Time Series Analysis - A Tutorial for Temporally Correlated Time Series

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

November 28, 2014

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.

This chapter covers some fundamental theory and is meant to give you a better understanding about the following chapters that give attention to the analysis of time series.

1.1 Definition

A time series is a record of values of a certain variable of interest taken at different points in time. Data are observed at equally spaced time intervals (Discrete time series) and the method of measurement is supposed to be consistent over time.

1.2 Applications

Depending on the field of work, time series analysis can be useful for several applications.

Geosciences and meteorology

- Weather forecasting, trends in weather patterns

Business and finance (econometrics)

- Stock market analysis, business forecasting

Multivariate statistical data analysis

- RS image analysis

Medicine

- Epidemic analysis

1.3 Goals depending on the study question

Climatologist interested in global warming

- Interested in the long term trend of CO2 or temperature, thus ignoring the daily/monthly/seasonal cycles

Economist interested in demand for electricity

- Interested in the long term trends(due to population growth? Global warming?) but also the daily/monthly/seasonal peaks

Epidemiologist interested in preparing for the flu season

- Not really interested in long term trends, the focus is set more on monthly/seasonal cycles and errors (abnormal spikes in the flu rates)

1.4 Decomposition

Decomposing a time series means separating it into its constituent components, which are usually a trend component and an irregular component, and if it is a seasonal time series, a seasonal component:

1.) **Trend**

2.) **Cycles** (including seasonal)

3.) **Irregular components**

These components can be described as follows:

$$X_t = T_t + S_t + C_t + R_t$$

where:

Xt = Value of the series at time t
Tt = trend component of the series
St = Cyclical or seasonal component with a period S
Rt = Random effect for which we have no explanation

Note: This idea is very old and is now out of favor but it is still widely used

Example: Decomposition of monthly deaths from lung diseases in the UK

```
> plot(ldeaths)
```

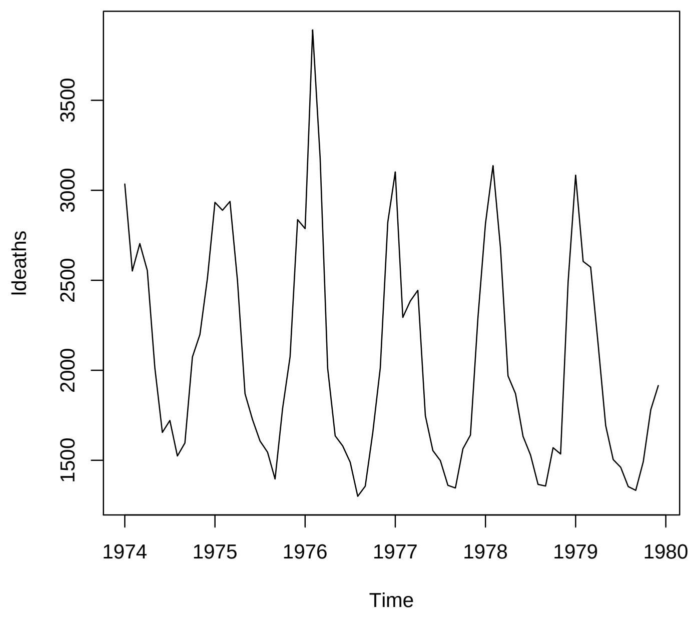


Figure 1: Plot of monthly deaths from lung diseases in the UK

```
> ldeathcomponents = decompose(ldeaths)
> plot(ldeathcomponents)
```

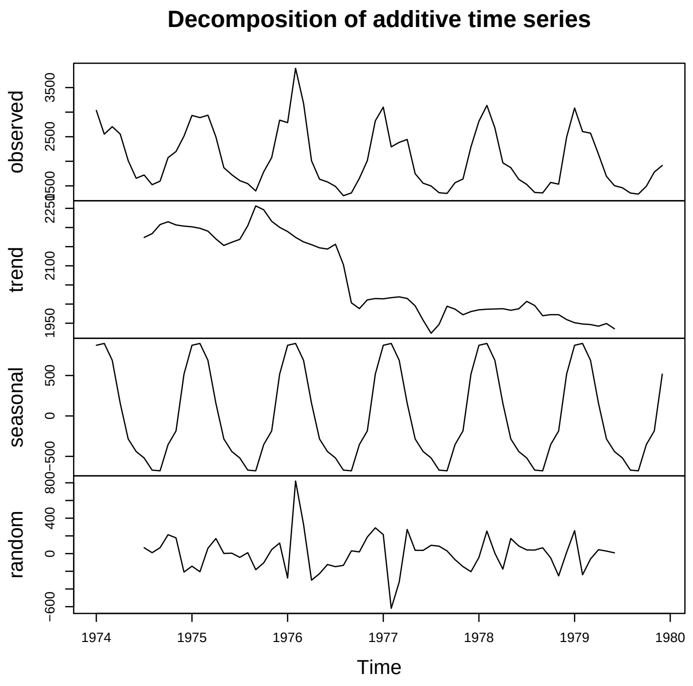


Figure 2: Decomposition of monthly deaths from lung diseases in the UK

1.4.1 Decomposing Seasonal Data

A seasonal time series consists of a **trend component**, a **seasonal component** and an **irregular component**. Decomposing the time series means separating the time series into these three components: that is, estimating these three components.

To estimate the trend component and seasonal component of a seasonal time series that can be described using an additive model, we can use the **"decompose()"** function in R. This function estimates the trend, seasonal, and irregular components of a time series that can be described using an additive model.

The function "decompose()" returns a list object as its result, where the estimates of the seasonal component, trend component and irregular component are stored in named elements of that list objects, called "seasonal", "trend", and "random" respectively.

1.5 Stationarity and differencing

A stationary time series is one whose statistical properties such as **mean, variance, autocorrelation, etc.** are all **constant over time**. Most statistical forecasting methods are based on the assumption that the time series can be rendered approximately stationary (i.e., "stationarized") through the use of mathematical transformations.

A **stationarized series** is relatively **easy to predict**: you simply predict that its **statistical properties will be the same in the future as they have been in the past!** The predictions for the stationarized series can then be "untransformed," by reversing whatever mathematical transformations were previously used, to obtain predictions for the original series. (The details are taken care of by R.)

Thus, finding the sequence of transformations needed to stationarize a time series often **provides important clues in the search for an appropriate forecasting model**. Stationarizing a time series through **differencing** (where needed) is an important part of the process of fitting an ARIMA mode.

Another reason for trying to stationarize a time series is to be able to obtain **meaningful sample statistics** such as means, variances, and correlations with other variables. Such statistics are useful as **descriptors of future behavior** only if the series is stationary.

For example, if the **series is consistently increasing over time**, the sample mean and variance will grow with the size of the sample, and they will **always underestimate the mean and variance in future periods**. And if the mean and variance of a series are not well-defined, then neither are its correlations with other variables. **For this reason you should be cautious about trying to extrapolate regression models fitted to nonstationary data.**
Most business and economic time series are far from stationary when expressed in their original units of measurement, and even after deflation or seasonal adjustment they will typically still exhibit trends, cycles, random-walking, and other non-stationary behavior.

If the series has a stable long-run trend and tends to revert to the trend line following a disturbance, it may be possible to stationarize it by de-trending (e.g., by fitting a trend line and subtracting it out prior to fitting a model, or else by including the time index as an independent variable in a regression or ARIMA model), perhaps in conjunction with logging or deflating.

Such a series is said to be trend-stationary. However, sometimes even de-trending is not sufficient to make the series stationary, in which case it may be necessary to transform it into a series of period-to-period and/or season-to-season differences. If the mean, variance, and autocorrelations of the original series are not constant in time, even after detrending, perhaps the statistics of the changes in the series between periods or between seasons will be constant.

Such a series is said to be difference-stationary. (Sometimes it can be hard to tell the difference between a series that is trend-stationary and one that is difference-stationary, and a so-called unit root test may be used to get a more definitive answer.)

1.5.1 How to make a series X stationary

1.) Check if there is variance that changes with time

- Make variance constant with log or square root transformation

- Call the transformed data X*

2.) Remove the trend in mean with regular differencing or fitting a trend line

- Call the new series X**

- The correlogram of X** should only have a few significant spikes at small lags

3.) If there is seasonal cycle left in the data, we must seasonally difference the series too

- Call the new series X**

1.6 Important terminology

Dependence

Correlation of observations of one variable at one point in time with observations of the same variable at prior points in time **(Serial correlation / autocorrelation)**

Stationarity

The **mean and variance** of the series **remains constant** over the time series (e.g., no systematic change in the mean, no trend)

Differencing

Data pre-processing step which de-trends the data to achieve stationarity

Specification

Using **diagnostic tests**, specifying the type of time series model to apply to the series

- Auto-regressive(AR), Moving average (MA), ARMA (combined) or ARIMA (combined integrated)

- Non linear models also possible

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)
> library(dlm)
> setwd("//csrv05/public$/Elenamarlene/BestpracticeR/timeseries/FINAL_VERSION/")
```

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*.

```
> diagnostics <- function (x)
 {
 normality = signif(shapiro.test(x$residuals)$p.value); #check for normal distributed values #
 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 *plotForecastErrors* 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 overlaid:
 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/backup/Elena/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"))
```

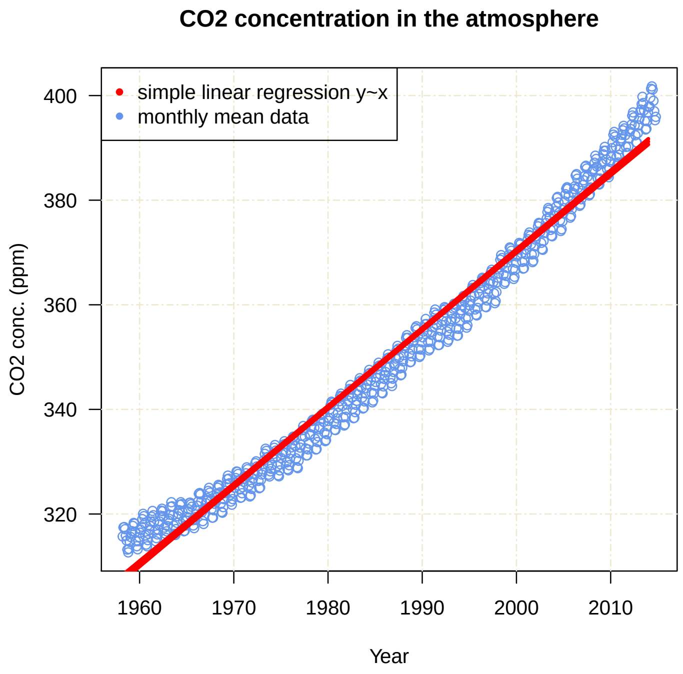


Figure 3: The raw data with a linear model

The plot 3 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, caption="Coefficients")
```

In the table 1, the standard errors of the linear regression are small. One possible attribute of a time series is, that the residuals are serially correlated. With this autocorrelation, the standard errors shown in the lm would be underestimated, thus far too small. Subsequently, the resulting significance from the

	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

Table 1: Coefficients

t-test (p-value) is lower than it should be. In our case this would mean, that CO2 concentrations are rising significantly and that we have a enormous problem with our atmospheric composition.

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 2: 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, however it is important to have a different class of *ts* 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"))
```

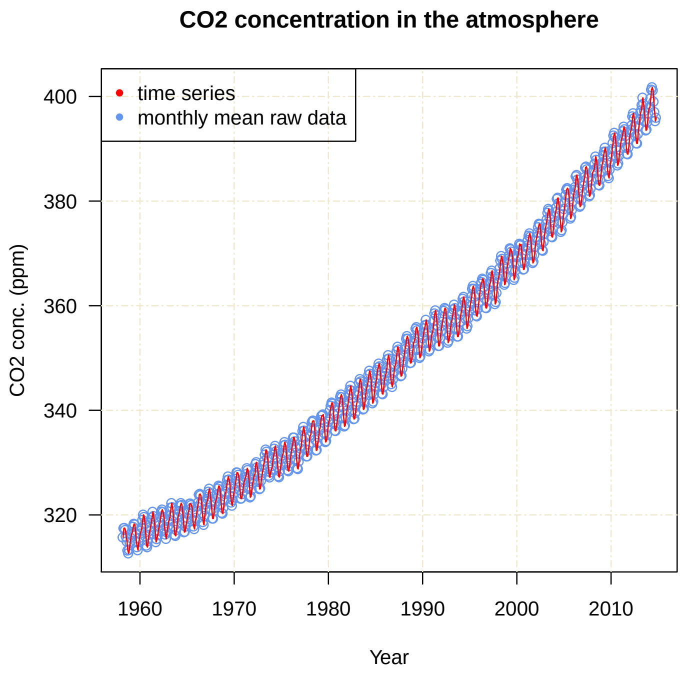


Figure 4: Visualization of the CO2 Concentrations

2.6.2 ACF, PACF, SPECTRUM

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 exceeding 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.

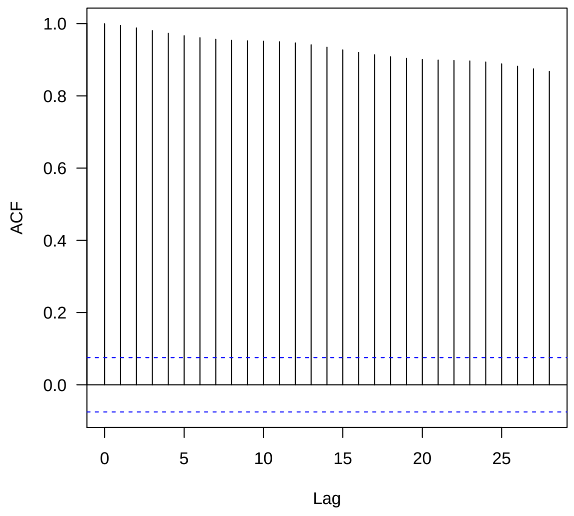
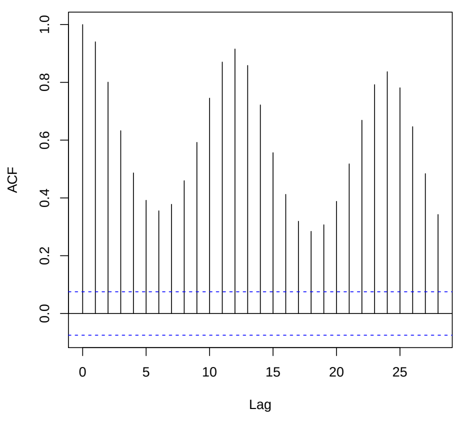


Figure 5: ACF of CO2 Concentrations with blue dotted lines as CI

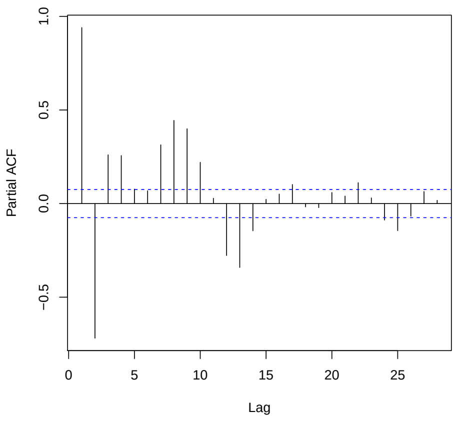
ACF: The autocorrelation function ACF() is one option producing a plot of coefficients of correlation between your time series and lags of itself. If you have a correlation at lag 1 and one at lag 2, the first correlation is reproduced to the second and so on to higher-order lags. This pattern, similar to the one in plot /refcorrelogramlm is like a flow-on effect from the 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.

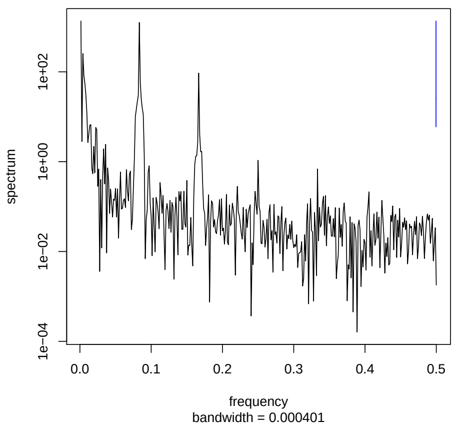
The **spectrum** will be the third function used to check for possible correlation structures showing the spectral density of the time series over frequencies. If a local maximum occurs, *1/(local max.)* can be used as equation to pick a possible periodic term. For our data we have a spectrum max at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



(a) ACF



(b) PACF



(c) spectrum

Figure 6: Correlogram of the residuals of datalm

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.

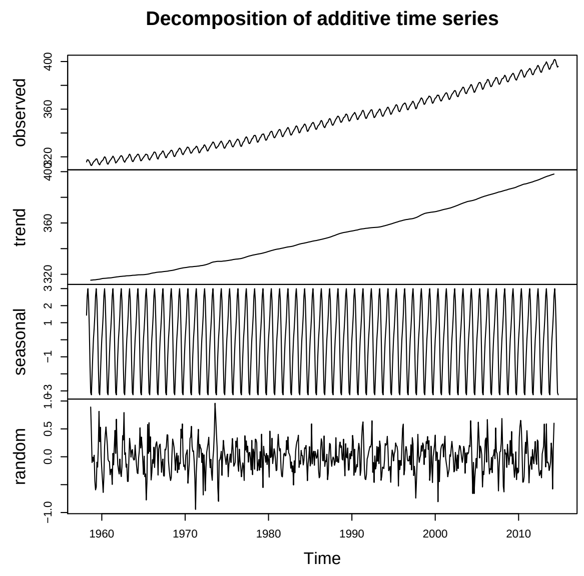
3 Decomposition of Time Series

As described in the Introduction, 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.

3.1 Decomposing Seasonal Data

We can decompose the ts and plot these components:

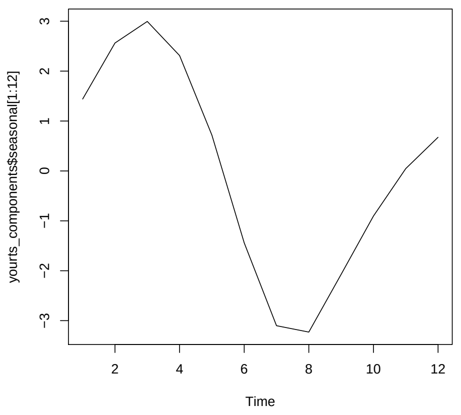
Figure 7: Decomposition of the CO2 Time Series



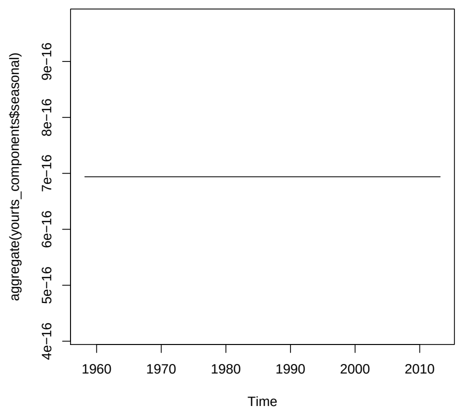
We can see each component with:

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

```
> yourts_components$seasonal
```



(a) monthly data

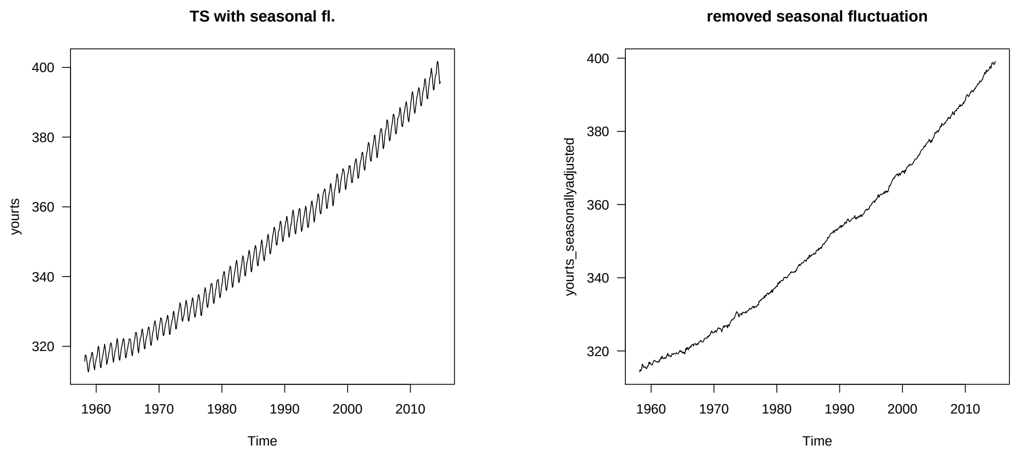


(b) annual

Figure 8: The seasonal component across the time

In figure 8 , 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 9: Comparison of seasonal vs. seasonally adjusted model

Figure 9 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 0.92, having a clear non-stationarity, what the future model needs to change.

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

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.

	stationarity p-value	alternative
1	0.928016	stationary

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

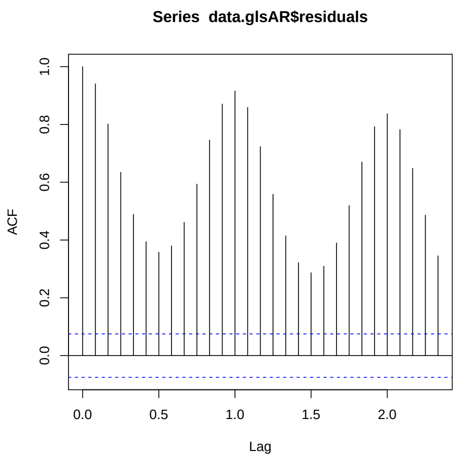
3. corCAR1: continuous time (time index does nto have to be integers)

4. corCompSymm: correlation does not decrease with higher distance

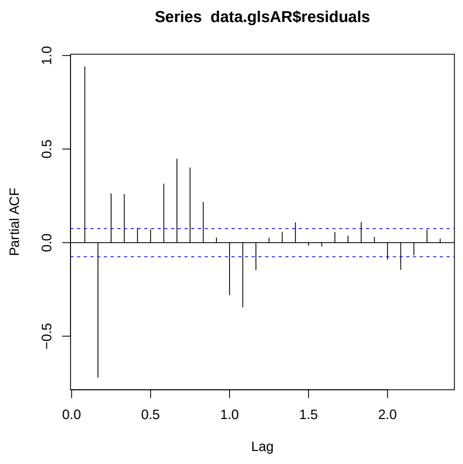
5. corSymm: general correlation only for few observations only, often overparameterized

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 10: 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:

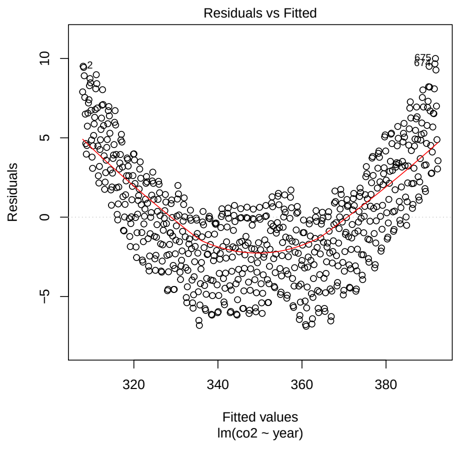
```
> dataglsARMA10 = gls ( yourts ~ time, cor = corARMA (p=1, q=0 ))
> dataglsARMA11 = update (dataglsARMA10, cor=corARMA(p=1, q=1))
> dataglsARMA12 = update (dataglsARMA10, cor=corARMA(p=1, q=2))
> dataglsARMA22 = update (dataglsARMA10, cor=corARMA(p=2, q=2))
> dataglsARMA21 = update (dataglsARMA10,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

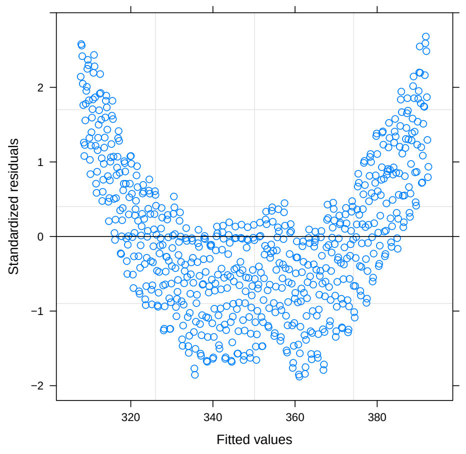
```
> anova(dataglsARMA10,dataglsARMA11,dataglsARMA12,dataglsARMA21,dataglsARMA22)

              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,] "0.111186713062342" "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=2))
> 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")
```

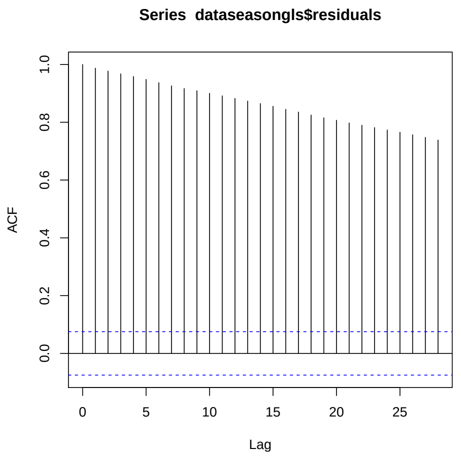


(a) residuals

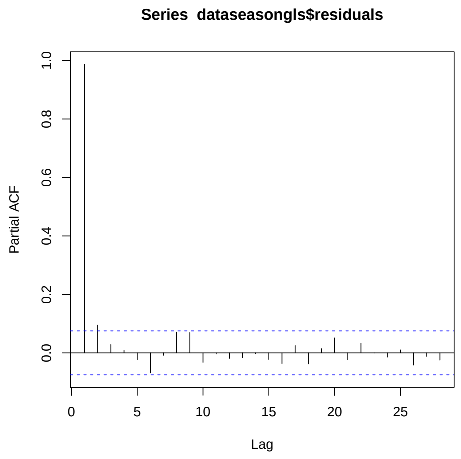


(b) seasadj.

Figure 11: Residuals fitted vs. observed in (a) LM and (b) GLS



(a) ACF



(b) PACF

Figure 12: Correlogram of seasonality-including model

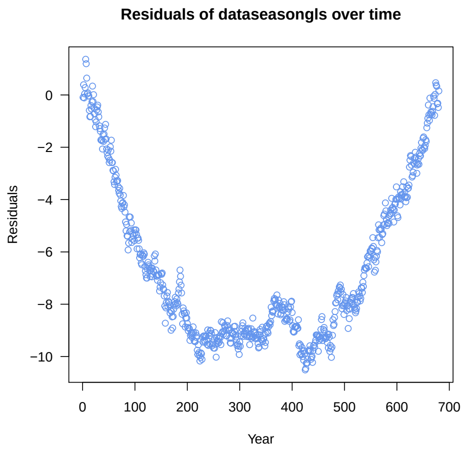


Figure 13: Residuals of seasonality-including model

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

```
> 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))
```

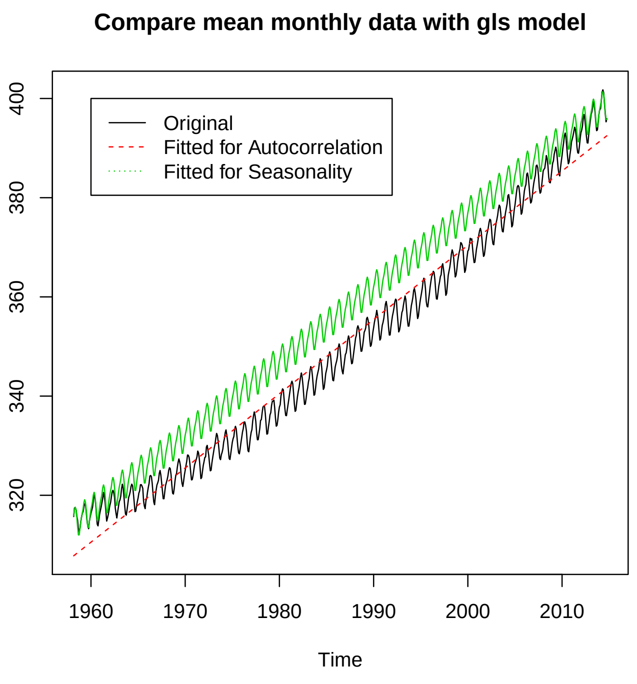


Figure 14: Comparison of seasonality-including model with only autocorrelation-including model

In the figure 14 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

> #(anova(harmonizedgls,harmonizedARMAgls))
```

	Model	df	AIC	BIC	logLik	Test	L.Ratio	p-value
harmonizedgls	1	17	396.86	473.36	-181.43			
harmonizedARMAgls	2	20	360.41	450.40	-160.20	1 vs 2	42.45	0.00

```
> 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))
```

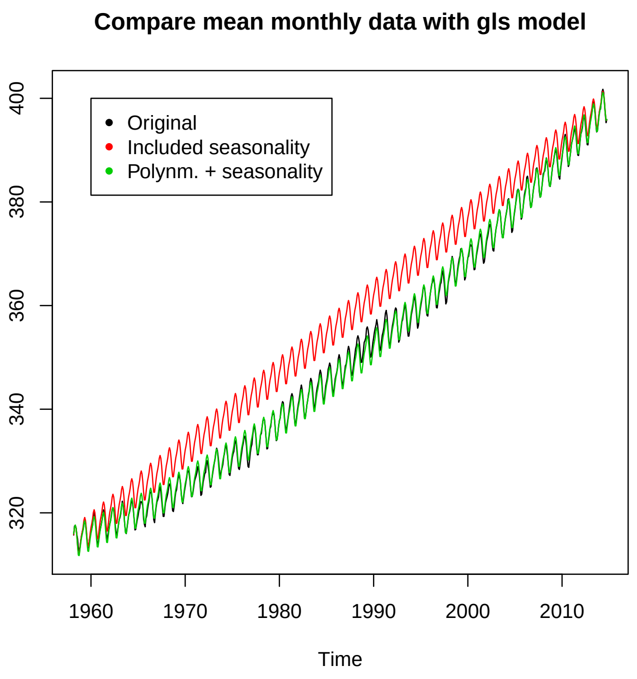
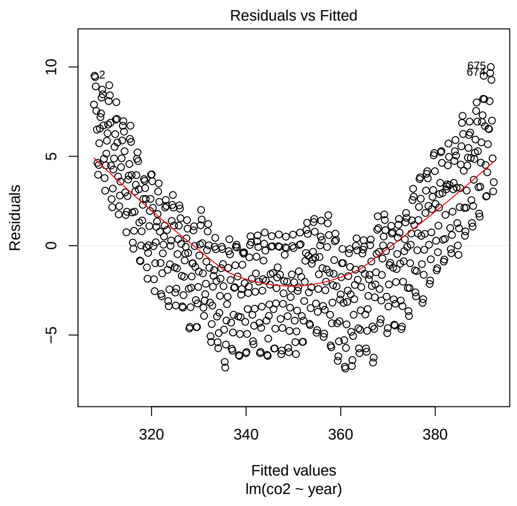


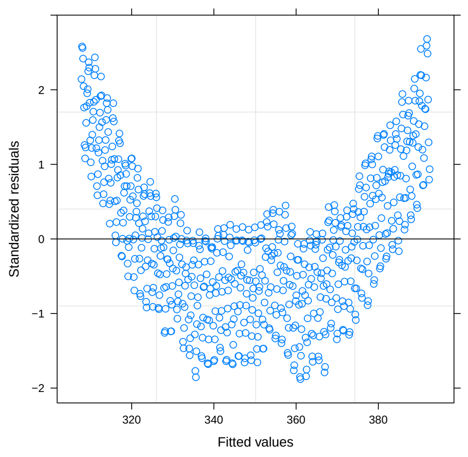
Figure 15: Comparison of season and quadr. term included model

In Figure 15 we were plotting our different approaches with the best AIC together and can see a significant improvement in the fit of the harmonized gls() which was including a quadratic term and the seasonality.

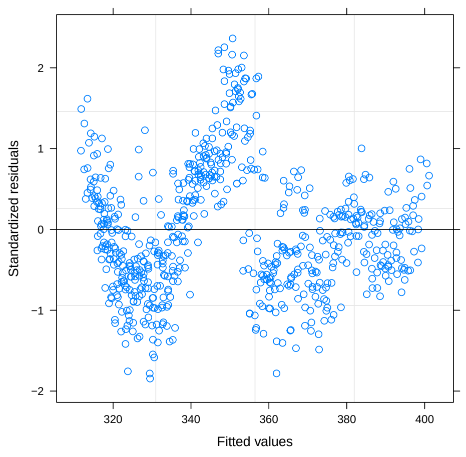
```
> diagnostics(harmonizedARMAgls)

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

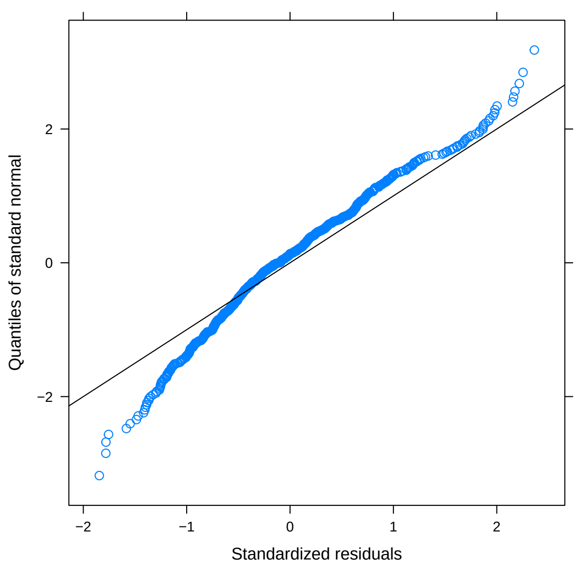


Figure 17: look at norm. distribution

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
```

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

sigma$^2 estimated as 0.08927 : log likelihood = -149.46 AIC = 312.92 AICc = 313.09 BIC = 344.44$

```
> 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))
```

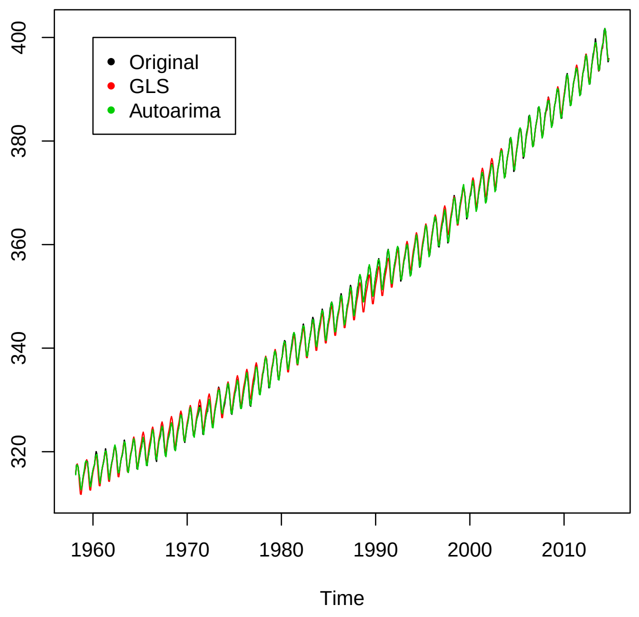
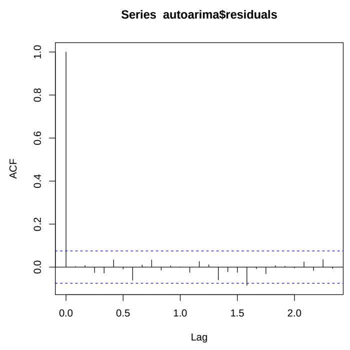


Figure 18: Fitted ARIMA values on time series

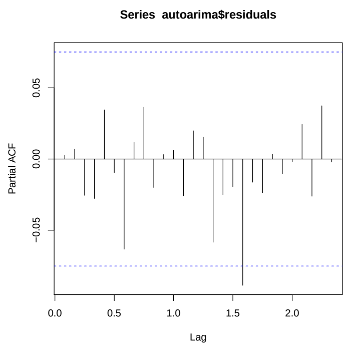
The comparison plot /reffittedarima shows that the bestmodel from gls() and the automatically adjusted arima model are barely distinguishable from the original dataset. However, the arima gives slightly better AIC than the best gls.

```
> AIC(autoarima)
```

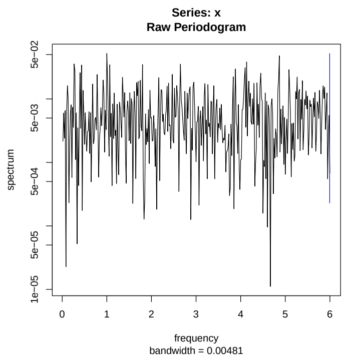
```
[1] 312.9221
```



(a) ACF



(b) PACF



(c) quadrARMA

Figure 19: Correlogram for ARIMA

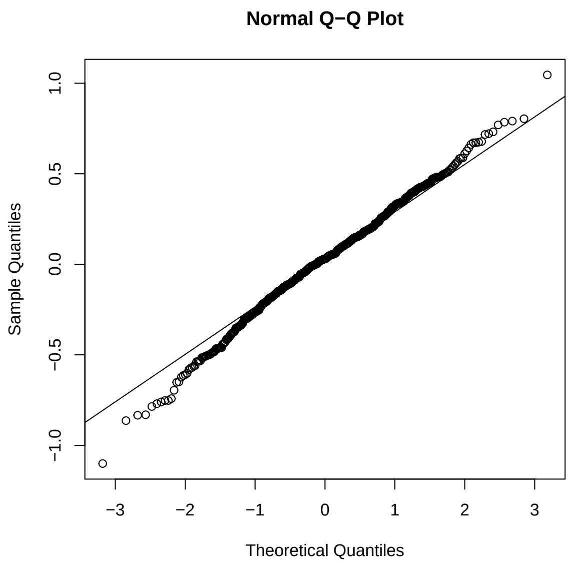


Figure 20: QQ Plot for ARIMA

```
> 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:

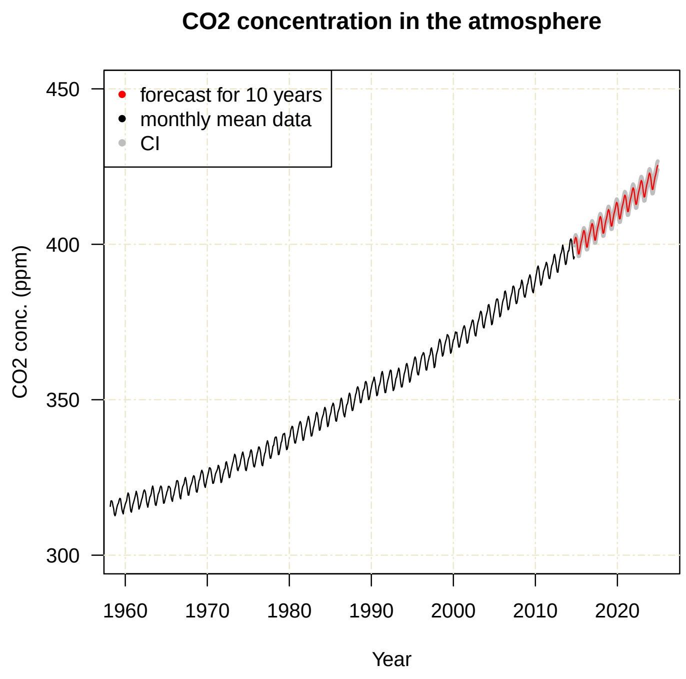
1. predict()
2. Holt Winters hw()
3. Arima forecasts forecast.Arima()

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"))
```

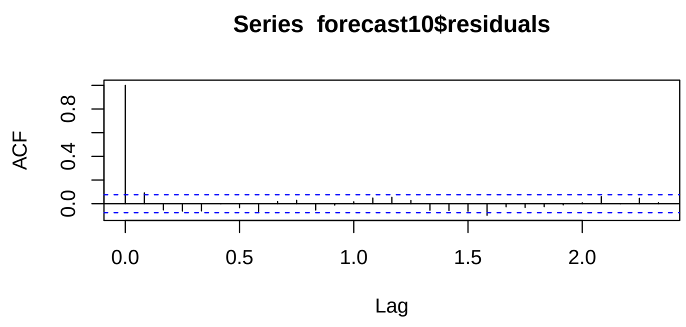


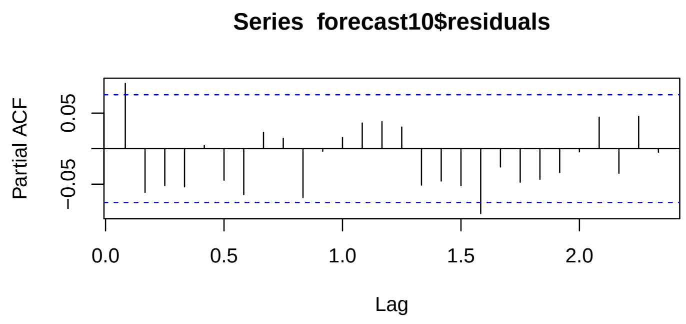
5.2 Holtwinters forecast function

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

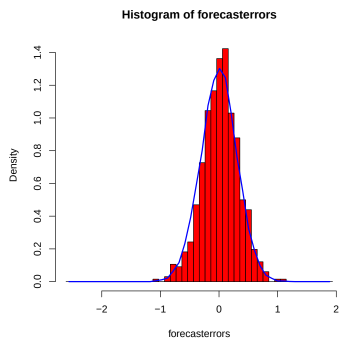
Now we try to forecast with the autoarima function as our bestmodel, which would save alot of time.

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

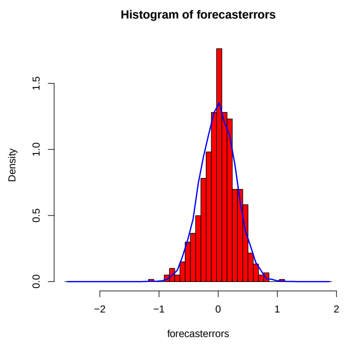
5.4 Comparison of HW and ARIMA

Now we can compare which forecast function complies better.

First thing on our list is, that we need to check for error distribution with the plotForecastErrors function from above:



(a) hw()



(b) arima

Figure 21: Forcast error distribution

The histogram of the time series in figure 21 shows that the forecast errors are roughly normally distributed and the mean seems to be close to zero for both functions.

Run diagnostics.

	normality	stat.res	stat.res.alt	autocorr	indep
1	0.357443	0.01	stationary	1.80314031521739	0.0174092

Table 3: Diagnostics hw-forecast

	normality	stat.res	stat.res.alt	autocorr	indep
1	0.037855	0.01	stationary	1.97955655809946	0.946122

Table 4: Diagnostics arima-forecast

Through the diagnostics, we would choose the arima forecast as best forecast function. The values are independent, the autocorrelation is handled and we have stationarity. One problem is the not-normal distributed residuals, which the arima can take care of. In the hw() forecast we still have problems with the independence of the residuals. Both forecasts solved the more important problems of stationarity and autocorrelation. 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.

5.5 Seasonal Decomposition of Time Series by Loess

Forecasting using stl objects is a fourth option, which is straitforward using like hw() the original time series 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
```

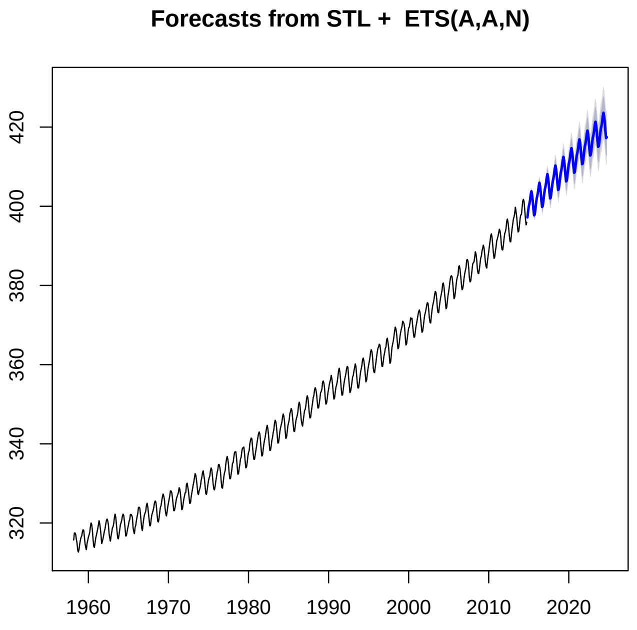
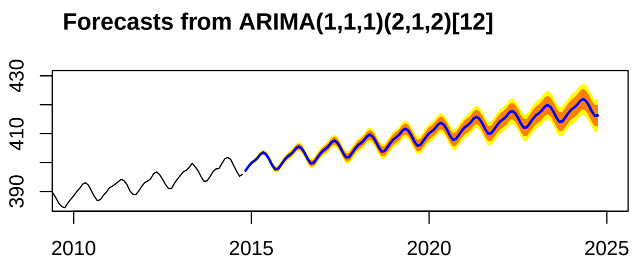


Figure 23: Forecasting using Loess

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



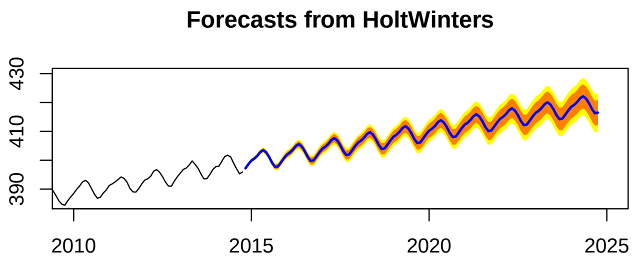


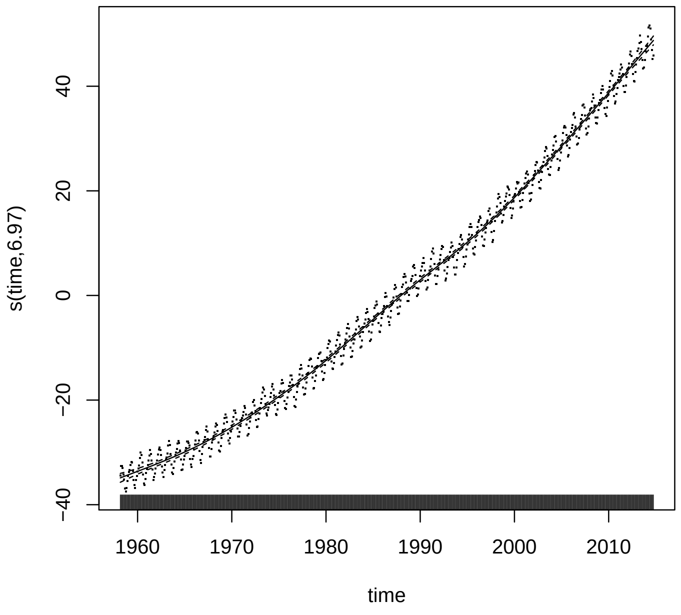
Figure 22: Comparison of different forecast functions

5.6 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 and recommend further reading on it, for example on this website: http://www.fromthebottomoftheheap.net/2014/05/09/modelling-seasonal-data-with-gam/

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

```
> 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))
> AIC(smoothed4gam)
```

[1] 926.9626

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

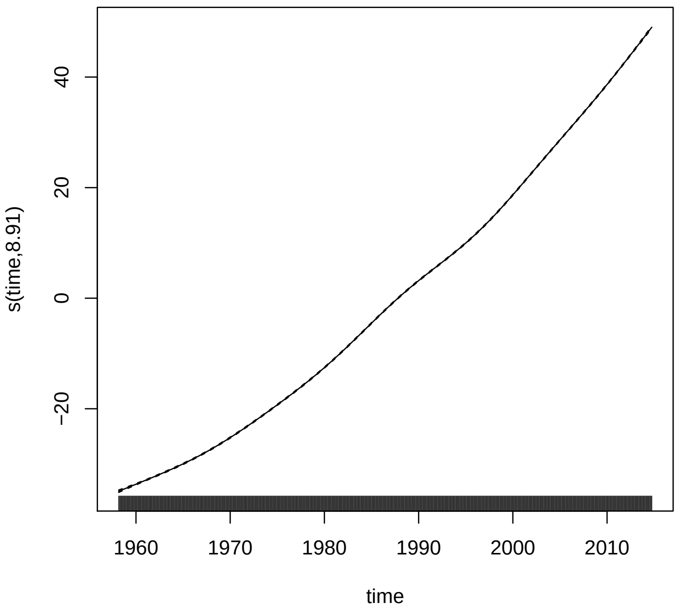


Figure 24: gam() with poly-1 smoothed term and sin-cos-wave

6 Dataset Nile (Non-seasonal)

Our example 2 contains the Measurements of the annual flow of the river Nile [m3/s] at Ashwan from 1871 to 1970. The Nile river data are included in any standard distribution of R as a time series object (i.e., a vector containing the data together with information about start/end time and sampling frequency); a detailed description of the data is given in the help file, ?Nile.

First we visualize our data typing:

```
> class(Nile)
> str(Nile)
```

Since the time-series consists of annual observations, there are no indices of a cycle/seasonality in the data. A non-seasonal time series consists of a trend and an irregular component. To estimate the trend component of a non-seasonal time series that can be described using an additive model, it is common to use a smoothing method, such as calculating the simple moving average of the time series.
The SMA() function in the "TTR" R package can be used to smooth time series data using a simple moving average.

```
> plot(Nile, main="Annual flow of the Nile", ylab="Flow [m3/s]", xlab="years")
```

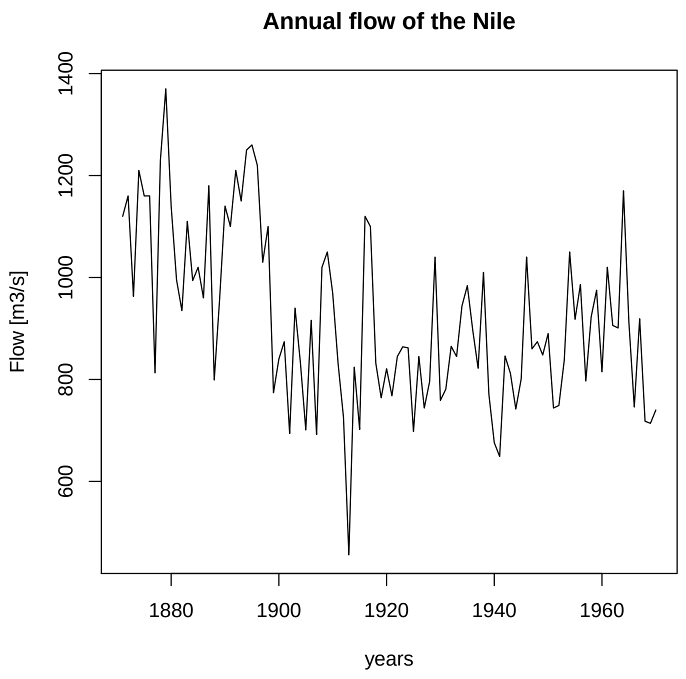


Figure 25: Annual Flow of the Nile

```
> library(TTR)
> sma = SMA(Nile)
> plot(SMA(Nile,n=20))
>
```

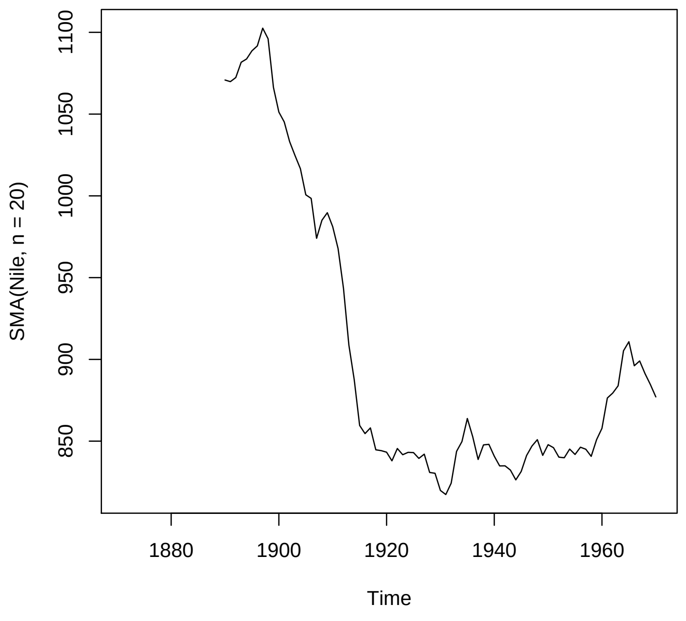


Figure 26: Trend of the Annual Flow of the Nile

After using the SMA smoothing-function, we can clearly see that there was a negative trend until 1920 which was followed by a positive trend that reached its climax around 1965.until the end of the

We can check if the data is stationary:

```
> adf.test(Nile)

        Augmented Dickey-Fuller Test

data:  Nile
Dickey-Fuller = -3.3657, Lag order = 4, p-value =
0.0642
alternative hypothesis: stationary

> kpss.test(Nile)

        KPSS Test for Level Stationarity

data:  Nile
KPSS Level = 1.3152, Truncation lag parameter = 2,
p-value = 0.01
```

The data is not stationary, we can conclude that the SMA-function was not appropriate for this time-series.

We try to fit a linear model to the data:

```
> nilelm = lm(Nile~time(Nile))
> summary(nilelm)

Call:
lm(formula = Nile ~ time(Nile))

Residuals:
    Min      1Q  Median      3Q     Max
-483.71  -98.17  -23.21  111.40  368.72

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 6132.1736  1001.7578   6.121 1.92e-08 ***
time(Nile)    -2.7143     0.5216  -5.204 1.07e-06 ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 150.6 on 98 degrees of freedom
Multiple R-squared:  0.2165,       Adjusted R-squared:  0.2085
F-statistic: 27.08 on 1 and 98 DF,  p-value: 1.072e-06

> confint(nilelm)

                  2.5 %      97.5 %
(Intercept) 4144.217893 8120.129266
time(Nile)    -3.749313   -1.679298
```

```
> par(mfrow=c(1,2))
> acf(resid(nilelm))
> pacf(resid(nilelm))
```

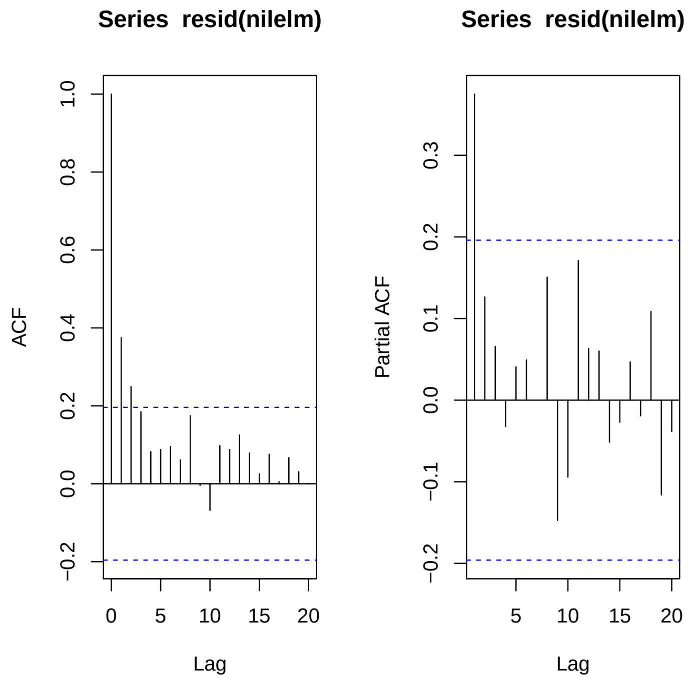


Figure 27: Autocorelations diagnostics of the Nile Time Series

We can see that the linear model is not appropriate,the correlograms indicate a positive autocorrelation in the first lags. Next we fit the gls function to provide better estimates to account for the autocorrelation in the residual series:

```
> nilegls = gls(Nile ~ time(Nile),cor= corAR1(acf(resid(nilelm))$acf[2]))
> save(nilegls, file="nilegls.RData")

> load("nilegls.RData")

> summary(nilegls)

Generalized least squares fit by REML
  Model: Nile ~ time(Nile)
  Data: NULL
       AIC      BIC    logLik
  1268.205 1278.545 -630.1026

Correlation Structure: AR(1)
 Formula: ~1
 Parameter estimate(s):
      Phi
0.4002726

Coefficients:
               Value Std.Error   t-value p-value
(Intercept) 6216.491 1518.1486  4.094784   1e-04
time(Nile)    -2.758    0.7904 -3.489523   7e-04

 Correlation:
           (Intr)
```

```
time(Nile) -1

Standardized residuals:
       Min         Q1        Med         Q3        Max
-3.1787691 -0.6428562 -0.1463738  0.7245317  2.4323739

Residual standard error: 152.3157
Degrees of freedom: 100 total; 98 residual
```

```
> confint(nilegls)

                  2.5 %      97.5 %
(Intercept) 3240.974234 9192.007340
time(Nile)    -4.307301   -1.208971
```

```
> acf(nilegls$residuals); pacf(nilegls$residuals)
```

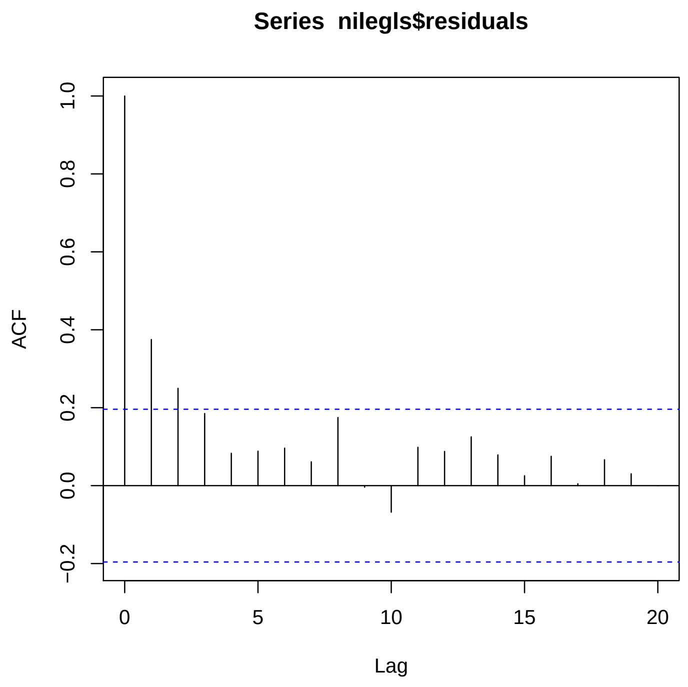


Figure 28: Autocorrelation diagnostics for the gls Model

The residuals look like white noise, next we test if they are independent:

```
> Box.test(nilegls$residuals, type="Ljung-Box")

        Box-Ljung test

data:  nilegls$residuals
X-squared = 14.496, df = 1, p-value = 0.0001405
```

The null hypothesis is rejected, the residuals are not independent (p<0.05).
We find a significant autocorrelation at the first lags only, that means that the autocorrelation decreases with the time. We see no indices of seasonality. The ACF does not show any significant autocorrelation.
INSERT ACF PICTURE

```
> plot(nilegls)
```

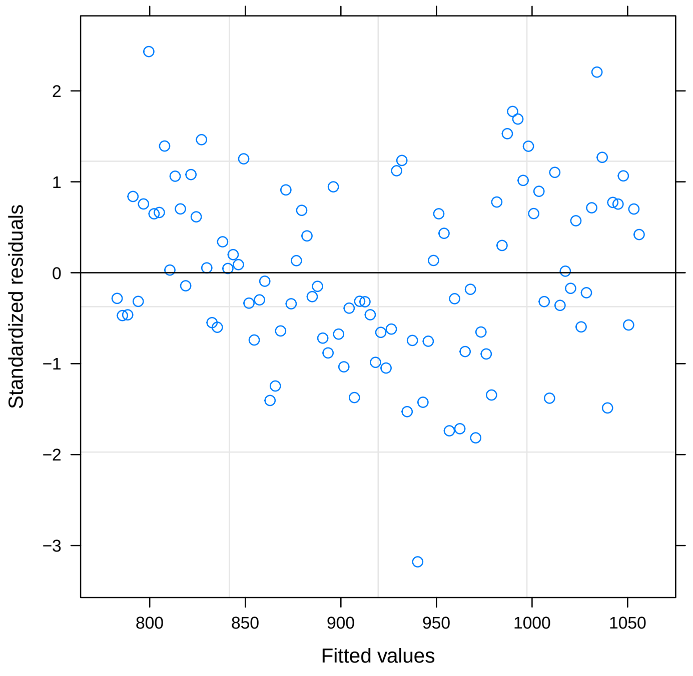


Figure 29: Residuals of the gls Model

6.1 Holt Winters exponential smoothing

```
> library(forecast)
> hwn = HoltWinters(Nile,alpha = 0.2, beta = FALSE, gamma = FALSE,
                    start.periods =c(1120,740) )
> plot(hwn)
>
```

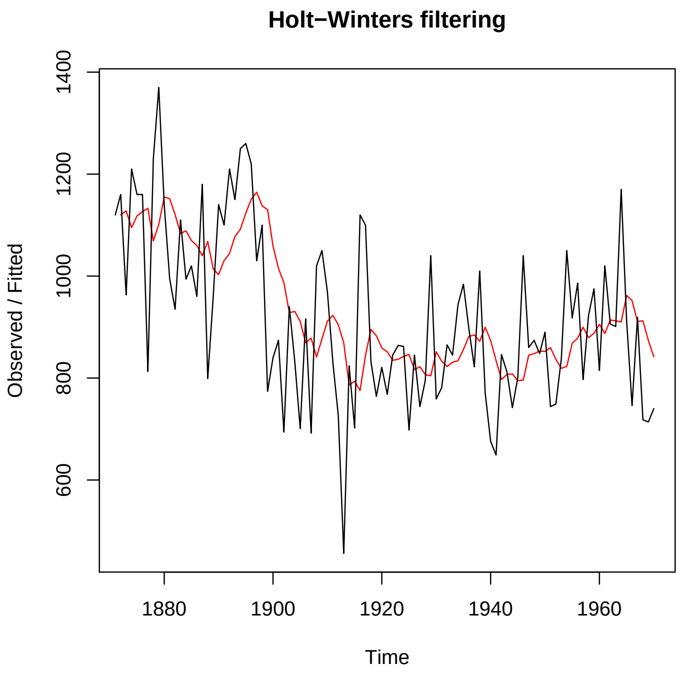


Figure 30: Holt Winters

```
> forecast_nile = forecast.HoltWinters(hwn, h=5)
> plot.forecast(forecast_nile, shadecols = "oldstyle")
>
```

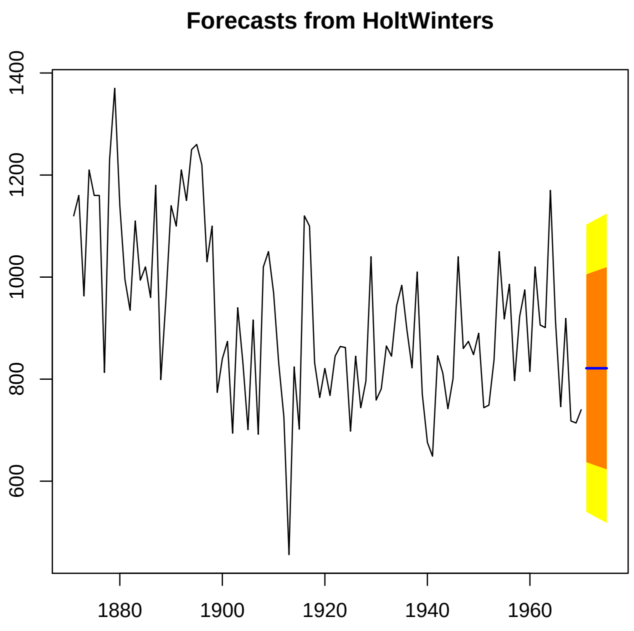


Figure 31: Holt Winter Forecasting

The forecasts are shown as a blue line, with the 80

```
> acf(forecast_nile$residuals, lag.max = 20)
> pacf(forecast_nile$residuals, lag.max = 20)
> Box.test(forecast_nile$residuals, lag=20, type="Ljung-Box")

        Box-Ljung test

data:  forecast_nile$residuals
X-squared = 15.8772, df = 20, p-value = 0.7242

>
```

The p-value of the Ljung-Box test (0.72) indicates that there is little evidence of non-zero autocorrelations in the in-sample forecast errors at lags 1-20. The residuals are now independent.

We should check if the forecast errors have constant variance over time, and are normally distributed with mean zero. We can do this by plotting the forecast errors and a histogram of the distribution of these errors.

```
> par(mfrow=c(1,2))
> plot.ts(forecast_nile$residuals);abline(h=0)
> plotForecastErrors(forecast_nile$residuals)
> par(mfrow=c(1,1))
```

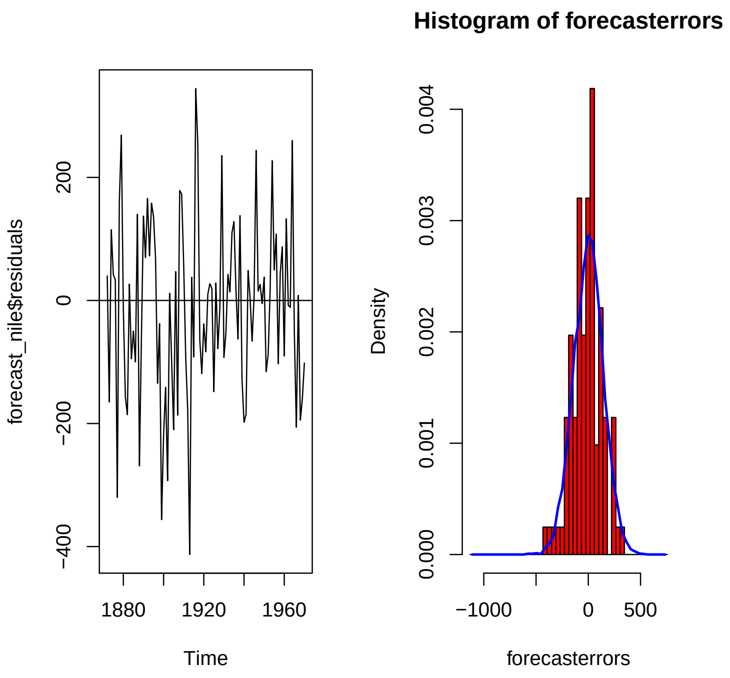


Figure 32: Forecast Errors

The forecast error plot shows roughly constant variance over time. The histogram of forecast errors emphasizes normally distributed residuals with mean zero and constant variance.

6.2 ARIMA

While exponential smoothing methods do not make any assumptions about correlations between successive values of the time series, in some cases you can make a better predictive model by taking correlations in the data into account. Autoregressive Integrated Moving Average (ARIMA) models include an explicit statistical model for the irregular component of a time series, that allows for non-zero autocorrelations in the irregular component.

```
> nile_arima = auto.arima(Nile)
> forecast_nile2 = forecast.Arima(nile_arima, h=10)
> plot.forecast(forecast_nile2, shadecols = "oldstyle")
>
```

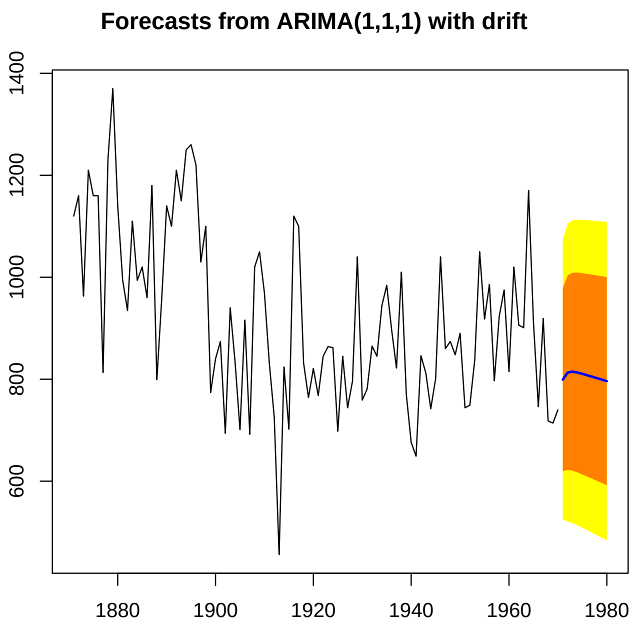


Figure 33: Arima forecasting

Check for autocorrelations:

```
> acf(forecast_nile2$residuals, lag.max = 20)
> Box.test(forecast_nile2$residuals, lag=20, type="Ljung-Box")

        Box-Ljung test

data:  forecast_nile2$residuals
X-squared = 12.1156, df = 20, p-value = 0.912
```

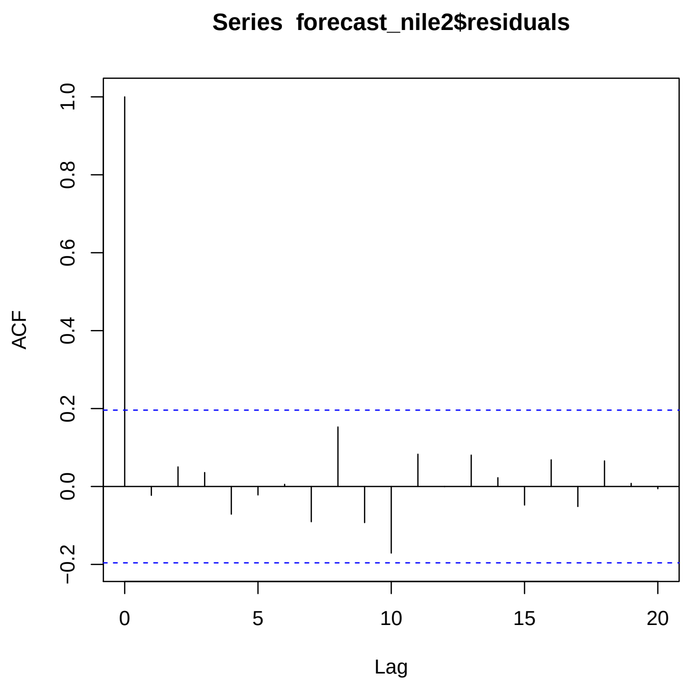


Figure 34: acf of the residuals

There is little evidence of non-zero autocorrelations in the in-sample forecast errors at lags 1-20. The residuals are now independent and the autocorrelation plots look nice.

```
> par(mfrow=c(1,2))
> plot.ts(forecast_nile2$residuals); abline(h=0)
> plotForecastErrors(forecast_nile2$residuals)
```

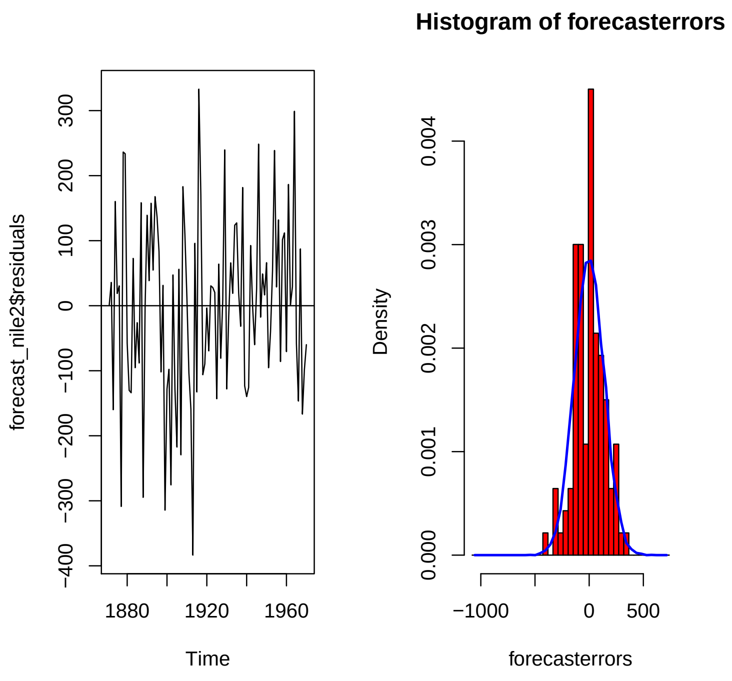


Figure 35: Residuals and Errors

We run a Durbin Watson Test to check for autocorrelations:

```
> library(car)
> dwt(as.vector(nilegls$residuals))

[1] 1.247133
```

The Durbin Watson Test of the residuals shows that there is no autocorrelation and the residuals are independent.

6.3 Structural time series models

The function StructTS fits a model by maximum likelihood.

```
> fit <- StructTS(Nile, type = "level")
> fit

Call:
StructTS(x = Nile, type = "level")

Variances:
  level  epsilon
   1469    15099
```

The Maximum Likelihood Estimates (MLEs) of the level- and observation error variances, 1469 and 15099, respectively, are included in the output as the coefficients of fit.

```
> par(mfrow = c(3, 1))
> plot(Nile)
> # local level model (contemporaneous smoothing)
> lines(fitted(fit), lty = "dashed", col=4)
> # fixed-interval smoothing
> lines(tsSmooth(fit), lty = "dotted", col = 6)
> legend("bottomright" ,col = c(4,6),c("filtered","smoothed"),
        lty=c("dashed", "dotted"), bty="n", cex=0.8)
> plot(residuals(fit)); abline(h = 0, lty = 3)
> # local trend model (constant trend fitted)
> fit2 <- StructTS(Nile, type = "trend")
> pred <- predict(fit, n.ahead = 30)
> # with 95% confidence interval
> ts.plot(Nile, pred$pred,
         pred$pred + 1.96*pred$se, pred$pred -1.96*pred$se,
         col=c(1,2,1))
> par(mfrow=c(1,1))
>
```

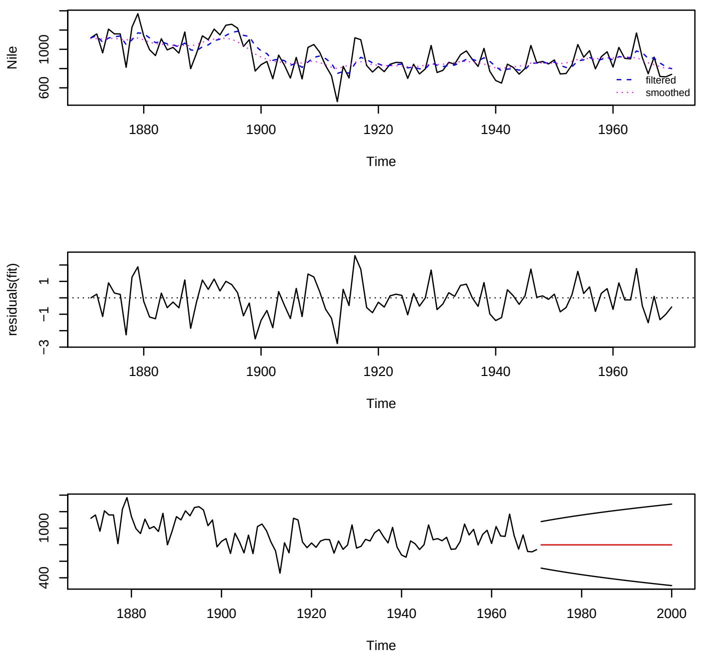


Figure 36: Fitting a Structural Model

The function (tsdiag) can be called on an object of class StructTS to obtain diagnostic plots based on the standardized one-step-ahead forecast errors.

```
> tsdiag(fit)
> tsdiag(fit2)
>
```

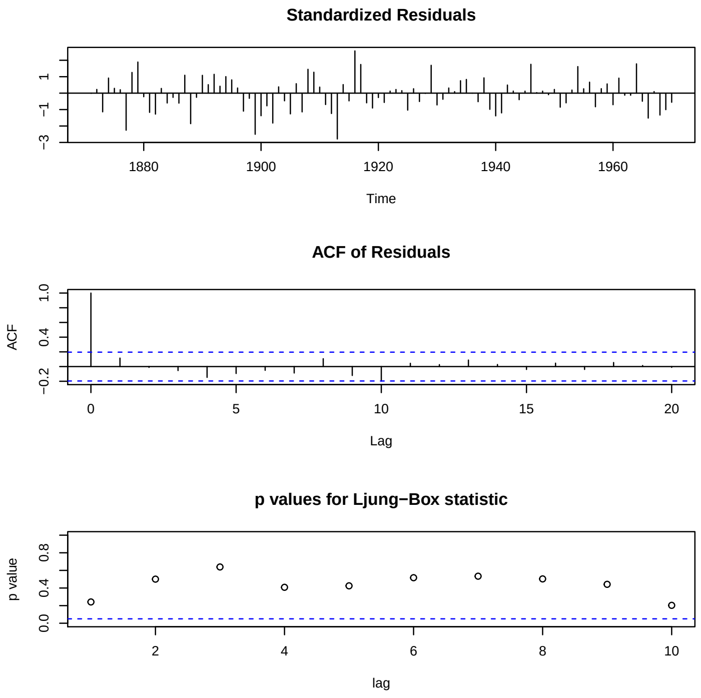


Figure 37: Models diagnostics

```
> library(forecast)
> plot(forecast(fit2, level = c(50,90), h = 10), xlim = c(1950, 1980),
      shadecols = "oldstyle")
```

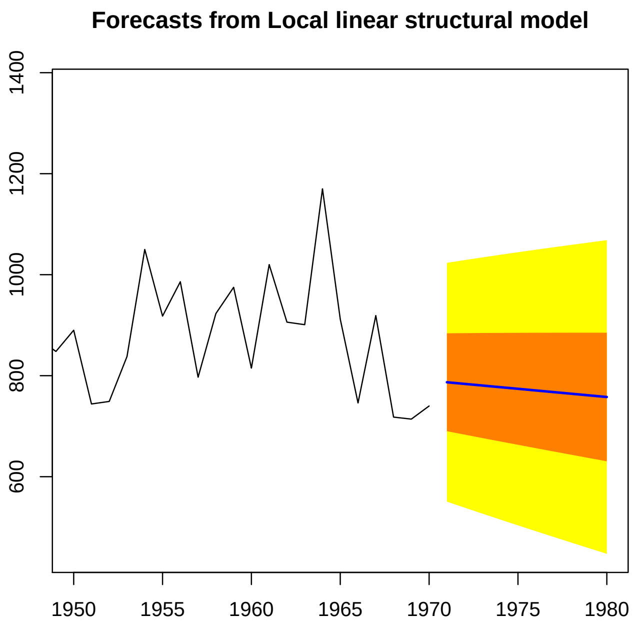


Figure 38: Forecasted values

Forecasts for structural time series, as objects of class StructTS, can be obtained by either the method function (predict) or (forecast) in package forecast (Hyndman 2011; Hyndman and Khandakar 2008). This package also provides a convenient plot method function for the resulting object of class forecast. The plot of forcasted values, obtained with the code below, shows the forecasted Nile river data until 1980, togeher with 50% and 90% probability intervals.

6.4 The package dlm and the Kalman filter

A polynomial DLM (a local level model is a polynomial DLM of order 1, a local linear trend is a polynomial DLM of order 2), is easily defined in dlm through the function (dlmModPoly).The function simulates one draw from the posterior distribution of the state vectors.

```
> library(dlm)
> nile_mod <- dlmModPoly(1, dV = 15099.8, dW = 1468.4)
> #values taken from the coefficients of fit
> nile_filt <- dlmFilter(Nile, nile_mod)
> nile_smooth <- dlmSmooth(nile_filt) # estimated "true" level
> plot(cbind(Nile, nile_smooth$s[-1]), plot.type = "s",
      col = c("black", "red"), ylab = "Level",
      main = "Nile river", lwd = c(2, 2))
> for (i in 1:10) # 10 simulated "true" levels
     lines(dlmBSample(nile_filt[-1]), lty=2, col= "mediumblue"   )
```

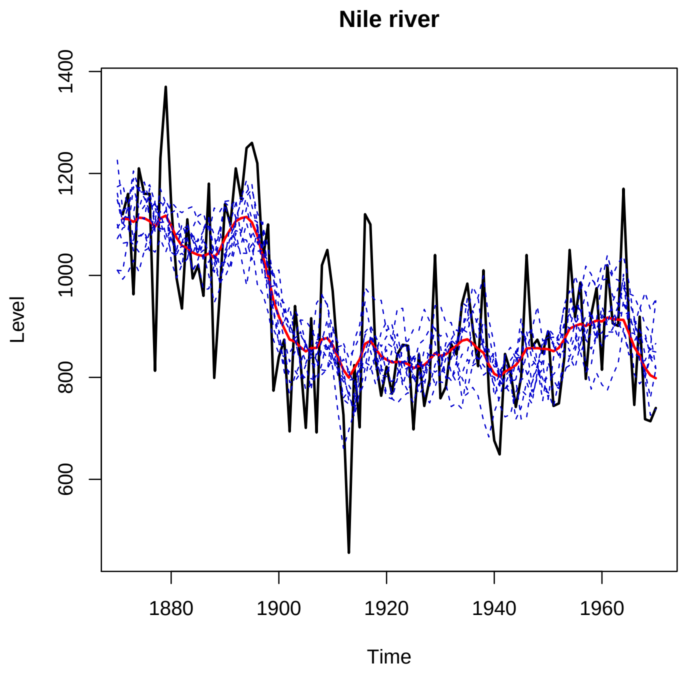


Figure 39: dlm Model

6.5 GAM (Generalized Additive Model)

```
> library(mgcv)
> niletime = time(Nile)
> gamnile = gam(as.numeric(Nile) ~ s(niletime),cor=corARMA(p=2, q=0))
> plot.gam(gamnile,residuals = T, se=T, main="GAM-Model", shade = T,
          seWithMean = T, all.terms = T , cex=2 )
```

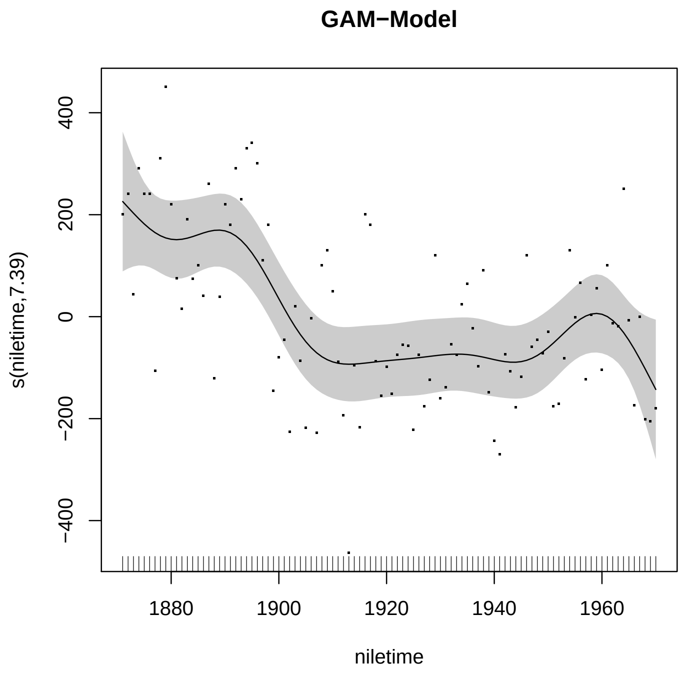


Figure 40: GAM Model

We can now check which model has the best AIC:

```
> AIC(nilelm)

[1] 1290.629

> AIC(nilegls)

[1] 1268.205

> AIC(nile_arima)

[1] 1254.913

> AIC(gamnile)

[1] 1274.499
```

The nile_arima Model seems to be the best one.

7 Compabitily with Linux and Windows

We as a group experienced a lot of compatibility problems between Linux and Windows users when working on the same sweave document. If you also deal with line break issues or other unexplainable errors, one possible option to fix this is by transforming your Linux sweave document with the terminal command “unix2dos yourfilename.Rnw”.

8 Conclusion and helpful links

Reminding of the Socratic paradox "I know one thing: that I know nothing" , we recommend further reading on the topic of time series. As we were noticing while dealing with the different correlation structures, our tutorial is considering the time series which are temporally autocorrelated. There are other covariance structures needed if you have spatially autocorrelated time series.

Helpful links for further reading:

Time-Series Analysis in R:
http://cran.r-project.org/web/views/TimeSeries.html

Additional Example-Datasets:
http://www.comp-engine.org/timeseries/browse-data-by-category
https://datamarket.com/data/list/?q=provider:tsdl

Introductory Time Series with R: Book by Cowpertwait and Metcalfe (Springer 2009)

9 Acknowledgements

We like to thank the whole TeX and R community for providing us with an incredible amount of open source software. Without You this tutorial would have never been created!