rep(NA,times=n) #vector of fitted values of the model  
z.fit[1]=a[1] #first fit   
z.fit[2]=ARMA12est.phi1\*z.fit[1]+ARMA12est.theta1\*a[1] #second fit  
  
for (t in 3:n) {  
 z.fit[t]=ARMA12est.phi1\*z[t-1]+a[t-1]\*ARMA21est.theta1+ARMA12est.theta2\*a[t-2]  
}  
  
plot(z,main="ARMA(1,2) ",xlab="time",ylab="wind energy",xlim=c(0,n),type="l",lwd=1)  
#fitted:  
lines(z.fit,col="red",lwd=2)  
legend(0, -250, legend=c("Observed values", "Fitted Values"),  
 col=c("black", "red"), lty=c(-1,1,1),lwd=c(2,2,2),pch=c(1,-1,-1), cex=0.8)



As we can see, the model fits data very well.Then, let’s look at the 95% confidence interval

#right conf int  
right=rep(0,times=n)  
right[1]=NA  
right[2]=NA  
  
for (t in 3:n) {  
 right[t]=z.fit[t]+1.960\*sd(z,na.rm=TRUE)   
}  
#left conf int  
left=rep(0,times=n)  
left[1]=NA  
left[2]=NA  
  
for (t in 3:n) {  
 left[t]=z.fit[t]-1.960\*sd(z,na.rm=TRUE)   
}   
#plot  
plot(z,main="ARMA(1,2) - 95% confidence interval",xlab="time",ylab="wind energy",type="l",lwd=2,ylim=c(-700,700))  
lines(left,col="blue",lwd=1)  
lines(right,col="blue",lwd=1)  
legend(0, -450, legend=c("Observed values", "Confidence Interval"),  
 col=c("black", "blue"), lty=c(1,1,1),lwd=c(2,2,2),pch=c(-1,-1,-1), cex=0.8)



Fitted values of the model follow well original observations, even if they fail to reach some peaks. Looking at confidence intervals, no observation is above or below the thresholds so we are happy with the results. To go on with the analysis, we check if the series is stationary. Since we are dealing with an ARMA model and the MA component is always (weakly) stationary, we need to investigate phi estimate to determine stationarity. This need to be, in modulo, <1.

ARMA12est.phi1

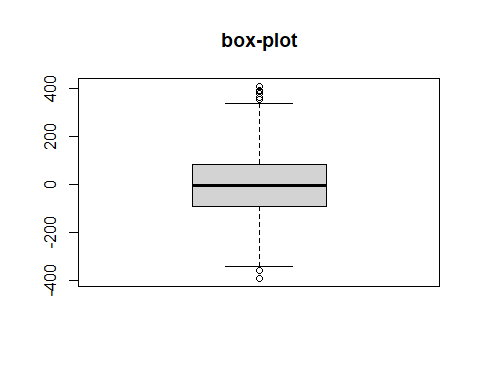
## ar1   
## 0.7165658

From the result we can say that we are working with a stationary model.

## ARMA(1,2) Model Diagnostic

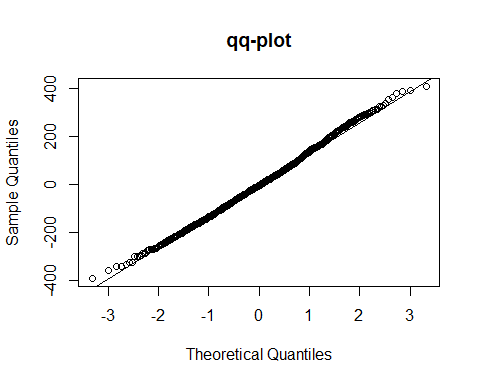
Now we can perform model diagnostic, with the same steps as before with AR model to look if errors respect i.i.d. assumptions. First, let’s look at the box-plot

ARMA12z.res <- ARMA12$residuals  
boxplot(ARMA12z.res,main="box-plot")



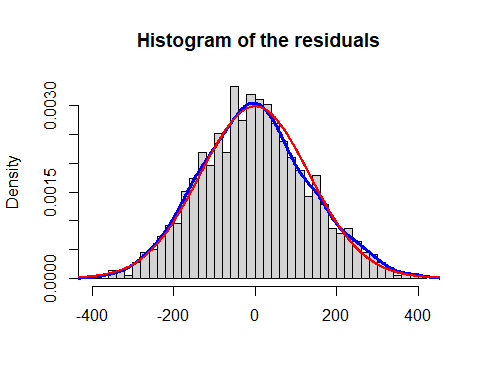
Residuals seem to be symmetric, and we look at qq-plot to see if they follow a Normal distribution

qqnorm(ARMA12z.res,main="qq-plot")  
qqline(ARMA12z.res)



From qq-plot residuals seem to follow a normal distribution. As last graphical check for normality, we plot histogram of model’s residuals

hist(ARMA12z.res,main="Histogram of the residuals",xlab="",freq=F,breaks=50)  
lines(density(ARMA12z.res),col="blue",lwd=3)  
zz=seq(-500,500,length=n)  
f.zz=dnorm(zz,mean(ARMA12z.res),sd(ARMA12z.res))  
lines(zz,f.zz,col="red",lwd=2)



Also from the histogram residuals follow the theoretical normal distribution. We male use of shapiro and jarque bera tests to confirm our intuition.

shapiro.test(ARMA12z.res)

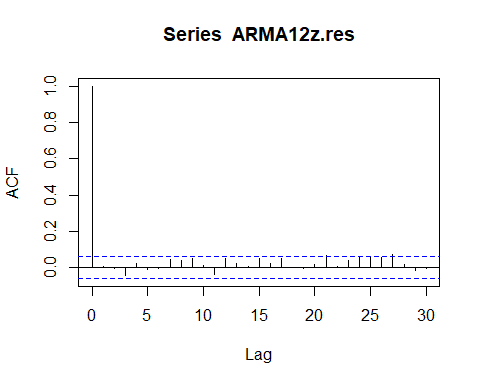
##   
## Shapiro-Wilk normality test  
##   
## data: ARMA12z.res  
## W = 0.99754, p-value = 0.09722

jarque.bera.test(ARMA12z.res)

##   
## Jarque Bera Test  
##   
## data: ARMA12z.res  
## X-squared = 5.0258, df = 2, p-value = 0.08103

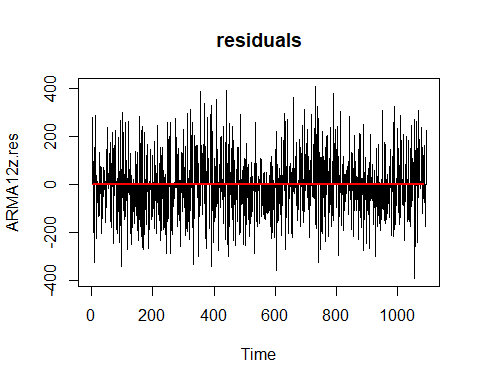
For both tests we have to accept the null hypotesis of normality at level alpha=5%, so residuals respect the assumption to be normally distributed. Now we check independence by looking at acf

acf(ARMA12z.res)



Even if there are three lags at which acf slightly breaks the threshold, in general residuals seem to be independent. We now look for potential patterns by plotting residuals.

plot(ARMA12z.res,main="residuals", type="h",lwd=1)  
lines(rep(0,times=n-10),type="l",col="red",lwd=2)



From the plot, it doesn’t look like they follow a specific pattern. We now calculate MAE and RMSE for future model comparison.

ARMA12\_RMSE=sqrt(mean(ARMA12z.res^2))  
  
ARMA12\_RMSE

## [1] 133.6679

ARMA12\_MAE=mean(abs(ARMA12z.res))  
  
ARMA12\_MAE

## [1] 106.4411

And normalize these two measures dividing the mean errors by the range of the response.

#normalized RMSE  
ARMA12\_RMSE/(abs(range(z[1:(n-10)])[1]-range(z[1:(n-10)])[2]))

## [1] 0.1534531

#normalized MAE  
ARMA12\_MAE/(abs(range(z[1:(n-10)])[1]-range(z[1:(n-10)])[2]))

## [1] 0.1221962

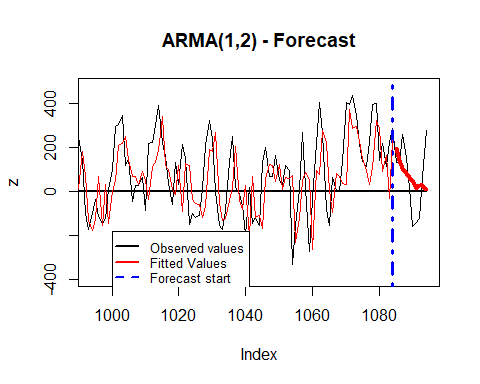
## ARMA(1,2) forecasting

Now that we have completed model diagnostic, and assumptions are respected, we can use the model for forecasting future values.

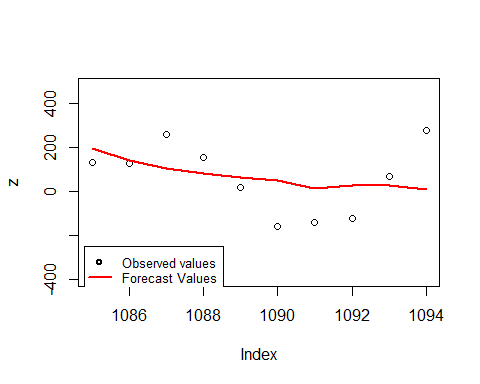
z.for=rep(NA,times=n)   
  
z.for[n-9]=ARMA12est.phi1\*z[n-10]+ARMA12est.theta1\*a[n-10]+ARMA12est.theta2\*a[n-11]  
  
z.for[n-8]=ARMA12est.phi1\*z.for[n-9]+ARMA12est.theta1\*a[n-9]+ARMA12est.theta2\*a[n-10]   
  
z.for[n-7]=ARMA12est.phi1\*z.for[n-8]+ARMA12est.theta1\*a[n-8]+ARMA12est.theta2\*a[n-9]   
  
for (t in (n-6):n) {   
 z.for[t]=ARMA12est.phi1\*z.for[t-1]+ARMA12est.theta1\*a[t-2]+ARMA12est.theta2\*a[t-3]   
}

We plot the forecast:

plot(z,main="ARMA(1,2) - Forecast",xlim=c(n-100,n),type="l")  
lines(rep(mean(z,na.rm=TRUE),times=n),col="black",lwd=2)  
abline(v=n-10,col="blue",lwd=3,lty=4)  
lines(z.for,type="l",col="red",lwd=4)  
lines(z.fit[1:(n-10)],type="l",col="red",lwd=1)  
legend(1000, -180, legend=c("Observed values", "Fitted Values","Forecast start","Forecasts"),  
 col=c("black", "red","blue","red"), lty=c(1,1,2,1),lwd=c(2,2,2,4),pch=c(-1,-1,-1), cex=0.8)



plot(z,xlim=c(n-9,n),type="p")  
lines(z.for,type="l",col="red",lwd=2)  
legend(1084.8, -250, legend=c("Observed values", "Forecast Values"),  
 col=c("black", "red"), lty=c(-1,1,1),lwd=c(2,2,2),pch=c(1,-1,-1), cex=0.8)



We now calculate errors of the forecast for future model comparison

er.for=rep(NA,times=n) # one-step-ahead forecast error  
  
for (t in (n-10):n) {  
 er.for[t]=z[t]-z.for[t]   
}  
  
ARMA12\_RMSE.for=sqrt(mean(er.for^2,na.rm=TRUE))  
  
ARMA12\_RMSE.for

## [1] 140.0237

ARMA12\_MAE.for=mean(abs(er.for),na.rm=TRUE)  
  
ARMA12\_MAE.for

## [1] 116.1629

#normalized forecast RMSE  
ARMA12\_RMSE.for/(abs(range(z[1:(n-10)])[1]-range(z[1:(n-10)])[2]))

## [1] 0.1607497

#normalized forecast MAE  
ARMA12\_MAE.for/(abs(range(z[1:(n-10)])[1]-range(z[1:(n-10)])[2]))

## [1] 0.133357

At this point, we need to calculate the estimate of the variance of the errors to plot forecasted values with their corresponding 95% confidence interval

#estimate of the variance  
psi=rep(0,times=10)  
psi.0=1  
psi[1]=ARMA12est.theta1+ARMA12est.phi1  
psi[2]=ARMA12est.theta2+ARMA12est.phi1\*psi[1]  
  
for (j in 3:10) {  
 psi[j]=ARMA12est.phi1\*psi[j-1] #since q=2, we don't consider estimates of theta at times t >2   
}

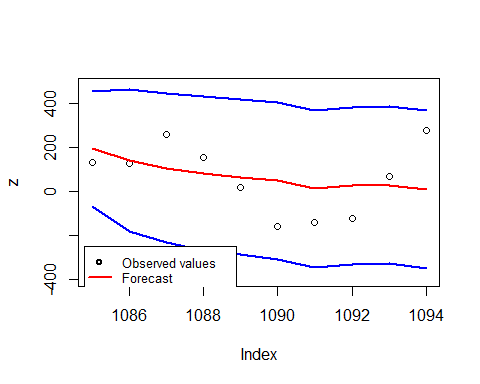
var.er=rep(0,times=10)  
  
var.er[1]=ARMA12est.sigma2  
  
for (j in 2:10) {  
 var.er[j]=ARMA12est.sigma2\*(psi.0+sum(psi[1:(j-1)]^2))  
}

We now build 95% confidence intervals

left.er=rep(NA,times=n)  
  
for (t in (n-9):n) {  
 left.er[t]=z.for[t]-1.960\*sqrt(var.er[t-n+10])   
}  
  
right.er=rep(NA,times=n)  
  
for (t in (n-9):n) {  
 right.er[t]=z.for[t]+1.960\*sqrt(var.er[t-n+10])  
}

And plot forecasts with corresponding CI

plot(z,xlim=c(n-9,n),type="p")  
lines(z.for,type="l",col="red",lwd=2)  
lines(left.er,type="l",col="blue",lwd=2)  
lines(right.er,type="l",col="blue",lwd=2)  
legend(1084.8, -250, legend=c("Observed values", "Forecast","Confidence Interval"),  
 col=c("black", "red","blue"), lty=c(-1,1,1),lwd=c(2,2,2),pch=c(1,-1,-1), cex=0.8)



# 4. ARX model

## Modelling with Exogenous variables

Due to the presence of wind capacity and temperature, we decided to model the Wind generation using an ARX model, to take into account the potential effect of exogenous variables.

df<-w

As first step, we scale the explanatory variables because their magnitude and their unit of measurement can affect how they impact on the response. To avoid this problem, we use directly the function “scale” that by default centers and scales vectors.

y=as.vector(df$wind\_generation\_actual)  
w1=as.vector(df$wind\_capacity)  
w2=as.vector(df$temperature)  
# we standardize the variables using the function "scale"  
x1=scale(w1)  
x2=scale(w2)  
n=length(y)

Now that our covariates have and , we can start modelling.

## ARX Model Specification

To decide the order of the model to use, we cannot rely on PACF due to the structure that contains the , so we’ll fit them and compare their AIC and, more important, the significance of the parameters.

Before this step, we define nt as the length of our data minus 10, because the last 10 values we’ll be used to evaluate the forecast performance of the model.

nt = n-10  
zy=y[2:n]  
zx1=y[1:(n-1)]  
zx2=x1[1:(n-1)]  
zx3=x2[1:(n-1)]  
set.seed(2023)  
ARX1=lm(zy[1:nt]~zx1[1:nt]+zx2[1:nt]+zx3[1:nt])  
summary(ARX1)

##   
## Call:  
## lm(formula = zy[1:nt] ~ zx1[1:nt] + zx2[1:nt] + zx3[1:nt])  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -463839 -94116 -17315 76986 528062   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.119e+05 8.513e+03 13.142 < 2e-16 \*\*\*  
## zx1[1:nt] 6.351e-01 2.357e-02 26.951 < 2e-16 \*\*\*  
## zx2[1:nt] 1.281e+04 4.634e+03 2.765 0.0058 \*\*   
## zx3[1:nt] -2.534e+04 4.853e+03 -5.221 2.14e-07 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 148500 on 1080 degrees of freedom  
## Multiple R-squared: 0.4813, Adjusted R-squared: 0.4799   
## F-statistic: 334.1 on 3 and 1080 DF, p-value: < 2.2e-16

zy=y[3:n]  
zx1=y[2:(n-1)]  
zx2=y[1:(n-2)]  
zx3=x1[2:(n-1)]  
zx4=x2[2:(n-1)]  
set.seed(2023)  
ARX2=lm(zy[1:nt]~zx1[1:nt]+zx2[1:nt]+zx3[1:nt]+zx4[1:nt])  
summary(ARX2)

##   
## Call:  
## lm(formula = zy[1:nt] ~ zx1[1:nt] + zx2[1:nt] + zx3[1:nt] + zx4[1:nt])  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -431087 -96322 -19579 74661 544572   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.237e+05 9.183e+03 13.470 < 2e-16 \*\*\*  
## zx1[1:nt] 6.985e-01 3.020e-02 23.133 < 2e-16 \*\*\*  
## zx2[1:nt] -1.019e-01 3.055e-02 -3.334 0.000885 \*\*\*  
## zx3[1:nt] 1.418e+04 4.633e+03 3.060 0.002267 \*\*   
## zx4[1:nt] -2.872e+04 4.937e+03 -5.816 7.92e-09 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 147800 on 1079 degrees of freedom  
## Multiple R-squared: 0.4868, Adjusted R-squared: 0.4849   
## F-statistic: 255.9 on 4 and 1079 DF, p-value: < 2.2e-16

zy=y[4:n]  
zx1=y[3:(n-1)]  
zx2=y[2:(n-2)]  
zx3=y[1:(n-3)]  
zx4=x1[3:(n-1)]  
zx5=x2[3:(n-1)]  
set.seed(2023)  
ARX3=lm(zy[1:nt]~zx1[1:nt]+zx2[1:nt]+zx3[1:nt]+zx4[1:nt]+zx5[1:nt])  
summary(ARX3)

##   
## Call:  
## lm(formula = zy[1:nt] ~ zx1[1:nt] + zx2[1:nt] + zx3[1:nt] + zx4[1:nt] +   
## zx5[1:nt])  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -421609 -95887 -18539 75849 543883   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.190e+05 9.973e+03 11.927 < 2e-16 \*\*\*  
## zx1[1:nt] 7.020e-01 3.028e-02 23.182 < 2e-16 \*\*\*  
## zx2[1:nt] -1.244e-01 3.689e-02 -3.372 0.000774 \*\*\*  
## zx3[1:nt] 3.408e-02 3.066e-02 1.112 0.266493   
## zx4[1:nt] 1.444e+04 4.649e+03 3.107 0.001942 \*\*   
## zx5[1:nt] -2.719e+04 5.051e+03 -5.383 8.98e-08 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 147600 on 1078 degrees of freedom  
## Multiple R-squared: 0.4888, Adjusted R-squared: 0.4864   
## F-statistic: 206.1 on 5 and 1078 DF, p-value: < 2.2e-16

library(AICcmodavg)  
models=list(ARX1,ARX2,ARX3)  
  
mod.names=c('arx1','arx2','arx3')  
  
aictab(cand.set=models,modnames=mod.names)

##   
## Model selection based on AICc:  
##   
## K AICc Delta\_AICc AICcWt Cum.Wt LL  
## arx3 7 28887.92 0.00 0.82 0.82 -14436.91  
## arx2 6 28890.98 3.06 0.18 1.00 -14439.45  
## arx1 5 28900.08 12.16 0.00 1.00 -14445.01

For a very small difference, the AIC is minimized with the ARX of order 3, but we decided to favors simpler model, ARX(2), since the parameter of is not significant as we’ve seen from the summary.

## ARX(2) Model Fitting

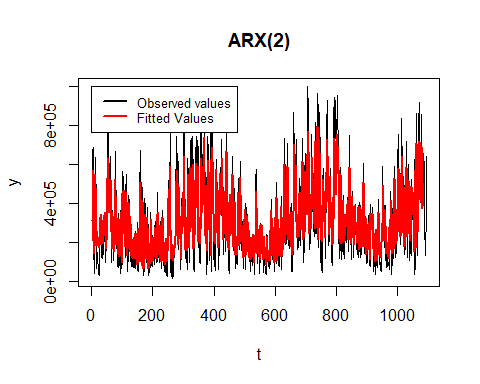
est.eta=ARX2$coefficients[1] #intercept  
  
# y coefficients:  
est.phi1=ARX2$coefficients[2]   
est.phi2=ARX2$coefficients[3]  
  
# X coefficients:  
est.lambda1=ARX2$coefficients[4]  
est.lambda2=ARX2$coefficients[5]  
# var  
est.sigma2=var(ARX2$residuals)

Using this estimates of the coefficients, we’ll create two vectors for fitted values and residuals and move to the model diagnostic.

## ARX(2) Model Diagnostic

y.fit=rep(NA,times=n)  
y.res=rep(NA,times=n)  
  
for (t in 3:nt) {  
 y.fit[t]=est.eta+est.phi1\*y[t-1]+est.phi2\*y[t-2]+est.lambda1\*x1[t]+est.lambda2\*x2[t]  
 y.res[t]=y[t]-y.fit[t]  
}

plot(y,main="ARX(2)",xlab="t",type="l")  
lines(y.fit,type="l",col="red",lwd=2)  
legend(0, 1000000, legend=c("Observed values", "Fitted Values"),  
 col=c("black", "red"), lty=c(1,1),lwd=c(2,2), cex=0.8)



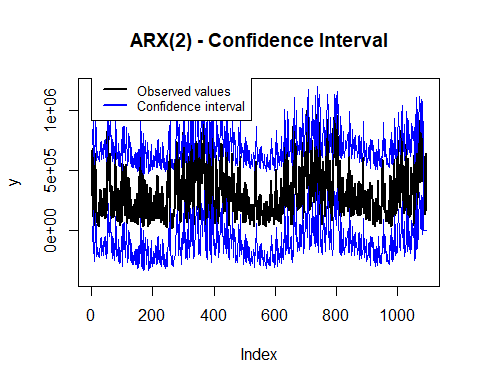
As first check, we compare the series of the fitted values and the original data, here we can observe that they fit very well the series.

After this point estimate, we check also the confidence intervals.

left=rep(0,times=n)  
left[1]=NA  
left[2]=NA  
  
for (t in 3:nt) {  
 left[t]=y.fit[t]-1.960\*sd(y,na.rm=TRUE)   
}

right=rep(0,times=n)  
right[1]=NA  
right[2]=NA  
  
for (t in 3:nt) {  
 right[t]=y.fit[t]+1.960\*sd(y,na.rm=TRUE) #P(|Z|<1.960)=0.95 Z=Standard Normal  
}

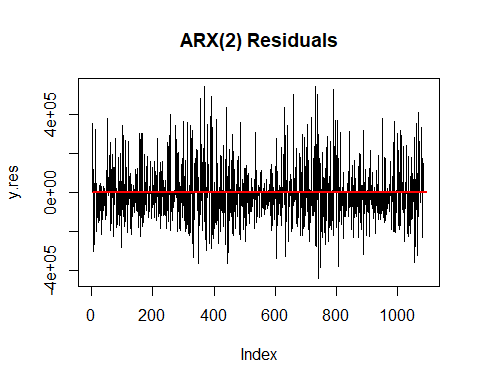
plot(y,main="ARX(2) - Confidence Interval",type="l",lwd=2, ylim=c(-400000,1200000))  
lines(left,col="blue",lwd=1)  
lines(right,col="blue",lwd=1)  
legend(0, 1300000, legend=c("Observed values", "Confidence interval"),  
 col=c("black", "blue"), lty=c(1,1),lwd=c(2,2), cex=0.8)



Let’s check the residuals.

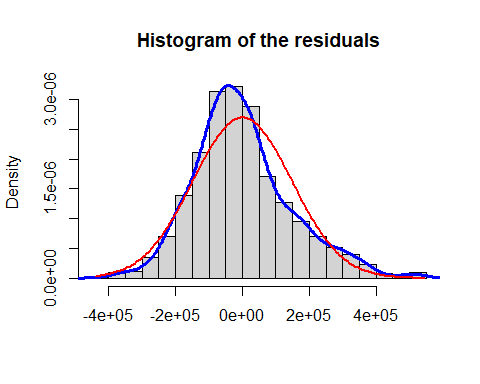
As first thing, we want to see how they’re distributed around 0:

plot(y.res,main="ARX(2) Residuals",type="h",lwd=1)  
lines(rep(0,times=n),type="l",col="red",lwd=2)



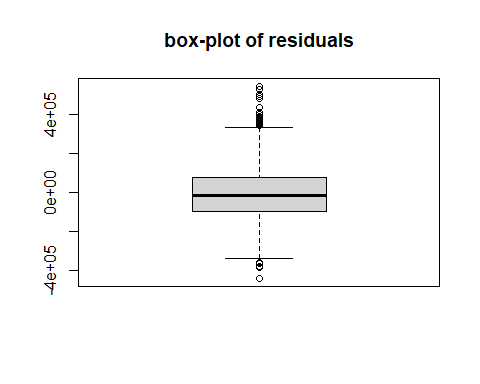
We don’t see any specific pattern, the magnitude is due to the high range of the original data.

hist(y.res[3:nt],main="Histogram of the residuals",xlab="",freq=F,breaks=20)  
lines(density(y.res[3:nt]),col="blue",lwd=3)  
zz=seq(range(y.res,na.rm=TRUE)[1],range(y.res,na.rm=TRUE)[2],length=200)  
f.zz=dnorm(zz,mean(y.res,na.rm=TRUE),sd(y.res,na.rm=TRUE))  
lines(zz,f.zz,col="red",lwd=2)



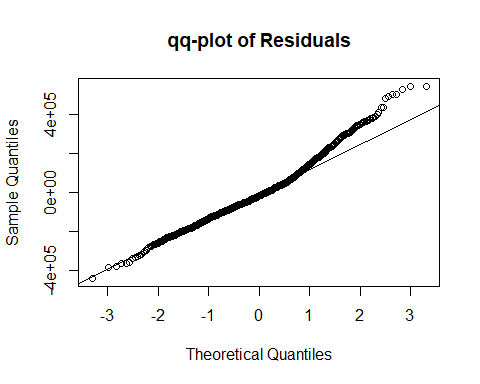
Looking at the histogram, we see that they’re not very far from a potential normal distribution.

boxplot(y.res,main="box-plot of residuals")

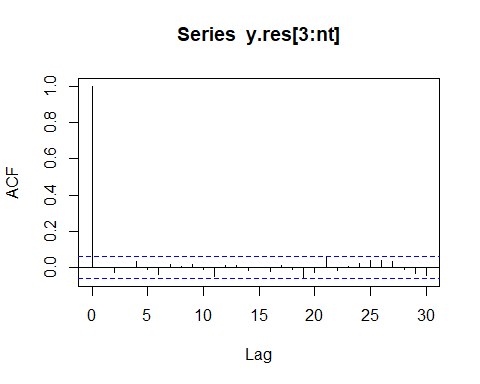


In the boxplot we observe some extreme values but there is no particular skewness. The same holds for qq-plot:

qqnorm(y.res,main="qq-plot of Residuals")  
qqline(y.res)



acf(y.res[3:nt],lag.max=30)



The last plot is used to detect dependency between residuals, but we can state there is nothing strange since all the values are very small, behind the threshold.

shapiro.test(y.res)

##   
## Shapiro-Wilk normality test  
##   
## data: y.res  
## W = 0.97625, p-value = 2.562e-12

library (tseries)  
jarque.bera.test(y.res[3:(n-10)])

##   
## Jarque Bera Test  
##   
## data: y.res[3:(n - 10)]  
## X-squared = 86.782, df = 2, p-value < 2.2e-16

Even if nothing strange raised from the plots, both the Shapiro-Wilk and Jarque-Bera tests show a very small p-value, telling that our data are far from a normal distribution.

RMSE.fit=sqrt(mean(y.res[3:nt]^2))  
  
RMSE.fit

## [1] 148116.8

MAE.fit=mean(abs(y.res[3:nt]))  
  
MAE.fit

## [1] 113536.3

These values for RMSE and MAE cannot be useful because they’re not related to the data, so we divide them by the range of the series:

#RMSE:  
RMSE.fit/(range(y)[2]-range(y)[1])

## [1] 0.1507677

#MAE  
MAE.fit/(range(y)[2]-range(y)[1])

## [1] 0.1155683

Now we can state that the fitted values are quite good.

## ARX(2) Model Forecasting

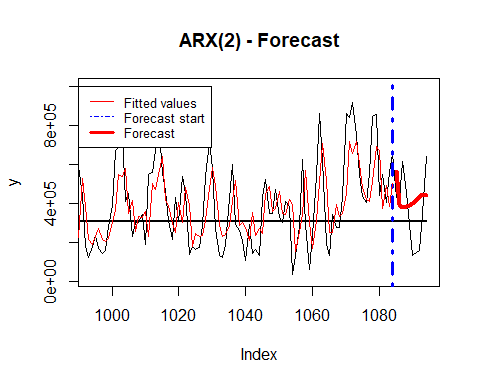
As last step, we evaluate the performance of our model in forecasting future values. This part, in models with exogenous variables, must take into account the covariates in different time - and we have done it by modelling all data except for the last 10 observations from the beginning.

We fix the first two forecasts by taking into account observed values, then we’ll use a “for loop” for the last 8 values that are related on forecasts only:

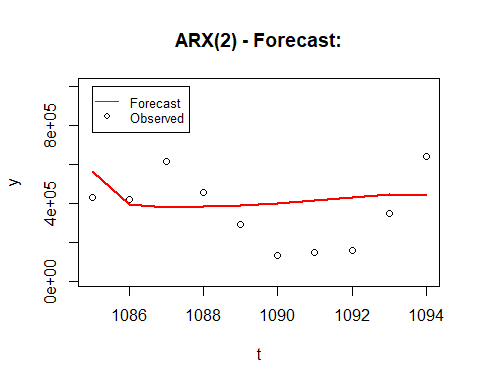
y.for=rep(NA,times=n) # we define a new vectors for storing forecasts, where the first values are the fitted  
# only the last 10 will be new forecasts  
  
y.for[nt+1] = est.eta+est.phi1\*y[nt]+est.phi2\*y[nt-1]+est.lambda1\*x1[nt+1]+est.lambda2\*x2[nt+1]  
y.for[nt+2] = est.eta+est.phi1\*y[nt+1]+est.phi2\*y[nt]+est.lambda1\*x1[nt+2]+est.lambda2\*x2[nt+2]  
  
  
for (t in (nt+3):n) {  
 y.for[t]=est.eta+est.phi1\*y.for[t-1]+est.phi2\*y.for[t-2]+est.lambda1\*x1[t]+est.lambda2\*x2[t]  
}

Let’s explore the forecast graphically viewing them also against observed and fitted values:

plot(y,main="ARX(2) - Forecast",xlim=c(n-100,n),type="l")  
lines(rep(mean(y,na.rm=TRUE),times=n),col="black",lwd=2)  
abline(v=nt,col="blue",lwd=3,lty=4)  
lines(y.for,type="l",col="red",lwd=4)  
lines(y.fit[1:nt],type="l",col="red",lwd=1)  
legend(990, 999900, legend=c("Fitted values", "Forecast start","Forecast"),  
 col=c("red", "blue","red"), lty=c(1,4,1),lwd=c(1,1,3), cex=0.8)



plot(y,main="ARX(2) - Forecast:",xlab="t",type="p",xlim=c(nt+1,n))  
lines(y.for,type="l",col="red",lwd=2)  
legend(1085, 999900, legend=c("Forecast","Observed"),  
 col=c("red", "black"), lty=c(1,-1),lwd=1,pch=c(-1,1), cex=0.8)



As we can see from the plots above, the value converges to the mean.

We can estimate the mean error as in the case of model fitting, and comparing it to the range of the series:

er.for=rep(NA,times=n-nt) # one-step-ahead forecast error  
  
for (t in (nt+3):n) {  
 er.for[t]=y[t]-y.for[t]  
}  
  
RMSE.for=sqrt(mean(er.for^2,na.rm=TRUE))  
  
RMSE.for

## [1] 204949.9

MAE.for=mean(abs(er.for),na.rm=TRUE)  
  
MAE.for

## [1] 188005.9

# But if we compare RMSE and MAE over the range of y:  
RMSE.for/(range(y)[2]-range(y)[1])

## [1] 0.2086181

MAE.for/(range(y)[2]-range(y)[1])

## [1] 0.1913707

The values don’t seem very high - namely the model performs quite good also in forecast task.

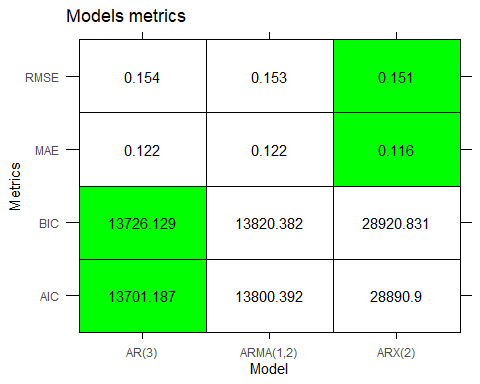
# 5. Conclusions

## Model performance summary

To have a final comparison, we will collect all the metrics we measured above for the models, to find out which suits better our data. We will look at:

* AIC
* BIC
* MAE
* MSE

data <- data.frame(model = c("AR(3)", "ARMA(1,2)", "ARX(2)"),  
 AIC = c(AIC(AR3), AIC12, AIC(ARX2)),  
 BIC = c(BIC(AR3), BIC12, BIC(ARX2)),  
 MAE = c(MAE/(abs(range(z[1:(n-10)])[1]-range(z[1:(n-10)])[2])), ARMA12\_MAE/(abs(range(z[1:(n-10)])[1]-range(z[1:(n-10)])[2])), MAE.fit/(range(y)[2]-range(y)[1])),  
 RMSE = c(RMSE/(abs(range(z[1:(n-10)])[1]-range(z[1:(n-10)])[2])), ARMA12\_RMSE/(abs(range(z[1:(n-10)])[1]-range(z[1:(n-10)])[2])), RMSE.fit/(range(y)[2]-range(y)[1])))  
  
data\_long <- tidyr::gather(data, key = "Metrics", value = "Value", AIC, BIC, MAE, RMSE)  
  
data\_long %>%  
 group\_by(Metrics) %>%  
 mutate(min\_value = min(Value)) %>%  
 ggplot(aes(x = model, y = Metrics, fill = ifelse(Value == min\_value, "green", "white"))) +  
 geom\_tile(color = "black") +  
 geom\_text(aes(label = round(Value, 3)), size = 4, color = "black") +  
 scale\_fill\_identity() +  
 labs(title = "Models metrics",  
 x = "Model",  
 y = "Metrics",  
 fill = "Value") +  
 theme\_minimal() +  
 theme(panel.grid.major = element\_line(color = "black")) +  
 guides(fill = guide\_legend(override.aes = list(color = NA)))



Before starting analyzing the table, it is worth to mention that the ARX model takes also in consideration the exogenous variables, while the AR and ARMA don’t, and that the ARX model was built on the original data, while the AR and ARMA were built after a transformation, taking the square root and adjusting for the mean. Having said this, even if as you can see from the table the ARX model is the best in MAE and RMSE, we will compare AR and ARMA, which work on the same data. Talking about measures of residuals, we can see that the results are pretty much the same in both models, but if we look at the AIC and BIC, we can see that the situation is different, since it’s clear that the AR model is better, since the value is lower for both metrics. This is not a surprise, since with the ACF plot we saw that there was no evidence of a need to use a MA process. We can then say that the best model for our data that we found is the AR(3). It’s worth to mention that the values of the normalized RMSE and MAE in the ARX case seem very good, since they are even lower than the other 2 models, and given that the potential values range from 0 to 1, we can say that we have a pretty good result.