Time Series Analysis - A Tutorial

Rosskopf,E.; Cordes, M.; Lumiko, J.

November 28, 2014

Abstract

Tutorial for time series analysis in R...

Contents

1 Introduction

This tutorial assumes that the reader has some basic knowledge of time series analysis, and the principal focus of the tutorial is not to explain time series analysis, but rather to explain how to carry out these analyses using R.
If you are new to time series analysis, and want to learn more about any of the concepts presented here, We would highly recommend the Open University book "Time series" (product code M249/02), available from from the Open University Shop.

2 Getting started

2.1 Packages

Before we get started, please make sure to set a working directory and download the necessary packages listed below.
Useful packages for time series analysis:

```
> #install.packages(tseries)
> #install.packages(nlme)
> #install.packages(car)
> #install.packages(knitr)
> #install.packages(xtable)
> #install.packages(SweaveListingUtils)
> #install.packages(stats)
> #install.packages(forecast)
> #install.packages(AICcmodavg)
> #install.packages(TTR)
> #install.packages(mgcv)
>
> library(tseries)
> library(nlme)
> library(car)
> library(knitr)
> library(xtable)
> library(SweaveListingUtils)
> library(stats)
> library(forecast)
> library(AICcmodavg)
> library(TTR)
> library(mgcv)
> setwd("//csrv05/public$/Elenamarlene/BestpracticeR/backup/Elena/")
```

2.2 Applied functions

It saves time and makes it easier to follow the tutorial, if the largest functions are placed first. If they apply lateron, they can simply be written in one line without losing focus.
The first function assembles necessary tests, we need iteratively to run after we changes a model structure. The performed function is a diagnostic check wie need to perform in order to revise if our model is adequate enough to stop the model adaptation. We need to be careful if we want to check for residuals or the fitted values so it will be specified in the function **??**

```
> diagnostics <- function (x)
 {
 normality = signif(shapiro.test(x$residuals)$p.value); #check for normal distributed valu
 stat.res = adf.test(x$residuals); #check both residuals
 #of the model for stationarity
 stat.res =signif(adf.test(x$residuals)$p.value);
 stat.res.alt = adf.test(x$residuals)$alternative;
```

```
x$residualsvector = as.vector(x$residuals);
autocorr= dwt(x$residualsvector) ; #check for autocorrelation
indep= signif(Box.test(x$residuals, type="Ljung-Box")$p.value) #check for independence
#lag for season is df: m-1 ( 12-1)
#write if seasonal = TRUE lag=12-1, else write nothing
#there is high evidence that there are non-zero autocorr.
c1= cbind(normality, stat.res, stat.res.alt, autocorr, indep);
c2 = c("normal distribution of residuals",
       "stationarity of residuals",
       "alternative stationarity type",
       "autocorrelation of residuals",
          "independence of residuals");

matrix = as.matrix(c1,c2, desparse.level=1);

return ( matrix )
}
```

The next function compiles **??** the visualization of the distribution of the errors of a forecast function and overlays it with a normally distributed data to depict mistakes in therespective forecast functions.

```
> plotForecastErrors <- function(forecasterrors)
 {
 # make a histogram of the forecast errors:
 mybinsize <- IQR(forecasterrors)/4
 mysd <- sd(forecasterrors)
 mymin <- min(forecasterrors) - mysd*5
 mymax <- max(forecasterrors) + mysd*3
 # generate normally distributed data with mean 0 and standard deviation mysd
 mynorm <- rnorm(10000, mean=0, sd=mysd)
 mymin2 <- min(mynorm)
 mymax2 <- max(mynorm)
 if (mymin2 < mymin) { mymin <- mymin2 }
 if (mymax2 > mymax) { mymax <- mymax2 }
 # make a red histogram of the forecast errors, with the normally distributed data overlai
 mybins <- seq(mymin, mymax, mybinsize)
 hist(forecasterrors, col="red", freq=FALSE, breaks=mybins)
 # freq=FALSE ensures density
 # generate normally distributed data with mean 0 and standard deviation mysd
 myhist <- hist(mynorm, plot=FALSE, breaks=mybins)
 # plot the normal curve as a blue line on top of the histogram of forecast errors:
 points(myhist$mids, myhist$density, type="l", col="blue", lwd=2)
 }
```

2.3 Dataset (CO2-Concentrations)

The first dataset in this tutorial consists of monthly data on CO2- Concentrations in the atmosphere in ppm (parts per million), obtained of daily data measured over time at the climatic station "Mauna Loa" on Hawaii. To download this dataset and store it as a table without the additional explanations run the code below:
To download this dataset, just use the code provided below.

```
> url<-"ftp://aftp.cmdl.noaa.gov/products/trends/co2/co2_mm_mlo.txt"
> dest<-"//csrv05/public$/Elenamarlene/BestpracticeR/timeseries/run.txt"
> download.file(url, dest )
> co2month=read.table(dest, skip=72)
> co2month
> data= co2month[,c(3,5)]
> colnames(data)=c( "year","co2")
```

Note: d̈estr̈epresents a randomly chosen name for a text file in which the CO2- dataset will be stored.

2.4 Dataset Visualization

It can be useful to visualize your original data before you transform it into a time series class .

2.4.1 Plotting the fitted values

```
> # Run a linear model
> attach(data)
> datalm = lm( co2 ~ year)
> # Fit predict values
> MyData=data.frame(year=seq(from=(1958),to=2014, by=0.1))
> pred=predict(datalm, newdata=MyData, type="response", se=T)
> # Plot the fitted values
> plot(year, co2, type="n",las=1, xlab="Year", ylab="CO2 conc. (ppm)",
      main="CO2 concentration in the atmosphere")
> grid (NULL,NULL, lty = 6, col = "cornsilk2")
> points(year, co2, col="cornflowerblue" )
> # Write confidence interval
> F=(pred$fit)
> FSUP=(pred$fit+1.96*pred$se.fit) # make upper conf. int.
> FSLOW=(pred$fit-1.96*pred$se.fit) # make lower conf. int.
> lines(MyData$year, F, lty=1, col="red", lwd=3)
> lines(MyData$year, FSUP,lty=1, col="red", lwd=3)
> lines(MyData$year, FSLOW,lty=1, col="red", lwd=3)
> legend("topleft",c("simple linear regression y~x", "monthly mean data"),
 pch=c(20,20), col=c("red", "cornflowerblue"))
```

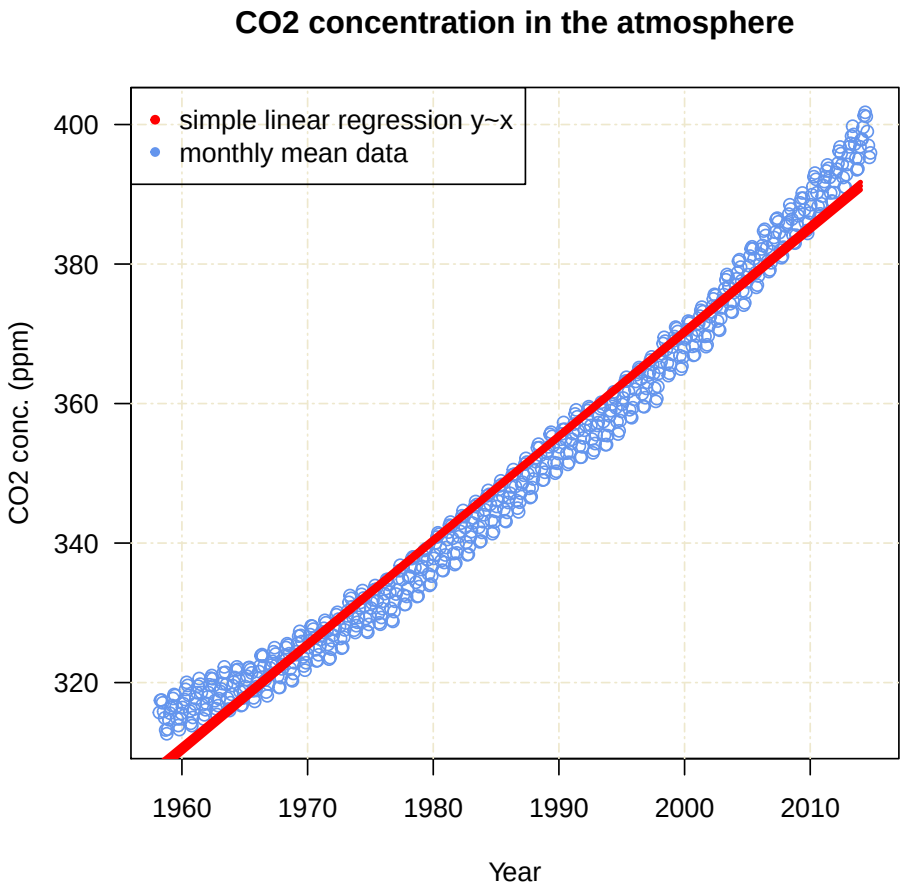


Figure 1: The raw data with a linear model

The plot /refplotorigin can be used to identify potential outliers in the dataset which could possibly bias the future model. Due to the plot we can see, that we don't have any outliers and we can go on, having a look at the added simple poly-1 linear regression. The lm is not accurate in fitting the dataset. There are missing a lot of model predictors.

```
> xtable(summary(datalm)$coef)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-2618.56	16.98	-154.23	0.00
year	1.49	0.01	174.86	0.00

In table /refstdelmthestandarderrorsofthelinearregressionaresmall.Onepossibleattributeofatimese
test(p−value)islowerthanitshouldbe.Inourcasethiswouldmean, thatCO2concentrationsarerisingsi

2.5 Dataset Transformation

It is essential to transform your dataset into a timeseries (ts) if you seek for an accurate and extensive analysis of the data. The data stored as a dataframe needs to be transformed with the important columns into the class of a time series to continue working on it properly. If you have monthly data you have to set the deltat of the function ts() to deltat=1/12 describing the sampling period parts between successive values xt and xt+1. Your time series should somehow look like table 1.

Original Data

```
> xtable(head(data), caption="Original CO2-Data")
```

	year	co2
1	1958.21	315.71
2	1958.29	317.45
3	1958.38	317.50
4	1958.46	317.10
5	1958.54	315.86
6	1958.62	314.93

Table 1: Original CO2-Data

Transformation

```
> yourts=ts(co2, c(1958,3),c(2014,10), deltat=1/12)
> class(yourts)
```

[1] "ts"

```
> time = time(yourts)
```

2.6 Time-Series Visualization

It is also helpful to get a quick overview of the time series class of our data with some plots. The original data and the time series data are the same, thus it is important to have a different class for further working on it.

2.6.1 Time-Series Plot

```
> par(mfrow=c(1,1))
> plot(time, co2,type="n",las=1, xlab="Year", ylab="CO2 conc. (ppm)",
        main="CO2 concentration in the atmosphere")
> grid (NULL,NULL, lty = 6, col = "cornsilk2")
> points(yourts, col="cornflowerblue" )
> lines(year, co2, col="red")
> legend("topleft",c("time series", "monthly mean raw data"),
 pch=c(20,20), col=c("red", "cornflowerblue"))
```

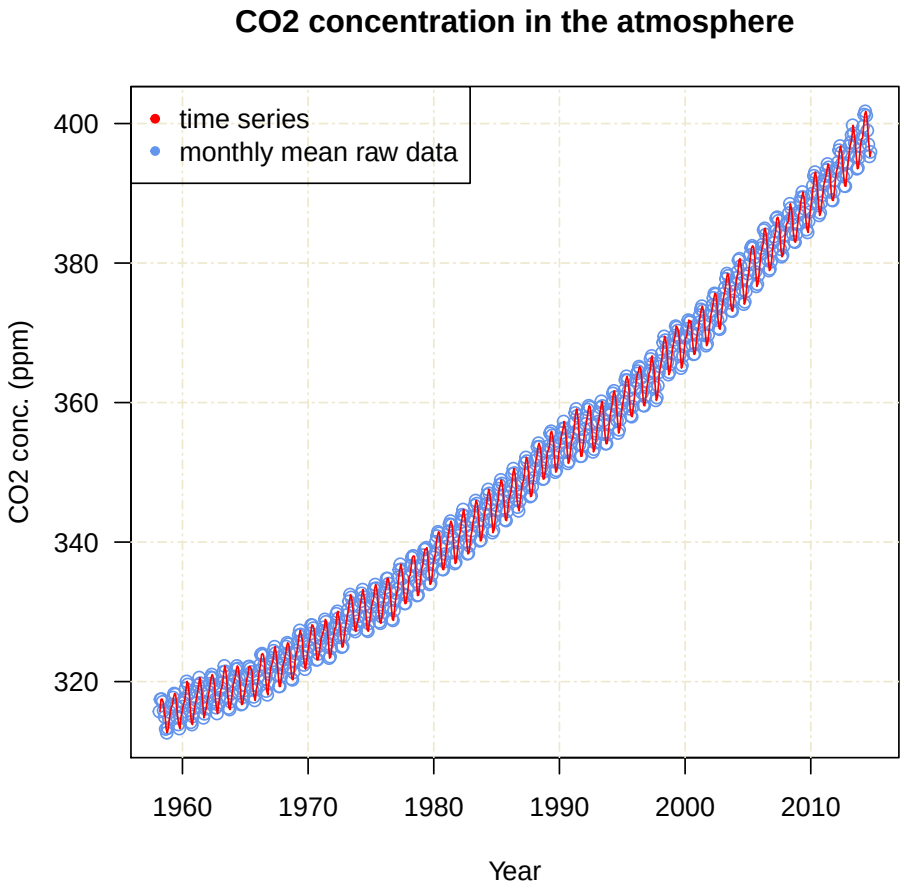
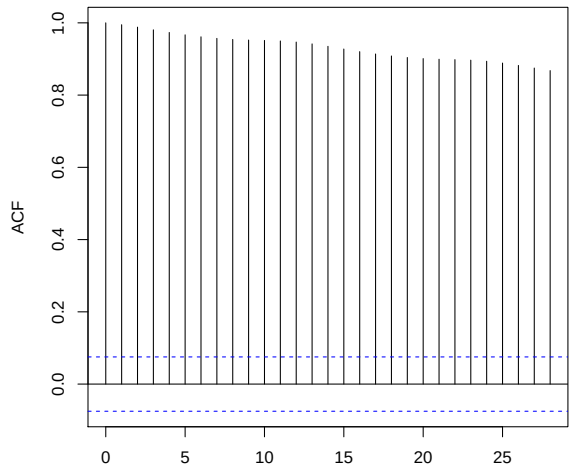


Figure 2: Visualization of the CO2 Concentrations

2.6.2 ACF, PACF, SPECTRUM

2. Correlogram: The correlogram is divided by three different functions all depicting important information on the autocorrelation and the cyclic component of our time series. The problem we face with time series is the possible violation of independence assumption of a model, meaning that the SE is possibly too small. If the data are equally spaced in time, the autocorrelation functions can be used to investigate residual correlations in the model errors. The correlograms produce lags which are either within the CI or outter CI. If a lag is bigger than the CI, you need to include the lag as a order of autocorrelation in your AR or and MA terms which are needed in the following models.
ACF: The autocorrelation function ACF() is one option producing a plot of coefficients of correlation between your time series and lags of itself. If you have a correlation at lag 1 and one at lag 2, the first correlation is reproduced to the second and so on to higher-order lags. This pattern, similar to the one in plot /refcorrelogramlm is like a flow-on effect from the

Figure 3: ACF of CO2 Concentrations with blue dotted lines as CI



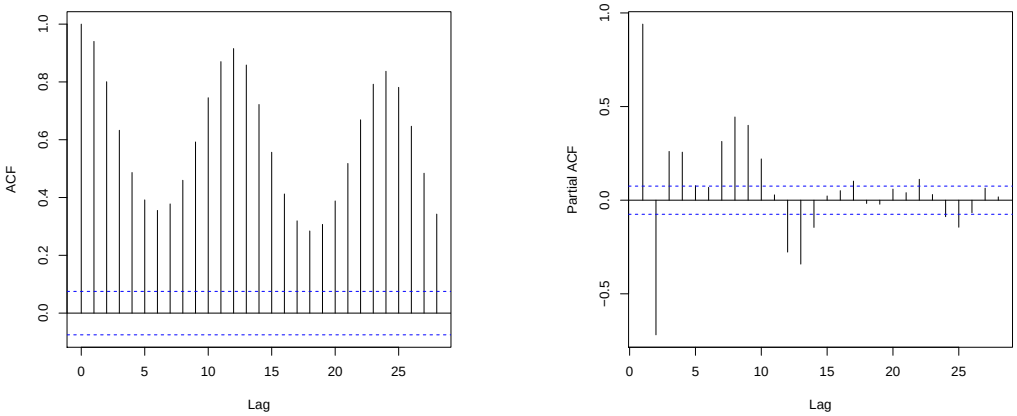
dependence at lag 1, which would be typical for an autoregressive process.

The **PACF (partial ACF)** is the accumulation of correlations between 2 variables excluding the correlation explained by their common correlation. It is not explained in the lower-order-lags. If you have correlations at lag 1 and at lag 2, the PACF computes the difference between the actual correlation at lag 2 and the expected correlation if reprdocution of the correlation at lag 1.

In our correlogram, the lag *p=1* included as a autoregressive order in the model will capture already a lot of the correlation structure.

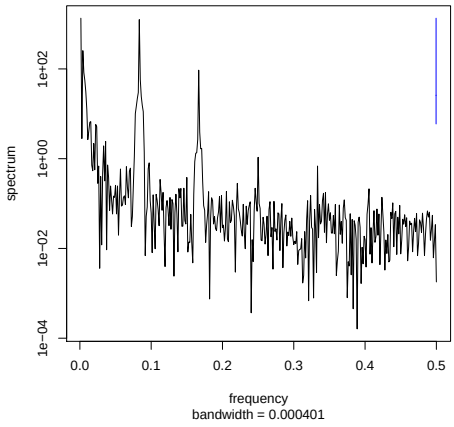
The **spectrum** will be the third function used to check for possible correlation structures showing the spectral density of the time series over frequencies. If a local maximum occurs, *1/(local max.)* can be used as equation to pick a possible periodic term. For our data we have a spectrum max at 0.75, meaning there might be a 12 * periodic cycle (12 months = 1 year) .

The residuals in a time series are serially correlated. The ACF is waving and decreases only slowly, which would be an identification of non-stationarity. We stop here all following diagnostic tests due to the clear autocorrelation in our data and investigate the different components of our time series.



(a) ACF

(b) PACF



(c) spectrum

Figure 4: Correlogram of the residuals of datalm

3 Decomposition of Time Series

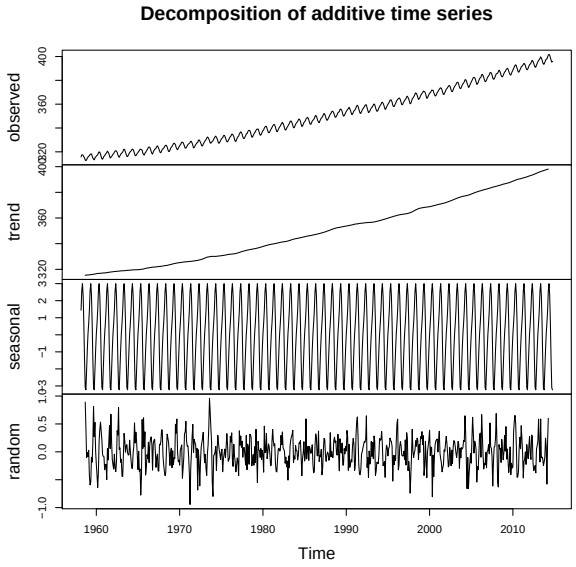
As described in the /refIntroduction , a time series consists usually of 3 components; a trend component, an irregular (random) component and (if it is a seasonal time series) a seasonal component.

Figure /refseasadj depicts the comparison between the data with and without the seasonal fluctuation. In some cases it might be handy to have the model without the seasonal fluctuations to empahize the trend and the random part or to compute means more acurately.

3.1 Decomposing Seasonal Data

We can decompose the ts and plot these components:

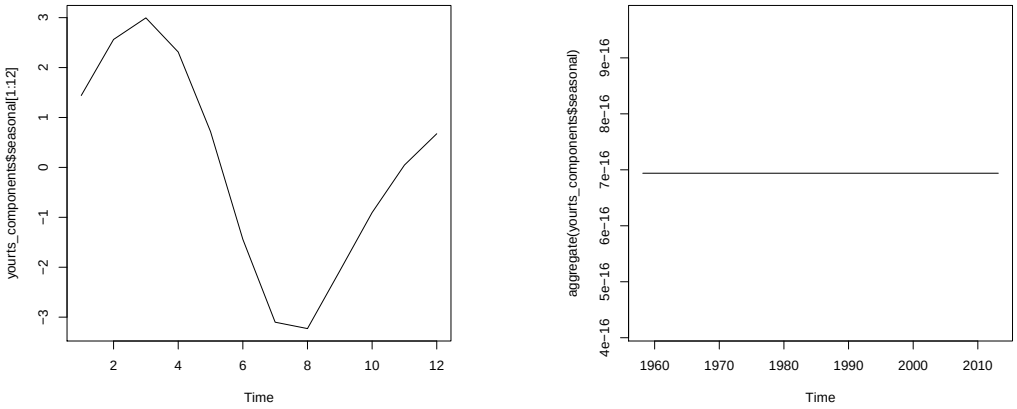
Figure 5: Decomposition of the CO2 Time Series



We can see each component with:

```
> yourts_components<- decompose(yourts)
```

```
> yourts_components$seasonal
```



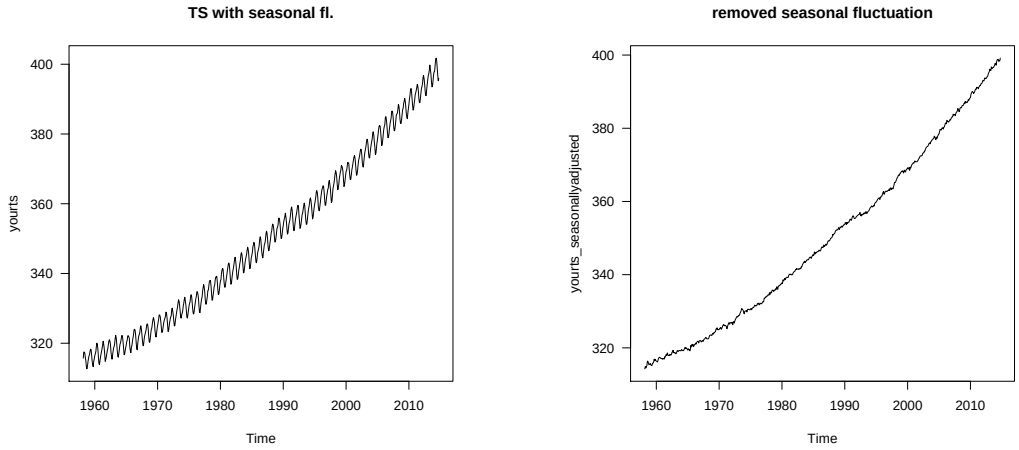
(a) monthly data

(b) annual

Figure 6: The seasonal component across the time

In the figure /refseascomp , the monthly and annual component are depicted, respectively. It seems that our seasonal component is positive in the first half of the year (month 1-6) and is negative in the second half of the year. The annual seasonal component is though constant within our time series. This assumes that we can fully include the seasonal term in an additive model without adding additional random noise to it or biasing the long-term trend.

```
> yourts_seasonallyadjusted <- yourts - yourts_components$seasonal
```



(a) original (b) seasadj.

Figure 7: Comparison of seasonal vs. seasonally adjusted model

Figure /refseasadj depicts the comparison between the data with and without the seasonal fluctuation. In some cases it might be handy to have the model without the seasonal fluctuations to empahize the trend and the random part or to compute means more acurately.

4 Modelling the time series

In this tutorial, the aim of the analysis of a seasonal time series is mainly to generate forecasts and make the time series stationary. For this, we need to define a model which has all the needed components of the time series.

```
> adftable <- function(x) {
 stat.res =signif(adf.test(x)$p.value);
 stat.alt = adf.test(x)$alternative;
 c1 = cbind( stat.res, stat.alt);
 c2 = c("stationarity of time series",
         "alternative");
 matrixsmall = as.matrix(c1,c2);
 colnames(matrixsmall)= c("stationarity p-value",
         "alternative");
 return ( matrixsmall)
 }
```

After looking at the simple linear regression datalm, we were facing some serious problems with our model. To be sure about the stationarity we run an adf.test() giving p-value of ....

```
> xtable(adftable(yourts), main="Adf test for stationarity")
```

```
% latex table generated in R 3.1.1 by xtable 1.7-4 package
% Fri Nov 28 12:47:51 2014
\begin{table}[ht]
\centering
\begin{tabular}{rll}
  \hline
 & stationarity p-value & alternative \\
  \hline
1 & 0.928016 & stationary \\
   \hline
\end{tabular}
\end{table}
```

Also we need a model, which is covering the serial correlation of our residuals. The ACF; PACF and the spectrum gives us certainty that there is some autocorrelation and seasonal fluctuation. glm() cannot allow for autocorrelation. The generalized least squares model is one option that can be used to allow for autocorrelation of standard errors and unequal variances.

4.1 Analysis of Seasonal Data with GLS

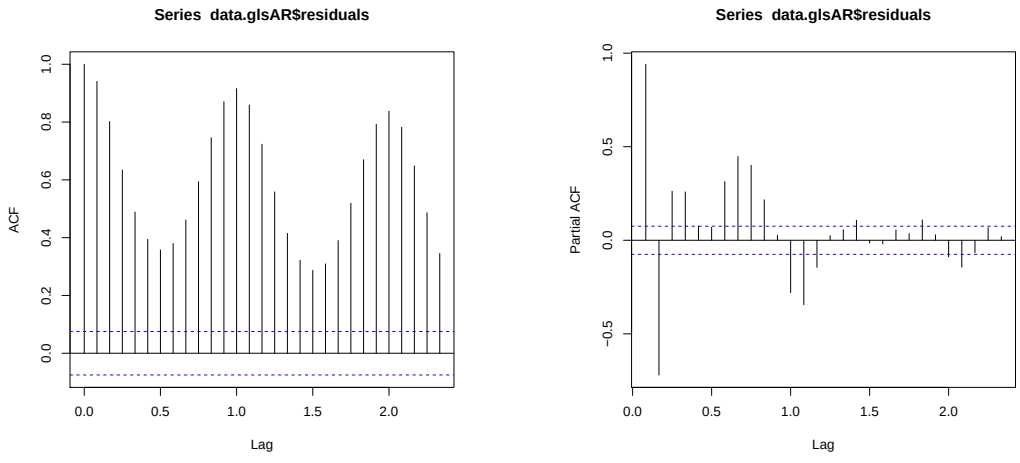
The generalized least squares model is one option that can be used to allow for autocorrelation of standard errors and unequal variances. In the gls() we have different options to choose for our covariance structure. Because our data is not spatially correlated we are not discussing spatial autocorrelation here.

For temporal correlation we have five options:

1. corAR1: in ACF exponential decreasing values of correlation with time distance
2. corARMA: either autoregressive order or moving average order or both
3. corCAR1: continuous time (time index does nto have to be integers)
4. corCompSymm: correlation does not decrease with higher distance
5. corSymm: general correlation only for few observations only, often overparameterized

Our first gls() model accounts for the AR1, which is clearly visible in the PACF. Our lag 1 is the lag computed from the linear regression ACF.

```
> data.glsAR = gls(yourts ~ time,cor= corAR1(acf(resid(datalm))$acf[2]))
> save(data.glsAR, file="data.glsAR.RData")

> load("data.glsAR.RData")
```



(a) original (b) seasadj.

Figure 8: Correlogram of GLS with AR(1) structure

We still face a lot of problems with the first GLS model only including 1 auto regressive order. One option is to allow the AR to use more parameters and/or to include a moving average or error variance to the model. This can be handled via the corARMA. We tried 2 versions, one with 1 lag and 1 moving average, the other with 2 lags and 2 moving averages.

The 0.2 are starting values for Phi, which are in the modelling process optimized. The next models are thus:

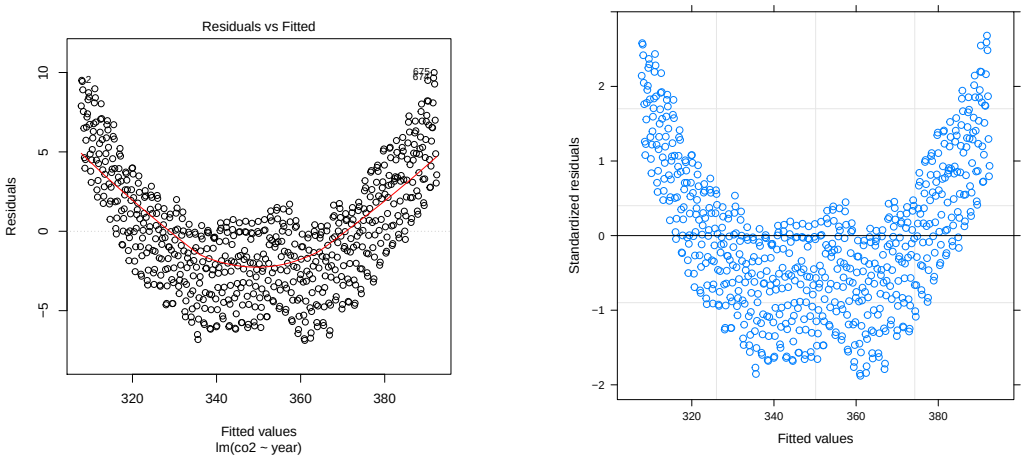
```
> dataglsARMA10 = gls ( yourts ~ time, cor = corARMA (p=1, q=0 ))
> dataglsARMA11 = update (dataglsARMA10, cor=corARMA(p=1, q=1))
> dataglsARMA12 = update (dataglsARMA10, cor=corARMA(p=1, q=2))
> dataglsARMA22 = update (dataglsARMA10, cor=corARMA(p=2, q=2))
> dataglsARMA21 = update (dataglsARMA10, cor=corARMA(p=2, q=1))
> save(dataglsARMA10, file="dataglsARMA10.RData")
> save(dataglsARMA11, file="dataglsARMA11.RData")
> save(dataglsARMA12, file="dataglsARMA12.RData")
> save(dataglsARMA22, file="dataglsARMA22.RData")
> save(dataglsARMA21, file="dataglsARMA21.RData")
>

> load(file="dataglsARMA10.RData")
> load( file="dataglsARMA11.RData")
> load( file="dataglsARMA12.RData")
> load( file="dataglsARMA22.RData")
> load( file="dataglsARMA21.RData")
```

The AIC (Akaike Information Criterion) is used to compare the different models. The smaller the AIC value, the better fits the model to the original dataset. To compare all the models we use anova and the best model is so far the dataglsARMA22 with the lowest AIC and significantly better than the ARMA(2,1), which is itself not better than the ARMA(1,2), but this is better than the previous 2 models.

```
> anova(dataglsARMA10,dataglsARMA11,dataglsARMA12,dataglsARMA21,dataglsARMA22)

              Model df      AIC      BIC     logLik   Test
dataglsARMA10     1  4 2189.575 2207.651 -1090.7874
dataglsARMA11     2  5 1760.113 1782.709  -875.0567 1 vs 2
dataglsARMA12     3  6 1587.709 1614.824  -787.8545 2 vs 3
dataglsARMA21     4  6 1549.076 1576.191  -768.5380
dataglsARMA22     5  7 1495.348 1526.982  -740.6742 4 vs 5
               L.Ratio p-value
dataglsARMA10
dataglsARMA11 431.4614  <.0001
dataglsARMA12 174.4045  <.0001
dataglsARMA21
dataglsARMA22  55.7275  <.0001

> diagnostics(dataglsARMA22)

     normality    stat.res   stat.res.alt
[1,] "1.8632e-08" "0.928016" "stationary"
     autocorr            indep
[1,] "0.111186713059271" "0"
```

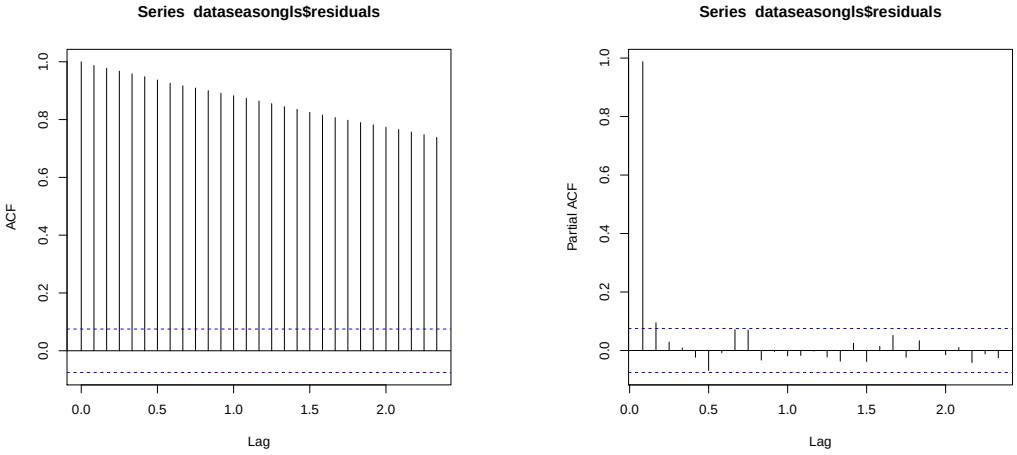
Now, the best correlation structure seems to be selected, however the seasonal component is still missing. We need to include that in a simple version:

```
> seas = cycle(yourts)
> dataseasongls = gls(yourts ~ time + factor(seas), cor=corARMA(c(0.2,0.2,0.2,0.2),p=2, q=
> save(dataseasongls, file="dataseasongls.RData")

> load("dataseasongls.RData")


> par(mfrow=c(1,1))
> plot(dataseasongls$residuals,ylab="Residuals", xlab="Year",
      las=1, type="p", col="cornflowerblue",
      main="Residuals of dataseasongls over time")
>
```



(a) residuals (b) seasadj.

Figure 9: Residuals fitted vs. observed in (a) LM and (b) GLS



(a) ACF (b) PACF

Figure 10: Correlogram of seasonality-including model

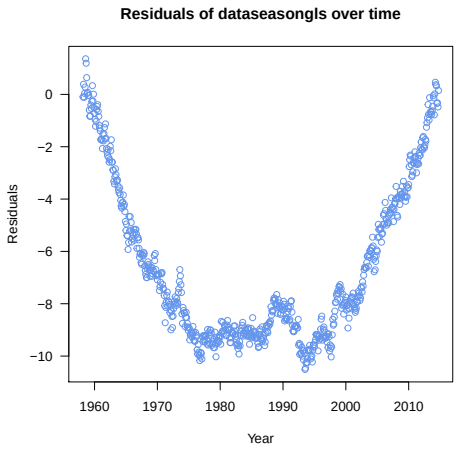


Figure 11: Residuals of seasonality-including model

The problem with anova.gls is that it cannot compare gls() with different fixed effects. The added seasonal term is such an effect, thus the anova does not work here. We need to compare the AIC by hand and see an improvement.

```
> AIC(dataglsARMA22)
```

[1] 1495.348

```
> AIC(dataseasongls)
```

[1] 401.2934

```
> #write function choosing best model
```

```
> ts.plot(cbind(yourts, dataglsARMA22$fitted,
                dataseasongls$fitted),
          lty=1:2, col=c(1,2,3),
          main="Compare mean monthly data with gls model")
> legend(1960,400,c("Original",
                    "Fitted for Autocorrelation",
                    "Fitted for Seasonality"),
         col=c(1,2,3),lty=c(1, 2,3))
```

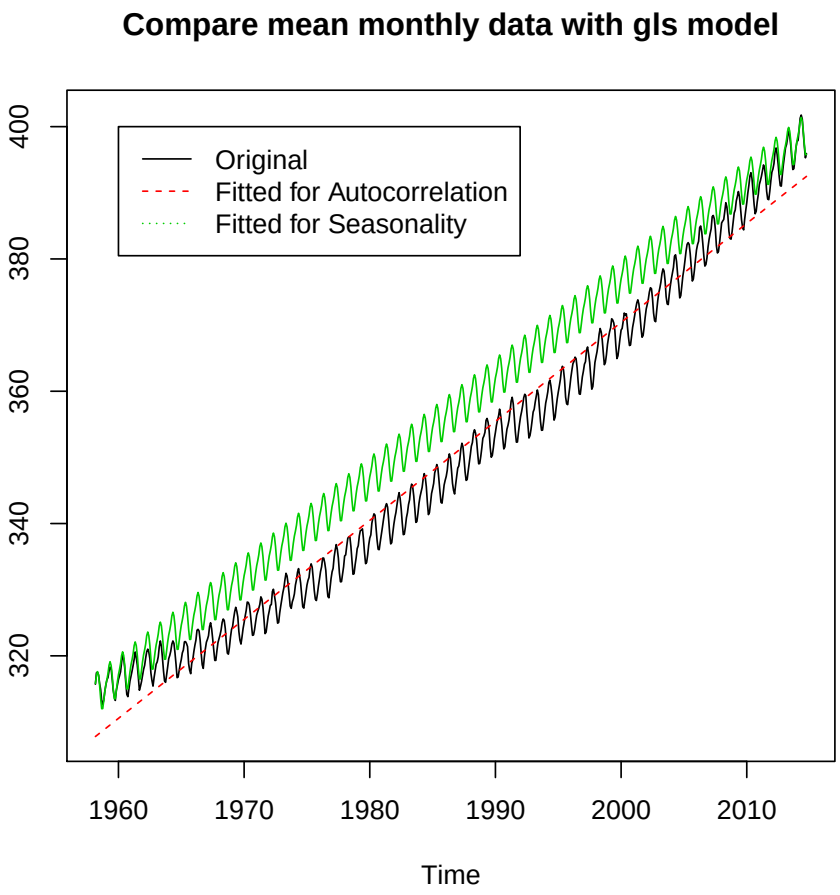


Figure 12: Comparison of seasonality-including model with only autocorrelation-including model

In the figure 12 you see the options we were trying so far to come closer to our adequate model. One option is the ARMA22 GLS including autocorrelation, the second option is to include the seasonality by adding a fixed effect (factor(season)) to our gls. The seasonal effect inclusion clearly improves our model. But the original data is slightly curved

and we see in plot /refresseas that the residuals get bigger as soon as the bending of the original data sets in and flatten again as soon as the model was close to the end point in 2014.

We should include a quadratic term and generalize our seasonality with a continuous sin-cos wave:

```
> SIN = COS = matrix(nr=length(yourts), nc=6)
> for (i in 1:6) {
   COS[,i] <- cos(2*pi*i*time(yourts))
   SIN[,i] <- sin(2*pi*i*time(yourts))
 }
> time = time(yourts)
```

also was done by hand in self writing seasonality removal above Thus we do not know how many changes in the wave we need to include, we include 6 changes in the wave to be certain that the next model can follow the best wave option.

Before this, add a quadratic term (polynomial (x,2)), x = years) will change the linear regression by including a slight bending in the fitted values over time, which we saw in the original data.

```
> harmonizedgls<-gls(yourts ~ time + I(time^2) +
                       COS[,1]+SIN[,1]+COS[,2]+SIN[,2]+
                       COS[,3]+SIN[,3]+COS[,4]+SIN[,4]+
                       COS[,5]+SIN[,5]+COS[,6]+SIN[,6],
                     corr=corAR1(acf(dataseasongls$residuals)$acf[2]))
> save(harmonizedgls, file="harmonizedgls.RData")
> harmonizedARMAgls<-gls(yourts ~ time + I(time^2) +
                         COS[,1]+SIN[,1]+COS[,2]+SIN[,2]+
                         COS[,3]+SIN[,3]+COS[,4]+SIN[,4]+
                         COS[,5]+SIN[,5]+COS[,6]+SIN[,6]
                        , cor=corARMA(p=2, q=2))
> save(harmonizedARMAgls, file="harmonizedARMAgls.RData")

> load("harmonizedARMAgls.RData")
> load("harmonizedgls.RData")

> AIC(harmonizedgls) #even smaller

[1] 396.86

> AIC(harmonizedARMAgls) #even smaller

[1] 360.4082

> xtable(anova(harmonizedgls,harmonizedARMAgls))

% latex table generated in R 3.1.1 by xtable 1.7-4 package
% Fri Nov 28 12:47:54 2014
\begin{table}[ht]
\centering
\begin{tabular}{rlrrrrrlrr}
  \hline
 & call & Model & df & AIC & BIC & logLik & Test & L.Ratio & p-value \\
  \hline
harmonizedgls & gls(model = yourts \~{} time + I(time\verb|^|2) + COS[, 1] + SIN[, 1] +
  harmonizedARMAgls & gls(model = yourts \~{} time + I(time\verb|^|2) + COS[, 1] + SIN[, 1
   \hline
\end{tabular}
\end{table}
```

```
> par(mfrow=c(1,1))
> ts.plot(cbind(yourts,dataseasongls$fitted, harmonizedARMAgls$fitted), col=c(1,2,3),
        main="Compare mean monthly data with gls model")
> legend(1960,400,c("Original", "Included seasonality ",
                    "Polynm. + seasonality"),col=c(1,2,3), pch=c(20,20,20))
```

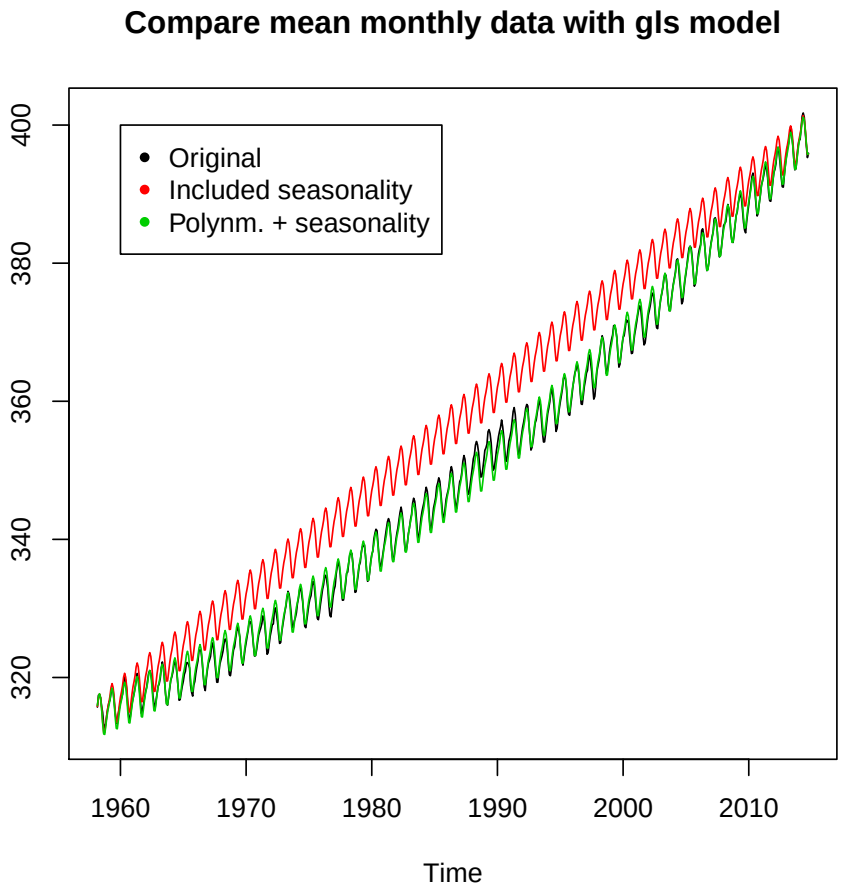
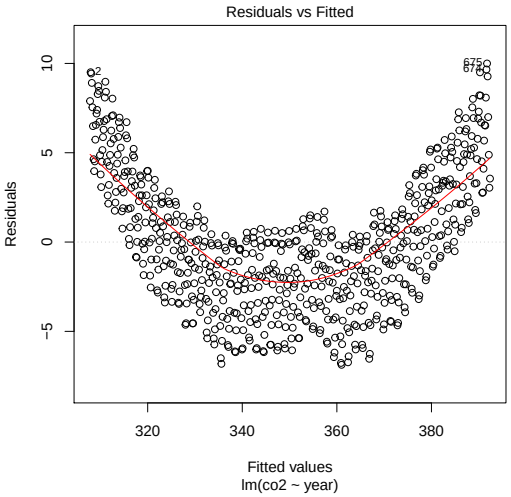


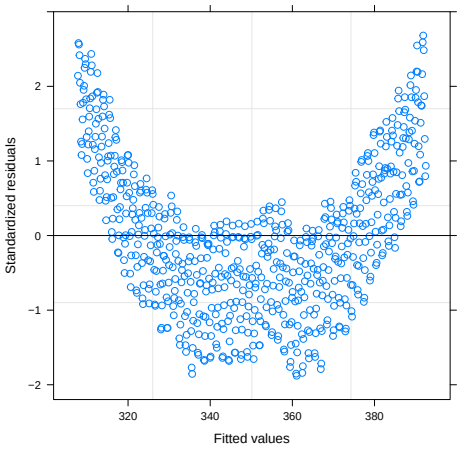
Figure 13: Comparison of season and quadr. term included model

TEXT!!

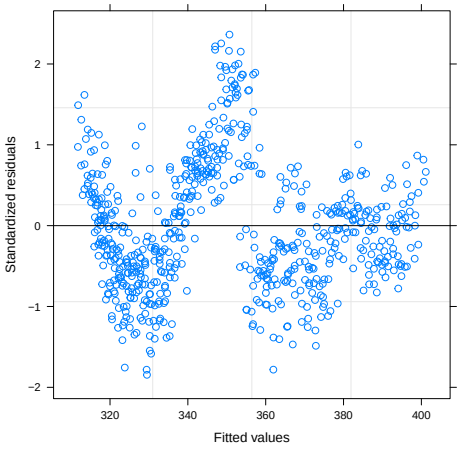
```
> diagnostics(harmonizedARMAgls)

     normality     stat.res   stat.res.alt
[1,] "5.22872e-08" "0.290355" "stationary"
     autocorr           indep
[1,] "0.18333304071156" "0"
```



(a) LM



(b) ARMA



(c) quadrARMA

Figure 14: Comparison of residuals vs. fitted values for lm, gls/arma, and the quadratic gls/arma

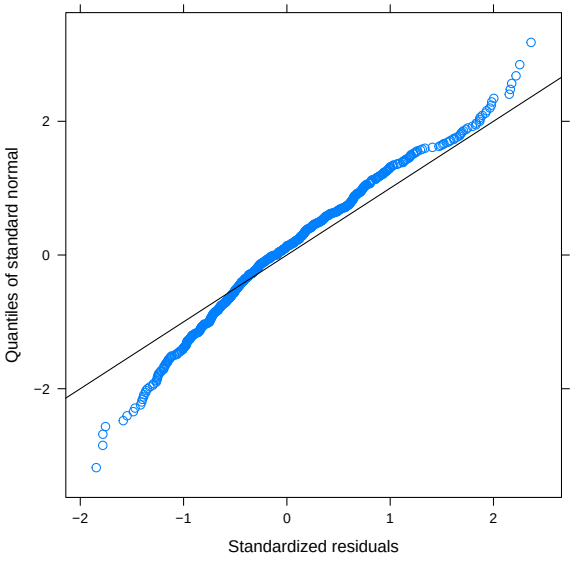


Figure 15: look at norm. distribution

bestmodel with smallest AIC:

```
> anova.all = anova( harmonizedgls,harmonizedARMAgls)
> #as.matrix(anova.all)
> bestmodel= anova.all$Model[anova.all$AIC ==min(anova.all$AIC)]
> bestmodel
```

```
[1] 2
```

Model	df	AIC	BIC	logLik	Test	L.Ratio	p-value
harmonizedgls	17	396.86	473.36	-181.43			
harmonizedARMAgls	20	360.41	450.40	-160.20	1 vs 2	42.45	3.21739e−09

In the beginning we were talking about the underestimation of standard errors in the linear model, if not considering the autocorrelation. As we see, the standard errors are now less underestimated.

	LM	ARMA22	bestmodel
(Intercept)	16.98	45.71	4322.03
year	0.01	0.02	4.35

After some trials to find the best gls model to our data, there are still some diagnostics failed. Our fitted values residuals are still not normally distributed, here you could think of normalizing them. Also the fitted values are still dependent, but our model seems not to care for this. A good result is, that the autocorrelation was solved. The biggest problem here is that we have still non-stationarity in our model and need to redo all the process after differencing our time series.

4.2 Modelling time series with ARIMA

ARIMA modelling is a time efficient option to the gls(). The ARIMA function does not care about stationarity of the time series, it will adjust the model to be stationary afterwards. The function auto.arima() will provide an adequately adjusted model with all needed components.

```
> autoarima = auto.arima(yourts)
```

Series: yourts ARIMA(1,1,1)(2,1,2)[12]

Coefficients: ar1 ma1 sar1 sar2 sma1 sma2 0.2166 -0.5763 -0.2861 -0.0364 -0.5992 -0.2292 s.e. 0.0936 0.0782 NaN 0.0404 NaN NaN

$\text{sigma}^2 \text{estimatedas} 0.08927 : \text{loglikelihood} = -149.46 \text{AIC} = 312.92 \text{AICc} = 313.09 \text{BIC} = 344.44$

```
> fitted=fitted(autoarima)
> par(mfrow=c(1,1))
> ts.plot(cbind(yourts, harmonizedARMAgls$fitted ), col=c(1,2,3))
> lines( fitted, col=3)
> legend(1960,400,c("Original",  "GLS", "Autoarima "),col=c(1,2,3), pch=c(20,20,20))
```

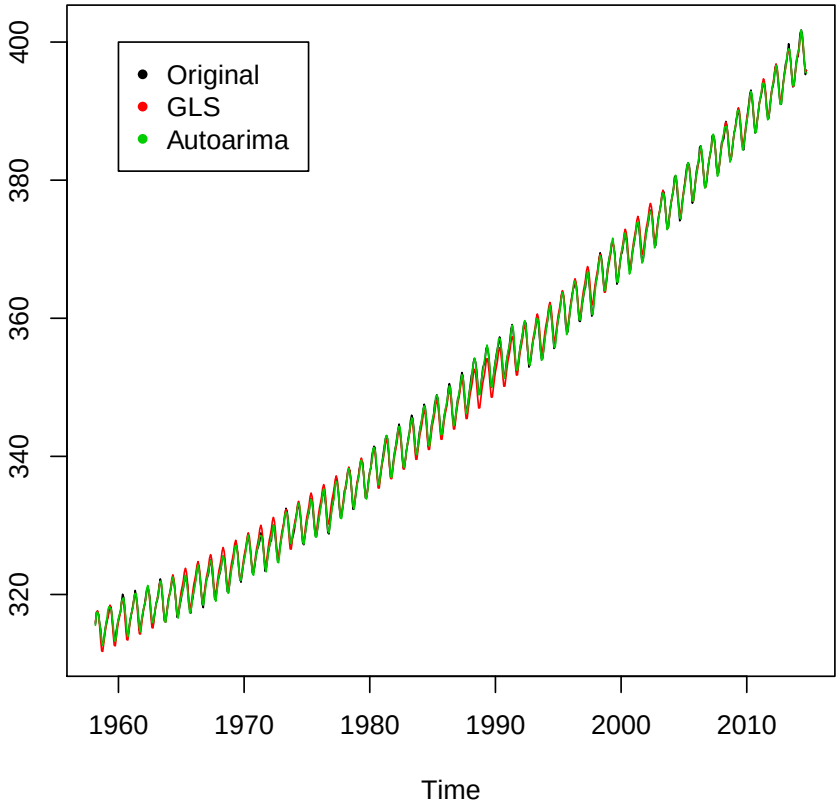
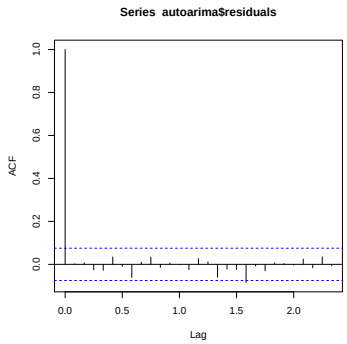


Figure 16: Fitted ARIMA values on time series

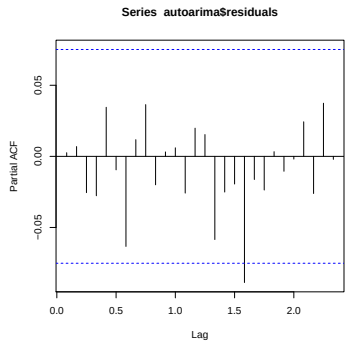
The comparison plot /reffittedarima shows that the bestmodel from gls() and the automatically adjusted arima model are barely distinguishable from the original dataset. However, the arima gives slightly better AIC than the best gls.

```
> AIC(autoarima)
```

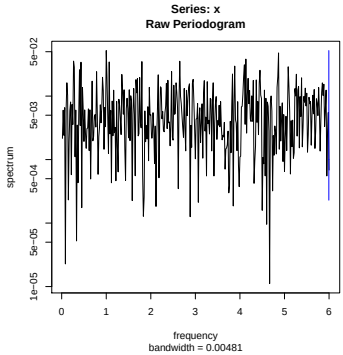
[1] 312.9221



(a) ACF



(b) PACF



(c) quadrARMA

Figure 17: Correlogram for ARIMA

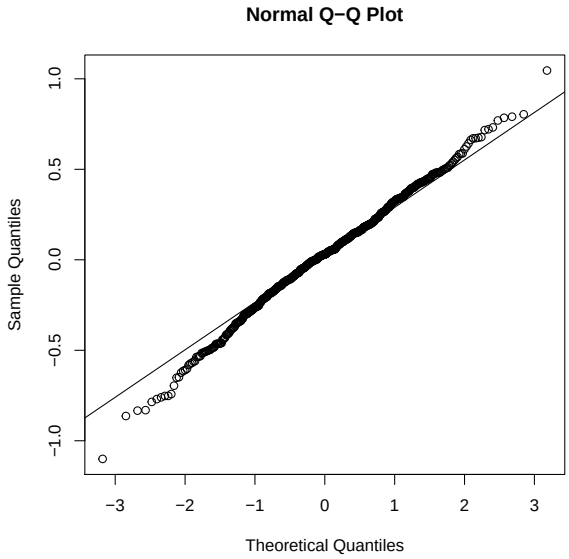


Figure 18: QQ Plot for ARIMA

```
> diagnostics(autoarima)
```

normality stat.res stat.res.alt autocorr [1,] ”0.037855” ”0.01” ”stationary” ”1.97955655809946” indep [1,] ”0.946122”

The diagnostics show a better estimate of our fitted model. The autocorrelation is solved

way better than in the gls-model. Also the stationarity is given now and the fitted values are independend.

5 Forecasts

Forecasting means generally to predict from past values (x1,x2,x3,...,xn) some future values x(n+k).

We can choose from three different options:

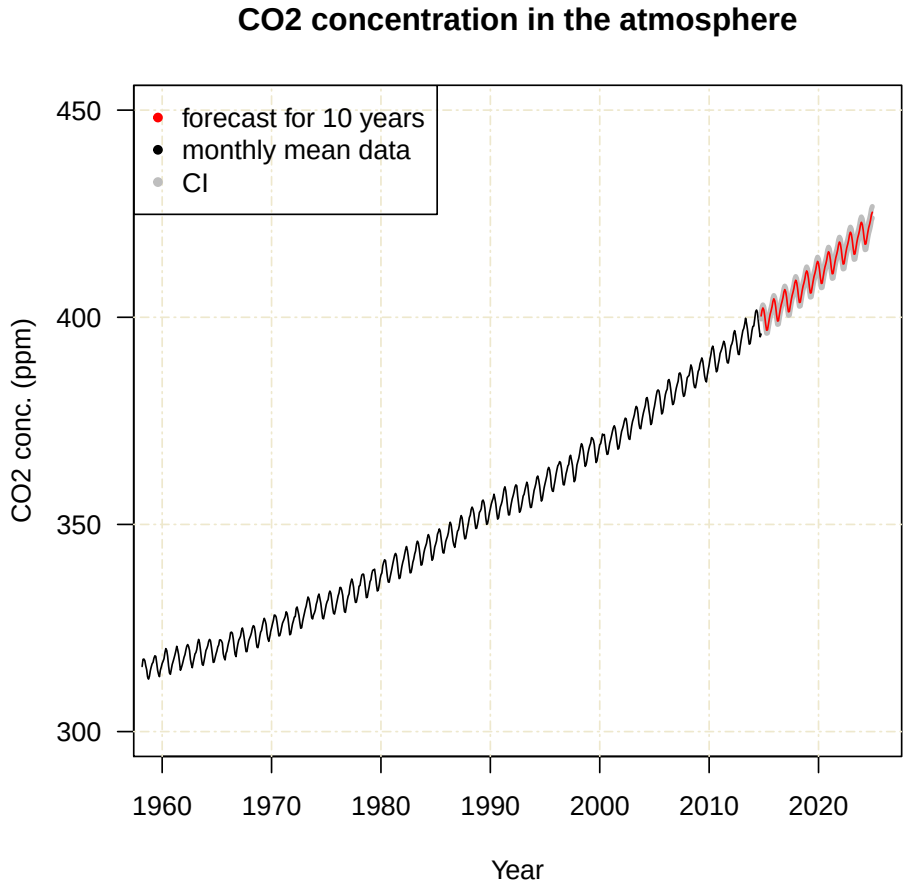
1. predict()
2. Holt Winters hw()
3. Arima forecasts forecast.Arima()

If we have a time series that can be described using an additive model,we can short-time forecast using exponential smoothing.
A conditions we need to check are the forecast errors distributions. They need to be uncorrelated and normally distributed with mean zero and constant variance. To check the forecast errors we have the visualization of the function plotForecastErrors from above.

5.1 Forecast with predict()

We use our best gls-model for this, the poly-2 model with SIN-COS-wave and predict for 10 years from the last period given in the data.

```
> newtime= ts(start=c(2014, 10),end=c(2024,12),deltat=1/12)
> pred = predict(harmonizedARMAgls, newdata=newtime, se=T)
> TIME <- as.numeric(time)
> time.df <- data.frame(TIME=TIME, COS, SIN)
> colnames(time.df)[-1] <- paste0("V", 1:12)
> smoothed <- gls(as.numeric(yourts) ~ TIME + I(TIME^2) + V1 + V2 + V3 + V4 + V5
                  +V6 +V7+V8 +V9 +V10 +V11 +V12,
                  corr=corAR1(acf(dataseasongls$residuals)$acf[2]),
                  data=time.df)
> new.df <- cbind.data.frame(TIME=as.numeric(time(newtime)),
                             COS=COS[1:123,], SIN=SIN[1:123,])
> colnames(new.df)[-1] <- paste0("V", 1:12)
> pred = predictSE(smoothed, newdata=new.df, se.fit=T)

> plot(yourts, type="n",las=1, xlim=c(1960, 2025),
      ylim=c(300, 450), xlab="Year", ylab="CO2 conc. (ppm)",
      main="CO2 concentration in the atmosphere")
> grid (NULL,NULL, lty = 6, col = "cornsilk2")
> points(yourts ,type="l" )
> par(mfrow=c(1,1))
> lines(as.numeric(time(newtime)), pred$fit, col="red")
> F=(pred$fit)
> FSUP=(pred$fit+1.96*pred$se.fit) # make upper conf. int.
> FSLOW=(pred$fit-1.96*pred$se.fit) # make lower conf. int.
> lines(new.df$TIME, FSUP,lty=1, col="grey", lwd=3)
> lines(new.df$TIME, FSLOW,lty=1, col="grey", lwd=3)
> lines(new.df$TIME, F, lty=1, col="red", lwd=1)
> legend("topleft",c("forecast for 10 years", "monthly mean data", "CI"),
        pch=c(20,20), col=c("red", "black", "grey"))
```

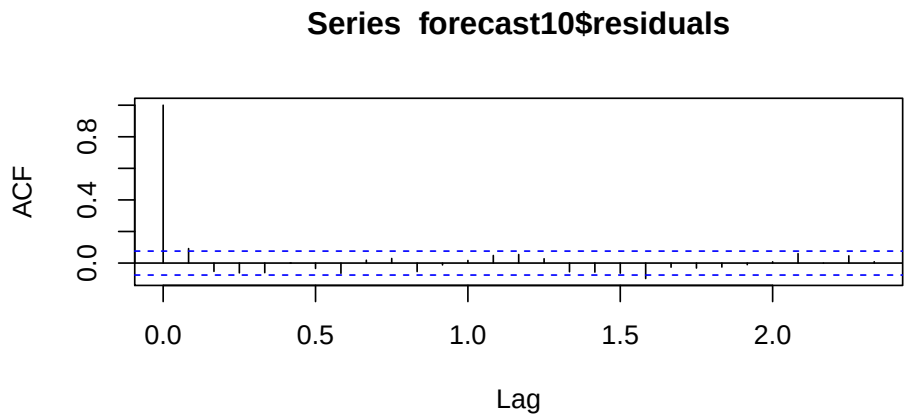


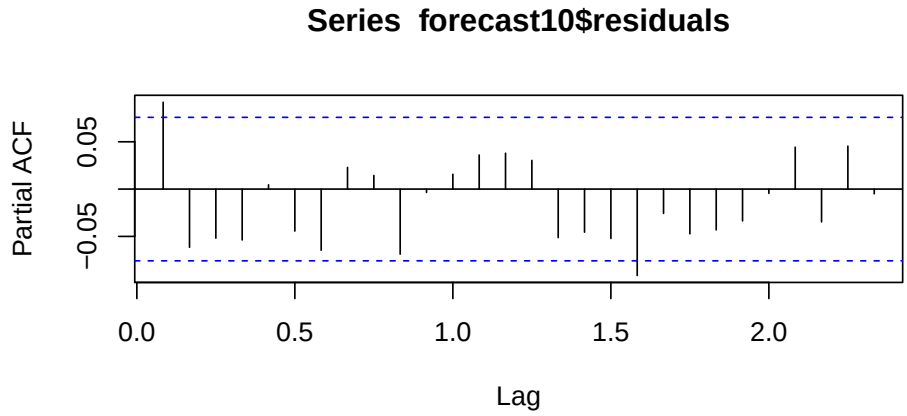
5.2 Holtwinters forecast function

The alpha value tells us the weight of the previous values for the forecasting. Gamma is used for the seasonality. Values of alpha that are close to 0 mean that little weight is placed on the most recent observations when making forecasts of future values and that the predicted values are highly smoothed estimates. If alpha is near 1, little smoothing is done and the estimates are at approximately previous xt.

It is important not to specify aphla, beta, gamma in order to include errors, trend and seasonal component in the forecast.We use the original data for hw() and predict in the period of 120*1 month = 10 years. Subsequently, we plot the predicted values with the original data and check the diagnostics.

```
> forecast <- HoltWinters(yourts)
> forecast10 <- forecast.HoltWinters(forecast,h=120)

> par(mfrow=c(1,1))
> plot.forecast(forecast10,shadecols = "oldstyle")

> par(mfrow=c(2,1))
> acf(forecast10$residuals)
> pacf(forecast10$residuals)
```





5.3 ARIMA forecast function

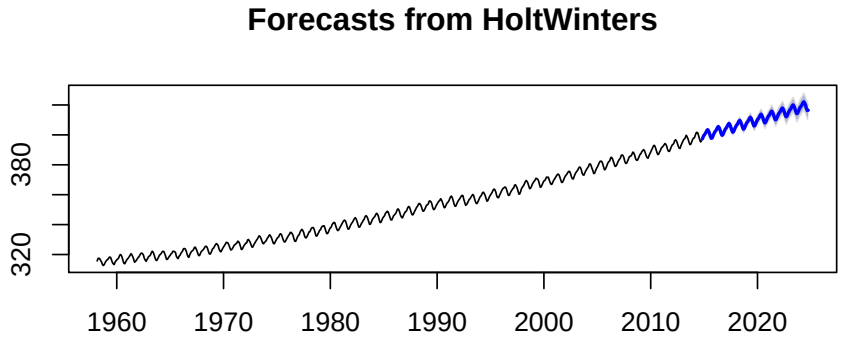
Now we try to forecast with the autoarima function as our bestmodel, which would save alot of time.

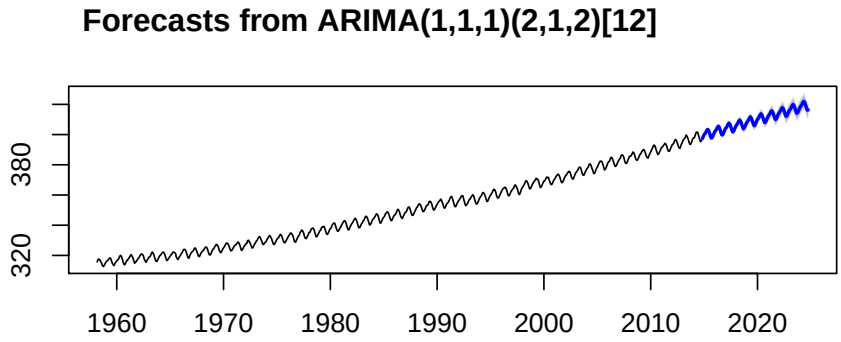
```
> forecast.arima = forecast.Arima(autoarima, h=120)
```

5.4 Comparison of HW and ARIMA

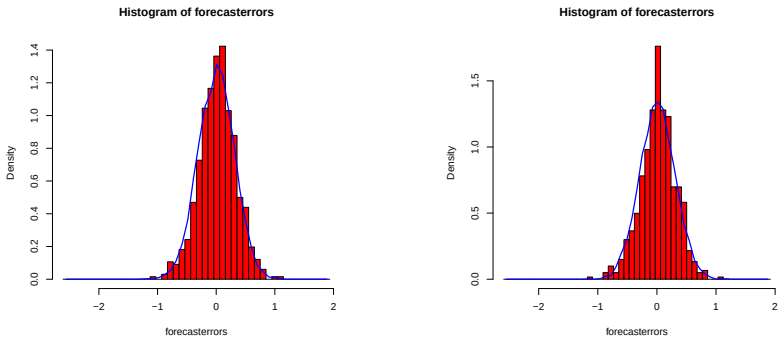
Lets compare what it better.

```
> par(mfrow=c(2,1))
> plot.forecast(forecast10)
> plot.forecast(forecast.arima)
>
```





Now we need to check for error distirbution with the plotForecastErrors function from above:



(a) hw()

(b) arima

Figure 19: Forcast error distribution

The histogram of the time series shows that the forecast errors are roughly normally distributed and the mean seems to be close to zero.

Run diagnostics

```
> diagnostics(forecast10)
```

```
     normality  stat.res stat.res.alt autocorr
[1,] "0.357443" "0.01"   "stationary" "1.80314031521739"
```

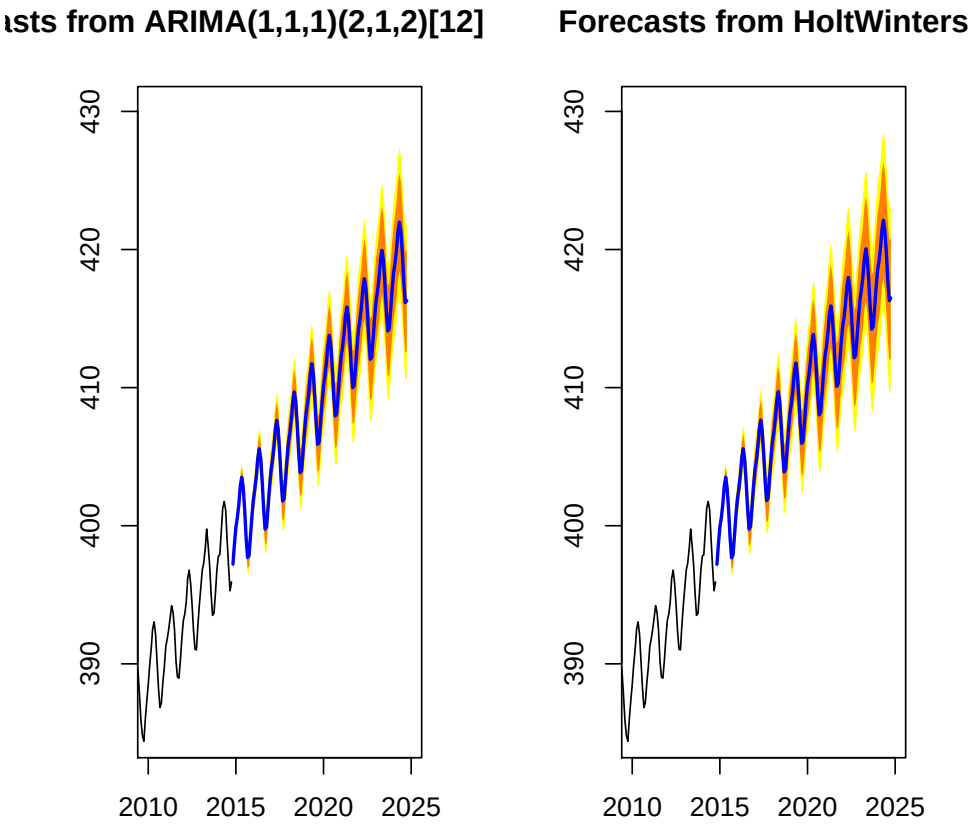
```
     indep
[1,] "0.0174092"
```

```
> diagnostics(forecast.arima)
```

```
     normality  stat.res stat.res.alt autocorr
[1,] "0.037855" "0.01"   "stationary" "1.97955655809946"
     indep
[1,] "0.946122"
```

Now we can have a final look at both the forecast functions and also here, they are barely distinuigshable from each other. Both fit well.

```
> par(mfrow=c(1,2))
> plot(forecast.arima, xlim=c(2010,2025), ylim=c(385,430),shadecols = "oldstyle")
> plot.forecast(forecast10 ,xlim=c(2010,2025), ylim=c(385,430),shadecols = "oldstyle")
```



5.5 Seasonal Decomposition of Time Series by Loess

Forecasting using stl objects is a fourth option, which is straitforward and plotting us nearly same results as other options.

```
> plot(stlf(yourts, lambda=0, h =120))
> (tslm(yourts~time(yourts)))

Call:
lm(formula = formula, data = "yourts", na.action = na.exclude)

Coefficients:
 (Intercept)  time(yourts)
   -2618.494         1.494
```

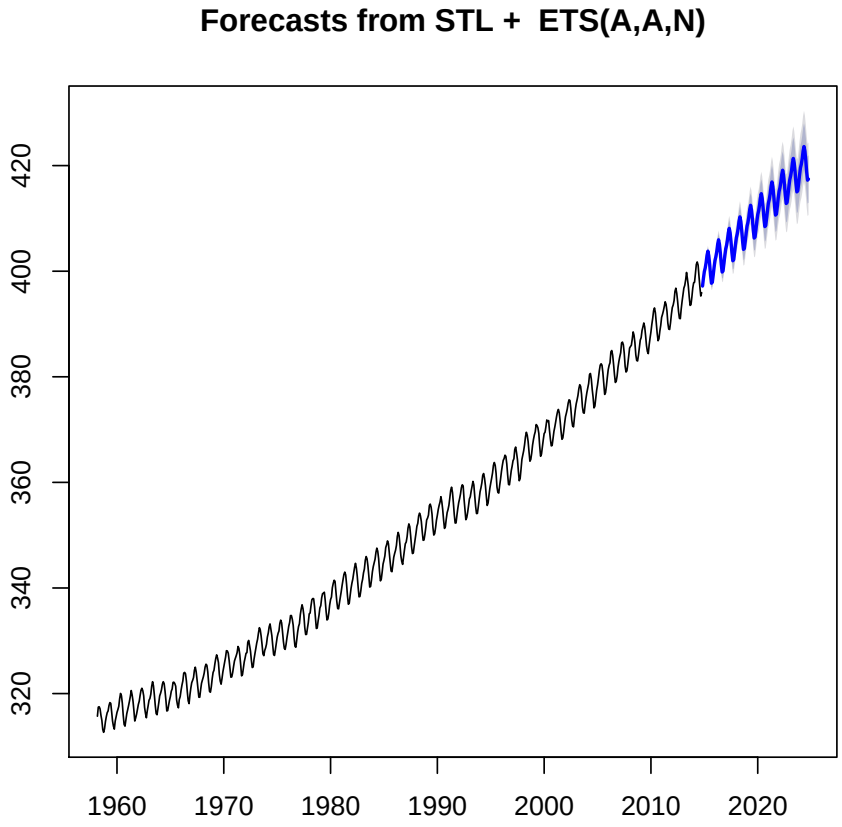


Figure 20: Forecasting using Loess

6 compabitily with Linux and Windows when working together on a sweave document.

If you have Linux and have problems working on a same sweave document with Windows users because the line breaks and other text configurations are different, try to type:

7 Links and Further Reading

7.1 Modelling time series with gam

The gam() generalized additive model could maybe also help finding a quick solution for time series modelling. Thus this model is not adequately adjusted there are examples for modelling also seasonally components with gam(). We present it here for further reading on it, for example on this website: ...

```
> model = gam ( yourts ~ s(time))
> summary(model)
```

Family: gaussian Link function: identity

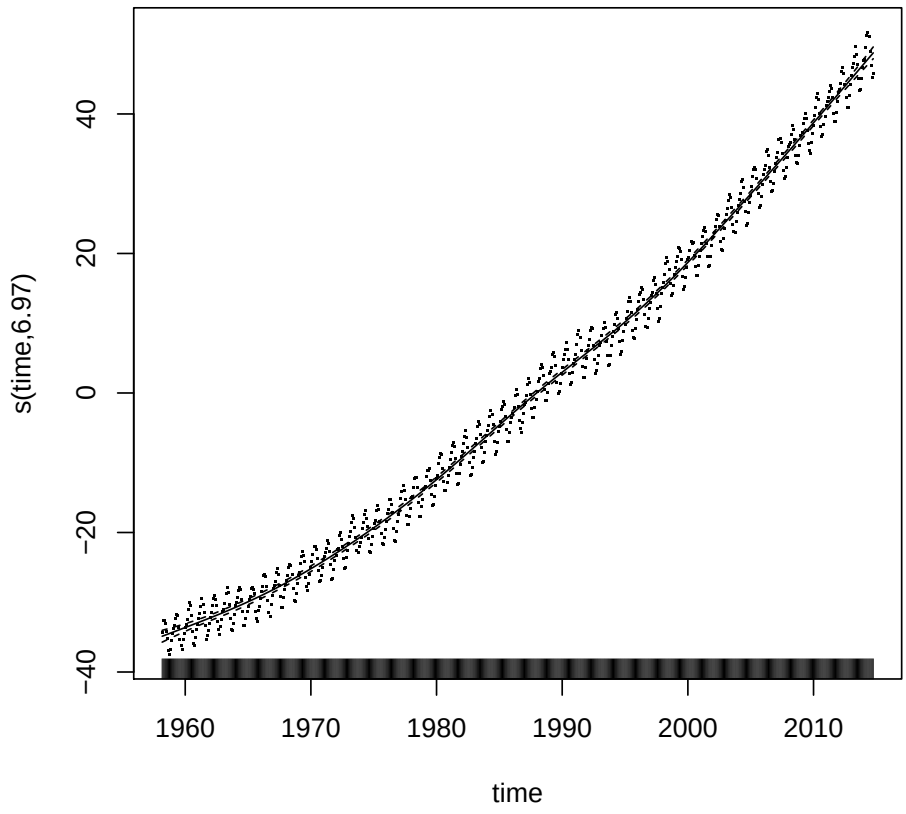
Formula: yourts s(time)

Parametric coefficients: Estimate Std. Error t value Pr(>|t|) (Intercept) 350.09641 0.08181 4279 <2e-16 *** — Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Approximate significance of smooth terms: edf Ref.df F p-value s(time) 6.969 8.047 11258 <2e-16 *** — Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

R-sq.(adj) = 0.993 Deviance explained = 99.3GCV = 4.6049 Scale est. = 4.551 n = 680

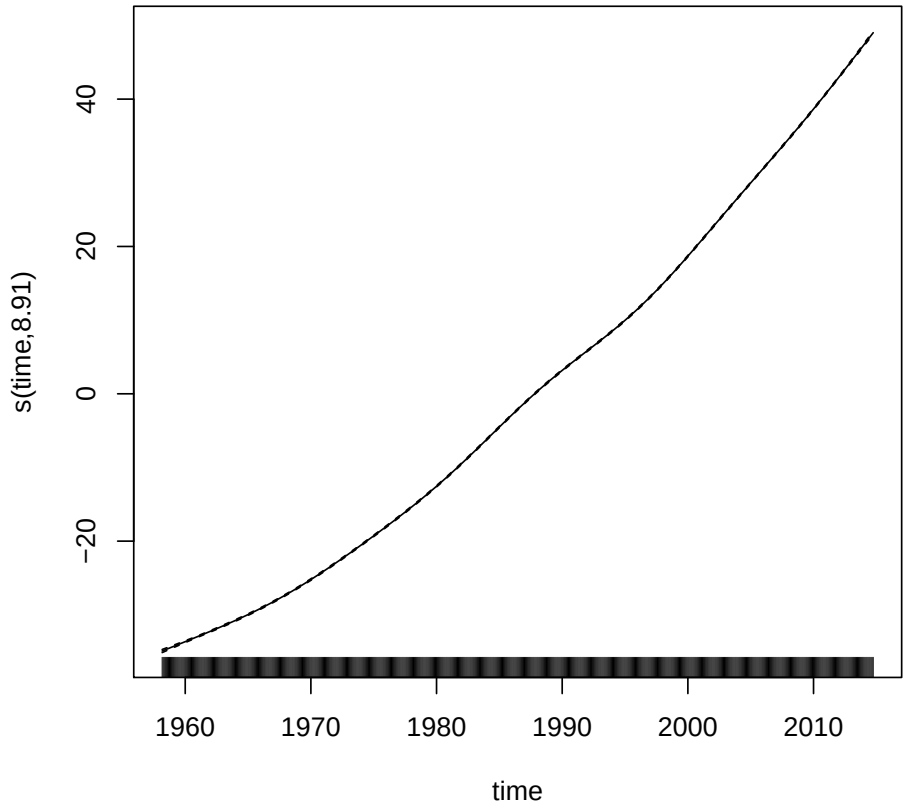
```
> par(mfrow=c(1,1))
> plot.gam(model, residuals=T, scheme=c(2,1), all.terms=T)
```




```
> smoothed4gam <- gam(as.numeric(yourts) ~ s(time)  +
                        COS[,1]+SIN[,1]+COS[,2]+SIN[,2]+
                        COS[,3]+SIN[,3]+COS[,4]+SIN[,4]+
                        COS[,5]+SIN[,5]+COS[,6]+SIN[,6]
                      , cor=corARMA(p=2, q=2))
> #summary(smoothed4gam)
> AIC(smoothed4gam)
```

[1] 926.9626

```
> plot(smoothed4gam)
> #8.91 is the edf:array of estimated degrees of freedom for the model terms, calculated i
```



and in:
http://cran.r-project.org/web/views/TimeSeries.html
Another Exemple Datasets are avaliable at:
http://www.comp-engine.org/timeseries/browse-data-by-category
https://datamarket.com/data/list/?q=provider:tsdl

8 Acknowledgements

Don't forget to thank TeX and R and other opensource communities if you use their products! The correct way to cite R is shown when typing "`citation()`", and "`citation("mgcv")`" for packages.