Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Applied Time Series Analysis

SS 2020

May 4, 2020

Dr. Marcel Dettling

Institute for Data Analysis and Process Design

Zurich University of Applied Sciences

CH-8401 Winterthur

Table of Contents

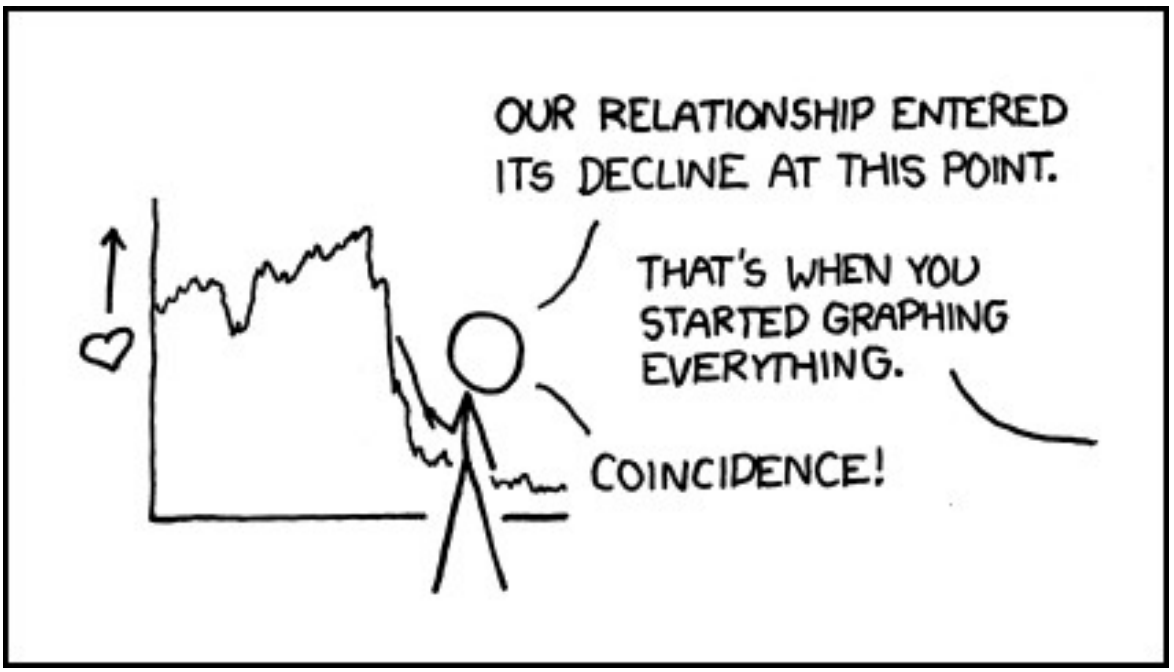
1 Introduction

1.1 Purpose

Time series data, i.e. records which are measured sequentially over time, are extremely common. They arise in virtually every application field, such as e.g.:

- **Business**
  Sales figures, production numbers, customer frequencies, ...
- **Economics**
  Stock prices, exchange rates, interest rates, ...
- **Official Statistics**
  Census data, personal expenditures, road casualties, ...
- **Natural Sciences**
  Population sizes, sunspot activity, chemical process data, ...
- **Environmetrics**
  Precipitation, temperature or pollution recordings, ...

In contrast to basic data analysis where the assumption of identically and independently distributed data is key, time series are serially correlated. The purpose of time series analysis is to visualize and understand these dependencies in past data, and to exploit them for forecasting future values. While some simple descriptive techniques do often considerably enhance the understanding of the data, a full analysis usually involves modeling the stochastic mechanism that is assumed to be the generator of the observed time series.



Once a good model is found and fitted to data, the analyst can use that model to forecast future values and produce prediction intervals, or he can generate simulations, for example to guide planning decisions. Moreover, fitted models are used as a basis for statistical tests: they allow determining whether fluctuations in monthly sales provide evidence of some underlying change, or whether they are still within the range of usual random variation.

The dominant main features of many time series are *trend* and *seasonal variation*. These can either be modeled deterministically by mathematical functions of time, or are estimated using non-parametric smoothing approaches. Yet another key feature of most time series is that adjacent observations tend to be correlated, i.e. serially dependent. Much of the methodology in time series analysis is aimed at explaining this correlation using appropriate statistical models.

While the theory on mathematically oriented time series analysis is vast and may be studied without necessarily fitting any models to data, the focus of our course will be applied and directed towards data analysis. We study some basic properties of time series processes and models, but mostly focus on how to visualize and describe time series data, on how to fit models to data correctly, on how to generate forecasts, and on how to adequately draw conclusions from the output that was produced.

1.2 Examples

1.2.1 Air Passenger Bookings

The numbers of international passenger bookings (in thousands) per month on an airline (*PanAm*) in the United States were obtained from the Federal Aviation Administration for the period 1949-1960. The company used the data to predict future demand before ordering new aircraft and training aircrew. The data are available as a time series in **R**. Here, we here show how to access them, and how to first gain an impression.

```
> data(AirPassengers)
> AirPassengers
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949 112 118 132 129 121 135 148 148 136 119 104 118
1950 115 126 141 135 125 149 170 170 158 133 114 140
1951 145 150 178 163 172 178 199 199 184 162 146 166
1952 171 180 193 181 183 218 230 242 209 191 172 194
1953 196 196 236 235 229 243 264 272 237 211 180 201
1954 204 188 235 227 234 264 302 293 259 229 203 229
1955 242 233 267 269 270 315 364 347 312 274 237 278
1956 284 277 317 313 318 374 413 405 355 306 271 306
1957 315 301 356 348 355 422 465 467 404 347 305 336
1958 340 318 362 348 363 435 491 505 404 359 310 337
1959 360 342 406 396 420 472 548 559 463 407 362 405
1960 417 391 419 461 472 535 622 606 508 461 390 432
```

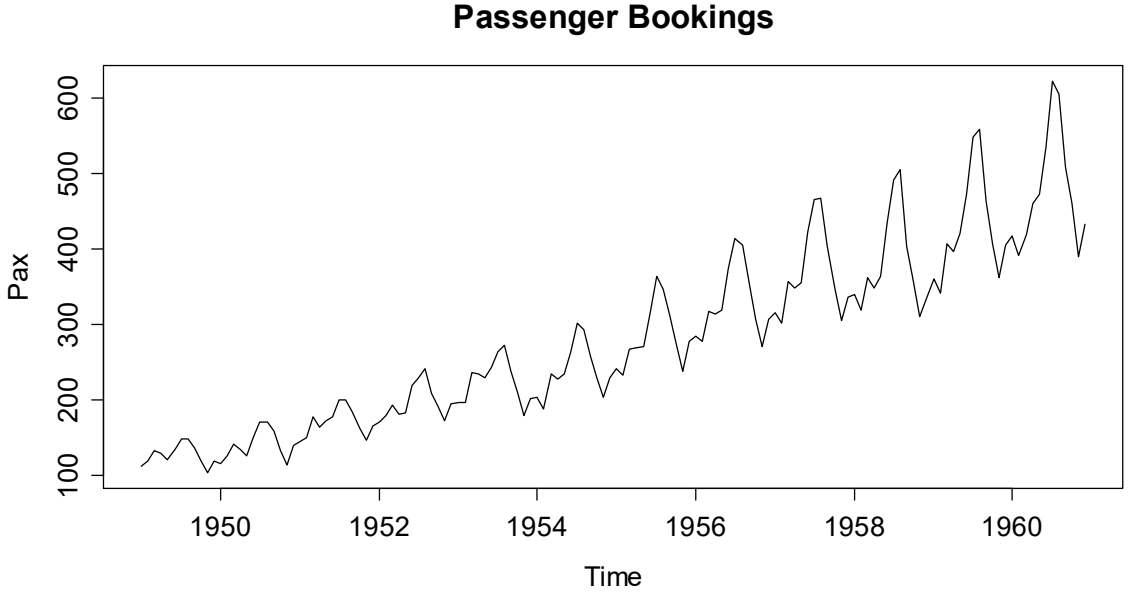
Some further information about this dataset can be obtained by typing `?AirPassengers` in **R**. The data are stored in an **R**-*object* of *class* `ts`, which is the specific class for time series data. However, for further details on how time series are handled in **R**, we refer to section 3.

One of the most important steps in time series analysis is to visualize the data, i.e. create a time series plot, where the air passenger bookings are plotted versus the time of booking. For a time series object, this can be done very simply in **R**, using the generic plot function:

```
> plot(AirPassengers, ylab="Pax", main="Passenger Bookings")
```

The result is displayed on the next page. There are a number of features in the plot which are common to many time series. For example, it is apparent that the number of passengers travelling on the airline is increasing with time. In general, a systematic change in the mean level of a time series that does not appear to be periodic is known as a *trend*. The simplest model for a trend is a linear increase or decrease, an often adequate approximation. We will discuss how to estimate trends, and how to decompose time series into trend and other components in section 4.3.

The data also show a repeating pattern within each year, i.e. in summer, there are always more passengers than in winter. This is known as a *seasonal effect*, or *seasonality*. Please note that this term is applied more generally to any repeating pattern over a fixed period, such as for example restaurant bookings on different days of week.



We can naturally attribute the increasing trend of the series to causes such as rising prosperity, greater availability of aircraft, cheaper flights and increasing population. The seasonal variation coincides strongly with vacation periods. For this reason, we here consider both trend and seasonal variation as deterministic components. As mentioned before, section 4.3 discusses visualization and estimation of these

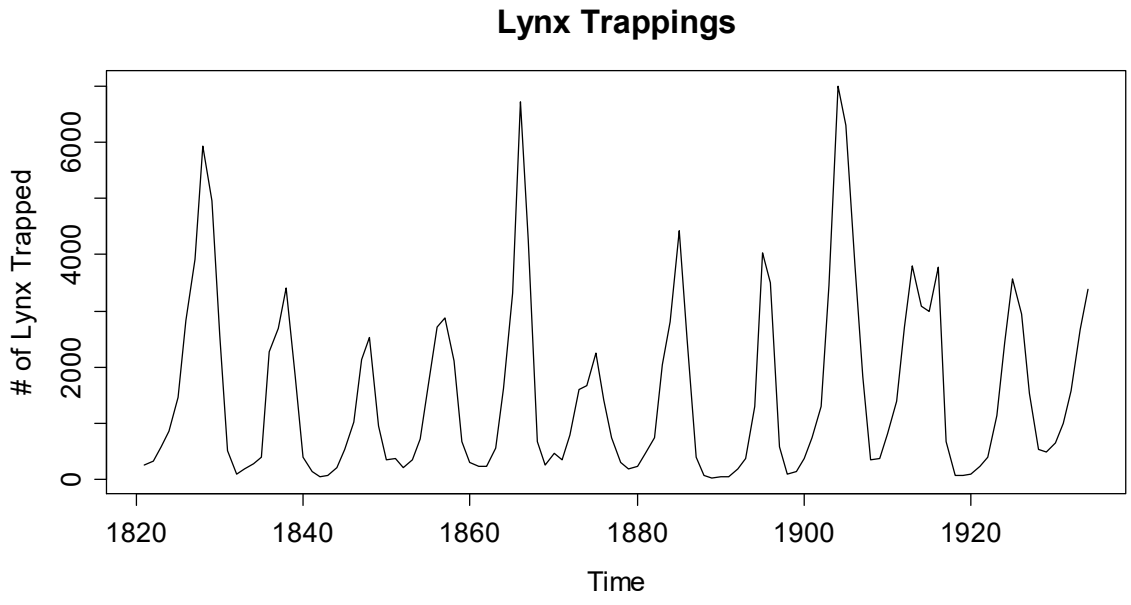
components, while in section 7, time series regression models will be specified to allow for underlying causes like these, and finally section 8 discusses exploiting these for predictive purposes.

1.2.2 Lynx Trappings

The next series which we consider here is the annual number of lynx trappings for the years 1821-1934 in the Mackenzie River District in Canada. We again load the data and visualize them using a time series plot:

```
> data(lynx)
> plot(lynx, ylab="# of Lynx Trapped", main="Lynx Trappings")
```

The plot on the next page shows that the number of trapped lynx reaches high and low values every about 10 years, and some even larger figure every about 40 years. While biologists often approach such data with predator-prey-models, we here focus on the analysis of the time signal only. This suggests that the prominent periodicity is to be interpreted as random, but not deterministic.



This leads us to the heart of time series analysis: while understanding and modeling trend and seasonal variation is a very important aspect, much of the time series methodology is aimed at *stationary series*, i.e. data which do not show deterministic, but only random (cyclic) variation.



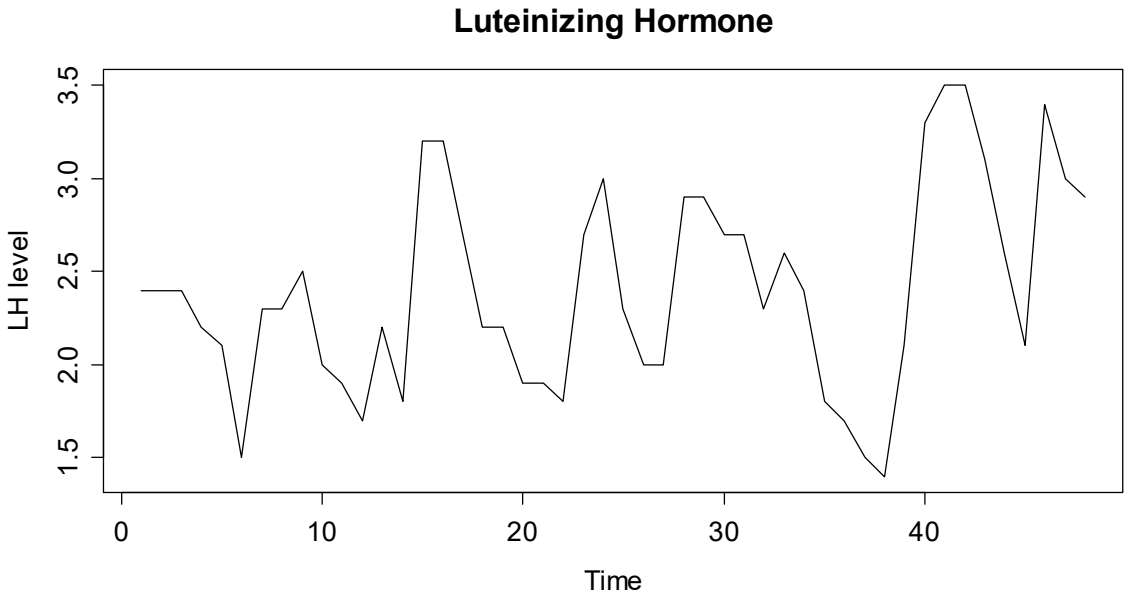
1.2.3 Luteinizing Hormone Measurements

One of the key features of the above lynx trappings series is that the observations apparently do not stem from independent and identically distributed (*iid*) random variables, but there is some serial correlation. If the previous value was high (or low, respectively), the next one is likely to be similar to the previous one. To explore, model and exploit such dependence lies at the root of time series analysis. We here show another series, where 48 luteinizing hormone levels were recorded from blood samples that were taken at 10 minute intervals from a human female. This hormone, also called *lutropin*, triggers ovulation.

```
> data(lh)
> lh
Time Series:
Start = 1; End = 48; Frequency = 1
 [1] 2.4 2.4 2.4 2.2 2.1 1.5 2.3 2.3 2.5 2.0 1.9 1.7 2.2 1.8
[15] 3.2 3.2 2.7 2.2 2.2 1.9 1.9 1.8 2.7 3.0 2.3 2.0 2.0 2.9
[29] 2.9 2.7 2.7 2.3 2.6 2.4 1.8 1.7 1.5 1.4 2.1 3.3 3.5 3.5
[43] 3.1 2.6 2.1 3.4 3.0 2.9
```

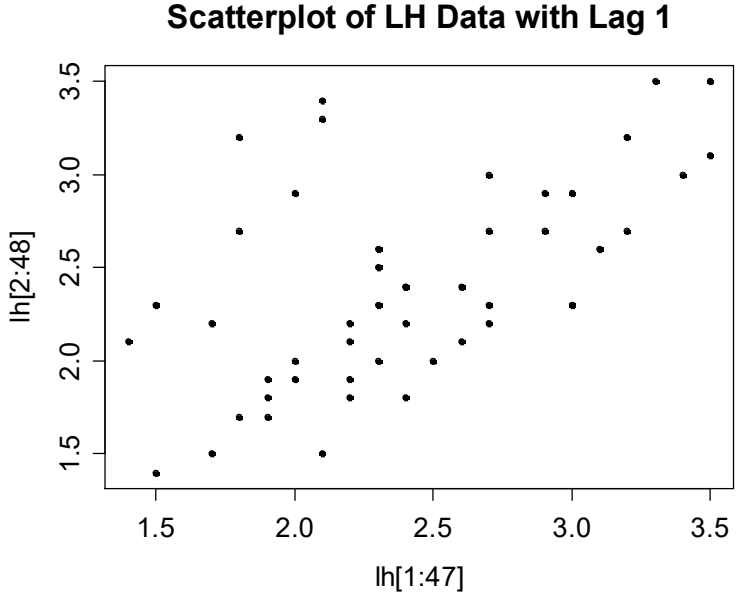
Again, the data themselves are of course needed to perform analyses, but provide little overview. We can improve this by generating a time series plot:

```
> plot(lh, ylab="LH level", main="Luteinizing Hormone")
```



For this series, given the way the measurements were made (i.e. 10 minute intervals), we can almost certainly exclude any deterministic seasonal pattern. But is there any stochastic cyclic behavior? This question is more difficult to answer. Normally, one resorts to the simpler question of analyzing the correlation of subsequent records, called *autocorrelations*. The autocorrelation for lag 1 can be visualized by producing a scatterplot of adjacent observations:

```
> plot(lh[1:47], lh[2:48], pch=20)
> title("Scatterplot of LH Data with Lag 1")
```



Besides the (non-standard) observation that there seems to be an inhomogeneity, i.e. two distinct groups of data points, it is apparent that there is a positive correlation between successive measurements. This manifests itself with the clearly visible fact that if the previous observation was above or below the mean, the next one is more likely to be on the same side. We can even compute the value of the Pearson correlation coefficient:

```
> cor(lh[1:47], lh[2:48])
[1] 0.5807322
```

Its value of 0.58 is an estimate for the so-called *autocorrelation coefficient at lag 1*. As we will see in section 4.4, the idea of considering lagged scatterplots and computing Pearson correlation coefficients serves as a good proxy for a mathematically more sound method. We also note that despite the positive correlation of +0.58, the series seems to always have the possibility of "reverting to the other side of the mean", a property which is common to *stationary series* – an issue that will be discussed in section 2.2.

1.2.4 Swiss Market Index

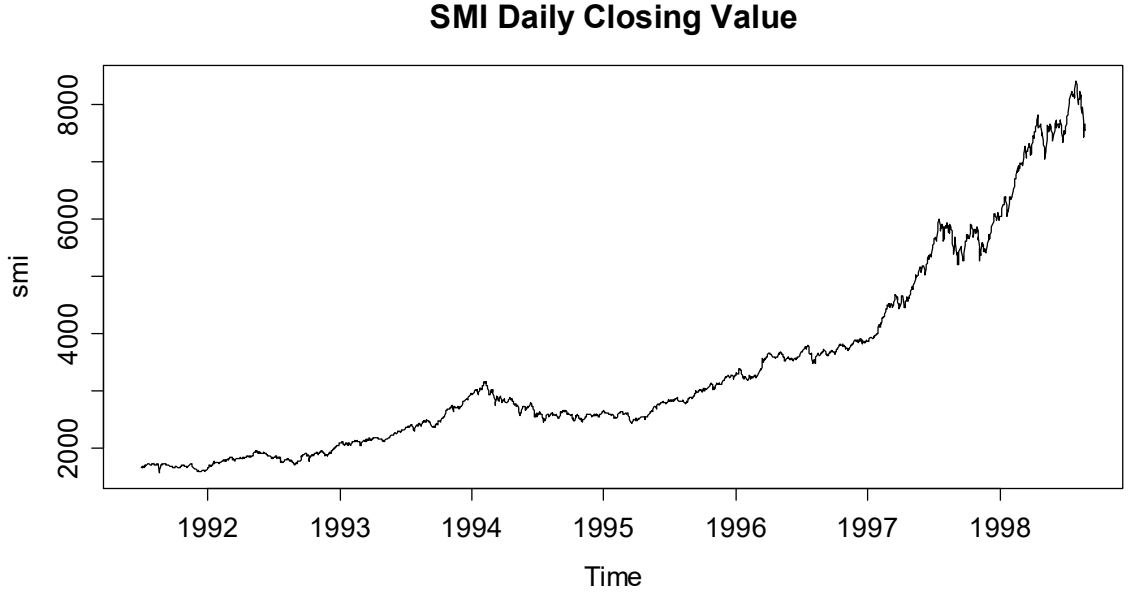
The *SMI* is the blue chip index of the Swiss stock market. It summarizes the value of the shares of the 20 most important companies, and currently contains nearly 90% of the total market capitalization. It was introduced on July 1, 1988 at a basis level of 1500.00 points. Daily closing data for 1860 consecutive trading days from 1991-1998 are available in **R**. We observe a more than 4-fold increase during that period. As a side note, the value in February 2016 is around 7'800 points, indicating a sidewards movement over the latest 15 years.

```
> data(EuStockMarkets)
> EuStockMarkets
Time Series:
Start = c(1991, 130)
End = c(1998, 169)
Frequency = 260
             DAX    SMI    CAC   FTSE
1991.496 1628.75 1678.1 1772.8 2443.6
1991.500 1613.63 1688.5 1750.5 2460.2
1991.504 1606.51 1678.6 1718.0 2448.2
1991.508 1621.04 1684.1 1708.1 2470.4
1991.512 1618.16 1686.6 1723.1 2484.7
1991.515 1610.61 1671.6 1714.3 2466.8
```

As we can see, `EuStockMarkets` is a multiple time series object, which also contains data from the German DAX, the French CAC and UK's FTSE. We will focus on the SMI and thus extract and plot the series:

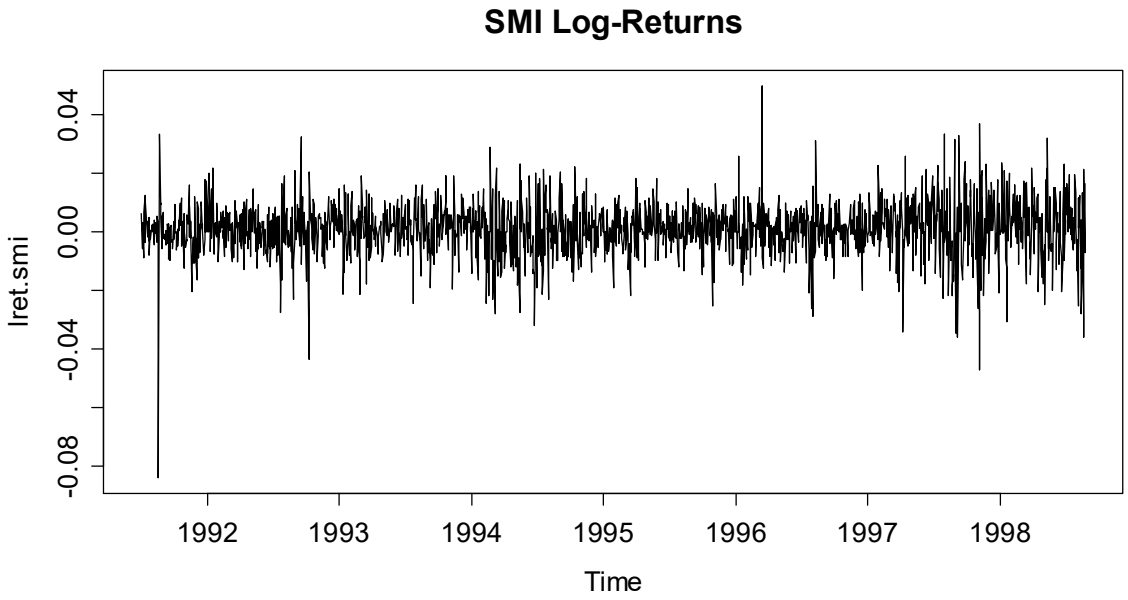
```
esm <- EuStockMarkets
tmp <- EuStockMarkets[,2]
smi <- ts(tmp, start=start(esm), freq=frequency(esm))
plot(smi, main="SMI Daily Closing Value")
```

Because subsetting from a multiple time series object results in a *vector*, but not a *time series object*, we need to regenerate a latter one, sharing the arguments of the original. In the plot we clearly observe that the series has a trend, i.e. the mean is obviously non-constant over time. This is typical for all financial time series.



Such trends in financial time series are nearly impossible to predict, and difficult to characterize mathematically. We will not embark in this, but analyze the so-called log-returns, i.e. the day-to-day changes after a log-transformation of the series:

```
> lret.smi <- diff(log(smi))
> plot(lret.smi, main="SMI Log-Returns")
```



These log-returns are a close approximation to the relative change (percent values) with respect to the previous day. As can be seen above, they do not exhibit a trend anymore, but show some of the stylized facts that most log-returns of financial time series share. Using *lagged scatterplots* or the *correlogram* (to be discussed later in section 4.4), you can convince yourself that there is no serial correlation. Thus, there is no dependency which could be exploited to predict tomorrows return based on the one of today and/or previous days.

However, it is visible that large changes, i.e. log-returns with high absolute values, imply that future log-returns tend to be larger than normal, too. This feature is also known as *volatility clustering*, and financial service providers are trying their best to exploit this property to make profit. Again, you can convince yourself of the volatility clustering effect by taking the squared log-returns and analyzing their serial correlation, which is different from zero.

1.3 Goals in Time Series Analysis

A first impression of the purpose and goals in time series analysis could be gained from the previous examples. We conclude this introductory section by explicitly summarizing the most important goals.

1.3.1 Exploratory Analysis

Exploratory analysis for time series mainly involves *visualization* with time series plots, *decomposition* of the series into deterministic and stochastic parts, and studying the *dependency structure* in the data.

1.3.2 Modeling

The formulation of a stochastic model, as it is for example also done in regression, can and does often lead to a deeper understanding of the series. The formulation of a suitable model usually arises from a mixture between background knowledge in the applied field, and insight from exploratory analysis. Once a suitable model is found, a central issue remains, i.e. the *estimation* of the parameters, and subsequent *model diagnostics* and *evaluation*.

1.3.3 Forecasting

An often-heard motivation for time series analysis is the prediction of future observations in the series. This is an ambitious goal, because time series forecasting relies on *extrapolation*, and is generally based on the assumption that past and present characteristics of the series continue. It seems obvious that good forecasting results require a very good comprehension of a series' properties, be it in a more descriptive sense, or in the sense of a fitted model.

1.3.4 Time Series Regression

Rather than just forecasting by extrapolation, we can try to understand the relation between a so-identified *response time series*, and one or more *explanatory series*. If all of these are observed at the same time, we can in principle employ the *ordinary least squares* (OLS) regression framework. However, the all-to-common assumption of (serially) uncorrelated errors in OLS is usually violated in a time series setup. We will illustrate how to properly deal with this situation, in order to generate correct confidence and prediction intervals.

1.3.5 Process Control

Many production or other processes are measured quantitatively for the purpose of *optimal management* and *quality control*. This usually results in time series data, to which a stochastic model is fit. This allows understanding the signal in the data, but also the noise: it becomes feasible to monitor which fluctuations in the production are normal, and which ones require intervention.

1.3.6 Time Series Clustering or Classification

In the modern world, more and more processes create online time series data. A typical task with these recordings is to *group snippets of time series according to their characteristics*, be it in a *supervised* fashion(classification) or *unsupervised* via clustering methods. Traditional approaches focus on extracting features from the time series methods, whereas novel ideas focus on neural networks that do not require feature specification.

2 Mathematical Concepts

For performing anything else than very basic exploratory time series analysis, even from a much applied perspective, it is necessary to introduce the mathematical notion of what a time series is, and to study some basic probabilistic properties, namely the *moments* and the *concept of stationarity*.

2.1 Definition of a Time Series

As we have explained in section 1.2, observations that have been collected over fixed sampling intervals form a time series. Following a statistical approach, we consider such series as realizations of random variables. A sequence of random variables, defined at such fixed sampling intervals, is sometimes referred to as a *discrete-time stochastic process*, though the shorter names *time series model* or *time series process* are more popular and will mostly be used in this scriptum. It is very important to make the distinction between a time series, i.e. observed values, and a process, i.e. a probabilistic construct.

Definition: A *time series process* is a set of random variables $\{X_t, t \in T\}$, where T is the set of times at which the process was, will or can be observed. We assume that each random variable X_t is distributed according some univariate distribution function F_t. Please note that for our entire course and hence scriptum, we exclusively consider time series processes with equidistant time intervals, as well as real-valued random variables X_t. This allows us to enumerate the set of times, so that we can write $T = \{1, 2, 3, \ldots\}$.

An observed time series, on the other hand, is seen as a realization of the random vector $X = (X_1, X_2, \ldots, X_n)$, and is denoted with small letters $x = (x_1, x_2, \ldots, x_n)$. It is important to note that in a multivariate sense, a time series is only *one single* realization of the n-dimensional random variable X, with its multivariate, n-dimensional distribution function $F_{1:n}$. As we all know, we cannot do statistics with just a single observation. As a way out of this situation, we need to impose some conditions on the joint distribution function $F_{1:n}$.

2.2 Stationarity

The aforementioned condition on the joint distribution $F_{1:n}$ will be formulated as the concept of *stationarity*. In colloquial language, stationarity means that the probabilistic character of the series must not change over time, i.e. that any section of the time series is “typical” for every other section with the same length. More mathematically, we require that for any indices s, t and k, the observations $x_t, \ldots, x_{t+k}$ could have just as easily occurred at times $s, \ldots, s+k$. If that is not the case practically, then the series is hardly stationary.

Imposing even more mathematical rigor, we introduce the concept of *strict stationarity*. A time series is said to be *strictly stationary* if and only if the $(k+1)$-dimensional joint distribution of $X_t,\ldots,X_{t+k}$ coincides with the joint distribution of $X_s,\ldots,X_{s+k}$ for any combination of indices t, s and k. For the special case of $k=0$ and $t=s$, this means that the univariate distributions F_t of all X_t are equal. For strictly stationary time series, we can thus leave off the index t on the distribution. As the next step, we will define the unconditional moments:

Expectation	μ	=	$E[X_t]$,
Variance	σ^2	=	$Var(X_t)$,
Covariance	$\gamma(h)$	=	$Cov(X_t, X_{t+h})$.

In other words, strictly stationary series have *constant (unconditional) expectation*, *constant (unconditional) variance* , and the covariance, i.e. the *dependency structure, depends only on the lag* h, which is the time difference between the two observations. However, the covariance terms are generally different from 0, and thus, the X_t are usually dependent. Moreover, the conditional expectation given the past of the series, $E[X_t \mid X_{t-1}, X_{t-2},...]$ is typically non-constant, denoted as μ_t. In some (rarer, e.g. for financial time series) cases, even the conditional variance $Var(X_t \mid X_{t-1}, X_{t-2},...)$ can be non-constant.

In practice however, except for simulation studies, we usually have no explicit knowledge of the latent time series process. Since strict stationarity is defined as a property of the process' joint distributions (all of them), it is impossible to verify from an observed time series, i.e. a single data realization. We can, however, try to verify whether a time series process shows *constant unconditional mean and variance*, and whether the *dependency only depends on the lag* h. This much less rigorous property is known as *weak stationarity*.

In order to do well-founded statistical analyses with time series, weak stationarity is a necessary condition. It is obvious that if a series' observations do not have common properties such as constant mean/variance and a stable dependency structure, it will be impossible to statistically learn from it. On the other hand, it can be shown that weak stationarity, along with the additional property of ergodicity (i.e. the mean of a time series realization converges to the expected value, independent of the starting point), is sufficient for most practical purposes such as model fitting, forecasting, etc.. We will, however, not further embark in this subject.

Remarks:

- From now on, when we speak of *stationarity*, we strictly mean weak stationarity. The motivation is that weak stationarity is sufficient for applied time series analysis, and strict stationarity is a practically useless concept.

- When we analyze time series data, we need to verify whether it might have arisen from a stationary process or not. Be careful with the wording: stationarity is always a property of the process, and never of the data.

- Moreover, bear in mind that stationarity is a hypothesis, which needs to be evaluated for every series. We may be able to reject this hypothesis with quite some certainty if the data strongly speak against it. However, we can never prove stationarity with data. At best, it is plausible that a series originated from a stationary process.

- Some obvious violations of stationarity are trends, non-constant variance, deterministic seasonal variation, as well as apparent breaks in the data, which are indicators for changing dependency structure.

2.3 Testing Stationarity

If, as explained above, stationarity is a hypothesis which is tested on data, students and users keep asking if there are any formal tests. The answer to this question is yes, and there are even quite a number of tests. This includes the *Augmented Dickey-Fuller Test*, the *Phillips-Perron Test*, the *KPSS Test*, which are all available in **R**'s `tseries` package. The `urca` package includes further tests such as the *Elliott-Rothenberg-Stock*, *Schmidt-Phillips* und *Zivot-Andrews*.

However, we will not discuss any of these tests here for a variety of reasons. First and foremost, they all focus on some very specific non-stationarity aspects, but do not test stationarity in a broad sense. While they may reasonably do their job in the narrow field they are aimed for, they have low power to detect general non-stationarity and in practice often fail to do so. Additionally, theory and formalism of these tests is quite complex, and thus beyond the scope of this course. In summary, these tests are to be seen as more of a pasttime for the mathematically interested, rather than a useful tool for the practitioner.

Thus, we here recommend assessing stationarity by visual inspection. The primary tool for this is the time series plot, but also the correlogram (see section 4.4) can be helpful as a second check. For long time series, it can also be useful to split up the series into several parts for checking whether mean, variance and dependency are similar over the blocks.

3 Time Series in R

3.1 Time Series Classes

In **R**, there are *objects*, which are organized in a large number of *classes*. These classes e.g. include *vectors*, *data frames*, *model output*, *functions*, and many more. Not surprisingly, there are also several classes for time series. We start by presenting `ts`, the basic class for regularly spaced time series. This class is comparably simple, as it can only represent time series with fixed interval records, and only uses numeric time stamps, i.e. (sophistically) enumerates the index set. However, it will be sufficient for most, if not all, of what we do in this course. Then, we also provide an outlook to more complicated concepts.

3.1.1 The ts Class

For defining a time series of class `ts`, we of course need to provide the *data*, but also the *starting time* as argument `start`, and the *frequency* of measurements as argument `frequency`. If no starting time is supplied, **R** uses its default value of 1, i.e. enumerates the times by the index set $1, ..., n$, where n is the length of the series. The frequency is the number of observations per unit of time, e.g. 1 for yearly, 4 for quarterly, or 12 for monthly recordings. Instead of the start, we could also provide the end of the series, and instead of the frequency, we could supply argument `deltat`, the fraction of the sampling period between successive observations. The following example will illustrate the concept.

Example: We here consider a simple and short series that holds the number of days per year with traffic holdups in front of the Gotthard road tunnel north entrance in Switzerland. The data are available from the Federal Roads Office.

2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
88	76	112	109	91	98	139	150	168	149

The start of this series is in 2004. The time unit is years, and since we have just one record per year, the frequency of this series is 1. This tells us that while there may be a trend, there cannot be a seasonal effect, as the latter can only be present in periodic series, i.e. series with frequency > 1. We now define a `ts` object in in **R**.

```
> rawdat <- c(88, 76, 112, 109, 91, 98, 139, 150, 168, 149)
> ts.dat <- ts(rawdat, start=2004, freq=1)
> ts.dat
Time Series: Start = 2004, End = 2013
Frequency = 1
[1]  88  76 112 109  91  98 139 150 168 149
```

There are a number of simple but useful functions that extract basic information from objects of class `ts`, see the following examples:

```
> start(ts.dat)
[1] 2004    1

> end(ts.dat)
[1] 2013    1

> frequency(ts.dat)
[1] 1

> deltat(ts.dat)
[1] 1
```

Another possibility is to obtain the measurement times from a time series object. As class `ts` only enumerates the times, they are given as fractions. This can still be very useful for specialized plots, etc.

```
> time(ts.dat)
Time Series:
Start = 2004
End = 2013
Frequency = 1
[1] 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013
```

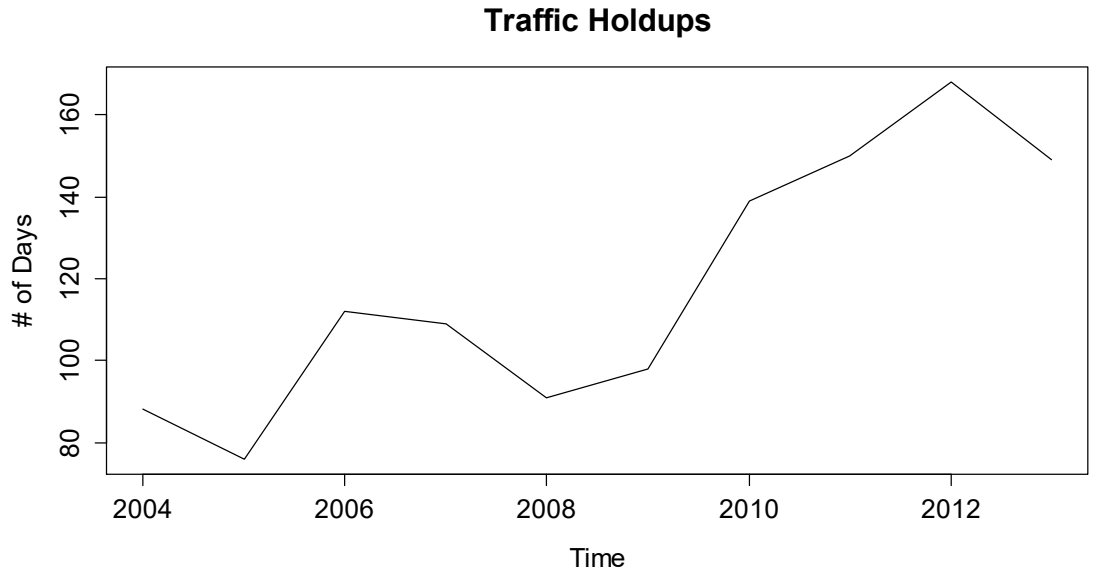
The next basic, but for practical purposes very useful function is `window()`. It is aimed at selecting a subset from a time series. Of course, also regular **R**-subsetting such as `ts.dat[2:5]` does work with the time series class. However, this results in a vector rather than a time series object, and is thus mostly of less use than the `window()` command.

```
> window(ts.dat, start=2006, end=2008)
Time Series:
Start = 2006
End = 2008
Frequency = 1
[1] 112 109  91
```

While we here presented the most important basic methods/functions for class `ts`, there is a wealth of further ones. This includes the `plot()` function, and many more, e.g. for estimating trends, seasonal effects and dependency structure, for fitting time series models and generating forecasts. We will present them in the forthcoming chapters of this scriptum.

To conclude the previous example, we will not do without showing the time series plot of the Gotthard road tunnel traffic holdup days, see next page. Because there are a limited number of observations, it is difficult to give statements regarding a possible trend and/or stochastic dependency.

```
> plot(ts.dat, ylab="# of Days", main="Traffic Holdups")
```



3.1.2 Finding the Frequency

Often, finding the frequency for defining the time series is straightforward. As mentioned above, it is the number of observations per unit of time, e.g. 1 for yearly, 4 for quarterly, or 12 for monthly recordings. However, some real-world situation are quite a bit more complex to handle. Principally, it is up to the user to choose the correct frequency from background and field knowledge about the measurements. Additionally, R function `findfrequency()` may assist. It provides the correct results for the traffic holdups and the air passenger data:

```
> findfrequency(ts.dat)
[1] 1
> findfrequency(AirPassengers)
[1] 12
```

On the other hand, it can also provide misleading results. If we apply the function to the lynx data, we obtain:

```
> findfrequency(lynx)
[1] 10
```

The lynx data are clearly cyclic with a period of about 10 years. We interpret these cycles as stochastic though and the frequency should be set to a value of 1. In other cases, there may be ambiguity in the definition of the frequency. If we for example consider the a time series of the minutely averaged electricity demand in a city, the frequency may be:

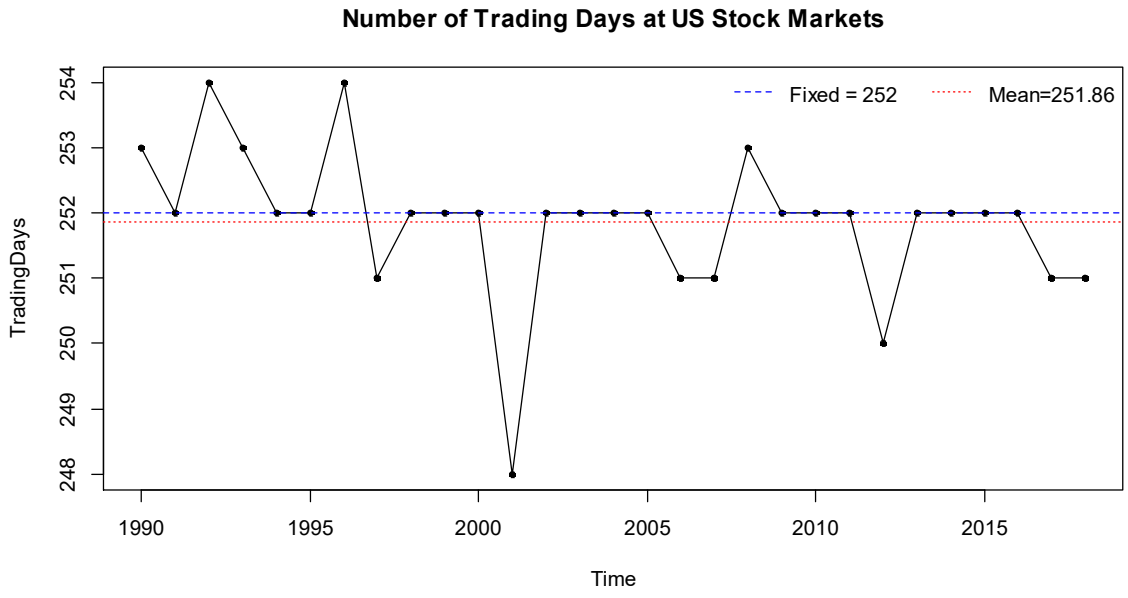
- Hourly (i.e. $f = 60$)
- Daily (i.e. $f = 24 \cdot 60 = 1'440$)
- Weekly (i.e. $f = 24 \cdot 60 \cdot 7 = 10'080$)
- Yearly (i.e. $f = 24 \cdot 60 \cdot 365 = 525'600$)

When working with the `ts()` class, we need to decide for one single frequency. That may be far from easy, because the power demand may have a hourly, daily, weekly and yearly pattern. The simple rule of the thumb is to pick the frequency which is the most natural, the strongest or the most central for the analysis which is carried out. Sometimes (*especially for providing accurate results in forecasting*), all cyclic components need to be kept under the radar. For achieving this, clever decomposition approaches may help. Moreover, there is the advanced `msts()` class in R that allows time series to have multiple "frequencies". As soon as one extensively deals with weekly or daily data, further problems will appear. Namely, the number of observations per time unit may not be constant or not an integer.

- *Weekly data*: even in the simple case where all (observation) years have exactly 365 days, we obtain a non-integer frequency of $f = 365/7 = 52.14$.

- *Daily data:* the problem here arises from the leap years that have 366 days. As they (roughly) happen every 4th year, the quick fix is to set $f = 365.25$. This is still somewhat imprecise, because the leap year rules are more complicated and sometimes leap seconds are used, altering the astronomically correct frequency slightly. In most practical cases (if the time series does not comprise of hundreds or thousands of observations years) it won't make much practical difference, though.

- *Trading or working day data:* the number of working days per year fluctuates even more, so that the definition of the frequency becomes tricky. One either works with a "representative" integer value or the mean resp. median number of working days per year. Most R functions also accept non-integer frequency values, making this strategy viable. Below we have the number of trading days at US Stock Markets. Commong knowledge says that "it's usually 252 trading days a year", suggesting this value for the frequency. Alternatively, the mean of 251.86 could be used.



3.1.3 Other Classes

Besides the basic `ts` class, there are several other classes which offer a variety of additional options. Most are designed for specific and advanced tasks, so that they will rarely to never be required during our course. Most prominently, this includes the `zoo` package, which provides infrastructure for both regularly and irregularly spaced time series using arbitrary classes for the time stamps. It is designed to be as consistent as possible with the `ts` class. Coercion from and to `zoo` is also readily available.

Some further packages which contain classes and methods for time series include `xts`, `its`, `tseries`, `fts`, `timeSeries` and `tis`. Additional information on their content and philosophy can be found on CRAN.

3.2 Dates and Times in R

While for the `ts` class, the handling of times has been solved very simply and easily by enumerating, doing time series analysis in R may sometimes also require to explicitly working with date and time. There are several options for dealing with date and date/time data. The built-in `as.Date()` function handles dates that come without times. The contributed package `chron` handles dates and times, but does not control for different time zones, whereas the sophisticated but complex `POSIXct` and `POSIXlt` classes allow for dates and times with time zone control.

As a general rule for date/time data in **R**, we suggest to use the simplest technique possible. Thus, for date only data, `as.Date()` will mostly be the optimal choice. If handling dates and times, but without time-zone information, is required, the `chron` package is the choice. The `POSIX` classes are especially useful in the relatively rare cases when time-zone manipulation is important.

Apart for the `POSIXlt` class, dates/times are internally stored as the number of days or seconds from some reference date. These dates/times thus generally have a numeric mode. The `POSIXlt` class, on the other hand, stores date/time values as a list of components (`hour`, `min`, `sec`, `mon`, etc.), making it easy to extract these parts. Also the current date is accessible by typing `Sys.Date()` in the console, and returns an object of class `Date`.

3.2.1 The Date Class

As mentioned above, the easiest solution for specifying days in R is with the `as.Date()` function. Using the format argument, arbitrary date formats can be read. The default, however, is four-digit year, followed by month and then day, separated by dashes or slashes:

```
> as.Date("2012-02-14")
[1] "2012-02-14"
```

```
> as.Date("2012/02/07")
[1] "2012-02-07"
```

If the dates come in non-standard appearance, we require defining their format using some codes. While the most important ones are shown below, we reference to the **R** help file of function `strptime()` for the full list.

Code	Value
%d	Day of the month (decimal number)
%m	Month (decimal number)
%b	Month (character, abbreviated)
%B	Month (character, full name)
%y	Year (decimal, two digit)
%Y	Year (decimal, four digit)

The following examples illustrate the use of the `format` argument:

```
> as.Date("27.01.12", format="%d.%m.%y")
[1] "2012-01-27"
> as.Date("14. Februar, 2012", format="%d. %B, %Y")
[1] "2012-02-14"
```

Internally, Date objects are stored as the number of days passed since the 1st of January in 1970. Earlier dates receive negative numbers. By using the `as.numeric()` function, we can easily find out how many days are past since the reference date. Also back-conversion from a number of past days to a date is straightforward:

```
> mydat <- as.Date("2012-02-14")
> ndays <- as.numeric(mydat)
> ndays
[1] 15384
> tdays <- 10000
> class(tdays) <- "Date"
> tdays
[1] "1997-05-19"
```

A very useful feature is the possibility of extracting weekdays, months and quarters from `Date` objects, see the examples below. This information can be converted to factors. In this form, they serve for purposes such as visualization, decomposition, or time series regression.

```
> weekdays(mydat)
[1] "Dienstag"
> months(mydat)
[1] "Februar"
> quarters(mydat)
[1] "Q1"
```

Furthermore, some very useful summary statistics can be generated from `Date` objects: `median`, `mean`, `min`, `max`, `range`, ... are all available. We can even subtract two dates, which results in a `difftime` object, i.e. the time difference in days.

```
> dat <- as.Date(c("2000-01-01","2004-04-04","2007-08-09"))
> dat
[1] "2000-01-01" "2004-04-04" "2007-08-09"

> min(dat)
[1] "2000-01-01"
> max(dat)
[1] "2007-08-09"
> mean(dat)
[1] "2003-12-15"
> median(dat)
[1] "2004-04-04"

> dat[3]-dat[1]
Time difference of 2777 days
```

Another option is generating time sequences. For example, to generate a vector of 12 dates, starting on August 3, 1985, with an interval of one single day between them, we simply type:

```
> seq(as.Date("1985-08-03"), by="days", length=12)
 [1] "1985-08-03" "1985-08-04" "1985-08-05" "1985-08-06"
 [5] "1985-08-07" "1985-08-08" "1985-08-09" "1985-08-10"
 [9] "1985-08-11" "1985-08-12" "1985-08-13" "1985-08-14"
```

The `by` argument proves to be very useful. We can supply various units of time, and even place an integer in front of it. This allows creating a sequence of dates separated by two weeks:

```
> seq(as.Date("1992-04-17"), by="2 weeks", length=12)
 [1] "1992-04-17" "1992-05-01" "1992-05-15" "1992-05-29"
 [5] "1992-06-12" "1992-06-26" "1992-07-10" "1992-07-24"
 [9] "1992-08-07" "1992-08-21" "1992-09-04" "1992-09-18"
```

3.2.2 The chron Package

The `chron()` function converts dates and times to `chron` objects. The dates and times are provided separately to the `chron()` function, which may well require some inital pre-processing. For such parsing, **R**-functions such as `substr()` and `strsplit()` can be of great use. In the `chron package`, there is no support for time zones and daylight savings time, and `chron` objects are internally stored as fractional days since the reference date of January 1st, 1970. By using the function `as.numeric()`, these internal values can be accessed. The following example illustrates the use of `chron`:

```
> library(chron)
```

```
> dat <- c("2007-06-09 16:43:20", "2007-08-29 07:22:40",
           "2007-10-21 16:48:40", "2007-12-17 11:18:50")
> dts <- substr(dat,  1, 10)
> tme <- substr(dat, 12, 19)
> fmt <- c("y-m-d","h:m:s")
> cdt <- chron(dates=dts, time=tme, format=fmt)
> cdt
[1] (07-06-09 16:43:20) (07-08-29 07:22:40)
[3] (07-10-21 16:48:40) (07-12-17 11:18:50)
```

As before, we can again use the entire palette of summary statistic functions. Of some special interest are time differences, which can now be obtained as either fraction of days, or in weeks, hours, minutes, seconds, etc.:

```
> cdt[2]-cdt[1]
Time in days:
[1] 80.61065
> difftime(cdt[2], cdt[1], units="secs")
Time difference of 6964760 secs
```

3.2.3 POSIX Classes

The two classes `POSIXct` and `POSIXlt` implement date/time information, and in contrast to the `chron` package, also support time zones and daylight savings time. We recommend utilizing this functionality only when urgently needed, because the handling requires quite some care, and may on top of that be system dependent. Further details on the use of the POSIX classes can be found on CRAN.

As explained above, the `POSIXct` class also stores dates/times with respect to the internal reference, whereas the `POSIXlt` class stores them as a list of components (`hour`, `min`, `sec`, `mon`, etc.), making it easy to extract these parts.

3.3 Data Import

We can safely assume that most time series data are already present in electronic form; however, not necessarily in **R**. Thus, some knowledge on how to import data into **R** is required. It is be beyond the scope of this scriptum to present the uncounted options which exist for this task. Hence, we will restrict ourselves to providing a short overview and some useful hints.

The most common form for sharing time series data are certainly spreadsheets, or in particular, Microsoft Excel files. While `library(ROBDC)` offers functionality to directly import data from Excel files, we discourage its use. First of all, this only works on Windows systems. More importantly, it is usually simpler, quicker and more flexible to export comma- or tab-separated text files from Excel, and import them via the ubiquitous `read.table()` function, respectively the tailored version `read.csv()` (for comma separation) and `read.delim()` (for tab separation).

With packages `ROBDC` and `RMySQL`, **R** can also communicate with SQL databases, which is the method of choice for large scale problems. Furthermore, after loading `library(foreign)`, it is also possible to read files from Stata, SPSS, Octave and SAS.

4 Descriptive Analysis

As always when working with data, i.e. “a pile of numbers”, it is important to gain an overview. In time series analysis, this encompasses several aspects:

- understanding the context of the problem and the data source
- making suitable plots, looking for general structure and outliers
- thinking about data transformations, e.g. to reduce skewness
- judging stationarity and potentially achieve it by decomposition
- for stationary series, the analysis of the autocorrelation function

We start by discussing time series plots, then discuss transformations, focus on the decomposition of time series into trend, seasonal effect and stationary random part and conclude by discussing methods for visualizing the dependency structure.

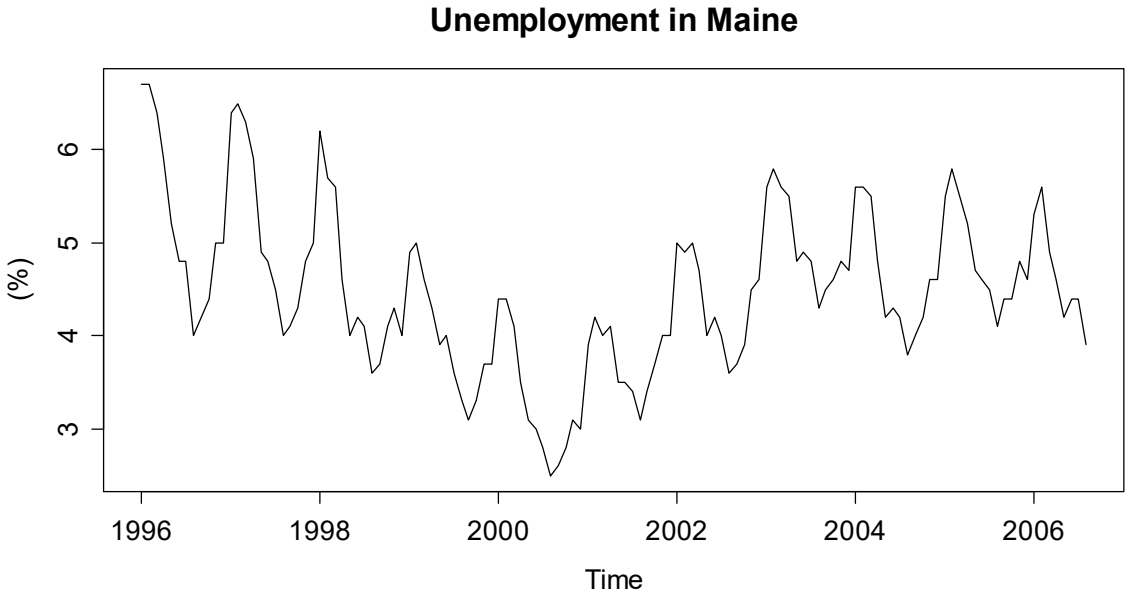
4.1 Visualization

4.1.1 Time Series Plot

The most important means of visualization is the time series plot, where the data are plotted versus time/index. There are several examples in section 1.2, where we also got acquainted with **R**’s generic `plot()` function. As a general rule, the data points are joined by lines in time series plots. This is despite the data are not continuous, as the plots are much easier to read in this form. The only exception where gaps are left is if there are missing values. Moreover, the reader expects that the axes are well-chosen, labeled and the measurement units are given.

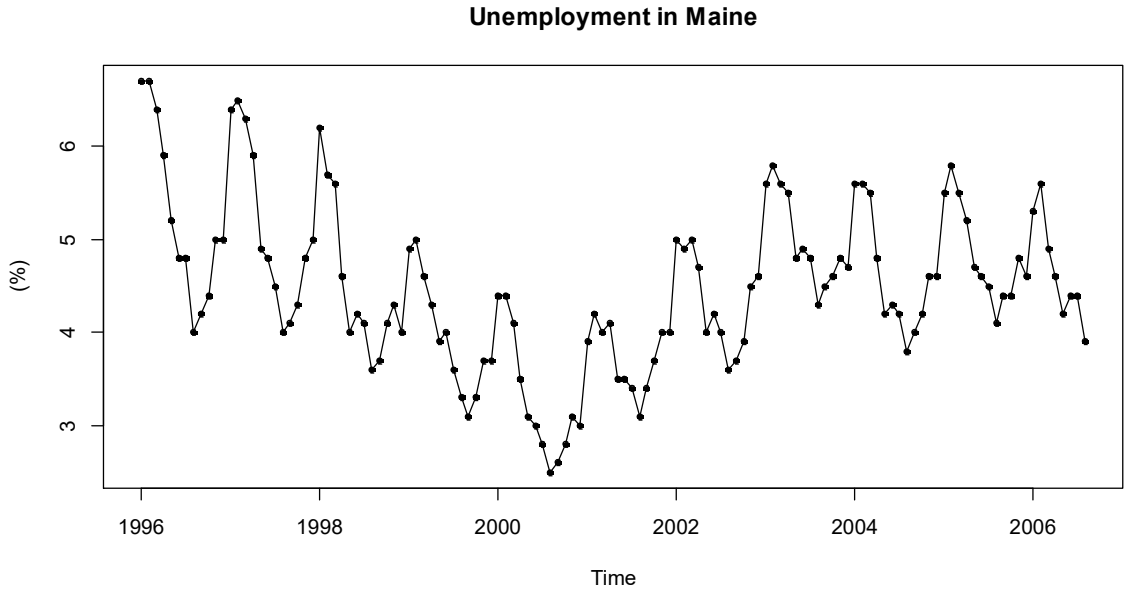
Another issue is the correct aspect ratio for time series plots: if the time axis gets too much compressed, it can become difficult to recognize the behavior of a series. Thus, we recommend choosing the aspect ratio appropriately. However, there are no hard and simple rules on how to do this. As a rule of the thumb, use the “banking the angle to 45 degrees” paradigm: increase and decrease in periodic series should not be displayed at angles much higher or lower than 45 degrees. For very long series, this can become difficult on either A4 paper or a computer screen. In this case, we recommend splitting up the series and display it in different frames. For illustration, we here show an example, the monthly unemployment rate in the US state of Maine, from January 1996 until August 2006. The data originate from the book "Introductory Time Series with R" by Cowpertwait & Metcalfe (https://www.springer.com/gp/book/9780387886978). The website has a zipped folder where you can find `Maine.dat`. Or alternatively, uses the lecturers preprocessed file `unemployment.rda`.

```
> load("unemployment.rda")
> plot(unemp, ylab="(%)", main="Unemployment in Maine")
```



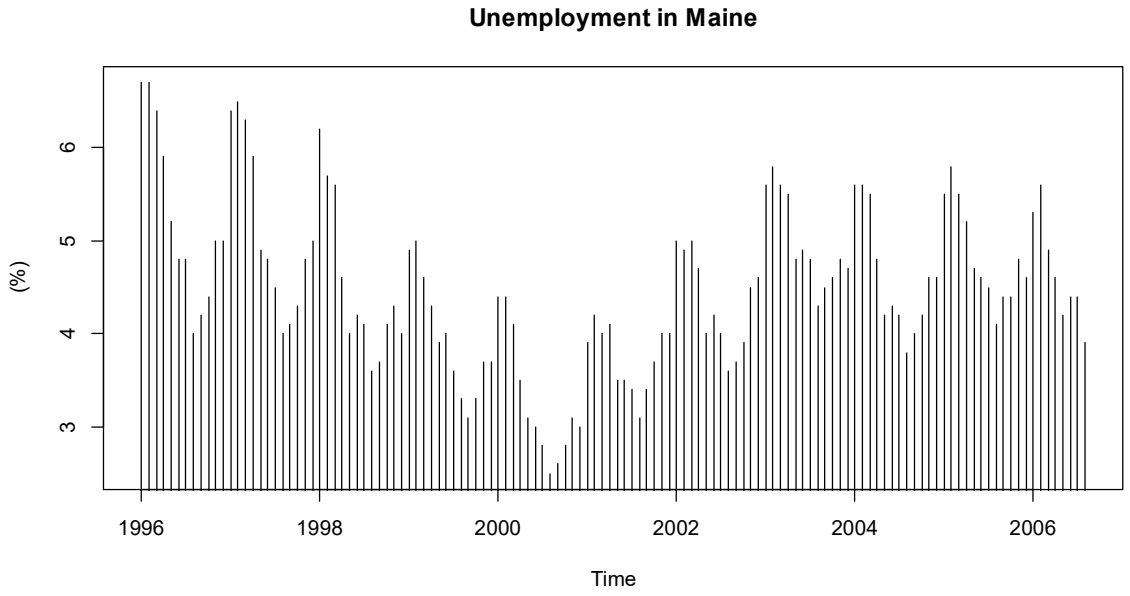
Not surprisingly for monthly economic data, the series shows both a non-linear trend and a seasonal pattern that increases with the level of the series. Hence, using a log-tranformation as explained in section 4.2 may be adviseable. Since unemployment rates are one of the main economic indicators used by politicians/decision makers, this series poses a worthwhile forecasting problem.

```
> plot(unemp, type="o", pch=20, ylab="(%)", main="…")
```



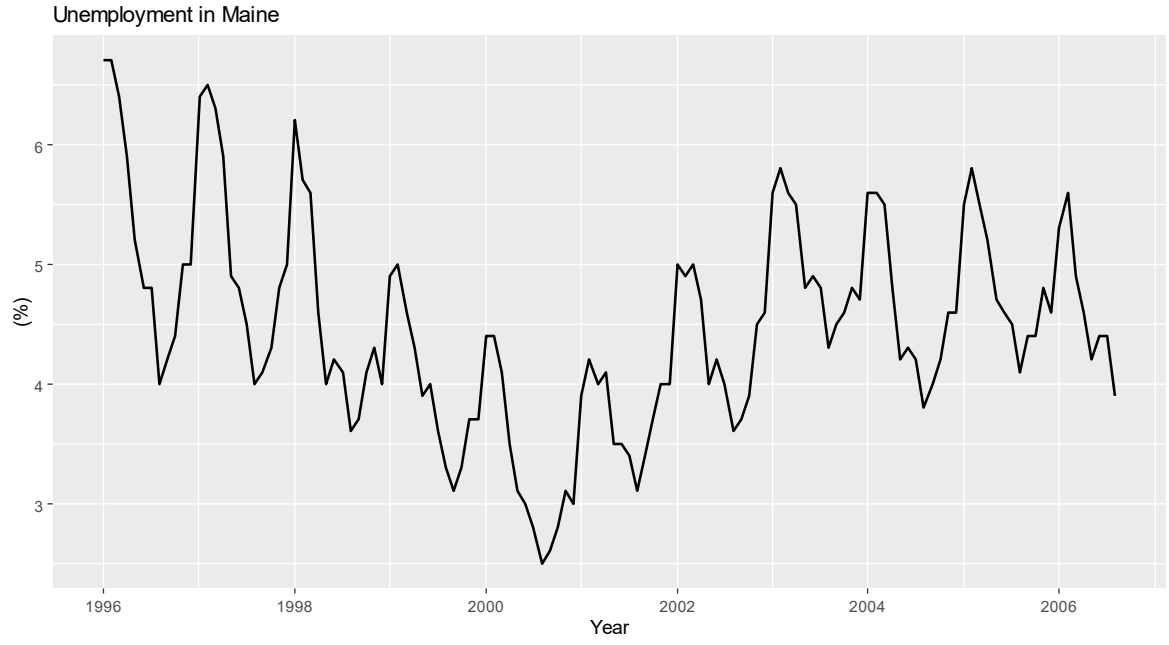
There are various ways by which time series plots can be enhanced. In some cases when only relatively few data points are present and there is no distinct seasonal pattern, "adding the points" with argument `type="o"` may be worthwhile. In other applications, it has become the quasi-norm to plot vertical lines by `type="h"` rather than the time series plot shown above.

```
> plot(unemp, type="h", ylab="(%)", main="…")
```



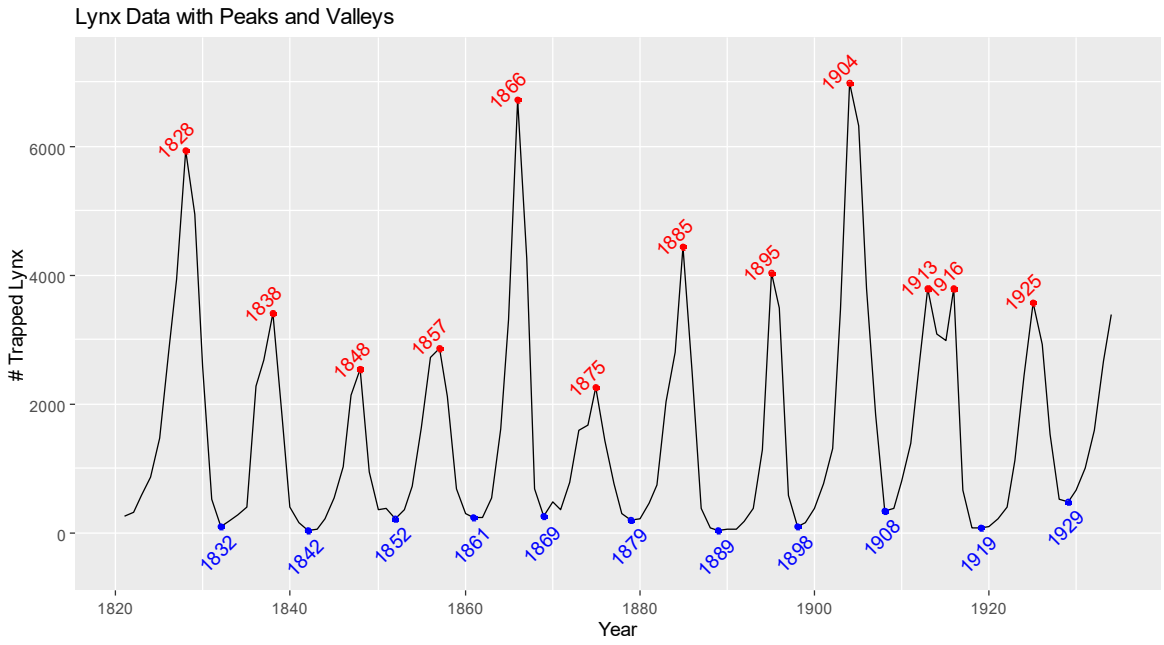
In recent years, the `ggplot2`-package in R with it's elegant graphics has become very popular. Generating a simple time series plot requires a bit more effort than with R standard graphics, but the main advantage lies in the numerous enhancements for complex data analysis situations that are relatively straightforward. It is however beyond the scope of this course to give extensive details about `ggplot2` and unless needed, we will work with R standard graphics throughout this script.

```
> ggplot(unemp, as.numeric=FALSE) + geom_line(size=1) +
+   ggtitle("Unemp…") + xlab("Year") + ylab("(%)")
```



We finish this chapter with an automatically annoted time series plot of the lynx data that was generated with an add-on package to `ggplot2`. Producing fancy graphics has become its own discipline in data science and is certainly worthwhile.

```
> ggplot(lynx, as.numeric = FALSE) + geom_line() +
    ggtitle("Lynx Data with Peaks and Valleys") +
    stat_peaks(colour = "red") +
    stat_peaks(geom = "text", colour = "red",
               vjust = -0.5, x.label.fmt = "%Y") +
    stat_valleys(colour = "blue") +
    stat_valleys(geom = "text", colour = "blue", angle = 45,
                 vjust = 1.5, hjust = 1, x.label.fmt = "%Y")+
    ylim(-500, 7300) + xlab("Year") + ylab("# Trapped Lynx")
```



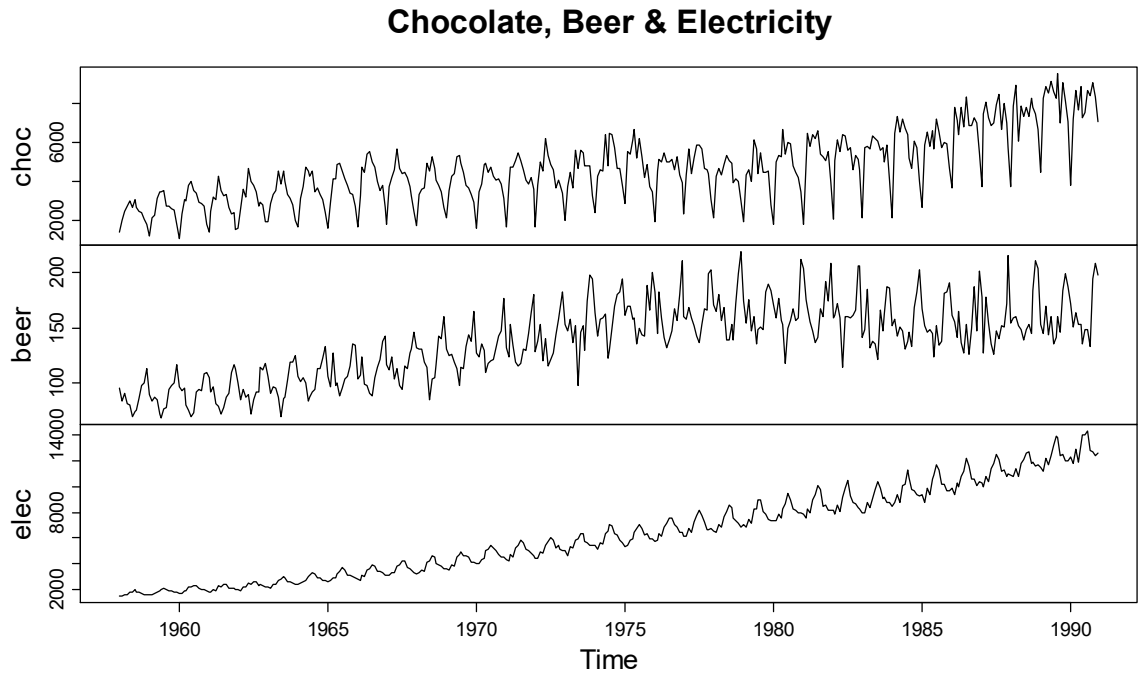
4.1.2 Multiple Time Series Plots

In applied problems, one is sometimes provided with multiple time series. Here, we illustrate some basics on import, definition and plotting. Our example exhibits the monthly supply of electricity (millions of kWh), beer (millions of liters) and chocolate-based production (tonnes) in Australia over the period from January 1958 to December 1990. These data were published by the Bureau of Australian Statistics and are presented in the book of Cowpertwait & Metcalfe.

```
> dat <- read.table("cbe.dat",sep="", header=T)
> cbe <- ts(dat, start=1958, freq=12)
```

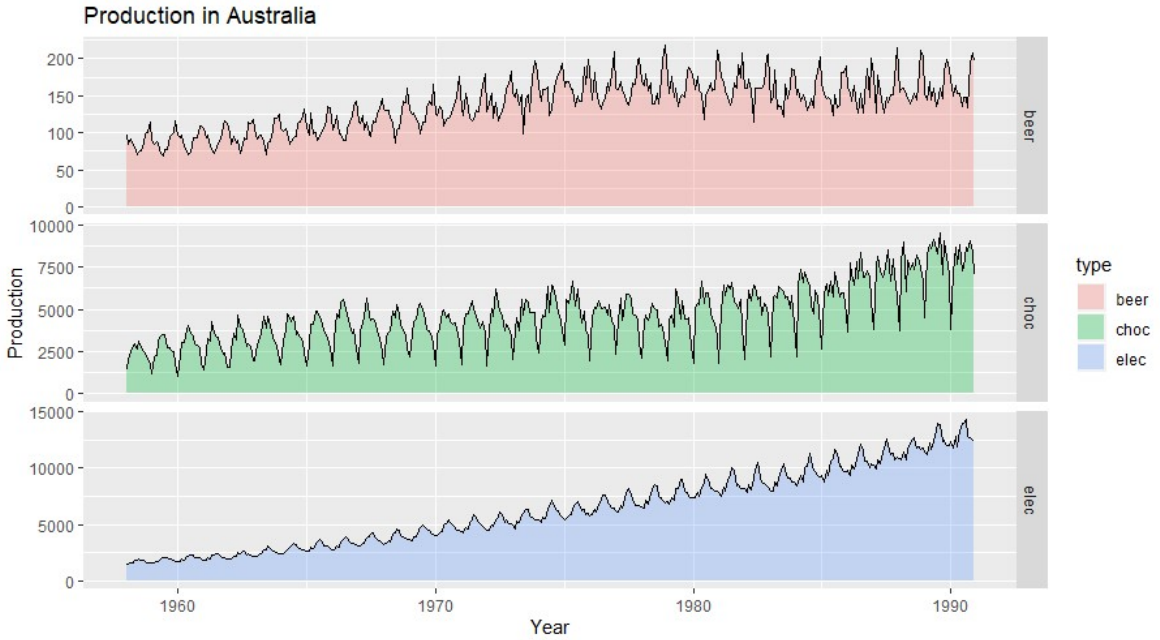
This creates a multiple time series object that can very easily be displayed using the generic plot command again, although the presentation turns out to be a bit dull, see next page.

```
> plot(tsd, main="Chocolate, Beer & Electricity")
```



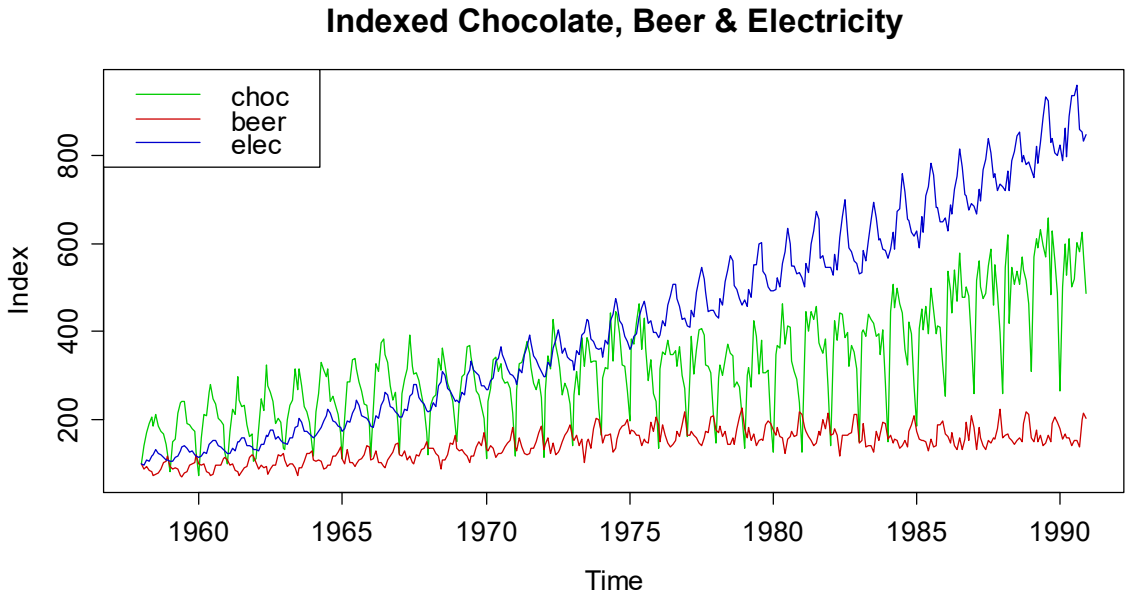
A much nicer plot can be produced using the `ggplot2`-package using different facets for the three series. However, it does not directly work with the multiple time series object as the input, but requires creating a data frame in long format.

```
> cbedf <- data.frame(t=rep(as.numeric(time(cbe)), times=3),
          values=c(cbe[,1], cbe[,2], cbe[,3]),
          type=rep(c("choc", "beer", "elec"), each=nrow(cbe)))
> ggplot(cbedf, aes(time, values, fill=type)) +
    geom_area(alpha=0.3, size=1) + geom_line() +
    facet_grid(type~., scales="free") +
    ggtitle("Production in Australia") +
    xlab("Year") + ylab("Production")
```



All three series show a distinct seasonal pattern, along with a trend. It is also instructive to know that the Australian population increased by a factor of 1.8 during the period where these three series were observed. As a general rule, using different frames for multiple series is the most recommended means of visualization. However, sometimes it can be more instructive to have them in the same frame. Of course, this requires that the series are either on the same scale, or have been indexed, resp. standardized to be so. Then, we can simply use `plot(ind.tsd, plot.type="single")`. When working with one single panel, we recommend to use different colors for the series, which is easily possible using a `col=c("green3", "red3", "blue3")` argument.

```
## Indexing the series
tsd     <- cbe
tsd[,1] <- tsd[,1]/tsd[1,1]*100
tsd[,2] <- tsd[,2]/tsd[1,2]*100
tsd[,3] <- tsd[,3]/tsd[1,3]*100

## Plotting in one single frame
clr <- c("green3", "red3", "blue3")
plot(tsd, plot.type="single", ylab="Index", col=clr)
title("Indexed Chocolate, Beer & Electricity")

## Legend
ltxt <- names(dat)
legend("topleft", lty=1, col=clr, legend=ltxt)
```



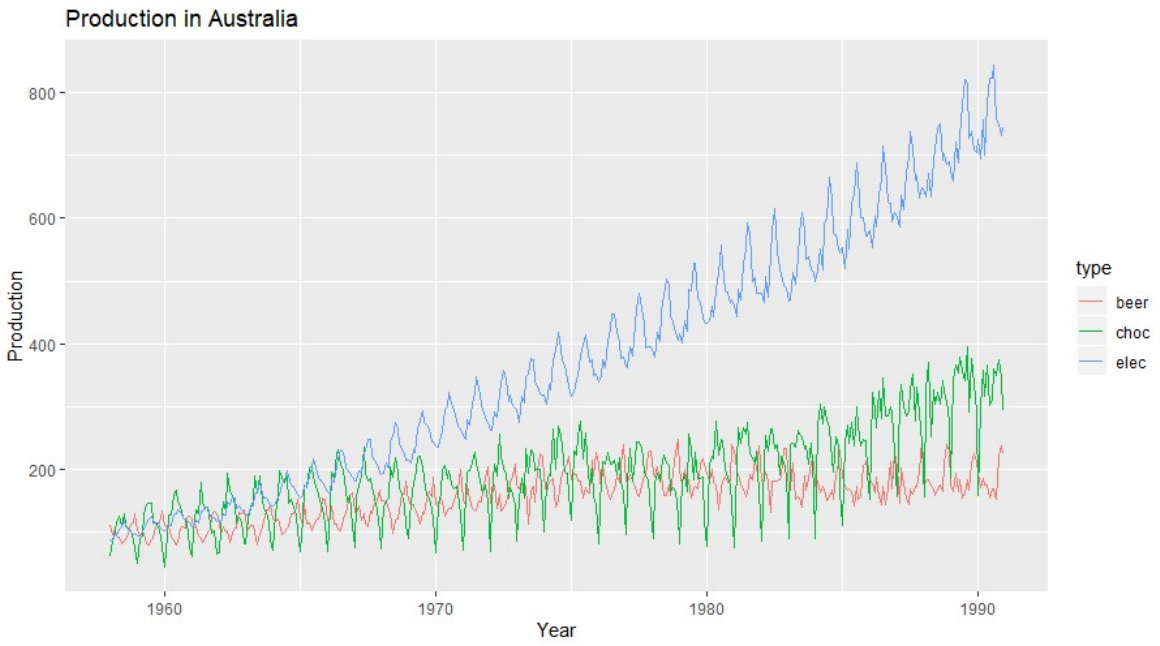
In the indexed single frame plot above, we can very well judge the relative development of the series over time. Due to different scaling, this was nearly impossible with the multiple frames on the previous page. We observe that electricity production increased around 8x during 1958 and 1990, whereas for chocolate the multiplier is around 4x, and for beer less than 2x. Also, the seasonal variation is most pronounced for chocolate, followed by electricity and then beer.

A special remark needs to be made about the indexing. In the bit of code above, the series were standardized using their first observed value. For seasonal time series, this may be a suboptimal strategy as one of the series may have the highpoint at the start observation, whereas for another series with an opposite pattern it may be the lowpoint. In such cases, it is usually beneficial to take the entire first period as the reference, i.e.:

```
> ## Indexing the series vs. the first period
> tsd     <- cbe
> tsd[,1] <- tsd[,1]/mean(tsd[1:12,1])*100
> tsd[,2] <- tsd[,2]/mean(tsd[1:12,2])*100
> tsd[,3] <- tsd[,3]/mean(tsd[1:12,3])*100
```

For complementing this chapter about visualization of time series, we present another output that was produced with `ggplot2`. As is typical for this package, adding different colors, legends et cetera, i.e. enhancing the basic plot is straightforward (if you know how to do so) and requires less code than the standard plots in R.

```
> ## Graphical display with ggplot
> ggplot(cbedf, aes(time, values, color=type)) +
    geom_line() +
    ggtitle("Production in Australia") +
    xlab("Year") +
    ylab("Production")
```



Qualitatively, there is also a marked difference between the first version of the plot using only the first value as the reference vs. this later one that standardizes with the first year, notably for the evolution of beer and chocolate consumption. We conclude the chapter by emphasizing that graphical displays of time series should be well chosen and reflected.

4.2 Transformations

Time series data do not necessarily need to be analyzed in the form they were provided to us. In many cases, it is much better, more efficient and instructive to transform the data. We will here highlight several cases and discuss their impact on the results.

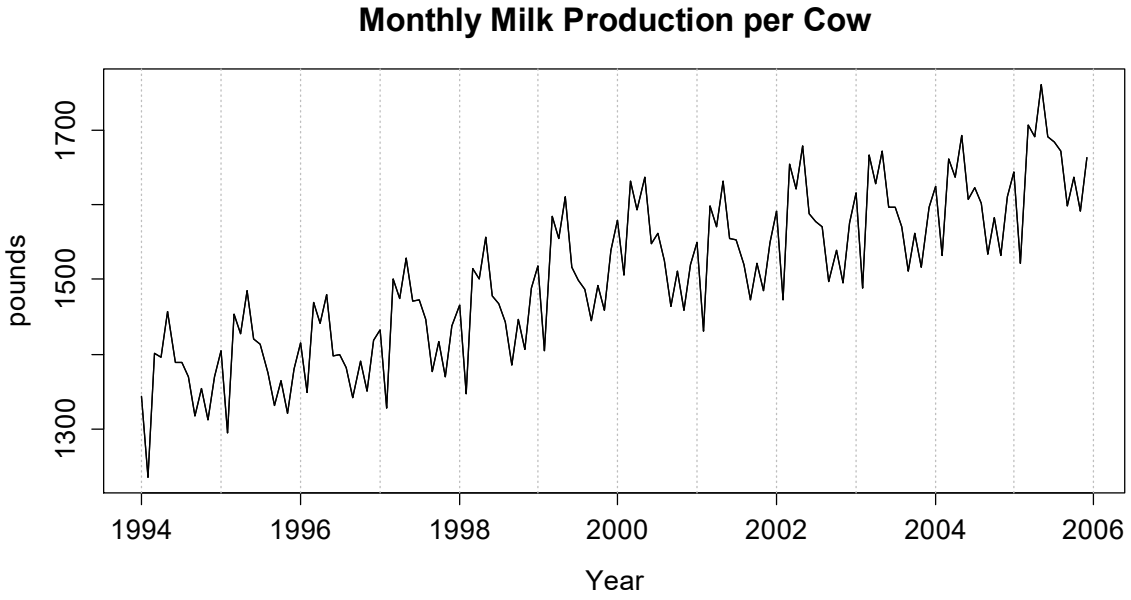
4.2.1 Linear Transformations

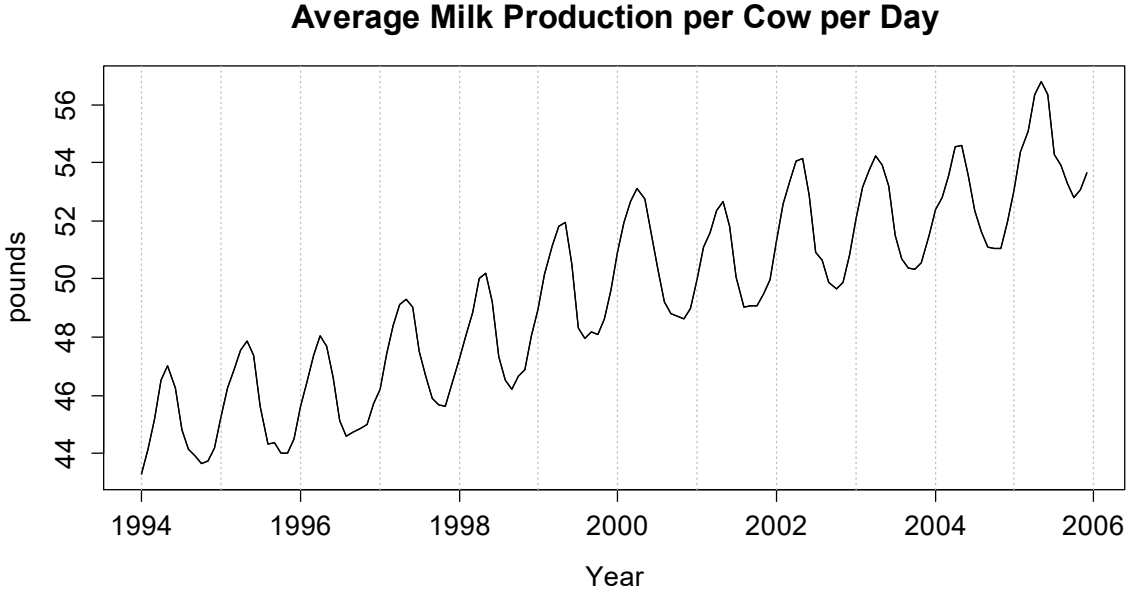
A linear transformation is of the form $Y_t = a + bX_t$. Examples include simple changes in units, e.g. from meters to kilometers, kilograms to tons, et cetera. Also slightly more complicated conversions as for example brining Fahrenheit temperatures to the Celsius scale fall under this definition. It is obvious that such linear transformations will not change the appearance of the series. Hence, all derived results (i.e. autocorrelations, models, forecasts) will be equivalent. As a consequence, we are free to perform linear transformations whenever it seems convenient.

4.2.2 Monthly Sums and Averages

Often in time series analysis we consider monthly data and often these are delivered as monthly totals. However, this adds unnecessary noise to the series, simply because of the different number of days per month. Often, the seasonal effect becomes much cleaner and easier to understand if we switch to the daily average per month rather than considering the monthly total. From simplified patterns, we humans as well as prediction models usually are more successful in extracting the relevant information. Additionally, using daily averages also manages to deal with the leap year problem, since in every fourth year, February will have 29 rather than 28 days. Obviously, this affects the monthly total, whereas the daily average is hardly affected. The following example of monthly milk production per cow clearly illustrates the issue. Please note that the `monthdays()` command from `library(TSA)` facilitates the standardization markedly.

```
> library(TSA)
>
> ## Monthly totals
> plot(milk, xlab="Year", ylab="pounds", main="Monthly …")
> abline(v=1994:2006, col="grey", lty=3)
> lines(milk, lwd=1.5)
>
> ## Monthly average per day
> milk.adj <- milk/monthdays(milk)
> plot(milk.adj,xlab="Year",ylab="pounds", main="Average …")
> abline(v=1994:2006, col="grey", lty=3)
> lines(milk.adj, lwd=1.5)
```



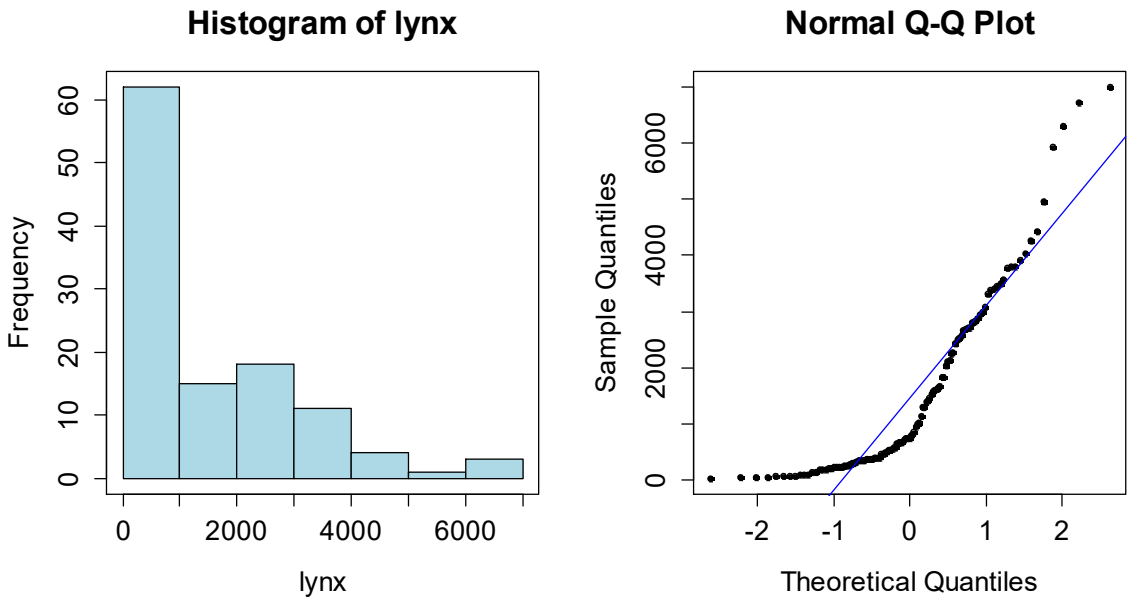


4.2.3 Log-Transformation

Many popular time series models and estimators (i.e. the usual ones for *mean*, *variance* and *correlation*) are based and most efficient in case of Gaussian distribution and additive, linear relations. However, data may exhibit different behavior. In such cases, we can often improve results by working with transformed values $g(x_1),...,g(x_n)$ rather than the original data $x_1,...,x_n$, The most popular and practically relevant transformation is $g(\cdot) = \log(\cdot)$. It is indicated if the variation in the series grows with the level, resp. if the series is on a relative scale where changes are better expressed in percent rather than in absolute values. This is another big advantage of the log-transformation: it is interpretable, i.e. the transformed values are the relative changes for the original values.

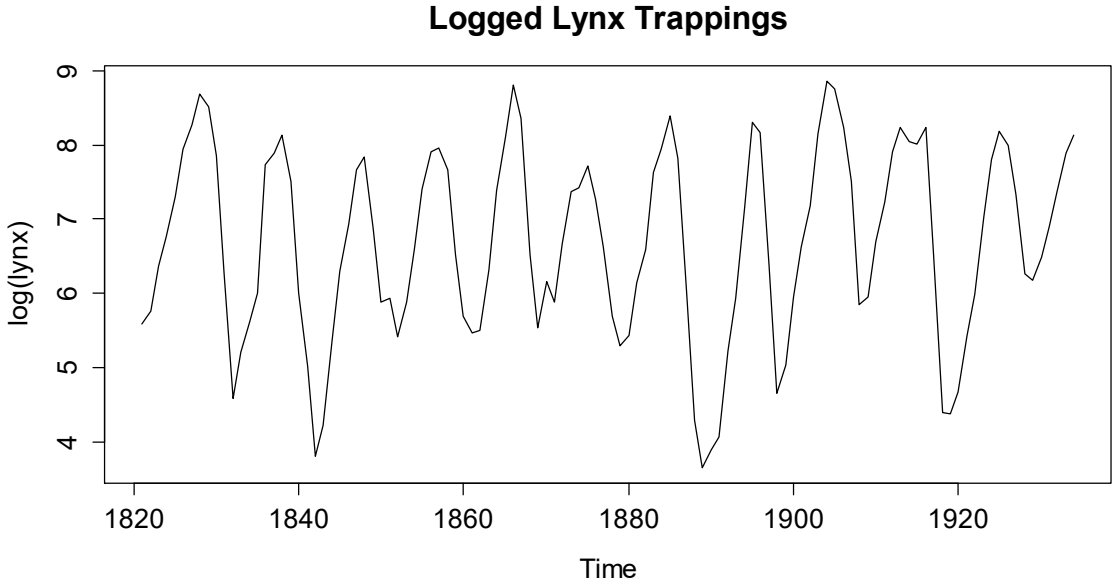
For time series where a log-transformation is beneficial, the marginal distribution is often (but not always!) right-skewed. Both properties are typical for time series which can take positive values only, such as the lynx trappings from section 1.2.2. It is easy to spot right-skewness by histograms and QQ-plots:

```
> hist(lynx, col="lightblue")
> qqnorm(lynx, pch=20); qqline(lynx, col="blue")
```



The lynx data are positive, on a relative scale and strongly right-skewed. Hence, a log-transformation proves beneficial. Implementing the transformation is easy in R:

```
> plot(log(lynx), main="Logged Lynx Trappings")
```



The data now follow a more symmetrical pattern; the extreme upward spikes are all gone. Another major advantage of the log-transformation is that model-based fitted values, forecasts and prediction intervals will not take negative values. Often, this is a must for series which are strictly positive. However, an eye has to be had in the back-transformation to the original scale. If only the simple $\exp(\cdot)$ is used, the back-transformed point forecast will not be the mean, but only the median of the forecast distribution. Fundamentally, the median may be a very reasonable summary statistic for a skewed distribution. Nevertheless, there are applications where unbiased predictions are a must, in which case a corrected back-transformation has to be applied. It is given by:

$$\exp(\hat{x}_t)\cdot\left(1+\frac{\hat{\sigma}_h^2}{2}\right), \text{ with } \hat{\sigma}_h^2 = \text{ estimated } h\text{-step forecast variance}$$

Obviously, the bigger the forecast variance is, the more pronounced the difference between median and mean in the forecast distribution will be.

4.2.4 Box-Cox and Power Transformations

Another type of transformations sometimes used are power transformations which are of the form $g(x_t)=x_t^p$. The most popular instance is perhaps the square-root transformation with $p=1/2$, which has some merit with count data. It's effect is similar to the one of the log-transformation, i.e. it stabilizes the variation of the series if that increases with the level. The drawback however is that the transformed values lack a direct interpretation – they are not relative changes as with the log, but just values on a different scale. The family of power transformations can be enhanced by the so-called Box-Cox transformation

$$g(x_t)=\frac{x_t^\lambda-1}{\lambda} \text{ with } \lambda\neq 0\,.$$

Please note that the (non-allowed) case of $\lambda=0$ corresponds to the (natural, i.e. base e) log-transformation discussed above. Again, Box-Cox transformed values lack a direct interpretation, but the method is of importance as many of the (to be presented) functions in `library(forecast)` allow for estimating λ. In fact, there is also the stand-alone function `BoxCox.lambda()` which allows for determining the most suitable transformation, i.e.:

```
> BoxCox.lambda(lynx)
[1] 0.1521849
```

The value turns out to be 0.15>0, indicating that a power transformation may be preferable to the log. On the other hand, we so lose the interpretability of the log, potentially without much practical benefit. We thus recommend favoring the log over a Box-Cox transformation for small λ's (i.e. smaller than $\approx\pm 0.3$). Likewise, we can often without any transformation at all if λ is estimated close to one.

If a Box-Cox transformation has been used, the issue of biased fitted values and point forecasts appears again. For obtaining the mean of the forecast distribution, a correction is needed:

$$(\lambda\hat{x}_t+1)^{1/\lambda}\cdot\left(1+\frac{\hat{\sigma}_h^2(1-\lambda)}{2(\lambda\hat{x}_t+1)^2}\right)$$

This formula looks quite complicated. Fortunately, if using the `forecast()` methods from `library(forecast)`, it is already implemented. Corrected, bias-free forecasts can be obtained by simply setting the argument `unbiased=TRUE`. We emphasize again that it is not obligatory to do so: in case it is not used, the point forecast will be the value where the realized value lies above resp. below with 50% probability each. If a corrected point forecast is given, it is the mean of all realized values – in case of a skewed distribution, this is not necessarily the better representant of what's going to happen.

4.3 Decomposition

4.3.1 The Basics

We have learned in section 2.2 that stationarity is an important prerequisite for being able to statistically learn from time series data. However, many of the example series exhibit either trend and/or seasonal effect, and thus are non-stationary. In this section, we will learn how to deal with that. First, we need to define what trend and seasonality mean.

Trend

A deterministic trend in a time series is a long-term change in the mean, induced by external factors. Typical examples among the series we have seen so far include the Air Passenger, Australian Production, Maine Unemployment and SMI data. The Lynx data on the other hand are considered to be without a trend. The obvious fluctuations are attributed to a random cyclic component.

Seasonal Effect

A seasonal component is a deterministic cyclic component in a time series with a fixed and known frequency, often caused by the way the measurements are obtained. The most typical case is the seasonal effect in monthly data where the measurement period comprises of multiple years, this coined the term. Typical examples include the Air Passenger, Australian Production and Maine Unemployment data. Seasonal components can also be present in e.g. hourly data that were observed over several days, it is a "daily pattern" in this case. On the other hand, the Lynx data are (as most yearly data) non-seasonal. They do have a random cyclic component, but it is not a seasonal effect!

The standard model describes a time series X_t as an *additive composition* of a (potentially absent) deterministic *trend* component m_t, a (potentially absent) *seasonal effect* s_t and a *stationary remainder* term R_t. Hence,

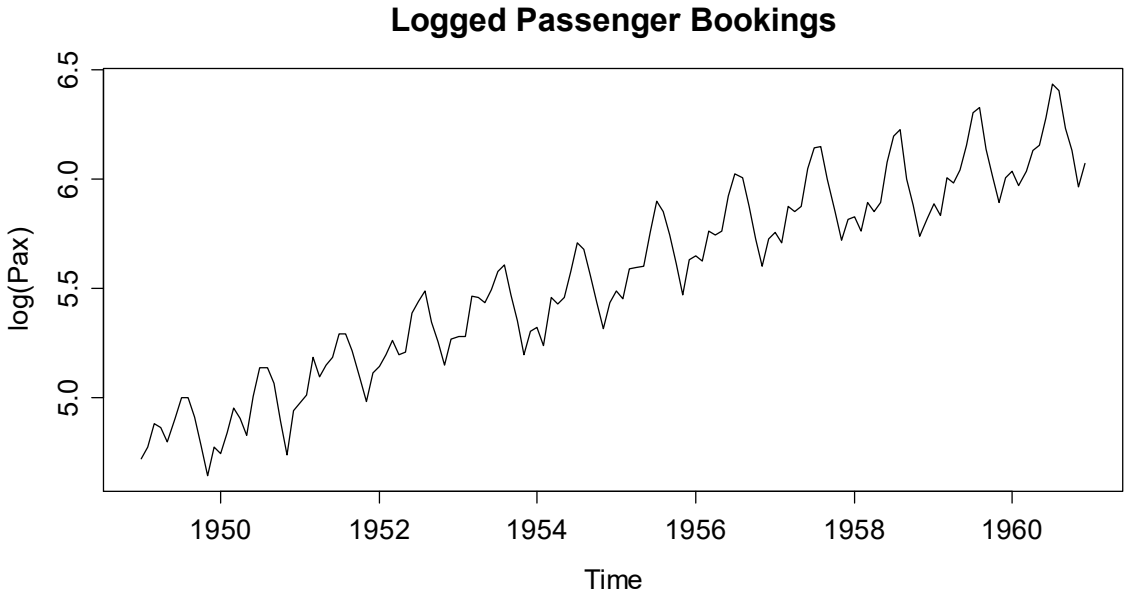
$$X_t = m_t + s_t + R_t,$$

where X_t is the time series process at time t, m_t is the trend, s_t is the seasonal effect, and R_t is the remainder, i.e. a sequence of usually correlated random variables with mean zero. In practice however, many time series exhibit an increase in seasonal and random variation with the (trend) level. This is the case in all seasonal series presented in this script, i.e. Air Passenger, Australian Production and Maine Unemployment data. For making the additive decomposition model a valid choice, the data need to be transformed with either a Box-Cox resp. power transformation, or much more often, the logarithm. Simple math demonstrates that an additive decomposition of a logged series means a multiplicative decomposition on the original scale.

$$\log(X_t) = \log(m_t \cdot s_t \cdot R_t) = \log(m_t) + \log(s_t) + \log(R_t) = m_t' + s_t' + R_t'$$

For illustration, we carry out a log-transformation on the air passenger bookings:

```
> plot(log(AirPassengers), ylab="log(Pax)", main=...)
```

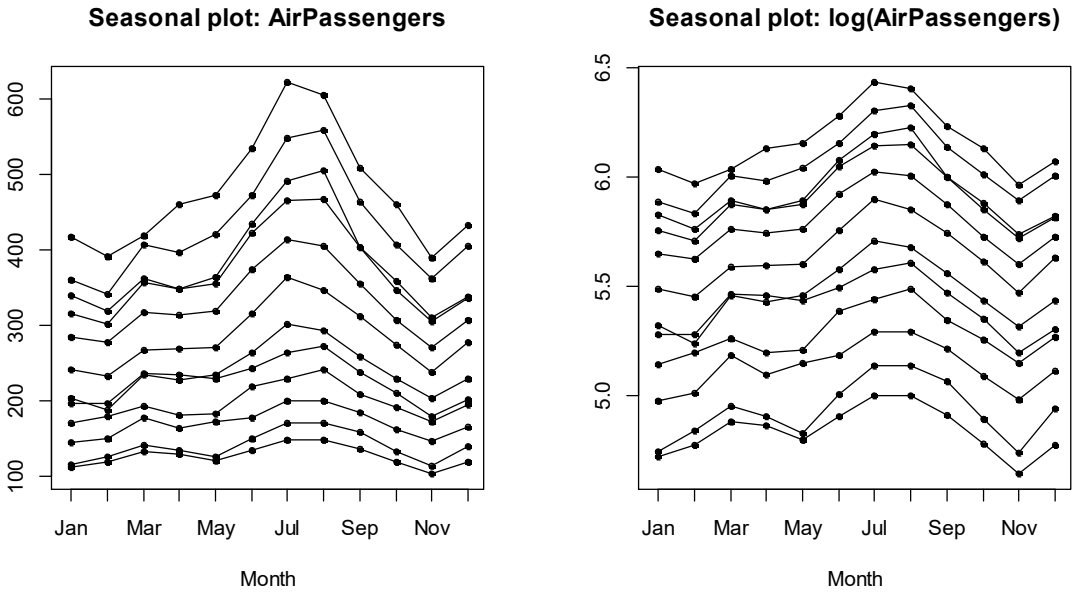


The plot shows that indeed, the magnitude of seasonal effect and random variation now seem to be less dependent of the level of the series than it was initially. Thus, the multiplicative model is much more appropriate for the Air Passenger data than the additive one. Alternatively, we could also estimate a Box-Cox transformation:

```
> BoxCox.lambda(AirPassengers)
[1] -0.2947156
```

The value is so close to zero that we prefer to work with the easier-to-interpret logarithm. Please note that if using any other Box-Cox transformation than the logarithm, an additive decomposition would be estimated on the transformed scale, but the original data do not follow a multiplicative decomposition model. Besides the time series plot of original and transformed data and the `BoxCox.lambda()` value, further evidence for a transformation can be found in the seasonal plots.

```
> seasonplot(AirPassengers, pch=20)
> seasonplot(log(AirPassengers), pch=20)
```



The left one on the untransformed data clearly shows that the difference between summer and winter is larger in the later years when the passenger figures are higher. After the log-transformation, the magnitude of the seasonal differences are more or less constant, though. However, a further snag is that the seasonal effect seems to alter over time rather than being constant. In earlier years, a prominent secondary peak in March is apparent. Over time, this erodes away, but on the other hand, the summer peak seems to be ever rising. The issue of how to deal with evolving seasonal effects will be addressed later in chapter 4.3.4.

4.3.2 Differencing

A simple approach for *removing* deterministic trends and/or seasonal effects from a time series is by taking differences. A practical interpretation of taking differences is that by doing so, the changes in the data will be monitored, but no longer the series itself. While this is conceptually simple and quick to implement, the main disadvantage is that it does not result in explicit estimates of the trend component m_t, the seasonal component s_t nor the remainder R_t. Hence, it does not really serve for a decomposition of a series X_t, but the approach has its merits, especially for the class of SARIMA models, presented in chapter 6.

We will first turn our attention to series with an additive trend, but without seasonal variation. By taking first-order differences with lag 1, and assuming a trend with little short-term changes, i.e. $m_t \approx m_{t-1}$, we have:

$$\begin{aligned} X_t &= m_t + R_t \\ Y_t &= X_t - X_{t-1} \approx R_t - R_{t-1} \end{aligned}$$

In practice, this kind of differencing approach "mostly works", i.e. manages to reduce presence of a trend in the series in a satisfactory manner. However, the trend is only fully removed if it is exactly linear, i.e. $m_t = \alpha + \beta t$. Then, we obtain:

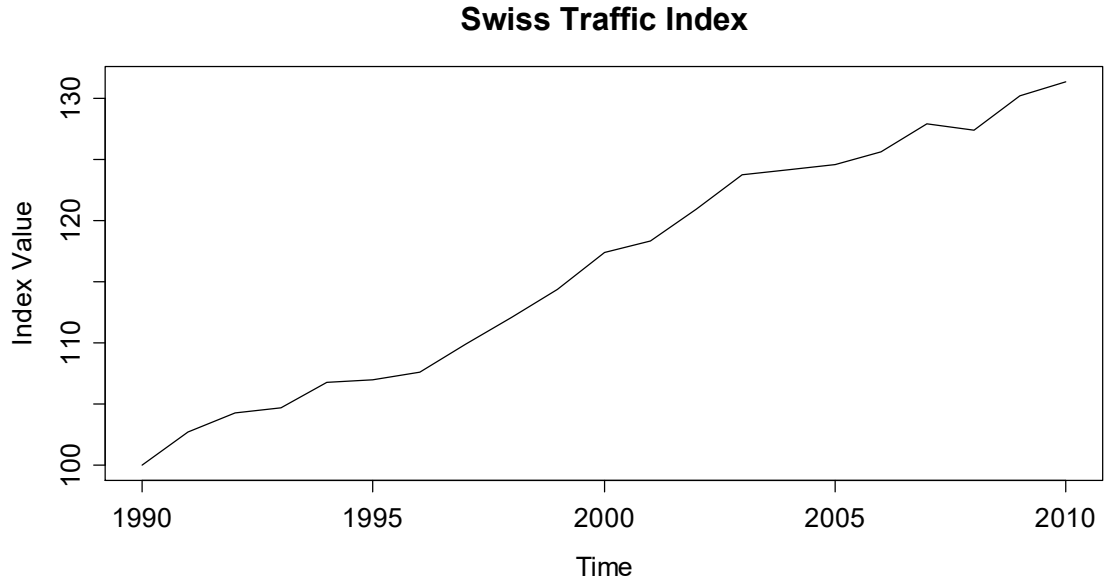
$$Y_t = X_t - X_{t-1} = \beta + R_t - R_{t-1}$$

Another somewhat disturbing property of the differencing approach is that strong, artificial new dependencies are created, meaning that the autocorrelation in Y_t is different from the one in R_t. For illustration, consider a stochastically independent remainder R_t: the differenced process Y_t has autocorrelation!

$$\begin{aligned} Cov(Y_t, Y_{t-1}) &\approx Cov(R_t - R_{t-1}, R_{t-1} - R_{t-2}) \\ &= -Cov(R_{t-1}, R_{t-1}) \\ &\neq 0 \end{aligned}$$

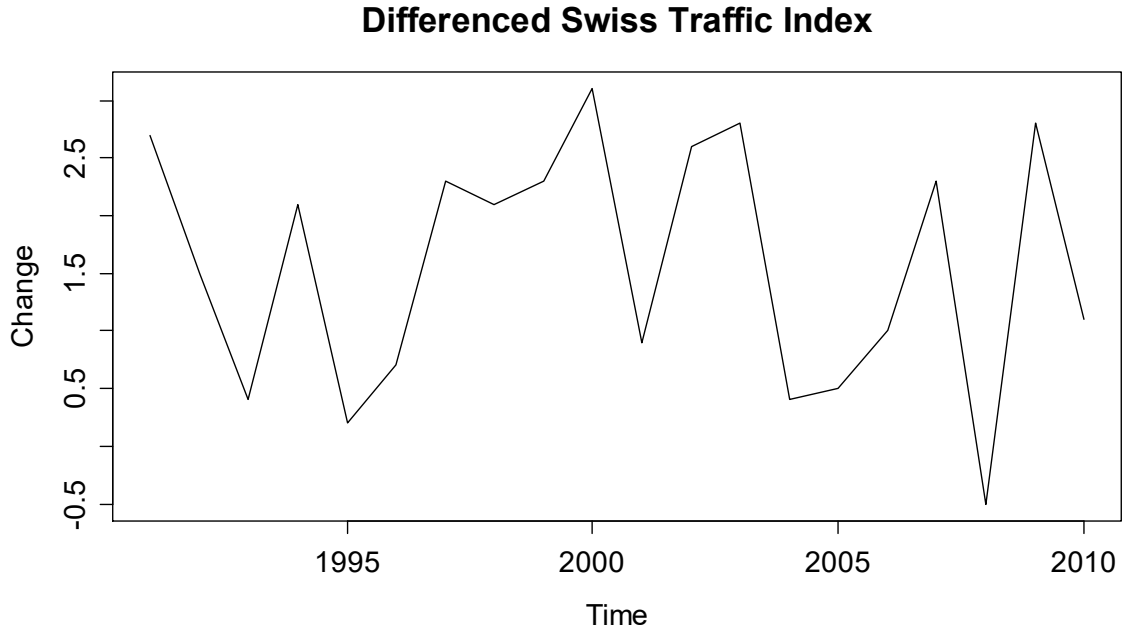
We illustrate how differencing works by using a dataset that shows the traffic development on Swiss roads. The data are available from the federal road office (ASTRA) and show the indexed traffic amount from 1990-2010. We type in the values and plot the original series:

```
> SwissTraffic <- ts(c(100.0, 102.7, 104.2, 104.6, 106.7,
                       106.9, 107.6, 109.9, 112.0, 114.3,
                       117.4, 118.3, 120.9, 123.7, 124.1,
                       124.6, 125.6, 127.9, 127.4, 130.2,
                       131.3), start=1990, freq=1)
>
> plot(SwissTraffic)
```



There is a clear trend, which is close to linear, thus the simple approach should work well here. Taking first-order differences with lag 1 shows the yearly changes in the Swiss Traffic Index, which must now be a stationary series. In **R**, the job is done with function `diff()`.

```
> diff(SwissTraffic)
Time Series:
Start = 1991
End = 2010
Frequency = 1
 [1]  2.7  1.5  0.4  2.1  0.2  0.7  2.3  2.1  2.3  3.1
[11]  0.9  2.6  2.8  0.4  0.5  1.0  2.3 -0.5  2.8  1.1
```



Please note that the time series of differences is now 1 instance shorter than the original series. The reason is that for the first year, 1990, there is no difference to the previous year available. The differenced series now seems to have a constant mean, i.e. the trend was successfully removed.

Log-Transformation and Differencing

On a sidenote, we consider a series that was log-transformed first, before first-order differences with lag 1 were taken. An example is the SMI data that were shown in section 1.2.4. The result is the so-called *log return*, which is an approximation to the relative change, i.e. the percent in- or decrease with respect to the previous instance. In particular:

$$Y_t = \log(X_t) - \log(X_{t-1}) = \log\left(\frac{X_t}{X_{t-1}}\right) = \log\left(\frac{X_t - X_{t-1}}{X_{t-1}} + 1\right) \approx \frac{X_t - X_{t-1}}{X_{t-1}}$$

The approximation of the log return to the relative change is very good for small changes, and becomes a little less precise with larger values. For example, if we have a 0.00% relative change, then $Y_t = 0.00\%$, for 1.00% relative change we obtain $Y_t = 0.995\%$ and for 5.00%, $Y_t = 4.88\%$. We conclude with summarizing that for any non-stationary series which is also due to a log-transformation, the transformation is always carried out first, and then followed by the differencing!

The Backshift Operator

We here introduce the backshift operator B because it allows for convenient notation. When the operator B is applied to X_t it returns the instance at lag 1, i.e.

$$B(X_t) = X_{t-1}.$$

Less mathematically, we can also say that applying B means “go back one step”, or “increment the time series index t by -1”. The operation of taking first-order differences at lag 1 as above can be written using the backshift operator:

$$Y_t = (1 - B)X_t = X_t - X_{t-1}$$

However, the main aim of the backshift operator is to deal with more complicated forms of differencing, as will be explained below.

Higher-Order Differencing

We have seen that taking first-order differences is able to remove linear trends from time series. What has differencing to offer for polynomial trends, i.e. quadratic or cubic ones? We here demonstrate that it is possible to take higher order differences to remove also these, for example, in the case of a quadratic trend.

$$\begin{aligned} X_t &= \alpha + \beta_1 t + \beta_2 t^2 + R_t,\ R_t \text{ stationary} \\ Y_t &= (1-B)^2 X_t \\ &= (X_t - X_{t-1}) - (X_{t-1} - X_{t-2}) \\ &= R_t - 2R_{t-1} + R_{t-2} + 2\beta_2 \end{aligned}$$

We see that the operator $(1-B)^2$ means that after taking "normal" differences, the resulting series is again differenced "normally". This is a discretized variant of taking the second derivative, and thus it is not surprising that it manages to remove a quadratic trend from the data. As we can see, Y_t is an additive combination of the stationary R_t's terms, and thus itself stationary. Again, if R_t was an independent process, that would clearly not hold for Y_t, thus taking higher-order differences (strongly!) alters the dependency structure.

Moreover, the extension to cubic trends and even higher orders d is straightforward. We just use the $(1-B)^d$ operator applied to series X_t. In **R**, we can employ function `diff()`, but have to provide argument `differences=d` for indicating the order of the difference d. In practice, we can use R function `ndiffs()` for determining the appropriate order of differencing d.

Removing Seasonal Effects by Differencing

For time series with monthly measurements, seasonal effects are very common. Using an appropriate form of differencing, it is possible to remove these, as well as potential trends. We take first-order differences with lag p:

$$Y_t = (1-B^p)X_t = X_t - X_{t-p},$$

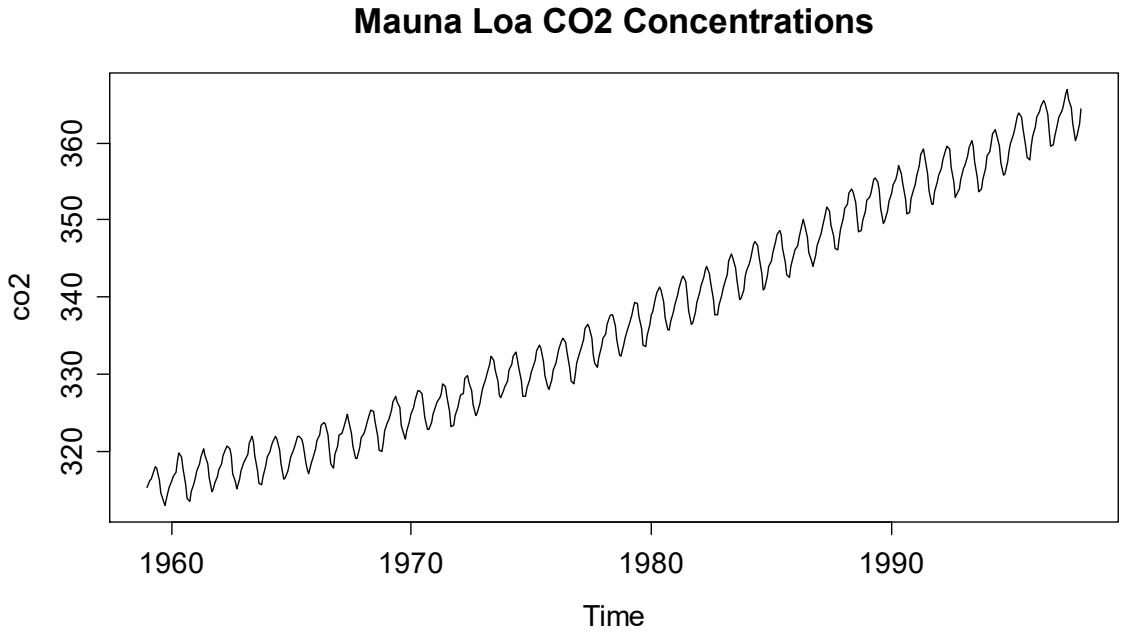
Here, p is the period of the seasonal effect, or in other words, the frequency of series, which is the number of measurements per time unit. The series Y_t then is made up of the changes compared to the previous period's value, e.g. the previous year's value. Also, from the definition, with the same argument as above, it is evident that not only the seasonal variation, but also a strictly linear trend will be removed.

Usually, trends are not exactly linear. We have seen that taking differences at lag 1 removes slowly evolving (non-linear) trends well due to $m_t \approx m_{t-1}$. However, here the relevant quantities are m_t and m_{t-p}, and especially if the period p is long, some trend will usually be remaining in the data. Then, further action is required.

Example

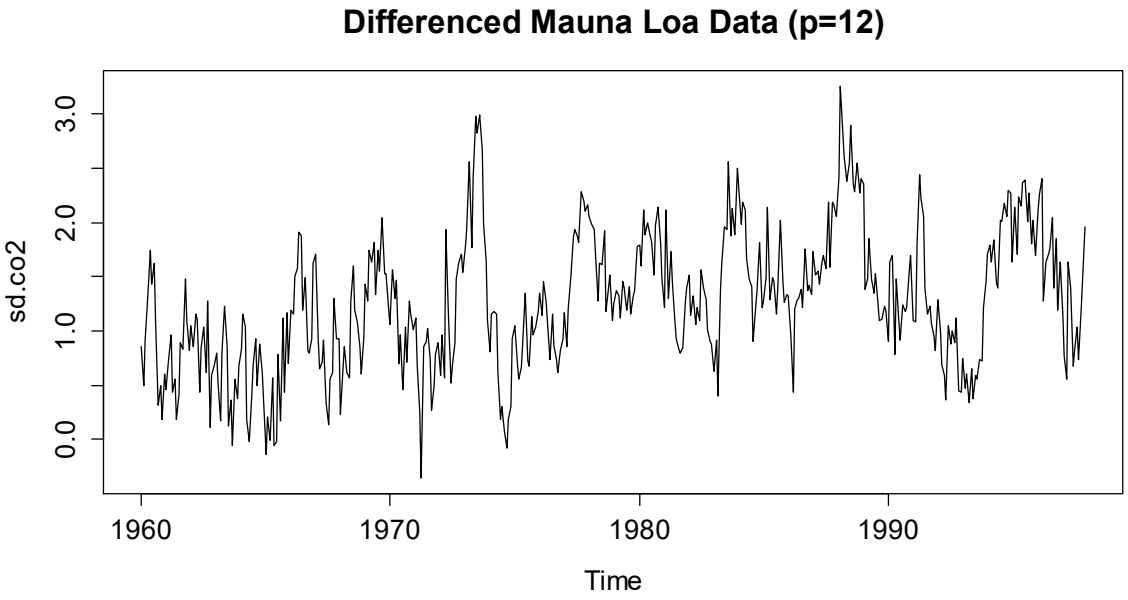
We are illustrating seasonal differencing using the Mauna Loa atmospheric CO_2 concentration data. This is a time series with monthly records from January 1959 to December 1997. It exhibits both a trend and a distinct seasonal pattern. We first load the data and do a time series plot:

```
> data(co2)
> plot(co2, main="Mauna Loa CO2 Concentrations")
```



Seasonal differencing is very conveniently available in **R**. We use function `diff()`, but have to set argument `lag=...`. For the Mauna Loa data with monthly measurements, the correct lag is 12. This results in the series shown on the next page. Because we are comparing every record with the one from the previous year, the resulting series is 12 observations shorter than the original one. It is pretty obvious that some trend is remaining and thus, the result from seasonal differencing cannot be considered as stationary. As the seasonal effect is gone, we could try to add some first-order differencing at lag 1.

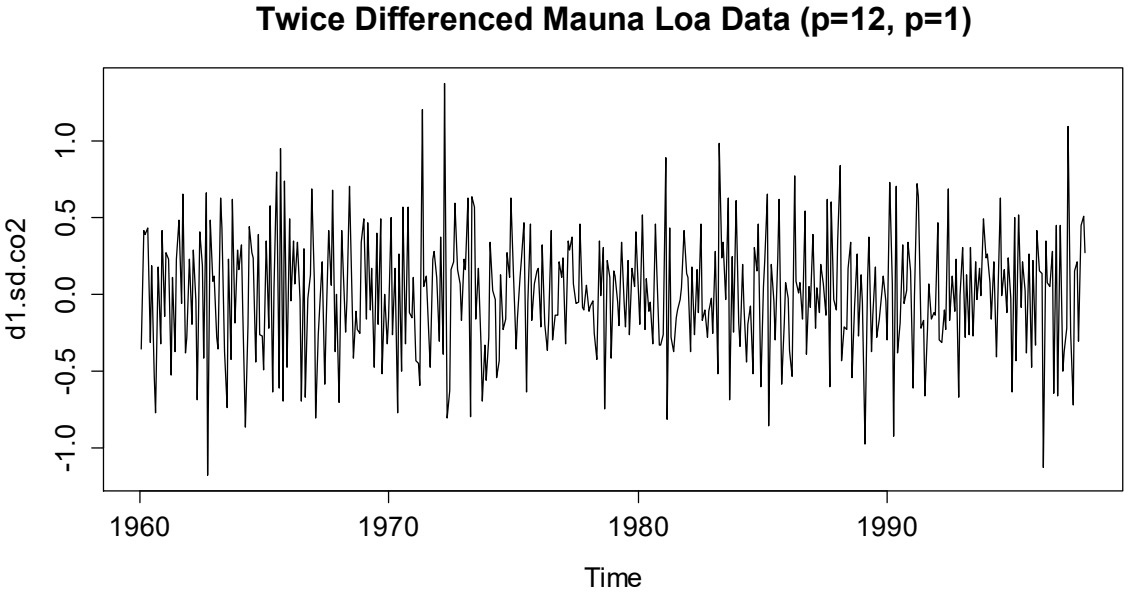
```
> sd.co2 <- diff(co2, lag=12)
> plot(sd.co2, main="Differenced Mauna Loa Data (p=12)")
```



The second differencing step indeed manages to produce a stationary series, as can be seen below. The equation for the final series is:

$$Z_t = (1-B)Y_t = (1-B)(1-B^{12})X_t .$$

The next step would be to analyze the autocorrelation of the series below and fit an $ARMA(p,q)$ model. Due to the two differencing steps, such constructs are also named $SARIMA$ models. They will be discussed in chapter 6.



We conclude this section by emphasizing that while differencing is quick and simple, and (correctly done) manages to remove any trend and/or seasonality, we do not obtain explicit estimates for trend m_t, seasonal effect s_t and remainder R_t which proves problematic in many applications.

4.3.3 Smoothing, Filtering

Our next goal is to define a decomposition procedure that yields explicit trend, seasonality and remainder estimates $\hat{m}_t$, $\hat{s}_t$ and $\hat{R}_t$. In the absence of a seasonal effect, the trend of a time series can simply be obtained by applying an *additive linear filter*:

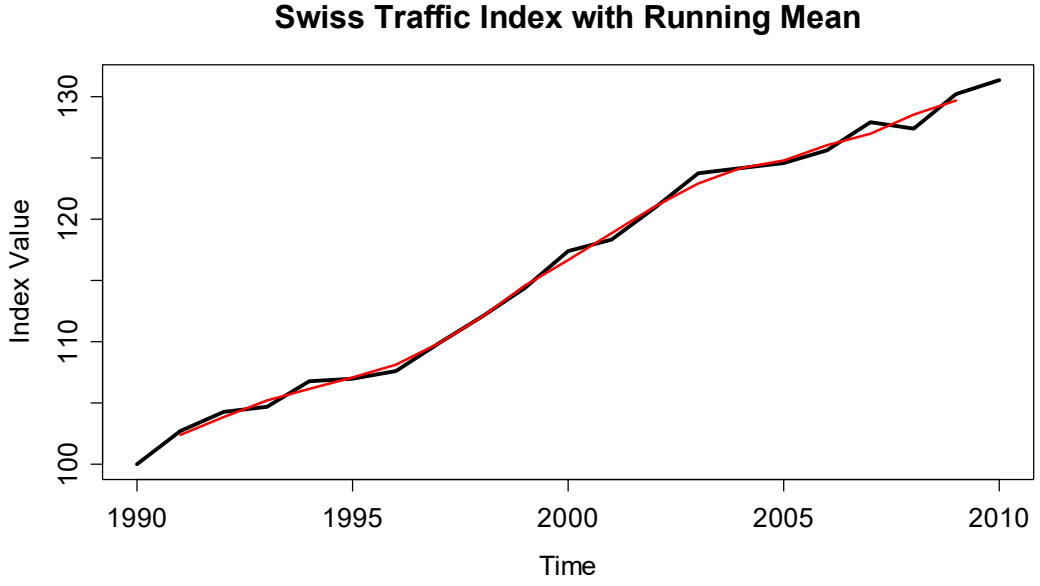
$$\hat{m}_t = \sum_{i=-p}^{q} a_i X_{t+i}$$

This definition is general, it allows for arbitrary weights and asymmetric windows. The most popular implementation is with $p = q$ and $a_i = 1/(2p+1)$, i.e. a *running mean* or *moving average estimator* with symmetric window and uniformly distributed weights. The window size is the smoothing parameter.

Example: Trend Estimation with Running Mean

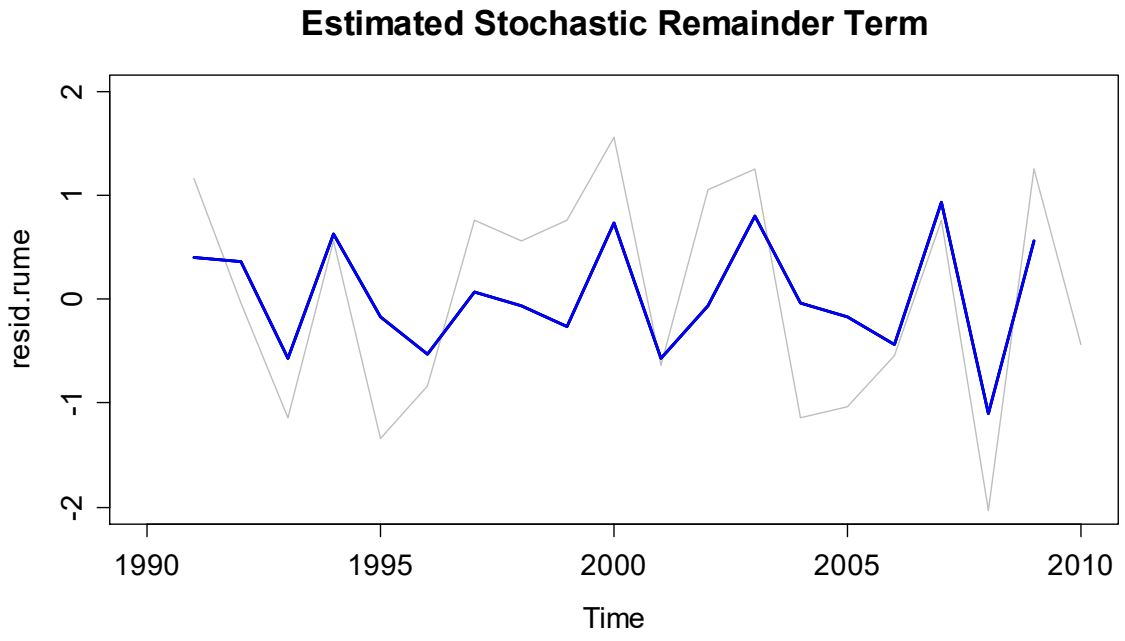
We here again consider the Swiss Traffic data that were already exhibited before. They show the indexed traffic development in Switzerland between 1990 and 2010. Linear filtering is available with function `filter()` from the base functionality in **R**., whereas for moving average computation, function `ma()` from `library(forecast)` is even more convenient.

```
> trend.est <- filter(SwissTraffic, filter=c(1,1,1)/3)
> trend.est <- ma(SwissTraffic, order=3)
```




```
> trend.est
Time Series: Start = 1990, End = 2010, Frequency = 1
 [1]       NA 102.3000 103.8333 105.1667 106.0667 107.0667
 [7] 108.1333 109.8333 112.0667 114.5667 116.6667 118.8667
[13] 120.9667 122.9000 124.1333 124.7667 126.0333 126.9667
[19] 128.5000 129.6333       NA
```

In our example, we chose the trend estimate to be the mean over three consecutive observations, resp. a 3-year moving average. This has the consequence that for both the first and the last instance of the time series, no trend estimate is available. We will later present more sophisticated methods that also allow for estimates near the endpoints. Furthermore, it is apparent that the Swiss Traffic series has a very strong trend signal, whereas the remaining stochastic term is comparably small in magnitude. We can now compare the estimated remainder term from the running mean trend estimation to the result from differencing:



The blue line is the remainder estimate from running mean approach, while the grey one resulted from differencing with lag 1. We observe that the latter has bigger variance; and, while there are some similarities between the two series, there are also some prominent differences – please note that while both seem stationary, they are different.

Trend Estimation for Seasonal Data

We now turn our attention to time series that show both trend *and* seasonal effect. The goal is to specify a filtering approach that allows trend estimation for periodic data. We still base this on the running mean idea, but have to make sure that we average over a full period. For monthly data, the formula is:

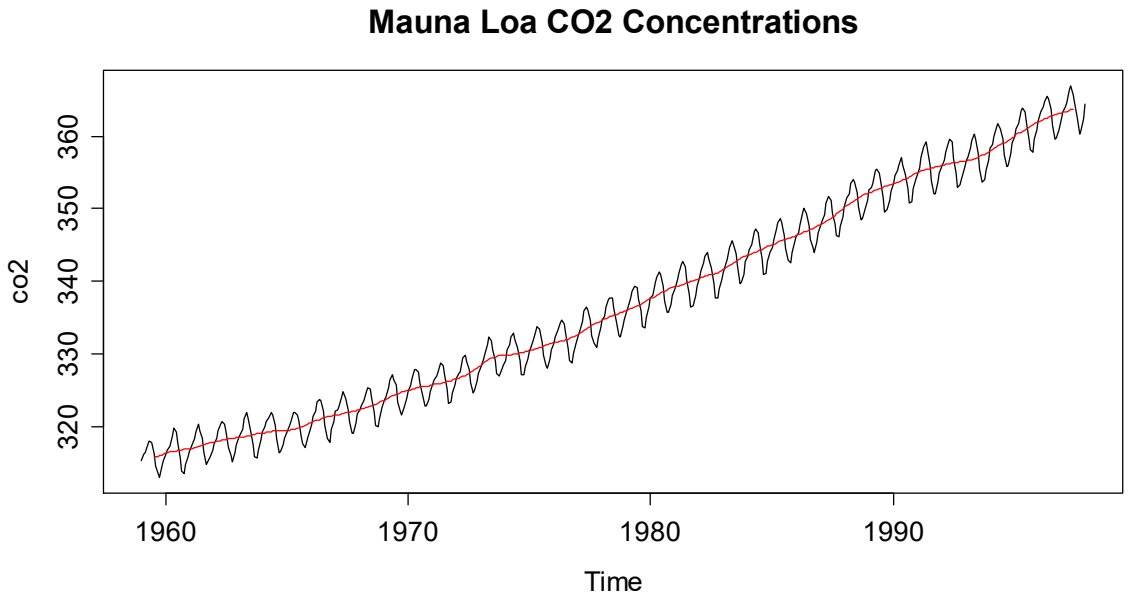
$$\hat{m}_t = \frac{1}{12}\left(\frac{1}{2}X_{t-6} + X_{t-5} + \ldots + X_{t+5} + \frac{1}{2}X_{t+6}\right), \text{ for } t = 7,\ldots,n-6$$

Be careful, as there is a slight snag if the frequency is even: if we estimate the trend for December, we use data from July to May, and then also add half of the value of the previous June, as well as half of the next June. This is required for having a window that is centered at the time we wish to estimate the trend. Using **R**'s function

`filter()`, with appropriate choice of weights, we can compute the seasonal running mean. Or we can use function `ma()` with argument `order=12` for the same task. We illustrate this with the Mauna Loa CO_2 data.

```
> wghts     <- c(.5,rep(1,11),.5)/12
> trend.est <- filter(co2, filter=wghts, sides=2)
> trend.est <- ma(co2, order=12, centre=TRUE)
> plot(co2, main="Mauna Loa CO2 Concentrations")
> lines(trend.est, col="red")
```

We obtain a trend which fits well to the data. It is not a linear trend, rather it seems to be slightly progressively increasing, and it has a few kinks, too.



We finish this section about trend estimation using linear filters by stating that other smoothing approaches, e.g. *running median estimation*, the *loess smoother* and many more are valid choices for trend estimation, too. In fact, several of them have clear advantages over simple movering average approaches.

Estimation of the Seasonal Effect

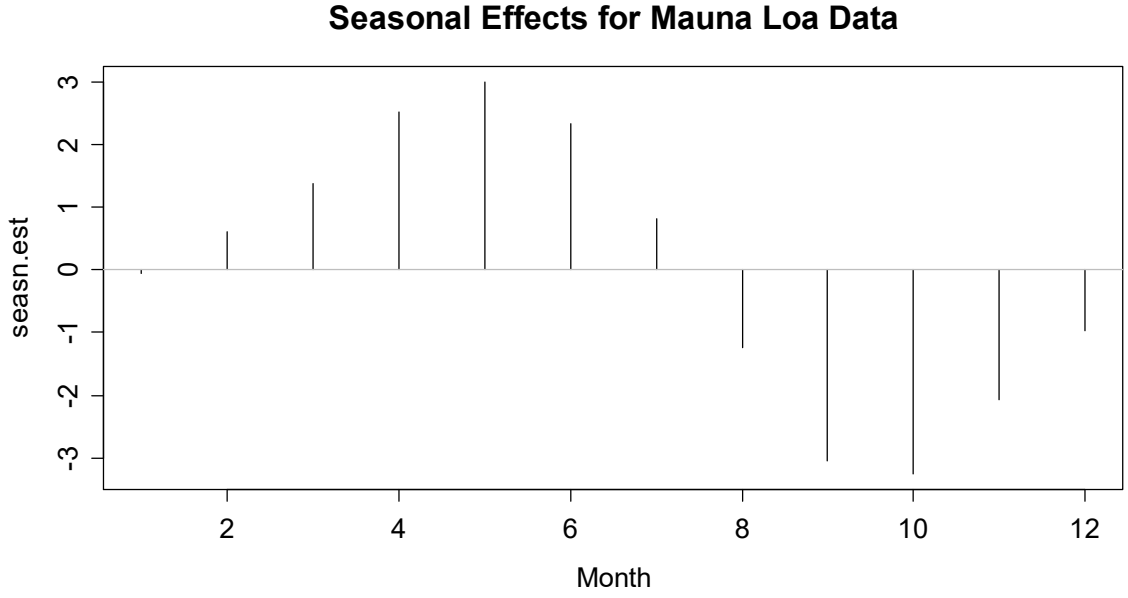
For fully decomposing periodic series such as the Mauna Loa data, we also need to estimate the seasonal effect. This is done on the basis of the trend adjusted data: simple averages over all observations from the same seasonal entity are taken. The following formula shows the January effect estimation for the Mauna Loa data, a monthly series which starts in January and has 39 years of data.

$$\hat{s}_{Jan} = \hat{s}_1 = \hat{s}_{13} = \ldots = \frac{1}{39} \cdot \sum_{j=0}^{38} (x_{12j+1} - \hat{m}_{12j+1})$$

In **R**, a convenient way of estimating such seasonal effects is by generating a factor for the months, and then using the `tapply()` function. Please note that the

seasonal running mean naturally generates NA values at the start and end of the series, which need be removed in the seasonal averaging process.

```
> trend.adj <- co2-trend.est
> month     <- factor(rep(1:12,39))
> seasn.est <- tapply(trend.adj, month, mean, na.rm=TRUE)
> plot(seasn.est, type="h", xlab="Month")
> title("Seasonal Effects for Mauna Loa Data")
> abline(h=0, col="grey")
```

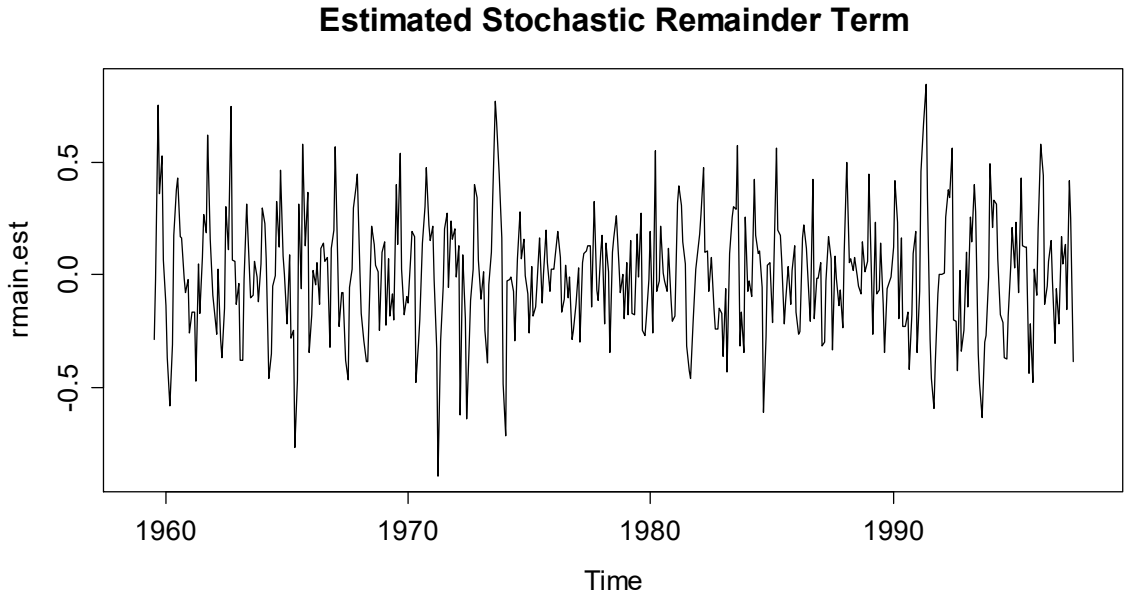


In the plot above, we observe that during a period, the highest values are usually observed in May, whereas the seasonal low is in October. The estimate for the remainder at time t is simply obtained by subtracting estimated trend and seasonality from the observed value.

$$\hat{R}_t = x_t - \hat{m}_t - \hat{s}_t$$

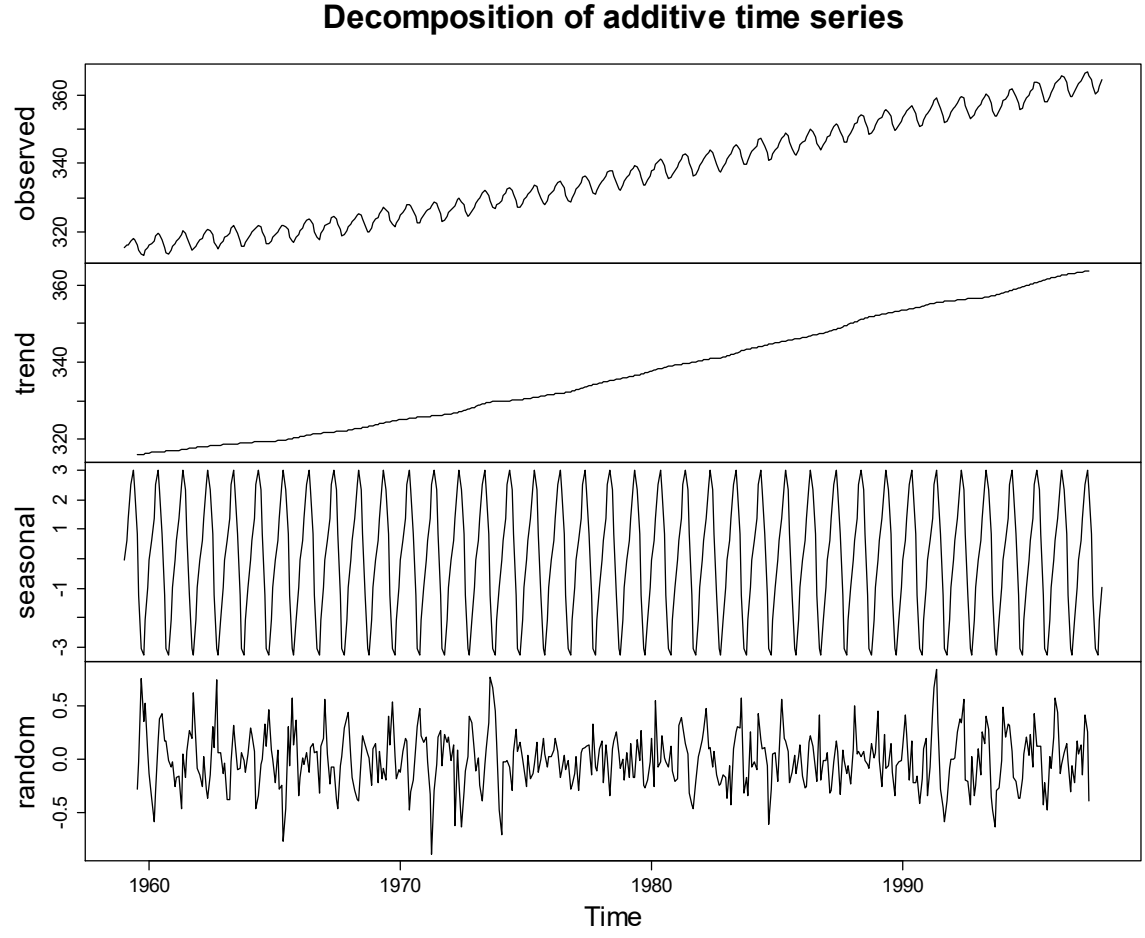
From the plot on the next page, it seems as if the estimated remainder still has some periodicity and thus it is questionable whether it is stationary. The periodicity is due to the fact that the seasonal effect is not constant but slowly evolving over time. In the beginning, we tend to overestimate it for most months, whereas in the end, we underestimate. We will address the issue on how to visualize evolving seasonality below in section 4.3.4 about STL-decomposition. A further option for dealing with non-constant seasonality is given by the exponential smoothing approach which is covered in chapter 8.

```
> rmain.est <- co2-trend.est-rep(seasn.est,39)
> plot(rmain.est, main="Estimated Stochastic Remainder Term")
```



Moreover, we would like to emphasize that R offers the convenient `decompose()` function for running mean estimation and seasonal averaging.

```
> co2.dec <- decompose(co2)
> plot(co2.dec)
```



Please note that `decompose()` only works with periodic series where at least two full periods were observed; else it is not mathematically feasible to estimate trend and seasonality from a series. The `decompose()` function also offers the neat plotting method shown above that generates the four frames above with the series, and the estimated trend, seasonality and remainder. Except for the different visualization, the results are exactly the same as what we had produced with our do-it-yourself approach.

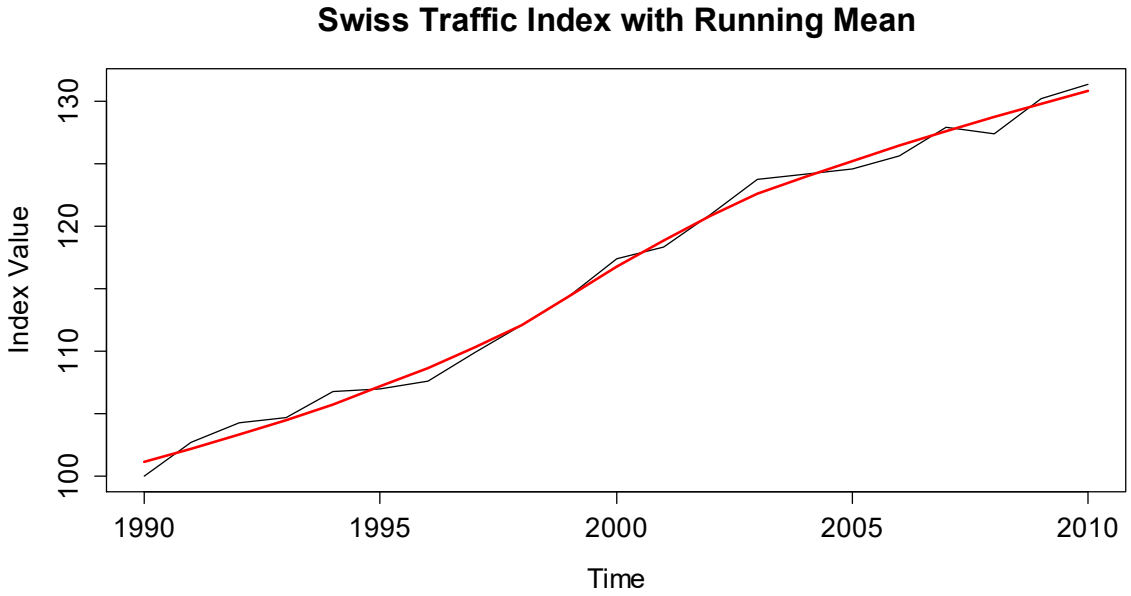
4.3.4 Seasonal-Trend Decomposition with LOESS

It is well known that the running mean resp. moving average is not the best smoother around. Thus, potential for improvement exists. While there is a dedicated R procedure for decomposing periodic series into trend, seasonal effect and remainder, we have to do some handwork in non-periodic cases.

Trend Estimation with LOESS

We here again consider the Swiss Traffic dataset, for which the trend had already been estimated above. Our goal is to re-estimate the trend with *LOESS*, a smoothing procedure that is based on local, weighted regression. The aim of the weighting scheme is to reduce potentially disturbing influence of outliers. Applying the LOESS smoother with (the often optimal) default settings is straightforward:

```
> fit   <- loess(SwissTraffic~time(SwissTraffic))
> trend <- predict(fit)
```



We observe that the estimated trend, in contrast to the running mean result, is now smooth and allows for interpolation within the observed time. Also, the `loess()` algorithm returns trend estimates which extend to the boundaries of the dataset. In summary, we recommend to always perform trend estimation with LOESS.

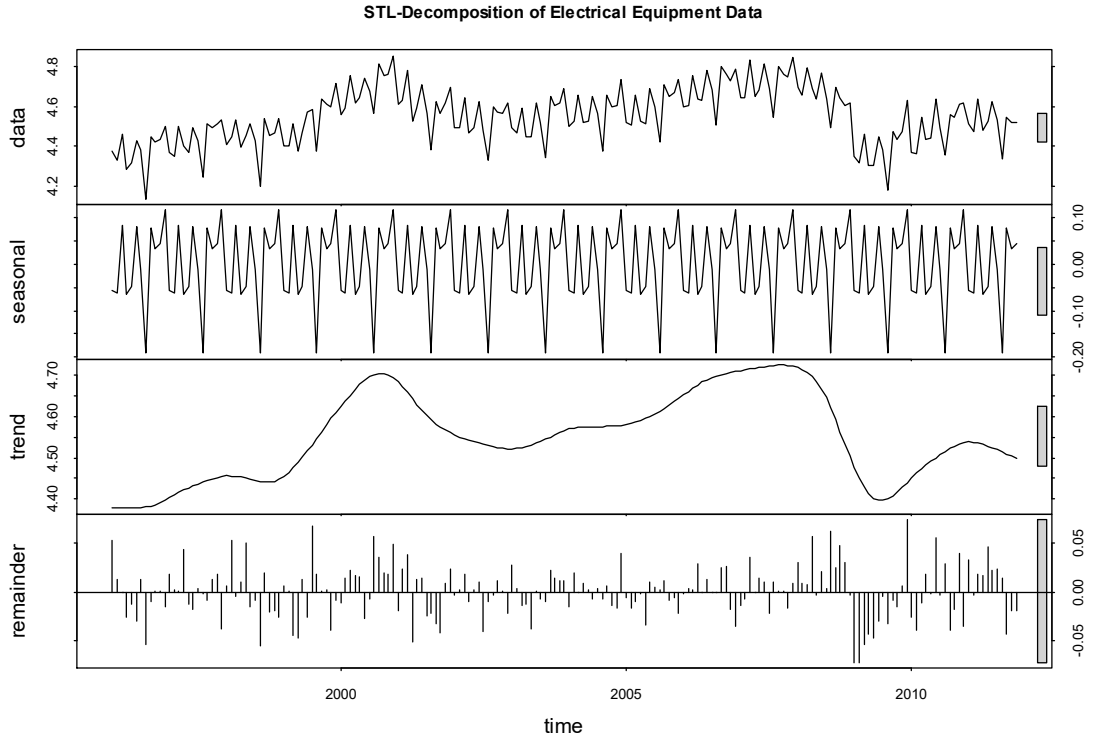
Using the `stl()` Procedure for Periodic Series

R's `stl()` procedure offers a versatile and robust decomposition of a periodic time series into trend, seasonality and remainder. All estimates are based on the LOESS smoother. STL has several advantages over the moving average decomposition from chapter 4.3.3. In particular, the seasonal component can remain stable over time, but it may also evolve with a rate of change that can be controlled by the user. Moreover, the user also has control over the smoothness of the trend. We do here without going into technical details about this iterative procedure, but focus on usage and interpretation. We illustrate with a time series on manufacturing of electrical equipment in the EU which can be found as `elecequip` in `library(fpp)`. It contains monthly indexed values from January 1996 to November 2011. The data are on a relative scale and also lambda estimation indicates that a log-transformation (i.e. multiplicative decomposition) is sensible.

```
> BoxCox.lambda(elecequip)
[1] 0.1822501
```

We then apply the `stl()` function with its default settings. For the `s.window` argument, there is no default value. We set it to `"periodic"` which mean assuming a seasonal pattern that remains unchanged over time.

```
> ee.stl <- stl(elecequip, s.window="periodic")
> plot(ee.stl, main="STL-Decomposition of Electrical …")
```



The graphical output is similar to the one from `decompose()` The grey bars on the right hand side facilitate interpretation of the decomposition: they show the relative magnitude of the effects, i.e. cover the same span on the y-scale in all of the frames. Hence, for the electrical equipment series, the trend contributes most the the

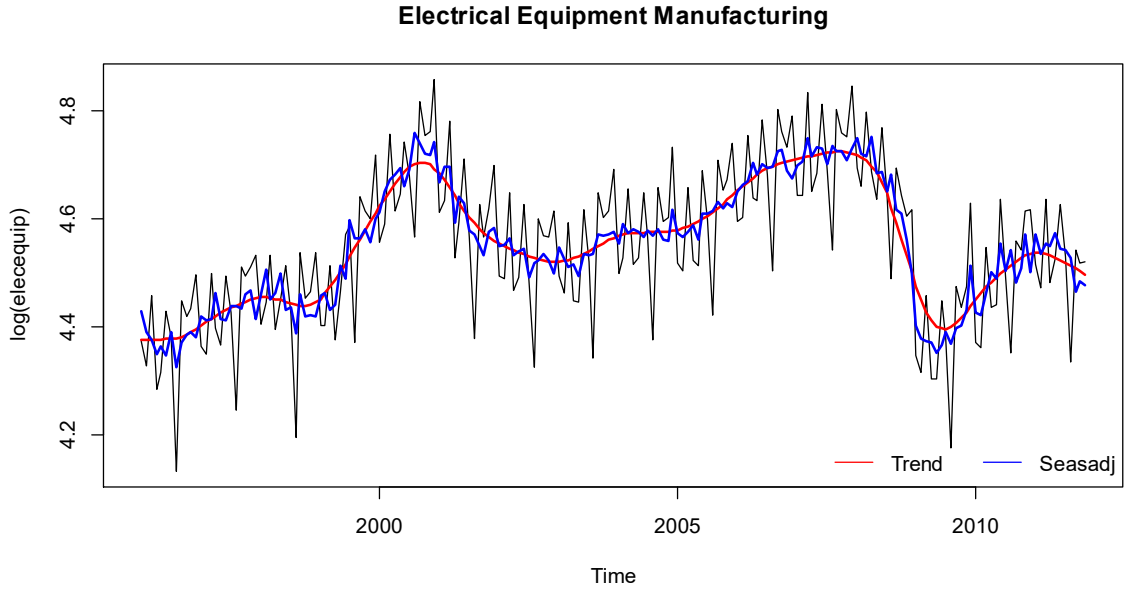
variation, followed by the seasonal effect and finally the remainder. The two principal arguments in function `stl()` are `t.window` and `s.window`: `t.window` controls the amount of smoothing for the trend, and has a default value which often yields good results. The value used can be inferred with:

```
> ee.stl$win[2]
 t
19
```

The result is the number of lags used as a window for trend extraction in LOESS. Increasing it means the trend becomes smoother; lowering it makes the trend rougher, but more adapted to the data. In our particular example, the trend already looks slightly wiggly, so it does not seem advisable to lower `t.window` further. On the other hand, the current decomposition seems unable to fully capture the sudden drops in years 2000 and 2009, because a couple of remainder terms before the drop are positive and some thereafter are negative. This is rooted somewhat in the fact that smoothers cannot cope well with sudden jumps in data, i.e. assume a continuous and smooth underlying function. Hence, the `elecequip` series manages to unveil the limits of the `stl()` procedure.

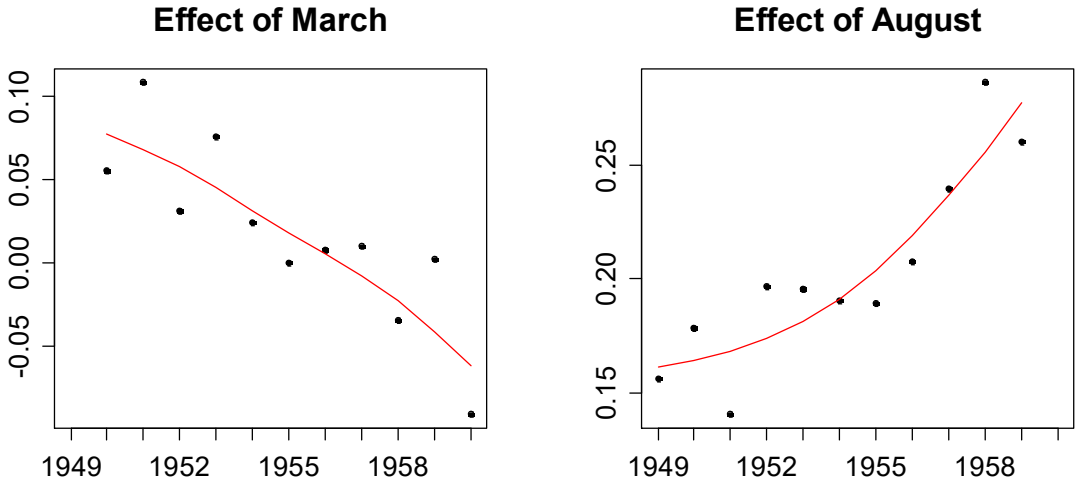
Once the decomposition is obtained, some functions can be useful: `seasonal()` will extract the seasonal component, `trendcycle()` yields the trend, `remainder()` undoubtedly outputs the remainder and finally `seasadj()` computes the seasonally adjusted series.

```
> plot(log(elecequip), main="Electrical Equipment …")
> lines(trendcycle(ee.stl), col="red", lwd=2)
> lines(seasadj(ee.stl), col="blue", lwd=2)
> legend("bottomright", c("Trend", "Seasadj"), col=…)
```



Evolving Seasonality

In some time series, the pattern of the seasonal effect evolves over time. To some extent, this is the case in the above electrical equipment data. The effect is more prominently visible and easier to explain in the air passenger series, to which we switch back here. With the `stl()` procedure, it is straightforward to obtain an estimate if we just set argument `s.window` to a numeric value which is the smoothing parameter. There is no default value and the optimal setting has to be determined exploratively from the data. As a starting value, `s.window=13` is often a good choice. For explaining the procedure, we here consider the logged air passenger data where the trend has been removed using a moving average. We here display all March and all August values of the trend-adjusted series and add a Loess smoother.

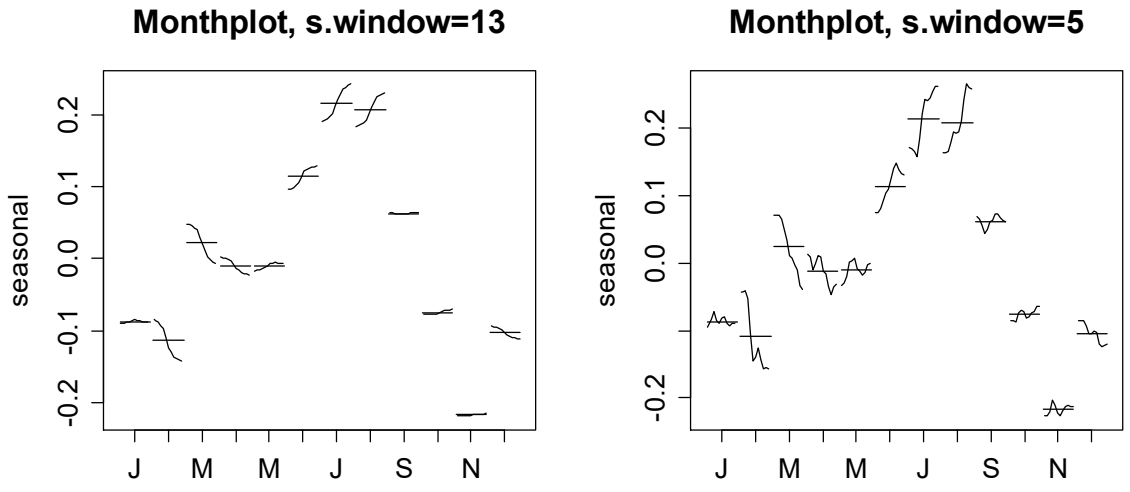


When assuming a non-evolving seasonal effect, the standard procedure would be to take the mean of the data points in each of the above scatterplots and declare that as the seasonal effect for March and August, respectively. This is a rather crude way of data analysis, and can of course be improved if the magnitude of the March and August effect develops as the smoothers suggest. Please note that the above plots and smoother estimation were presented for didactic purpose only. In practice, we can conveniently use the `stl()` procedure. We fit two decompositions with differing smoothing parameters.

```
> fit.05 <- stl(lap, s.window= 5)
> fit.13 <- stl(lap, s.window=13)
```

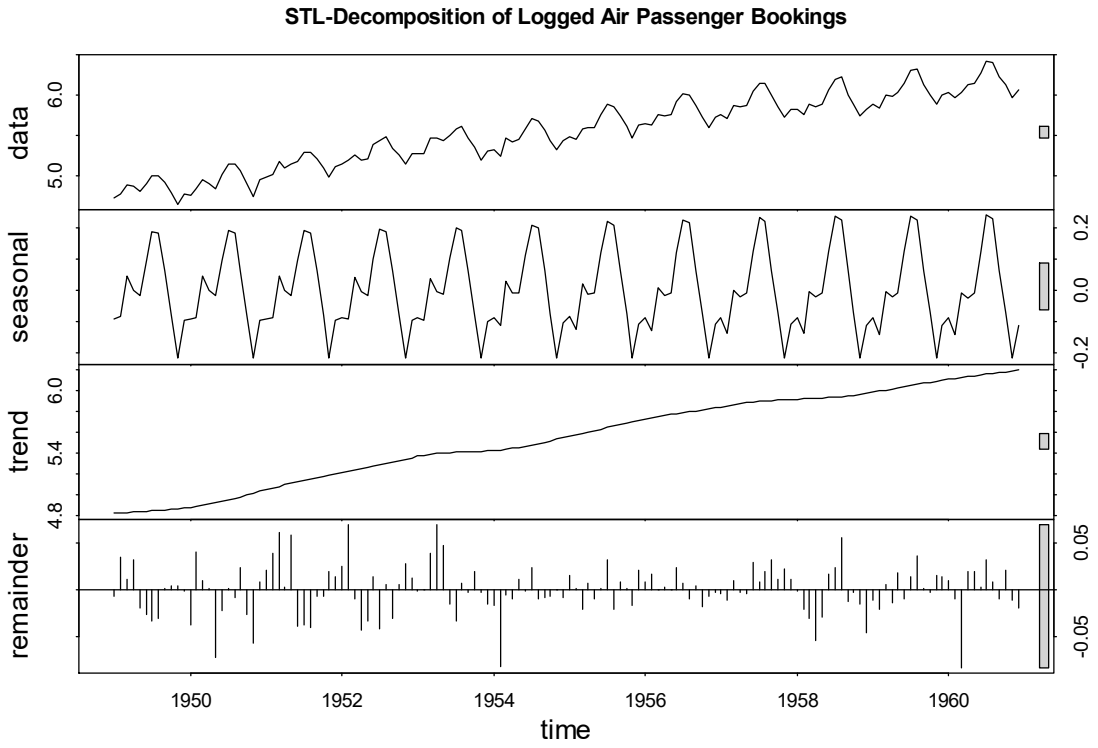
Please be reminded again that there is no default value for the seasonal span, and the optimal choice is left to the user upon visual inspection. An excellent means for doing so is the `monthplot()` function which shows the seasonal effects that were estimated by `stl()`.

```
> monthplot(fit.13, main="Monthplot, s.window=13")
> monthplot(fit.05, main="Monthplot, s.window=5")
```



The amount of smoothing seems appropriate in the left panel with `s.window=13`. However on the right, with smaller span, i.e. `s.window=5`, we observe overfitting: the seasonal effects do not evolve in a smooth way, and it means that this is not a good decomposition estimate. We finally display the decomposition with the chosen seasonal smoothing parameter.

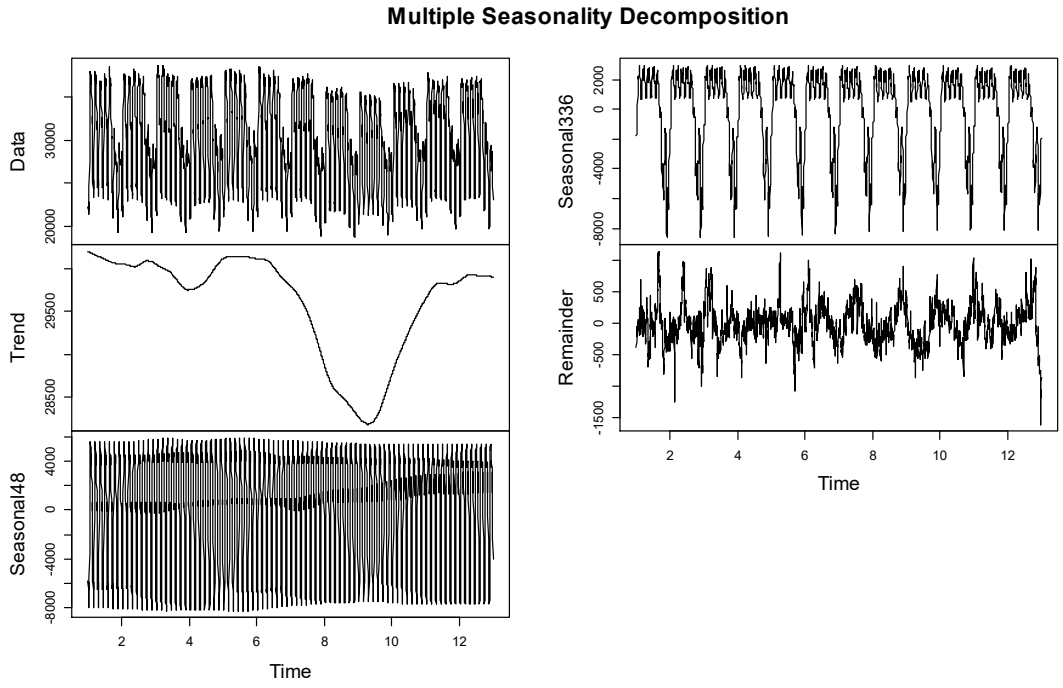
```
> plot(fit.13, main="STL-Decomposition ...")
```



Multiple Seasonalities

Some time series, e.g. with daily or intra-daily records, may exhibit various seasonalities. An example is series `taylor` from `library(forecast)` which shows half-hour electricity demands in England and Wales from Monday 5 June 2000 to Sunday 27 August 2000. It may show both a daily and a weekly pattern. This can conveniently be visualized by the fully automatic `mstl()` procedure from `library(forecast)`. It automatically detects all present seasonalities and determines estimates for the Box-Cox λ, `t.window` and `s.window` parameters.

```
> fit <- mstl(taylor)
> plot(fit, main="Multiple Seasonality Decomposition")
```



The output shows that two seasonal components with a frequency of 48 (i.e. a daily pattern for the half-hourly measurements) and 336 (a weekly pattern) were identified. Potentially, these data also feature a yearly pattern. But since the data were only observed over around 13 weeks, this cannot be identified and becomes part of the trend resp. remainder components (which are also displayed).

As always with fully automatic procedures, the results have to be critically evaluated and verified. It is possible to take control and alter all estimated values which may or may not be necessary. Moreover, the `mstl()` procedure does not yield the grey side bars and completely lacks information about the significance of the different components. For gaining further insight or also dealing with "seasonaltities" such as e.g. easter which change their position in the calendar, we better resort to the parametric modelling approach discussed below. There, we have access to all inference tools from regression and can fit even more sophisticated decompositions.

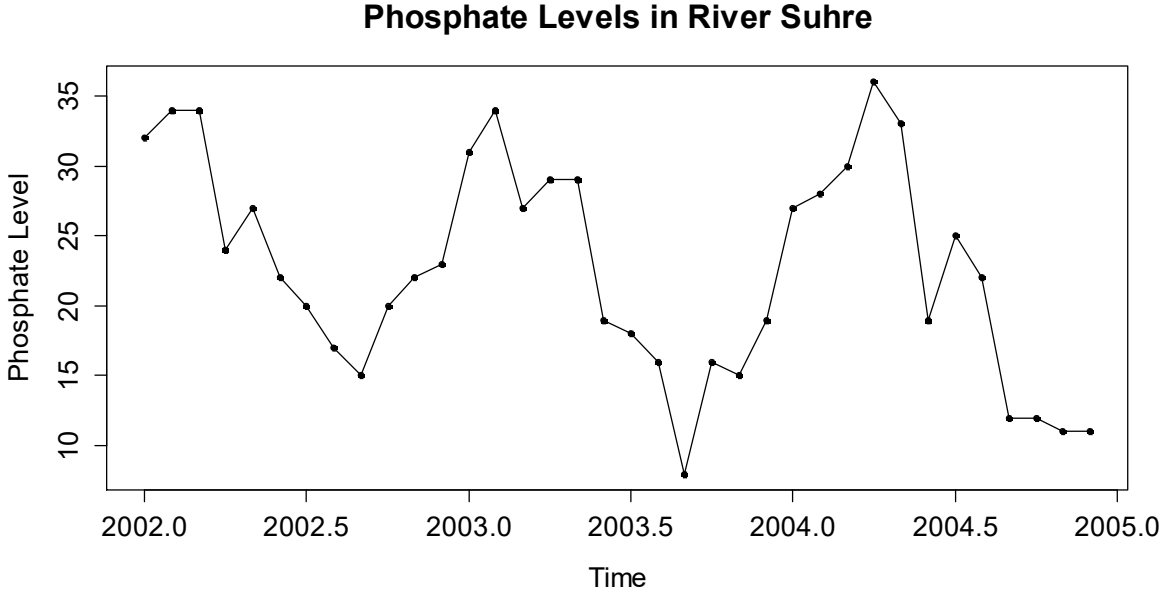
4.3.5 Parametric Modeling

A powerful approach for decomposing time series is parametric modeling. In fact, it allows for mimicking all the decomposition approaches that were discussed here previously. We will here focus on two examples. The first is a very short time series where a parsimonius parametric model is set up for accurate estimation of a trend, seasonal and remainder component. The strength of the parametric modeling approach in this example lies in the fact that we only need spending four degrees of freedom for estimating a full decomposition which is much less than the smoothing approaches presented above would require. Additionally, this regression based approach also allows for formal significance testing for the trend and the seasonal component which is often very valuable in explorative data analysis. Some prudence is required though due to the potentially correlated residuals in the linear models, an issue which will only be thoroughly discussed in chapter 7. In a second example, we use parametric modeling in a more flexible way with a smooth trend component and a dummy variable for the seasonal component, which yields results that are close to a STL decomposition or the smoothing approach implemented in R's `decompose()` procedure.

Parsimonius Decomposition of Phosphate Measurements

This example is an excerpt from a joint research project of the lecturer with Environmental Protection Office of the Swiss Canton Lucerne (UWE Luzern). Part of this project included the analysis of Phosphate levels in the river Suhre, which is an effluent of Lake Sempach. The time series with 36 monthly measurements over a period of 3 years is displayed below.

```
> plot(spt, type="o", ylab="Phosphate Level", pch=20)
> title("Phosphate Levels in River Suhre")
```



The time series features a prominent seasonal effect and potentially a slight downward trend. The aims in the project included a decomposition of the series, as well as statements whether there are trends and seasonalities in the various pollutants that were analyzed for a large number of rivers in the canton.

As the time series only has 36 observations and there seems to be a considerable amount of (weather, i.e. rainfall or draught induced) noise, using smoothing approaches or STL did not seem promising. These methods unavoidably spend many degrees of freedom, primarily due to simple averaging in the seasonal component. A way out is to set up a parametric decomposition model that is based on a linear trend in time plus a cyclic seasonal component and a remainder.

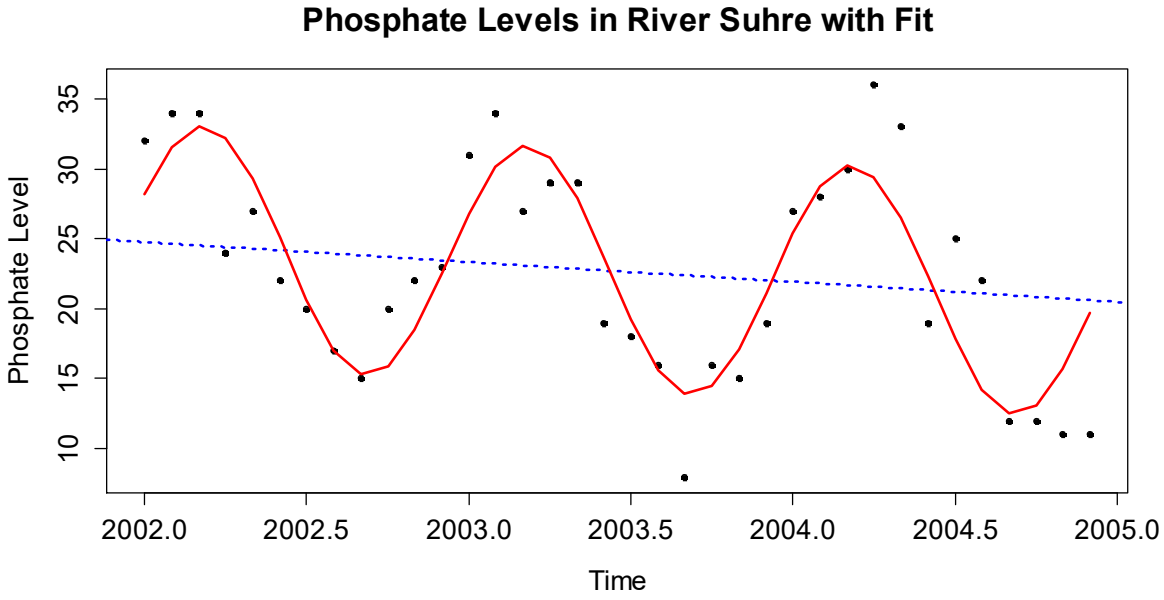
$$X_t = \beta_0 + \beta_1 \cdot t + \beta_2 \cdot \sin(2\pi t) + \beta_3 \cdot \cos(2\pi t) + R_t,\ t = 2002 + i/12 \text{ with } i = 1,...,36.$$

This model achieves a full trend/season/remainder decomposition with only four unknowns that can be estimated using the least squares approach, though using robust fitting methods might provide a very good alternative. We here provide the code for estimating the model in R.

```
> tnum  <- as.numeric(time(spt))
> fit   <- lm(spt ~ tnum + sin(2*pi*tnum) + cos(2*pi*tnum))
> cf    <- coef(fit); summary(fit)
Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)        2831.2732  1692.7075   1.673  0.10415
tnum                 -1.4019     0.8449  -1.659  0.10684
sin(2 * pi * tnum)    7.8420     1.0320   7.599 1.17e-08 ***
cos(2 * pi * tnum)    3.4357     1.0004   3.434  0.00166 **
---
Residual standard error: 4.234 on 32 degrees of freedom
Multiple R-squared: 0.7247, Adjusted R-squared:  0.6989
F-statistic: 28.08 on 3 and 32 DF,   p-value: 4.332e-09
```

The coefficients and inference results can be seen from the summary output, but we have to be careful with their interpretation. The error term in the linear model is a stationary, but potentially serially correlated time series R_t. If correlation exists, the assumptions for the least square algorithm are violated. Chapter 7 contains a full expositions of these topics, but in short summary, the coefficients would be unbiased though slightly inefficiently estimated, whereas the standard erros are biased and the derived p-values are not trustworthy. We plot the fit.

```
> plot(spt, type="p", pch=20, ylab="Phosphate Level")
> abline(cf[1], cf[2], col="blue", lty=3, lwd=2)
> lines(tnum, fitted(fit), col="red", lwd=2)
```



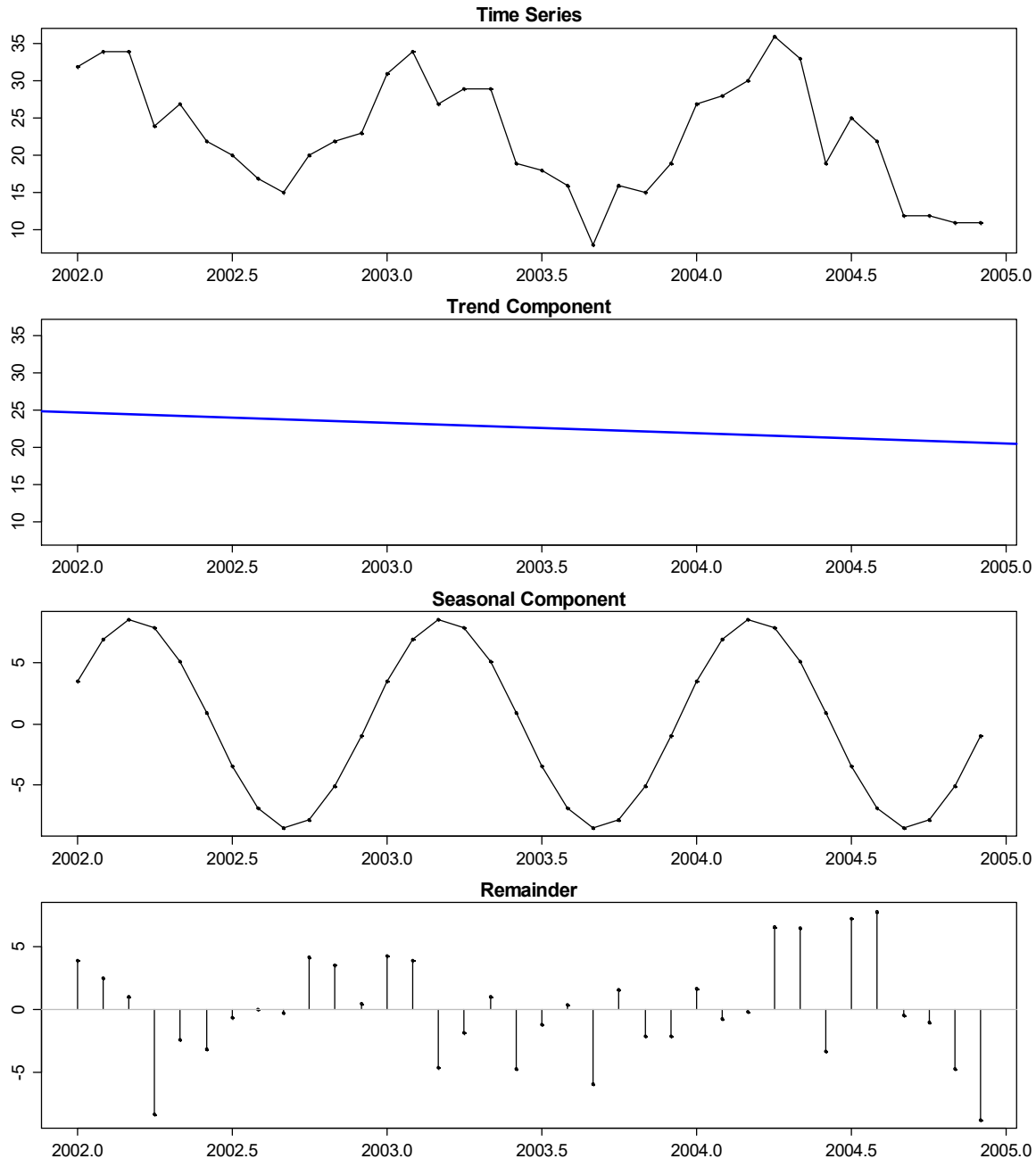
The red line shows the fitted values, i.e. the estimated average Phosphate levels. In blue, the trend function has been added. We can also provide a full decomposition plot as for example STL provides, but we have to construct it ourselves – see next page for the output.

```
> plot(spt, type="o", pch=20)
> plot(spt, type="n"); abline(cf[1], cf[2], col="blue")
> plot(tnum, cf[3]*sin(2*pi*tnum) + cf[4]*cos(2*pi*tnum),
       type="o", pch=20)
> plot(tnum, residuals(fit), type="h")
> points(tnum, residuals(fit), pch=20)
> abline(h=0, col="grey")
```

Despite the fact that a simple, linear trend function and a cyclic sine/cosine seasonality was used, the remainder seems like a stationary series with mean zero. There is no apparent serial correlation among the remainder terms, hence in this situation, we can even rely on the inference results. Please note that while the chosen model is fully adequate for the present situation, being so simplistic and parsimonius is not the correct strategy on all datasets. There is absolutely no need that a seasonal component is cyclic, with the Airline Pax being a prominent counterexample.

Flexible Decomposition of Maine Unemployment Data

We consider the Maine unemployment data from section 4.1.1. Our goal is to fit a smooth trend, along with a seasonal effect that is obtained from averaging. Sometimes, polynomial functions are used for modeling the trend function. However, we recommend to stay away from high-order polynomials due to their often very erratic behavior at the boundaries (cf. *Runge's Phenomenon*), so that anything beyond a quadratic trend should be avoided.



The way out lies in using a generalized additive model (GAM) with a flexible trend function. The seasonal effect is included as a factor variable. In mathematical notation, the model is:

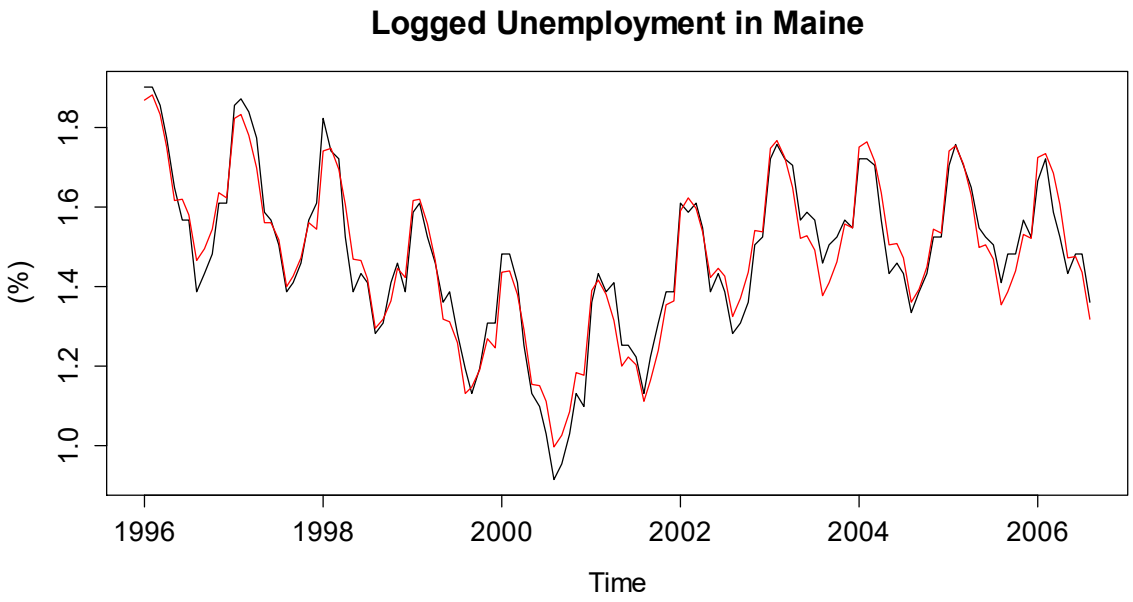
$$X_t = f(t) + \alpha_{i\langle t\rangle} + R_t, ,$$

where $t = 1,\ldots,128$ and $i\langle t\rangle \in \{1,\ldots,12\}$, i.e. $\alpha_{i\langle t\rangle}$ is a factor variable encoding for the month the observation was made in, see the **R** code below. Two questions immediately pop up, namely how to determine the smooth trend function $f(\cdot)$, and how fo fit the model as a whole. Both can conveniently be done using the `gam()` function from `library(mgcv)` in R. Please note that here, we model resp. decompose the logged Maine data, since their variation clearly increases with increasing level of the series.

```
> library(mgcv)
> tnum  <- as.numeric(time(maine))
> mm    <- rep(c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
+                "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"))
> mm    <- factor(rep(mm,11),levels=mm)[1:128]
> fit   <- gam(log(maine) ~ s(tnum) + mm)
```

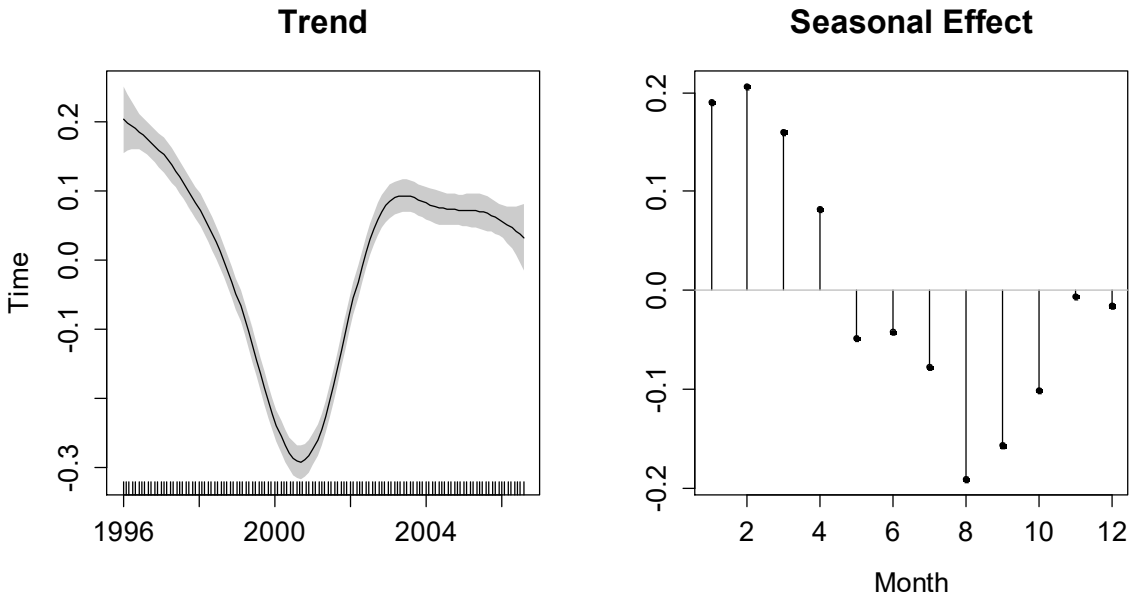
We do without displaying the summary output, because it is rather long and requires (as using this decomposition strategy) some knowledge in GAM. Let us just mention that the method decides to spend 8.196 degrees of freedom for the trend function, which corresponds to a polynomial of 8th grade (which however, would provide a much worse fit). The degrees of freedom are estimated internally using a cross validation approach and usually provide a sensible solution. It is possible to display the results graphically. It is normally very instructive to show the time series together with the fitted values. Furthermore, we also present the estimated trend function (via the partial residual plot obtained from function `plot.gam()`) plus the seasonal effect which is extracted from the dummy variable coeffcients.

```
> plot(log(maine), ylab="(%)", main="Logged Unemployment…")
> lines(tnum, fitted(fit), col="red")
```



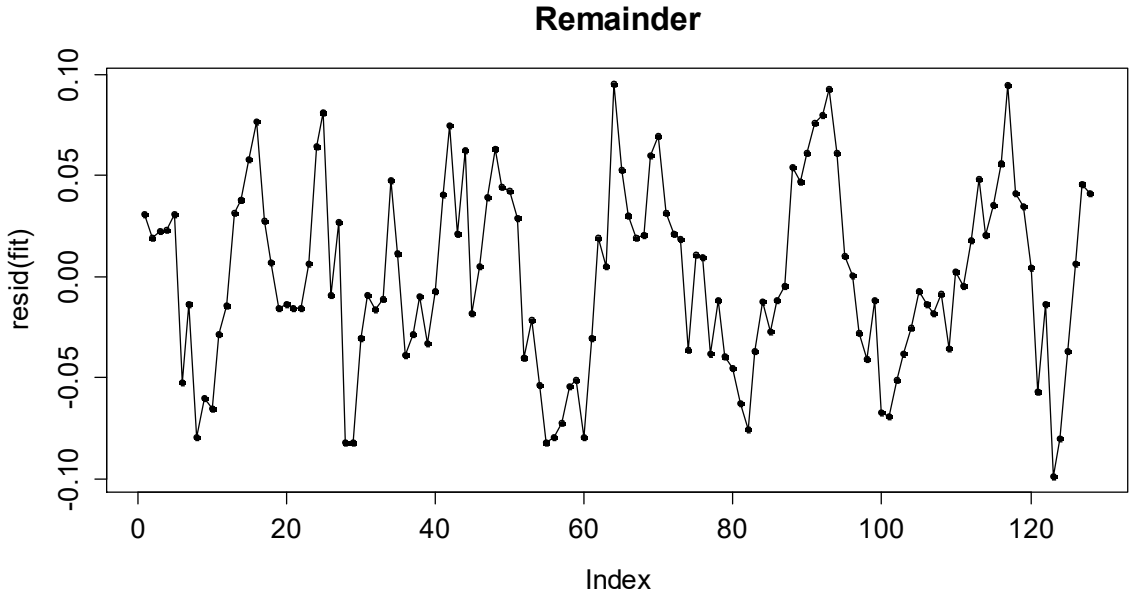

```
> plot(fit, shade=TRUE, xlab="", ylab="Time")
> seas.eff <- c(0,coef(fit)[2:12])-mean(c(0,coef(fit)[2:12]))
> plot(1:12, seas.eff, xlab="Month", ylab="", type="h")
> points(1:12, seas.eff, pch=20)
> abline(h=0, col="grey")
```

As we can see from the estimated trend and seasonal components (see next page), a simple model using a linear trend or a cyclic seasonal component would not have been suitable here. Please also note that both the trend component and the seasonal effect are centered to mean zero here.



Finally, we extract the remainder term. These are just the residuals from the GAM model, which are readily available and very quickly plotted.

```
> plot(resid(fit), type="o", pch=20)
```



The plot strongly raises the question whether the remainder term can be seen as stationary. It seems as if the behavior over the first 50 observations is markedly different than in the second two thirds of the series. Moreover, the late observations show a prominent perdiodicity with an off-season period of roughly 20 observations. Hence, further investigation of these features would certainly be required. However, we conclude our exposition on parametric modeling for time series decomposition at this point.

4.4 Autocorrelation

An important feature of time series is their (potential) serial correlation. This section aims at analyzing and visualizing these correlations. We first display the autocorrelation between two random variables X_{t+k} and X_t, which is defined as:

$$\mathrm{Cor}(X_{t+k}, X_t) = \frac{Cov(X_{t+k}, X_t)}{\sqrt{Var(X_{t+k})Var(X_t)}}$$

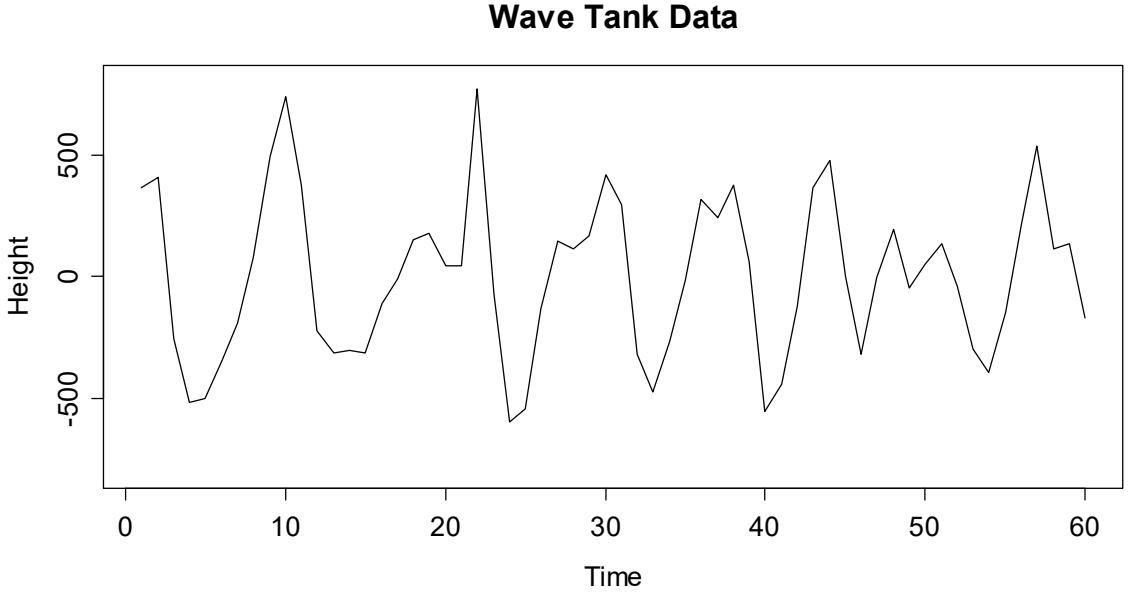
This is a dimensionless measure for the linear association between the two random variables. Since for stationary series, we require the moments to be non-changing over time, we can drop the index t for these, and write the autocorrelation as a function of the lag k:

$$\rho(k) = Cor(X_{t+k}, X_t)$$

The goals in the forthcoming sections are estimating these autocorrelations from observed time series data, and to study the estimates' properties. The latter will prove useful whenever we try to interpret sample autocorrelations in practice.

The example we consider in this chapter is the wave tank data. The values are wave heights in millimeters relative to still water level measured at the center of the tank. The sampling interval is 0.1 seconds and there are 396 observations. For better visualization, we here display the first 60 observations only:

```
> www <- "http://staff.elena.aut.ac.nz/Paul-Cowpertwait/ts/"
> dat <- read.table(paste(www,"wave.dat",sep=""), header=T)
> wave <- ts(dat$waveht)
> plot(window(wave, 1, 60), ylim=c(-800,800), ylab="Height")
> title("Wave Tank Data")
```

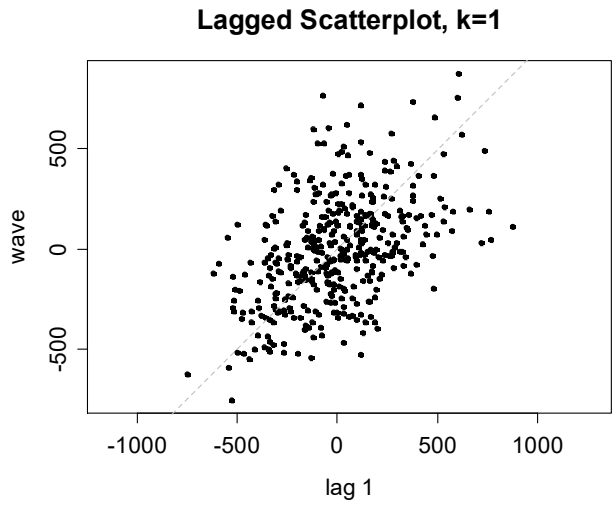


These data show some pronounced cyclic behavior. This does not come as a surprise, as we all know from personal experience that waves do appear in cycles. The series shows some very clear serial dependence, because the current value is quite closely linked to the previous and following ones. But very clearly, it is also a stationary series.

4.4.1 Lagged Scatterplot

An appealing idea for analyzing the correlation among consecutive observations in the above series is to produce a scatterplot of (x_t, x_{t+1}) for all $t = 1,...,n-1$. There is a designated function `lag.plot()` in **R**. The result is as follows:

```
> lag.plot(wave, do.lines=FALSE, pch=20)
> title("Lagged Scatterplot, k=1")
```

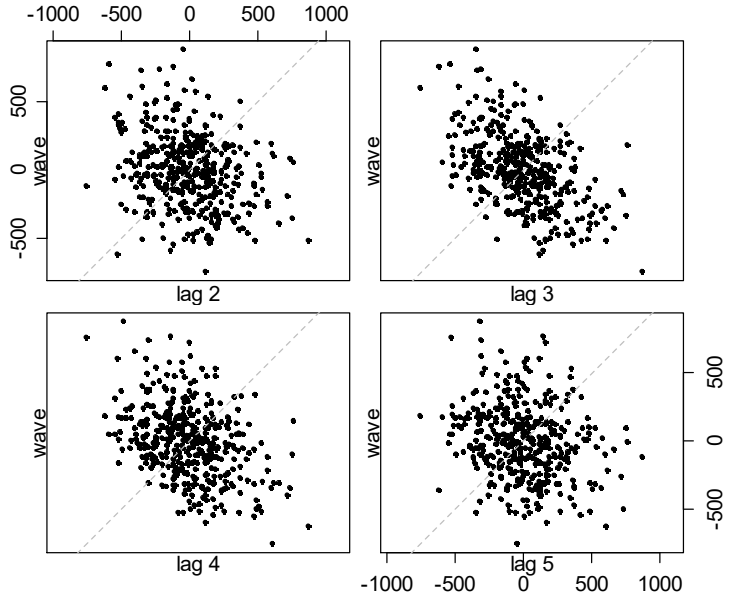


The association seems linear and is positive. The Pearson correlation coefficient turns out to be 0.47, thus moderately strong. How to interpret this value from a practical viewpoint? Well, the square of the correlation coefficient, $0.47^2 = 0.22$, is the percentage of variability explained by the linear association between x_t and its respective predecessor. Here in this case, x_{t-1} explains roughly 22% of the variability observed in x_t. We can of course extend the very same idea to higher lags. We here analyze the lagged scatterplot correlations for lags $k = 2,...5$, see next page. When computed, the estimated Pearson correlations turn out to be -0.27, -0.50, -0.39 and -0.22, respectively. The formula for computing them is:

$$\tilde{\rho}(k) = \frac{\sum_{s=1}^{n-k}(x_{s+k} - \bar{x}_{(k)})(x_s - \bar{x}_{(1)})}{\sqrt{\sum_{s=k+1}^{n}(x_s - \bar{x}_{(k)})^2 \cdot \sum_{t=1}^{n-k}(x_t - \bar{x}_{(1)})^2}} \text{ for } k = 1,...,n-2,$$

where $\bar{x}_{(1)} = \frac{1}{n-k}\sum_{i=1}^{n-k} x_i$ and $\bar{x}_{(k)} = \frac{1}{n-k}\sum_{i=k+1}^{n} x_i$

It is important to notice that while there are $n-1$ data pairs for computing $\tilde{\rho}(1)$, there are only $n-2$ for $\tilde{\rho}(2)$, and then less and less, i.e. $n-k$ pairs for $\tilde{\rho}(k)$. Thus for the last autocorrelation coefficient which can be estimated, $\tilde{\rho}(n-2)$, there is only one single data pair which is left. Of course, they can always be interconnected by a straight line, and the correlation in this case is always ± 1. Of course, this is an estimation snag, rather than perfect linear association for the two random variables. Intuitively, it is clear that because there are less and less data pairs at higher lags, the respective estimated correlations are less and less precise. Indeed, by digging deeper in mathematical statistics, one can prove that the variance of $\tilde{\rho}(k)$ increases with k. This is undesired, as it will lead to instable results and spurious effects. The remedy is discussed in the next section.



4.4.2 Plug-In Estimation

For mitigating the above mentioned problem with the lagged scatterplot method, autocorrelation estimation is commonly done using the so-called plug-in approach, using estimated autocovariances as the basis. The formula is as follows:

$$\hat{\rho}(k) = \frac{\hat{\gamma}(k)}{\hat{\gamma}(0)}, \text{ for } k = 1,...,n-1,$$

where $\hat{\gamma}(k) = \frac{1}{n}\sum_{s=1}^{n-k}(x_{s+k} - \bar{x})(x_s - \bar{x})$, with $\bar{x} = \frac{1}{n}\sum_{t=1}^{n} x_t$.

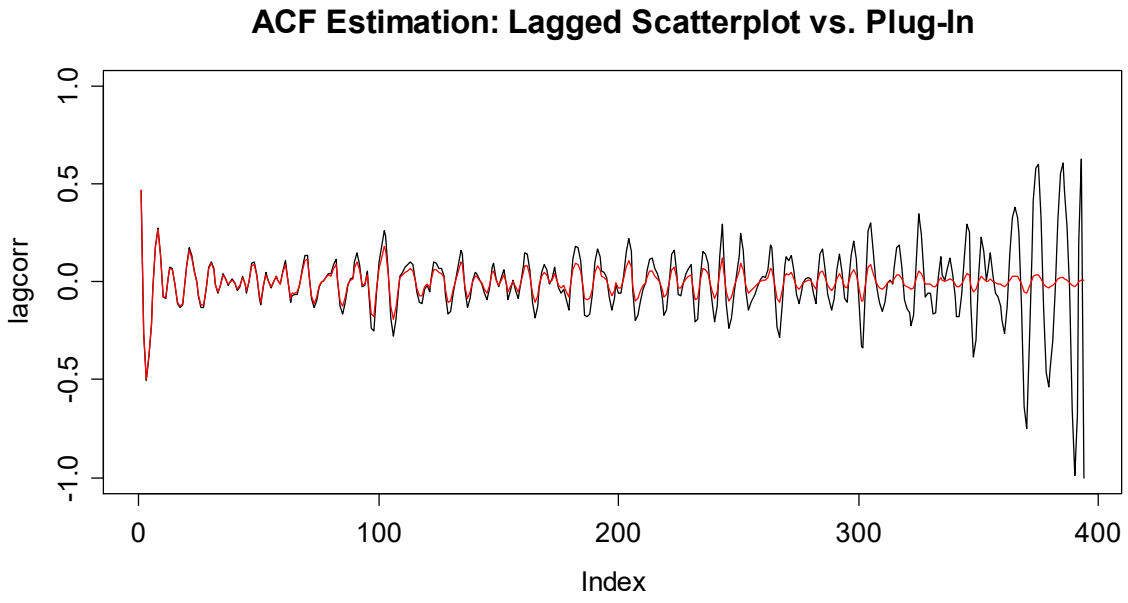
Note that here, n is used as a denominator irrespective of the lag and thus the number of summands. This has the consequence that $\hat{\gamma}(0)$ is not an unbiased estimator for $\gamma(0) = \sigma_X^2$, but as explained above, there are good reasons to do so. When plugging in the above terms, the estimate for the kth autocorrelation coefficient turns out to be:

$$\hat{\rho}(k) = \frac{\sum_{s=1}^{n-k}(x_{s+k} - \bar{x})(x_s - \bar{x})}{\sum_{t=1}^{n}(x_t - \bar{x})^2}, \text{ for } k = 1,...,n-1.$$

It is straightforward to compute these in **R**, function `acf()` does the job, and we below do so for the wave tank data. As for the moment, we are interested in the numerical results, we set argument `plot=FALSE`. However, as we will see below, it is usually better to visualize the estimated autocorrelation coefficients graphically, as it will be explained below in section 4.4.3. Also note that **R** by default does not return all autocorrelations which are estimable in this series with 396 observations, but only the first 25.

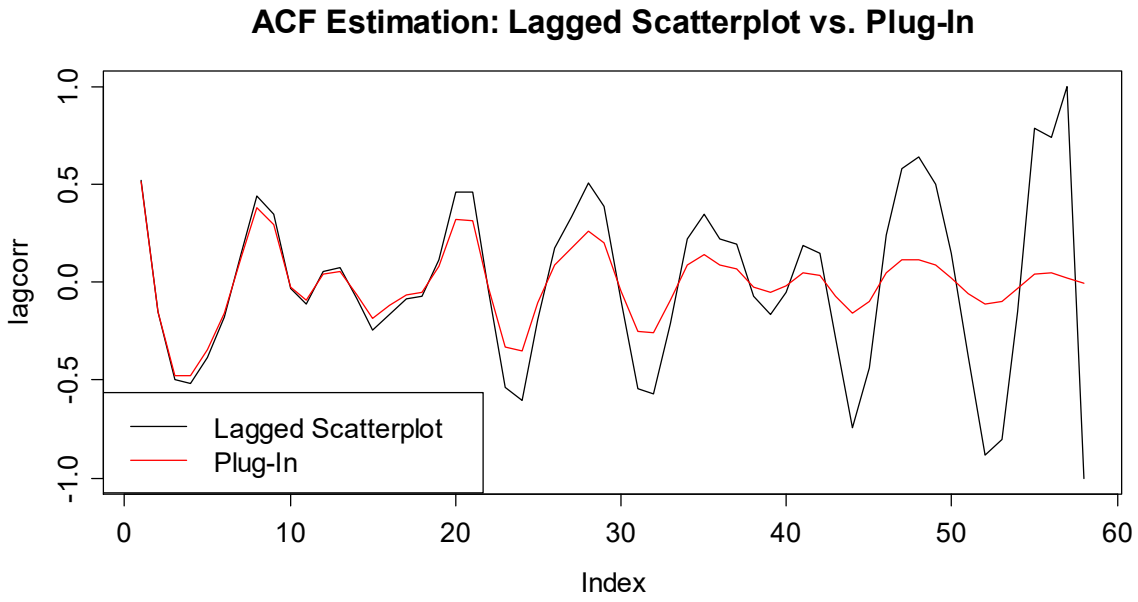
```
> acf(wave, plot=FALSE)

Autocorrelations of series wave, by lag
     0      1      2      3      4      5      6      7
 1.000  0.470 -0.263 -0.499 -0.379 -0.215 -0.038  0.178
     8      9     10     11     12     13     14     15
 0.269  0.130 -0.074 -0.079  0.029  0.070  0.063 -0.010
    16     17     18     19     20     21     22     23
-0.102 -0.125 -0.109 -0.048  0.077  0.165  0.124  0.049
    24     25
-0.005 -0.066
```

Next, we compare the autocorrelations from lagged scatterplot estimation vs. the ones from the plug-in approach. These are displayed below. While for the first 50 lags, there is not much of a difference, the plug-in estimates are much more damped for higher lags. As claimed above, the lagged scatterplot estimate shows a value of -1 for lag 394, and some generally very erratic behavior in the few lags before.



We can “prove”, or rather, provide evidence that this is an estimation artifact if we restrict the series to the first 60 observations and then repeat the estimation of autocorrelations, see next page. Again, for the highest few lags which are estimable, the lagged scatterplot approach shows erratic behavior – and this was not present at the same lags, when the series was still longer. We do not observe this kind of effect with the plug-in based autocorrelations, thus this is clearly the method of choice.

We finish this chapter by repeating that the bigger the lag, the fewer data pairs remain for estimating the autocorrelation coefficient. We discourage of the use of the lagged scatterplot approach. While the preferred plug-in approach is biased due to the built-in damping mechanism, i.e. the estimates for high lags are shrunken towards zero; it can be shown that it has lower mean squared error. This is because it produces results with much less (random) variability. It can also be shown that the plug-in estimates are consistent, i.e. the bias disappears asymptotically.



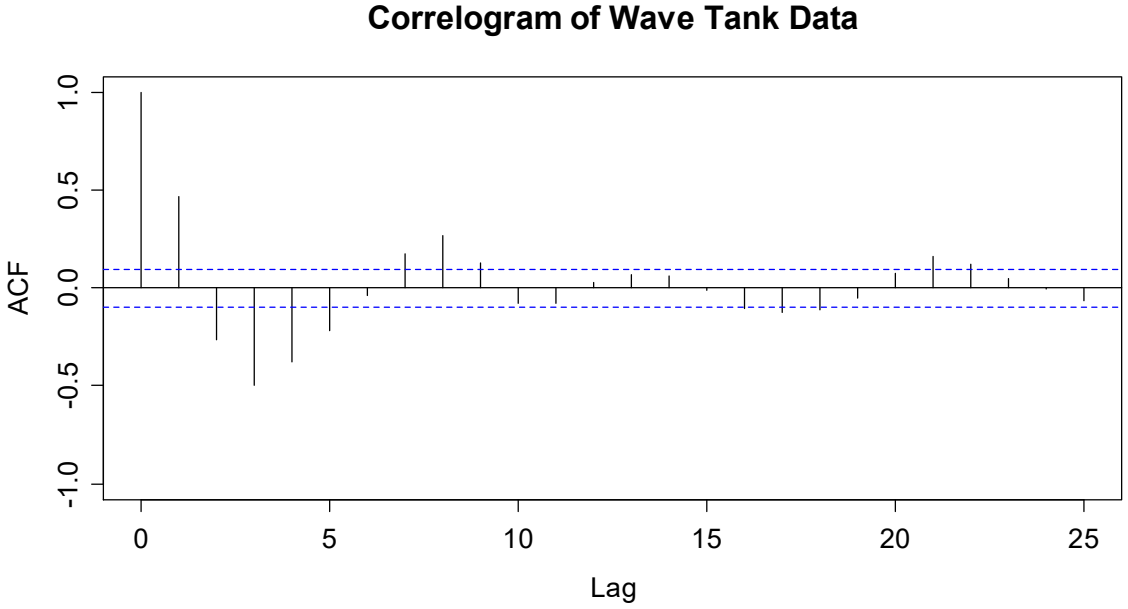
Nevertheless, all our findings still suggest that it is a good idea to consider only a first portion of the estimated autocorrelations. A rule of the thumb suggests that $10 \cdot \log_{10}(n)$ is a good threshold. For a series with 100 observations, the threshold becomes lag 20. A second rule operates with $n/4$ as the maximum lag to which the autocorrelations are shown.

4.4.3 Correlogram

Now, we know how to estimate the autocorrelation function (ACF) for any lag k. Here, we introduce the *correlogram*, the standard means of visualization for the ACF. We will then also study the properties of the ACF estimator. We employ **R** and type (see next page for the graphical output):

```
> acf(wave, ylim=c(-1,1))
```

It has become a widely accepted standard to use vertical spikes for displaying the estimated autocorrelations. Also note that the ACF in **R** by default starts with lag 0, at which it always takes the value 1. If one does not like the spike at lag 0, one can alternatively use the `Acf()` function from `library(forecast)`. For better judgment, we also recommend setting the y-range to the interval $[-1,1]$. Apart from these technicalities, the ACF reflects the properties of the series. We also observe a cyclic behavior with a period of 8, as it is apparent in the time series plot of the original data. Moreover, the absolute value of the correlations attenuates with increasing lag. Next, we will discuss the interpretation of the correlogram.



Confidence Bands

It is obvious that even for an iid series without any serial correlation, and thus $\rho(k)=0$ for all k, the estimated autocorrelations $\hat{\rho}(k)$ will generally not be zero. Hopefully, they will be small, but the question is how much they can differ from zero just by chance. An answer is indicated by the confidence bands, i.e. the blue dashed lines in the plot above. The confidence bands are based on an asymptotic result: for long iid time series, it can be shown that the $\hat{\rho}(k)$ approximately follow a $N(0,1/n)$ distribution. Thus, under the null hypothesis that a series is iid and hence $\rho(k)=0$ for all k, the 95% acceptance region for the null is given by the interval $\pm 1.96/\sqrt{n}$. This leads us to the following statement that facilitates interpretation of the correlogram:

"*for any stationary time series, sample autocorrelation coefficients* $\hat{\rho}(k)$ *that fall within the confidence band of* $\pm 1.96/\sqrt{n}$ *are considered to be different from* 0 *only by chance, while those outside the confidence band are considered to be truly different from* 0*.*"

On the other hand, the above statement means that even for iid series, we expect 5% of the estimated ACF coefficients to exceed the confidence bounds; these

correspond to type 1 errors in the statistical testing business. Please note again that the indicated bounds are asymptotic and derived for iid series. The properties of serially dependent finite length series are much harder to derive!

Ljung-Box Test

The Ljung-Box approach tests the null hypothesis that a number of autocorrelation coefficients are simultaneously equal to zero. Or, more colloquially, it evaluates whether there is any significant autocorrelation in a series. The test statistic is:

$$Q(h) = n \cdot (n+2) \cdot \sum_{k=1}^{h} \frac{\hat{\rho}_k^2}{n-k}$$

Here, n is the length of the time series, $\hat{\rho}_k$ are the sample autocorrelation coefficients at lag k and h is the lag up to which the test is performed. It is typical to use $h = 1$, 3, 5, 10 or 20. The test statistic asymptotically follows a χ^2 distribution with h degrees of freedom. As an example, we compute the test statistic and the respective p-value for the wave tank data with $h = 10$.

```
> nn <- length(wave)
> qq <- nn*(nn+2)*sum((acf(wave)$acf[2:11]^2)/(nn-(1:10)))
> qq
[1] 344.0155
> 1-pchisq(qq, 10)
[1] 0
```

We observe that $Q(10) = 344.0155$ which is far in excess of what we would expect by chance on independent data.The critical value, i.e. the 95%-quantile of the χ^2_{10} is at 18.3 and thus, the p-value is close to (but not exactly) zero. There is also a dedicated R function which can be used to perform Ljung-Box testing:

```
> Box.test(wave, lag=10, type="Ljung-Box")
Box-Ljung test
data: wave
X-squared = 344.0155, df = 10, p-value < 2.2e-16
```

The result is, of course, identical. Please be aware that the test is sometimes also referred to as Box-Ljung test. Also R is not very consistent in its nomenclature. However, the two are one and the same. Moreover, with a bit of experience the results of the Ljung-Box test can usually be guessed quite well from the correlogram by eyeballing.

ACF of Non-Stationary Series

Estimation of the ACF from an observed time series assumes that the underlying process is stationary. Only then we can treat pairs of observations at lag k as being probabilistically “equal” and compute sample covariance coefficients. Hence, while stationarity is at the root of ACF estimation, we can of course still apply the formulae given above to non-stationary series. The ACF then usually exhibits some typical

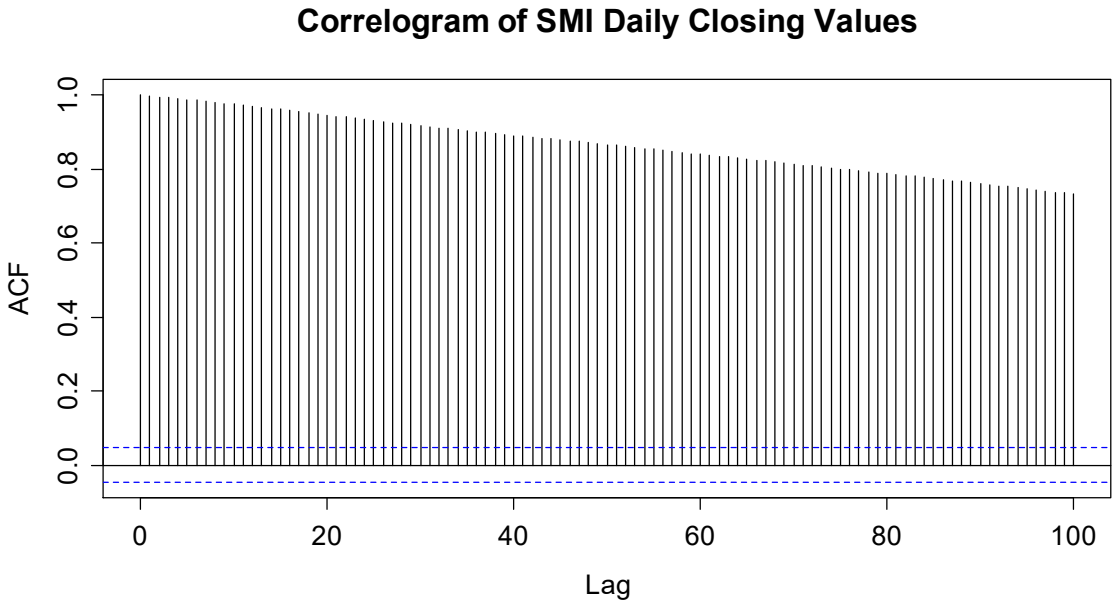
patterns. This can serve as a second check for non-stationarity, i.e. helps to identify it, should it have gone unnoticed in the time series plot. We start by showing the correlogram for the SMI daily closing values from section 1.2.4. This series does not have seasonality, but a very clear trend.

```
> acf(smi, lag.max=100)
```

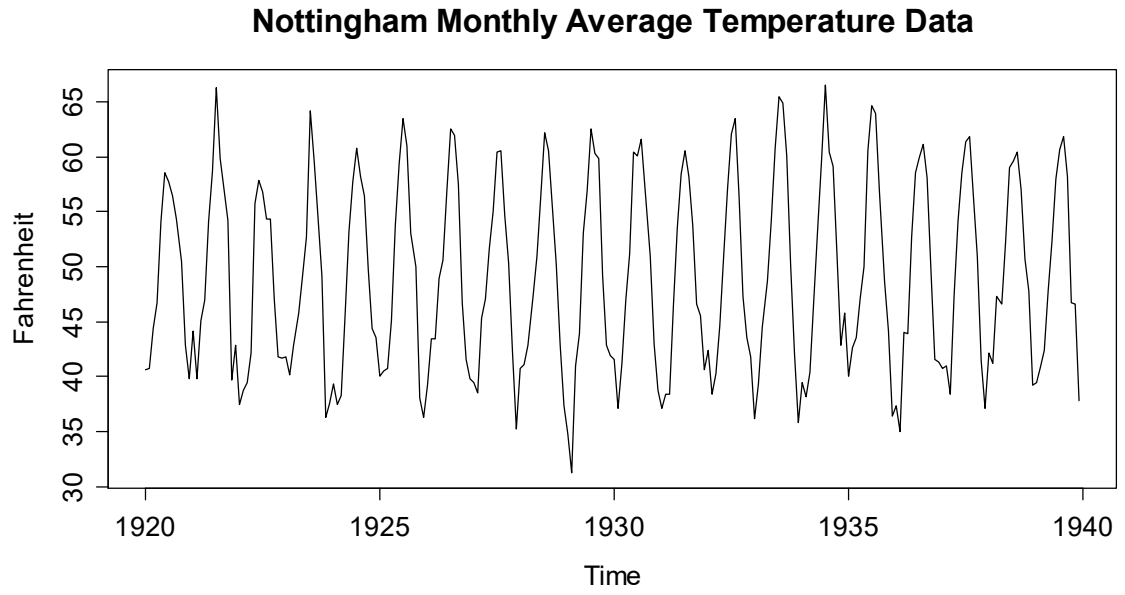
We observe that the ACF decays very slowly. The reason is that if a time series features a trend, the observations at consecutive observations will usually be on the same side of the series' global mean $\bar{x}$. This is why that for small to moderate lags k, most of the terms

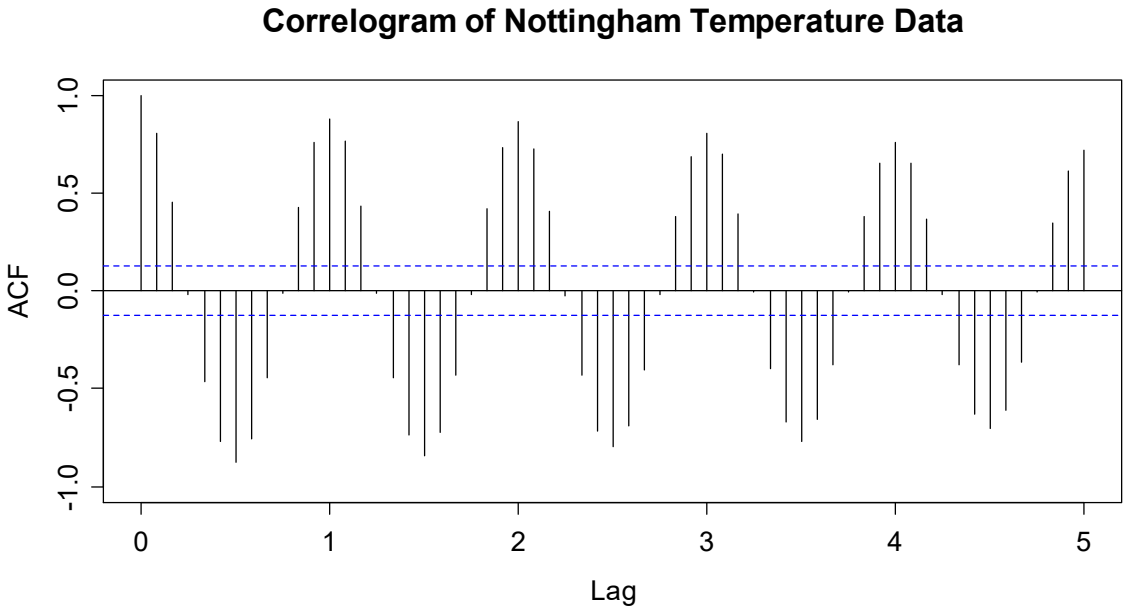
$$(x_{s+k} - \bar{x})(x_s - \bar{x})$$

are positive. For this reason, the sample autocorrelation coefficient will be positive as well, and is most often also close to 1. Thus, a very slowly decaying ACF is an indicator for non-stationarity, i.e. a trend which was not removed before autocorrelations were estimated.



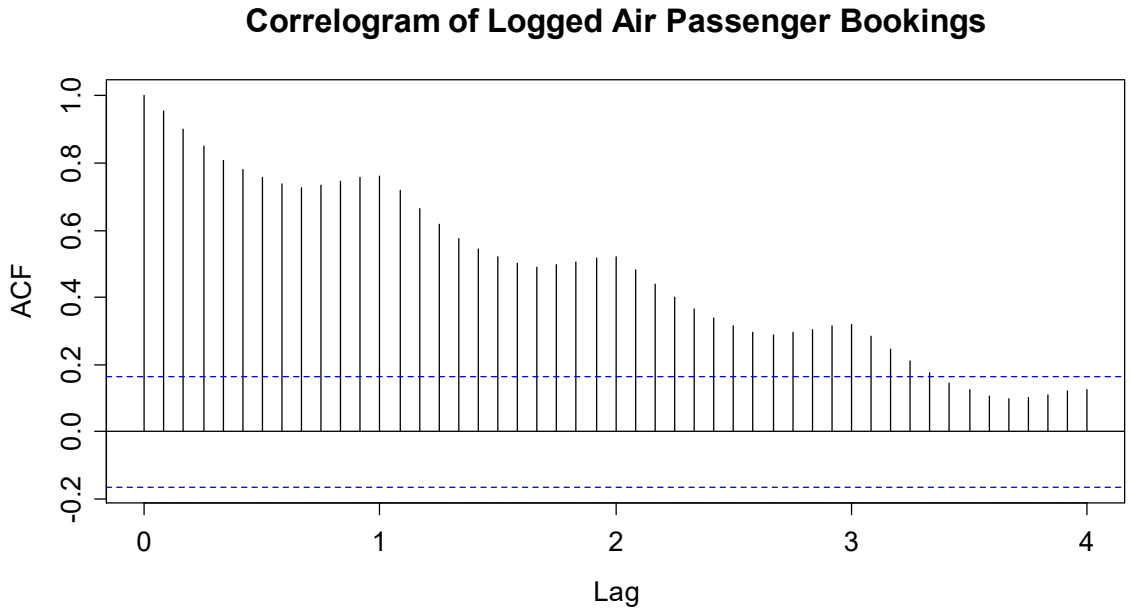
Next, we show an example of a series that has no trend, but a strongly recurring seasonal effect. We use **R**'s `data(nottem)`, a time series containing monthly average air temperatures at Nottingham Castle in England from 1920-1939. Time series plot and correlogram are as follows:





The ACF is cyclic, and owing to the recurring seasonality, the envelope again decays very slowly. Also note that for periodic series, **R** has periods rather than lags on the x-axis – often a matter of confusion. We conclude that a hardly, or very slowly decaying periodicity in the correlogram is an indication of a seasonal effect which was forgotten to be removed. Finally, we also show the correlogram for the logged air passenger bookings. This series exhibits both an increasing trend and a seasonal effect. The result is as follows:

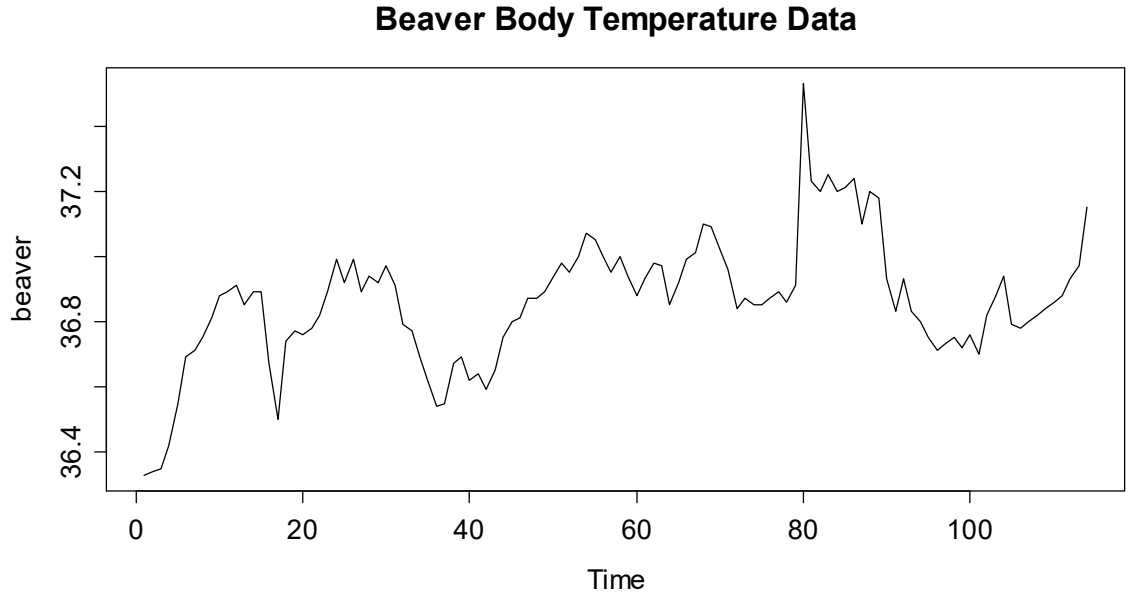
```
> data(AirPassengers)
> txt <- "Correlogram of Logged Air Passenger Bookings"
> acf(log(AirPassengers), lag.max=48, main=txt)
```



Here, the two effects described above are interspersed. We have a (here dominating) slow decay in the general level of the ACF, plus some periodicity. Again, this is an indication for a non-stationary series. It needs to be decomposed, before the serial correlation in the stationary remainder term can be studied.

The ACF and Outliers

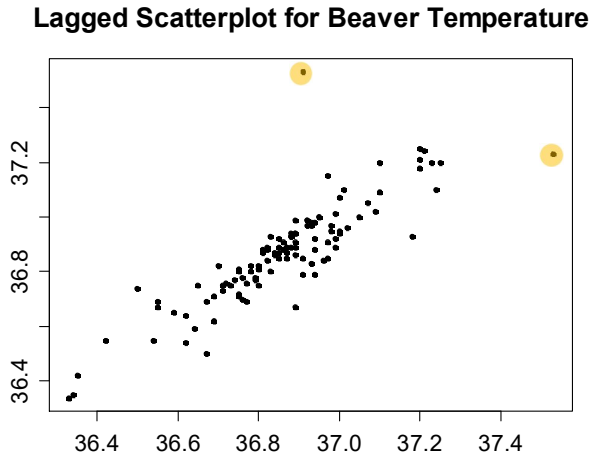
If a time series has an outlier, it will appear twice in any lagged scatterplot, and will thus potentially have "double" negative influence on the $\hat{\rho}(k)$. As an example, we consider variable `temp` from data frame `beaver1`, which can be found in **R**'s `data(beavers)`. This is the body temperature of a female beaver, measured by telemetry in 10 minute intervals. We first visualize the data with a time series plot.



Observation 80 is a moderate, but distinct outlier. It is unclear to the author whether this actually is an error, or whether the reported value is correct. But because the purpose of this section is showing the potential influence of erroneous values, that is not important. Neither the Pearson correlation coefficient, nor the plug-in autocorrelation estimator is robust, thus the appearance of the correlogram can be altered quite strongly due to the presence of just one single outlier.

```
> plot(beaver[1:113], beaver[2:114], pch=20,)
> title("Lagged Scatterplot for Beaver Temperature")
```

The two data points where the outlier is involved are highlighted. The Pearson correlation coefficients with and without these two observations are 0.86 and 0.91. Depending on the outliers severity, the difference can be much bigger. The next plot shows the entire correlogram for the beaver data, computed with (black) and without (red) the outlier. Also here, the difference may seem small and rather academic, but it could easily be severe if the outlier was just pronounced enough.

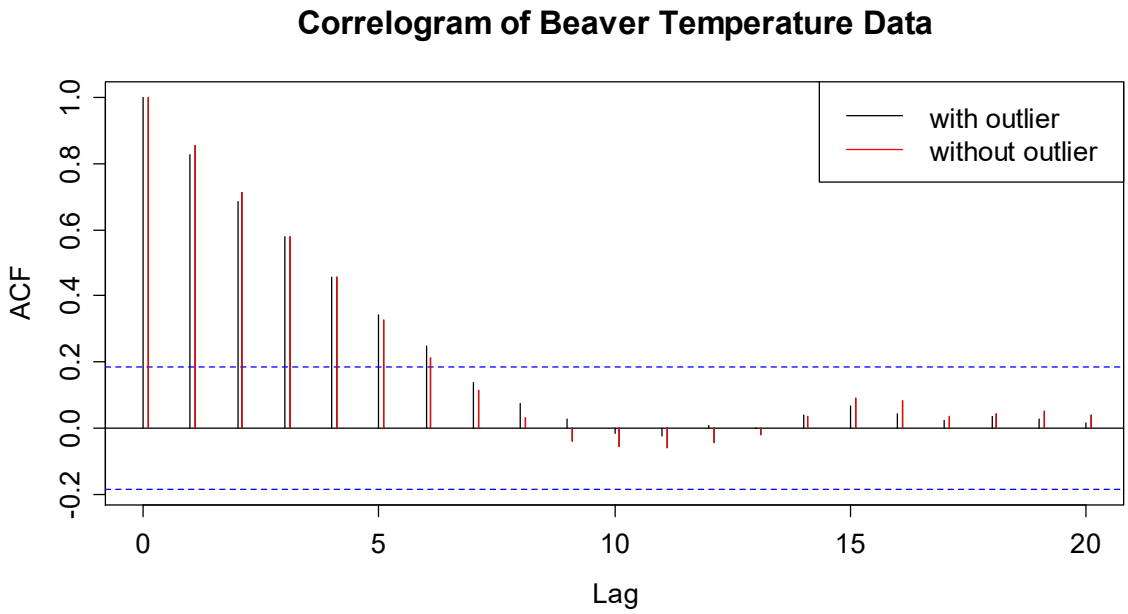


The question is, how do we handle missing values in time series? In principle, we cannot just omit them without breaking the time structure. And breaking it means going away from our paradigm of equally spaced points in time. A popular choice is thus replacing the missing value. This can be done with various degrees of sophistication:

a) replacing the value with the global mean
b) using a local mean, i.e. +/- 3 observations
c) model based imputation by forecasting

The best strategy depends upon the case at hand. And in fact, there is a fourth alternative: while R's `acf()` function by default does not allow for missing values, it still offers the option to proceed without imputation. If argument is set as `na.action=na.pass`, the covariances are computed from the complete cases, and the correlogram is shown as usual. However, having missed values in the series has the consequence that the estimates produced may well not be a valid (i.e. positive definite) autocorrelation sequence, and may contain missing values. From

a practical viewpoint, these drawbacks can often be neglected, though. Also many other R functions for time series analysis allow for the presence of missing values if the arguments are set properly.



4.4.4 Quality of ACF Estimates

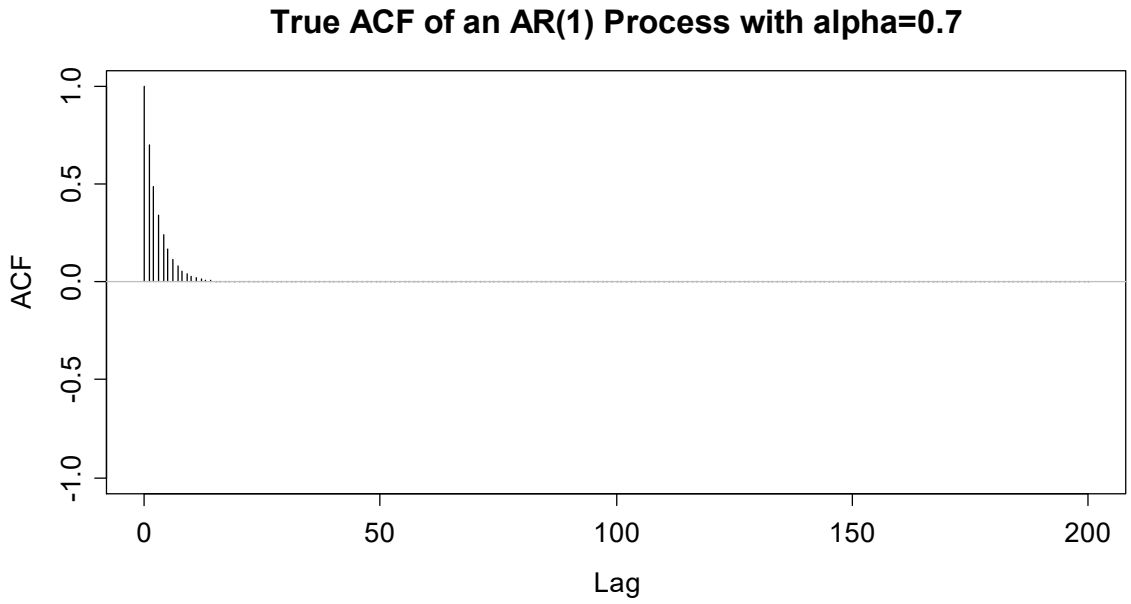
In this section we will deal with the quality of the information that is contained in the correlogram. We will not do this from a very theoretical viewpoint, but rather focus on the practical aspects. We have already learned that the ACF estimates from the plug-in approach are generally biased, i.e. shrunken towards zero for higher lags. This means that it is better to cut off the correlogram at a certain lag. Furthermore, non-stationarities in the series can hamper the interpretation of the correlogram and we have also seen that outliers can have a quite strong impact. But there are even more aspects in ACF estimation that are problematic...

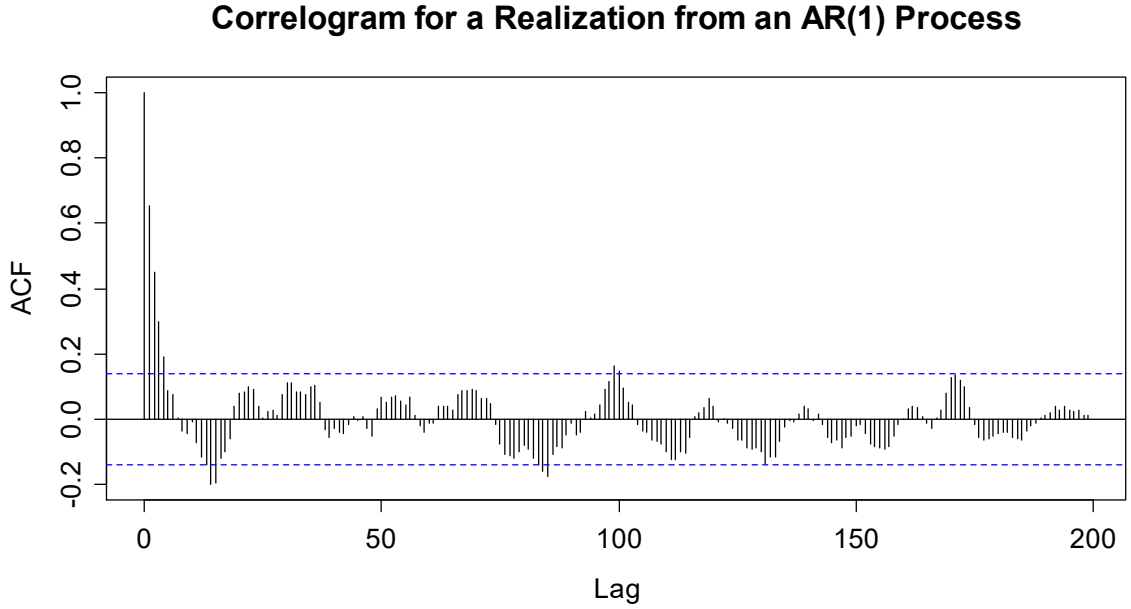
The Compensation Issue

One can show that the sum of all autocorrelation coefficients which can be estimated from a time series realization, i.e. the sum over all $\hat{\rho}(k)$ for lags $k = 1,...,n-1$, adds up to -1/2. Or, written as a formula:

$$\sum_{k=1}^{n-1} \hat{\rho}(k) = -\frac{1}{2}$$

We omit a formal proof here, but give empirical evidence below. It is clear that the above condition will lead to quite severe artifacts, especially when a time series process has only positive correlations. We here show both the true, theoretical ACF of an $AR(1)$ process with $\alpha_1 = 0.7$, which, as we will see in section 5, has $\rho(k) = \alpha_1^k = (0.7)^k > 0$ for all k, and the sample correlogram for a realization of that process with a length 200 observations.





The respective R-commands for producing these plots are as follows:

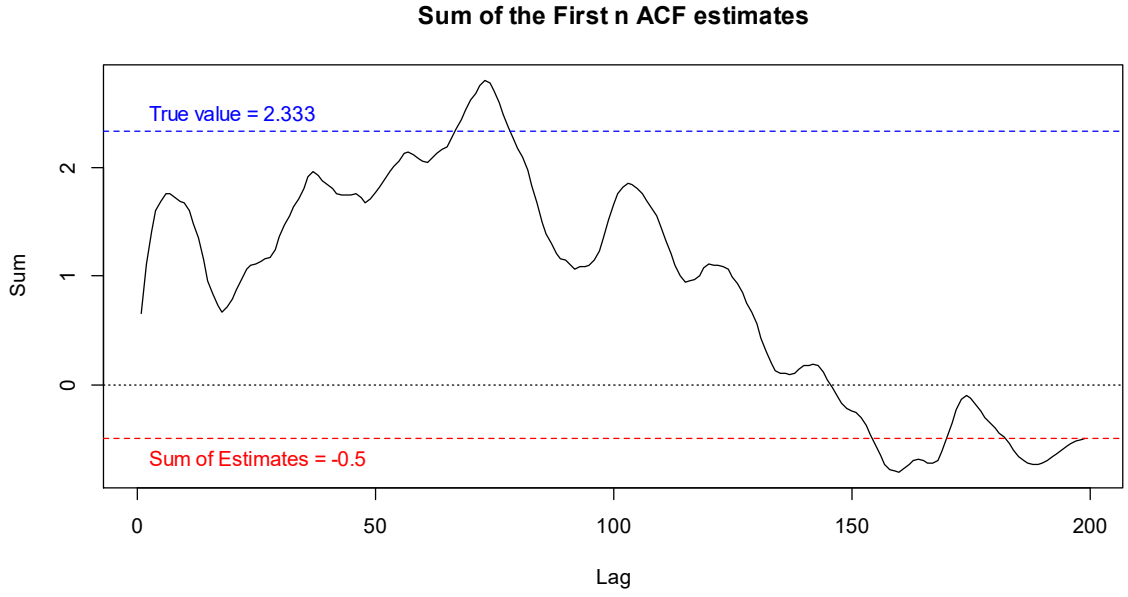
```
## True ACF
true.acf <- ARMAacf(ar=0.7, lag.max=200)
plot(0:200, true.acf, type="h", xlab="Lag", ylim=c(-1,1))
title("True ACF of an AR(1) Process with alpha=0.7")
abline(h=0, col="grey")

## Simulation and Generating the ACF
set.seed(25)
ts.simul <- arima.sim(list(ar=0.7), 200)
acf(ts.simul, lag=200, main="Correlogram ...")
```

What we observe is quite striking: only for the very first few lags, the sample ACF does match with its theoretical counterpart. As soon as we are beyond lag $k = 6$, the sample ACF turns negative. This is an artifact, because the sum of the estimated autocorrelations coefficients needs to add up to -1/2. We quickly verify this using the following R command on the simulated $n = 200$ series. Please be aware that the `acf()` command in R also outputs the ACF estimate at lag 0, so the sum only starts from the second term in the output object:

```
> est <- acf(ts.simul,length(ts.simul))$acf
> sum(est[2:length(ts.simul)])
[1] -0.5
```

Some of these spurious, negative correlation estimates are so big that they even exceed the confidence bounds – an observation that has to be well kept in mind if one analyzes and tries to interpret the correlogram. We conclude this section by visualizing the cumulative sum of estimated autocorrelation coefficients for the realization of the above $AR(1)$ process.

```
> sum.acf <-numeric()
> for (i in 2:(length(simul))) sum.acf[i-1] <- sum(est[2:i])
> plot(1:(length(simul)-1), sum.acf, type="l", main="…")
> abline(h=0, lty=3)
> abline(h=-0.5, col="red", lty=2)
> abline(h=sum(.7^(1:199)), col="blue", lty=2)
> text(0,2.5, "True value = 2.333", col="blue", pos=4)
> text(0,-.67, "Sum of Estimates = -0.5", col="red", pos=4)
```

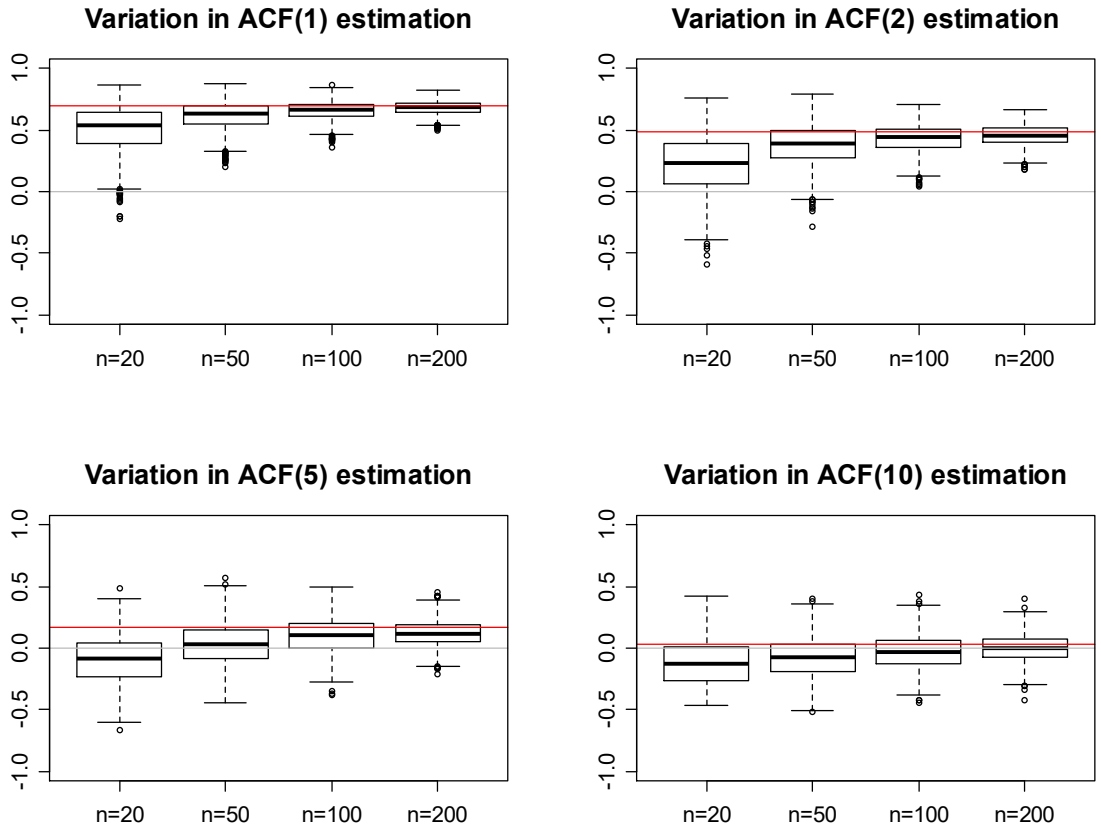


The true value for the sum of $\rho(k),\ k = 1,...,199$ is $\sum_{k=1}^{199}(0.7)^k = 2.333$, so the sum of the estimated autocorrelation coefficients is very far from the true value. This again emphasizes that the estimates should be taken with a grain of salt and that the confidence bands are optimistically small.

Simulation Study

Last but not least, we will run a small simulation study that visualizes bias and variance in the sample autocorrelation coefficients. We will again base this on the simple $AR(1)$ process with coefficient $\alpha_1 = 0.7$. For further discussion of the process' properties, we refer to section 5. There, it will turn out that the k^{th} autocorrelation coefficient of such a process takes the value $(0.7)^k$, as visualized on the previous page.

For understanding the variability in $\hat{\rho}(1)$, $\hat{\rho}(2)$, $\hat{\rho}(5)$ and $\hat{\rho}(10)$, we simulate from the aforementioned $AR(1)$ process. We generate series of length $n = 20$, $n = 50$, $n = 100$ and $n = 200$. We then obtain the correlogram, record the estimated autocorrelation coefficients and repeat this process 1000 times. This serves as a basis for displaying the variability in $\hat{\rho}(1)$, $\hat{\rho}(2)$, $\hat{\rho}(5)$ and $\hat{\rho}(10)$ with boxplots. They can be found below.



We observe that for "short" series with less than 100 observations, estimating the ACF is difficult: the $\hat{\rho}(k)$ are strongly biased, and there is huge variability. Only for longer series, the consistency of the estimator "kicks in", and yields estimates which are reasonably precise. For lag $k = 10$, on the other hand, we observe less bias, but the variability in the estimate remains large, even for "long" series.

We conclude this situation by summarizing: by now, we have provided quite a bit of evidence that the correlogram can be tricky to interpret at best, sometimes even

misleading, or plain wrong. However, it is the best means we have for understanding the dependency in a time series. And we will base many if not most of our decisions in the modeling process on the correlogram. However, please be aware of the bias and the estimation variability there is.

4.4.5 Confidence Interval for the Time Series Mean

An important application of the theory on autocorrelations discussed above is the construction of a confidence interval for the mean of a time series. Let us assume we are given a stationary time series $X = (X_1, X_2, \ldots, X_n)$. Then, the global mean of the series is estimated as:

$$\hat{\mu} = \frac{1}{n}\sum_{t=1}^{n} X_t$$

For the construction of a confidence interval we require an estimation of $Var(\hat{\mu})$. In case of iid observation, we have $Var(\hat{\mu}) = \sigma_X^2 / n$, so that plugging in the sample variance $\hat{\sigma}_X^2$ does the job. Unfortunately, with a time series that has autocorrelated instances, things are more complicated. In particular:

$$\begin{aligned}
Var(\hat{\mu}) &= \frac{1}{n^2} \cdot Var\left(\sum_{t=1}^{n} X_t\right) \\
&= \frac{1}{n^2} \cdot Cov\left(\sum_{t=1}^{n} X_t, \sum_{t=1}^{n} X_t\right) \\
&= \frac{1}{n^2} \cdot \sum_{t=1}^{n}\sum_{s=1}^{n} Cov(X_s, X_t) \\
&= \frac{1}{n^2} \cdot \sum_{t=1}^{n}\sum_{s=1}^{n} \gamma(|t-s|) \\
&= \frac{\gamma(0)}{n^2} \cdot \sum_{t=1}^{n}\sum_{s=1}^{n} \rho(|t-s|) \\
&= \frac{\gamma(0)}{n^2} \cdot \left(n + 2 \cdot \sum_{k=1}^{n-1} (n-k)\rho(k)\right)
\end{aligned}$$

As we can see, it depends on all autocorrelations $\rho(k)$ whether $Var(\hat{\mu})$ is bigger or smaller than under independence. Unless we have knowledge about all these coefficients, we cannot make a statement.

In reality, one often has to deal with time series that only feature positive autocorrelation coefficients. In that case $Var(\hat{\mu})$ will be larger than for an iid series. Hence, falsely assuming independence may lead to deflated confidence intervals and spuriously significant results.

So how to practice in practice? Plugging in all autocorrelations down to $\hat{\rho}(n-1)$ into the above formula seems like a poor choice given the mediocre quality of the estimates at higher lags. A reasonable compromise is to plug-in $\hat{\rho}(k)$ for lags

$k = 1,...,10 \cdot \log_{10}(n)$ and zero for the higher lags. The confidence interval for the mean is then derived from the usual Gaussian asymptotics:

$$\hat{\mu} \pm 1.96 \cdot \sqrt{\frac{\gamma(0)}{n^2} \cdot \left(n + 2 \cdot \sum_{k=1}^{10 \cdot \log_{10}(n)} (n-k)\rho(k) \right)}$$

We illustrate the issue based on the series with the body temperature of the beaver from above. The mean and the faulty confidence interval under iid assumption are simply computed as:

```
> mean(beaver)
[1] 36.862
> mean(beaver)+c(-1.96,1.96)*sd(beaver)/sqrt(length(beaver))
[1] 36.827 36.898
```

When adjusting for the sequential correlation of the observations, the confidence interval becomes around 2.7x longer, which can make a big difference!

```
> n      <- length(beaver)
> var.ts <- 1/n^2*
            acf(beaver, lag=0, type="covariance")$acf[1]*
            (n+2*sum(((n-1):(n-10))*acf(beaver,10)$acf[-1]))
> mean(beaver) + c(-1.96,1.96)*sqrt(var.ts)
[1] 36.765 36.959
```

4.5 Partial Autocorrelation

For the above, pure $AR(1)$ process, with its strong positive correlation at lag 1, it is somehow “evident” that the autocorrelation for lags 2 and higher will be positive as well – just by propagation: if A is highly correlated to B, and B is highly correlated to C, then A is usually highly correlated to C as well. It would now be very instructive to understand the direct relation between A and C, i.e. exploring what dependency there is in excess to the one associated to B. In a time series context, this is exactly what the partial autocorrelations do. The mathematical definition is the one of a conditional correlation:

$$\pi(k) = Cor(X_{t+k}, X_t \mid X_{t+1} = x_{t+1}, \ldots, X_{t+k-1} = x_{t+k-1})$$

In other words, we can also say that the partial autocorrelation is the association between X_t and X_{t+k} with the linear dependence of X_{t+1} through X_{t+k-1} removed. Another instructive analogy can be drawn to linear regression. The autocorrelation coefficient $\rho(k)$ measures the simple dependence between X_t and X_{t+k}, whereas the partial autocorrelation $\pi(k)$ measures the contribution to the multiple dependence, with the involvement of all intermediate instances $X_{t+1},...,X_{t+k-1}$ as explanatory variables. There is a (theoretical) relation between the partial autocorrelations $\pi(k)$ and the plain autocorrelations $\rho(1),...,\rho(k)$, i.e. they can be derived from each other, e.g.:

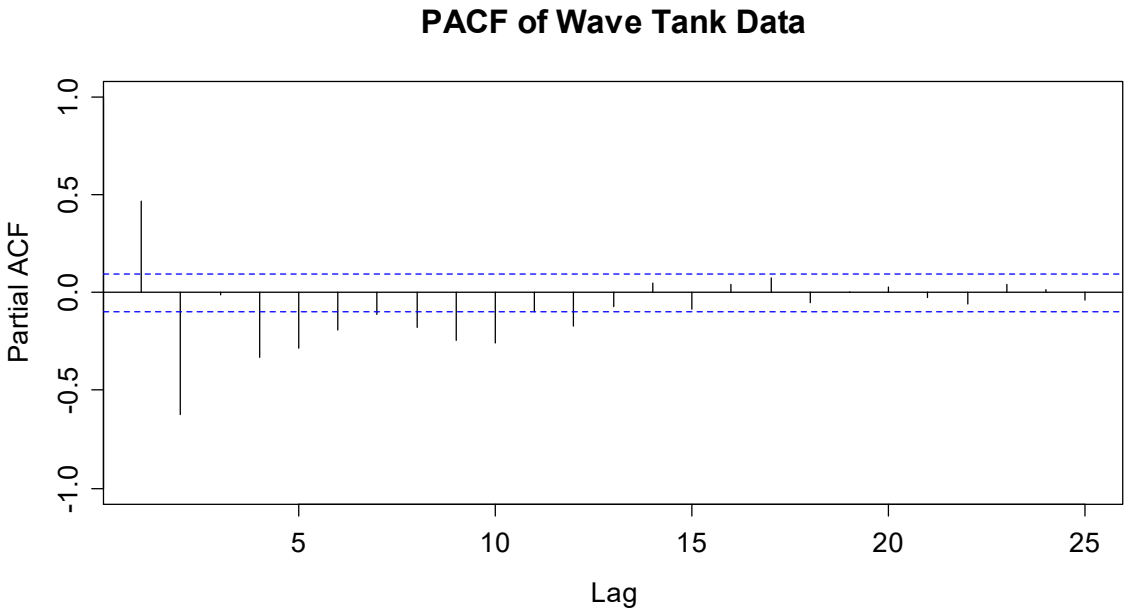
$$\pi(1) = \rho(1) \text{ and } \pi(2) = (\rho(2) - \rho(1)^2) / (1 - \rho(1)^2)$$

The formula for higher lags k exists, but get complicated rather quickly, so we do without displaying them. However, another absolutely central property of the partial autocorrelations $\pi(p)$ is that the k^{th} coefficent of the $AR(p)$ model, denoted as α_p, is equal to $\pi(p)$. While there is an in depth discussion of $AR(p)$ models in section 5, we here briefly sketch the idea, because it makes the above property seem rather logical. An autoregressive model of order p, i.e. an $AR(p)$ is:

$$X_t = \alpha_1 X_{t-1} + \ldots + \alpha_k X_{t-p} + E_t,$$

where E_t is a sequence of iid random variables. Making the above statement concrete, this means that in an $AR(3)$ process, we have $\pi(3) = \alpha_3$, but generally $\pi(2) \neq \alpha_2$ and $\pi(1) \neq \alpha_1$. Moreover, we have $\pi(k) = 0$ for all $k > p$. These properties are used in R for estimating partial autocorrelation coefficients. Estimates $\hat{\pi}(p)$ are generated by fitting autoregressive models of successively higher orders. The job is done with function `pacf()`: input/output are equal/similar to ACF estimation. In particular, the confidence bounds are also presented for the PACF. We conclude this section by showing the result for the wave tank data.

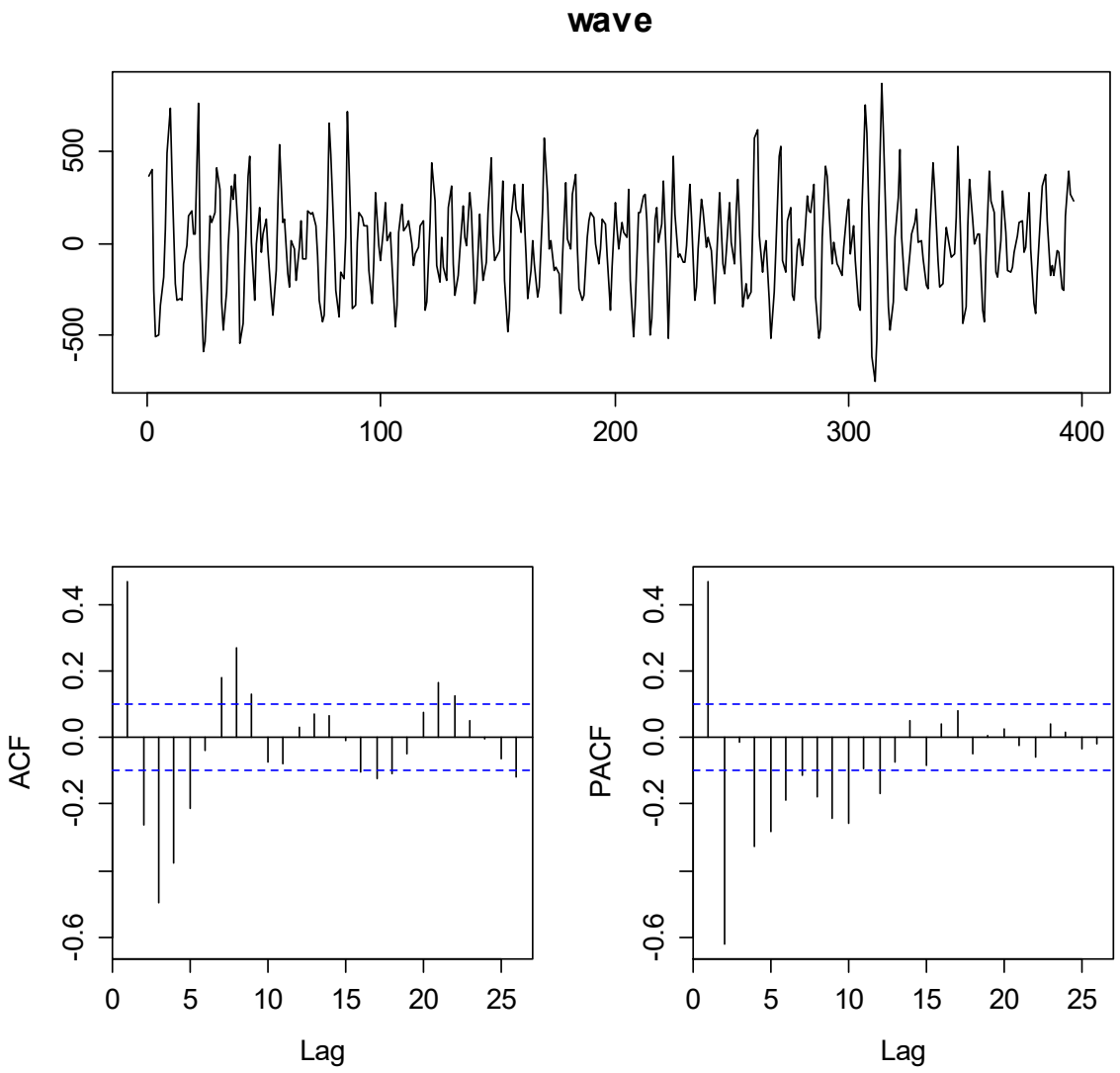
```
> pacf(wave, ylim=c(-1,1), main="PACF of Wave Tank Data")
```



We observe that $\hat{\pi}(1) \approx 0.5$ and $\hat{\pi}(2) \approx -0.6$. Some further PACF coefficients up to lag 10 seem significantly different from zero, but are smaller. From what we see here, we could try to describe the wave tank data with an $AR(2)$ model. The next section will explain why.

As a last remark in this chapter, we here introduce the `tsdisplay()` function from **R**'s `library (forecast)`. Using the default settings, it will show a time series plot along with both ACF and PACF. This will turn out to be very convenient when a model for a time series shall be found.

```
> library(forecast)
> tsdisplay(wave, points=FALSE)
```



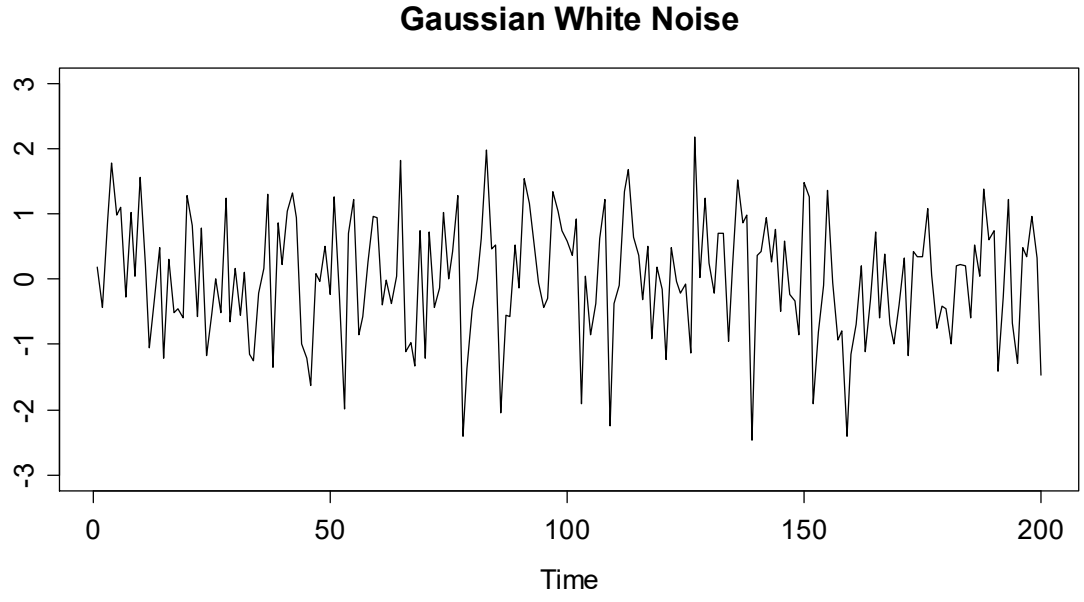
5 Stationary Time Series Models

Rather than simply describing observed time series data, we now aim for fitting time series models. This will prove useful for a deeper understanding of the data, but is especially beneficial when forecasting is the main goal. We here focus on parametric models for stationary time series, namely the broad class of *autoregressive moving average (ARMA) processes* – these have shown great importance in modeling real-world data.

5.1 White Noise

As the most basic stochastic process, we introduce discrete White Noise. A time series $(W_1, W_2, ..., W_n)$ is called *White Noise* if the random variables $W_1, W_2, ...$ are independent and identically distributed with mean zero. This also implies that all random variables W_t have identical variance, and there are no autocorrelations and partial autocorrelations either: $\rho(k) = 0$ and $\pi(k) = 0$ for all lags k. If in addition, the variables also follow a Gaussian distribution, i.e. $W_t \sim N(0, \sigma_W^2)$, the series is called *Gaussian White Noise*.

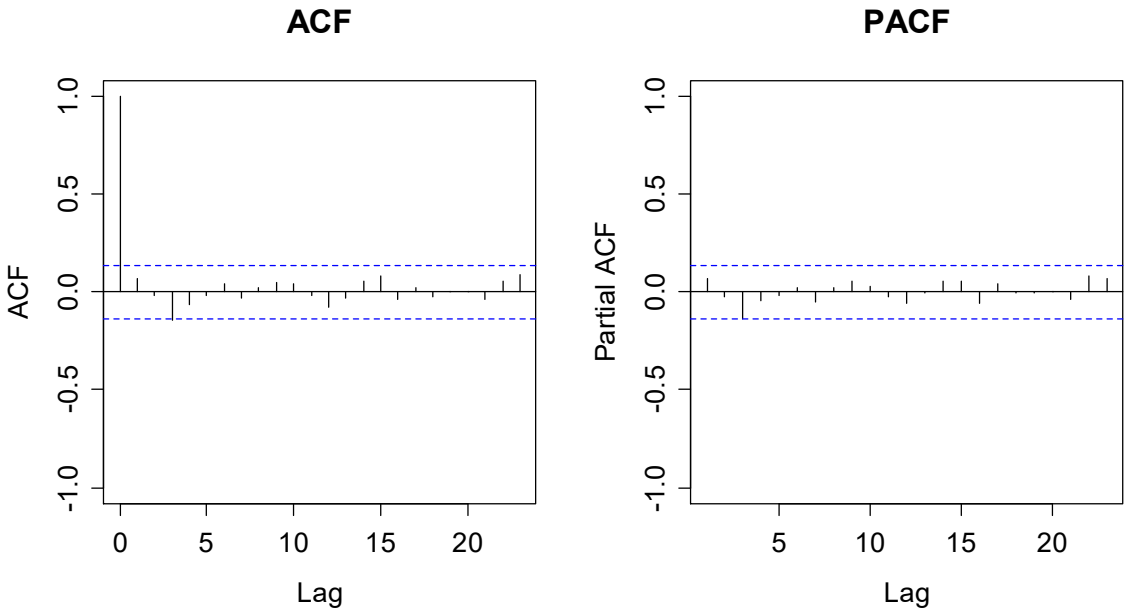
Before we show a realization of a White Noise process, we state that the term "White Noise" was coined in an article on heat radiation published in Nature in April 1922. There, it was used to refer to series time series that contained all frequencies in equal proportions, analogous to white light. It is possible to show that iid sequences of random variables do contain all spectral frequencies in equal proportions, and hence, here we are.



In **R**, it is easy to generate Gaussian White Noise, we just type:

```
> ts(rnorm(200, mean=0, sd=1))
```

Well, by giving more thought on how computers work, i.e. by relying on deterministic algorithms, it may seem implausible that they can really generate independent data. We do not embark into these discussions here, but treat the result of `rnorm()` as being "good enough" for a realization of a White Noise process. Here, we show ACF and PACF of the above series. As expected, there are no (strongly) significant estimates.



White Noise series are important, because they usually arise as residual series when fitting time series models. The correlogram generally provides enough evidence for attributing a series as White Noise, provided the series is of reasonable length – our studies in section 4.4 suggests that 100 or 200 is such a value. Please also note that while there is not much structure in Gaussian White Noise, it still has a parameter. It is the variance σ_W^2.

5.2 Estimating the Conditional Mean

Before we present some time series models, it is important to build some understanding of what we are actually doing. All the $AR(p)$, $MA(q)$ and $ARMA(p,q)$ models that will be presented below are based on the assumption that the time series can be written as:

$$X_t = \mu_t + E_t .$$

Hereby, μ_t is the conditional mean of the series, i.e. $\mu_t = E[X_t \mid X_{t-1}, X_{t-2}, ...]$ and E_t is a disturbance term. For all models in section 5, the disturbance term is assumed to be a White Noise innovation.

It is very important to notice that while stationary series feature a constant marginal expectation μ, the conditional mean μ_t is can be and often is non-constant and time-dependent. Or in other words, there is some short-term memory in the series. The $ARMA(p,q)$ processes that will be discussed here in this section are built on the following notion:

$$\mu_t = f(X_{t-1}, X_{t-2}, ..., X_{t-p}, E_{t-1}, E_{t-2}, ..., E_{t-q}).$$

In words, the conditional mean is a function of past instances of the series as well as past innovations. We will see that usually, a selection of the involved terms is made, and that the function $f(\cdot)$ is a linear combination of the arguments.

5.3 Autoregressive Models

5.3.1 Definition and Properties

The most natural formulation of a time series model is a linear regression approach on the past instances, i.e. a regression on the series itself. This coined the term *autoregressive*. In practice, such models prove to be very important; they are the most popular way of describing time series.

Model and Terms

An *autoregressive model of order* p, abbreviated as $AR(p)$, is based on a linear combination of past observations according to the following equation:

$$X_t = \alpha_1 X_{t-1} + \alpha_2 X_{t-2} + ... + \alpha_p X_{t-p} + E_t.$$

Hereby, the disturbance term E_t comes from a White Noise process, i.e. is iid. Moreover, we require that it is an *innovation*, i.e. that it is stochastically independent of $X_{t-1}, X_{t-2}, ...$. The term innovation is illustrative, because (under suitable conditions), it has the power to drive the series into a new direction, meaning that it is strong enough so that it can overplay the dependence of the series from its own past. An alternative notation for $AR(p)$ models is possible with the backshift operator:

$$(1 - \alpha_1 B - \alpha_2 B^2 - ... - \alpha_p B^p) X_t = E_t, \text{ or short } \Phi(B) X_t = E_t$$

Hereby, $\Phi(B)$ is called the *characteristic polynomial*. It determines all the relevant properties of the process. The most important questions that we will deal with in this chapter are of course the choice of the order p and the estimation of the coefficients $\alpha_1, ..., \alpha_p$. But first, a very important point:

- *$AR(p)$ models must only be fitted to stationary time series. Any potential trends and/or seasonal effects need to be removed first. We will also make sure that the $AR(p)$ processes are stationary.*

When is an $AR(p)$ stationary? Not always, but under some mild conditions. First of all, we study the unconditional expectation of an $AR(p)$ process X_t which we assume to be stationary, hence $E[X_t]=\mu$ for all t. When we take expectations on both sides of the model equation, we have:

$$\mu = E[X_t] = E[\alpha_1 X_{t-1} + ... + \alpha_p X_{t-p} + E_t] = (\alpha_1 + ... + \alpha_p)\cdot\mu + 0, \text{ hence } \mu = 0.$$

Thus, any stationary $AR(p)$ process has a global mean of zero. But please be aware of the fact that the conditional mean is time dependent and generally different from zero.

$$\mu_t = E[X_t \mid X_{t-1},...,X_{t-p}] = \alpha_1 x_{t-1} + ... + \alpha_p x_{t-p}$$

The question remains if $AR(p)$ processes are practically useful, because most of the real-word time series have a global mean μ that is different from zero. However, that generally poses little difficulties if we add an additional parameter m to the model definition:

$$Y_t = m + X_t$$

In that case, Y_t is a ***shifted*** $AR(p)$ ***process***, i.e. it has all dependency properties from an $AR(p)$, but its mean is different from zero. In fact, all R methodology that exists for fitting $AR(p)$'s assumes the process Y_t and thus estimates a global mean m unless this is explicitly excluded. In practice, if one colloquially speaks of an $AR(p)$, mostly one thinks of Y_t rather than X_t.

However, for the stationarity of an $AR(p)$, some further conditions on the model coefficients $\alpha_1,...,\alpha_p$ are required. The general derivation is quite complicated and will be omitted here. But for illustrative purpose, we assume a stationary $AR(1)$ which has $Var(X_t)=\sigma_X^2$ for all t. If we determine the centralized second moment on both sides of the model equation, we obtain:

$$\sigma_X^2 = Var(X_t) = Var(\alpha_1 X_{t-1} + E_t) = \alpha_1^2\sigma_X^2 + \sigma_E^2, \text{ hence } \sigma_X^2 = \frac{\sigma_E^2}{1-\alpha_1^2}.$$

From this we derive that an $AR(1)$ can only be stationary if $|\alpha_1|<1$. That limitation means that the dependence from the series' past must not be too strong, so that the memory fades out. If $|\alpha_1|>1$, the process diverges. The general condition for $AR(p)$ models is (as mentioned above) more difficult to derive. We require that:

> *The (potentially complex) roots of the characteristic polynomial* $\Phi(B)$ *must all exceed 1 in absolute value for an* $AR(p)$ *process to be stationary.*

In R, there is function `polyroot()` for finding a polynomials roots. If we want to verify whether an $AR(3)$ with $\alpha_1=0.4, \alpha_2=-0.2, \alpha_3=0.3$ is stationary, we type:

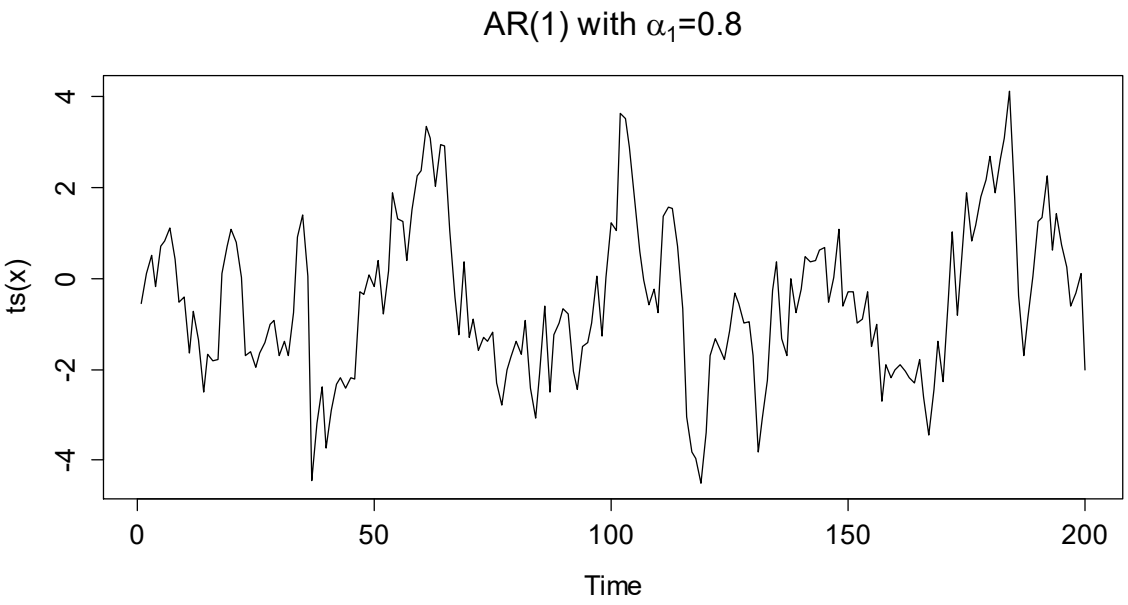
```
> abs(polyroot(c(1,-0.4,0.2,-0.3)))
[1] 1.405467 1.540030 1.540030
```

Thus, the $AR(3)$ we specified is stationary. We will proceed by studying the dependency in $AR(p)$ processes. For illustration, we first simulate from an $AR(1)$ with $\alpha_1 = 0.8$. The model equation is:

$$X_t = 0.8 \cdot X_{t-1} + E_t$$

So far, we had only required that E_t is a White Noise innovation, but not a distribution. We use the Gaussian here and set $x_1 = E_1$ as the starting value.

```
> set.seed(24)
> E    <- rnorm(200, 0, 1)
> x    <- numeric()
> x[1] <- E[1]
> for(i in 2:200) x[i] <- 0.8*x[i-1] + E[i]
> plot(ts(x), main= "AR(1) with...")
```



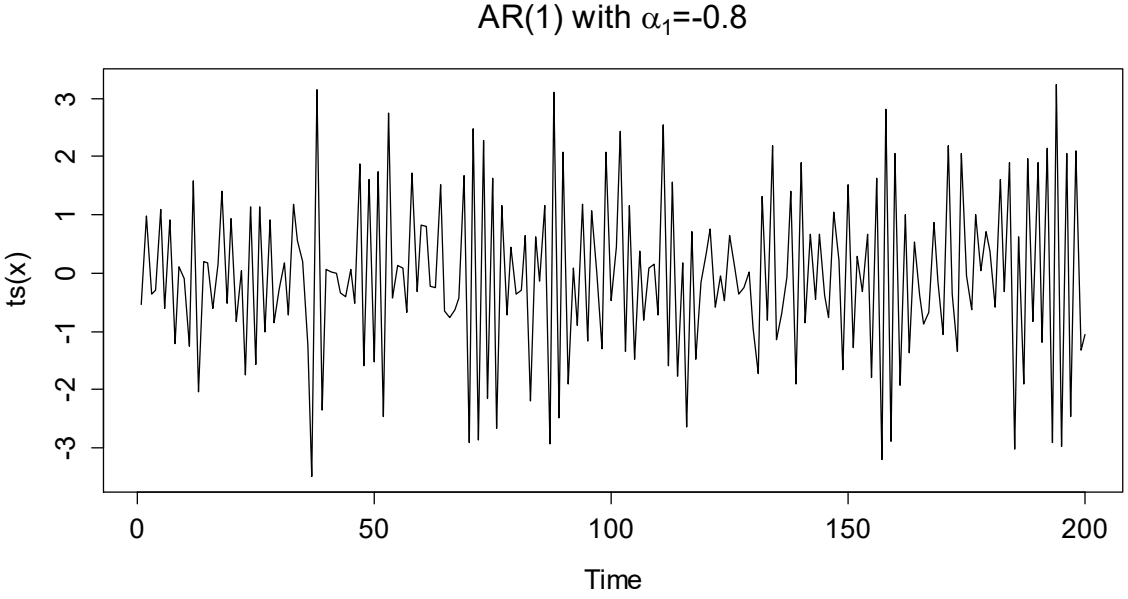
We observe some cycles with exclusively positive and others with only negative values. That is not surprising: if the series takes a large value, then the next one is determined as 0.8 times that large value plus the innovation. Thus, it is more likely that the following value has the same sign as its predecessor. On the other hand, the innovation is powerful enough so that jumping to the other side of the global mean is always an option. Given that behavior, it is evident that the autocorrelation at lag 1 is positive. We can compute it explicitly from the model equation:

$$\rho(1) = Cor(X_t, X_{t-1}) = \frac{Cov(X_t, X_{t-1})}{\gamma(0)} = \frac{Cov(\alpha_1 X_{t-1} + E_t, X_{t-1})}{\gamma(0)} = \frac{\alpha_1 \gamma(0) + 0}{\gamma(0)} = \alpha_1$$

Thus we have $\rho(1) = 0.8$ here, or in general $\rho(1) = \alpha_1$. The correlation for higher lags can be determined similarly by repeated plug-in of the model equation. It is:

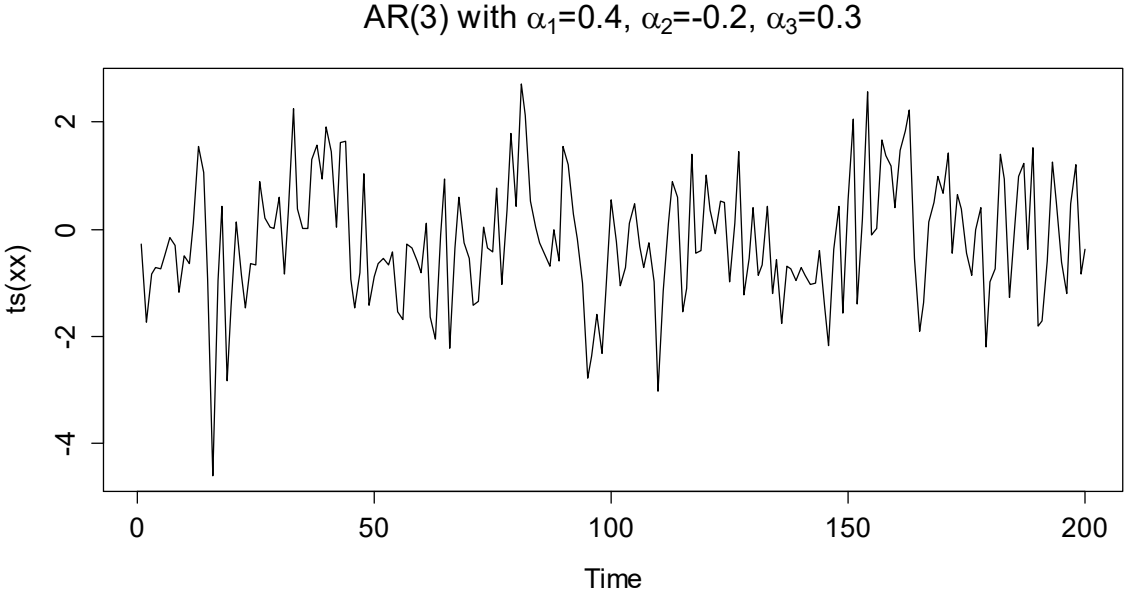
$$\rho(k) = \alpha_1^k .$$

Thus, for stationary $AR(1)$ series, we have an exponential decay of the autocorrelation coefficients. Of course, it is also allowed to have a negative value for α_1, as long as $|\alpha_1| < 1$. A realization of length 200 with $\alpha_1 = -0.8$ is as follows:



The series shows an alternating behavior: the next value is more likely to lie on the opposite side of the global mean zero, but there are exceptions when the innovation takes a large value. The autocorrelation still follows $\rho(k) = \alpha_1^k$. It is also alternating between positive and negative values with an envelope for $|\rho(k)|$ that is exponentially decaying.

We will now focus on appeareance and dependency of an $AR(3)$ (with the coefficients from above). While we could still program the simulation code by ourselves, it is more convenient to use function `arima.sim()`.

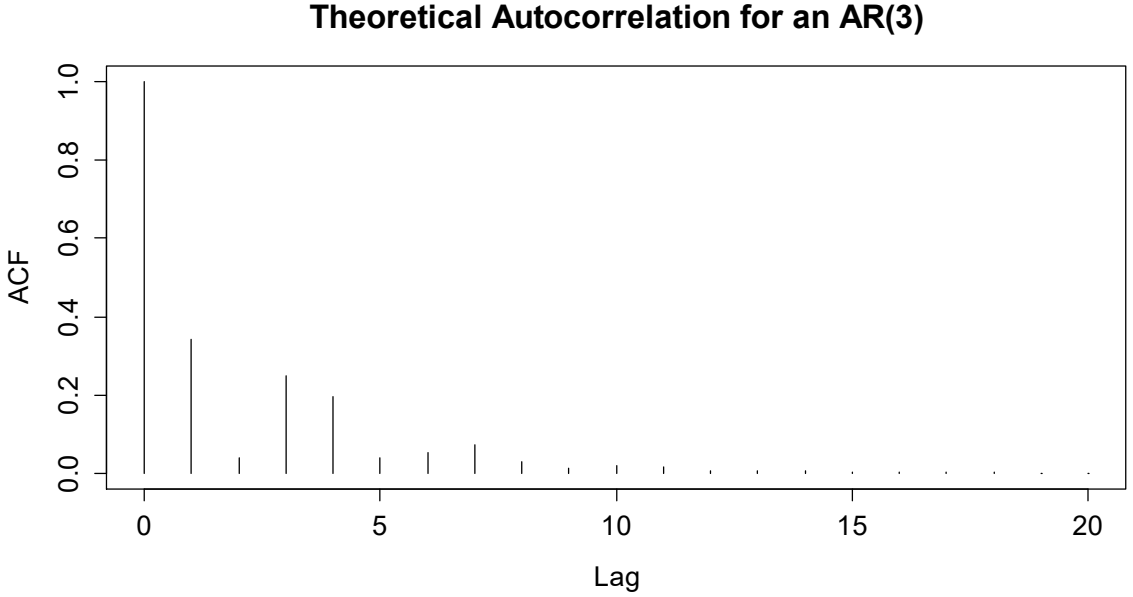


What is now the (theoretical) correlation in this $AR(3)$? We apply the standard trick of plugging-in the model equation. This yields:

$$\begin{aligned}\rho(k) &= \gamma(0)^{-1} \cdot Cov(X_{t+k}, X_t) \\ &= \gamma(0)^{-1} \cdot Cov(\alpha_1 X_{t+k-1} + ... + \alpha_p X_{t+k-p} + E_t, X_t) \\ &= \alpha_1 \rho(k-1) + ... + \alpha_p \rho(k-p)\end{aligned}$$

with $\rho(0) = 1$ and $\rho(-k) = \rho(k)$. For $k = 1,..., p$ this results in a $p \times p$ linear equation system called the *Yule-Walker equations*. It can be solved to obtain the autocorrelation coefficients which can finally be propagated for $k = p+1, p+2,...$. In R, there is function `armaACF()` that allows to determine the autocorrelation from autoregressive model coefficients.

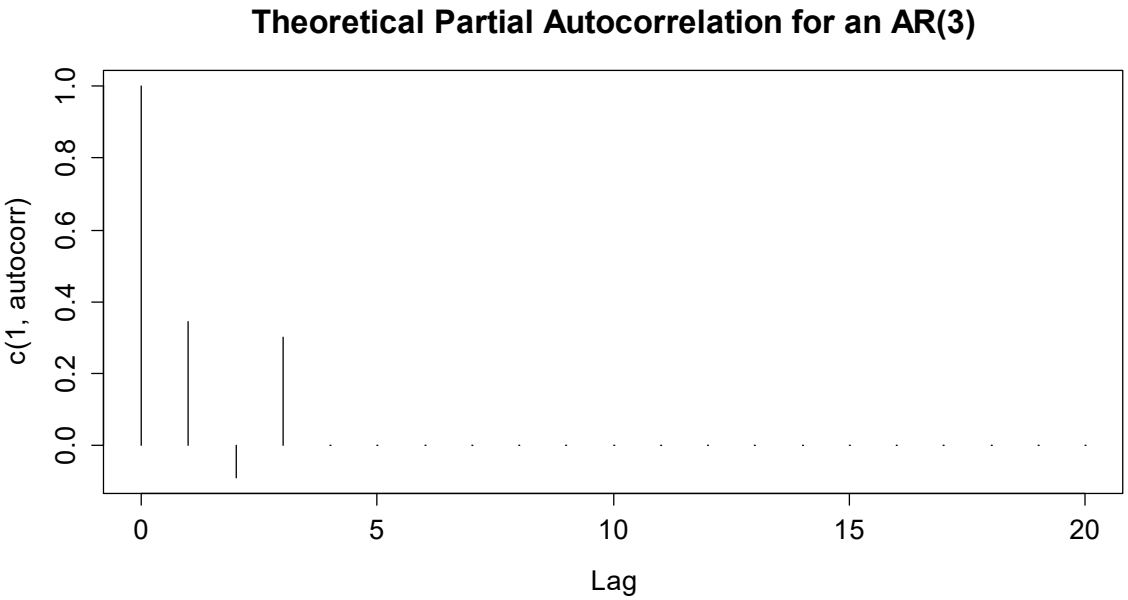
```
> autocorr <- ARMAacf(ar=c(0.4, -0.2, 0.3), lag.max=20)
> plot(0:20, autocorr, type="h", xlab="Lag")
```



We observe that the theoretical correlogram shows a more complex structure than what could be achieved with an $AR(1)$. Nevertheless, one can still find an exponentially decaying envelope for the magnitude of the autocorrelations $|\rho(k)|$. That is a property which is common to all $AR(p)$ models.

From the above, we can conclude that the autocorrelations are generally non-zero for all lags, even though in the underlying model, X_t only depends on the p previous values $X_{t-1},...,X_{t-p}$. In section 4.5 we learned that the partial autocorrelation at lag k illustrates the dependence between X_t and X_{t+k} when the linear dependence on the intermittent terms was already taken into account. It is evident by definition that for any $AR(p)$ process, we have $\pi(k) = 0$ for all $k > p$. This can and will serve as a useful indicator for deciding on the model order p if we are trying to identify the suitable model order when fitting real world data. In this section, we focus on the PACF for the above $AR(3)$.

```
> autocorr <- ARMAacf(ar=..., pacf=TRUE, lag.max=20)
> plot(0:20, autocorr, type="h", xlab="Lag")
```



As claimed previously, we indeed observe $\rho(1) = \pi(1) = 0.343$ and $\pi(3) = \alpha_3 = 0.3$. All partial autocorrelations from $\pi(4)$ on are exactly zero.

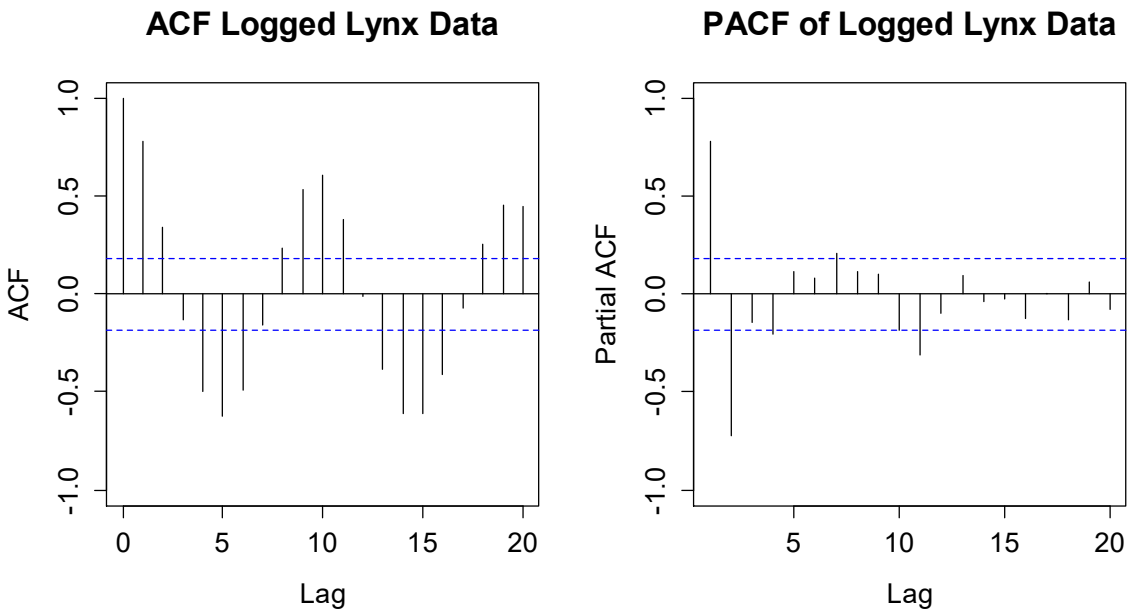
5.3.2 Fitting

Fitting an $AR(p)$ model to data involves three main steps. First, the model and its order need to be identified. Second, the model parameters need to be estimated and third, the quality of the fitted model needs to be verified by residual analysis.

Model Identification

The model identification step first requires verifying that the data show properties which make it plausible that they were generated from an $AR(p)$ process. In particular, the time series we are aiming to model needs to be stationary, show an ACF with approximately exponentially decaying envelope and a PACF with a recognizable cut-off at some lag p smaller than about $5-10$. If any of these three properties is strongly violated, it is unlikely that an $AR(p)$ will yield a satisfactory fit, and there might be models which are better suited for the problem at hand.

The choice of the model order p then relies on the analysis of the sample PACF. Following the paradigm of parameter parsimony, we would first try the simplest model that seems plausible. This means choosing the smallest p at which we suspect a cut-off, i.e. the smallest after which none, or only few and weakly significant partial autocorrelations follow. We illustrate the concept with the logged Lynx data that were already discussed in section 1.2.2. We need to generate both ACF and PACF, which can be found on the next page.



There is no reason to doubt the stationarity of the Lynx series. Moreover, the ACF shows a cyclic behavior that has an exponentially decaying envelope. Now does the PACF show a cut-off? That is a bit less clear, and several orders $p \ (=2,4,7,11)$ come into question. However in summary, we conjecture that there are no strong objections against fitting an $AR(p)$. The choice of the order is debatable, but the parsimony paradigm tells us that we should try with the smallest candidate first, and that is $p=2$.

Parameter Estimation

Observed time series are rarely centered and thus, it is usually inappropriate to fit a pure $AR(p)$ process. In fact, all **R** routines for fitting autoregressive models by default assume the shifted process $Y_t = m + X_t$. Hence, we have a regression-type equation with observations:

$$(Y_t - m) = \alpha_1(Y_{t-1} - m) + ... + \alpha_p(Y_{t-p} - m) + E_t \text{ for } t = p+1,...,n.$$

The goal here is to estimate the parameters $m, \alpha_1, ..., \alpha_p$ such that the data are *fitted well*. There are several concepts that define *well fitting*. These include ordinary least squares estimation (OLS), Burg's algorithm (Burg), the Yule-Walker approach (YW) and maximum likelihood estimation (MLE). Already at this point we note that while the four methods have fundamental individuality, they are asymptotically equivalent (under some mild assumptions) and yield results that mostly only differ slightly in practice. Still, it is worthwhile to study all the concepts.

OLS

The OLS approach is based on the notion with the centering; the above equation defines a multiple linear regression problem without intercept. The goodness-of-fit criterion is $(x_t - \hat{x}_t)^2$ resp. $(y_t - \hat{y}_t)^2$, the two quantities are equal. The first step with this approach is to center the data, which is based on subtracting the global mean:

Estimate $\hat{m} = \bar{y} = \sum_{t=1}^{n} y_t$ and then compute $x_t = y_t - \hat{m}$ for all $t = 1,...,n$.

On the x_t, an OLS (auto)regression without intercept is performed. Note that this regression is (technically) conditional on the first p observations $x_1,...,x_p$, which are only used as predictors, but not as response terms. In other words, the goodness-of-fit of the model is only evaluated for the last $n-p$ observations. The following code chunk implements the procedure for the logged lynx data:

```
> llc     <- log(lynx)-mean(log(lynx))
> resp    <- llc[3:114]
> pred1   <- llc[2:113]
> pred2   <- llc[1:112]
> fit.ols <- lm(resp ~ -1 + pred1 + pred2)
> summary(fit.ols)

Coefficients:
      Estimate Std. Error t value Pr(>|t|)
pred1  1.38435    0.06359   21.77   <2e-16 ***
pred2 -0.74793    0.06364  -11.75   <2e-16 ***
---
Residual standard error: 0.528 on 110 degrees of freedom
Multiple R-squared: 0.8341,  Adjusted R-squared: 0.8311
F-statistic: 276.5 on 2 and 110 DF,  p-value: < 2.2e-16
```

We can extract $\hat{m} = 6.686$, $\hat{\alpha}_1 = 1.384$, $\hat{\alpha}_2 = -0.748$ and $\hat{\sigma}_E = 0.528$. But while this is an instructive way of estimating $AR(p)$ models, it is a bit cumbersome and time consuming. Not surprisingly, there are procedures that are dedicated to fitting such models in R. We here display the use of function `ar.ols()`. To replicate the hand-produced result, we type:

```
> f.ar.ols <- ar.ols(log(lynx), aic=F, intercept=F, order=2)
> f.ar.ols

Coefficients:
      1        2
 1.3844  -0.7479

Order selected 2  sigma^2 estimated as  0.2738
```

Note that for producing the result, we need to avoid AIC-based model fitting with `aic=FALSE`. The shift m is automatically estimated, and thus we need to exclude an intercept term in the regression model using `intercept=FALSE`. We observe that the estimated AR-coefficients $\hat{\alpha}_1, \hat{\alpha}_2$ take exactly the same values as with the hand-woven procedure above. The estimated shift $\hat{m}$ can be extracted via

```
> fit.ar.ols$x.mean
[1] 6.685933
```

and corresponds to the global mean of the series. Finally, the estimate for the innovation variance requires some prudence. The `lm()` summary output yields an

estimate of σ_E that was computed as $RSS/(n-p)$, whereas the value in the `ar.ols()` output is an estimate of σ_E^2 that was computed as RSS/n. The former is intended to be an unbiased estimate (though it should use the denominator $n-p-1$ due to the estimation of the shift m), and the latter is the MLE-estimator for the innovation variance. In practice, the numerical difference between the two is neglectable for any series that has reasonable length for fitting an AR model.

```
> sum(na.omit(fit.ar.ols$resid)^2)/112
[1] 0.2737594
```

Burg's Algorithm

While the OLS approach works, its downside is the asymmetry: the first p terms are never evaluated as responses. That is cured by Burg's Algorithm, an alternative approach for estimating $AR(p)$ models. It is based on the notion that any $AR(p)$ process is also an $AR(p)$ if the time is run in reverse order. Under this property, minimizing the forward and backward 1-step squared prediction errors makes sense:

$$\sum_{t=p+1}^{n}\left\{\left(X_t-\sum_{k=1}^{p}\alpha_k X_{t-k}\right)^2+\left(X_{t-p}-\sum_{k=1}^{p}\alpha_k X_{t-p+k}\right)^2\right\}$$

In contrast to OLS, there is no explicit solution and numerical optimization is required. This is done with a recursive method called the Durbin-Levison algorithm. We do not explain its details here, but refer to the R implementation `ar.burg()`.

```
> f.ar.burg <- ar.burg(log(lynx), aic=FALSE, order.max=2)
> f.ar.burg

Call:
ar.burg.default(x = log(lynx), aic = FALSE, order.max = 2)

Coefficients:
      1        2
 1.3831  -0.7461

Order selected 2  sigma^2 estimated as  0.2707

> f.ar.burg$x.mean
[1] 6.685933
> sum(na.omit(f.ar.burg$resid)^2)/112
[1] 0.2737614
```

There are a few interesting points which require commenting. First and foremost, Burg's algorithm also uses the arithmetic mean to estimate the global mean $\hat{m}$. The fitting procedure is then done on the centered observations x_t. On a side remark, note that assuming centered observations is possible. If argument `demean=FALSE` is set, the global mean is assumed to be zero and not estimated.

The two coefficients $\hat{\alpha}_1, \hat{\alpha}_2$ take some slightly different values than with OLS estimation. While often, the difference between the two methods is practically neglectable, it is nowadays generally accepted that the Burg solution is better for finite samples. Asymptotically, the two methods are equivalent. Finally, we observe that the `ar.burg()` output specifies $\hat{\sigma}_E^2 = 0.2707$. This is different from the MLE estimate of 0.27376 on the residuals. The explanation is that for Burg's Algorithm, the innovation variance is estimated from the Durbin-Levinson updates; see the R help file for further reference.

Yule-Walker Equations

A third option for estimating $AR(p)$ models is to plugging-in the sample ACF into the Yule-Walker equations. In section 5.3.1 we had learned that there is a $p \times p$ linear equation system $\left|\rho(k) = \alpha_1 \rho(k-1) + ... + \alpha_p \rho(k-p)\right|$ for $k = 1, ..., p$. Hence we can and will explicitly determine $\hat{\rho}(0), ..., \hat{\rho}(k)$ and then solve the linear equation system for the coefficients $\alpha_1, ..., \alpha_p$. The procedure is implemented in R function `ar.yw()`.

```
> f.ar.yw <- ar.yw(log(lynx), aic=FALSE, order.max=2)
> f.ar.yw

Call: ar.yw.default(x=log(lynx), aic=FALSE, order.max=2)

Coefficients:
      1        2
 1.3504  -0.7200

Order selected 2  sigma^2 estimated as  0.3109
```

Again, the two coefficients $\hat{\alpha}_1, \hat{\alpha}_2$ take some slightly different values than compared to the two methods before. Mostly this difference is practically neglectable and Yule-Walker is asymptotically equivalent to OLS and Burg. Nevertheless, for finite samples, the estimates from the Yule-Walker method are often worse in the sense that their (Gaussian) likelihood is lower. Thus, we recommend preferring Burg's algorithm. We conclude this section by noting that the Yule-Walker method also involves estimating the global mean m with the arithmetic mean as the first step. The innovation variance is estimated from the fitted coefficients and the autocovariance of the series and thus again takes a different value than before.

Maximum-Likelihood Estimation (MLE)

The MLE is based on determining the model coefficients such that the likelihood given the data is maximized, i.e. the density function takes its maximal value under the present observations. This requires assuming a distribution for the $AR(p)$ process, which comes quite naturally if one assumes that for the innovations, we have $E_t \sim N(0, \sigma_E^2)$, i.e. they are *iid* Gaussian random variables. With some theory (which we omit), one can then show that an $AR(p)$ process $X_1, ..., X_n$ is a random vector with a multivariate Gaussian distribution.

MLE then provides a simultaneous estimation of the shift m, the innovation variance σ_E^2 and the model coefficients $\alpha_1,...,\alpha_p$. The criterion that is optimized can, in a simplified version, be written as:

$$L(\alpha, m, \sigma_E^2) \propto \exp\left(\sum_{t=1}^{n}(x_t - \hat{x}_t)^2\right)$$

The details are quite complex and several constants are part of the equation, too. But we here note that the MLE derived from the Gaussian distribution is based on minimizing the sum of squared errors and thus equivalent to the OLS approach. Due to the simultaneous estimation of model parameters and innovation variance, a recursive algorithm is required. There is an implementation in R:

```
> f.ar.mle
Call: arima(x = log(lynx), order = c(2, 0, 0))

Coefficients:
         ar1      ar2  intercept
      1.3776  -0.7399     6.6863
s.e.  0.0614   0.0612     0.1349

sigma^2 = 0.2708:  log likelihood = -88.58,  aic = 185.15
```

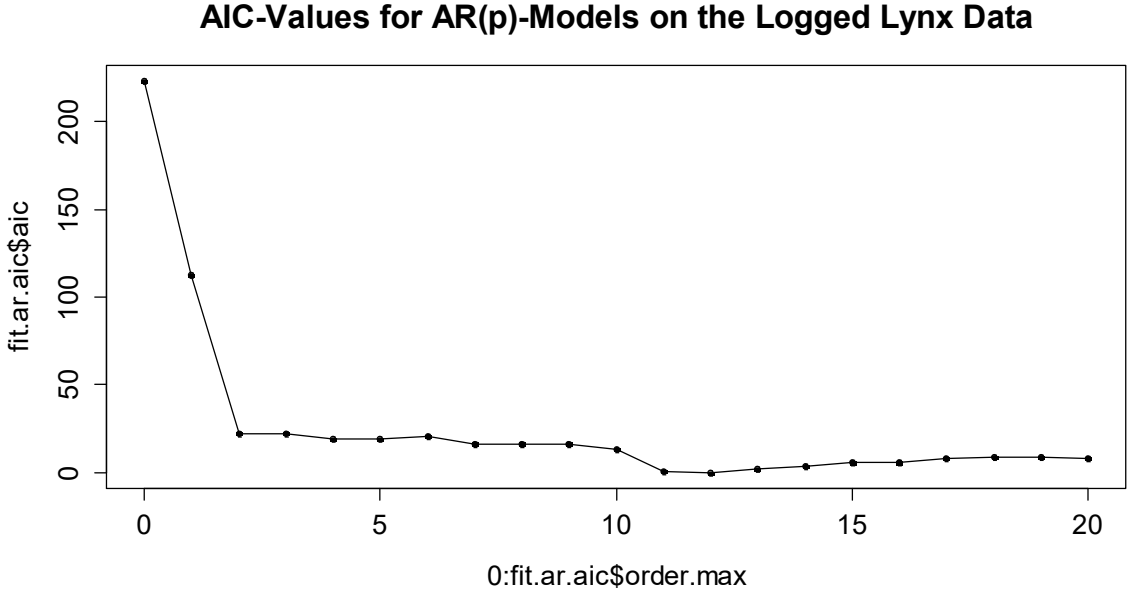
We observe estimates which are again slightly different from the ones computed previously. Again, those differences are mostly neglectable for practical data analysis. What is known from theory is that the MLE is (under mild assumptions) asymptotically normal with minimum variance among all asymptotically normal estimators. Note that the MLE based on Gaussian distribution still works reasonably well if that assumption is not met, as long as we do not have strongly skewed data (apply a transformation in that case) or extreme outliers.

Practical Aspects

We presented four different methods for fitting $AR(p)$ models. How to make a choice in practice? We explained that all methods are asymptotically equivalent and even on finite samples; the differences among them are little. Also, all methods are non-robust with respect to outliers and perform best on data which are approximately Gaussian. There is one practical aspect linked to the fitting routines that are available in R, though. Function `arima()` yields standard errors for m and $\alpha_1,...,\alpha_p$. Approximate 95% confidence intervals can be obtained by taking the point estimate +/- twice the standard error. Hence, statements about the significance of the estimates can be made, and a confidence interval for the mean is much more easily constructed as by the procedure describen in section 4.4.5.

On the other hand, `ar.ols()`, `ar.yw()` und `ar.burg()` do not provide standard errors, but allow for convenient determination of the model order p with the AIC statistic. While we still recommend investigating on the suitable order by analyzing ACF and PACF, the parsimonity paradigm and inspecting residual plots, using AIC as a second opinion is still recommended. It works as follows:

```
> fit.aic <- ar.burg(log(lynx))
> plot(0:fit.aic$order.max, fit.aic$aic)
```



We observe that already $p = 2$ yields a good AIC value. Then there is little further improvement until $p = 11$, and a just slightly lower value is found at $p = 12$. Hence, we will evaluate $p = 2$ and $p = 11$ as two competing models with some further tools in the next section.

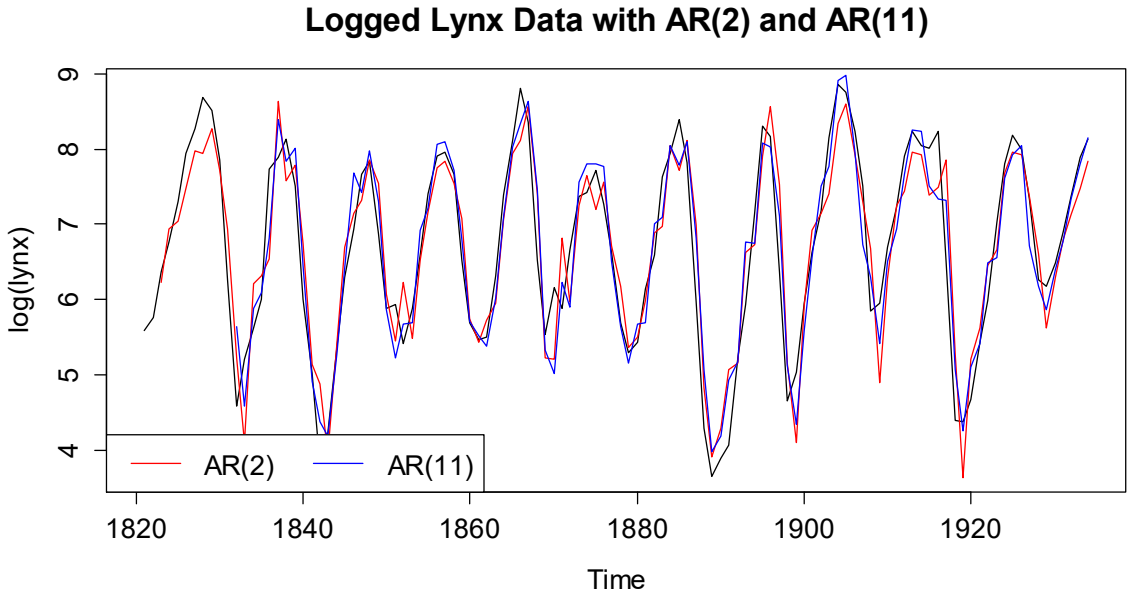
5.3.3 Residual Analysis

When comparing different models, a simple approach is to plot the original series along with the fitted model values. However, one has to keep in mind that this is an insample analysis, i.e. the bigger model has an advantage which does not necessarily persist once out-of-sample data are analyzed. Please note that the residuals are estimates of the innovations E_t. Thus, a good model yields residuals that resemble a White Noise process. We require mean zero, constant variance and no autocorrelation. If these properties are not met, the model is not adequate.

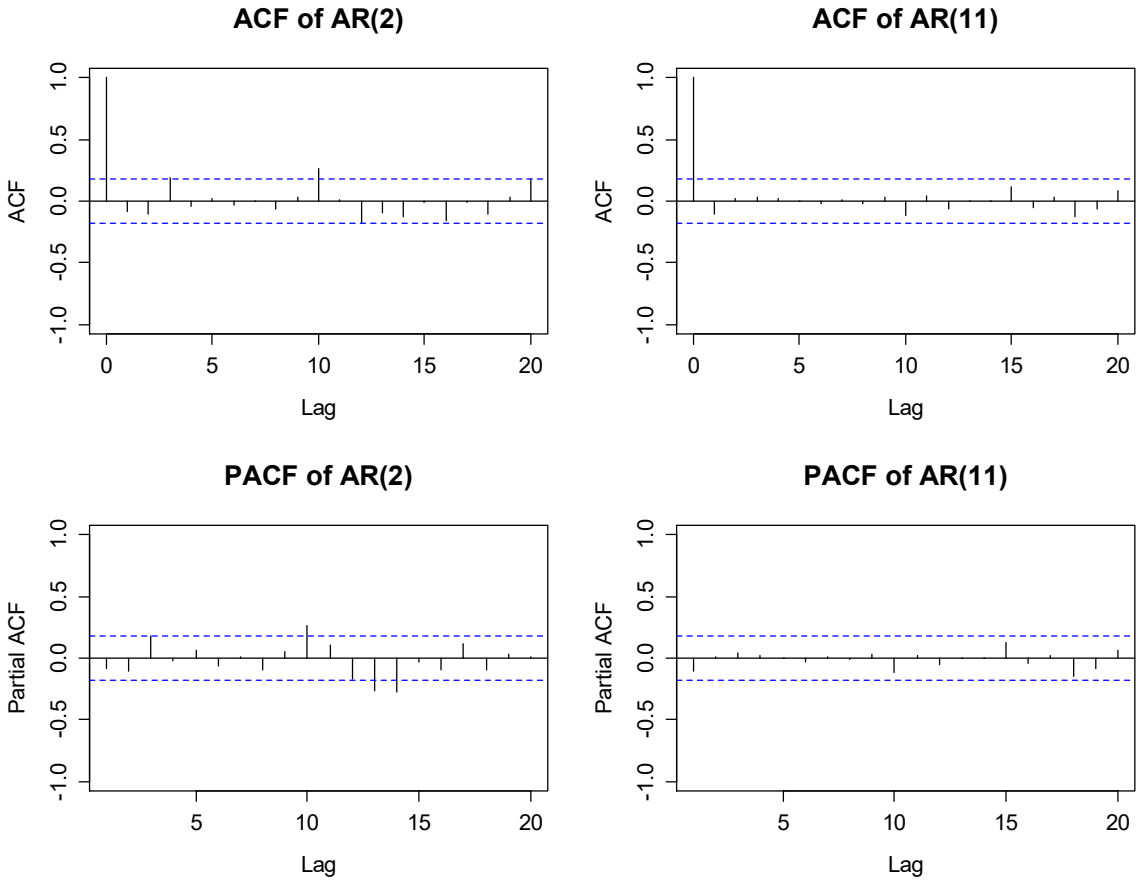
```
> fit.ar02 <- ar.burg(log(lynx), aic=FALSE, order.max=2)
> fit.ar11 <- ar.burg(log(lynx), aic=FALSE, order.max=11)
> plot(log(lynx), main="Logged Lynx Data with ...")
> lines(log(lynx)-fit.ar02$resid, col="red")
> lines(log(lynx)-fit.ar11$resid, col="blue")
```

The output is displayed on the next page. While some differences are visible, it is not easy to judge from the fitted values, which of the two models is preferable. A better focus on the quality of the fit is obtained when the residuals and their dependance are inspected with time series plots as well as ACF/PACF correlograms. The graphical output is again displayed on the next page. We observe that the $AR(2)$ residuals are not iid. Hence they do not form a White Noise process

and thus, we conclude that the $AR(11)$ model yields a better description of the logged lynx data.

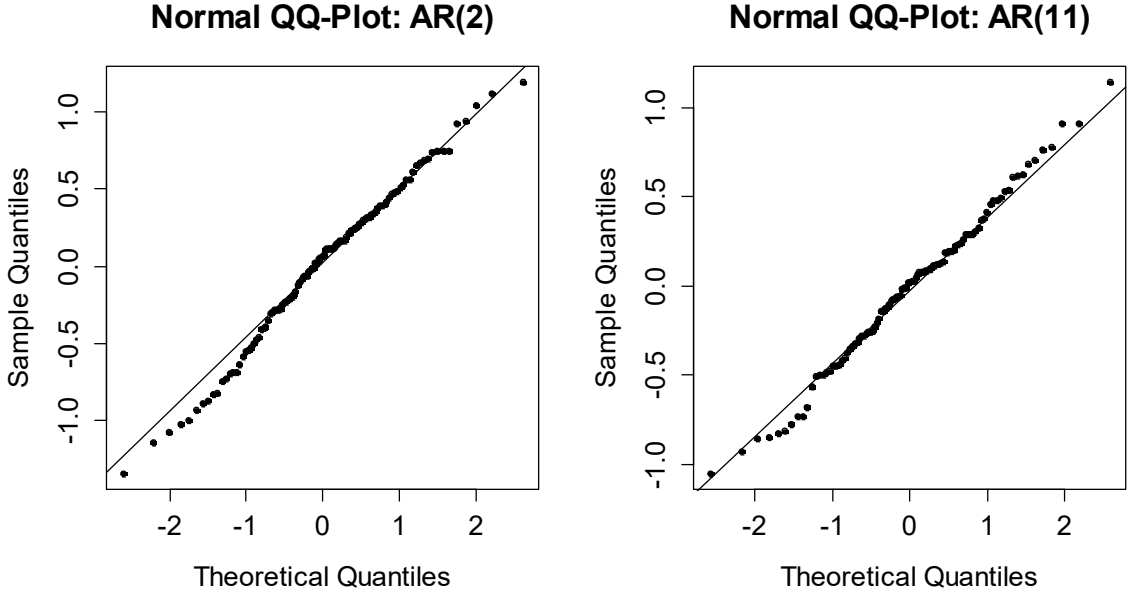



```
> acf(fit.ar02$resid, na.action=na.pass, ylim=c(-1,1))
> pacf(fit.ar02$resid, na.action=na.pass, ylim=c(-1,1))
> acf(fit.ar11$resid, na.action=na.pass, ylim=c(-1,1))
> pacf(fit.ar11$resid, na.action=na.pass, ylim=c(-1,1))
```



Because our estimation routines to some extent rely on the Gaussian distribution, it is always worthwhile to generate a Normal QQ-Plot for verification. We obtain:

```
> par(mfrow=c(1,2))
> qqnorm(as.numeric(fit.ar02$resid))
> qqline(as.numeric(fit.ar02$resid))
> qqnorm(as.numeric(fit.ar11$resid))
> qqline(as.numeric(fit.ar11$resid))
```

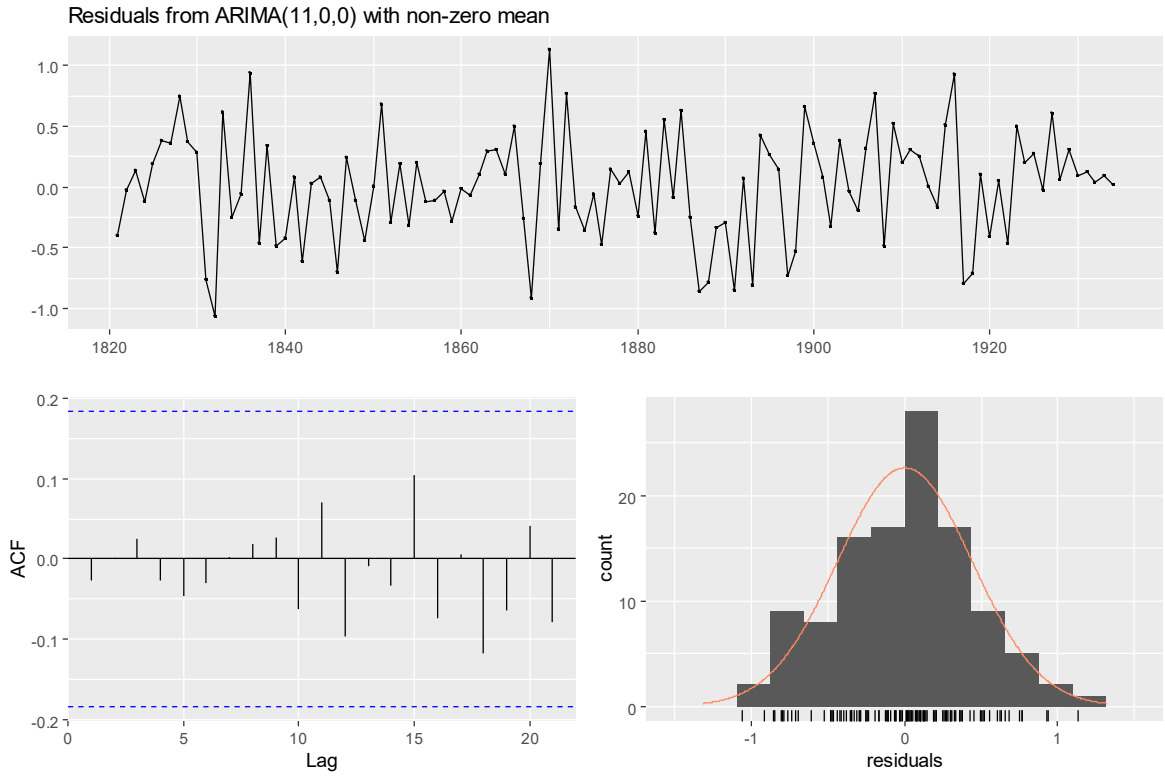


We observe that in the left plot from the AR(2) model, negative residuals seem to prevail, i.e. their distribution is skewed. This further indicates that this model may not be appropriate. The distribution of residuals is more symmetrical in the right panel, the assumption of normally distributed innovations seems justified. In summary, if the distribution of residuals is distinctly non-normal, improving the model is mandatory. Typical ways of action include transforming the data with either the log or Box-Cox transformations or changing model order or type.

The `checkresiduals()` Function

In library(forecast) there is the checkresiduals() function. It provides both a graphical and a text output. The former involves the time series plot of the residuals, their ACF correlogram plus a histogram of residuals. This is very similar than what was suggested above. In the text output, the result of a Ljung-Box test for correlation among the residuals is printed. Please note that this only works if fitting was done with function `arima()`.

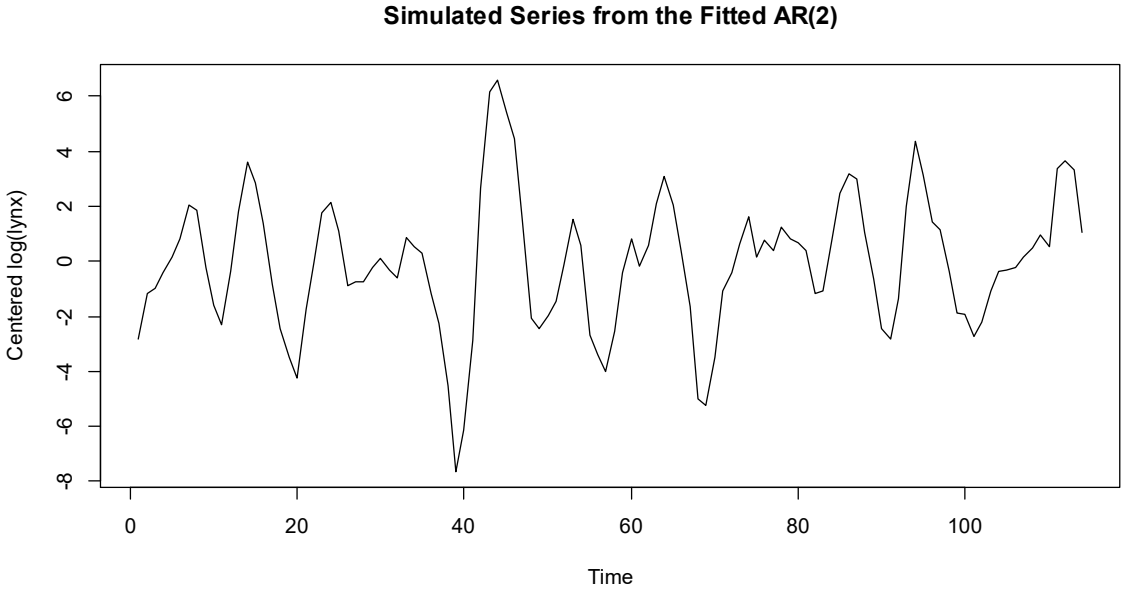
```
> f.arima <- arima(log(lynx), c(11,0,0))
> checkresiduals(f.arima)
Ljung-Box test
data:  Residuals from ARIMA(11,0,0) with non-zero mean
Q* = 4.7344, df = 3, p-value = 0.1923
Model df: 12.   Total lags used: 15
```

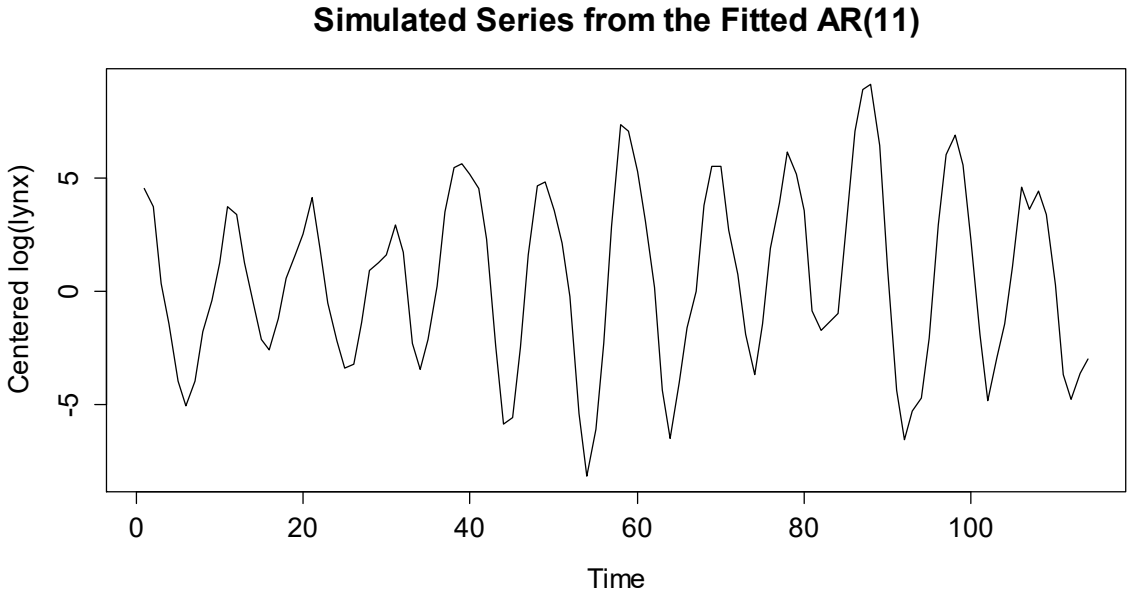


Simulation from the Fitted Model

If there are competing models and none of the other criterions dictate which one to use, another option is to generate realizations from the fitted process using R's function `arima.sim()`. It usually pays off to generate multiple realizations from each fitted model. By eyeballing, one then tries to judge which model yields data that resemble the true observations best. We here do the following:

```
> ## Repeat these commands a number of times
> plot(arima.sim(n=114, list(ar=fit.ar02$ar)))
> plot(arima.sim(n=114, list(ar=fit.ar11$ar)))
```





In summary, the simulations from this bigger model look more realistic than the ones from the $AR(2)$. The clearest answer about the model which is preferable here comes from the ACF/PACF correlograms, though. We conclude this section about model fitting by saying that the logged lynx data are best modeled with the $AR(11)$.

5.4 Moving Average Models

Here, we discuss moving average models. They are an extension of the White Noise process, i.e. X_t is as a linear combination of the current plus a few of the most recent innovation terms. As we will see, this leads to a time series process that is always stationary, but not iid. Furthermore, we will see that in many respects, moving average models are complementary to autoregressive models.

5.4.1 Definition and Properties

As we had mentioned above, a *moving average process of order* q, or abbreviated, an $MA(q)$ model for a series X_t is a linear combination of the current innovation term E_t, plus the q most recent ones $E_{t-1},...,E_{t-q}$. The model equation is:

$$X_t = E_t + \beta_1 \cdot E_{t-1} + \ldots + \beta_q \cdot E_{t-q}$$

We require that E_t is an innovation, which means independent and identically distributed, and also independent of any X_s where $s < t$. For simple notation, we can make use of the backshift operator and rewrite the model:

$$X_t = (1 + \beta_1 B + \ldots + \beta_q B^q) E_t = \Theta(B) E_t$$

We call $\Theta(B)$ the characteristic polynomial of the $MA(q)$ process and obviously, it defines all properties of the series. As a remark, please note that a number of textbooks define the $MA(q)$ process with negative signs for the β_j. While this is mathematically equivalent, we prefer our notation with the '+' signs, as it matches the way how things are implemented in **R**. We turn our sights towards the motivation for the moving average process.

What is the rationale for the $MA(q)$ process?

Firstly, they have been applied successfully in many applied fields, particularly in econometrics. Time series such as economic indicators are affected by a variety of random events such as strikes, government decisions, referendums, shortages of key commodities, et cetera. Such events will not only have an immediate effect on the indicator, but may also affect its value (to a lesser extent) in several of the consecutive periods. Thus, it is plausible that moving average processes appear in practice. Moreover, some of their theoretical properties are in a nice way complementary to the ones of autoregressive processes. This will become clear if we study the moments and stationarity of the MA process.

Moments and Dependence

A first, straightforward but very important result is that any $MA(q)$ process X_t, as a linear combination of innovation terms, has zero mean and constant variance:

$E[X_t] = 0$ for all t, and $Var(X_t) = \sigma_E^2 \cdot \left(1 + \sum_{j=1}^{q} \beta_j^2\right) = const$

In practice, we can always enhance $MA(q)$'s by adding a constant m that accounts for a non-zero mean, i.e. we can consider the shifted $MA(q)$ process

$$Y_t = m + X_t .$$

Hence, the zero mean property does not affect the possible field of practical application. Now, if we could additionally show that the autocovariance in MA processes is independent of the time t, we had already proven their stationarity. This is indeed the case. We start by considering a $MA(1)$ with $X_t = E_t + \beta_1 \cdot E_{t-1}$

$$\gamma(1) = Cov(X_t, X_{t-1}) = Cov(E_t + \beta_1 E_{t-1}, E_{t-1} + \beta_1 E_{t-2}) = \beta_1 \sigma_E^2 .$$

For any lag k exceeding the order $q = 1$, we use the same trick of plugging-in the model equation and directly obtain a perhaps somewhat surprising result:

$$\gamma(k) = Cov(X_t, X_{t-k}) = 0 \text{ for all } k > q = 1 .$$

Thus, there is no more unconditional serial dependence in lags > 1. For the autocorrelation of a $MA(1)$ process, we have:

$$\rho(1) = \frac{\gamma(1)}{\gamma(0)} = \frac{\beta_1}{1 + \beta_1^2} \text{ and } \rho(k) = 0 \text{ for all } k > q = 1 .$$

From this we conclude that $\rho(1) \leq 0.5$, no matter what the choice for β_1 is. Thus if in practice we observe a series where the first-order autocorrelation coefficient clearly exceeds this value, we have counterevidence to a $MA(1)$ process. Furthermore, we have shown that any $MA(1)$ has zero mean, constant variance and an ACF that only depends on the lag k, hence it is stationary. Note that the stationarity does (in contrast to AR processes) not depend on the choice of the parameter β_1. The stationarity property can be generalized to $MA(q)$ processes. Using some calculations and $\beta_0 = 1$, we obtain:

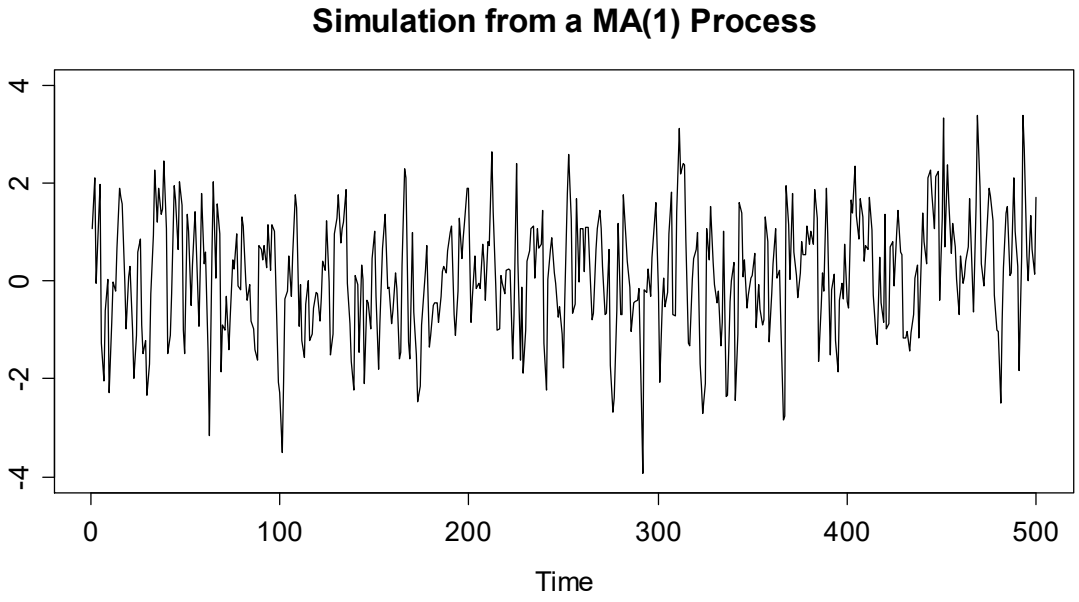
$$\rho(k) = \begin{cases} \sum_{j=0}^{q-k} \beta_j \beta_{j+k} \Big/ \sum_{j=0}^{q} \beta_j^2 & for \quad k = 1, ..., q \\ 0 & for \quad k > q \end{cases}$$

Hence, $\rho(k)$ is independent of the time t for any $MA(q)$ process, irrespective of the order q. The main results which can be derived from this property is that $MA(q)$ processes are *always stationary, independent of* $\beta_1, ..., \beta_q$. Moreover, we learn from the above that the autocorrelation is zero for all orders $k > q$. And there obviously is a relation between the model parameters and the autocorrelation, although it gets quite complex for higher orders. While this formula may be employed for finding the true ACF of a given $MA(q)$, the most convenient way of doing this in practice remains with the **R** function `ARMAacf()`.

Example of a $MA(1)$

For illustration, we generate a realization consisting of 500 observations, from a $MA(1)$ process with $\beta_1 = 0.7$, and display a time series plot, along with both estimated and true ACF/PACF.

```
> set.seed(21)
> ts.ma1 <- arima.sim(list(ma=0.7), n=500)
> plot(ts.ma1, ylab="", ylim=c(-4,4), main="…")
> title("Simulation from a MA(1) Process")
```




```
> acf.true  <- ARMAacf(ma=0.7, lag.max=20)
> pacf.true <- ARMAacf(ma=0.7, pacf=TRUE, lag.max=20)
```

We observe (see next page) that the estimates are pretty accurate: the ACF has a clear cut-off, whereas the PACF seems to feature some alternating behavior with an exponential decay in absolute value. This behavior is typical: the PACF of any $MA(q)$ process shows an exponential decay, while the ACF has a cut-off. In this respect, $MA(q)$ processes are in full contrast to the $AR(p)$'s, i.e. the appearance of ACF and PACF is swapped.

Invertibility

It is easy to show that the first autocorrelation coefficient $\rho(1)$ of an $MA(1)$ process can be written in standard form, or also as follows:

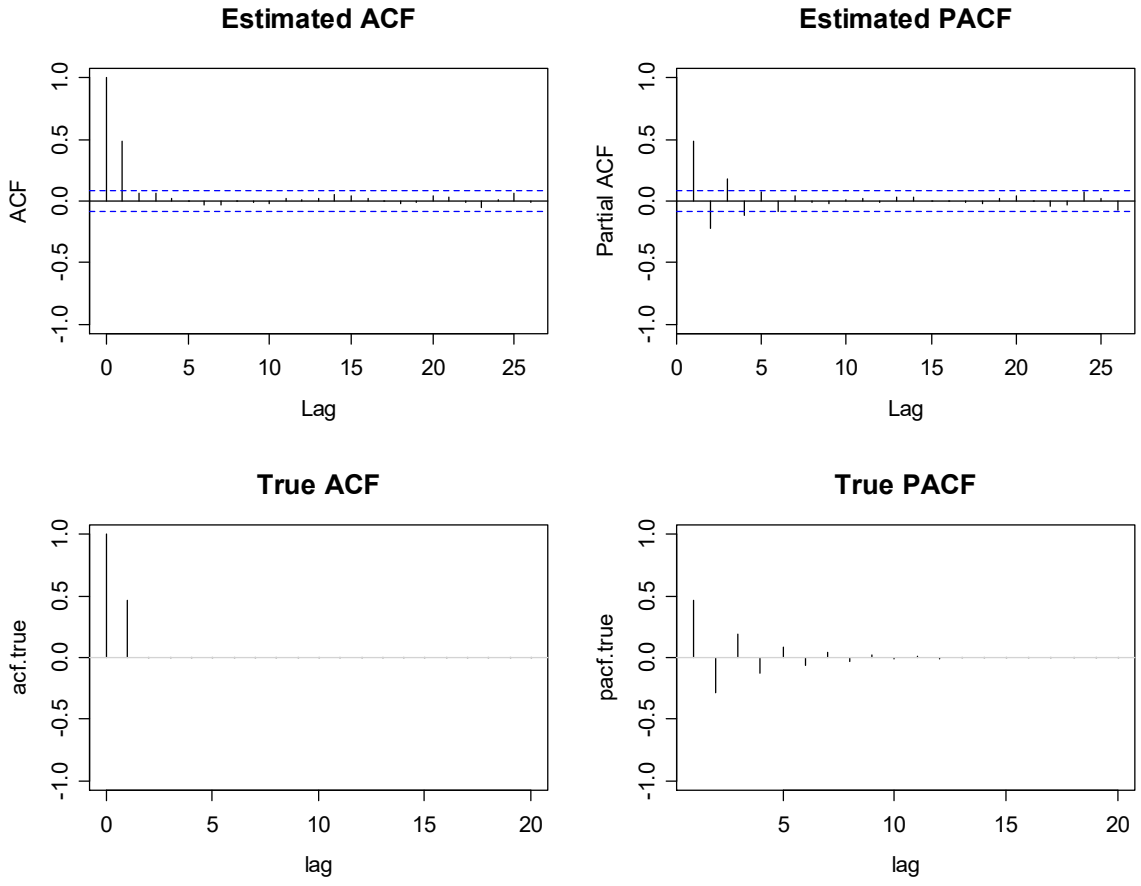
$$\rho(1) = \frac{\beta_1}{1+\beta_1^2} = \frac{1/\beta_1}{1+(1/\beta_1)^2}$$

Apparently, any $MA(1)$ process with coefficient β_1 has exactly the same ACF as the one with $1/\beta_1$. Thus, the two processes $X_t = E_t + 0.5 \cdot E_{t-1}$ and $U_t = E_t + 2 \cdot E_{t-1}$ have

the same dependency structure. Or in other words, given some ACF, we cannot identify the generating MA process uniquely. This problem of ambiguity leads to the concept of *invertibility*. Now, if we express the processes X_t and U_t in terms of $X_{t-1}, X_{t-2}, \ldots$ resp. $U_{t-1}, U_{t-2}, \ldots$, we find by successive substitution:

$$E_t = X_t - \beta_1 X_{t-1} + \beta_1^2 X_{t-2} - \ldots$$
$$E_t = U_t - (1/\beta_1) U_{t-1} + (1/\beta_1^2) U_{t-2} - \ldots$$

Hence, if we rewrite the $MA(1)$ as an $AR(\infty)$, only one of the processes will converge. That is the one where $|\beta_1| < 1$, and it will be called *invertible*. It is important to know that invertibility of MA processes is central when it comes to fitting them to data, because parameter estimation is based on rewriting them in the form of an $AR(\infty)$.



For higher-order $MA(q)$ processes, the property of invertibility is equally central. If it is met, the series can be rewritten in form of an $AR(\infty)$ and it is guaranteed that there is a unique MA process for any given ACF. Invertibilty of a $MA(q)$ is met if the roots of the characteristic polynomial $\Theta(B)$ all lie outside of the unit circle. As was explained earlier in chapter 5.3.1, we can verify this using the **R** function `polyroot()`. Please note that the estimation procedure described below will always result in coefficients $\hat{\beta}_1, \ldots, \hat{\beta}_q$ that define an invertible $MA(q)$ process.

5.4.2 Fitting

The process of fitting $MA(q)$ models to data is more difficult than for $AR(p)$, as there are no (efficient) explicit estimators and numerical optimization is mandatory. Perhaps the simplest idea for estimating the parameters is to exploit the relation between the model parameters and the autocorrelation coefficients, i.e.:

$$\rho(k) = \begin{cases} \sum_{j=0}^{q-k} \beta_j \beta_{j+k} \,/\, \sum_{j=0}^{q} \beta_j^2 & for \quad k = 1,...,q \\ 0 & for \quad k > q \end{cases}$$

Hence in case of a $MA(1)$, we would determine $\hat{\beta}_1$ by plugging-in $\hat{\rho}(1)$ into the equation $\rho(1) = \beta_1 / (1+\beta_1^2)$. This can be seen as an analogon to the Yule-Walker approach in AR modelling. Unfortunately, the plug-in idea yields an inefficient estimator and is not a viable option for practical work.

Conditional Sum of Squares

Another appealing idea would be to use some (adapted) least squares procedure for determining the parameters. A fundamental requirement for doing so is that we can express the sum of squared residuals $\sum E_t^2$ in terms of the observations $X_1,...,X_n$ and the parameters $\beta_1,...,\beta_q$ only, and do not have to rely on the unobservable $E_1,...,E_n$ directly. This is (up to the choice of some initial values) possible for all invertible $MA(q)$ processes. For simplicity, we restrict our illustration to the $MA(1)$ case, where we can replace any innovation term E_t by:

$$E_t = X_t - \beta_1 X_{t-1} + \beta_1^2 X_{t-2} + ... + (-\beta_1)^{t-1} X_1 + \beta_1^t E_0$$

By doing so, we managed to express the innovation/residual at time t as a function of the model parameter β_1 and a combination of the current and past observations of the series. What is also remaining is the (hypothetical) initial innovation term E_0. Conditional on the assumption $E_0 = 0$, we can indeed rewrite the residuals sum of squares $\sum E_t^2$ of any $MA(1)$ using $X_1,...,X_n$ and β_1 only. However, there is no closed form solution for the minimization of $\sum E_t^2$, since powers of the parameter β_1 appear; but the problem can be tackled using numerical optimization. This approach is known as the *Conditional Sum of Squares* (CSS) method. It works similarly for higher orders q, i.e. fundamentally relies on the invertibility of the $MA(q)$ and assumes that $E_t = 0$ for all $t = -\infty,...,0$. In R, the method is implemented in function `arima()` if argument `method="CSS"` is set.

Maximum-Likelihood Estimation

As can be seen from the **R** help file, the Conditional Sum of Squares method is only secondary to `method="CSS-ML"` in the R function `arima()`. This means that it is preferable to use CSS only to obtain a first estimate of the coefficients $\beta_1,...,\beta_q$. They are then used as starting values for a Maximum-Likelihood estimation, which is based on the assumption of Gaussian innovations E_t. It is pretty obvious that $X_t = E_t + \beta_1 E_{t-1} + ... + \beta_q E_{t-q}$, as a linear combination of normally distributed random

variables, follows a Gaussian too. By taking the covariance terms into account, we obtain a multivariate Gaussian for the time series vector:

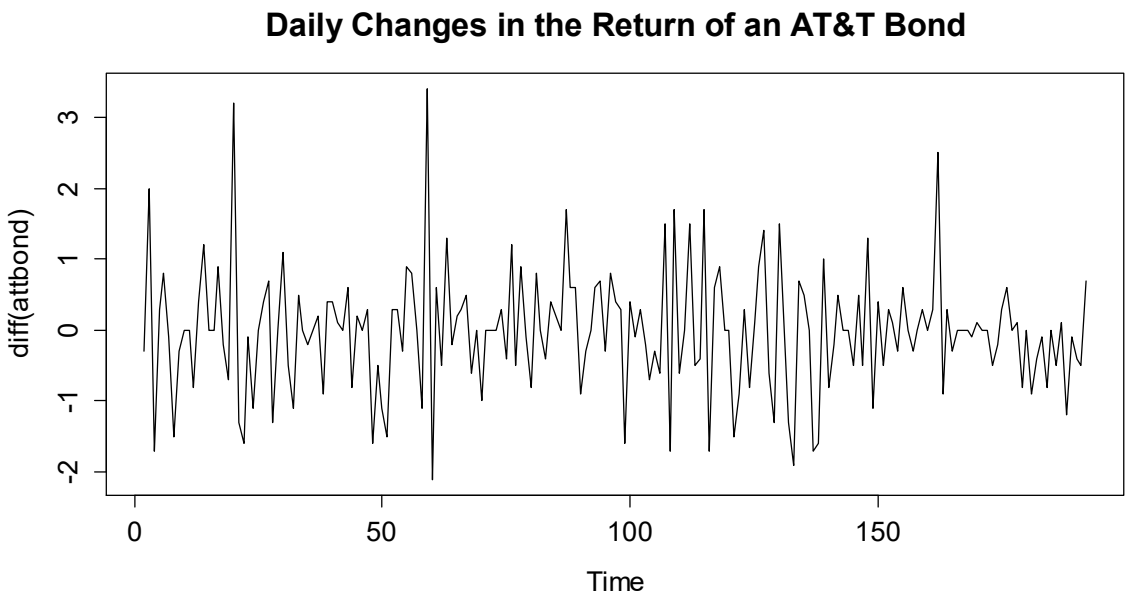
$$X = (X_1,...,X_n) \sim N(0,V) \text{, resp. } Y = (Y_1,...,Y_n) \sim N(m \cdot \underline{1},V).$$

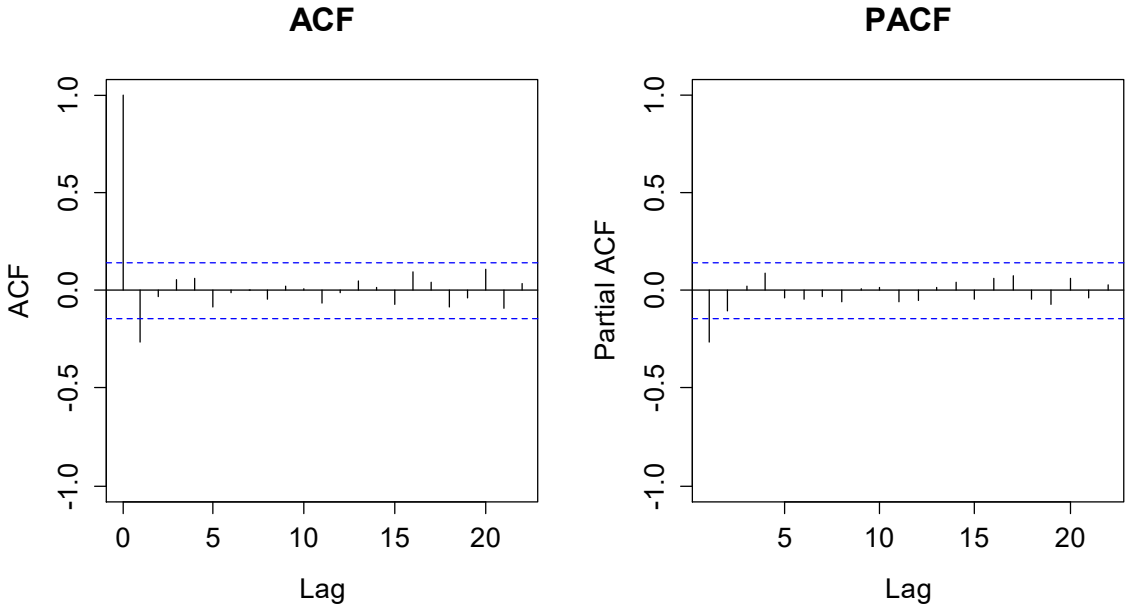
MLE then relies on determining the parameters m (if a shifted $MA(q)$ is estimated), $\beta_1,...,\beta_q$ and σ_E^2 simultaneously by maximizing the probability density function of the above multivariate Gaussian with assuming the data $x_1,...,x_n$ as given quantities. This is a quite complex non-linear problem which needs to be solved numerically. A good implementation is found in **R**'s `arima()`.

The benefit of MLE is that (under mild and in practice usually fulfilled conditions) certain optimality conditions are guaranteed. It is well known that the estimates are asymptotically normal with minimum variance among all asymptotically normal estimators. Additionally, it is pretty easy to derive standard errors for the estimates, which further facilitates their interpretation. And even though MLE is based on assuming Gaussian innovations, it still produces reasonable results if the deviations from that model are not too strong. Be especially wary in case of extremely skewed data or massive outliers. In such cases, applying a log-transformation before the modelling/estimation starts is a wise idea.

5.4.3 Example: Return of AT&T Bonds

As an example, we consider the daily changes in the return of an AT&T bond from April 1975 to December 1975, which makes for a total of 192 observations. The data are displayed along with their ACF and PACF on the next page.





The series seems to originate from a stationary process. There are no very clear cycles visible, hence it is hard to say anything about correlation and dependency, and it is impossible to identify the stochastic process behind the generation of these data from a time series plot alone. Using the ACF and PACF as a visual aid, we observe a pretty clear cut-off situation in the ACF at lag 1 which lets us assume that a $MA(1)$ might be suitable. That opinion is undermined by the fact that the PACF drops off to small values quickly, i.e. we can attribute some exponential decay to it for lags 1 and 2. Our next goal is now to fit the $MA(1)$ to the data. As explained above, the simplest solution would be to determine $\hat{\rho}(1) = -0.266$ and derive $\hat{\beta}_1$ from $\rho(1) = \beta_1 / (1+\beta_1^2)$. This yields two solutions, namely $\hat{\beta}_1 = -0.28807$ and $\hat{\beta}_1 = -3.47132$. Only one of these (the former) defines an invertible $MA(1)$, hence we would stick to that solution. A better alternative is to use the CSS approach for parameter estimation. The code for doing so is as follows:

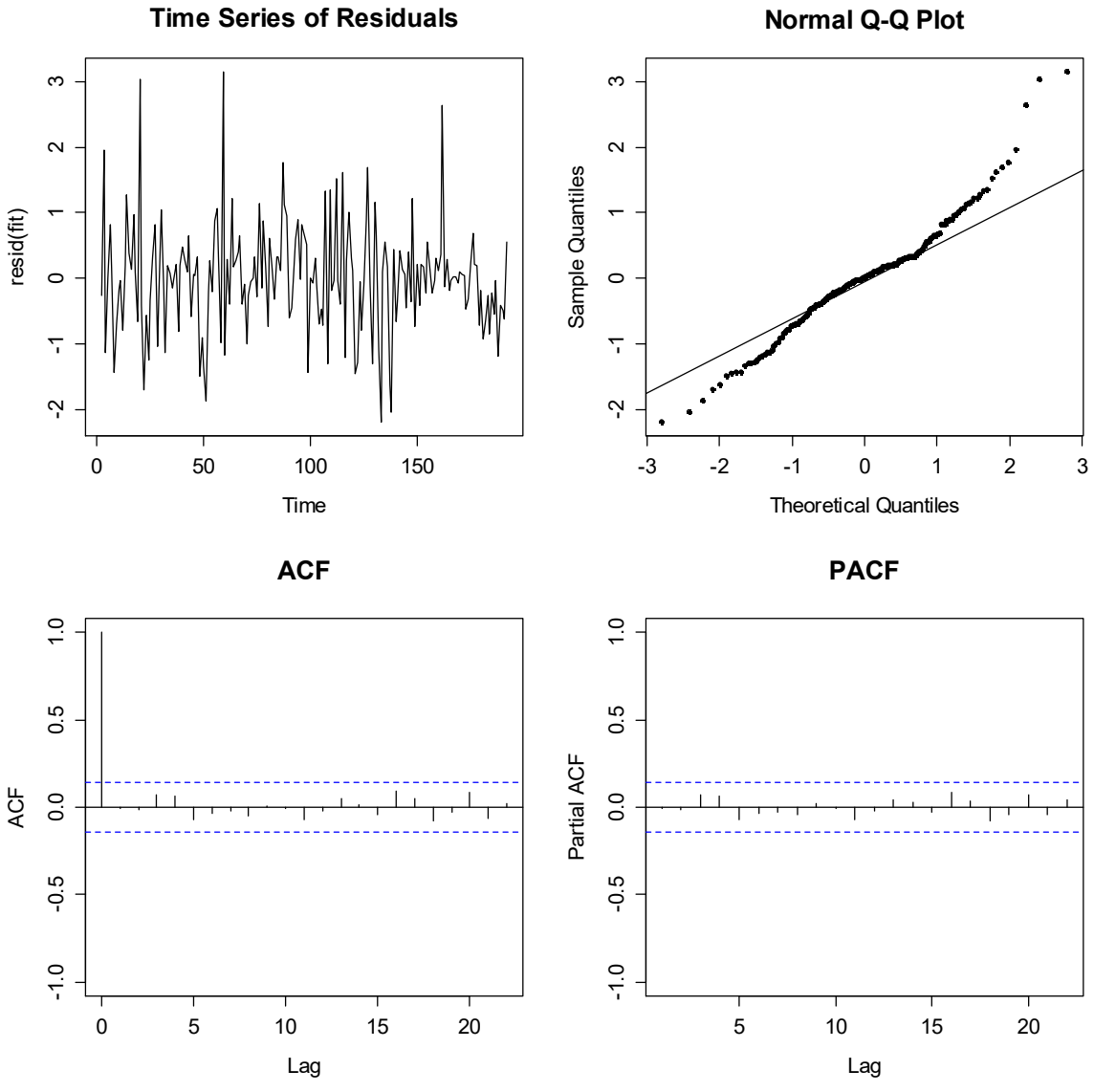
```
> arima(diff(attbond), order=c(0,0,1), method="CSS")
Coefficients:
          ma1  intercept
      -0.2877    -0.0246
s.e.   0.0671     0.0426

sigma^2 estimated as 0.6795:  part log likelihood = -234.11
```

Even more elegant and theoretically sound is the MLE. We can also perform this in **R** using function `arima()`. It yields a very similar but not identical result:

```
> arima(diff(attbond), order=c(0,0,1))
Coefficients:
          ma1  intercept
      -0.2865    -0.0247
s.e.   0.0671     0.0426

sigma^2 = 0.6795: log likelihood = -234.16, aic = 474.31
```

Please note that the application of the three estimation procedures here was just for illustrative purposes, and to show the (slight) differences that manifest themselves when different estimators are employed. In any practical work, you can easily restrict yourself to the application of the `arima()` procedure using the default fitting by `method="CSS-ML"`. For verifying the quality of the fit, a residual analysis is mandatory. The residuals of the $MA(1)$ are estimates of the innovations E_t. The model can be seen as adequate if the residuals reflect the main properties of the innovations. Namely, they should be stationary and free of any dependency, as well as approximately Gaussian. We can verify this by producing a time series plot of the residuals, along with their ACF and PACF, and a Normal QQ-Plot. Sometimes, it is also instructive to plot the fitted values into the original data, or to simulate from the fitted process, as this further helps verifying that the fit is good.

```
> fit <- arima(diff(attbond), order=c(0,0,1))
> plot(resid(fit))
> qqnorm(resid(fit)); qqline(resid(fit))
> acf(resid(fit)); pacf(resid(fit))
```



There are no autocorrelations or partial autocorrelations that exceed the confidence bounds, hence we can safely conjecture that the residuals are not correlated and hence, all the dependency signal has been captured by the $MA(1)$. When inspecting the time series of the residuals, it seems stationary. However, what catches the attention is the presence of three positive outliers and the fact that the residuals are generally long-tailed. We might try to overcome this problem by a log-transformation, but this is left to the reader.

5.5 ARMA(p,q) Models

Here, we discuss models that feature both dependency on previous observations $X_{t-1}, X_{t-2}, \ldots$ as well as previous innovations terms $E_{t-1}, E_{t-2}, \ldots$. Thus, they are a hybrid between $AR(p)$ and $MA(q)$ models, and aptly named $ARMA(p,q)$. Their importance lies in the fact that it is possible to model a far wider spectrum of dependency structures, and that they are parsimonious: often, an $ARMA(p,q)$ requires (far) fewer parameters than pure AR or MA processes would.

5.5.1 Definition and Properties

The formal definition of an $ARMA(p,q)$ process is as follows:

$$X_t = \alpha_1 X_{t-1} + \ldots + \alpha_p X_{t-p} + E_t + \beta_1 E_{t-1} + \ldots + \beta_q E_{t-q}$$

As before, we assume that E_t is causal and White Noise, i.e. an innovation with mean $E[E_t] = 0$ and finite variance $Var(E_t) = \sigma_E^2$. It is much more convenient to use the characteristic polynomials $\Phi(\cdot)$ for the AR part, and $\Theta(\cdot)$ for the MA part, because this allows for a very compact notation:

$$\Phi(B) X_t = \Theta(B) E_t .$$

It is obvious that all relevant properties of an $ARMA(p,q)$ process lie in the characteristic polynomials. If the roots of $\Phi(\cdot)$ are outside of the unit circle, the process will be stationary and have mean zero. On the other hand, if the roots of $\Theta(\cdot)$ are outside of the unit circle, the process is invertible. Both properties are important for practical application. If they are met, we can rewrite any $ARMA(p,q)$ in the form of a $AR(\infty)$ or an $MA(\infty)$. This explains why fitting an $ARMA(p,q)$ can in practice often be replaced by fitting AR- or MA-models with high orders (although it is not a good idea to do so!). As has been argued above, any stationary $ARMA(p,q)$ has mean zero, i.e. $E[X_t] = 0$. Thus, in practice we will often consider shifted $ARMA$-processes that are of the form:

$Y_t = m + X_t$, where X_t is an $ARMA(p,q)$.

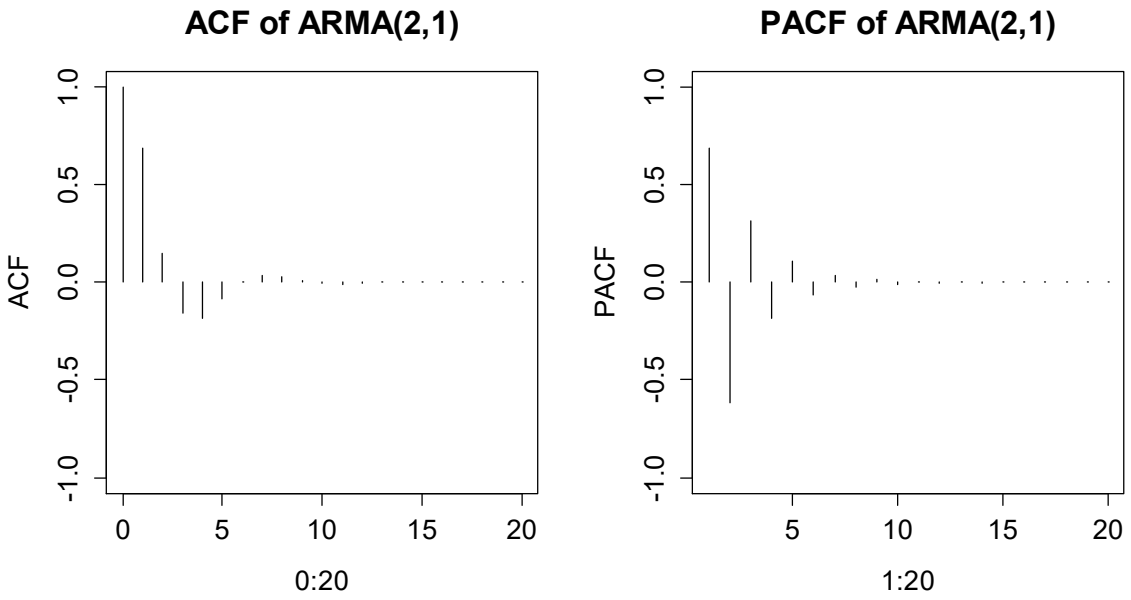
In principle, it is straightforward to derive the ACF of an $ARMA(p,q)$, though algebraically a bit tedious. Given the applied focus of this scriptum, we do without and focus on the results and consequences instead. We illustrate the typical behavior of the $ARMA$ autocorrelations on the basis of an example. Namely, we consider the $ARMA(2,1)$ defined by:

$$X_t - 0.8 X_{t-1} + 0.4 X_{t-2} = E_t + 0.6 E_{t-1}$$

On the next page, we exhibit the (theoretical) ACF and PACF. It is typical that neither the ACF nor the PACF cut-off strictly at a certain lag. Instead, they both show some infinite behavior, i.e. an exponential decay in the magnitude of the coefficients. However, superimposed on that is a sudden drop-off in both ACF and PACF. In our

example, it is after lag 1 in the ACF, as induced by the moving average order $q = 1$. In the PACF, the drop-off happens after lag 2, which is the logical consequence of the autoregressive order of $p = 2$. The general behavior of the ACF and PACF is summarized in the table below.

Model	ACF	PACF
$AR(p)$	infinite / exp. decay	cut-off at lag p
$MA(q)$	cut-off at lag q	infinite / exp. decay
$ARMA(p,q)$	infinite / mix of decay & cut-off	infinite / mix of decay & cut-off



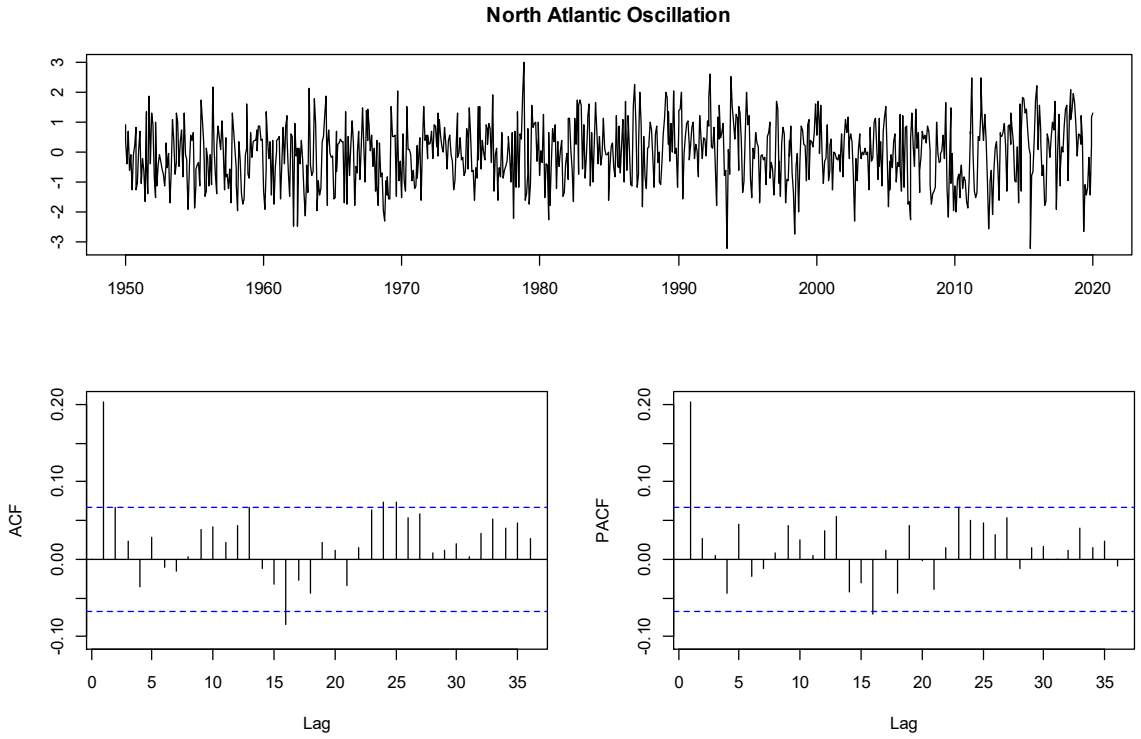
It is important to know that with $ARMA(p,q)$ processes, a wealth of autocorrelation structures can be generated. As to how visible the two cut-offs in ACF and PACF are, resp. whether the cut-off or the decay is dominating, depends on the model's coefficients. There are $ARMA$'s where the AR part is dominating, there are others where the MA is stronger, and of course they can also be on an equal footing.

5.5.2 Fitting

The above properties of ACF and PACF can be exploited for choosing the type and order of a time series model. In particular, if neither the ACF nor the PACF shows a pronounced cut-off, where after some low lag (i.e. p or $q < 10$) all the following correlations are non-significantly different from zero, then it is usually wise to choose an $ARMA(p,q)$. For determining the order (p,q), we search for the superimposed cut-off in the ACF (for q), respectively PACF (for p). The drop-off is not always easy to identify in practice. In "difficult" situations, it has also proven beneficial to support the choice of the right order with the AIC criterion. We could for example perform a grid search on all possible $ARMA(p,q)$ models which do not use more than 5 parameters, i.e. $p + q < 5$. This can readily be done in **R** by programming a `for()` loop or using `auto.arima()` in `library(forecast)`.

It is very important to know that $ARMA$ models are parsimonious, i.e. they usually do not require high orders p and q. In particular, they work well with far fewer parameters then pure AR or MA models would. Or in other words: often it is possible to fit high-order $AR(p)$'s or $MA(q)$'s instead of a low-order $ARMA(p,q)$. That property does not come as a surprise: as we conjectured above, any stationary and invertible $ARMA$ can be represented in the form of an $AR(\infty)$ or an $MA(\infty)$. However, this is not a good idea in practice: estimating parameters "costs money", i.e. will lead to less precise estimates. Thus, a low-order $ARMA(p,q)$ is always to be preferred over a high-order $AR(p)$ or $MA(q)$. As an example, we here study the time series of the North Atlantic Oscillation, obtained from https://www.ncdc.noaa.gov/teleconnections/nao/. It reports the atmospheric pressure difference at sea level between the Icelandic Low and the Azores High from January 1950 to January 2020. We inspect the series, ACF and PACF.

```
> tsdisplay(nao, points=FALSE, main="North Atlantic …")
```



We observe a series for which we can withhold the stationarity assumption and that fluctuates around a global mean of around zero. The first ACF and PACF coefficients clearly exceed the confidence bands, while all further correlations seem to be much smaller and if, only barely exceed the bounds. A nearby model for this series is an ARMA(1,1). For estimating the coefficients, we are again confronted with the fact that there are no explicit estimators available. This is due to the MA component in the model which involves innovation terms that are unobservable. By rearranging terms in the model equation, we can again represent any $ARMA(p,q)$ in a form where it only depends on the observed X_t, the coefficients $\alpha_1,...,\alpha_p;\beta_1,...,\beta_q$ and some previous innovations E_t with $t<1$. If these latter terms are all set to zero, we can determine the optimal set of model coefficients by

minimizing the sum of squared residuals (i.e. innovations) with a numerical method. This is the CSS approach that was already mentioned in 5.4.2 and is implemented in function `arima()` when `method="CSS"`. By default however, these CSS estimates are only used as starting values for a MLE. If Gaussian innovations are assumed, then the joint distribution of any $ARMA(p,q)$ process vector $X=(X_1,...,X_n)$ has a multivariate normal distribution.

$$X=(X_1,...,X_n)\sim N(0,V)\text{, resp. } Y=(Y_1,...,Y_n)\sim N(m\cdot\underline{1},V).$$

MLE then relies on determining the parameters m (if a shifted $ARMA(p,q)$ is estimated), $\alpha_1,...,\alpha_p;\beta_1,...,\beta_q$ and σ_E^2 simultaneously by maximizing the probability density function of the above multivariate Gaussian with assuming the data $x_1,...,x_n$ as given quantities. This is a quite complex non-linear problem which needs to be solved numerically. A good implementation is found in **R**'s `arima()`. As was stated previously, the benefit of MLE is that (under mild and mostly met conditions) some optimality is guaranteed. In particular, the estimates are asymptotically normal with minimum variance among all asymptotically normal estimators. Another benefit is provided by the standard errors which allow for judging the precision of the estimates. We proceed with our example from above and now fit an ARMA(1,1) to the North Atlantic Oscillation series:

```
> fit0 <- arima(nao, order=c(1,0,1)); fit0
Coefficients:
         ar1      ma1  intercept
      0.3273  -0.1285    -0.0012
s.e.  0.1495   0.1565     0.0446
sigma^2=0.9974; log likelihood=-1192.28; aic=2392.55
```

It turns out that the global mean (i.e. the `intercept` in the model) is not significantly different from zero, because the 95% confidence interval of $-0.0012\pm2\cdot0.0446$ contains the value zero. Thus, we can remove the intercept from the model which saves estimating one useless parameter.

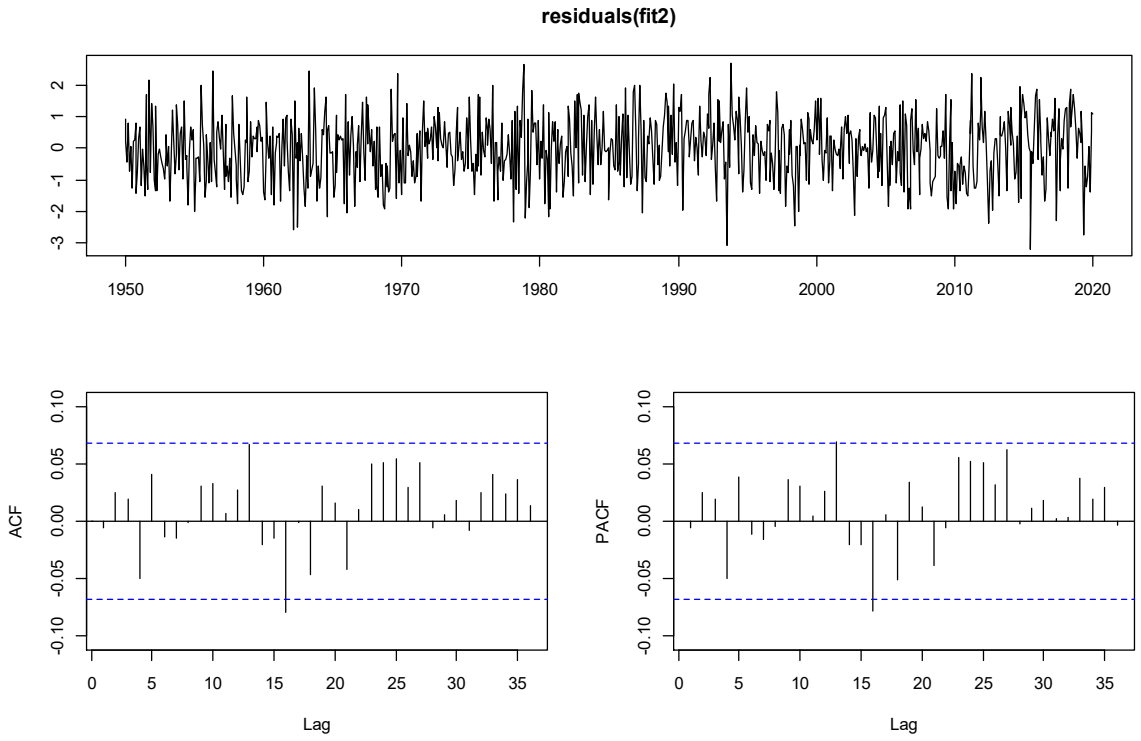
```
> fit1 <- arima(nao, order=c(1,0,1), include.mean = F); fit1
Coefficients:
         ar1      ma1
      0.3273  -0.1285
s.e.  0.1495   0.1565
sigma^2=0.9974; log likelihood=-1192.28; aic=2390.55
```

Apparently, this changes all other estimated quantities unsubstantially. If we further proceed with optimizing the model, we notice that also $\hat{\beta}_1$ is not significantly different from zero. Thus, we reduce the model to an AR(1) without the constant:

```
> fit2 <- arima(nao, order=c(1,0,0), include.mean = F); fit2
Coefficients:
         ar1
      0.2041
s.e.  0.0338
sigma^2=0.9982; log likelihood=-1192.59; aic=2389.18
```

Now, the 95% confidence interval for α_1 clearly indicates that the the AR(1) coefficient is significantly different from zero. Moreover, reducing the model also seems beneficial in terms of AIC (see below for a thorough explanation of this quantity). Thus, despite an ACF/PACF that indicated an ARMA(1,1), a simpler model seems to do the job here. However, we should not be overly quick with our conjectures, but first verify that the residuals meet the White Noise assumption.

```
> tsdisplay(residuals(fit2), points=FALSE)
```



Except for the ACF and PACF at lag 16, the situation looks unproblematic. Can we ignore those two estimates that crack the confidence bands? In the opinion of the lecturer, the answer is yes in this particular example. In the first place, the dependency of the ARMA(1,1) residuals (not shown here) does not look markedly different and an even bigger model does not seem to be justified. Second, the p-value of a Ljung-Box test over the first 24 levels is at 0.3056, providing further evidence that the remaining dependence is insignificant.

5.5.3 AIC-Based Model Choice

We have explained above how the order of $ARMA(p,q)$ models can be found by inspecting ACF and PACF and complementing this with classical model selection approaches and residual analysis. Another alternative is to run a criterion-based model selection. In R, this is conveniently possible by using function `auto.arima()` from `library(forecast)`. However, handle this with care: the function will always identify a "best fitting" $ARMA(p,q)$ model, but it is of course not guaranteed that it fits the data well. Moreover, usage of the function is somewhat

tricky, as many arguments need being set. We first address the definition of the information criteria, as they are central to the `auto.arima()` function.

$$AIC = -2\log(L) + 2(p+q+k+1)$$

Here, the first term measures how well the model fits the training data with the value of the Log-Likelihood function as the goodness-of-fit measure. The second term penalizes for model complexity, where p,q are the AR- resp. MA-orders, $k=1$ if a global mean was estimated (else $k=0$) and the final $+1$ stands for the innovation variance which always needs to be estimated. Function `auto.arima()` by default relies on a small sample corrected version $AICc$:

$$AICc = AIC + \frac{2(p+q+k+1)(p+q+k+2)}{n-p-q-k-2}$$

A third option consists of using the BIC criterion, which penalizes model size somewhat differently. The definition is as follows:

$$BIC = -2\log(L) + \log(n)(p+q+k+1) = AIC + (\log(n)-2)(p+q+k+1)\,.$$

It is noteworthy that the outcome can be sensitive to the criterion used, because in practice many models may perform similarly. In those cases, there is usually no "right or wrong" in model selection, but several nearly equivalent alternatives may exist. Next, we focus on the algorithm behind the convenient `auto.arima()`. It is concisely summarized on https://otexts.com/fpp2/arima-r.html by Hyndman & Athanasopoulos (2018), from where we copy the following scheme:

Hyndman-Khandakar algorithm for automatic ARIMA modelling

1. The number of differences $0 \leq d \leq 2$ is determined using repeated KPSS tests.

2. The values of p and q are then chosen by minimising the AICc after differencing the data d times. Rather than considering every possible combination of p and q, the algorithm uses a stepwise search to traverse the model space.

   a. Four initial models are fitted:
   - ARIMA$(0,d,0)$,
   - ARIMA$(2,d,2)$,
   - ARIMA$(1,d,0)$,
   - ARIMA$(0,d,1)$.

   A constant is included unless $d=2$. If $d \leq 1$, an additional model is also fitted:
   - ARIMA$(0,d,0)$ without a constant.

   b. The best model (with the smallest AICc value) fitted in step (a) is set to be the "current model".

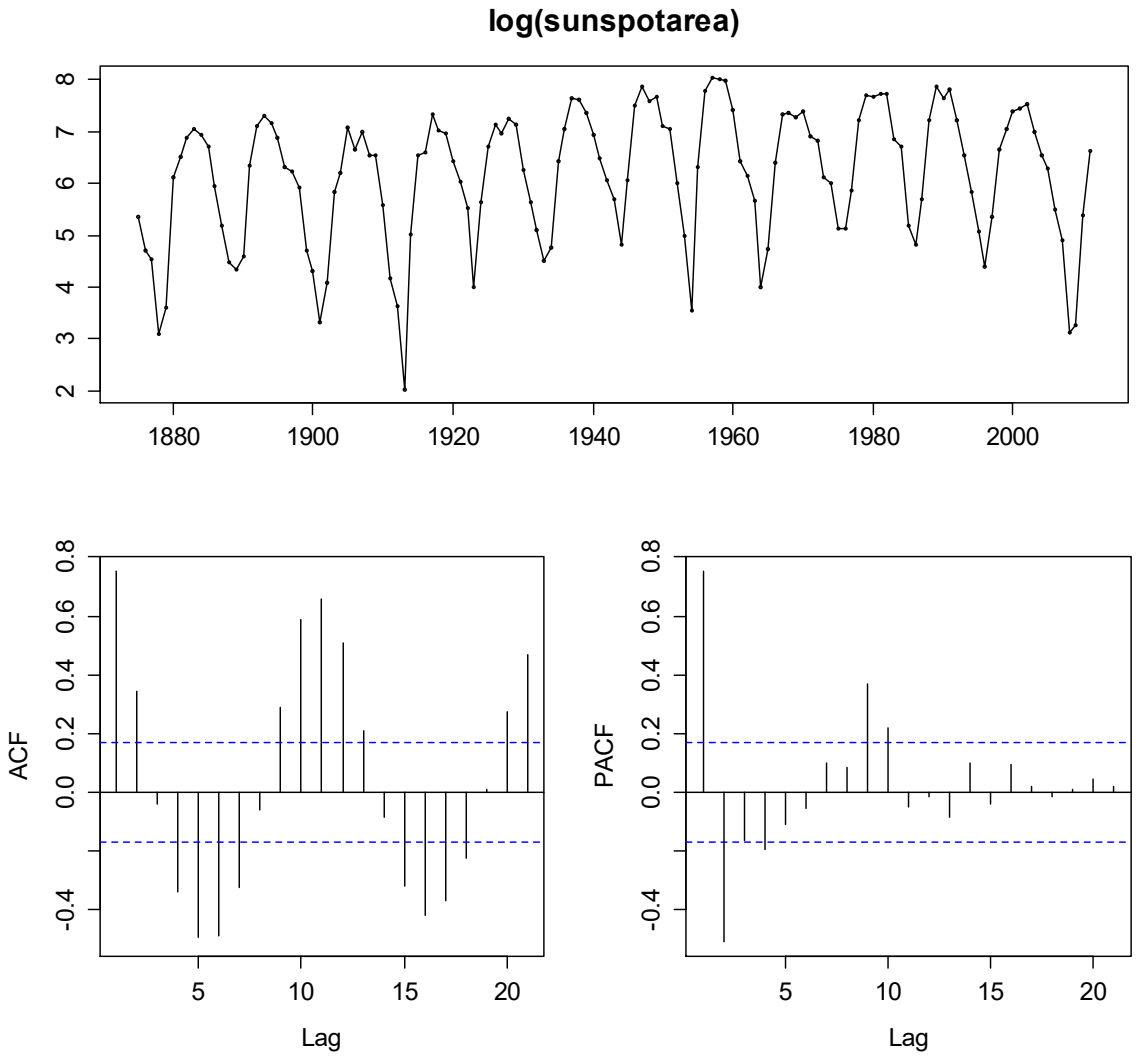
   c. Variations on the current model are considered:
   - vary p and/or q from the current model by ± 1;
   - include/exclude c from the current model.

   The best model considered so far (either the current model or one of these variations) becomes the new current model.

   d. Repeat Step 2(c) until no lower AICc can be found.

Please be aware that we have not yet addressed true $ARIMA(p,d,q)$ models with $d>0$ i.e. where differencing is involved to cope with non-stationary series. Hence, the first step is currently to be ignored. We illustrate the use of this function using `data(sunspotarea)` from `library(fpp)`. It contains annual averages of the daily sunspot areas (in units of millionths of a hemisphere) for the full sun. Sunspots are magnetic regions that appear as dark spots on the surface of the sun. The Royal Greenwich Observatory compiled daily sunspot observations from May 1874 to 1976. Later data are from the US Air Force and the US National Oceanic and Atmospheric Administration and were calibrated to be consistent across the whole history of observations. We assume the data to be on a relative scale and hence use a log-transformation before modelling them.

```
> tsdisplay(sunspotarea)
```

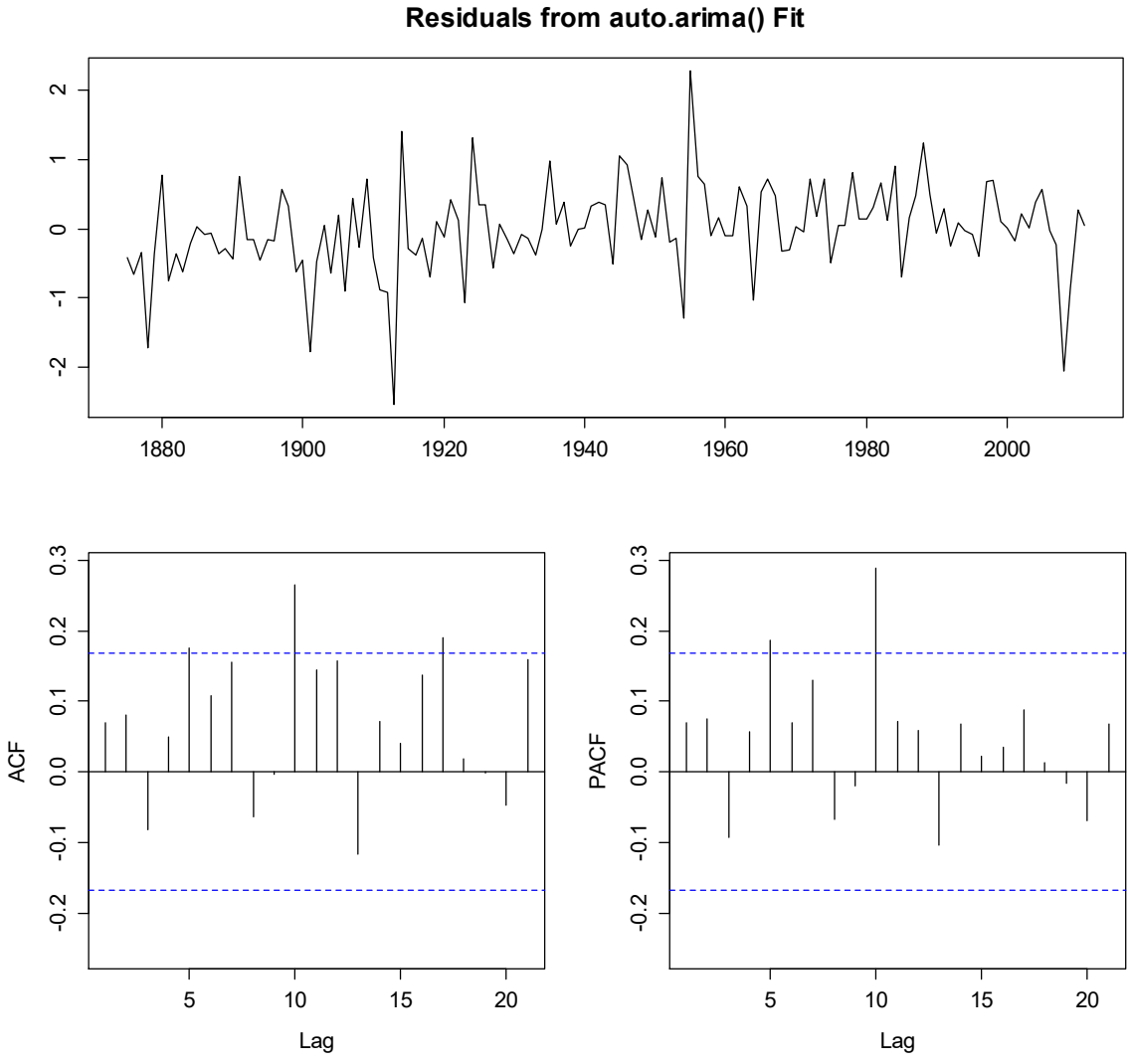


The series, much like the lynx data, shows some strong periodic behavior. However, these periods are seen to be stochastic, hence the series as a whole is considered being stationary. The ACF has a slow decay and the PACF cuts-off after lag 10, suggesting an $AR(10)$. We complement with an exhaustive AIC-based search over all $ARMA(p,q)$ up to $p,q \le 10$.

```
> fit <- auto.arima(log(sunspotarea), max.p=10, max.q=10,
         stationary=TRUE, seasonal=FALSE, ic="aic",
         stepwise=FALSE); fit
ARIMA(2,0,3) with non-zero mean
Coefficients:
         ar1      ar2      ma1      ma2     ma3  intercept
      1.6548  -0.9775  -0.8583  -0.0425  0.4484     6.1968
s.e.  0.0210   0.0192   0.0830   0.1004  0.0785     0.0935
sigma^2 estimated as 0.4127:  log likelihood=-135.33
AIC=284.67   AICc=285.53   BIC=305.11
```

Note that we need to set the information criterion argument `ic="aic"`. Moreover, if it is computationally feasible, we recommend to set `stepwise=FALSE`, because else a non-exhaustive, stepwise search strategy will be employed which may not result in the AIC-optimal model. As it turns out, an $ARMA(2,3)$ yields the lowest AIC value, i.e. is better in this respect than an $AR(10)$ and spends fewer parameters, too. To verify whether the model fits adequately, we need to run a residual analysis.

```
> tsdisplay(resid(fit))
```



As we can observe, the time series of residuals is not White Noise, since there are several ACF and PACF coefficients that exceed the confidence bands. Hence, the AIC-selected model clearly underfits these data. If an $AR(10)$ is used in place of the $ARMA(2,3)$, the residuals feature the desired White Noise property. As a conjecture, we would reject the $ARMA(2,3)$ here despite its better AIC value and note that blindly trusting in automatic model selection procedures may well lead to models that fit poorly.

6 SARIMA and GARCH Models

As we have discovered previously, many time series are non-stationary due to trends and/or seasonal effects. While we have learned to decompose these and then explain the remainder with some time series models, there are other models that can directly incorporate trend and seasonality. While they usually lack some transparency for the decomposition, their all-in-one approach allows for convenient forecasting, and also AIC-based decisions for choosing the right amount of trend and seasonality modeling become feasible.

Time series from financial or economic background often show serial correlation in the conditional variance, i.e. are conditionally heteroskedastic. This means that they exhibit periods of high and low volatility. Understanding the behavior of such series pays off, and the usual approach is to set up autoregressive models for the conditional variance. These are the famous ARCH models, which we will discuss along with their generalized variant, the GARCH class.

6.1 ARIMA Models

ARIMA models are aimed at describing series which exhibit a trend that can be removed by differencing at lag 1; and where these differences can be described by an $ARMA(p,q)$ model. Thus, the definition of an $ARIMA(p,d,q)$ process arises naturally:

Definition: A series X_t follows an $ARIMA(p,d,q)$ model if the d th order lag 1 difference of X_t is an $ARMA(p,q)$ process. If we introduce

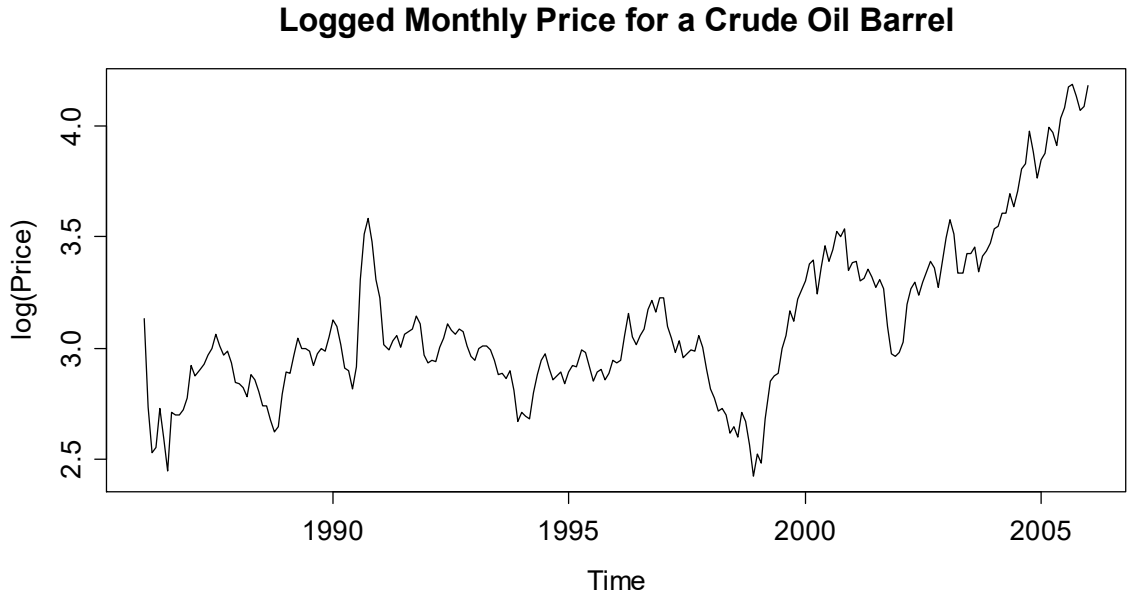
$$Y_t = (1-B)^d X_t,$$

where B is the backshift operator, then we can write the $ARIMA$ process using the characteristics polynomials, i.e. $\Theta(\cdot)$ that accounts for the MA, and $\Phi(\cdot)$ that stands for the AR part.

$$\begin{aligned} \Phi(B)Y_t &= \Theta(B)E_t \\ \Phi(B)(1-B)^d X_t &= \Theta(B)E_t \end{aligned}$$

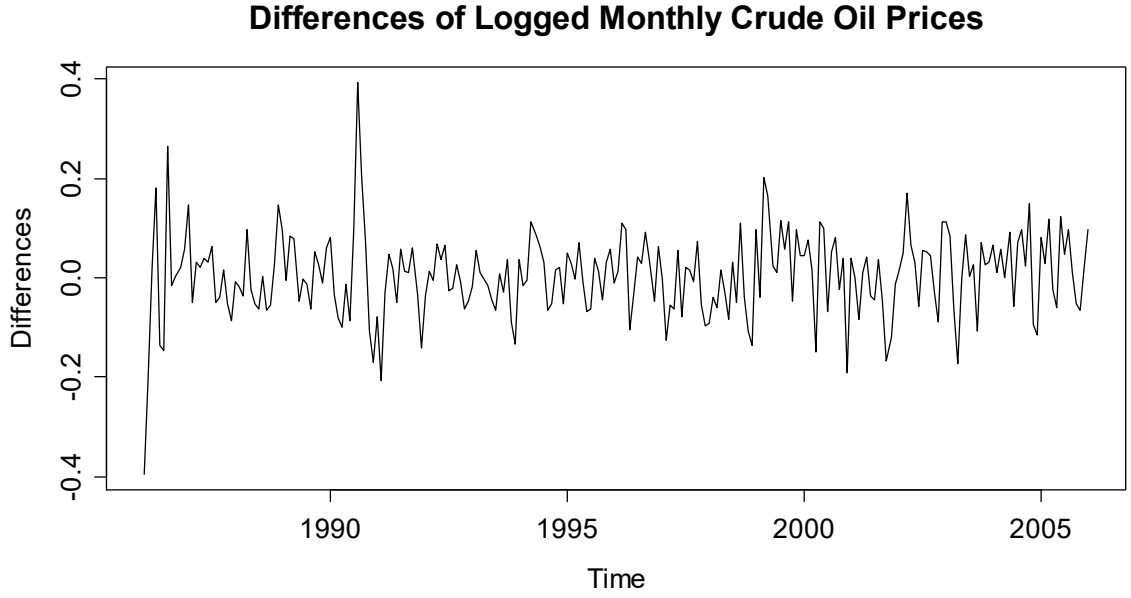
Such series do appear in practice, as our example of the monthly prices for a barrel of crude oil (in US$) from January 1986 to January 2006 shows. To stabilize the variance, we decide to log-transform the data, and model these.

```
> library(TSA)
> data(oil.price)
> lop <- log(oil.price)
> plot(lop, ylab="log(Price)")
> title("Logged Monthly Price for a Crude Oil Barrel")
```



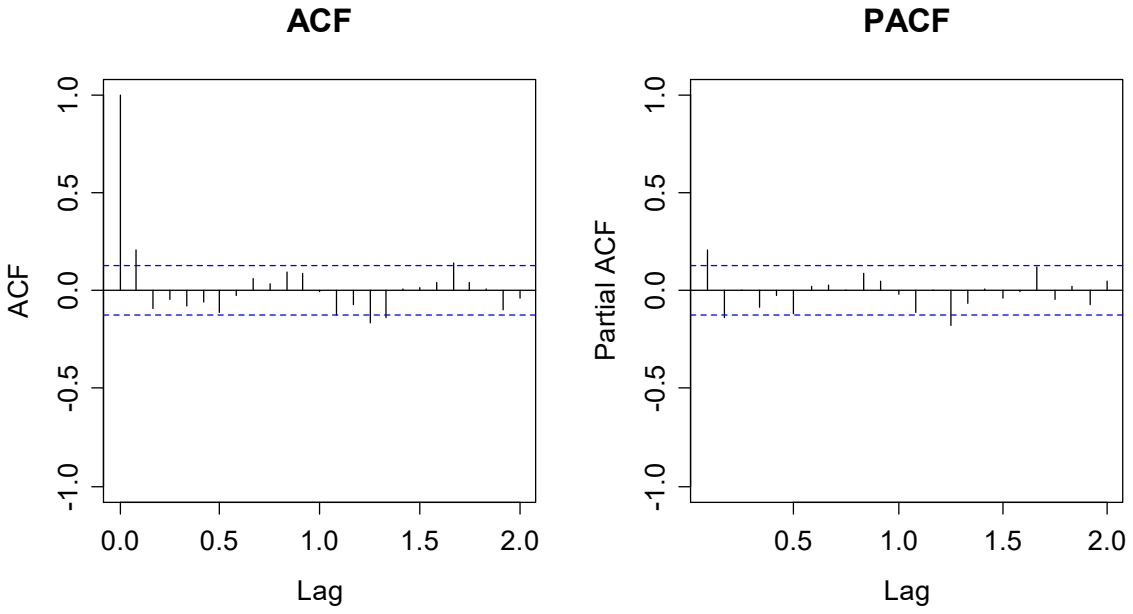
The series does not exhibit any apparent seasonality, but there is a clear trend, so that it is non-stationary. We try first-order (i.e. $d = 1$) differencing at lag 1, and then check whether the result is stationary.

```
> dlop <- diff(lop)
> plot(dlop, ylab="Differences")
> title("Differences of Logged Monthly Crude Oil Prices")
```



The trend was successfully removed by taking differences. ACF and PACF show that the result is serially correlated. There may be a drop-off in the ACF at lag 1, and in the PACF at either lag 1 or 2, suggesting an $ARIMA(1,1,1)$ or an $ARIMA(2,1,1)$ for the logged oil prices. We base our choice on the AIC value which suggests using the smaller model $ARIMA(1,1,1)$.

```
> par(mfrow=c(1,2))
> acf(dlop, main="ACF", ylim=c(-1,1), lag.max=24)
> pacf(dlop, main="ACF", ylim=c(-1,1), lag.max=24)
```



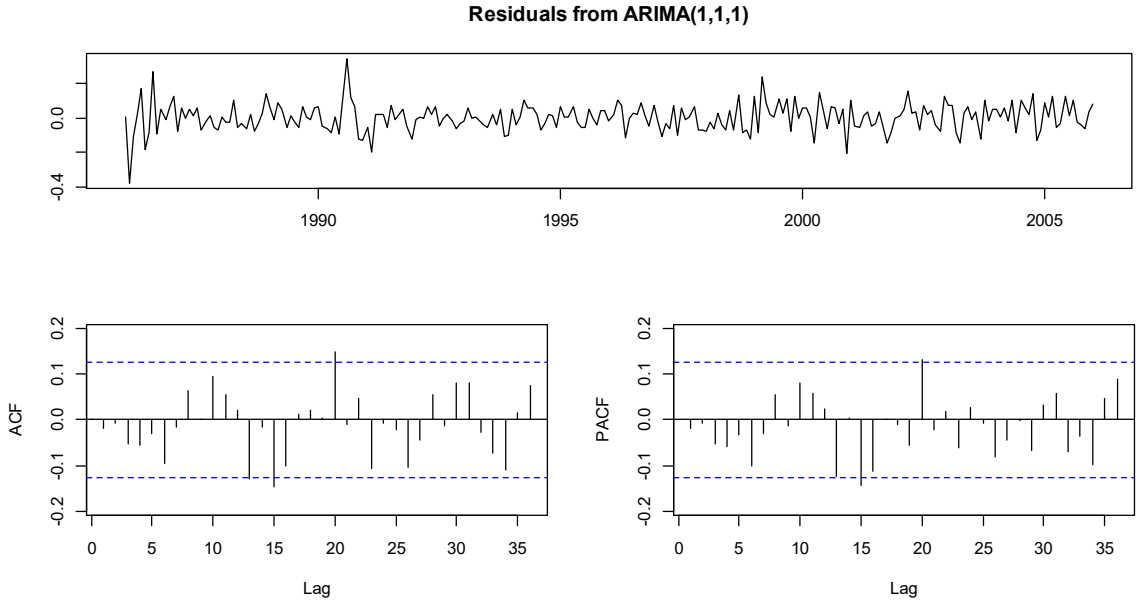
The fitting can be done with the `arima()` procedure that (by default) estimates the coefficients using Maximum Likelihood with starting values obtained from the Conditional Sum of Squares method. We can either let the procedure do the differencing:

```
> arima(lop, order=c(1,1,1))

Call: arima(x = lop, order = c(1, 1, 1))

Coefficients:
          ar1      ma1
      -0.2987   0.5700
s.e.   0.2009   0.1723

sigma^2 = 0.006642:  log likelihood = 261.11,  aic = -518.22
```

Or, we can use the differenced series `dlop` as input and fit an $ARMA(1,1)$. However, we need to tell **R** to not include an intercept – this is not necessary when the trend was removed by taking differences and the constant would result in a so-called (sometimes useful) drift-term, see chapter 8.2.1. The command is:

```
> arima(dlop, order=c(1,0,1), include.mean=FALSE)
```

The output from this is exactly the same as above, although it is generally better to use the first approach and fit a true $ARIMA$ model. The next step is to perform residual analysis – if the model is appropriate, they must look like White Noise. This is more or less the case, see next page. For decisions on the correct model order, also the AIC statistics can provide valuable information.



We finish this section by making some considerations on the model equation. We have:

$$\begin{aligned} Y_t &= -0.30 \cdot Y_{t-1} + E_t + 0.57 \cdot E_{t-1} \\ X_t - X_{t-1} &= -0.30 \cdot (X_{t-1} - X_{t-2}) + E_t + 0.57 \cdot E_{t-1} \\ X_t &= +0.70 \cdot X_{t-1} - 0.30 \cdot X_{t-2} + E_t + 0.57 \cdot E_{t-1} \end{aligned}$$

Thus, the $ARIMA(1,1,1)$ can be rewritten as a non-stationary $ARMA(2,1)$. The non-stationarity is due to a unit root in the AR parts' characteristic polynomial. We can identify this using the `polyroot()` function in R.

```
> abs(polyroot(c(1,0.7, -0.3)))
[1] 1.000000 3.333333
```

Finally, we give some recipe for fitting $ARIMA$ models:

1) Choose the appropriate order of differencing, usually $d = 1$ or (in rare cases) $d = 2$, such that the result is a stationary series.

2) Analyze ACF and PACF of the differenced series. If the stylized facts of an $ARMA$ process are present, decide for the orders p and q.

3) Fit the model using the `arima()` procedure. This can be done on the original series by setting d accordingly, or on the differences, by setting $d = 0$ and argument `include.mean=FALSE`.

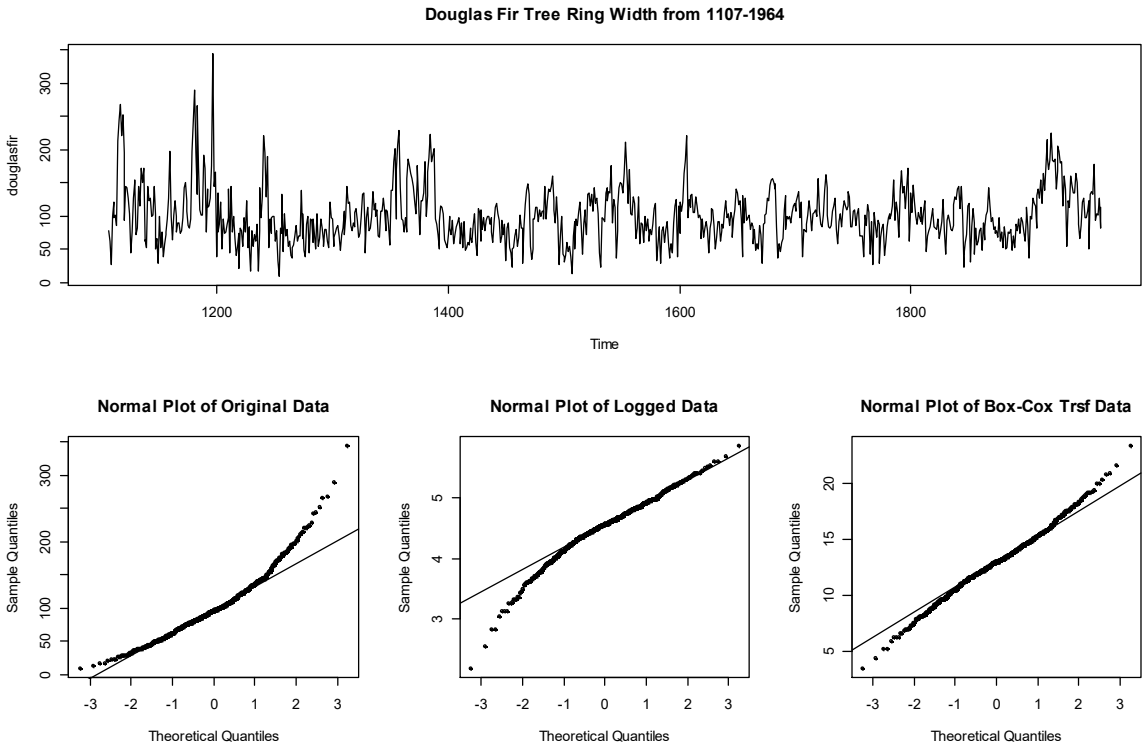
4) Analyze the residuals; these must look like White Noise. If several competing models are appropriate, use AIC to decide for the winner.

The fitted ARIMA model can then be used to generate forecasts including prediction intervals. This will, however, only be discussed in section 8.

Example: Ambiguity in ARIMA Modeling

We here discuss another example where the tree ring widths of a douglas fir are considered over a very long period lasting from 1107 to 1964. Modeling these data is non-trivial, since there remains a lot of ambiguity on how to approach them. The first question is about how to transform these data before modelling. They take positive values only and show some pronounced right skewness, hence a log-transformation might be indicated. After the log-transformation however, the data are left-skewed. In order to achieve a symmetrical distribution, we can use a Box-Cox transformation instead. Function `BoxCox()` suggests $\lambda \approx 0.6$, but since the data are more symmetrically distributed with $\lambda = 0.4$, we choose that value.

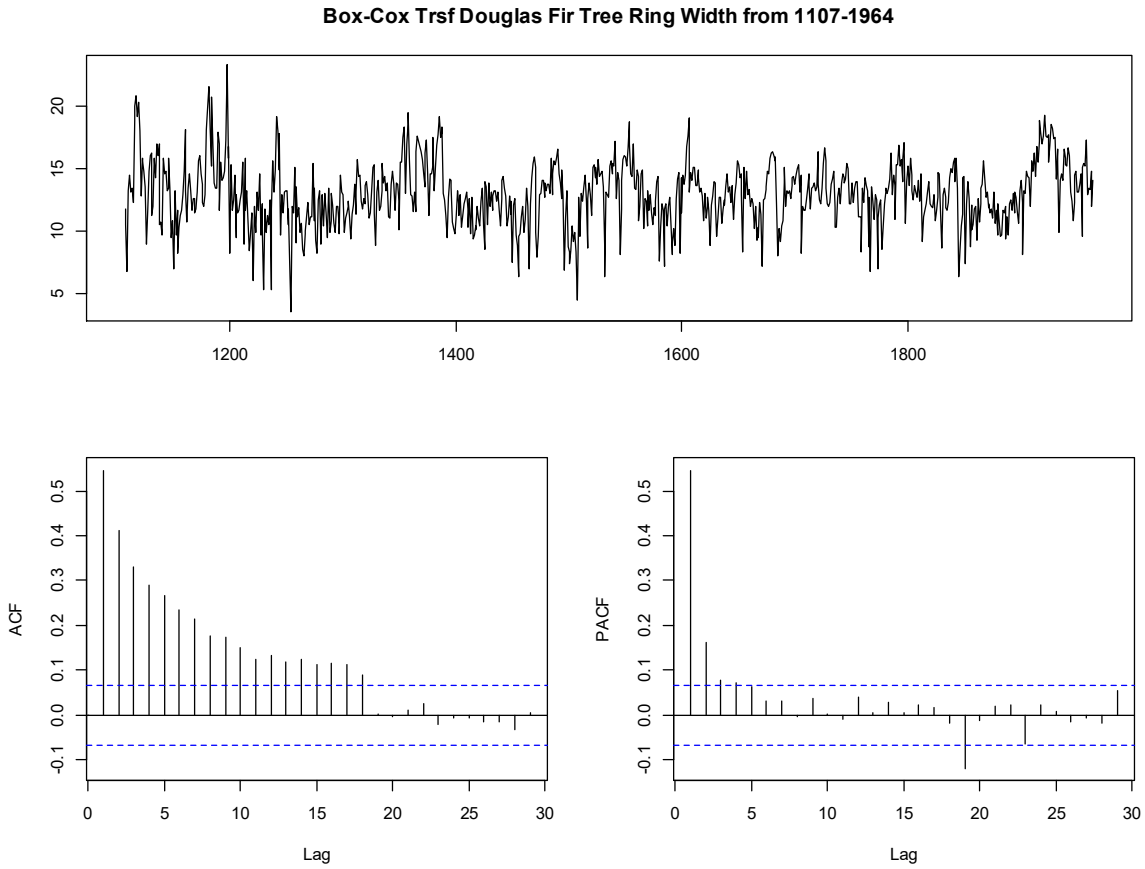
```
> layout(matrix(c(1, 1, 1, 2, 3, 4), 2, 3, byrow = TRUE))
> plot(douglasfir, main="Douglas Fir Tree Ring Width…")
> qqnorm(douglasfir, pch=20); qqline(douglasfir)
> qqnorm(log(douglasfir), pch=20); qqline(log(douglasfir))
> tdf <- BoxCox(douglasfir, lambda=0.4)
> qqnorm(tdf, pch=20, main="…"); qqline(tdf)
```



The next step is the analysis of ACF and PACF (*see next page*). It raises the important question whether the data generating process was stationary or not. The ACF shows a relatively slow decay and the local mean of the series seems to persist on higher/lower levels for longer periods of time. On the other hand, what we observe in this series would clearly still fit within the envelope of what can be produced by a stationary time series process. But then, the differenced data (*see the page thereafter*) look clearly "more stationary". Hence, both options, a pure $ARMA(p,q)$ and an $ARIMA(p,1,q)$ remain open. We here lay some focus on the aspects that are involved in the decision process.

First and foremost, the model chosen needs to fit with the series, the ACF and PACF that we observe. As mentioned above, this here leaves both the option for a stationary and integrated model. Next, we can of course try both approaches and compare the insample fit via the AIC. If function `auto.arima()` is employed, we can even set the scope such that the search includes both stationary and integrated models in one step.

```
> tsdisplay(tdf, points=FALSE, main="Box-Cox Trsf Douglas …")
```

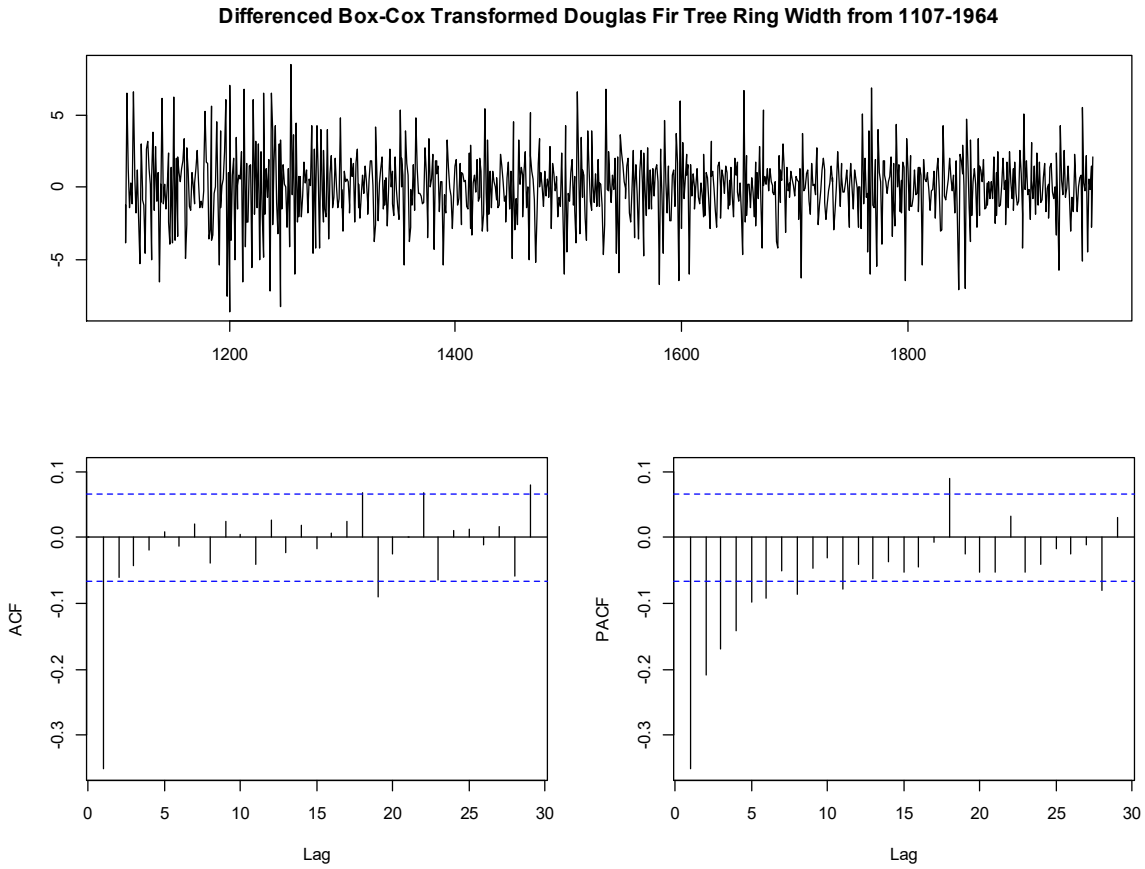


Let us here first focus on modelling the non-differenced data. The analysis of ACF/PACF above suggests using an $ARMA(2,0)$ or $ARMA(1,1)$ as the most parsimonius models, a slightly bigger option would be to use an $ARMA(2,1)$. The residuals of all these look similar and acceptable; the lowest AIC value is achieved with the $ARMA(2,1)$ which is also the model that `auto.arima()` suggests.

```
> fit <- auto.arima(tdf, max.p=5, max.q=5, ic="aic",
                    stationary=T, allowmean=T, stepwise=T)
> fit
Series: tdf; ARIMA(2,0,1) with non-zero mean
Coefficients:
         ar1      ar2      ma1     mean
      1.1333  -0.2163  -0.6973  12.9683
s.e.  0.1044   0.0743   0.0933   0.2605
sigma^2 estimated as 4.46:  log likelihood=-1857.11
AIC=3724.22   AICc=3724.29   BIC=3747.99
```

The settings in the `auto.arima()` search were such that only stationary models (including an intercept) are allowed. The maximum order for p and q equals 5, but also $p+q \leq 5$ due to the `max.order=5` default setting in the function. If larger models are desired, then this needs to be adjusted accordingly. Moreover, for a full search over all possible models, we need setting `stepwise=FALSE`, which may change the output. We now inspect the differenced data.

```
> tsdisplay(diff(tdf), points=FALSE, main="…")
```



There is a clear cut-off in the ACF at lag 1. In the PACF, there is some decay, perhaps with an additional cut-off at lag 1. Hence, the most plausible parsimonious integrated models include the $ARIMA(0,1,1)$ and the $ARIMA(1,1,1)$. The former cannot capture the dependencies in a reasonable way, the residuals are still correlated and violate the White Noise assumption. The $ARIMA(1,1,1)$ is much better in this regard. However, its AIC value is worse than the one of the $ARIMA(2,0,1)$ considered previously. We again employ `auto.arima()` for a non-stepwise grid search over all $ARIMA(p,1,q)$ with $p,q \leq 5$ and $p+q \leq 5$. Since we want to avoid a drift-term and directly work on the differenced data, we have to set `allowmean=FALSE`.

```
> fit <- auto.arima(diff(tdf), max.p=5, max.q=5,
                    stationary=TRUE, allowmean=FALSE,
                    stepwise=FALSE, ic="aic")
```

```
> fit
Series: diff(tdf)
ARIMA(2,0,1) with zero mean
Coefficients:
         ar1     ar2     ma1
      0.4219  0.1249  -0.961
s.e.  0.0484  0.0460   0.032
sigma^2 estimated as 4.557:  log likelihood=-1865.15
AIC=3738.29   AICc=3738.34   BIC=3757.31
```

Somewhat surprisingly, this yields an $ARIMA(2,1,1)$ which is not really obvious from the PACF. So we end up with a number of different models whose residuals look reasonable and have AIC values that are quite close. Hence, a decision is far from being easy. While for the model order, the choice is somewhat arbitrary, the decision for a pure or integrated $ARMA$ is much more important. This is where practical aspects, i.e. the meaning of the model should come into play as well. With a stationary $ARMA(p,q)$, we would here focus more on the long-term aspects of the series, i.e. the climatic changes that happen over decades or even centuries. If the data are differenced, we consider the changes in growth from year to year. This puts the focus on the bio-chemical aspect, while climate change is ruled out. Not surprisingly, the autocorrelation among the differenced data is strongly negative at lag 1. This means that a big positive change in growth is more likely to be followed by a negative change in growth and vice versa. Hence this model focuses more on the recovery of the tree after strong resp. weak growth in one year versus the next. Hence it would not primarily be the climate which is modelled, but more the bio-chemical processes within the tree. Thus, it is also a matter of the applied research question which of the two models is more suited.

6.2 SARIMA Models

After becoming acquainted with the $ARIMA$ models, it is quite natural to ask for an extension to seasonal series; especially, because we learned that differencing at a lag equal to the period s does remove seasonal effects, too. Suppose we have a series X_t with monthly data. Then, series

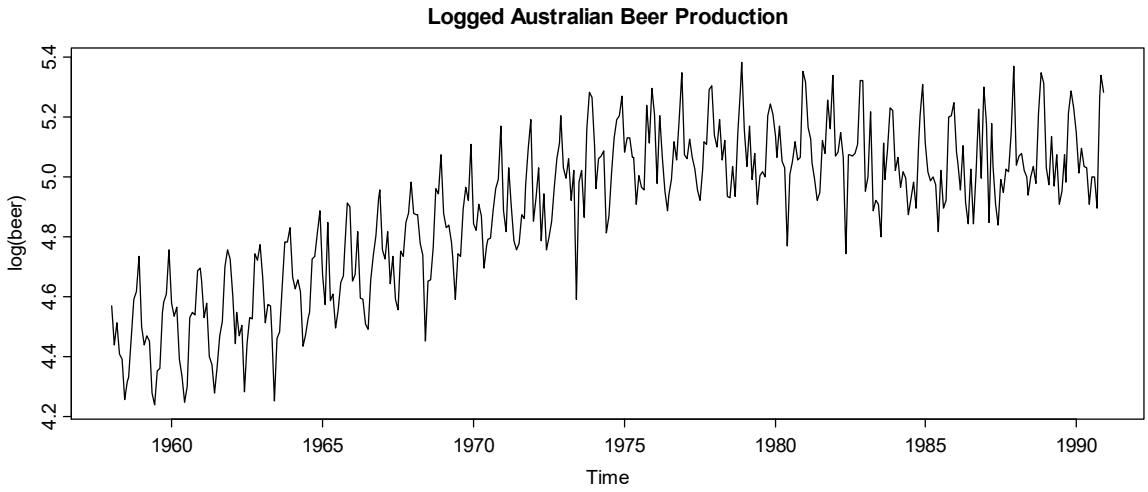
$$Y_t = X_t - X_{t-12} = (1-B^{12})X_t$$

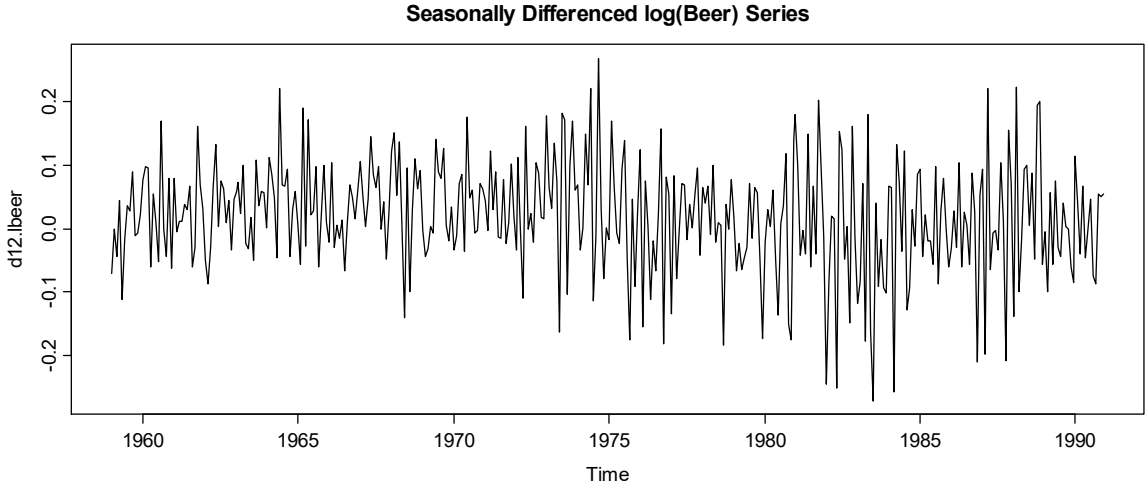
usually has the seasonality removed. However, it is quite often the case that the result has not yet constant global mean, and thus, some further differencing at lag 1 is required to achieve stationarity:

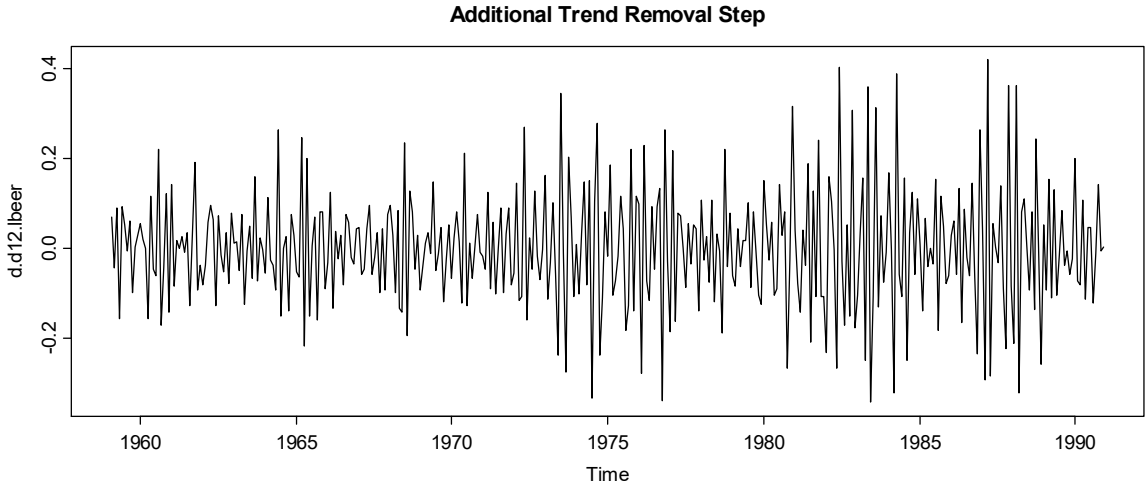
$$Z_t = Y_t - Y_{t-1} = (1-B)Y_t = (1-B)(1-B^{12})X_t = X_t - X_{t-1} - X_{t-12} + X_{t-13}$$

We illustrate this using the Australian beer production series that we had already considered in section 4. It has monthly data that range from January 1958 to December 1990. A log-transformation to stabilize the variance is indicated. We display the transformed series X_t, the seasonally differenced series Y_t and finally the seasonal-trend differenced series Z_t.

```
> www <- "http://staff.elena.aut.ac.nz/Paul-Cowpertwait/ts/"
> dat <- read.table(paste(www,"cbe.dat",sep="", header=T)
> beer      <- ts(dat$beer, start=1958, freq=12)
> d12.lbeer <- diff(log(beer), lag=12)
> d.d12.lbeer <- diff(d12.lbeer)
> plot(log(beer))
> plot(d12.lbeer)
> plot(d.d12.lbeer))
```

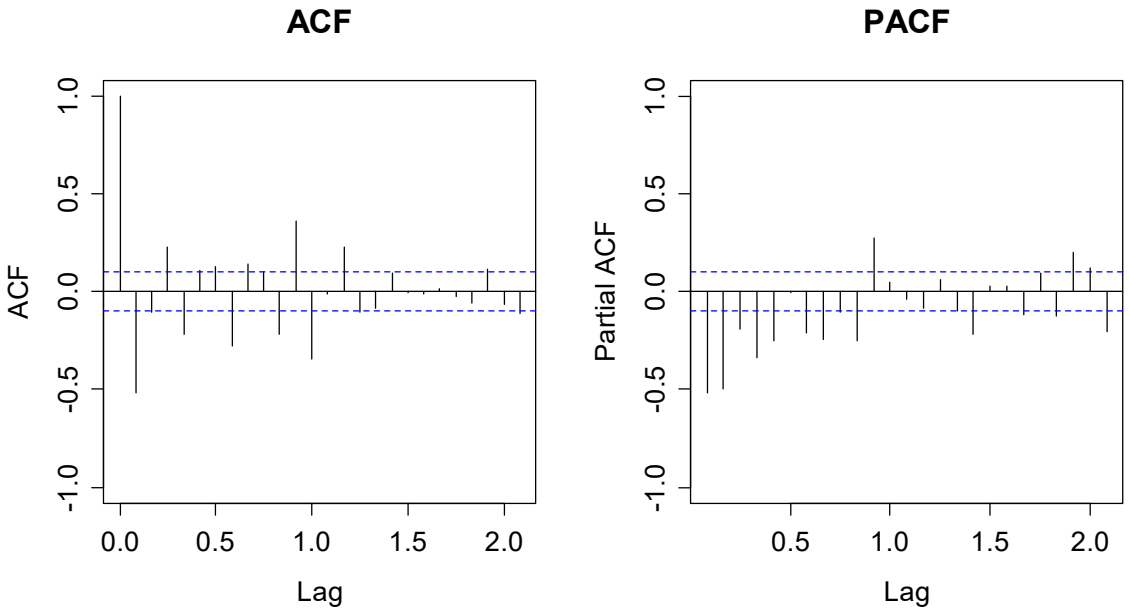






While the two series X_t and Y_t are non-stationary, the last one, Z_t may be, although it is a bit debatable whether the assumption of constant variation is violated or not. We proceed by analyzing ACF and PACF of series Z_t.

```
> par(mfrow=c(1,2))
> acf(d.d12.lbeer, ylim=c(-1,1))
> pacf(d.d12.lbeer, ylim=c(-1,1), main="PACF")
```



There is very clear evidence that series Z_t is serially dependent, and we could try an $ARMA(p,q)$ to model this dependence. As for the choice of the order, this is not simple on the basis of the above correlograms. They suggest that high values for p and q are required, and model fitting with subsequent residual analysis and AIC inspection confirm this: $p=14$ and $q=11$ yield a good result.

It is (not so much in the above, but generally when analyzing data of this type) quite striking that the ACF and PACF coefficients have large values at multiples of the period s. This is very typical behavior for seasonally differenced series, in fact it originates from the evolution of resp. changes in the seasonality over the years. A simple model accounting for this is the so-called *airline model*:

$$\begin{aligned} Z_t &= (1+\beta_1 B)(1+\xi_1 B^{12})E_t \\ &= (1+\beta_1 B+\xi_1 B^{12}+\beta_1\xi_1 B^{13})E_t \\ &= E_t+\beta_1 E_{t-1}+\xi_1 E_{t-12}+\beta_1\xi_1 E_{t-13} \end{aligned}$$

This is a $MA(13)$ model, where many of the coefficients are equal to 0. Because it was made up of an $MA(1)$ with B as an operator in the characteristic polynomial, and another one with B^s as the operator, we call this a $SARIMA(0,1,1)(0,1,1)^{12}$. This idea can be generalized: we fit AR and MA parts with both B and B^s as operators in the characteristic polynomials, which again results in a high order $ARMA$ model for Z_t.

Definition: A series X_t follows a $SARIMA(p,d,q)(P,D,Q)^s$ process if the following equation holds:

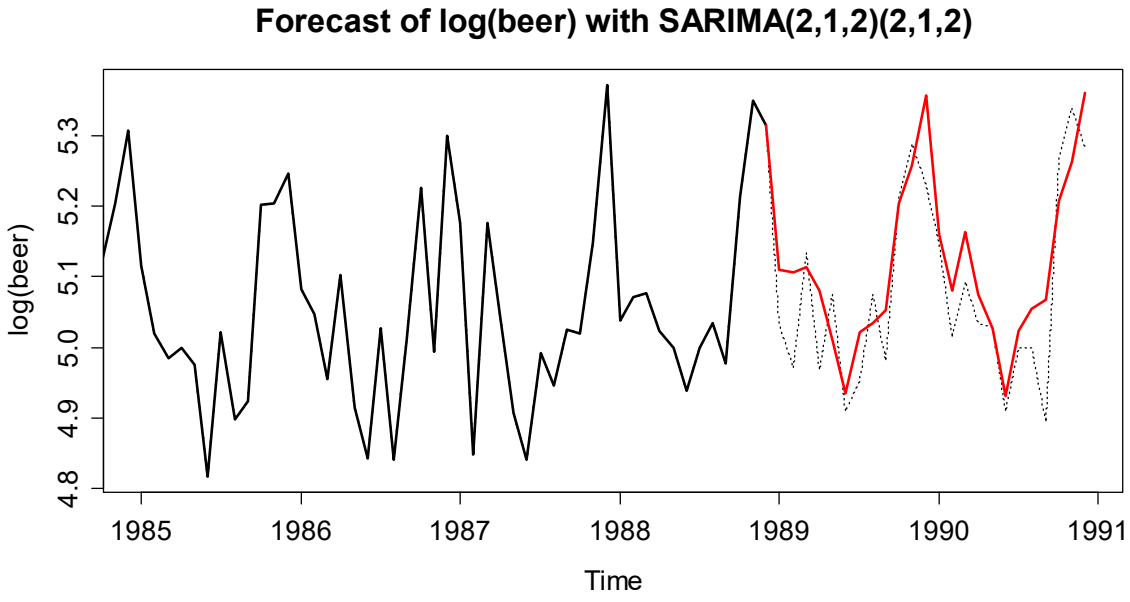
$$\Phi(B)\Phi_S(B^s)Z_t = \Theta(B)\Theta_S(B^s)E_t,$$

where series Z_t originated from X_t after appropriate seasonal and trend differencing, i.e. $Z_t = (1-B)^d(1-B^s)^D$.

Fortunately, it turns out that usually $d = D = 1$ is enough. As for the model orders p, q, P, Q, the choice can be made on the basis of ACF and PACF, by searching for cut-offs. Mostly, these are far from evident, and thus, an often applied alternative is to consider all models with $p, q, P, Q \leq 2$ and doing an AIC-based grid search, function `auto.arima()` may be very handy for this task.

For our example, the $SARIMA(2,1,2)(2,1,2)^{12}$ has the lowest value and also shows satisfactory residuals, although it seems to perform slightly less well than the $SARIMA(14,1,11)(0,1,0)^{12}$. The R-command for the former is:

```
> fit <- arima(log(beer), order=c(2,1,2), seasonal=c(2,1,2))
```



As it was mentioned in the introduction to this section, one of the main advantages of $ARIMA$ and $SARIMA$ models is that they allow for quick and convenient forecasting. While this will be discussed in depth later in section 8, we here provide a first example to show the potential.

From the logged beer production data, the last 2 years were omitted before the $SARIMA$ model was fitted to the (shortened) series. On the basis of this model, a 2-year-forecast was computed, which is displayed by the red line in the plot above. The original data are shown as a solid (insample, 1958-1988) line, respectively as a dotted (out-of-sample, 1989-1990) line. We see that the forecast is reasonably accurate.

To facilitate the fitting of $SARIMA$ models, we finish this chapter by providing some guidelines:

1) Perform seasonal differencing on the data. The lag s is determined by the periodicity of the data, for the order, in most cases $D = 1$ is sufficient.

2) Do a time series plot of the output of the above step. Decide whether it is stationary, or whether additional differencing at lag 1 is required to remove a potential trend. If not, then $d = 0$, and proceed. If yes, $d = 1$ is enough for most series.

3) From the output of step 2, i.e. series Z_t, generate ACF and PACF plots to study the dependency structure. Look for coefficients/cut-offs at low lags that indicate the direct, short-term dependency and determine orders p and q. Then, inspect coefficients/cut-offs at multiples of the period s, which imply seasonal dependency and determine P and Q.

4) Fit the model using procedure `arima()`. In contrast to $ARIMA$ fitting, this is now exclusively done on the original series, with setting the two arguments `order=c(p,d,q)` and `seasonal=c(P,D,Q)` accordingly.

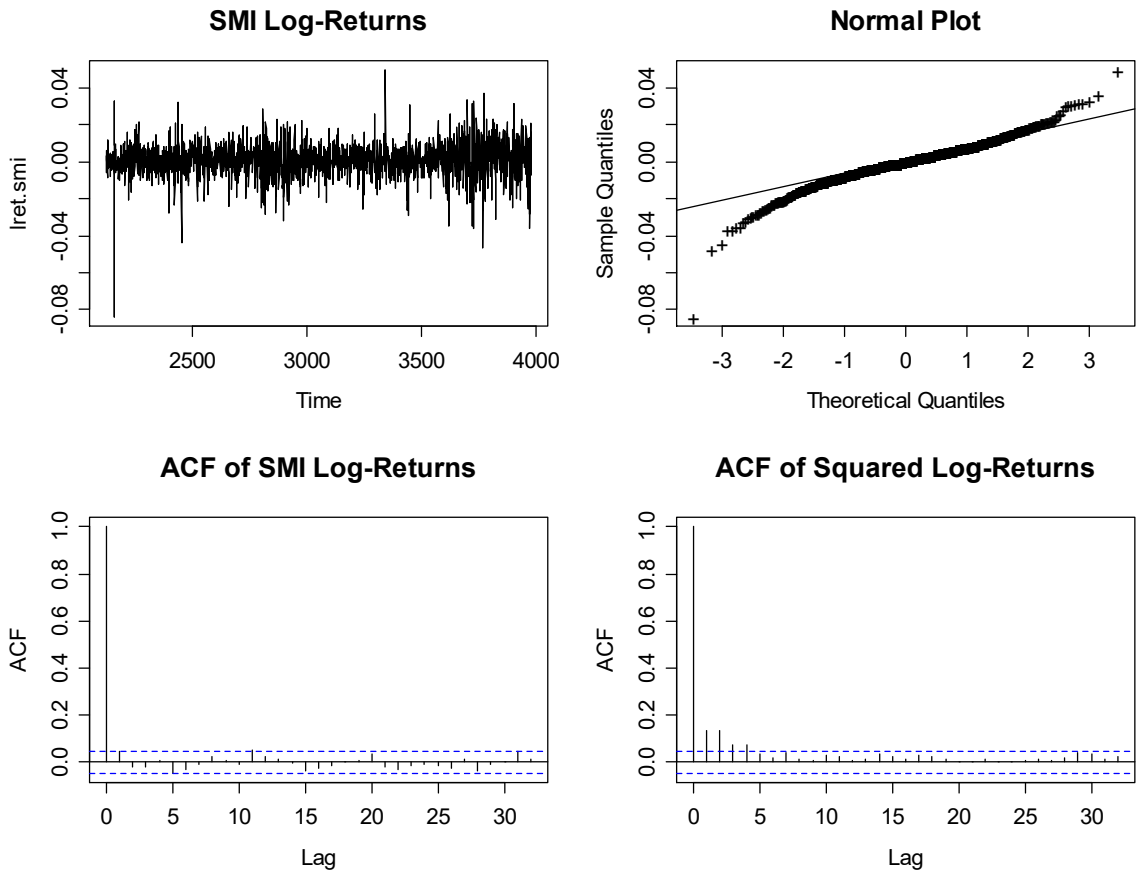
5) Check the accuracy of the fitted model by residual analysis. These must look like White Noise. If thus far, there is ambiguity in the model order, AIC analysis can serve to come to a final decision.

Next, we turn our attention to series that have neither trend nor seasonality, but show serial dependence in the conditional variance.

6.3 ARCH/GARCH Models

In this chapter, we consider the SMI log-returns that were already presented in section 1.2.4. By closer inspection of the time series plot, we observe some long-tailedness, and also, the series exhibits periods of increased variability, which is usually termed volatility in the (financial) literature. We had previously observed series with non-constant variation, such as the oil prices and beer production in the previous sections. Such series, where the variance increases with increasing level of the series, are called *heteroskedastic*, and can often be stabilized using a log-transformation.

However, that matter is different with the SMI log-returns: here, there are periods of increased volatility, and thus the conditional variance of the series is serially correlated, a phenomenon that is called *conditional heteroskedasticity*. This is not a violation of the stationarity assumption, but some special treatment for this type of series is required. Furthermore, the ACF of such series typically does not differ significantly from White Noise. Still, the data are not *iid*, which can be shown with the ACF of the squared observations. With the plots on the next page, we illustrate the presence of these stylized facts for the SMI log-returns:



6.3.1 The ARCH and GARCH Models

In order to account for volatility, we require a model that reflects the dependency in the conditional variance. We operate under the assumption that:

$$X_t = \mu_t + E_t,$$

where the disturbance term E_t can be rewritten as $\sigma_t W_t$: $X_t = \mu_t + \sigma_t W_t$. Here, W_t is a White Noise innovation and $\sigma_t = Var(X_t \mid X_{t-1}, X_{t-2}, ...)$ is the conditional variance that is assumed to be non-constant. Finally $\mu_t = E[X_t \mid X_{t-1}, X_{t-2}, ...]$ is the conditional expectation as before. It is perfectly allowed to have both dependence in the conditional mean and variance, and hence a mixture of $ARMA$ and $GARCH$ processes. However, for simplicity we assume throughout this section that both the conditional and the global mean are zero: $\mu = \mu_t = 0$ and thus $X_t = \sigma_t W_t$.

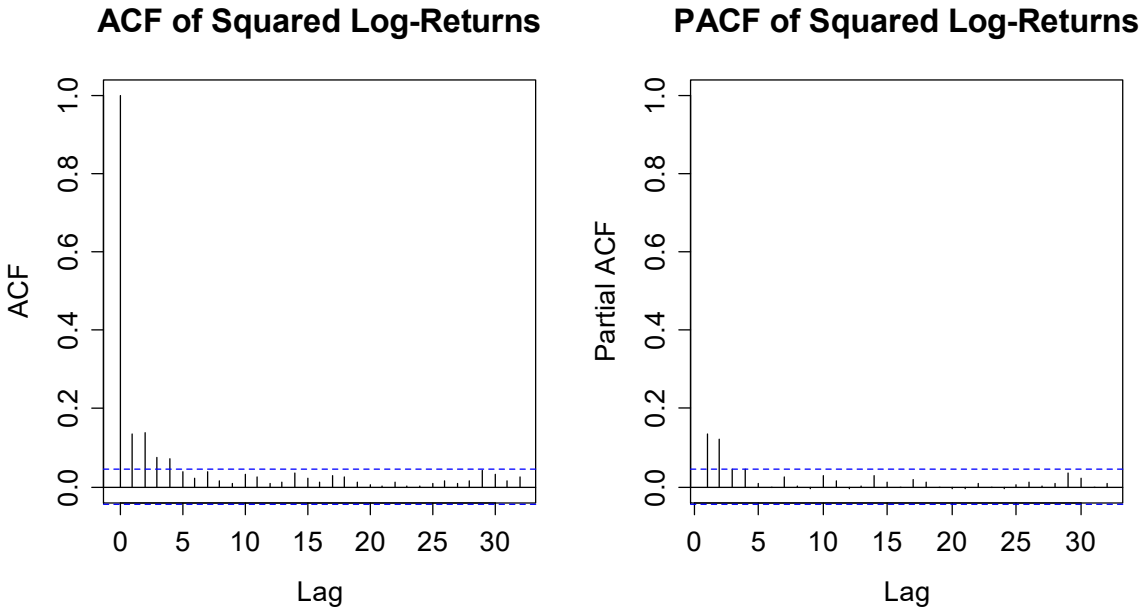
The most simple and intuitive way of doing this is to use an autoregressive model for the variance process. Thus, a series E_t is first-order autoregressive conditional heteroskedastic, denoted as $ARCH(1)$, if:

$$E_t = W_t \sqrt{\alpha_0 + \alpha_1 E_{t-1}^2}\ .$$

Here, W_t is a White Noise process with mean zero and unit variance. The two parameters α_0, α_1 are the model coefficients. An $ARCH(1)$ process shows volatility, as can easily be derived:

$$\begin{aligned} Var(E_t) &= E[E_t^2] \\ &= E[W_t^2]E[\alpha_0 + \alpha_1 E_{t-1}^2] \\ &= E[\alpha_0 + \alpha_1 E_{t-1}^2] \\ &= \alpha_0 + \alpha_1 \cdot Var(E_{t-1}) \end{aligned}$$

Note that this derivation is based on $E[W_t^2] = 1$ and $E[E_t] = E[W_t] = 0$. As we had aimed for, the variance of an $ARCH(1)$ process behaves just like an $AR(1)$ model. Hence, the decay in the autocorrelations of the squared residuals should indicate whether an $ARCH(1)$ is appropriate or not.



In our case, the analysis of ACF and PACF of the squared log-returns suggests that the variance may be well described by an $AR(2)$ process. This is not what we had discussed, but an extension exists. An $ARCH(p)$ process is defined by:

$$E_t = W_t \sqrt{\alpha_0 + \sum_{i=1}^{p} \alpha_p E_{t-i}^2}$$

Fitting in **R** can be done using procedure `garch()`. This is a more flexible tool, which also allows for fitting GARCH processes, as discussed below. The command in our case is as follows:

```
> fit <- garch(lret.smi, order = c(0,2), trace=FALSE); fit
Call: garch(x = lret.smi, order = c(0, 2), trace = FALSE)
Coefficient(s):
       a0          a1          a2
6.568e-05  1.309e-01  1.074e-01
```

For verifying appropriate fit of the $ARCH(2)$, we need to check the residuals of the fitted model. This includes inspecting ACF and PACF for both the “normal” and the squared residuals. We here do without showing plots, but the $ARCH(2)$ is OK.

A nearby question is whether we can also use an $ARMA(p,q)$ process for describing the dependence in the variance of the process. The answer is yes. This is what a $GARCH(p,q)$ model does. A series $E_t = W_t\sqrt{H_t}$ is $GARCH(p,q)$ if:

$$H_t = \alpha_0 + \sum_{i=1}^{q} \alpha_i E_{t-i}^2 + \sum_{j=1}^{p} \beta_j H_{t-j}$$

6.3.2 Use of GARCH Models

GARCH models are useless for the prediction of the level of a series, i.e. for the SMI log-returns, they do not provide any idea whether the stocks’ value will increase or decrease on the next day. However, they allow for a more precise understanding in the (up or down) changes that might be expected during the next day(s). This allows stockholders to adjust their position, so that they do not take any unduly risks.

7 Time Series Regression

7.1 What Is the Problem?

It is often the case that we aim for describing some time series Y_t with a linear combination of some explanatory series $x_{t1},...,x_{tp}$. As we will see below, the predictors can either be true covariates, or terms that are derived from time, as for example linear trends or seasonal effects. We employ the universally known linear model for linking the response series with the predictors:

$$Y_t = \beta_0 + \beta_1 x_{t1} + ... + \beta_p x_{tp} + E_t$$

The regression coefficients $\beta_1,...,\beta_p$ are usually estimated with the least squares algorithm, for which an error term with zero expectation, constant variation and no correlation is assumed. However, if response and predictors are time series with autocorrelation, the last condition often turns out to be violated, though this is not always the case.

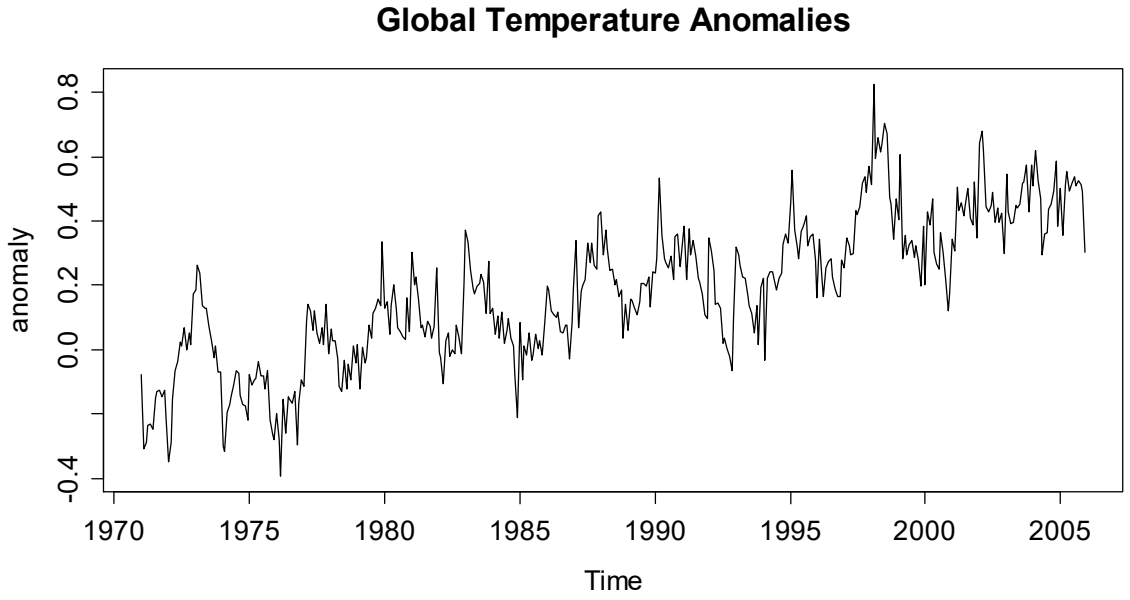
Now, if we are facing a (time series) regression problem *with* correlated errors, the estimates $\hat{\beta}_j$ will remain being unbiased, but the least squares algorithm is no longer efficient. Or in other words: more precisely working estimators exist. Even more problematic are the standard errors of the regression coefficients $\hat{\beta}_j$: they are often grossly wrong in case of correlated errors. As they are routinely underestimated, inference on the predictors often yields spurious significance, i.e. one is prone to false conclusions from the analysis.

Thus, there is a need for more general linear regression procedures that can deal with serially correlated errors, and fortunately, they exist. We will here discuss the simple, iterative Cochrane-Orcutt procedure, and the Generalized Least Squares method, which marks a theoretically more sound approach to regression with correlated errors. But first, we present some time series regression problems to illustrating what we are dealing with.

Example 1: Global Temperature

In climate change studies time series with global temperature values are analyzed. The scale of measurement is anomalies, i.e. the difference between the monthly global mean temperature versus the overall mean between 1961 and 1990. The data can be obtained at http://www.cru.uea.ac.uk/cru/data or in file `anomalies.rda`. For illustration, we restrict to a period from 1971 to 2005 which corresponds to a series of 420 records. For a time series plot, see the next page.

```
> ## Time Series Plot
> load("anomalies.rda")
> plot(anomalies, ylab="anomaly")
> title("Global Temperature Anomalies")
```



There is a clear trend which seems to be linear. Despite being monthly measured, the data show no evident seasonality. This is not overly surprising, since we are considering a global mean, i.e. the season should not make for a big difference. But on the other hand, because the landmass is not uniformly distributed over both halves of the globe, it could still be present. It is natural to try a season-trend-decomposition for this series. We will employ a parametric model featuring a linear trend plus a seasonal factor.

$$Y_t = \beta_0 + \beta_1 \cdot t + \beta_2 \cdot 1_{[month="Feb"]} + ... + \beta_{12} \cdot 1_{[month="Dec"]} + E_t,$$

where $t = 1,\ldots,420$ and $1_{[month="Feb"]}$ is a dummy variable that takes the value 1 if an observation is from month February, and zero else. Clearly, this is a time series regression model. The response Y_t is the global temperature anomalies, and even the predictors, i.e. the time and the dummies, can be seen as time series, even if simple ones.

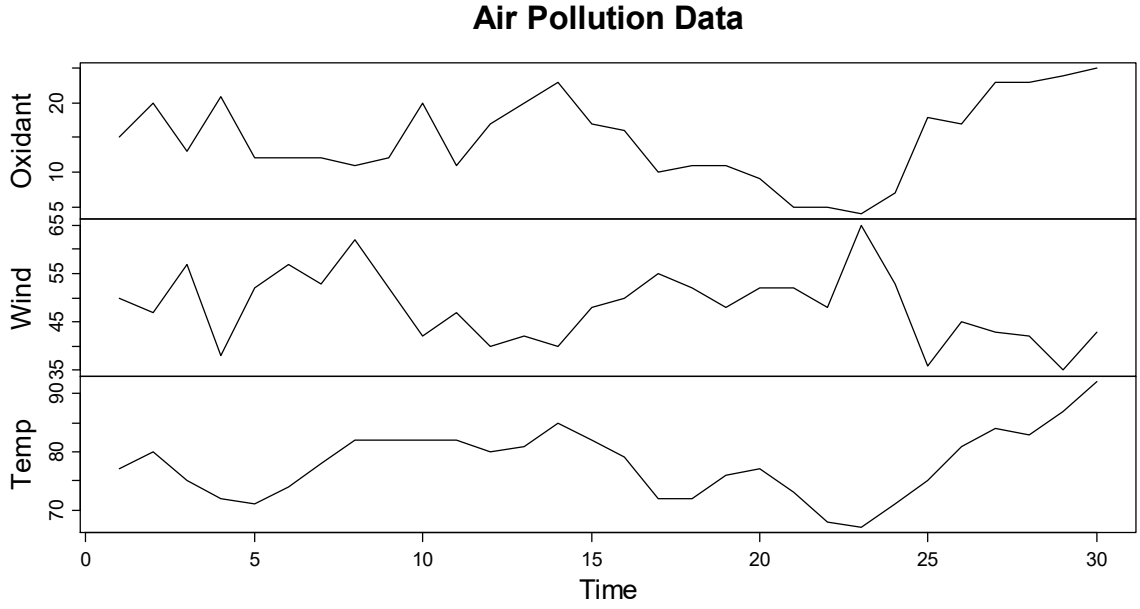
As we have seen previously, the goal with such parametric decomposition models is to obtain a stationary remainder term E_t. But stationary does not necessarily mean White Noise, and in practice it often turns out that E_t shows some serial correlation. Thus, if the regression coefficients are obtained from the least squares algorithm, we apparently feature some violated assumption.

This violation can be problematic, even in an applied setting: a question of utter importance with the above series is whether trend and seasonal effect are significantly present. It would be nice to answer such questions using the inference approaches (tests and confidence intervals) that linear regression provides. However, for obtaining reliable inference results, we need to account for the correlation among the errors. We will show this below, after introducing some more examples and theory.

Example 2: Air Pollution

In this second example, we consider a time series that is stationary, and where the regression aims at understanding the series, rather than decomposing it into some deterministic and random components. We examine the dependence of a photochemical pollutant (morning maximal value) on the two meteorological variables wind and temperature. The series, which constitute of 30 observations taken on consecutive days, come from the Los Angeles basin. They are not publicly available, but can be obtained from the lecturer upon request.

```
> ## Loading the data
> load("oxidant.rda")

> ## Visualizing the data
> plot(dat, main="Air Pollution Data")
```



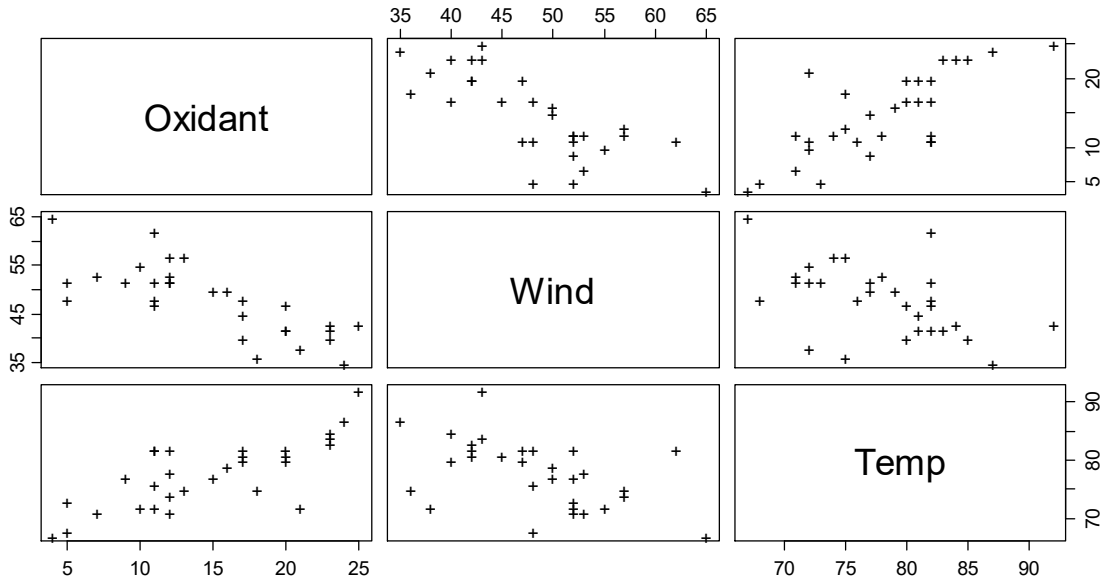
There is no counterevidence to stationarity for all three series. What could be the goal here? Well, we aim for enhancing the understanding of how the pollution depends on the meteorology, i.e. what influence wind and temperature have on the oxidant values. We can naturally formulate the relation with a linear regression model:

$$Y_t = \beta_0 + \beta_1 x_{t1} + \beta_2 x_{t2} + E_t .$$

In this model, the response Y_t is the oxidant, and as predictors we have x_{t1}, wind, and x_{t2}, the temperature. For the index, we have $t = 1,...,30$, and obviously, this is a time series regression model.

For gaining some more insight with these data, it is also instructive to visualize the data using a pairs plot, as shown on the next page. There, a strong, positive linear association is recognizable between pollutant and the temperature. In contrast,

there is a negative linear relation between pollutant and wind. Lastly, between the predictors wind and temperature, there is not much of a correlation. This data structure is not surprising because wind causes a stronger movement of the air and thus the pollutant is "better" distributed. Also, the wind causes some cooling.



For achieving our practical goals with this dataset, we require precise and unbiased estimates of the regression coefficients β_1 and β_2. Moreover, we might like to give some statements about the significance of the predictors, and thus, we require some sound standard errors for the estimates. However, also with these data, it is well conceivable that the error term E_t will be serially correlated. Thus again, we will require some procedure that can account for this.

Time Series Regression Model

The two examples have shown that time series regression models do appear when decomposing series, but can also be important when we try to understand the relation between response and predictors with measurements that were taken sequentially. Generally, with the model

$$Y_t = \beta_0 + \beta_1 x_{t1} + ... + \beta_p x_{tp} + E_t$$

we assume that the influence of the series $x_{t1},\ldots,x_{tp}$ on the response Y_t is simultaneous. Nevertheless, lagged variables are also allowed, i.e. we can also use terms such as $x_{(t-k);j}$ with $k > 0$ as predictors. While this generalization can be easily built into our model, one quickly obtains models with many unknown parameters. Thus, when exploring the dependence of a response series to lags of some predictor series, there are better approaches than regression. In particular, this is the cross correlations and the transfer function model, which will be exhibited in later chapters of this script.

In fact, there are not many restrictions for the time series regression model. As we have seen, it is perfectly valid to have non-stationary series as either the response or as predictors. However, it is crucial that there is no feedback from Y_t to the x_{tj}. Additionally, the error E_t must be independent of the explanatory variables, but it may exhibit serial correlation.

7.2 Finding Correlated Errors

When dealing with a time series regression problem, we first always assume uncorrelated errors and start out with an ordinary least squares regression. Based on its residuals, the assumption can be verified, and if necessary, action can be taken. For identifying correlation among the residuals, we analyze their time series plot, ACF and PACF.

Example 1: Global Temperature

Our goal is the decomposition of the global temperature series into a linear trend plus some seasonal factor. First and foremost, we prepare the data:

```
> num.temp <- as.numeric(anomalies)
> num.time <- as.numeric(time(anomalies))
> mn01     <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun")
> mn02     <- c("Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
> month    <- factor(cycle(my.temp), labels=c(mn01, mn02))
> dat      <- data.frame(temp=num.temp, time=num.time, month)
```

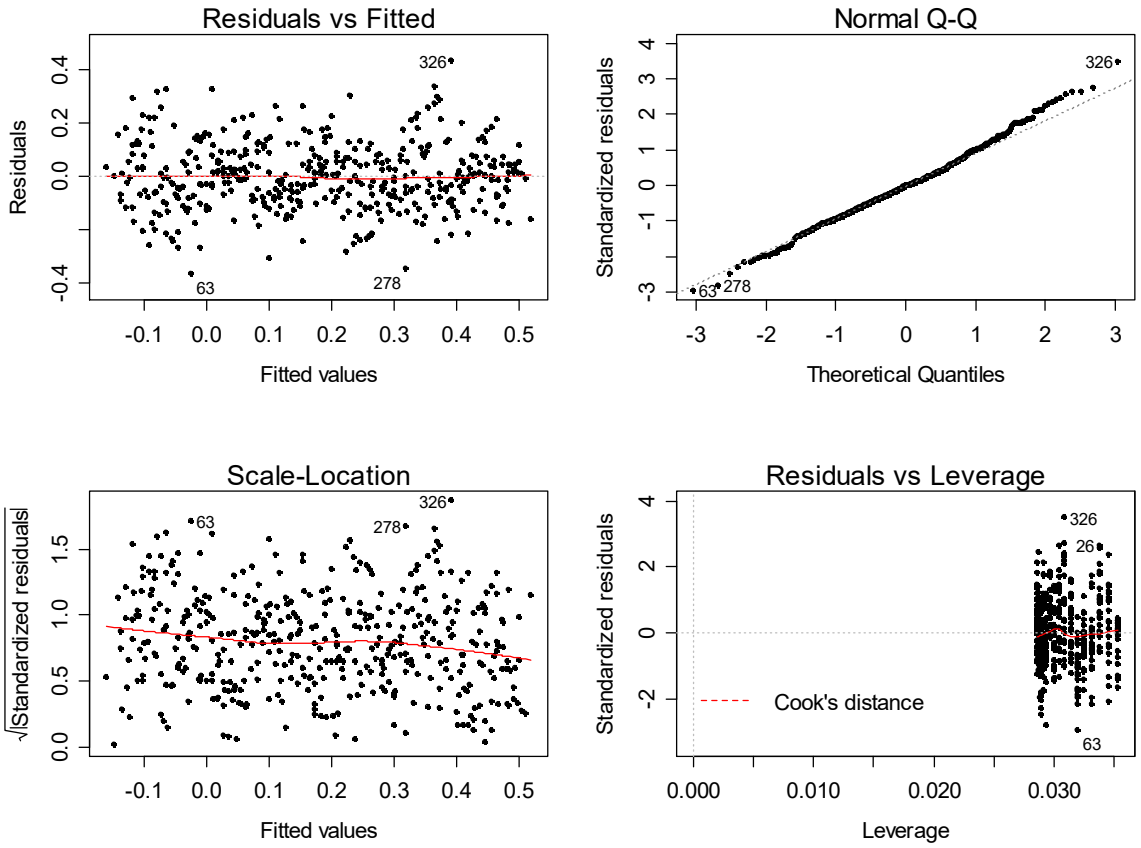
The regression model is the estimated with **R**'s function `lm()`. The summary function returns estimates, standard errors plus the results from some hypothesis tests. It is important to notice that all of these results are in question should the errors turn out to be correlated.

```
> fit.lm <- lm(temp ~ time + season, data=dat)
> summary(fit.lm)

Call:
lm(formula = temp ~ time + season, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-0.36554 -0.07972 -0.00235  0.07497  0.43348

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.603e+01  1.211e+00 -29.757   <2e-16 ***
time         1.822e-02  6.089e-04  29.927   <2e-16 ***
seasonFeb    6.539e-03  3.013e-02   0.217   0.8283
seasonMar   -1.004e-02  3.013e-02  -0.333   0.7392
seasonApr   -1.473e-02  3.013e-02  -0.489   0.6252
seasonMay   -3.433e-02  3.013e-02  -1.139   0.2552
```

```
seasonJun   -2.628e-02  3.013e-02  -0.872   0.3836
seasonJul   -2.663e-02  3.013e-02  -0.884   0.3774
seasonAug   -2.409e-02  3.013e-02  -0.799   0.4245
seasonSep   -3.883e-02  3.013e-02  -1.289   0.1982
seasonOct   -5.212e-02  3.013e-02  -1.730   0.0844 .
seasonNov   -6.633e-02  3.013e-02  -2.201   0.0283 *
seasonDec   -4.485e-02  3.013e-02  -1.488   0.1374
---

Residual standard error: 0.126 on 407 degrees of freedom
Multiple R-squared: 0.6891,  Adjusted R-squared:  0.68
F-statistic: 75.18 on 12 and 407 DF,  p-value: < 2.2e-16
```

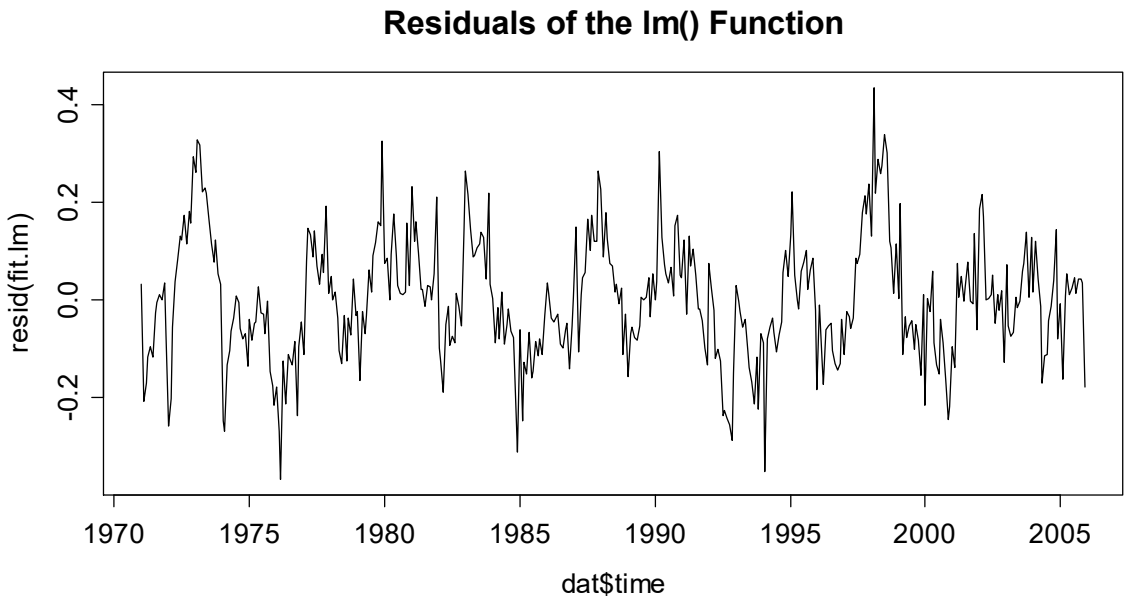
As the next step, we need to perform some residual diagnostics. The `plot()` function, applied to a regression fit, serves as a check for zero expectation, constant variation and normality of the errors, and can give hints on potentially problematic leverage points.

```
> par(mfrow=c(2,2))
> plot(fit.lm, pch=20)
```



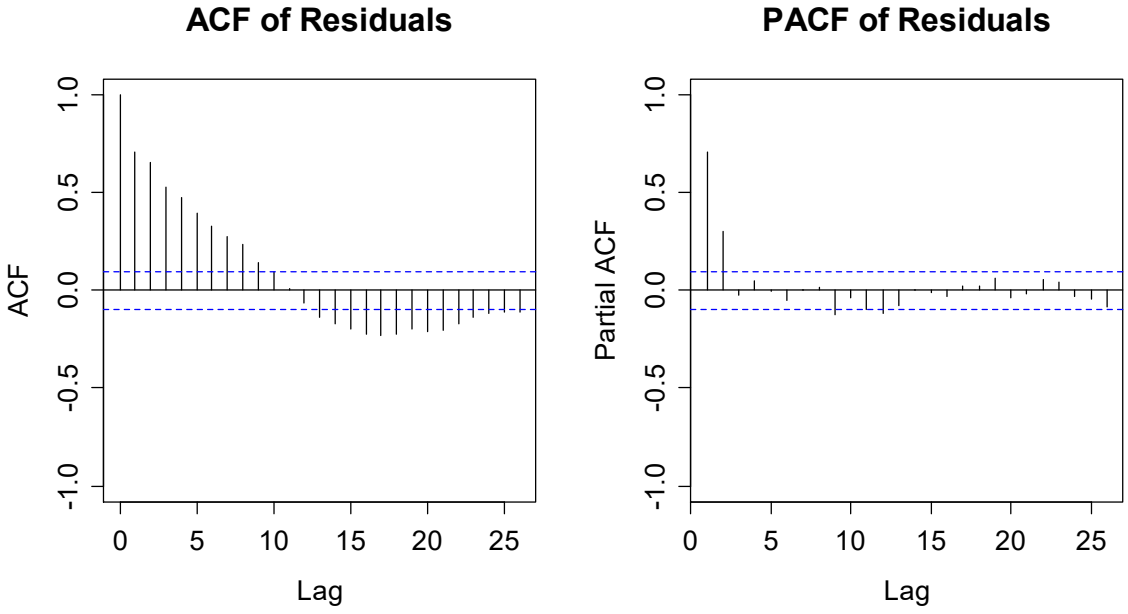
Except for some very slightly long tailed errors, which do not require any action, the residual plots look fine. What has not yet been verified is whether there is any serial correlation among the residuals. If we wish to see a time series plot, the following commands are useful:

```
> plot(time(anomalies), resid(fit.lm), type="l")
```



It is fairly obvious from the time series plot that the residuals are correlated. Our main tool for describing the dependency structure is the ACF and PACF plots, however. These are as follows:

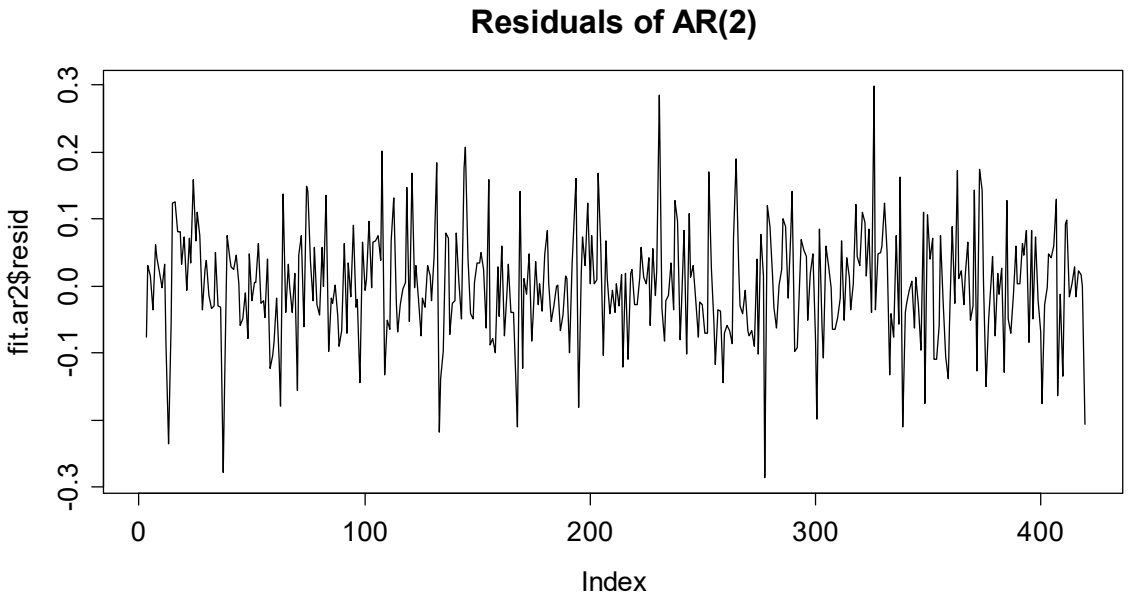
```
> par(mfrow=c(1,2))
> acf(resid(fit.lm), main="ACF of Residuals")
> pacf(resid(fit.lm), main="PACF of Residuals")
```

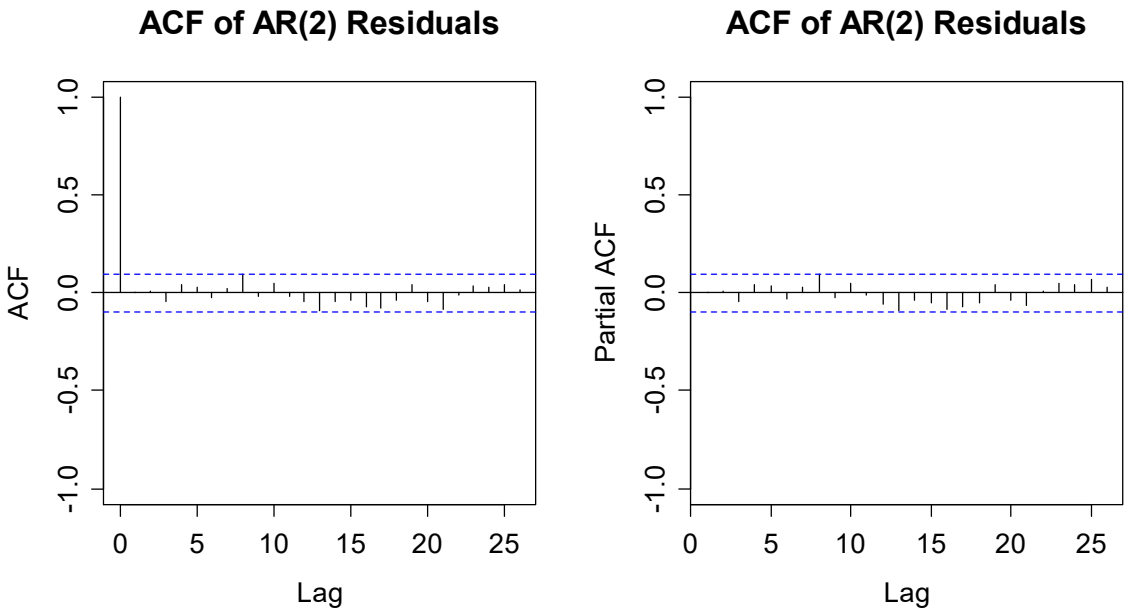


The ACF shows a rather slow exponential decay, whereas the PACF shows a clear cut-off at lag 2. With these stylized facts, it might well be that an $AR(2)$ model is a good description for the dependency among the residuals. We verify this:

```
> fit.ar2 <- ar.burg(resid(fit.lm)); fit.ar2
Call: ar.burg.default(x = resid(fit.lm))
Coefficients:
     1       2
0.4945  0.3036
Order selected 2  sigma^2 estimated as  0.00693
```

When using Burg's algorithm for parameter estimation and doing model selection by AIC, order 2 also turns out to be optimal. For verifying an adequate fit, we visualize the residuals from the $AR(2)$ model. These need to look like White Noise.





There is no contradiction to the White Noise hypothesis for the residuals from the $AR(2)$ model. Thus, we can summarize as follows: the regression model that was

used for decomposing the global temperature series into a linear trend and a seasonal factor exhibit correlated errors that seem to originate from an $AR(2)$ model. Theory tells us that the point estimates for the regression coefficients are still unbiased, but they are no longer efficient. Moreover, the standard errors for these coefficients can be grossly wrong. Thus, we need to be careful with the regression summary approach that was displayed above. And since our goal is inferring significance of trend and seasonality, we need to come up with some better suited method.

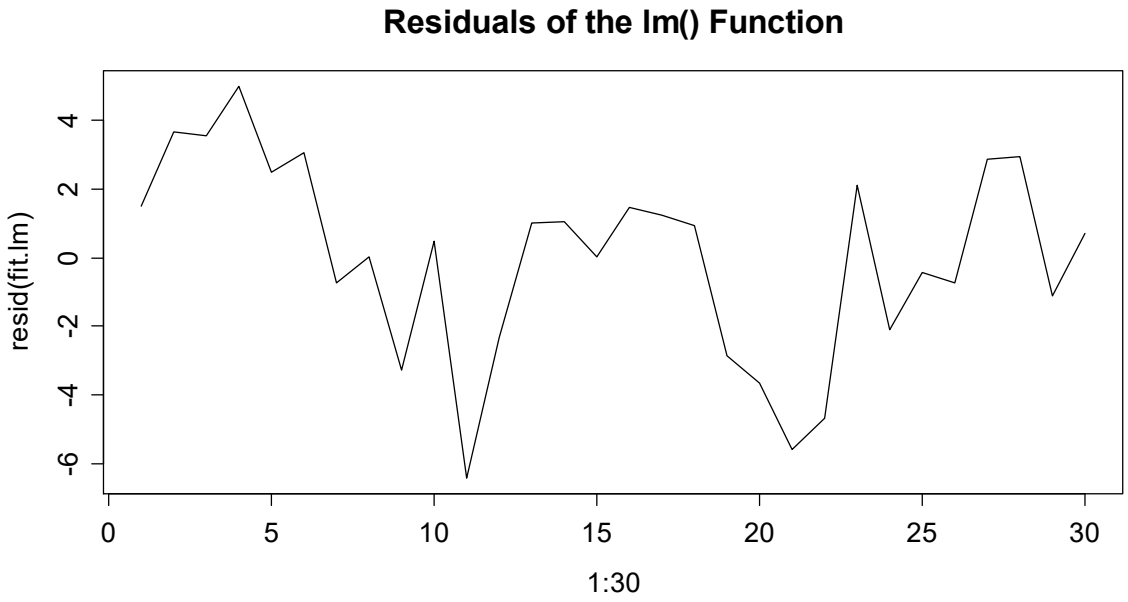
Example 2: Air Pollution

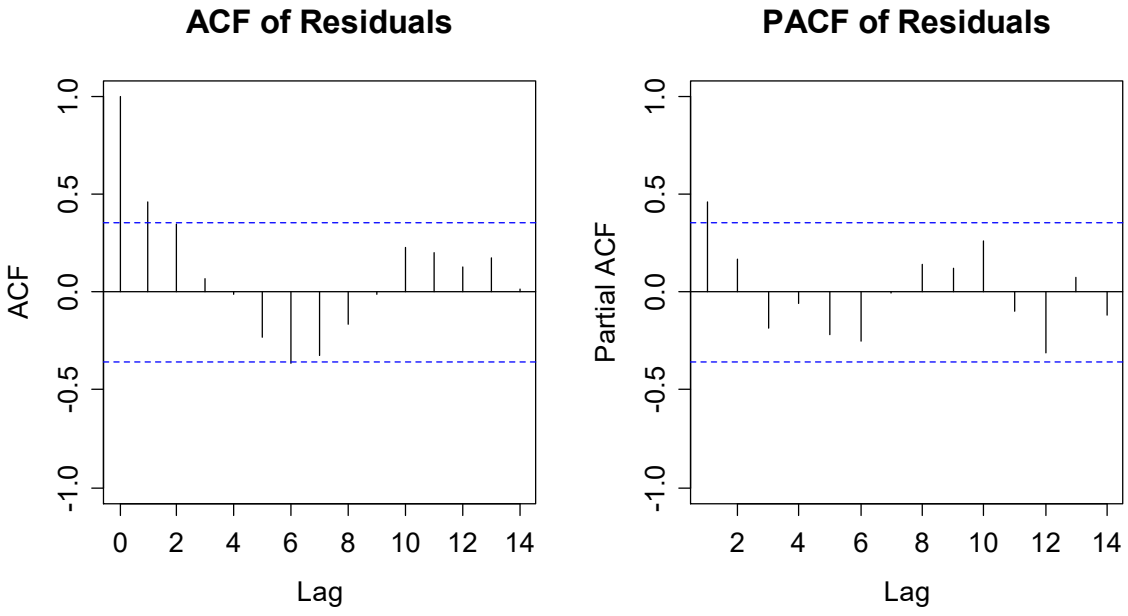
Now, we are dealing with the air pollution data. Again, we begin our regression analysis using the standard assumption of uncorrelated errors. Thus, we start out by applying the `lm()` function and printing the `summary()`.

```
> fit.lm <- lm(Oxidant ~ Wind + Temp, data=dat)
> summary(fit.lm)

Call:
lm(formula = Oxidant ~ Wind + Temp, data = dat)

Residuals:
    Min      1Q  Median      3Q     Max
-6.3939 -1.8608  0.5826  1.9461  4.9661

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -5.20334   11.11810  -0.468    0.644
Wind        -0.42706    0.08645  -4.940 3.58e-05 ***
Temp         0.52035    0.10813   4.812 5.05e-05 ***
---

Residual standard error: 2.95 on 27 degrees of freedom
Multiple R-squared: 0.7773,  Adjusted R-squared: 0.7608
F-statistic: 47.12 on 2 and 27 DF,  p-value: 1.563e-09
```

We will do without showing the 4 standard diagnostic plots, and here only report that they do not show any model violations. Because we are performing a time series regression, we also need to check for potential serial correlation of the errors. As before, this is done on the basis of time series plot, ACF and PACF:

```
> plot(1:30, resid(fit.lm), type="l")
> title("Residuals of the lm() Function")
> par(mfrow=c(1,2))
> acf(resid(fit.lm), ylim=c(-1,1), main="ACF of Residuals")
> pacf(resid(fit.lm), ylim=c(-1,1), main="PACF of Residuals")
```





Also in this example, the time series of the residuals exhibits serial dependence. Because the ACF shows an exponential decay and the PACF cuts off at lag 1, we hypothesize that an $AR(1)$ model is a good description. While the AIC criterion suggests an order of $p = 14$, the residuals of an $AR(1)$ show the behavior of White Noise. Additionally, using an $AR(14)$ would be spending too many degrees of freedom for a series with only 30 observations.

Thus, we can summarize that also in our second example with the air pollution data, we feature a time series regression that has correlated errors. Again, we must not communicate the above regression summary and for sound inference, we require more sophisticated models.

7.2.1 Durbin-Watson Test

For the less proficient user, hypothesis tests always seem like an attractive alternative to visual inspection of graphical output. This is certainly also the case when the task is identifying a potential serial correlation in a regression analysis. Indeed, there is a formal test that addresses the issue, called the Durbin-Watson test. While we will here briefly go into it, we do not recommend it for practical application.The Durbin-Watson test tests the null hypothesis $H_0 : \rho(1) = 0$ against the alternative $H_A : \rho(1) \neq 0$. The test statistic $\hat{D}$ is calculated as follows

$$\hat{D} = \frac{\sum_{t=2}^{n} (r_t - r_{t-1})^2}{\sum_{t=1}^{n} r_t^2}$$

where $r_t = y_t - \hat{y}_t$ is the residual from the regression model, observed at time t. There is an approximate relationship between the test statistic $\hat{D}$ and the autocorrelation coefficient at lag 1:

$$\hat{D} \approx 2(1 - \hat{\rho}(1))$$

The test statistic takes values between 0 if $r_t = r_{t-1}$ and 4 if $r_t = -r_{t-1}$. These extremes indicate perfect correlation of the residuals. Values around 2, on the other hand, are evidence for uncorrelated errors. The exact distribution of $\hat{D}$ is rather difficult to derive. However, we do not need to bother with this. The **R** package `lmtest` holds an implementation of the Durbin-Watson test with function `dwtest()`, where the p-value is either (for large sample size) determined by a normal approximation, or (for short series) by an iterative procedure.

Example 1: Global Temperature

```
> dwtest(fit.lm)
data:  fit.lm
DW = 0.5785, p-value < 2.2e-16
alt. hypothesis: true autocorrelation is greater than 0
```

Example 2: Air Pollution

```
> dwtest(fit.lm)
data:  fit.lm
DW = 1.0619, p-value = 0.001675
alt. hypothesis: true autocorrelation is greater than 0
```

Thus, the null hypothesis is rejected in both examples and we come to the same conclusion (“errors are correlated”) as with our visual inspection. It is very important to note that this is not necessary: In cases where the errors follow an $AR(p)$ process where $p > 1$ and $|\rho(1)|$ is small, the null hypothesis will not be rejected despite the fact that the errors are correlated.

7.3 Cochrane-Orcutt Method

The goal of this section is to solve the time series regression problem with errors that have an $AR(1)$ structure. This simple case is illustrative and helps to build the comprehension for more complicated error dependencies. We consider the Air Pollution example, where we have:

$Y_t = \beta_0 + \beta_1 x_{t1} + \beta_2 x_{t2} + E_t$ with $E_t = \alpha E_{t-1} + U_t$, where $U_t \sim N(0, \sigma_U^2)$ iid .

The fundamental trick, on which in fact all time series regression methods are based, will be presented here and now. We make the transformation:

$$Y_t' = Y_t - \alpha Y_{t-1}$$

Next, we plug-in the model equation and rearrange the terms. Finally, we build on the fundamental property that $E_t - \alpha E_{t-1} = U_t$. The result is:

$$\begin{aligned} Y_t' &= \beta_0 + \beta_1 x_{t1} + \beta_2 x_{t2} + E_t - \alpha(\beta_0 + \beta_1 x_{t-1,1} + \beta_2 x_{t-1,2} + E_{t-1}) \\ &= \beta_0(1-\alpha) + \beta_1(x_{t1} - \alpha x_{t-1,1}) + \beta_2(x_{t2} - \alpha x_{t-1,2}) + E_t - \alpha E_{t-1} \\ &= \beta_0' + \beta_1 x_{t1}' + \beta_2 x_{t2}' + U_t \end{aligned}$$

Obviously, this is a time series regression problem where the error term U_t is *iid*. Also note that both the response and the predictors have undergone a transformation. The coefficients however, are identical in both the original and the modified regression equation. For implementing this approach in practice, we require knowledge about the $AR(1)$ parameter α. Usually, it is not known previously. A simple idea to overcome this and solve the time series regression problem for the Air Pollution data is as follows:

1) Run OLS regression to obtain estimates $\hat{\beta}_0, ..., \hat{\beta}_p$
2) Estimate an $AR(1)$ on the OLS residuals to obtain $\hat{\alpha}$
3) Determine the prime variables $Y'; x'$ and derive $\hat{\beta}_0^*, \hat{\beta}_1, ..., \hat{\beta}_p$ by OLS

This procedure is know as the *Cochrane-Orcutt* iterative method. Please note that the estimates $\hat{\beta}_0', \hat{\beta}_1, ..., \hat{\beta}_p$ and their standard errors from the OLS regression in step 3) are sound and valid. But while the Cochrane-Orcutt procedure has its historical importance and is very nice for illustration, it lacks of a direct **R** implementation, and, as an iterative procedure, also of mathematical closedness and quality. The obvious improvement is to solve the prime regression problem by simultaneous Maximum-Likelihood estimation of all parameters:

$$\beta_0, ... \beta_p; \alpha; \sigma_U^2$$

This is possible and implemented in the **R** function `gls()`. Also, we need to be able to handle more complicated structure for the regression error E_t. For this, we resort to matrix notation, see the next section.

7.4 Generalized Least Squares

The ordinary least squares regression model assumes that $Var(E) = \sigma^2 I$, i.e. the covariance matrix of the errors is diagonal with identical values on the diagonal itself. As we have seen in our examples above, this is not a good model for time series regression. There, we rather have $Var(E) = \sigma^2 \Sigma$, where Σ reports the correlation among the errors. Using a Cholesky decomposition, we can write $\Sigma = SS^T$, where S is a triangular matrix. This allows us to rewrite the regression model in matrix notation as follows:

$$\begin{aligned} y &= X\beta + E \\ S^{-1}y &= S^{-1}X\beta + S^{-1}E \\ y' &= X'\beta + E' \end{aligned}$$

This transformation is successful, because in the prime model, we have uncorrelated errors again:

$$Var(E) = Var(S^{-1}E) = S^{-1}Var(E)S^{-T} = S^{-1}\sigma^2 SS^T S^{-T} = \sigma^2 I$$

With some algebra, it is easy to show that the estimated regression coefficients for the generalized least squares approach turn out to be:

$$\hat{\beta} = (X^T \Sigma^{-1} X) X^T \Sigma^{-1} y$$

This is what is known as the *generalized least squares estimator*. Moreover, the covariance matrix of the coefficient vector is:

$$Var(\hat{\beta}) = (X^T \Sigma^{-1} X)^{-1} \sigma^2$$

This covariance matrix then also contains standard errors in which the correlation of the errors has been accounted for, and with which sound inference is possible. However, while this all neatly lines up, we of course require knowledge about the error covariance matrix Σ, which is generally unknown in practice. What we can do is estimate it from the data, for which two approaches exist.

Cochrane-Orcutt Method

As we have seen above, this method is iterative: it starts with an ordinary least squares (OLS) regression, from which the residuals are determined. For these residuals, we can then fit an appropriate $ARMA(p,q)$ model and with its estimated model coefficients $\alpha_1,...,\alpha_p$ and $\beta_1^{MA(q)},...,\beta_q^{MA(q)}$. On the basis of the estimated $AR(MA)$ model coefficients, an estimate of the error covariance matrix Σ can be derived. We denote it by $\hat{\Sigma}$, and plug it into the formulae presented above. This yields adjusted regression coefficients and correct standard errors for these regression problems. As mentioned above, the iterative approach is secondary to a simultaneous MLE. Thus, we do without further performing Cochrane-Orcutt on our examples.

The `gls()` Procedure

A better, yet more sophisticated approach is to estimate the regression coefficients and the $ARMA$ parameters simultaneously. This can be done using the Maximum-Likelihood principle. Even under the assumption of Gaussian errors, this is a nonlinear and numerically difficult problem. However, for practical application, we do not need to worry. The **R** package `nlme` features the `gls()` procedure which tackles this problem. Thus, we focus on correct application of the **R** function.

Example 1: Global Temperature

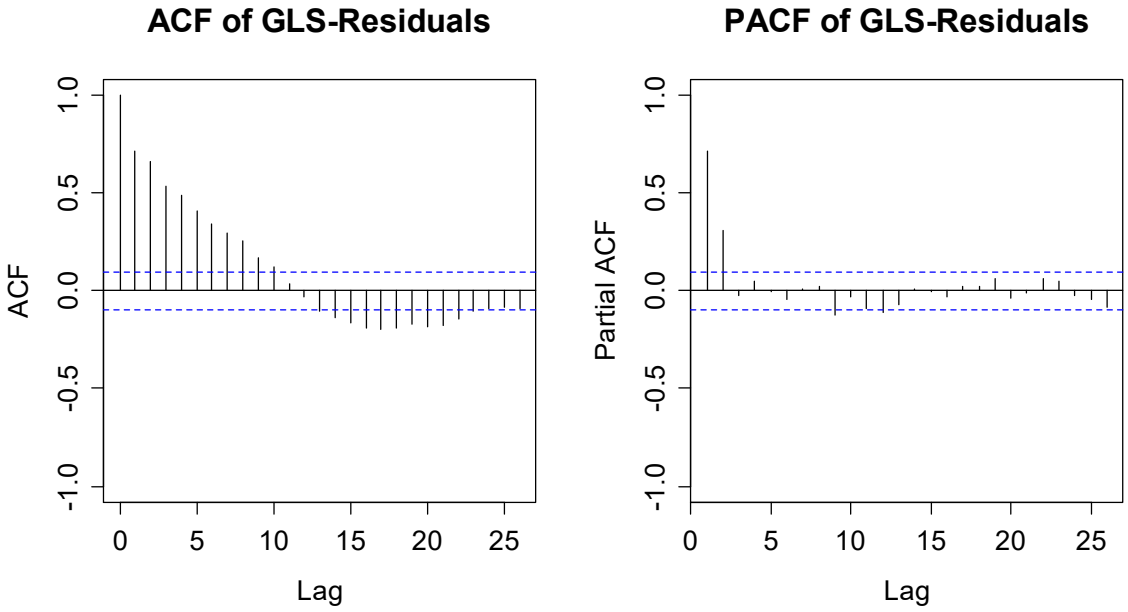
Every GLS regression analysis starts by fitting an OLS an analyzing the residuals, as we have done above. Remember that the only model violation we found were correlated residuals that were well described with an $AR(2)$ model. Please note that for performing GLS, we need to provide a dependency structure for the errors. Here, this is the $AR(2)$ model, in general, it is an appropriate $ARMA(p,q)$. The syntax and output is as follows:

```
> library(nlme)
> corStruct <- corARMA(form=~time, p=2)
> fit.gls <- gls(temp~time+season, data=dat, corr=corStruct)
> fit.gls
Generalized least squares fit by REML
  Model: temp ~ time + season
  Data: dat
  Log-restricted-likelihood: 366.3946

Coefficients:
  (Intercept)          time     seasonFeb     seasonMar
-39.896981987   0.020175528   0.008313205  -0.006487876
    seasonApr     seasonMay     seasonJun     seasonJul
 -0.009403242  -0.027232895  -0.017405404  -0.015977913
    seasonAug     seasonSep     seasonOct     seasonNov
 -0.011664708  -0.024637218  -0.036152584  -0.048582236
    seasonDec
 -0.025326174

Correlation Structure: ARMA(2,0)
 Formula: ~time
 Parameter estimate(s):
      Phi1       Phi2
 0.5539900 -0.1508046
Degrees of freedom: 420 total; 407 residual
Residual standard error: 0.09257678
```

The result reports the regression and the AR coefficients. Using the `summary()` function, even more output with all the standard errors can be generated. We omit this here and instead focus on the necessary residual analysis for the GLS model. We can extract the residuals using the usual `resid()` command. Important: these

residuals must not look like White Noise, but as from an $ARMA(p,q)$ with orders p and q as provided in the `corStruct` object – which in our case, is an $AR(2)$.

```
> par(mfrow=c(1,2))
> acf(resid(fit.gls), main="ACF of GLS-Residuals")
> pacf(resid(fit.gls), main="PACF of GLS-Residuals")
```



The plots look similar to the ACF/PACF plots of the OLS residuals. This is often the case in practice, only for more complex situations, there can be a bigger discrepancy. And because we observe an exponential decay in the ACF, and a clear cut-off at lag 2 in the PACF, we conjecture that the GLS residuals meet the properties of the correlation structure we hypothesized, i.e. of an $AR(2)$ model. Thus, we can now use the GLS fit for drawing inference. We first compare the OLS and GLS point estimate for the trend, along with its confidence interval:

```
> coef(fit.lm)["time"]
      time
0.01822374
> confint(fit.lm, "time")
          2.5 %     97.5 %
time 0.01702668 0.0194208
> coef(fit.gls)["time"]
      time
0.02017553
> confint(fit.gls, "time")
          2.5 %      97.5 %
time 0.01562994 0.02472112
```

We obtain a temperature increase of 0.0182 degrees per year with the OLS, and of 0.0202 with the GLS. While this may seem academic, the difference among the point estimates is around 10%, and theory tells us that the GLS result is more reliable. Moreover, the length of the confidence interval is 0.0024 with the OLS, and 0.0091

and thus 3.5 times as large with the GLS. Thus, without accounting for the dependency among the errors, the precision of the trend estimate is by far overestimated. Nevertheless, also the confidence interval obtained from GLS regression does not contain the value 0, and thus, the null hypothesis on no global warming trend is rejected (with margin!).

Finally, we investigate the significance of the seasonal effect. Because this is a factor variable, i.e. a set of dummy variables, we cannot just produce a confidence interval. Instead, we have to rely on a significance test, i.e. a partial F-test. Again, we compare what is obtained from OLS and GLS:

```
> drop1(fit.lm, test="F")
Single term deletions
Model: temp ~ time + season

       Df Sum of Sq     RSS     AIC  F value  Pr(F)
<none>               6.4654 -1727.0
time    1   14.2274 20.6928 -1240.4 895.6210 <2e-16 ***
season 11    0.1744  6.6398 -1737.8   0.9982 0.4472

> anova(fit.gls)
Denom. DF: 407
            numDF  F-value p-value
(Intercept)     1 78.40801  <.0001
time            1 76.48005  <.0001
season         11  0.64371  0.7912
```

As for the trend, the result is identical with OLS and GLS. The seasonal effect is non-significant with p-values of 0.45 (OLS) and 0.79 (GLS). Again, the latter is the value we must believe in. We conclude that there is no seasonality in global warming – but there is a trend. Thus, the seasonality should be omitted from the model and the computations need to be repeated (not shown here).

Example 2: Air Pollution

For finishing the air pollution example, we also perform a GLS fit here. We identified an $AR(1)$ as the correct dependency structure for the errors. Thus, we define it accordingly:

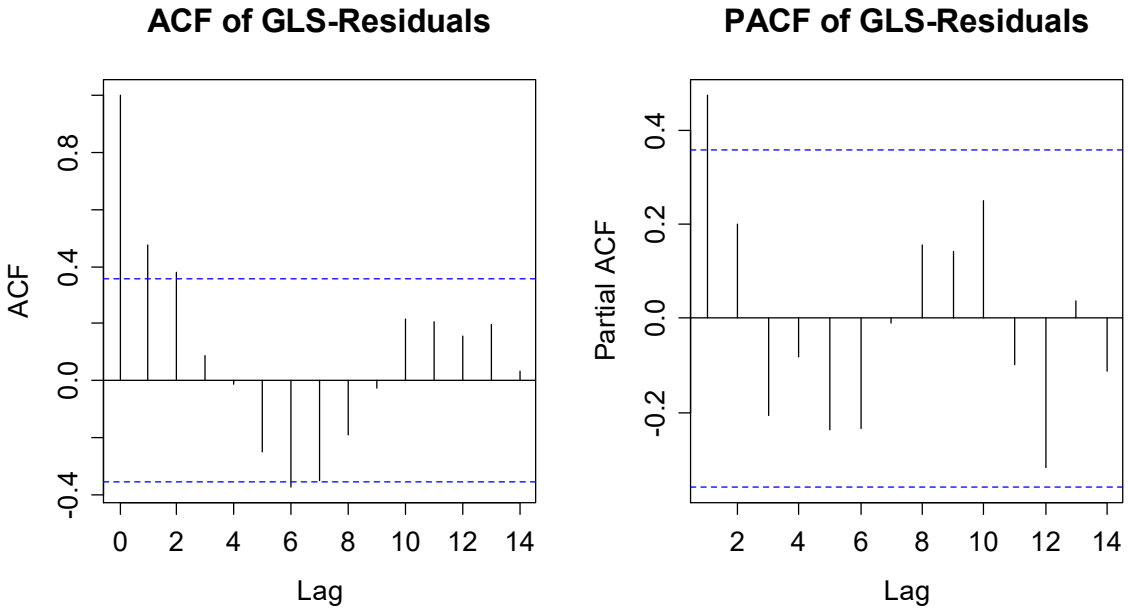
```
> dat       <- cbind(dat, time=1:30)
> corStruct <- corARMA(form=~time, p=1)
> model     <- formula(Oxidant ~ Wind + Temp)
> fit.gls   <- gls(model, data=dat, correlation=corStruct)
```

The output then is as follows:

```
> fit.gls
Generalized least squares fit by REML
  Model: model
  Data: dat
  Log-restricted-likelihood: -72.00127
```

```
Coefficients:
(Intercept)        Wind        Temp
 -3.7007024  -0.4074519   0.4901431

Correlation Structure: AR(1)
 Formula: ~time
 Parameter estimate(s):
      Phi
0.5267549
Degrees of freedom: 30 total; 27 residual
Residual standard error: 3.066183
```

Again, we have to check if the GLS residuals show the stylized facts of an $AR(1)$:



This is the case, and thus we can draw inference from the GLS results. The confidence intervals of the regression coefficients are:

```
> confint(fit.lm, c("Wind", "Temp"))
          2.5 %     97.5 %
Wind -0.6044311 -0.2496841
Temp  0.2984794  0.7422260

> confint(fit.gls, c("Wind", "Temp"))
          2.5 %     97.5 %
Wind -0.5447329 -0.2701709
Temp  0.2420436  0.7382426
```

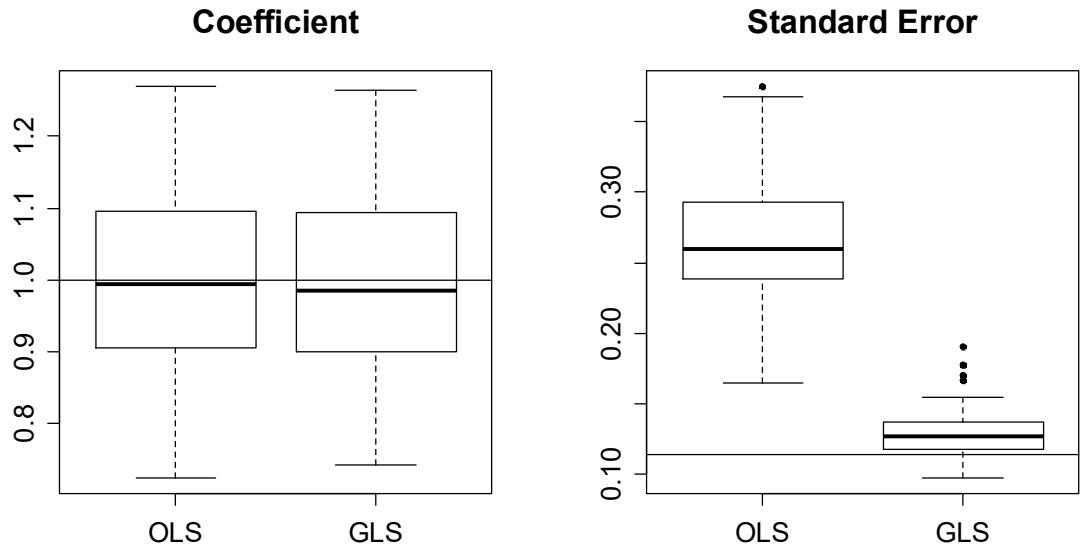
Here the differences among point estimates and confidence intervals are not very pronounced. This has to do with the fact that the correlation among the errors is weaker than in the global temperature example. But we emphasize again that the GLS results are the one to be relied on and the magnitude of the difference between OLS and GLS can be tremendous.

Simulation Study

We provide further evidence for the importance of the GLS approach by performing a simulation study in which the resulting coefficients and standard errors are compared to the ones obtained from OLS regression. We consider the following, relatively simple model:

$$\begin{aligned} x_t &= t/50 \\ y_t &= x_t + 2(x_t)^2 + E_t \end{aligned}$$

where E_t is from an $AR(1)$ process with $\alpha_1 = -0.65$. The innovations are Gaussian with $\sigma = 0.1$. Regression coefficients and the true standard deviations of the estimators are known in this extraordinary situation. However, we generate 100 realizations with series length $n = 50$, on each perform OLS and GLS regression and record both point estimate and standard error.



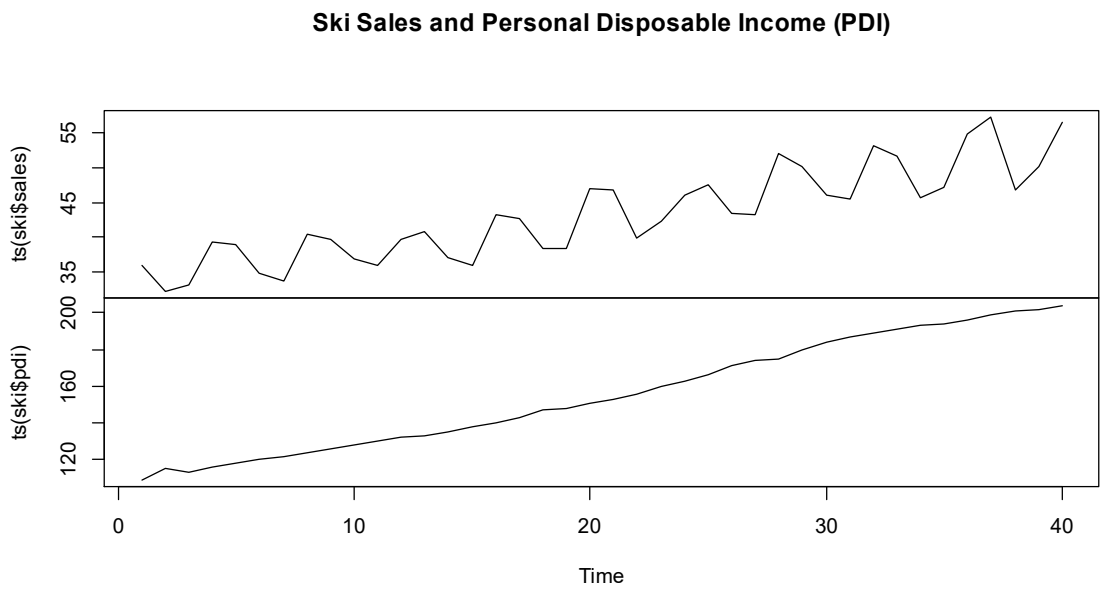
The simulation outcome is displayed by the boxplots in the figure above. While the point estimator for β_1 in the left panel is unbiased for both OLS and GLS, we observe that the standard error for $\hat{\beta}_1$ is very poor when the error correlation is not accounted for. We emphasize again that OLS regression with time series will inevitably lead to spuriously significant predictors and thus, false conclusions. Hence, it is absolutely key to inspect for possible autocorrelation in the regression residuals and apply the `gls()` procedure if necessary.

However, while `gls()` can cure the problem of autocorrelation in the error term, it does not solve the issue from the root. Sometimes, even this is possible. In the next subsection, we conclude the chapter about time series regression by showing how correlated errors can originate, and what practice has to offer for deeper understanding of the problem.

7.5 Missing Predictor Variables

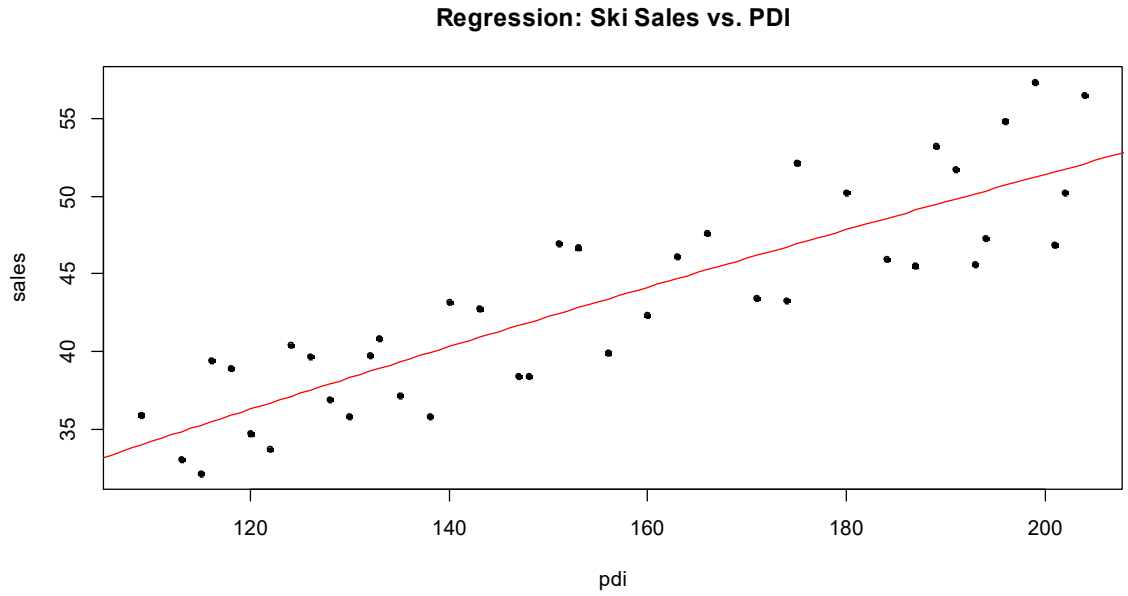
The presence correlated errors is often due to missing predictors. For illustration, we consider a straightforward example of a ski selling company in the US. The quarterly sales Y_t are regressed on the personal disposable income (PDI) which is the one and only predictor x_t. We start out with loading the data (which are available from the lecturer upon request) and presenting a time series plot.

```
> ## Loading the data
> load("ski2.rda")
> ski.ts <- ts.union(ts(ski$sales), ts(ski$pdi))
> plot(ski.ts, main="Ski Sales and Personal Disposable …")
```



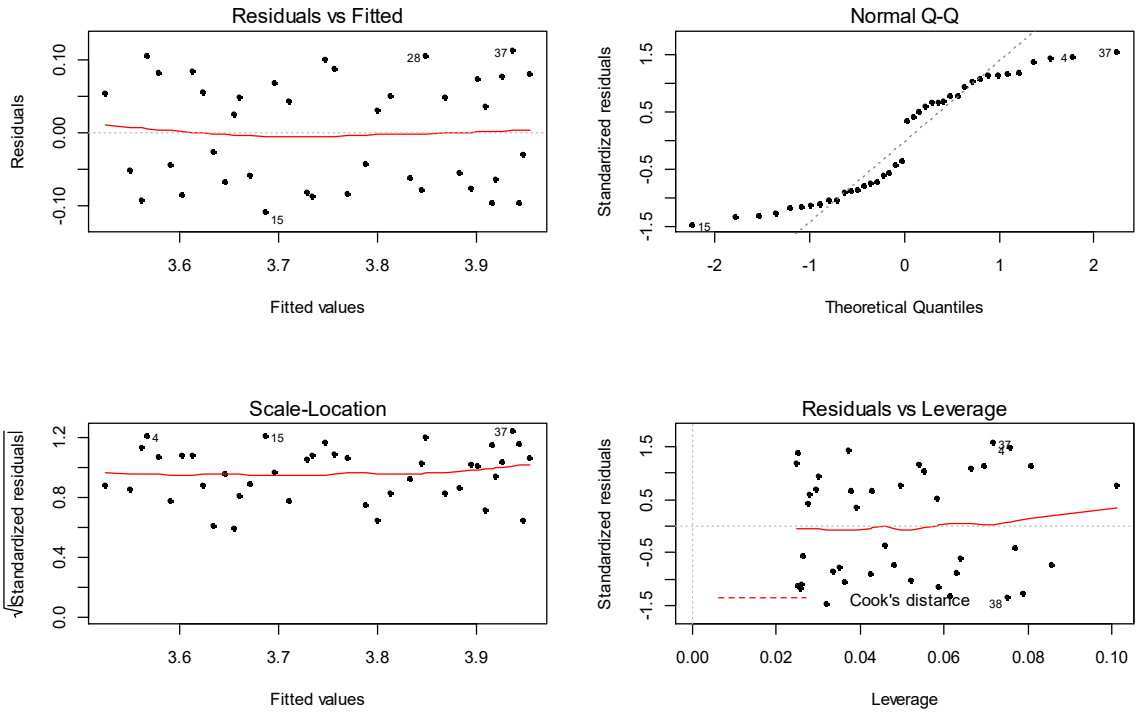
Next, we treat the two series in a regression problem where the sales are the response variable and PDI is the predictor. As both variables are on a relative scale, a log-transformation is indicated. This means that we fit a linear model of type $\log(sales) \sim \log(PDI)$. This corresponds to a power law with character $sales \sim \beta_0 \cdot PDI^{\beta_1}$ on the original scale. A scatterplot with the fit obtained from applying OLS on the transformed variables is shown below.

```
> ## Scatterplot
> par(mfrow=c(1,1))
> plot(sales ~ pdi, data=ski, pch=20, main="Regression: …")
>
> ## OLS model for the transformed variables
> fit <- lm(log(sales) ~ log(pdi), data=ski)
>
> Plotting the OLS fit on the original scale
> newpdi <- 100:220; newdf <- data.frame(pdi=newpdi)
> lines(newpdi, exp(predict(fit, newdf)), col="red")
```



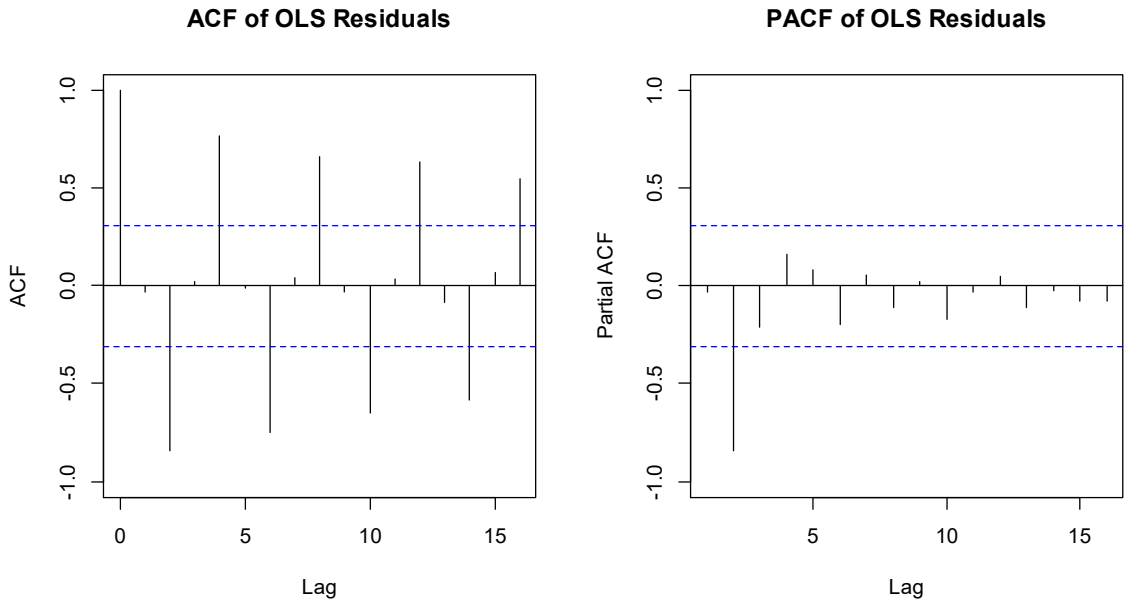
The coefficient of determination is rather large, i.e. $R^2 = 0.766$ and the fit seems adequate, i.e. the power law seems to correctly describe the systematic relation between sales and PDI. However, the model diagnostic plots (see the next page) show some rather special behavior, i.e. there are hardly any "small" residuals (in absolute value). Or more precisely, the data points almost lie on two lines around the regression line, with almost no points near or on the line itself.

```
> ## Residual diagnostics
> par(mfrow=c(2,2))
> plot(fit, pch=20)
```

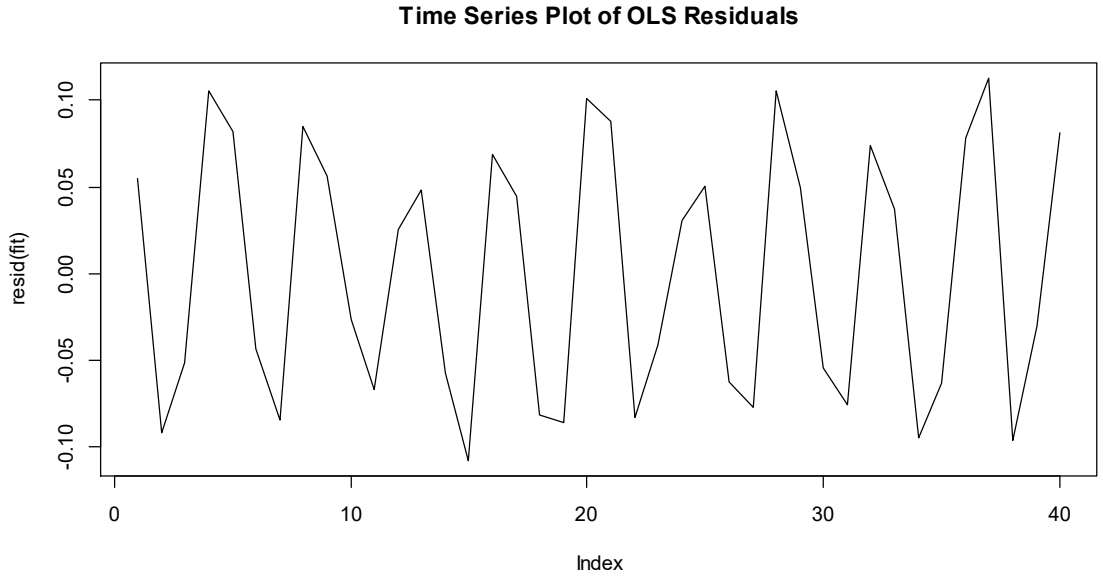


As the next step, we analyze the correlation of the residuals and perform a Durbin-Watson test. The result is as follows:

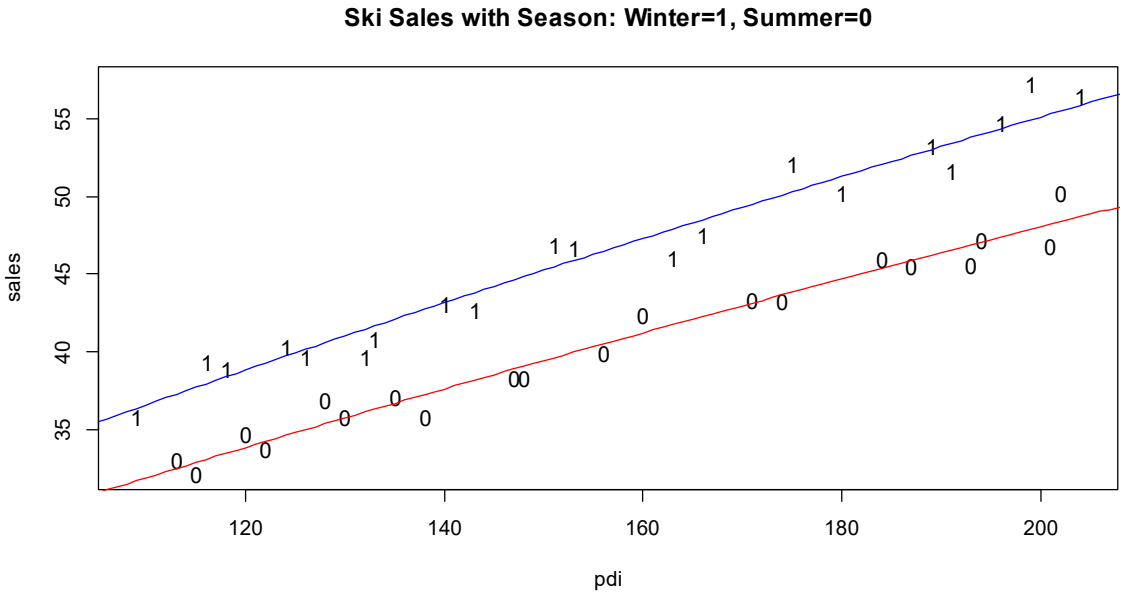
```
> dwtest(fit)
data:  fit
DW = 2.0224, p-value = 0.4609
alt. hypothesis: true autocorrelation is greater than 0
```



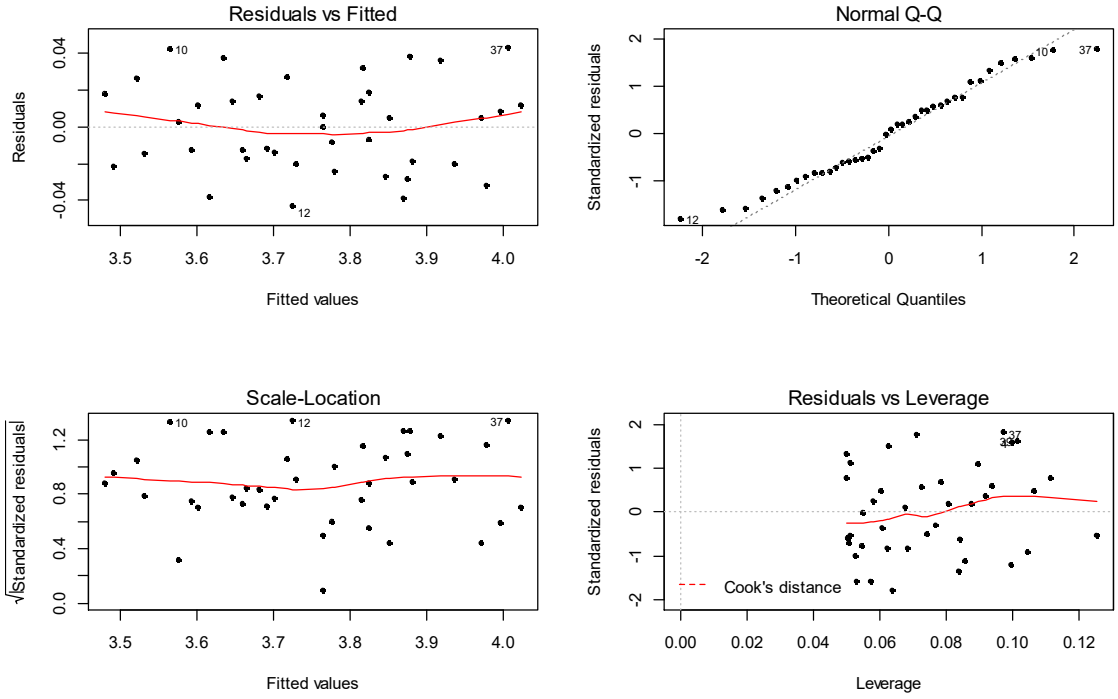
While the Durbin-Watson test does not reject the null hypothesis, the residuals seem very strongly correlated. The ACF exhibits some decay that may qualify as exponential, and the PACF has a clear cut-off at lag 2. Thus, an $AR(2)$ model could be appropriate. And because it is an $AR(2)$ where α_1 and $\rho(1)$ are very small, the Durbin-Watson test fails to detect the dependence in the residuals. The time series plot is as follows:



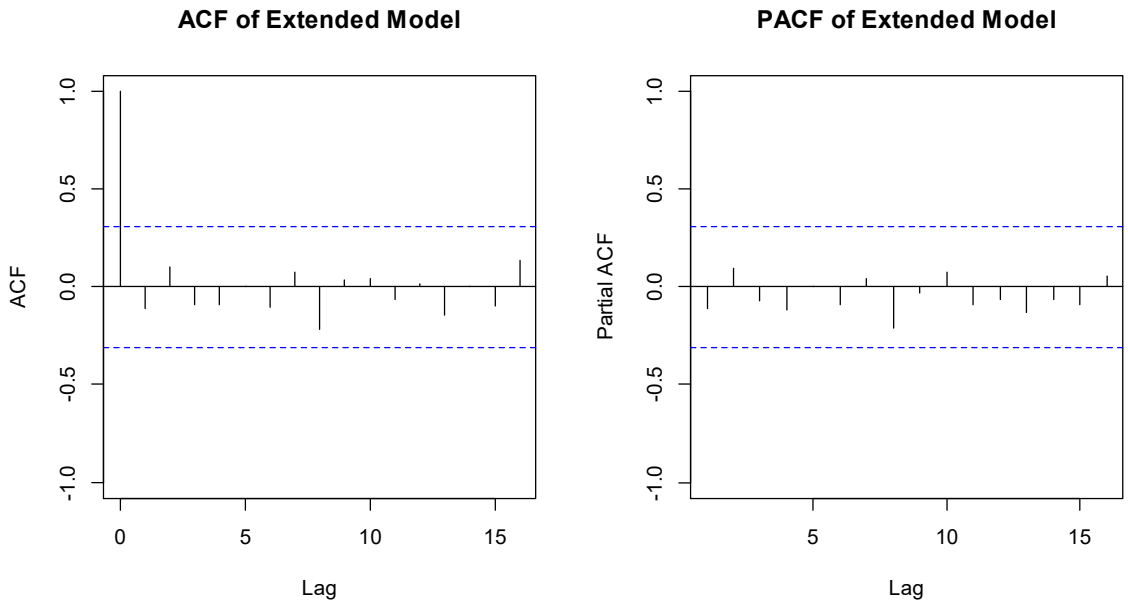
While we could now account for the error correlation with a GLS, it is always better to identify the reason behind the dependence. I admit this is suggestive here, but as mentioned in the introduction of this example, these are quarterly data and we might have forgotten to include the seasonality. It is not surprising that ski sales are much higher in fall and winter and thus, we introduce a factor variable which takes the value 0 in spring and summer, and 1 else.



Introducing the seasonal factor variable accounts to fitting two separate power laws for summer and winter. Eyeballing already lets us assume that the fit is good. This is confirmed when we visualize the diagnostic plots:



The unwanted structure is now gone, as is the correlation among the errors:



Apparently, the addition of the season as an additional predictor has removed the dependence in the errors. Rather than using GLS, a sophisticated estimation procedure, we have found a simple model extension that describes the data well and is certainly easier to interpret (especially when it comes to prediction) than a model that is built on correlated errors.

We conclude by saying that using GLS for modeling dependent errors should only take place if care has been taken that no important and/or obvious predictors are missing in the model.

8 Forecasting

One of the major motivations for and principal goals with time series analysis is to produce predictions which show the future evolution of the data, i.e. time series forecasting. It is important to recognize that this is an *extrapolation* in the time domain. In all other areas of statistical data analysis, applying models beyond the range of observed training data is applied with great care only, as it is prone to false conclusions. Of course, this is no different with time series forecasting, although it is often neglected or presented with naïve optimism.

The task we are faced with in time series forecasting can be compared to driving a car by looking through the rear window mirror. While this may work well on a wide motorway that runs mostly straight on and has a few gentle bends only, things get more complicated as soon as we are on a narrow mountain road with sharp and unexpected bends. Then, we would need to drive very slowly to stay on track. This all translates directly to time series analysis. For series where the signal is much stronger than the noise, accurate forecasting is possible. However, for noisy series, there is a great deal of uncertainty in the predictions, and they are at best reliable for a very short horizon. From this, one might conclude that the principal source of uncertainty is inherent in the process, i.e. comes from the stochastic and unpredictable innovation terms. However, in practice, there are several other factors that can threaten the reliability of any forecasting procedure. In particular:

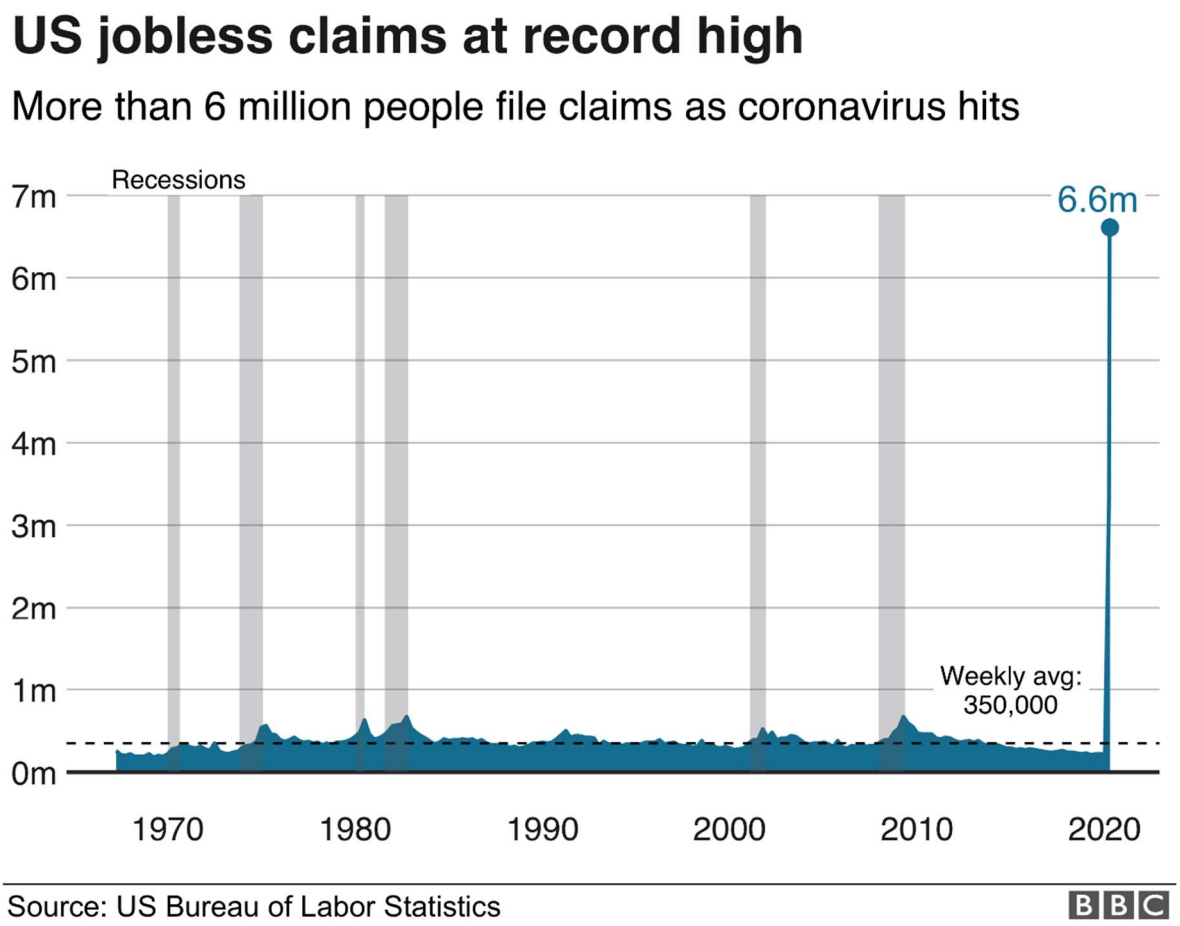
- We need to be certain that the data generating process does not show a disruption at some point in time, i.e. continues in the future as it was observed in the past. Let's e.g. consider the case when we drive on a motorway by looking through the rear window mirror, but it (unexpectedly) ends and suddenly turns into a mountain road, a clear recipe for disaster.

- When we choose/fit a model based on a realization of data, we have no guarantee that it is the correct, i.e. data-generating one. Translated to our car-driving example we never truly told on what kind of road we are driving, i.e. we can only guess if it's a motorway or a mountain road. In real-world time series practice, we are never guaranteed that the true data generating process is e.g. $ARMA(p,q)$ and even if it is, we may commit mistakes in choosing the orders p,q.

- Even if we are so lucky to identify the correct data-generating process (or in cases we "know" it, e.g. in a simulation), there is additional uncertainty arising from the estimation of the parameters.

It is also important to understand what a time series forecast delivers. In uninformed public opinion, it is often perceived as "the future evolution of the series". But this is mathematically wrong, as a time series forecast only yields (ane estimate of) the conditional mean of the future instances. Often, the influence of the unpredictable innovation terms is huge and adds a major source of variation. Hence, it is absolutely

essential to complement conditional mean forecasts with prediction intervals, i.e. a measure of uncertainty for the future observed values. This also creates a novel view on the task of time series forecasting: we cannot say what exactly will happen in the future. But we can hope for providing a realistic view on what we can expect both in terms of mean and variation around it, i.e. to what extent a time series is predictable at all. It often helps tremendously to accept this perspective and communicate it offensively. Before providing an outlook over the contents of this forecasting chapter, we here display the weekly number of jobless claims in the US from the late 1960's until March 2020.



The raw data are available from the US Bureau of Labor Statistics, the visual display was taken from https://www.bbc.com/news/business-52231929, accessed on April 28, 2020. The weekly average of jobless claims hovered around an average of 350k, sometimes increasing to twice that value in recession periods. But then, due to the SARS-COV2 lockdown and crisis in March 2020, that figure soared to 6.6m in a week. This is a major disruption – could anyone have foreseen and forecasted this value with a few months lead?!? Obviously, this is a completely unrealistic expectation and even if an interval forecast (as we advocate, rather than giving just the point forecast) would have been provided, it would have been broken by orders of magnitude. In summary, methods for time series forecasting are not the "magical crystal ball" that we may hope for, they will never be able to forecast outliers and groundbreaking events. Still, they can be useful and have their place in many applied fields.

Keeping these caveats in mind, we will now present several approaches to time series forecasting. First, we deal with stationary processes and present, how AR, MA and ARMA processes can be forecasted. These principles can be extended to the case of ARIMA and SARIMA models, such that forecasting series with either trend and/or seasonality is also possible. As we had seen in section 4.3, the decomposition approach for non-stationary time series helps a great deal for visualization and modelling. Thus, we will present some heuristics about how to produce forecasts with series that were decomposed into trend, seasonal pattern and a stationary remainder. Last but not least, we present the method of exponential smoothing. This was constructed as a model-free, intuitive weighting scheme that allows forecasting of time series. Due to its simplicity and the convenient implementation in the `HoltWinters()` and other procedures in **R**, it is very popular and often used in applied sciences.

8.1 Stationary Time Series

We assume a stationary time series, for which an appropriate $AR(p)$, $MA(q)$ or $ARMA(p,q)$ model was identified, the parameters were successfully estimated and where the residuals exhibited the required properties, i.e. looked like White Noise. Under these circumstances, forecasts may be readily computed. Given data up to time n, the forecasts will either involve the past observations, and/or the unobservable past innovation terms that are in practice replaced with residuals.

In mathematical statistics, many forecasting methods have been studied on a theoretical basis with the result that the minimum mean squared error forecast $\hat{X}_{n+k,1:n}$ for k steps ahead is given by the conditional expectation, i.e.:

$$\hat{X}_{n+k,1:n} = E[X_{n+k} \mid X_1,...,X_n]$$

In evaluating this term, we use the fact that the best forecast of all future innovation terms $E_t, t > n$ is simply zero. We will be more specific in the following subsections. Besides providing a point forecast with the conditional expectation, it is in practice equally important to produce an interval forecast that makes a statement about its precision.

8.1.1 Forecasting AR(1)

For simplicity, we first consider a mean-zero, stationary $AR(1)$ process:

$$X_t = \alpha_1 X_{t-1} + E_t ,$$

E_t is the innovation, for which we do not need to assume a particular distribution. As we will see below, it is convenient to assume Gaussian E_t, because this allows for an easy derivation of a prediction interval. The conditional expectation at time $n+1$ is given by:

$$E[X_{n+1} \mid X_1,...,X_n] = \alpha_1 x_n .$$

Thus, we can easily forecast the next instance of a time series with the observed value of the previous one, as long as it is available. In particular:

$$\hat{X}_{n+1,1:n} = \alpha_1 x_n .$$

For the k-step forecast with $k > 1$, we need to repeatedly plug-in the model equation, and use the fact that $E[E_{n+k} \mid X_1,...,X_n] = 0$ for all $k > 0$.

$$\begin{aligned} \hat{X}_{n+k,1:n} &= E[X_{n+k} \mid X_1,...,X_n] \\ &= E[\alpha_1 X_{n+k-1} + E_{n+k} \mid X_1,...,X_n] \\ &= \alpha_1 E[X_{n+k-1} \mid X_1,...,X_n] \\ &= ... \\ &= \alpha_1^k x_n \end{aligned}$$

Apparently, for any stationary $AR(1)$ process, the k-step forecast beyond the end of a series depends on the last observation x_n only and goes to zero exponentially quickly. Note that the value of zero also corresponds to the unconditional, global mean of the process. For practical implementation with real data, we would just plug-in the estimated model parameter $\hat{\alpha}_1$ and can so produce a forecast for arbitrary horizon. In case of a shifted $AR(1)$ with non-zero mean, m is subtracted first so that the forecast can be obtained on the pure process in the above manner, before m is finally added again. As always, a prediction is much more useful in practice if one knows how precise it is and for what kind of variability in the values that materialize we have to prepare. Under the assumption of Gaussian innovations, a 95% prediction interval can be derived from the conditional variance $Var(X_{n+k} \mid X_1,...,X_n)$. For the special case of $k = 1$ we obtain:

$$\alpha_1 x_n \pm 1.96 \cdot \sigma_E ,$$

where 1.96 is the 97.5% quantile of the standard Normal distribution, which also dictates how intervals with different level can be obtained. Again, for practical implementation of the interval, we need to plug-in $\hat{\alpha}_1$ and $\hat{\sigma}_E$. However, the formula does not account for the uncertainty that arises from plugging-in these estimates, so the coverage of the interval will in practice be smaller than 95%. By how much this is the case largely depends on the quality of the estimates, i.e. the series length n. For a k-step forecast, the theoretical 95% prediction interval is:

$$\alpha_1^k x_n \pm 1.96 \cdot \left(1 + \sum_{j=1}^{k-1} \alpha_1^{2j}\right) \cdot \sigma_E .$$

For increasing prediction horizon k, the conditional variance goes to $\sigma_E^2 / (1 - \alpha_1^2)$, which is the process variance σ_X^2. Thus, for the 1-step forecast, the uncertainty in the prediction is given by the innovation variance σ_E alone, while for increasing horizon k the prognosis interval gets wider is finally determined by the unconditional process variance.

Simulation Study

As we have argued above, the 95% prediction interval does not account for the uncertainty in the parameter estimates, the choice of the model or the continuity of the data generating process. We run a small simulation study for pointing out the effect of plugging-in the parameter estimates. It consists of generating a length $(n+1)$ realization from an $AR(1)$ process with $\alpha_1 = 0.5$ and Gaussian innovations where $\sigma_E = 1$. From the first n data points, an $AR(1)$ model, respectively the parameters $\hat{\alpha}_1, \hat{\sigma}_E$, are estimated by MLE and the point forecast along with the prediction interval is determined. Finally, it was checked whether the next instance of the time series fell within the interval, from which an empirical coverage level could be determined. The values were:

$n = 20$	$n = 50$	$n = 100$	$n = 200$
91.01%	93.18%	94.48%	94.73%

As we notice, the coverage is clearly too small in case of $n = 20$. However, already for a series with length $n = 100$, it reaches a reasonable level. Please note the undercoverage here arises simply from parameter estimation in a benign setting. In real-world examples, the undercoverage is often worse since additional uncertainty arises from model misspecification or potential disruption in the data-generating process.

Practical Example

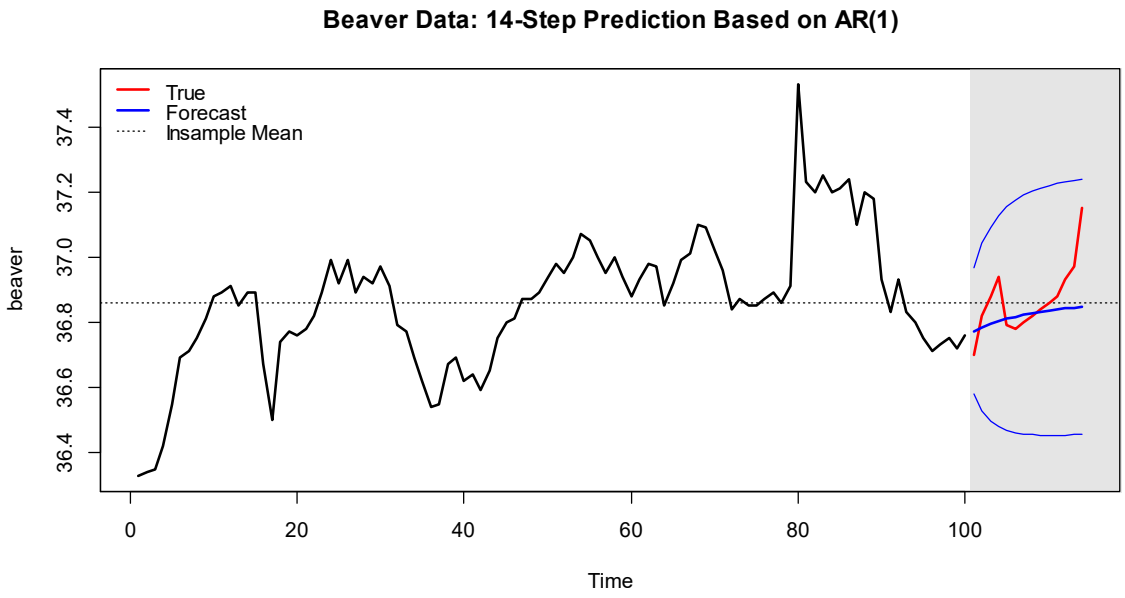
We now apply the **R** functions that implement the above theory on the Beaver data from section 4.4.3. An $AR(1)$ seems appropriate for this series. In order to compare the forecast with true values that materialized, we retain the last 14 observations of the series from the fitting process. These will then be predicted, and the true values will be used for verifying the prediction. The **R** commands for fitting the model on the training data and producing the 14-step prediction are simple and straightforward.

```
> btrain   <- window(beaver, 1, 100)
> btest  <- window(beaver, 101, 114)
> fit      <- ar.burg(btrain, order=1)
> forecast <- predict(fit, n.ahead=14)
```

The `forecast` object is a list that has two components, `pred` and `se`, which contain the point predictions and the predictions' standard errors, respectively. We now turn our attention to how the forecast can be visualized:

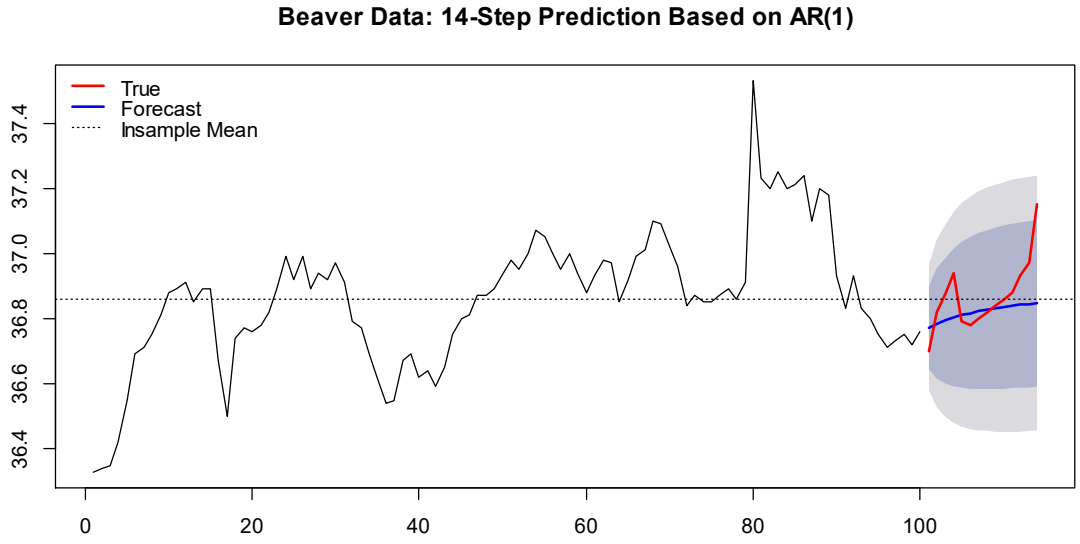
```
> plot(beaver, col="blue", lwd=2, type="n")
> rect(100.5, 35, 120 ,40, col="grey90", border=NA)
> lines(btrain, lwd=2)
> lines(btest, lwd=2, col="red")
> lines(pred$pred, lwd=2, col="blue")
```

```
> lines(pred$pred+pred$se*1.96, col="blue")
> lines(pred$pred-pred$se*1.96, col="blue")
> abline(h=mean(btrain), lty=3)
> box()
```



A simpler alternative to the self-construction of the above plot lies in relying on the `plot.forecast()` function. This only requires the following code:

```
> plot(forecast(fit, h=14), main="Beaver Data: …")
> lines(btest, lwd=2, col="red")
> abline(h=mean(btrain), lty=3)
```



The shaded regions are 80% (darker/smaller) and 95% prediction intervals.

One more issue requires some attention here: for the Beaver data, a pure $AR(1)$ process is not appropriate, because the global series mean is clearly different from zero. The way out is to de-mean the series, then fit the model and produce forecasts, and finally re-adding the global mean. **R** does all this automatically. We conclude by summarizing what we observe in the example: the forecast is based on the last observed value $x_{100} = 36.76$, and from there approaches the global series mean $\hat{\mu} = 36.86$ exponentially quickly. Because the estimated coefficient is $\hat{\alpha}_1 = 0.87$, and thus relatively close to one, the convergence to the global mean takes some time. On the other hand, from a practical viewpoint the forecast seems rather dull, as it does not track the future evolution of the series. But again, we have to be aware of the situation in these data: our forecast shows the conditional mean, whereas the observed values are to a large extent driven by the unpredictable innovation terms. The point forecast quickly converging point forecast along with a large prediction interval clearly explains that one has to be prepared to major fluctuations in the beaver body temperature. This is the message that a time series forecast can deliver – expecting more is unrealistic.

Measuring Forecasting Error

Often one wishes to express the forecasting error for a time series model for understanding the magnitude of the deviations that we need to expect. Moreover, correctly implemented, forecasting errors can also serve for model choice. For truly evaluating the performance of a model, it is important to study the out-of-sample performance. This means that as above, the last part of the data need to be withheld from the fitting process. These values can then be forecasted and compared against the observed ones. Please note that we cannot rely on the insample resp. training error for such considerations. A good or even perfect training data fit can be achieved by overparametrizing a model, but this does not imply good forecasting performance, yet in fact overfitting is usually detrimental to the out-of-sample results.

For a correct assessment of the accuracy, a suitable measure has to be found. The choice depends on whether absolute or relative errors are considered. For time series that did not require a transformation, one usually relies on the following absolute error measures:

$$MAE = \frac{1}{h} \cdot \sum_{t=n+1}^{n+h} \left| x_t - \hat{x}_t \right| = mean(\left| e_t \right|)$$

$$RMSE = \sqrt{\frac{1}{h} \cdot \sum_{t=n+1}^{n+h} (x_t - \hat{x}_t)} = \sqrt{mean(e_t^2)}$$

In the above formulae, h is the forecasting horizon, $\hat{x}_t$ the forecasted value for time t and e_t the difference between observation and forecast. Both error measures tell us how "big the difference between the observed and forecasted value on average is". Fundamentally, the $RMSE$ is usually better suited, as all unbiased prediction methods aim for minimizing $RMSE$ rather than MAE.

We compute the forecasting errors for the Beaver test data:

```
> mae <- mean(abs(btest-pred$pred)); mae
[1] 0.07202408
> rmse <- sqrt(mean((btest-pred$pred)^2)); rmse
[1] 0.1044069
```

They tell us that "on average", we miss the correct temperature by about 0.1 degrees Celsius. The $RMSE$ takes the larger value, because big deviations (as they exist in the last observations) count more. An alternative to the self-coded error computation lies in using the `accuracy()` function from `library(forecast)`. It outputs a wealth of error measures and it is important to understand their meaning and suitability. The numerical results for MAE and $RMSE$ are identical to above. Some of the other error measures are explained in later sections where they are appropriate for the respective examples, details about the further ones can be accessed in Hyndmans "Forecasting: principles and practice", section 3.4, accessible at: https://otexts.com/fpp2/accuracy.html.

```
> round(accuracy(forecast(fit, h=14), btest),3)
               ME  RMSE   MAE   MPE  MAPE  MASE   ACF1 Theils U
Training  0.004 0.096 0.062 0.012 0.168 0.939 -0.068        NA
Test set  0.049 0.104 0.072 0.132 0.195 1.092  0.337     1.333
```

8.1.2 Forecasting AR(p)

Forecasting from $AR(p)$ processes works with the same concepts as explained above for $AR(1)$, i.e. we use the conditional expectation as a basis. The algebra for writing the forecasting formulae is somewhat more laborious, but not really more difficult. Thus, we do without displaying it here, and directly present the formula for the 1-step-forecast:

$$\hat{X}_{n+1,1:n} = \alpha_1 x_n + \alpha_2 x_{n-1} + ... + \alpha_p x_{n-p}$$

The question is, what do we do for longer forecasting horizons? There, the forecast is again based on the linear combination of the p past instances. For the ones with an index between 1 and n, the observed value x_t is used. Else, if the index exceeds n, we just plug-in the forecasted values $\hat{x}_{t,1:n}$. Thus, the general formula is:

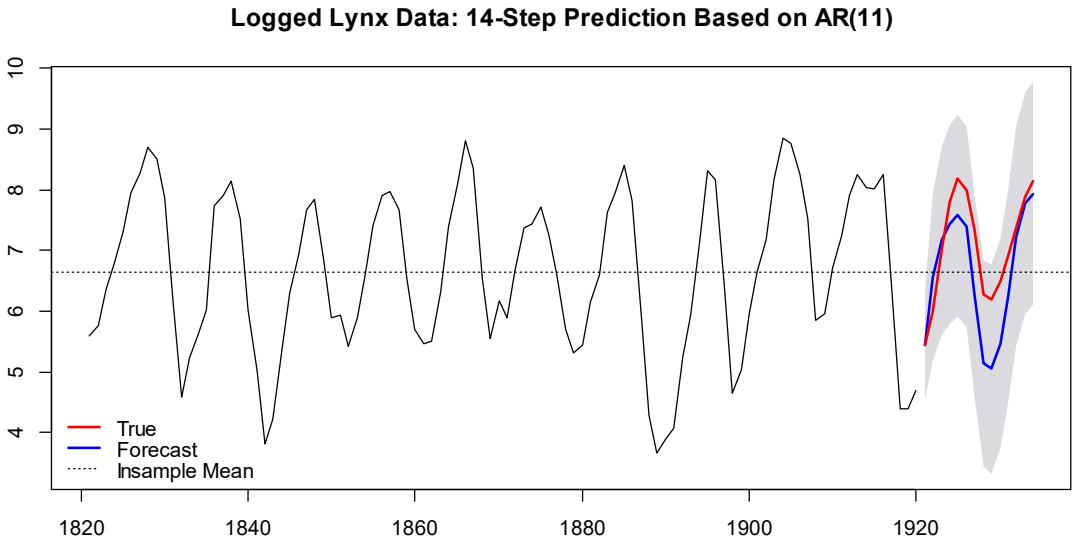
$$\hat{X}_{n+k,1:n} = \alpha_1 \hat{X}_{n+k-1,1:n} + ... + \alpha_p \hat{X}_{n+k-p,1:n} = \psi_1^{(k)} x_n + ... \psi_p^{(k)} x_{n-p},$$

where $\hat{X}_{t,1:n} = x_t$ in all cases where $t \le n$, i.e. an observed value is available. All forecasted values $\hat{X}_{t,1:n} = x_t$ for $\hat{X}_{n+k;1:n}$ for all k will ultimately only depend on $x_{n-p},...,x_{n-1}$, i.e. we have a Markov property for $AR(p)$ forecasts. We can even rewrite an $AR(p)$ forecast as a linear combination of the p last observed instances with some set of coefficients $\psi_1^{(k)},...,\psi_p^{(k)}$ that depend on the forecasting horizon k. It is generally difficult to present formulae for the $\psi_i^{(k)}$ and the iterative plug-in approach from above is more fruitful. In R, we do not need to worry much about these details anyway, as we have access to the `predict()` function.

Practical Example

We consider the logged lynx data for which we had identified an $AR(11)$ as a suitable model. Again, we use the first 100 observations for fitting the model and lay aside the last 14, which are in turn used for verifying the result. We display the result again using the plot.forecast() function where we also add the true evolution of the series. For not cluttering the plot we restrict to displaying the 95% prediction interval.

```
> plot(forecast(fit.ar11, h=14, level=95), main="…")
> lines(test, col="red", lwd=2)
> abline(h=mean(train), lty=3)
```



We observe that the forecast tracks the general behavior of the series pretty well, though the level of the series is underestimated in some years. This is, however, not due to an “error” of ours, it is just that the values were higher than our forecasting model resp. the conditional mean suggested. We also notice that the convergence of the forecast towards the global mean is much slower here than for the Beaver data. This is due to a much stronger signal-to-noise ration in the logged lynx data. However, for longer forecasting horizon k (resp. with increasing h in the R function), the predicted values would also converge to the global mean.

Another absolutely crucial point is that in practice, we won't be interested in the logged number of lynx shot, but our focus lies in the original scale. Hence, we have to back-transform the forecast. The inverse of $\log()$ is $\exp()$, but this basic back-transformation requires prudence. We here reiterate our statement from chapter 4.2.3: By simply using $\exp(\cdot)$, the back-transformed point forecast will not be the mean, but only the median of the forecast distribution. Fundamentally, the median may be a very reasonable summary statistic for a skewed distribution. Nevertheless, there are applications where unbiased predictions are a must, in which case a corrected back-transformation has to be applied. It is given by:

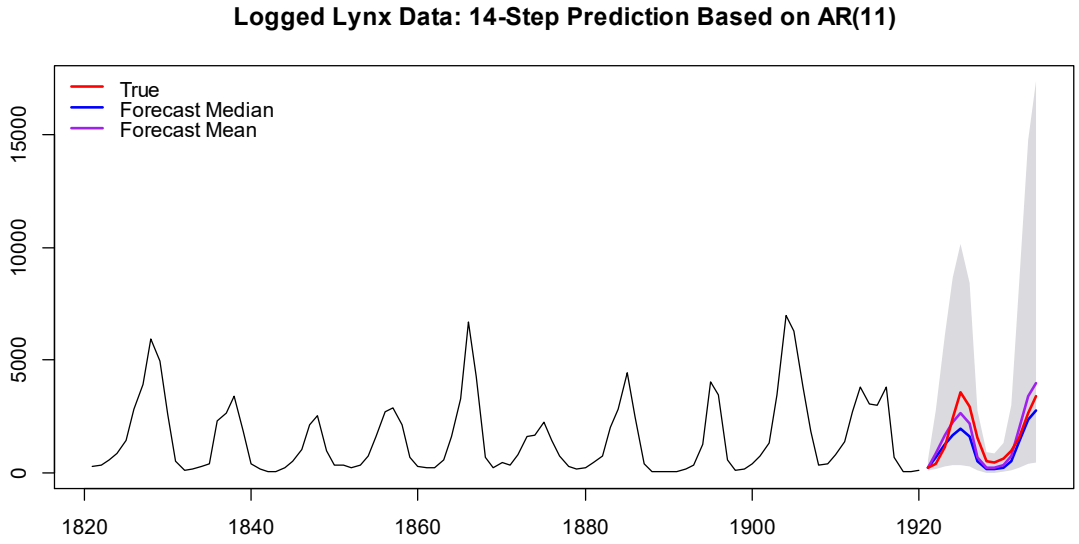
$$\exp(\hat{x}_t)\cdot\left(1+\frac{\hat{\sigma}_h^2}{2}\right), \text{ with } \hat{\sigma}_h^2 = \text{ estimated } h\text{-step forecast variance}$$

The easiest way of obtaining results from this formula is to rely on the `forecast()` function with arguments `lambda=0` for indicating that the model was fitted on log-transformed data and `biasadj=TRUE` for requiring an unbiased forecast i.e.:

```
> forecast(fit.ar11, h=14, lambda=0, biasadj=TRUE)$mean
Time Series:
Start = 1921
End = 1934
Frequency = 1
 [1]  252.4828  871.7136 1703.2313 2270.0263 2640.0801
 [6] 2191.1800  717.0368  232.5009  215.4718  328.9176
[11]  734.6918 1957.1592 3409.2451 4006.8297
```

We continue with plot that shows the lynx data on the original scale (*red*) with both a mean (*purple*) and a median (*blue*) forecast. The mean will of course always be at a higher value than the median. Also, the mean is the average value of all future realizations that we might obtain, whereas for the median, 50% of the realizations will lie above and 50% below.

```
> plot(forecast(fit.ar11, h=14, lambda=0, level=95), main="…")
> lines(forecast(fit.ar11, h=14, lambda=0, biasadj=TRUE)$mean,
        col="purple", lwd=2)
> lines(exp(test), col="red", lwd=2)
```



We notice that especially the interval forecast is getting huge and takes values that were never observed in the past. The point forecast produces peaks similar to the ones observed in the past. However, since it relies on the last 11 observed values only, it won't be able to reproduce the "superhigh" that appears every 40 years.

Usually after forecasting, we want to understand their quality and provide an error measure. As long as we do so on the log-scale, we could again use MAE or $RMSE$ that were presented in the Beaver example. However, there is normally more interest in an error measure for the original scale. Then, due to the fact that errors are relative, we have to use the mean absolut percentage error ($MAPE$).

$$MAPE = \frac{100}{h} \cdot \sum_{t=n+1}^{n+h} \left| \frac{(x_t - \hat{x}_t)}{x_t} \right|$$

Please note that the $MAPE$ is not defined if a time series takes zero values. We compute the error measure for the $AR(11)$ as well as for an $AR(2)$ forecast, as we had considered the latter model in the early stages but rejected it due to the residuals not looking like White Noise.

```
> f11 <- forecast(fit.ar11, h=14, lambda=0, biasadj=TRUE)
> f02 <- forecast(fit.ar02, h=14, lambda=0, biasadj=TRUE)
> mean(100*abs((exp(test)-f11$mean)/exp(test)))
[1] 39.1808
> mean(100*abs((exp(test)-f02$mean)/exp(test)))
[1] 54.98648
```

The output means that with the $AR(11)$, we miss the true number of shot lynx on average by around 39%. The figure for the $AR(2)$ is clearly worse at 55%, indicating that this model performs more poorly. With respect to R, it is also possible to use the `accuracy()` function for computing $MAPE$.

```
> round(accuracy(f11, exp(test)),2)
                ME   RMSE    MAE    MPE  MAPE MASE  ACF1
Training set 32.98 693.28 444.06 -12.66 39.70 0.52 -0.22
Test set     73.32 540.51 472.98   3.67 39.18 0.55  0.66
```

As mentioned previously, it is crucial to pick a suitable error measure and to be aware that most of the numbers reported by `accuracy()` are not sensible for use in a particular time series example.

[SCRIPT HAS ONLY BEEN UPDATED UP TO HERE, MORE UPDATES ON THE REMAINING FORECASTING TOPICS WILL FOLLOW]

8.1.3 Forecasting MA(1)

We here consider a pure, invertible $MA(1)$ process with mean zero:

$$X_t = E_t + \beta_1 E_{t-1}$$

E_t is an innovation with expectation zero and constant variance. As above, the forecast $\hat{X}_{n+k,1:n}$ will again be based on the conditional expectation $E[X_{n+k} \mid X_1,...,X_n]$. We get to a solution if we plug-in the model equation. First, we assume that $k \geq 2$, i.e. predict at least 2 time steps ahead.

$$\begin{aligned}\hat{X}_{n+k,1:n} &= E[X_{n+k} \mid X_1,...,X_n] \\ &= E[E_{n+k} + \beta_1 E_{n+k-1} \mid X_1,...,X_n] \\ &= E[E_{n+k} \mid X_1,...,X_n] + \beta_1 E[E_{n+k-1} \mid X_1,...,X_n] \\ &= 0\end{aligned}$$

The best $MA(1)$ forecast for horizons 2 and up is thus zero. Remember that we require E_t being an innovation, and thus independent from previous instances $X_s, s<t$ of the time series process. Next, we address the 1-step forecast. This is more problematic, because the above derivation leads to:

$$\begin{aligned}\hat{X}_{n+1,1:n} &= ... \\ &= \beta_1 E[E_n \mid X_1,...,X_n] \\ &\neq 0\ (generally)\end{aligned}$$

The 1-step forecast is generally different from zero. The term $E[E_n \mid X_1,...,X_n]$ is difficult to determine. Using some mathematical trickery, we can at least propose an approximate value. This trick is to move the point of reference into the infinite past, i.e. conditioning on all previous instances of the $MA(1)$ process. We denote

$$e_n := E[E_n \mid X_{-\infty}^n].$$

By successive substitution, we then write the $MA(1)$ as an $AR(\infty)$. This yields

$$E_n = \sum_{j=0}^{\infty} (-\beta_1)^j X_{n-j}.$$

If we condition the expectation of E_n on the infinite past of the series X_t, we can plug-in the realizations x_t and obtain:

$$E[E_n \mid X_{-\infty}^n] = e_n = \sum_{j=0}^{\infty} (-\beta_1)^j x_{n-j}.$$

This is of course somewhat problematic for practical implementation, because we only have realizations for $x_1,...,x_n$. However, because for invertible $MA(1)$ processes, $|\beta_1| < 1$, the impact of early observations dies out exponentially quickly. Thus, we let $x_t = 0$ for $t < 1$, and thus also have that $e_t = 0$ for $t < 1$. Also, we plug-in the estimated model parameter $\hat{\beta}_1$, and thus, the 1-step forecast for an $MA(1)$ is:

$$\hat{X}_{n+1,1:n} = \sum_{j=0}^{n-1} \hat{\beta}_1 (-\hat{\beta}_1)^j x_{n-j}$$

This is a sum of all observed values, with exponentially decaying weights.

8.1.4 Forecasting MA(q)

When forecasting from $MA(q)$ processes, we encounter the same difficulties as above. The prediction for horizons exceeding q are all zero, but anything below contains terms for which the considerations in section 8.1.3 are again necessary. We do without displaying this, and proceed to giving the formulae for $ARMA(p,q)$ forecasting, from which the ones for $MA(q)$ can be learned.

8.1.5 Forecasting ARMA(p,q)

We are considering stationary and invertible $ARMA(p,q)$ processes. The model equation for X_{n+1} then is:

$$X_{n+1} = \alpha_1 X_n + \ldots + \alpha_p X_{n+1-p} + E_{n+1} + \beta_1 E_n + \ldots + \beta_q E_{n+1-q}$$

As this model equation contains past innovations, we face the same problems as in section 8.1.3 when trying to derive the forecast for horizons $\leq q$. These can be mitigated, if we again condition on the infinite past of the process.

$$\begin{aligned}
\hat{X}_{n+1,\mathrm{l}:n} &= E[X_{n+1} \mid X_{-\infty}^{n}] \\
&= \sum_{i=1}^{p} \alpha_i E[X_{n+1-i} \mid X_{-\infty}^{n}] + E[E_{n+1} \mid X_{-\infty}^{n}] + \sum_{j=1}^{q} \beta_j E[E_{n+1-j} \mid X_{-\infty}^{n}] \\
&= \sum_{i=1}^{p} \alpha_i x_{n+1-i} + \sum_{i=1}^{q} \beta_j E[E_{n+1-j} \mid X_{-\infty}^{n}]
\end{aligned}$$

If we are aiming for k-step forecasting, we can use a recursive prediction scheme:

$$\hat{X}_{n+k,\mathrm{l}:n} = \sum_{i=1}^{p} \alpha_i E[X_{n+k-i} \mid X_{-\infty}^{n}] + \sum_{j=1}^{q} \beta_j E[E_{n+k-j} \mid X_{-\infty}^{n}],$$

where for the AR- and MA-part the conditional expectations are:

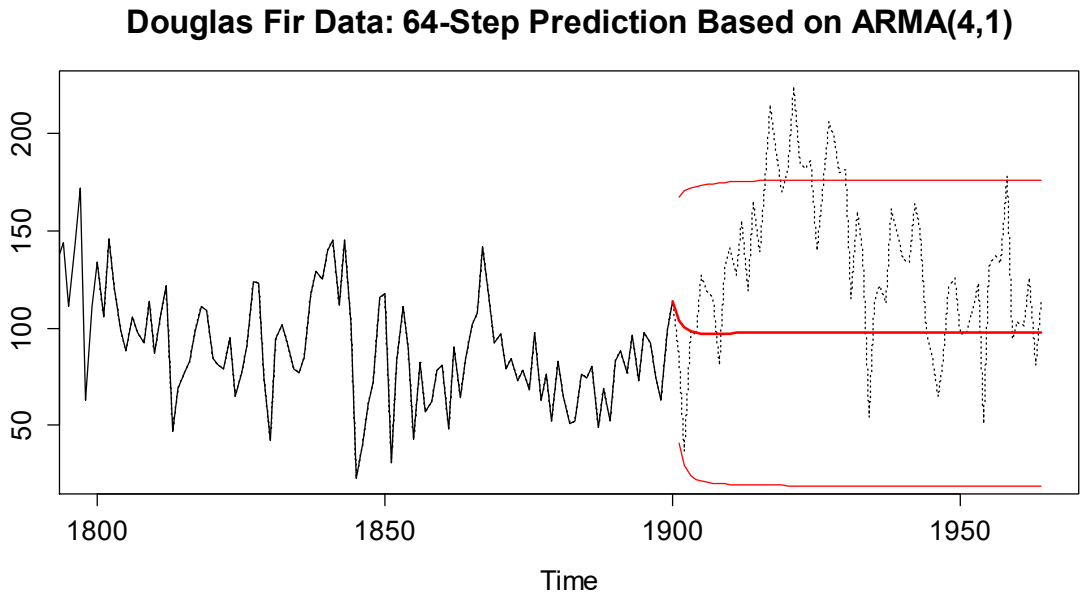
$$E[X_t \mid X_{-\infty}^{n}] = \begin{cases} x_t, & if\ t \leq n \\ \hat{X}_{t,\mathrm{l}:n}, & if\ t > n \end{cases}$$

$$E[E_t \mid X_{-\infty}^{n}] = \begin{cases} e_t, & if\ 0 < t \leq n \\ 0, & if\ t > n \end{cases}$$

The terms e_t are then determined as outlined above in section 8.1.3, and for the model parameters, we are plugging-in the estimates. This allows us to generate any forecast from an $ARMA(p,q)$ model that we wish. The procedure is also known as Box-Jenkins procedure, after the two researchers who first introduced it. Next, we illustrate this with a practical example, though in **R**, things are quite unspectacular. It is again the `predict()` procedure that is applied to a fit from `arima()`, the Box-Jenkins scheme that is employed runs in the background.

Practical Example

We here consider the Douglas Fir data which show the width of a tree's year rings over a period from 1107 to 1964. We choose to model the data without taking differences first. The `auto.arima()` solution with the lowest AIC value turned out to be an $ARMA(4,1)$ which will be used for generating the forecasts. For illustrative purpose, we choose to put the last 64 observations of the series aside so that we can verify our predictions. Then, the model is fitted and the Box-Jenkins forecasts are obtained. The result, including a 95% prognosis interval, is shown below. The R code used for producing the results, follows thereafter.




```
> train <- window(douglasfir, start=1107, end=1900)
> fit <- arima(train, order=c(4,0,1))
> fc  <- predict(fit, n.ahead=64)
> plot(window(douglasfir, 1800, 1964), lty=3, ylab="")
> lines(train, lwd=1)
> lines(fc$pred, lwd=2, col="red")
> lines(fc$pred+fc$se*1.96, col="red")
> lines(fc$pred-fc$se*1.96, col="red")
> title("Douglas Fir Data: 64-Step Prediction Based on …")
```

We observe that the forecast approaches the global mean of the series very quickly, in fact in an exponential decay. However, because there is an AR part in the model, all forecasts will be different from the global mean (but only slightly so for larger horizons). Then, since there is also a MA term in the model, all time series observations down to the first one from 1107 have some influence on the forecast. Again, the $ARMA$ model combines the properties from pure AR and MA processes. Regarding the quality of the forecast, we notice that it does not really provide much value for the true evolution of the series. Furthermore, the prediction intervals seem rather small. As it turns out, 12 out of 64 predictions (18.75%) violate the 95%

prediction interval. Is it bad luck or a problem with the model? We will reconsider this in the discussion about $ARIMA$ forecasting.

8.2 Series with Trend and Season

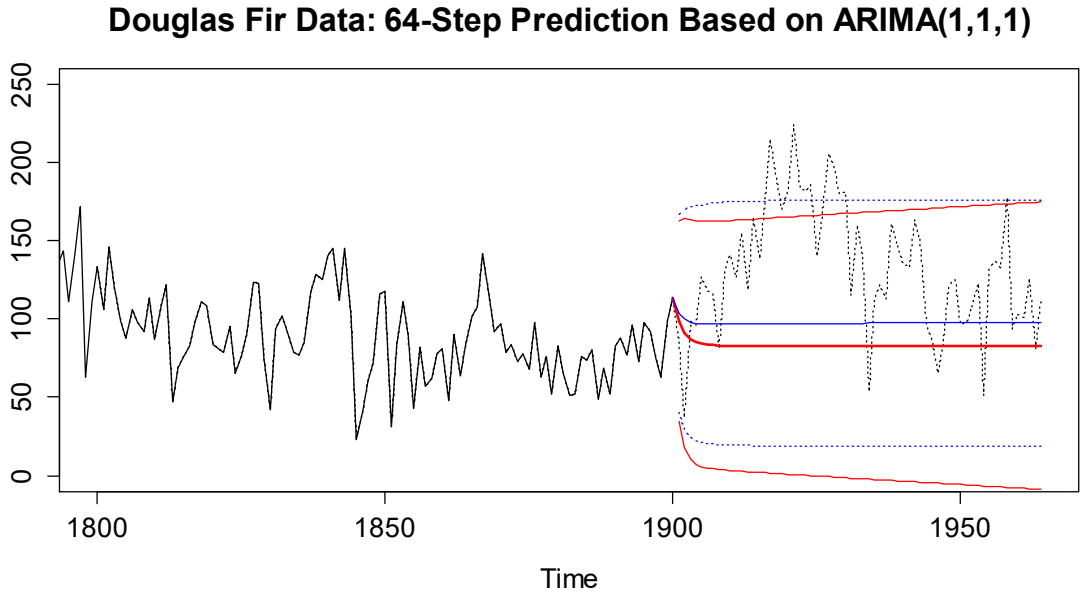
It is also possible and very important for practical purposes to produce forecasts for time series which have a trend, a seasonal effect or both. In this chapter, we present two different approaches, namely the one based on $ARIMA$ and $SARIMA$ models, and the other based on decomposing the series into trend, seasonal effect and stationary remainder.

8.2.1 Forecasting ARIMA and SARIMA

We here assume that we are given a series X_t which follows an $ARIMA(p,1,q)$. After taking differences at lag 1, we remain with $Y_t = X_t - X_{t-1}$ which is stationary and follows an $ARMA(p,q)$. Hence for Y_t, we know how to generate forecasts according to the recipe given in the previous section, and obtain $\hat{Y}_{n+1;1:n},...,\hat{Y}_{n+k;1:n}$. We are now seeking the k-step forecast for the original series X_t that has a trend. These are based on the notion of $\hat{X}_{n+1;1:n} = \hat{Y}_{n+1;1:n} + X_n$ and for obtaining arbitrary forecasts $\hat{X}_{n+k;1:n}$, we have to integrate and hence:

$$\begin{aligned}
\hat{X}_{n+1;1:n} &= \hat{Y}_{n+1;1:n} + X_n \\
\hat{X}_{n+2;1:n} &= \hat{Y}_{n+2;1:n} + \hat{X}_{n+1;1:n} = X_n + \hat{Y}_{n+1;1:n} + \hat{Y}_{n+2;1:n} \\
&\vdots \\
\hat{X}_{n+k;1:n} &= X_n + \hat{Y}_{n+1;1:n} + ... + \hat{Y}_{n+k;1:n}
\end{aligned}$$

As we can see, the k-step forecast for the original data is the cumulative sum of all forecasted terms of the differenced data. The formulae for the prediction intervals in an $ARIMA(p,1,q)$ forecast are difficult to derive and are beyond the scope of this script. All we say at this moment is that the width of the prediction interval does not converge as for an $ARMA(p,q)$ but is growing indefinitely with increasing forecasting horizon k. We illustrate this with a forecast for the Douglas Fir data using the non-stationary $ARIMA(1,1,1)$ model (*red*) and compare it to what we had obtained when a stationary $ARMA(4,1)$ was used (*blue*).



In this particular example, the $ARMA(4,1)$ forecast is more accurate and even the empirical coverage of its prediction interval is closer to the 95% that are are required by construction. However, this is an observation on one single dataset, it would be plain wrong to conclude that ARIMA forecasts are generally less accurate than the ones which are obtained from stationary models.

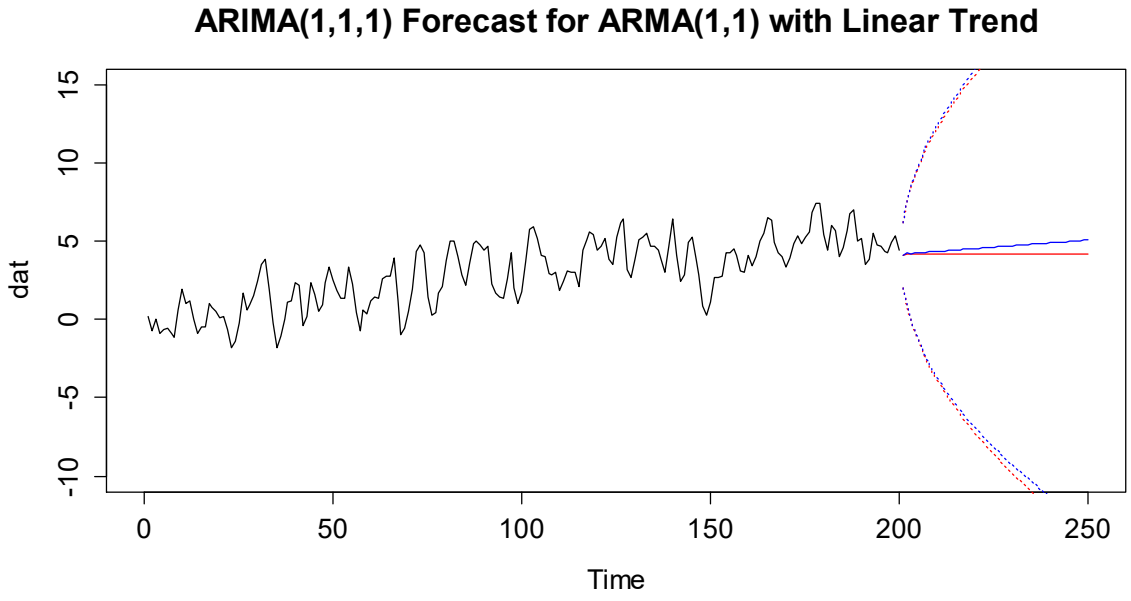
However, it is crucial to understand what ARIMA forecasts can do and what they cannot do. Despite the fact that ARIMA models are for non-stationary time series, the forecast will converge to a constant if $d = 1$. So in case of a series with a deterministic, linear trend, a default ARIMA forecast will miserably fail, see the example below. To be fair however, we need to point out that default ARIMA processes feature a unit root and are non-stationary, but are not compatible with a deterministic linear trend.

To underline the issue, we present an artificial example, where an $ARMA(1,1)$ process was superimposed with a linear trend. The resulting series is non-stationary, and as differencing can make it stationary (though with non-zero mean), a $ARIMA(1,1,1)$ was used for modelling and generating forecasts. As the output shows, the forecast generated with the R function `arima()` fails to pick up the obvious trend and hence is of poor quality. The alternatives consist in enhancing the ARIMA models or using the method presented in section 8.2.2.

```
> dat <- arima.sim(list(ar=0.5,ma=0.5), n=200) + (1:200)*0.03
> fit <- arima(dat, order=c(1,1,1))
> plot(dat, xlim=c(0,250), ylim=c(-10,15), main="…")
> pred <- predict(fit, n.ahead=50)
> lines(pred$pred, col="red")
> lines(pred$pred + 1.96*pred$se, col="red", lty=3)
> lines(pred$pred - 1.96*pred$se, col="red", lty=3)
```

As mentioned above, a default $ARIMA(1,1,1)$ model is misspecified in a situation with linear trend, as it assumes an $ARMA(1,1)$ with zero mean after differencing, which is not the case. To act correctly, we need to add a so-called drift term (i.e. a non-trivial global mean for the $ARMA(1,1)$) to the model which leads to a forecast that (due to reintegration of the constant) has a linear increase and much better reflects reality. In R function `arima()`, adding such a term is possible, but not that obvious as the `xreg` argument needs being used, see the code below.

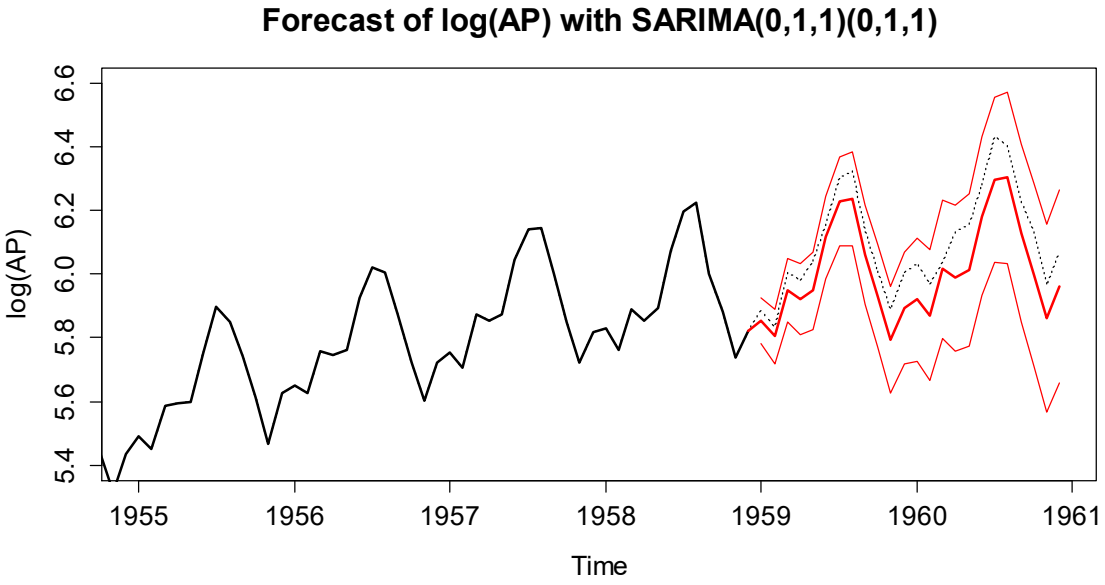
```
> fitd  <- arima(dat, order=c(1,1,1), xreg=1:200)
> predd <- predict(fitd, n.ahead=50, newxreg=201:250)
> lines(predd$pred, col="blue")
> lines(predd$pred + 1.96*pred$se, col="blue", lty=3)
> lines(predd$pred - 1.96*pred$se, col="blue", lty=3)
```



Two things need to be emphasized. First, function `auto.arima()` does consider adding drift terms automatically if the arguments are set accordingly. In the example presented here, the method identifies an $ARIMA(3,1,2)$ with drift term as the best fitting model and produces a forecast that is linearly increasing. Nevertheless, in practice, where we do not have intimate knowledge about the data generating process, careful modelling with ARIMA (and potentially adding drift terms) is important for producing successful forecasts. In many cases, it may seem easier to decompose a series into trend, (season) and remainder as it is easier to take care of each component on its own.

We continue with presenting an example of a SARIMA forecast. We do without giving much detail here, but only remark that these are also based on producing forecasts for the differenced, stationary series and subsequent integration. Again, careful modelling is generally required to get the trend extrapolation right. As we see in our example for the Air Pax data, we also undershoot the trend development somewhat. Again, adding a drift term may be successful here.

```
> fit    <- arima(shlap, order=c(0,1,1), seasonal=c(0,1,1))
> pred   <- predict(fit, n.ahead=24)
> plot(…)
```



8.2.2 Forecasting Decomposed Series

Another approach for forecasting series with deterministic trend and/or seasonality is based on the descriptive decomposition. The paradigm is as follows:

- **Trend**

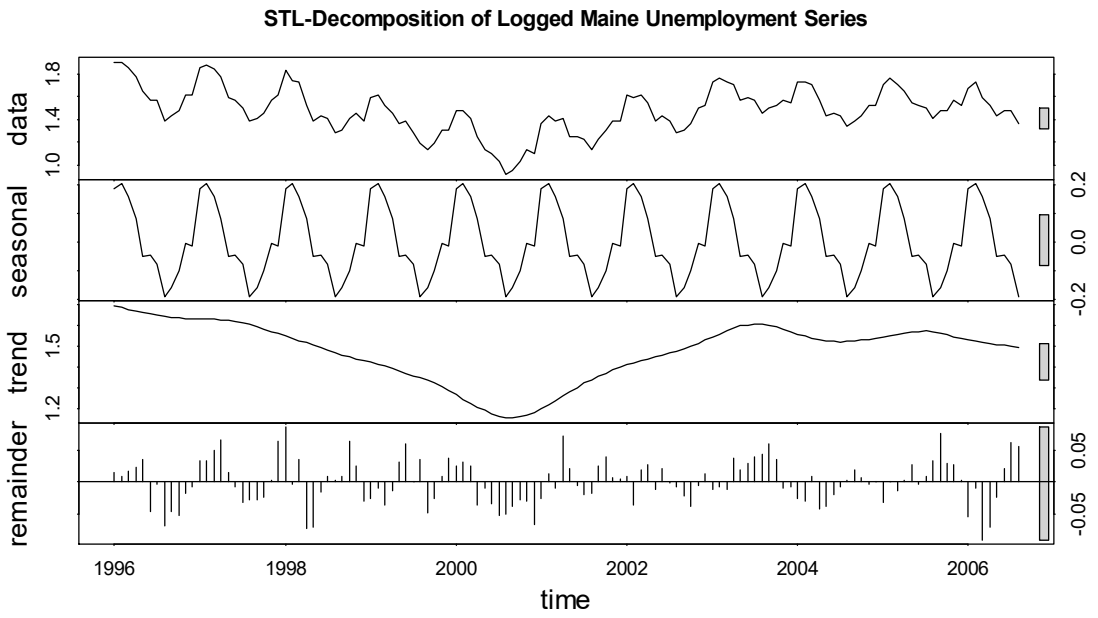
  We assume a smooth trend for which we recommend linear extrapolation.
- **Seasonal Effect**

  We extrapolate the seasonal effect according to the last observed period.
- **Stationary Remainder**

  We fit an $ARMA(p,q)$ and determine the forecast as discussed above.

We illustrate the procedure on the Maine unemployment data. We will work with the log-transformed data, for which an STL decomposition under assuming a constant seasonal effect was performed.

```
> fit <- stl(log(tsd), s.window="periodic")
> plot(fit, main="…")
```

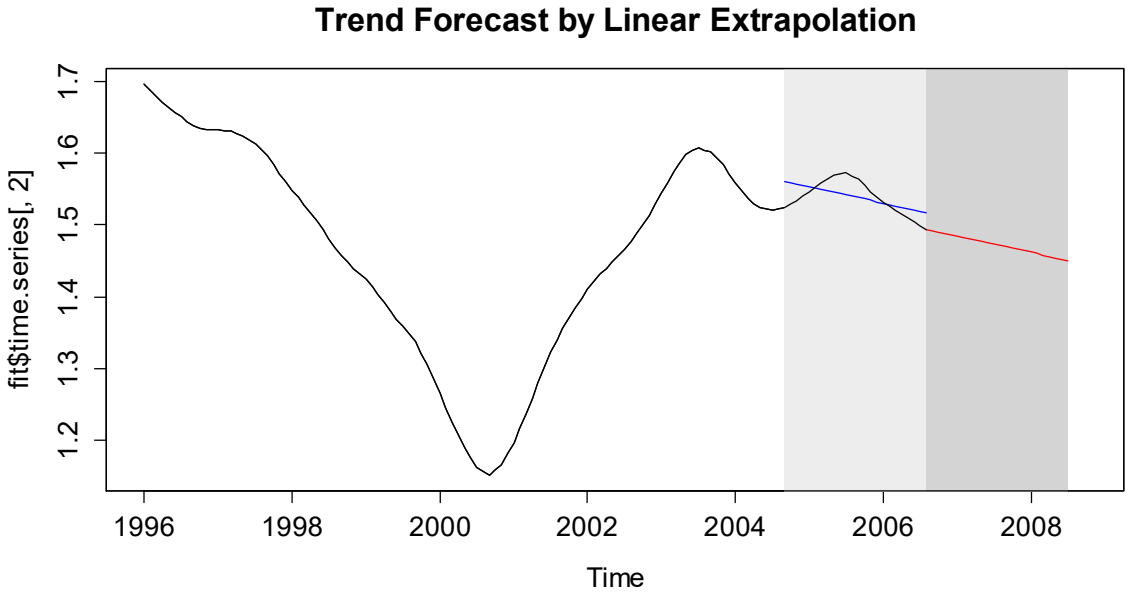


We first focus on the central issue which is the trend extrapolation. We recommend the following procedure:

> *Fit a least squares regression line into the past trend values. The window on which this fit happens is chosen such that is has the same length as the forecasting horizon. In our particular example, where we want to forecast the upcoming two years of the series, these are the last 24 data points. Or in other words: for the trend forecast, we use the last observation as an anchor point and predict with the average slope from the last two years.*

Please note that the so-produced trend forecast is a recommendation, but not necessarily the best solution. If some expert knowledge from the application field suggests another trend extrapolation, then it may well be used. It is however important, to clearly declare how the trend forecast was determined. The following code does the job, see next page for the result:

```
> ## STL decomposition
> fit <- stl(log(tsd), s.window="periodic")
>
> ## Trend Forecast by Linear Extrapolation
> plot(fit$time.series[,2], xlim=c(1996, 2008+9/12))
> rect(2004+8/12, 1 , 2006+7/12, 2, col="grey93", border=NA)
> rect(2006+7/12, 1, 2008+6/12, 2, col="grey83", border=NA)
> title("Trend Forecast by Linear Extrapolation")
> xx <- time(fit$time.series[,2])[105:128]
> yy <- fit$time.series[105:128,2]
> fit.regr <- lm(yy~xx)
> t.fore   <- 1.494 + (0:23)/12 * coef(fit.regr)[2])
> lines(xx, fitted(fit.regr), col="blue")
> lines(xx[1]+(23:46)/12, t.fore, col="red")
> lines(fit$time.series[,2])
> box()
```

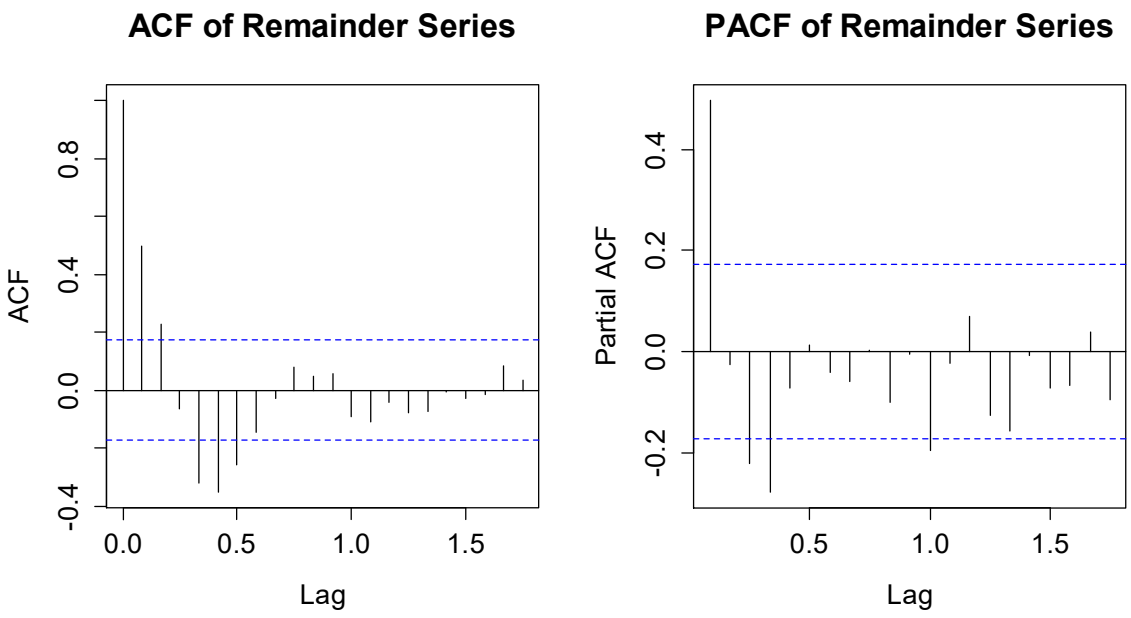


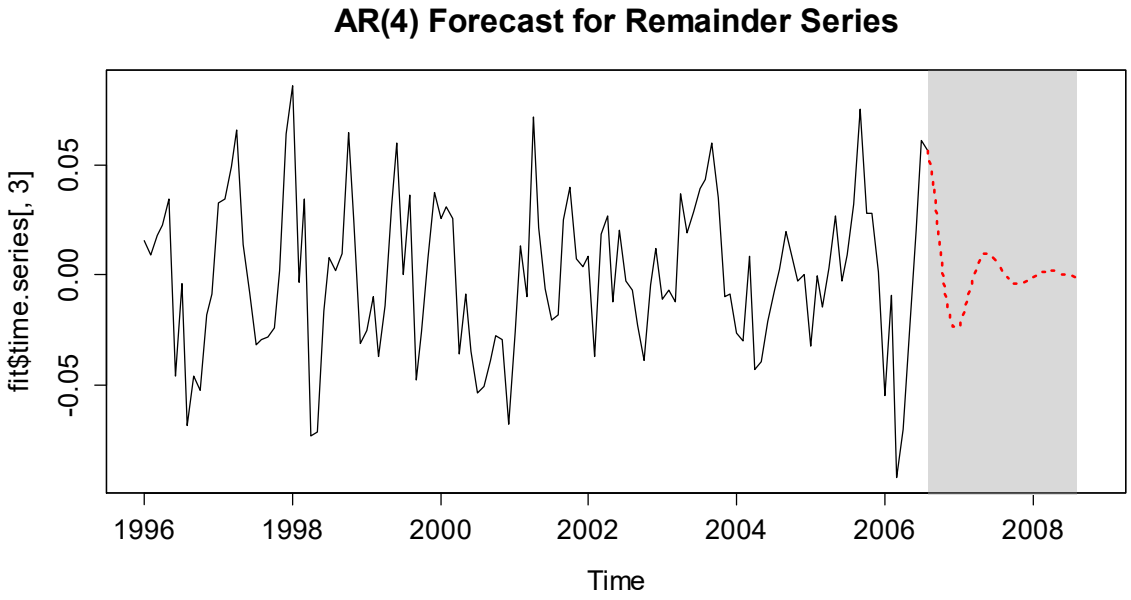
The blue line in the light grey window indicates the average trend over the last two years of observations. For the trend extrapolation, we use the last observed trend value as the anchor, and then continue using the determined slope. Please note that generally (though not in the `stl()` context), function `loess()` in R allows for extrapolation if argument `surface= "direct "`. However, according to the author's experience, such trend extrapolations are often extreme and perform worse than the linear extrapolation that is suggested here. As we have now solved the issue with the trend, we remain with forecasting the seasonal effect and the remainder term. For the former, things are trivial, as we assume that it stays as it was last. The R code for producing the forecast of the seasonal component is:

```
## Seasonal Forecast Using Last Values
season <- fit$time.series[,1]
l2y    <- window(season,start=c(2004,9),end=c(2006,8))
s.fore <- ts(l2y, start=c(2006,9), end=c(2008,8), freq=12)
```

Hence, we only need to take care of the stationary remainder. Generally, this is a stationary series that will be described with an $ARMA(p,q)$, for which the forecasting method has already been presented in chapter 8.1. Hence, we here focus on the particular case at hand, where a simple solution is to recognize an exponential decay in the ACF and a cut-off at lag 4 in the PACF, so that an $AR(4)$ model (here without using a global mean!) will be fitted. Residual analysis (not shown here) indicates that the White Noise assumption for the estimated innovation terms is justified in this case. Using the `predict()` command, we then produce a 24-step forecast from the $AR(4)$ for the stationary remainder.

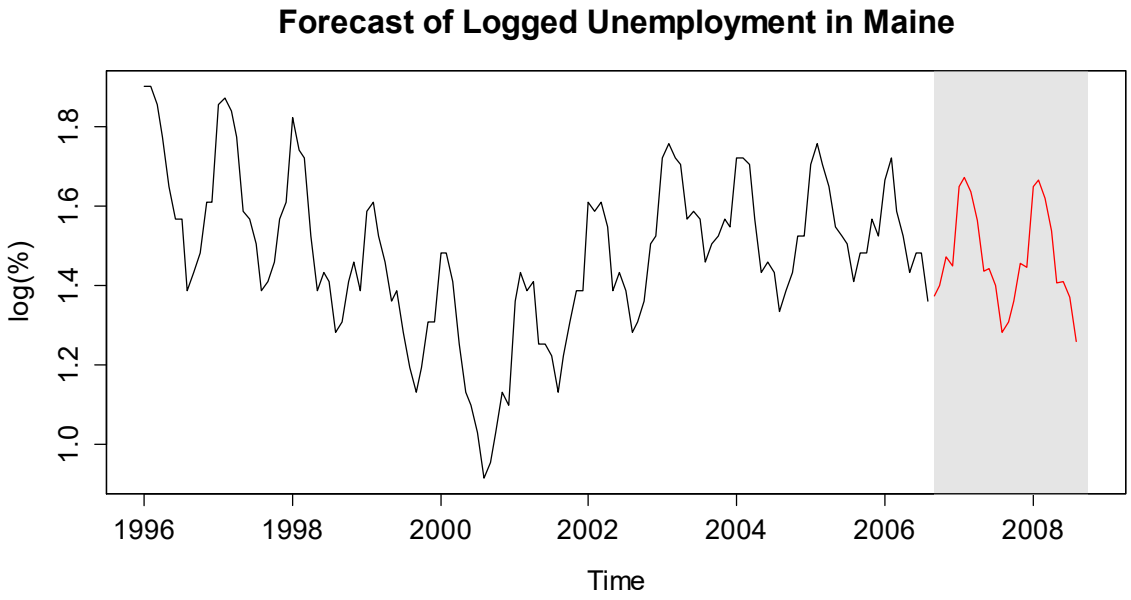
```
> rmndr     <- fit$time.series[,3]
> fit.rmndr <- arima(rmndr, order=c(4,0,0), include.mean=F)
> r.fore    <- predict(fit.rmndr, n.ahead=24)$pred
```





The final task is then to couple the forecasts for all three parts (trend, seasonal component and remainder) to produce a 2-year-forecast for the original series with the logged unemployment figures from the state of Maine. This is based on a simple addition of the three components.

```
> ## Adding the 3 Components
> fore <- t.fore + s.fore + r.fore
> ## Displaying the Output
> plot(log(tsd), xlim=c(1996, 2008.75), ylab="log(%)")
> rect(2006+8/12, 0, 2008+9/12, 2, col="grey90", border=NA)
> lines(fore, col="red")
> box()
```



This procedure is a practical way for forecasting decomposed series. It is especially attractive if one wants to have a second thought on the predicted trend, and maybe correct it manually, based on deeper insight e.g. into the corporate plans about increasing or decreasing the market share, upcoming competitors, et cetera. On the downside, the procedure requires somewhat more effort for coming up with the forecasts, when compared to the SARIMA model and exponential smoothing. The choice of the right method however, depends on the use case. Another disadvantage is the lack of prediction intervals here – most of the uncertainty in the prediction comes from the trend extrapolation, for which it is not possible to give a reasonable interval. While the other methods for forecasting non-stationary series technically do provide prediction intervals, they are according to the opinion of the author, often too small as they do not reflect the uncertainties that come from trend extrapolation. Hence it may be more genuine not to provide an interval at all, rather than a flawed one.

8.3 Exponential Smoothing

8.3.1 Simple Exponential Smoothing

The objective in this section is to predict some future values X_{n+k} given an observed series $\{X_1,...,X_n\}$, and thus no different than before. We first assume that the data do not exhibit any deterministic trend or seasonality, or that these have been identified and removed. The (conditional) expected value of the process can change from one time step to the next, but we do not have any information about the direction of this change. A typical application is forecasting sales of a well-established product in a stable market. The model is:

$$X_t = \mu_t + E_t,$$

where μ_t is the non-stationary mean of the process at time t, and E_t are independent random innovations with expectation zero and constant variance σ_E^2. We will here use the same notation as **R** does, and let a_t, called *level* of the series, be our estimate of μ_t. By assuming that there is no deterministic trend, an intuitive estimate for the level at time t is to take a weighted average of the current time series observation and the previous level:

$$a_t = \alpha x_t + (1-\alpha)a_{t-1}, \text{ with } 0 < \alpha < 1.$$

Apparently, the value of α determines the amount of smoothing: if it is near 1, there is little smoothing and the level a_t closely tracks the series x_t. This would be appropriate if the changes in the mean of the series are large compared to the innovation variance σ_E^2. At the other extreme, an α-value near 0 gives highly smoothed estimates of the current mean which take little account of the most recent observation. This would be the way to go for series with a large amount of noise compared to the signal size. A typical default value is $\alpha = 0.2$, chosen in the light that for most series, the change in the mean between t and $t-1$ is smaller than σ_E^2. Alternatively, it is (with **R**) also possible to estimate α, see below.

Because we assume absence of deterministic trend and seasonality, the best forecast at time n for the future level of the series, no matter what horizon we are aiming for, is given by the level estimate at time n, i.e.

$$\hat{X}_{n+k,1:n} = a_n, \text{ for all } k = 1, 2, \ldots.$$

We can rewrite the weighted average equation in two further ways, which yields insight into how exponential smoothing works. Firstly, we can write the level at time t as the sum of a_{t-1} and the 1-step forecasting error and obtain the *update formula*:

$$a_t = \alpha(x_t - a_{t-1}) + a_{t-1}$$

Now, if we repeatedly apply back substitution, we obtain:

$$a_t = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + \ldots$$

When written in this form, we see that the level a_t is a linear combination of the current and all past observations with more weight given to recent observations. The restriction $0 < \alpha < 1$ ensures that the weights $\alpha(1-\alpha)^i$ become smaller as i increases. In fact, they are exponentially decaying and form a geometric series. When the sum over these terms is taken to infinity, the result is 1. In practice, the infinite sum is not feasible, but can be avoided by specifying $a_1 = x_1$.

For any given smoothing parameter α, the update formula plus the choice of $a_1 = x_1$ as a starting value can be used to determine the level a_t for all times $t = 2, 3, \ldots$. The 1-step prediction errors e_t are given by:

$$e_t = x_t - \hat{x}_{t,1:(t-1)} = x_t - a_{t-1}.$$

By default, R obtains a value for the smoothing parameter α by minimizing the sum of squared 1-step prediction errors, called $SS1PE$:

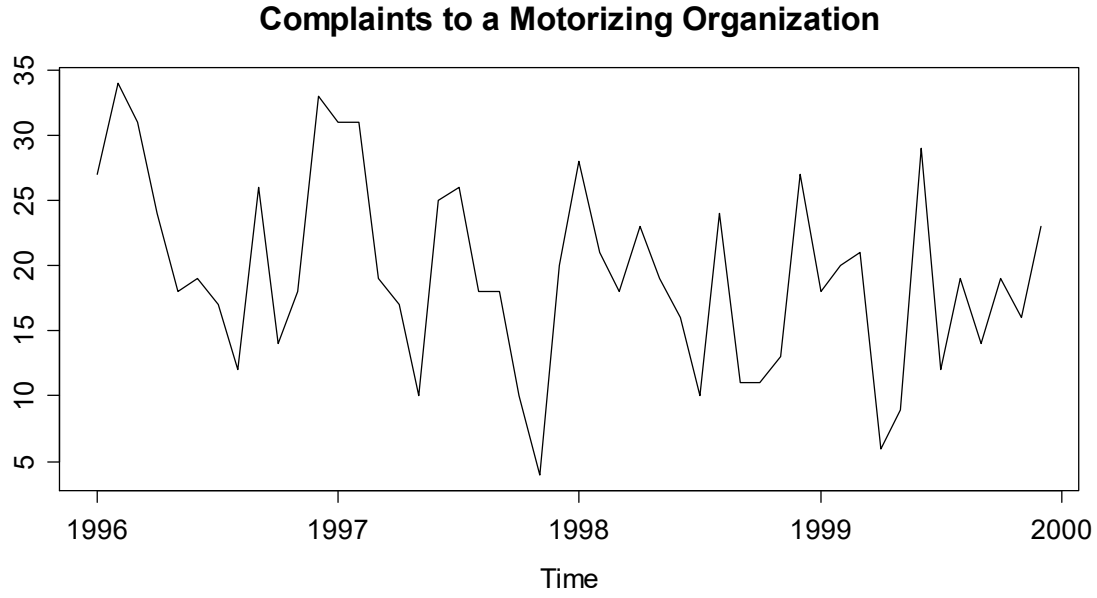
$$SS1PE = \sum_{t=2}^{n} e_t^2 .$$

There is some mathematical theory that examines the quality of the $SS1PE$-minimizing α. Not surprisingly, this depends very much on the true, underlying process. However in practice, this value is reasonable and allows for good predictions.

Practical Example

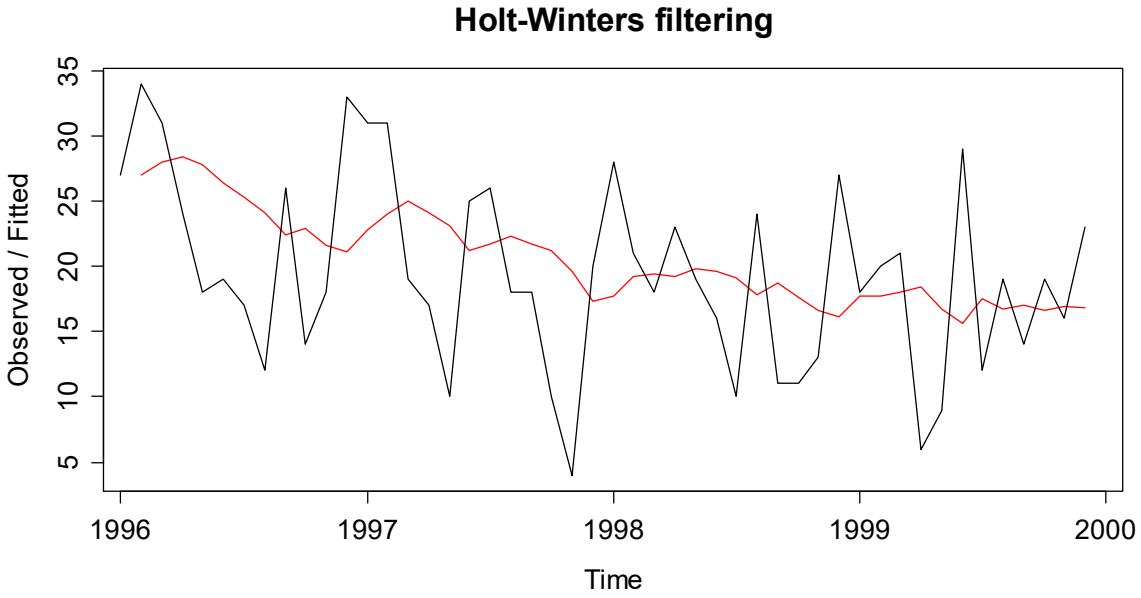
We here consider a time series that shows the number of complaint letters that were submitted to a motoring organization over the four years 1996-1999. At the beginning of year 2000, the organization wishes to estimate the current level of complaints and investigate whether there was any trend in the past. We import the data and do a time series plot:

```
> www <- "http://staff.elena.aut.ac.nz/Paul-Cowpertwait/ts/"
> dat <- read.table(paste(www,"motororg.dat",sep="", head=T)
> cmpl <- ts(dat$complaints, start=c(1996,1), freq=12)
> plot(cmpl, ylab="", main="Complaints ...")
```



The series is rather short, and there is no clear evidence for a deterministic trend and/or seasonality. Thus, it seems sensible to use exponential smoothing here. The algorithm that was described above is implemented in **R**'s `HoltWinters()` procedure. Please note that `HoltWinters()` can do more than plain exponential smoothing, and thus we have to set arguments `beta=FALSE` and `gamma=FALSE`. If we do not specify a value for the smoothing parameter α with argument `alpha`, it will be estimated using the $SS1PE$ criterion.

```
> fit <- HoltWinters(cmpl, beta=FALSE, gamma=FALSE); fit
Call: HoltWinters(x = cmpl, beta = FALSE, gamma = FALSE)
Smoothing parameters:
 alpha:  0.1429622
 beta :  FALSE
 gamma:  FALSE
Coefficients:
      [,1]
a 17.70343
> plot(fit)
```



The output shows that the level in December 1999, this is a_{48}, is estimated as 17.70. The optimal value for α according to the $SS1PE$ criterion is 0.143, and the sum of squared prediction errors was 2502. Any other value for α will yield a worse result, thus we proceed and display the result visually.

8.3.2 The Holt-Winters Method

The simple exponential smoothing approach from above can be generalized for series which exhibit deterministic trend and/or seasonality. As we have seen in many examples, such series are the norm rather than the exception and thus, such a method comes in handy. It is based on these formulae:

$$\begin{aligned} a_t &= \alpha(x_t - s_{t-p}) + (1-\alpha)(a_{t-1} + b_{t-1}) \\ b_t &= \beta(a_t - a_{t-1}) + (1-\beta)b_{t-1} \\ s_t &= \gamma(x_t - a_t) \quad + (1-\gamma)s_{t-p} \end{aligned}$$

In the above equations, a_t is again the level at time t, b_t is called the slope and s_t is the seasonal effect. There are three smoothing parameters α, β, γ which are aimed at level, slope and season. The explanation of the equations is as follows:

- The first updating equation for the level takes a weighted average of the most recent observation with the existing estimate of the previous' period seasonal effect term subtracted, and the 1-step level forecast at $t-1$, which is given by level plus slope.

- The second updating equation takes a weighted average of the difference between the current and the previous level with the estimated slope at time $t-1$. Note that this can only be computed if a_t is available.

- Finally, we obtain another estimate for the respective seasonal term by taking a weighted average of the difference between observation and level with the previous estimate of the seasonal term for the same unit, which was made at time $t-p$.

If nothing else is known, the typical choice for the smoothing parameters is $\alpha = \beta = \gamma = 0.2$. Moreover, starting values for the updating equations are required. Mostly, one chooses $a_1 = x_1$, the slope $b_1 = 0$ and the seasonal effects $s_1,...,s_p$ are either also set to zero or to the mean over the observations of a particular season. When applying the R function `HoltWinters()`, the starting values are obtained from the `decompose()` procedure, and it is possible to estimate the smoothing parameters through $SS1PE$ minimization. The most interesting aspect are the predictions, though: the k-step forecasting equation for X_{n+k} at time n is:

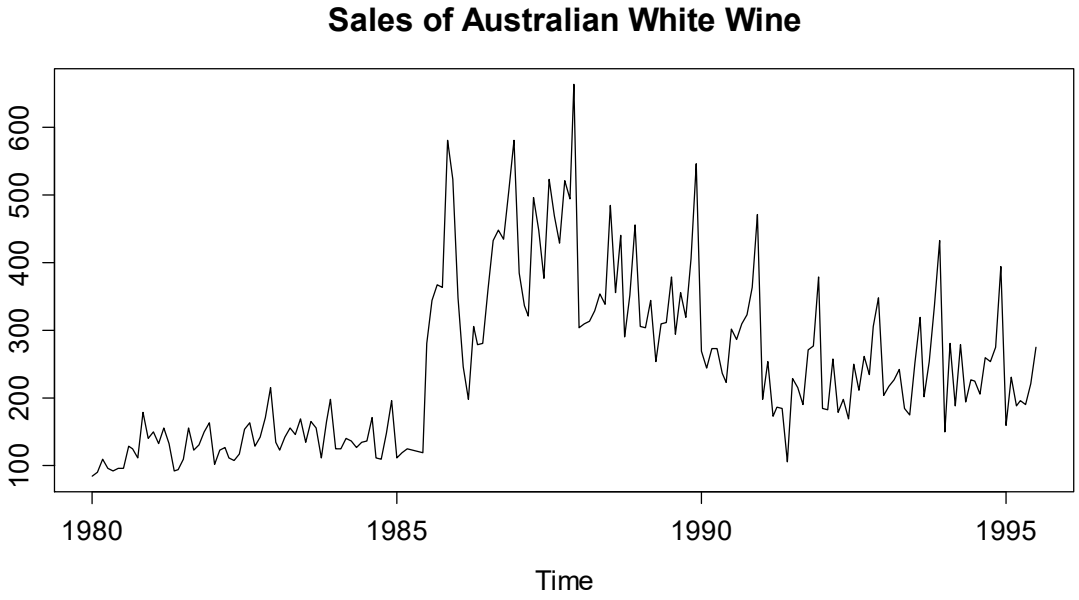
$$\hat{X}_{n+k,1:n} = a_n + kb_n + s_{n+k-p},$$

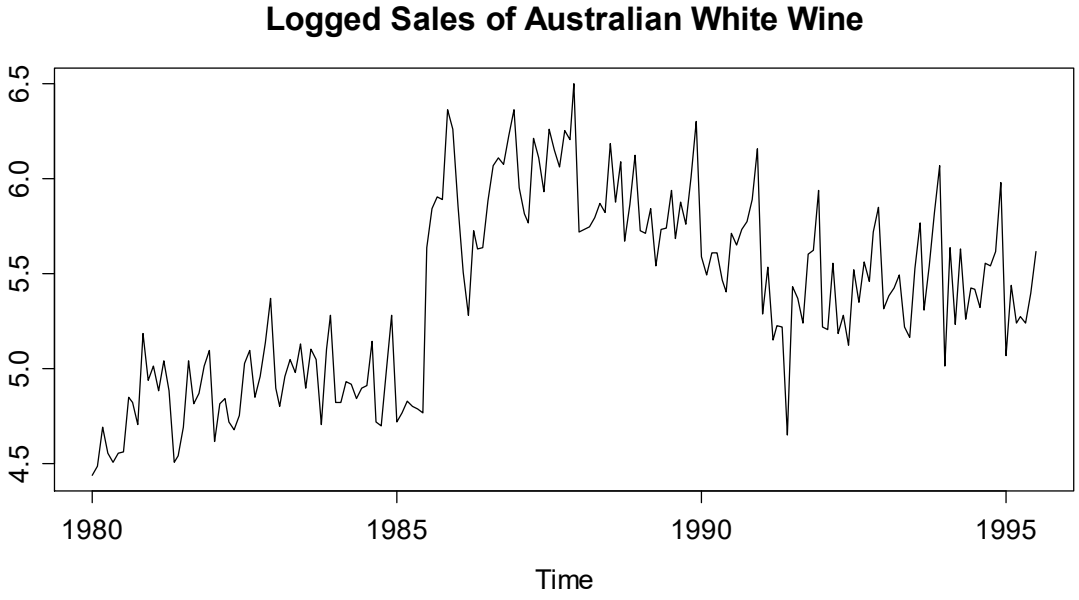
i.e. the current level with linear trend extrapolation plus the appropriate seasonal effect term. The following practical example nicely illustrates the method.

Practical Example

We here discuss the series of monthly sales (in thousands of litres) of Australian white wine from January 1980 to July 1995. This series features a deterministic trend, the most striking feature is the sharp increase in the mid-80ies, followed by a reduction to a distinctly lower level again. The magnitude of both the seasonal effect and the errors seem to be increasing with the level of the series, and are thus multiplicative rather than additive. We will cure this by a log-transformation of the series, even though there exists a multiplicative formulation of the Holt-Winters algorithm, too.

```
> www <- "http://staff.elena.aut.ac.nz/Paul-Cowpertwait/ts/"
> dat <- read.table(paste(www,"wine.dat",sep="", header=T)
> aww  <- ts(dat$sweetw, start=c(1980,1), freq=12)
> plot(aww, ylab="", main="Sales of Australian White Wine")
> plot(log(aww), ylab="", main="Logged Sales ...")
```





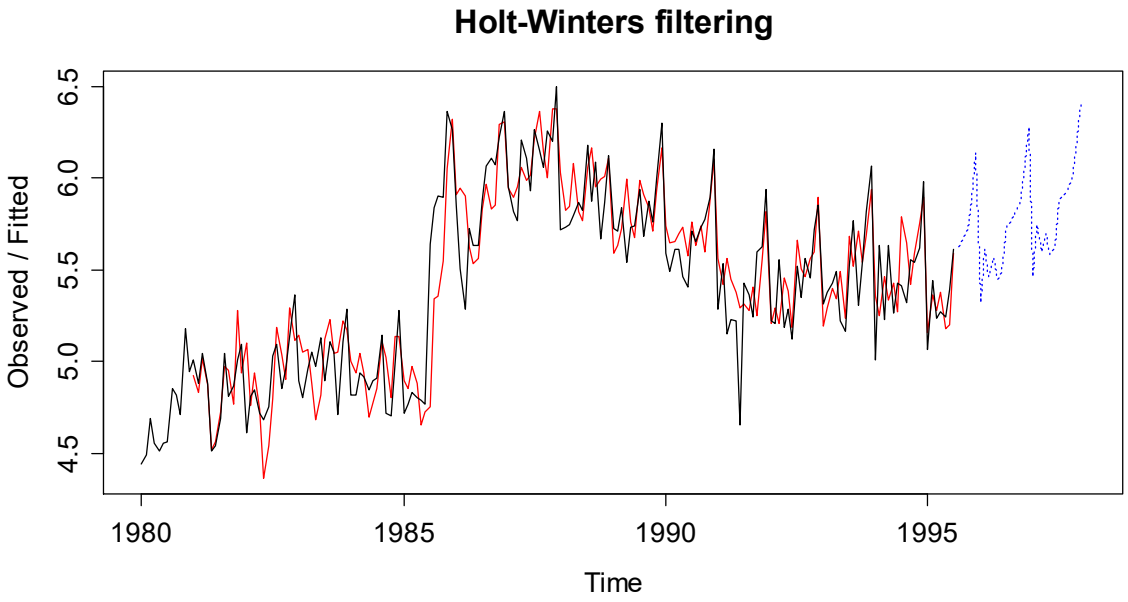
The transformation seems successful, thus we proceed to the Holt-Winters modeling. When we apply parameter estimation by $SS1PE$, this is straightforward. The fit contains the current estimates for level, trend and seasonality. Note that these are only valid for time n, and not for the entire series. Anyhow, it is much better to visualize the sequence of a_t, b_t and γ_t graphically. Moreover, plotting the fitted values along with the time series is informative, too.

```
> fit
Call: HoltWinters(x = log(aww))
Smoothing parameters:
 alpha:  0.4148028
 beta :  0
 gamma:  0.4741967
```

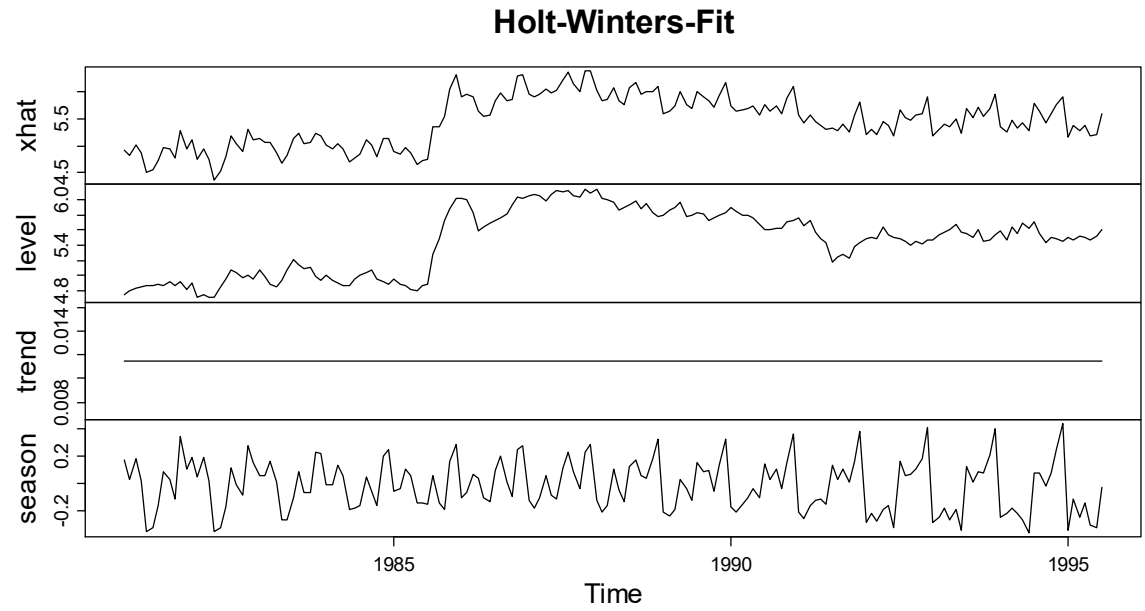
```
Coefficients:
a     5.62591329   s4    0.20894897   s9   -0.17107682
b     0.01148402   s5    0.45515787   s10 -0.29304652
s1   -0.01230437   s6   -0.37315236   s11 -0.26986816
s2    0.01344762   s7   -0.09709593   s12 -0.01984965
s3    0.06000025   s8   -0.25718994
```

The coefficient values (at time n) are also the ones which are used for forecasting from that series with the formula given above. We produce a prediction up until the end of 1998, which is a 29-step forecast. The R commands are:

```
> plot(fit, xlim=c(1980, 1998))
> lines(predict(fit, n.ahead=29), col="blue", lty=3)
```

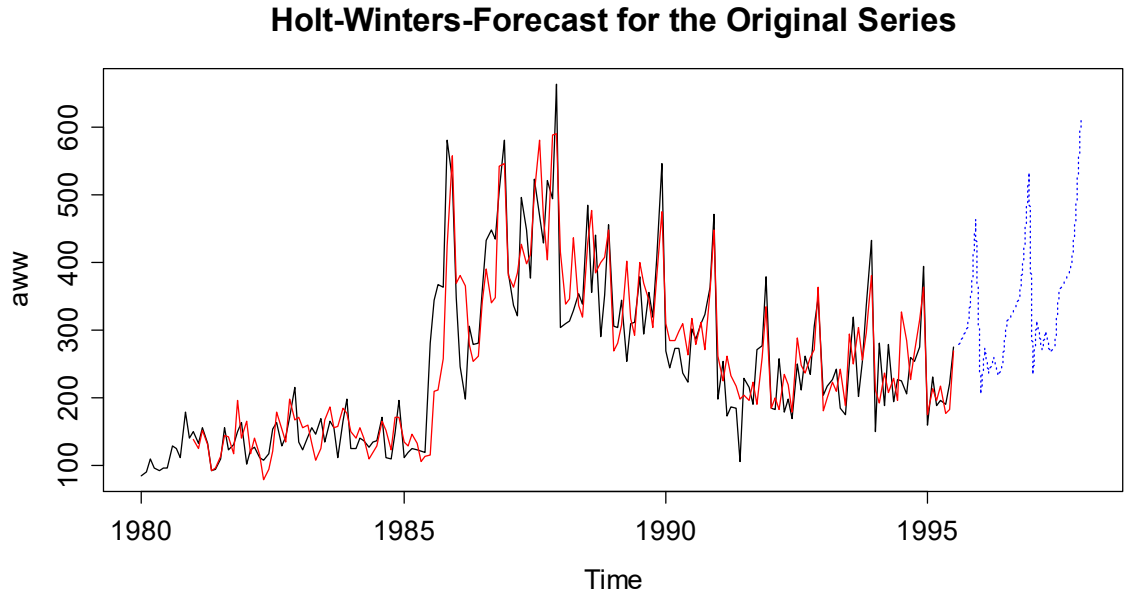



```
> plot(fit$fitted, main="Holt-Winters-Fit")
```



The last plot on the previous page shows how level, trend and seasonality evolved over time. However, since we are usually more interested in the prediction on the original scale, i.e. in liters rather than log-liters of wine, we just re-exponentiate the values. Please note that the result is an estimate of the median rather than the mean of the series. There are methods for correction, but the difference is usually only small.

```
> plot(aww, xlim=c(1980, 1998))
> lines(exp(fit$fitted[,1]), col="red")
> lines(exp(predict(fit, n.ahead=29)), col="blue", lty=3)
```



Also, we note that the (insample) 1-step prediction error is equal to 50.04, which is quite a reduction when compared to the series' standard deviation which is 121.4. Thus, the Holt-Winters fit has substantial explanatory power. Of course, it would now be interesting to test the accuracy of the predictions. We recommend that you, as an exercise, put aside the last 24 observations of the Australian white wine data, and run a forecasting evaluation where all the methods (SARIMA, decomposition approaches, Holt-Winters) compete against each other.

9 Multivariate Time Series Analysis

While the header of this section says *multivariate* time series analysis, we will here restrict to two series series $X_1 = (X_{1,t})$ and $X_2 = (X_{2,t})$, and thus *bivariate* time series analysis, because an extension to more than two series is essentially analogous. Please note that a prerequisite for all the theory in this section is that the *series* X_1 *and* X_2 *are stationary*.

Generally speaking, the goal of this section is to describe and understand the (inter)dependency between two series. We introduce the basic concepts of cross correlation and transfer function models, warn of arising difficulties in interpretation and show how these can be mitigated.

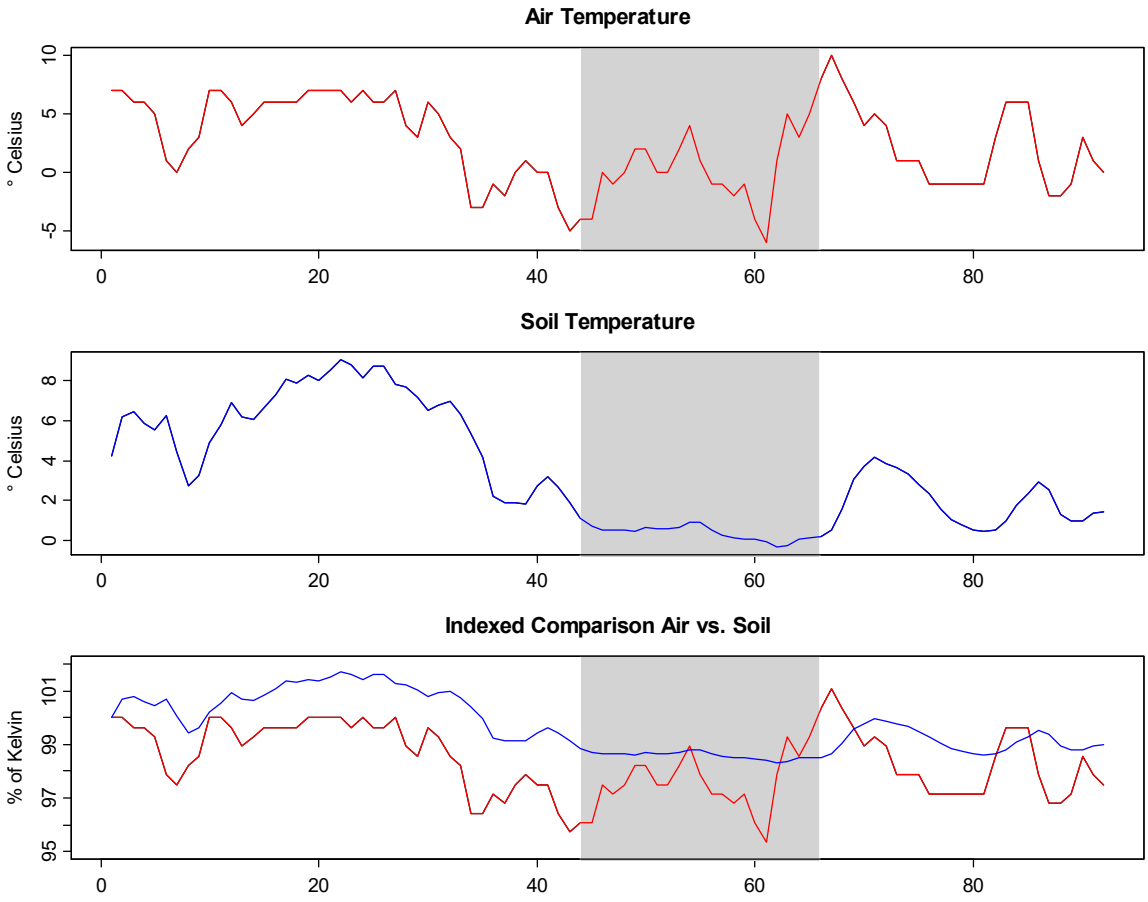
9.1 Practical Example

We will illustrate the theory on multivariate time series analysis with a practical example. The data were obtained in the context of the diploma thesis of Evelyn Zenklusen Mutter, a former WBL student who works for the Swiss Institute for Snow and Avalanche Research SLF. The topic is how the ground temperature in permafrost terrain depends on the ambient air temperature. The following section gives a few more details.

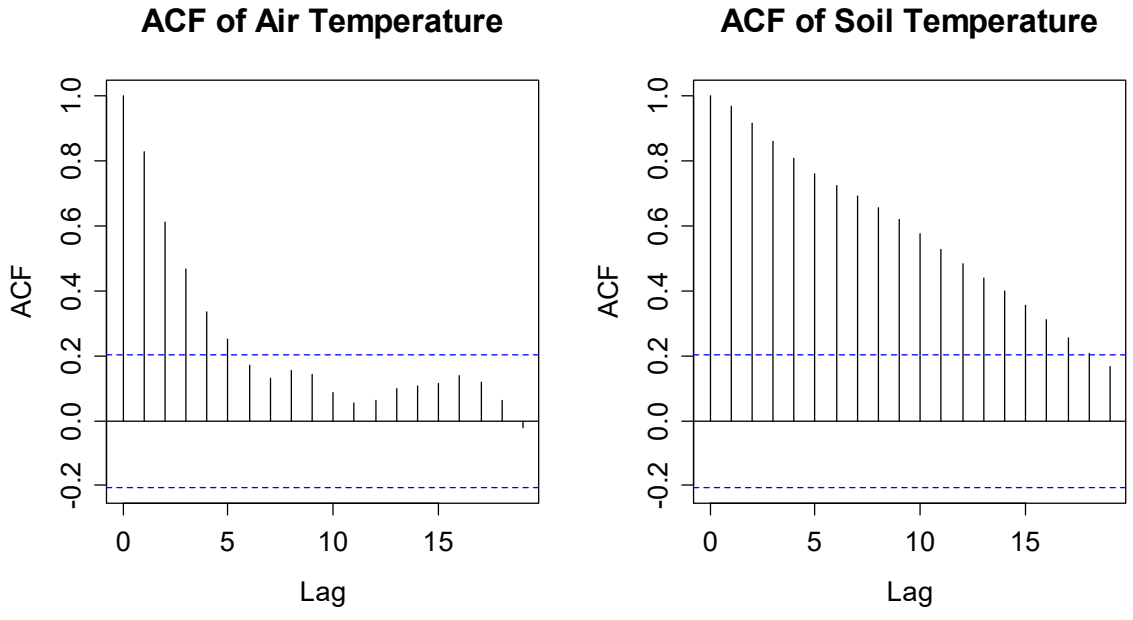
Ambient air temperatures influence ground temperatures with a certain temporal delay. Borehole temperatures measured at 0.5m depth in alpine permafrost terrain, as well as air temperatures measured at or nearby the boreholes will be used to model this dependency. The reaction of the ground on the air temperature is influenced by various factors such as ground surface cover, snow depth, water or ground ice content. To avoid complications induced by the insulating properties of the snow cover and by phase changes in the ground, only the snow-free summer period when the ground at 0.5m is thawed will be considered.

We here consider only one single borehole, it is located near the famous *Hörnli hut* at the base of *Matterhorn* near Zermatt/CH on 3295m above sea level. The air temperature was recorded on the nearby *Platthorn* at 3345m of elevation and 9.2km distance from the borehole. Data are available from beginning of July 2006 to the end of September 2006. After the middle of the observation period, there is a period of 23 days during which the ground was covered by snow, highlighted in grey color in the time series plots on the next page.

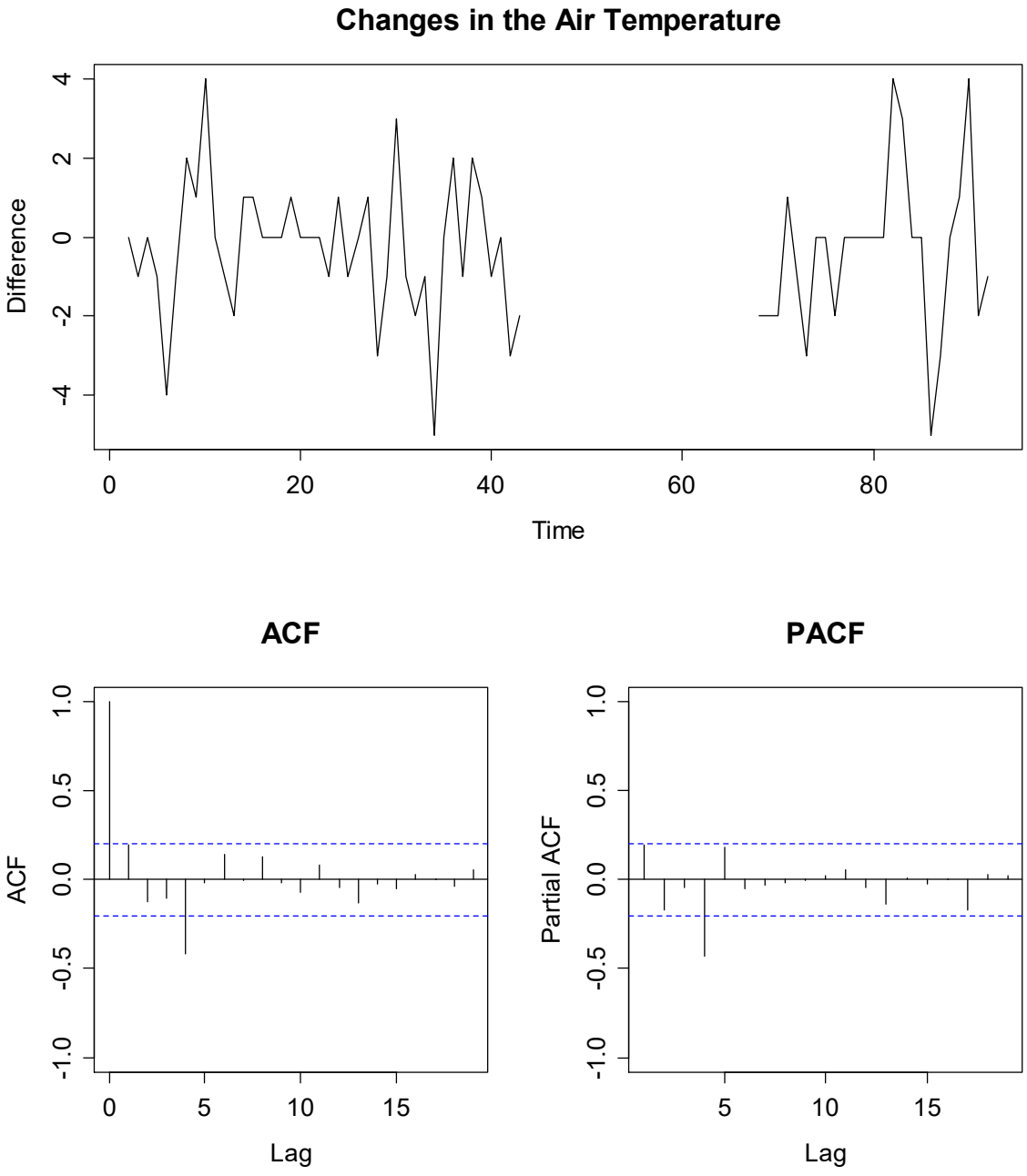
Because the snow insulates the ground, we do not expect the soil to follow the air temperature during that period. Hence, we set all values during that period equal to NA. The time series plots, and especially the indexed plot where both series are shown, clearly indicate that the soil reacts to the air temperature with a delay of a few days. We now aim for analyzing this relationship on a more quantitative basis, for which the methods of multivariate time series analysis will be employed.



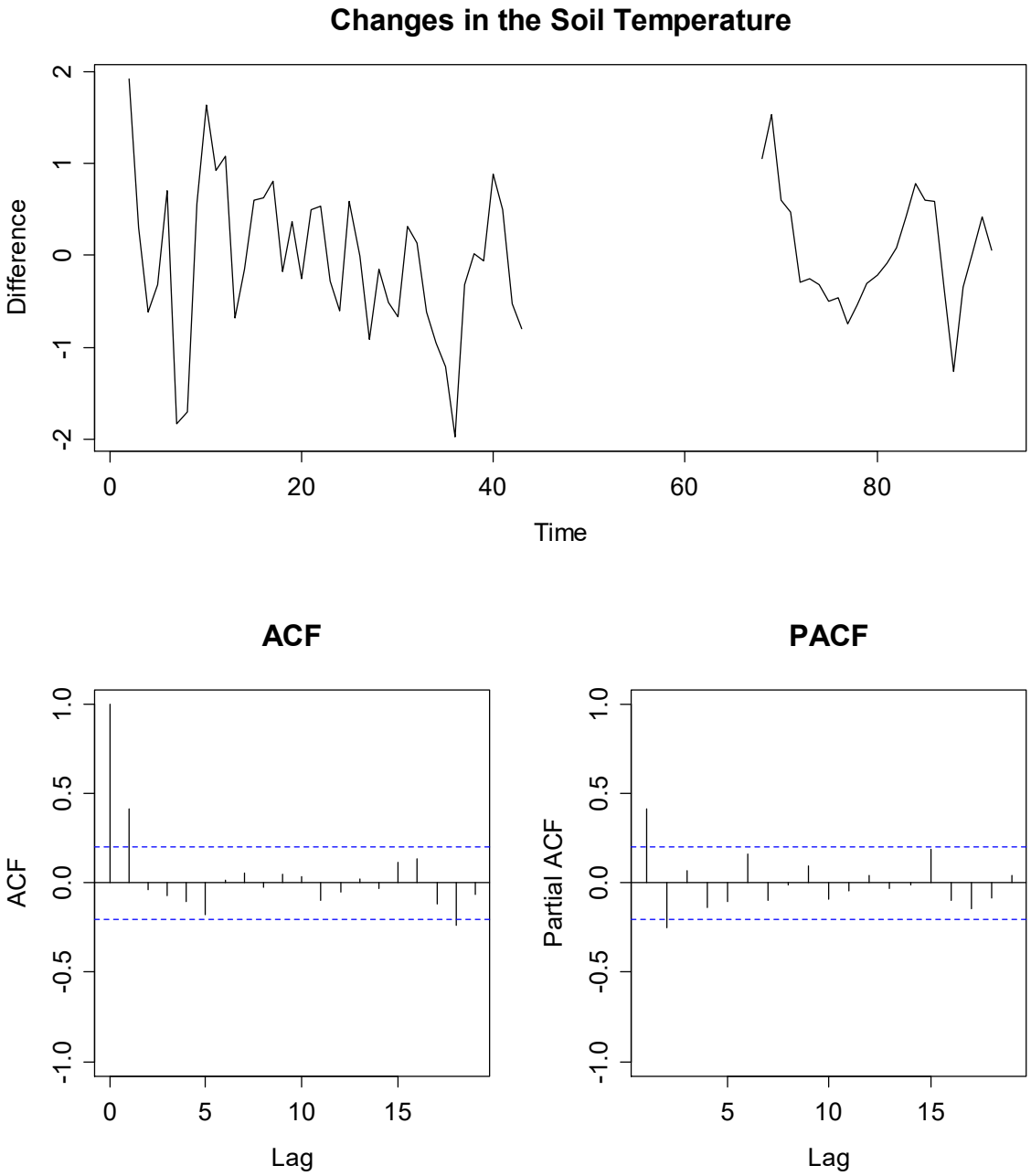
As we had stated above, multivariate time series analysis requires stationarity. Is this met with our series? The time series plot does not give a very clear answer. Science tells us that temperature has a seasonal pattern. Moreover, the correlogram of the two series is enlightening.



The ACF exhibits a slow decay, especially for the soil temperature. Thus, we decide to perform lag 1 differencing before analyzing the series. This has another advantage: we are then exploring how changes in the air temperature are associated with changes in the soil temperature and if so, what the time delay is. These results are easier to interpret than a direct analysis of air and soil temperatures. Next, we display the differenced series with their ACF and PACF. The observations during the snow cover period are now omitted.



The differenced air temperature series seems stationary, but is clearly not iid. There seems to be some strong negative correlation at lag 4. This may indicate the properties of the meteorological weather patterns at that time of year in that part of Switzerland. We now perform the same analysis for the changes in the soil temperature.



In the course of our discussion of multivariate time series analysis, we will require some $ARMA(p,q)$ models fitted to the changes in air and soil temperature. For the former series, model choice is not simple, as in both ACF and PACF, the coefficient at lag 4 sticks out. A grid search shows that an $AR(5)$ model yields the best AIC value, and also, the residuals from this model do look as desired, i.e. it seems plausible that they are White Noise.

For the changes in the soil temperature, model identification is easier. ACF and PACF suggest either a $MA(1)$, an $ARMA(2,1)$ or an $AR(2)$. From these three models, the $MA(1)$ shows both the lowest AIC value as well as the “best looking” residuals. Furthermore, it is the parsimonious choice, and hence we got with it.

9.2 Cross Correlation

To begin with, we consider the (theoretical) cross covariance, the measure that describes the amount of linear dependence between the two time series processes. Firstly, we recall the definition of the within-series autocovariances, denoted by $\gamma_{11}(k)$ and $\gamma_{22}(k)$:

$$\gamma_{11}(k) = Cov(X_{1,t+k}, X_{1,t}),\ \gamma_{22}(k) = Cov(X_{2,t+k}, X_{2,t})$$

The cross covariances between the two processes X_1 and X_2 are given by:

$$\gamma_{12}(k) = Cov(X_{1,t+k}, X_{2,t}),\ \gamma_{21}(k) = Cov(X_{2,t+k}, X_{1,t})$$

Note that owing to the stationarity of the two series, the cross covariances $\gamma_{12}(k)$ and $\gamma_{21}(k)$ both do not depend on the time t. Moreover, there is some obvious symmetry in the cross covariance:

$$\gamma_{12}(-k) = Cov(X_{1,t-k}, X_{2,t}) = Cov(X_{1,t}, X_{2,t+k}) = \gamma_{21}(k)$$

Thus, for practical purposes, it suffices to consider $\gamma_{12}(k)$ for positive and negative values of k. Note that we will preferably work with correlations rather than covariances, because they are scale-free and thus easier to interpret. We can obtain the cross correlations by standardizing the cross covariances:

$$\rho_{12}(k) = \frac{\gamma_{12}(k)}{\sqrt{\gamma_{11}(0)\gamma_{22}(0)}},\ \rho_{21}(k) = \frac{\gamma_{21}(k)}{\sqrt{\gamma_{11}(0)\gamma_{22}(0)}}.$$

Not surprisingly, we also have symmetry here, i.e. $\rho_{12}(-k) = \rho_{21}(k)$. Additionally, the cross correlations are limited to the interval between -1 and +1, i.e. $|\rho_{12}(k)| \leq 1$. As for the interpretation, $\rho_{12}(k)$ measures the linear association between two values of X_1 and X_2, if the value of the first time series is k steps ahead. Concerning estimation of cross covariances and cross correlations, we apply the usual sample estimators:

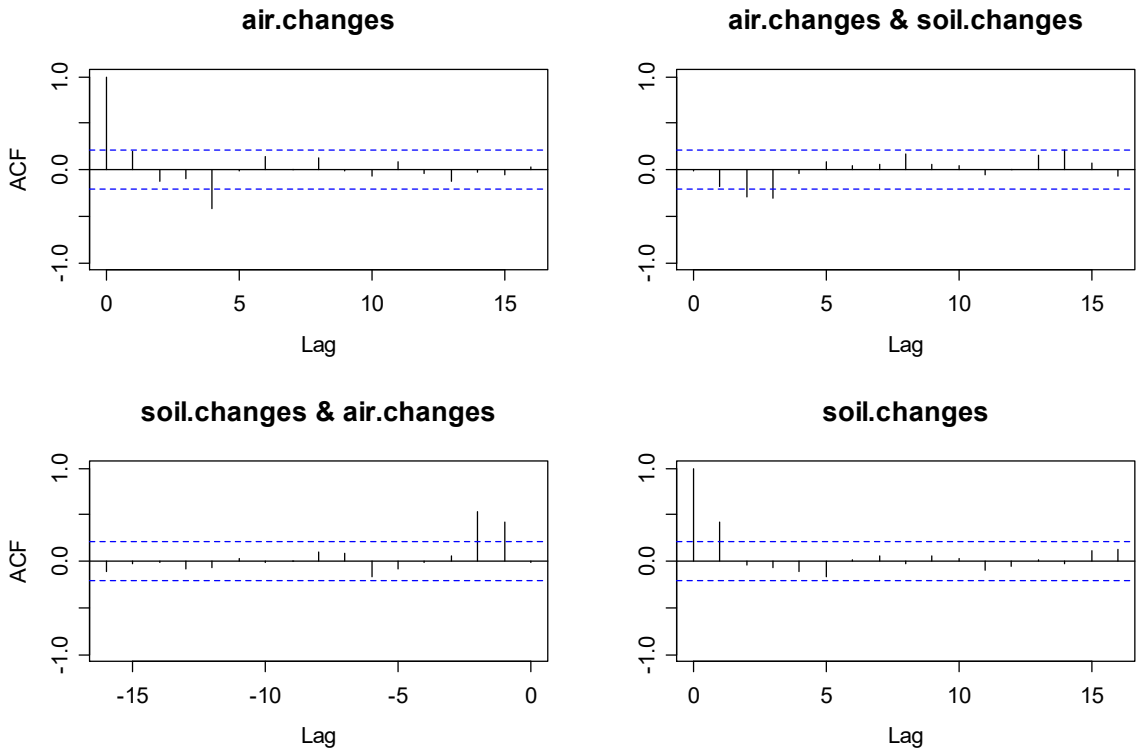
$$\hat{\gamma}_{12}(k) = \frac{1}{n}\sum_t (x_{1,t+k} - \bar{x}_1)(x_{2,t} - \bar{x}_2) \text{ and } \hat{\gamma}_{21}(k) = \frac{1}{n}\sum_t (x_{2,t+k} - \bar{x}_2)(x_{1,t} - \bar{x}_1),$$

where the summation index t for $k \geq 0$ goes from 1 to $n-k$ and for $k < 0$ goes from $1-k$ to n. With $\bar{x}_1$ and $\bar{x}_2$ we denote the mean values of $x_{1,t}$ and $x_{2,t}$, respectively. We define the estimation of the cross-correlations as

$$\hat{\rho}_{12}(k) = \frac{\hat{\gamma}_{12}(k)}{\sqrt{\hat{\gamma}_{11}(0)\hat{\gamma}_{22}(0)}},\ \hat{\rho}_{21}(k) = \frac{\hat{\gamma}_{21}(k)}{\sqrt{\hat{\gamma}_{11}(0)\hat{\gamma}_{22}(0)}}.$$

The plot of $\hat{\rho}_{12}(k)$ against k is called the cross-correlogram. Note that this must be viewed for both positive and negative k. In **R**, we the job is done by the acf() function, applied to a multiple time series object.

```
> both <- ts.union(diff(air.na), diff(soil.na))
> acf(both, na.action=na.pass, ylim=c(-1,1))
```



The top left panel shows the ACF of the differenced air temperature, the bottom right one holds the pure autocorrelations of the differenced soil temperature. The two off-diagonal plots contains estimates of the cross correlations: The top right panel has $\hat{\rho}_{12}(k)$ for positive values of k, and thus shows how changes in the air temperature depend on changes in the soil temperature.

Note that we do not expect any significant correlation coefficients here, because the ground temperature has hardly any influence on the future air temperature at all. Conversely, the bottom left panel shows $\hat{\rho}_{12}(k)$ for negative values of k, and thus how the changes in the soil temperature depend on changes in the air temperature. Here, we expect to see significant correlation.

9.2.1 Interpreting the Cross Correlogram

Interpreting the cross correlogram is tricky, because the within-series dependency results in a mixing of the correlations. It is very important to note that the confidence bounds shown in the above plots are usually wrong and can thus be strongly misleading. If not the additional steps to be discussed below are taken, interpreting the raw cross correlograms will lead to false conclusions.

The reason for these problems is that the variances and covariances of the $\hat{\rho}_{12}(k)$ are very complicated functions of $\rho_{11}(j), \rho_{22}(j)$ and $\rho_{12}(j), j \in \mathbb{Z}$. For illustrative purposes, we will treat some special cases explicitly.

Case 1: No correlation between the two series for large lags

In the case where the cross correlation $\rho_{12}(j) = 0$ for $|\, j \,| \geq m$, we have for $|\, k \,| \geq m$:

$$Var(\hat{\rho}_{12}(k)) \approx \frac{1}{n} \sum_{j=-\infty}^{\infty} \{\rho_{11}(j)\rho_{22}(j) + \rho_{12}(j+k)\rho_{12}(j-k)\}.$$

Thus, the variance of the estimated cross correlation coefficients goes to zero for $n \rightarrow \infty$, but for a deeper understanding with finite sample size, we must know *all* true auto and cross-correlations, which is of course impossible in practice.

Case 2: No correlation between the series for all lags

If the two processes X_1 and X_2 are independent, i.e. $\rho_{12}(j) \equiv 0$ for all j, then the variance of the cross correlation estimator simplifies to:

$$Var(\hat{\rho}_{12}(k)) \approx \frac{1}{n} \sum_{j=-\infty}^{\infty} \rho_{11}(j)\rho_{22}(j).$$

If, for example, X_1 and X_2 are two independent $AR(1)$ processes with parameters α_1 and α_2, then $\rho_{11}(j) = \alpha_1^{|j|}$, $\rho_{22}(j) = \alpha_2^{|j|}$ and $\rho_{12}(j) \equiv 0$. For the variance of $\hat{\rho}_{12}(k)$ we have, because the autocorrelations form a geometric series:

$$Var(\hat{\rho}_{12}(k)) \approx \frac{1}{n} \sum_{j=-\infty}^{\infty} (\alpha_1\alpha_2)^{|j|} = \frac{1}{n} \cdot \frac{1+\alpha_1\alpha_2}{1-\alpha_1\alpha_2}.$$

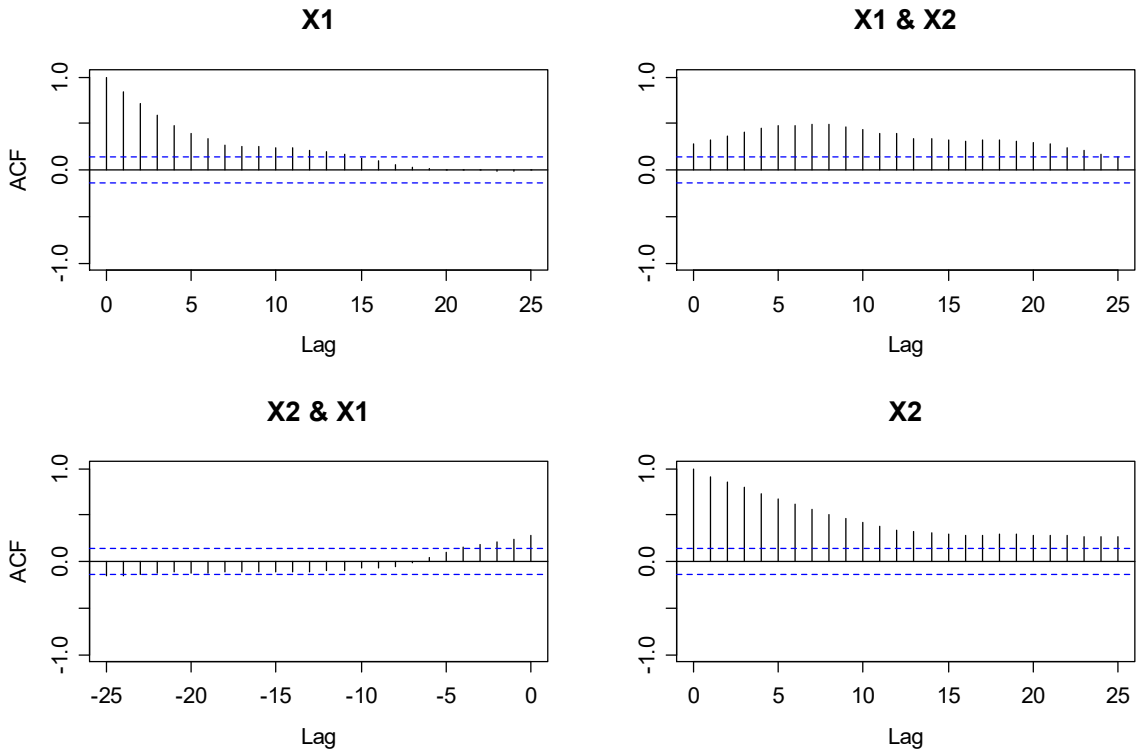
For $\alpha_1 \rightarrow 1$ and $\alpha_2 \rightarrow 1$ this expression goes to ∞, i.e. the estimator $\hat{\rho}_{12}(k)$ can, for a finite time series, differ greatly from the true value 0. We would like to illustrate this with two simulated $AR(1)$ processes with $\alpha_1 = \alpha_2 = 0.9$. According to theory all cross-correlations are 0. However, as we can see in the figure on the next page, the estimated cross correlations differ greatly from 0, even though the length of the estimated series is 200. In fact, $2\sqrt{Var(\hat{\rho}_{12}(k))} \approx 0.44$, i.e. the 95% confidence interval is ± 0.44. Thus even with an estimated cross-correlation of 0.4 the null hypothesis “true cross-correlation is equal to 0” cannot be rejected.

Case 3: No cross correlations for all lags and one series uncorrelated

Only now, in this special case, the variance of the cross correlation estimator is significantly simplified. In particular, if X_1 is a White Noise process which is independent of X_2, we have, for large n and small k:

$$Var(\hat{\rho}_{12}(k)) \approx \frac{1}{n}.$$

Thus, in this special case, the rule of thumb $\pm 2/\sqrt{n}$ yields a valid approximation to a 95% confidence interval for the cross correlations and can help to decide whether they are significantly or just randomly different from zero.



In most practical examples, however, the data will be auto- and also cross correlated. Thus, the question arises whether it is at all possible to do something here. Fortunately, the answer is yes: with the method of *prewhitening*, described in the next chapter, we do obtain a theoretically sound and practically useful cross correlation analysis.

9.3 Prewhitening

The idea behind *prewhitening* is to transform one of the two series such that it is uncorrelated, i.e. a White Noise series, which also explains the name of the approach. Formally, we assume that the two stationary processes X_1 and X_2 can be transformed as follows:

$$U_t = \sum_{i=0}^{\infty} a_i X_{1,t-i}$$

$$V_t = \sum_{i=0}^{\infty} b_i X_{2,t-i}$$

Thus, we are after coefficients a_i and b_i such that an infinite linear combination of past terms leads to White Noise. We know from previous theory that such a representation exists for all stationary and invertible $ARMA(p,q)$ processes, it is the $AR(\infty)$ representation. For the cross-correlations between U_t and V_t and between X_t and Y_t, the following relation holds:

$$\rho_{UV}(k) = \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} a_i b_j \rho_{X_1X_2}(k+i-j)$$

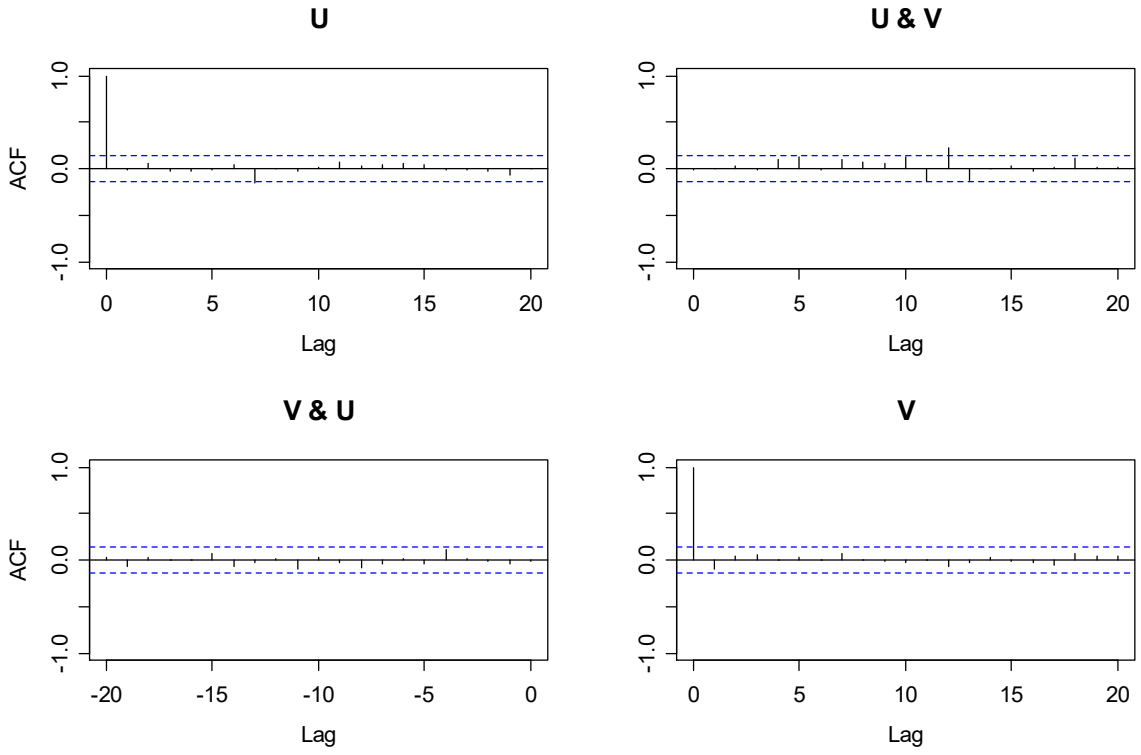
We conjecture that for two independent processes X_1 and X_2, where all cross correlation coefficients $\rho_{X_1X_2}(k) = 0$, also all $\rho_{UV}(k) = 0$. Additionally, the converse is also true, i.e. it follows from "U_t and V_t uncorrelated" that the original processes X_1 and X_2 are uncorrelated, too. Since U_t and V_t are White Noise processes, we are in the above explained case 3, and thus the confidence bounds in the cross correlograms are valid. Hence, any cross correlation analysis on "real" time series starts with representing them in terms of u_t and v_t.

Example: $AR(1)$ **Simulations**

For our example with the two simulated $AR(1)$ processes, we can estimate the AR model coefficients with the Burg method and plug them in for prewhitening the series. Note that this amounts considering the residuals from the two fitted models!

$u_t = x_{1,t} - \hat{\alpha}_1 x_{1,t-1}$, where $\hat{\alpha}_1 = 0.889$, and

$v_t = x_{2,t} - \hat{\alpha}_2 x_{2,t-1}$, where $\hat{\alpha}_2 = 0.917$.



The figure on the previous page shows both the auto and cross correlations of the prewhitened series. We emphasize again that we here consider the residuals from the $AR(1)$ models that were fitted to series X_1 and X_2. We observe that, as we expect, there are no significant autocorrelations, and there is just one cross

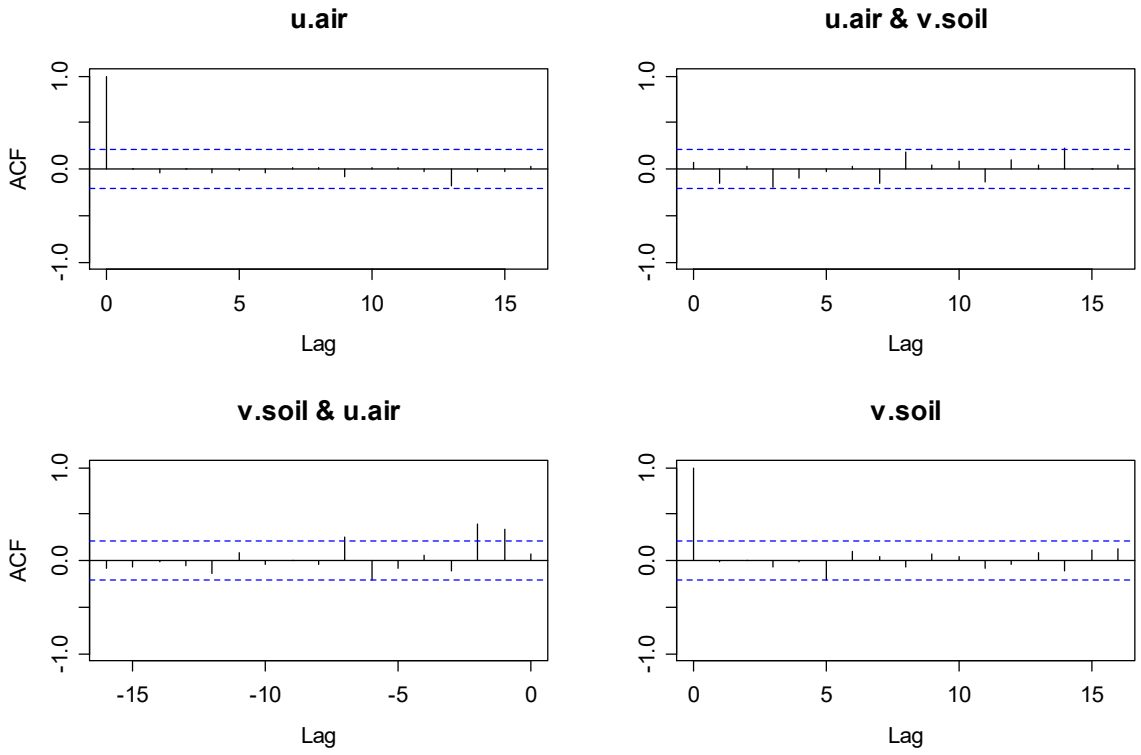
correlation coefficient that exceeds the 95% confidence bounds. We can attribute this to random variation.

The theory suggests, because U_t and V_t are uncorrelated, that also X_1 and X_2 do not show any linear dependence. Well, owing to how we set up the simulation, we know this for a fact, and take the result as evidence that the prewhitening approach works in practice.

Example: Air and Soil Temperatures

For verifying whether there is any cross correlation between the changes in air and soil temperatures, we have to perform prewhitening also for the two differenced series. Previously, we had identified an $AR(5)$ and a $MA(1)$ model as. We can now just take their residuals and perform a cross correlation analysis:

```
> fit.air  <- arima(diff(air.na), order=c(5,0,0))
> fit.soil <- arima(diff(soil.na), order=c(0,0,1))
> u.air    <- resid(fit.air); v.soil   <- resid(fit.soil)
> acf(ts.union(u.air, v.soil), na.action=na.pass)
```



The bottom left panel shows some significant cross correlations. A change in the air temperature seems to induce a change in the ground with a lag of 1 or 2 days.

9.4 Transfer Function Models

In the previous section we had observed significant cross correlations between the prewhitened air and soil temperature changes. This means that the cross

correlations between the original air and soil temperature changes will also be different from zero. However, due to the prewhitening, inferring the magnitude of the linear association is different. The aim of this section is to clarify this issue.

The transfer function models are a possible way to capture the dependency between two time series. We must assume that the first series influences the second, but the second does not influence the first. Furthermore, the influence occurs only at simultaneously or in the future, but not on past values. Both assumptions are met in our example. The transfer function model is:

$$X_{2,t} - \mu_2 = \sum_{j=0}^{\infty} \nu_j (X_{1,t-j} - \mu_1) + E_t$$

We call X_1 the *input* and correspondingly, X_2 is named the *output*. For the error term E_t we require zero expectation and that they are independent from the input series, in particular:

$E[E_t] = 0$ and $Cov(E_t, X_{1,s}) = 0$ for all t and s.

However, the errors E_t are usually autocorrelated. Note that this model is very similar to the time series regression model. However, here we have infinitely many unknown coefficients ν_j, i.e. we do not know (a priori) on which lags to regress the input for obtaining the output. For the following theory, we assume (w.l.o.g.) that $\mu_1 = \mu_2 = 0$, i.e. the two series were adjusted for their means. In this case the cross covariances $\gamma_{21}(k)$ are given by:

$$\gamma_{21}(k) = Cov(X_{2,t+k}, X_{1,t}) = Cov(\sum_{j=0}^{\infty} \nu_j X_{1,t+k-j}, X_{1,t}) = \sum_{j=0}^{\infty} \nu_j \gamma_{11}(k-j).$$

In cases where the transfer function model has a finite number of coefficients ν_j only, i.e. $\nu_j = 0$ for $j > K$, then the above formula turns into a linear system of $K+1$ equations that we could theoretically solve for the unknowns $\nu_j, j = 0,\ldots,K$.

If we replaced the theoretical γ_{11} and γ_{21} by the empirical covariances $\hat{\gamma}_{11}$ and $\hat{\gamma}_{21}$, this would yield, estimates $\hat{\nu}_j$. However, this method is statistically inefficient and the choice of K proves to be difficult in practice. We again resort to some special case, for which the relation between cross covariance and transfer function model coefficients simplifies drastically.

Special Case: Uncorrelated input series X_1

In this case, $\gamma_{11}(k) = 0$ for $k \neq 0$ and we have $\gamma_{21}(k) = \nu_k \gamma_{11}(0)$. For the coefficients ν_k this results in the simplified transfer function model:

$$\nu_k = \frac{\gamma_{21}(k)}{\gamma_{11}(0)} = \rho_{21}\sqrt{\frac{\gamma_{22}(0)}{\gamma_{11}(0)}}, \text{ for } k \geq 0.$$

However, X_1 generally is not a White Noise process. We can resort to prewhitening the input series. As we will show below, we can obtain an equivalent transfer function model with identical coefficients if a smart transformation is applied to the output series. Namely, we have to filter the output with the model coefficients from the input series.

$$X_{1,t} = 0.296 \cdot X_{1,t-1} - 0.242 \cdot X_{1,t-2} - 0.119 \cdot X_{1,t-3} - 0.497 \cdot X_{1,t-4} + 0.216 \cdot X_{1,t-5} + D_t ,$$

where D_t is the innovation, i.e. a White Noise process, for which we estimate the variance to be $\hat{\sigma}_D^2 = 2.392$. We now solve this equation for D_t and get:

$$\begin{aligned} D_t &= X_{1,t} - 0.296 \cdot X_{1,t-1} + 0.242 \cdot X_{1,t-2} + 0.119 \cdot X_{1,t-3} + 0.497 \cdot X_{1,t-4} - 0.216 \cdot X_{1,t-5} \\ &= (1 - 0.296B + 0.242B^2 + 0.119B^3 + 0.497B^4 - 0.216B^5) X_{1,t} \end{aligned}$$

We now apply this same transformation, i.e. the characteristic polynomial of the AR(5) also on the output series X_2 and the transfer function model errors E_t:

$$Z_t = (1 - 0.296B + 0.242B^2 + 0.119B^3 + 0.497B^4 - 0.216B^5) X_{2,t}$$

$$U_t = (1 - 0.296B + 0.242B^2 + 0.119B^3 + 0.497B^4 - 0.216B^5) E_t .$$

We can now equivalently write the transfer function model with the new processes D_t, Z_t and U_t. It takes the form:

$$Z_t = \sum_{j=0}^{\infty} \nu_j D_{t-j} + U_t ,$$

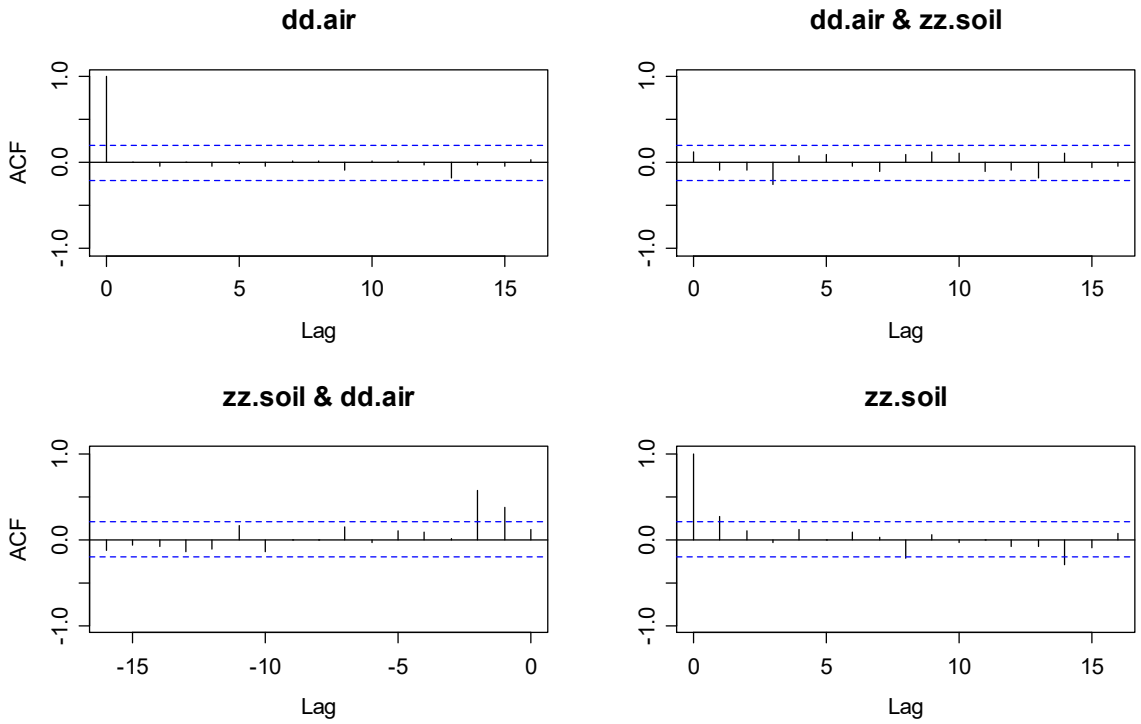
where the coefficients ν_j are identical than for the previous formulation of the model. The advantage of this latest formulation, however, is that the input series D_t is now White Noise, such that the above special case applies, and the transfer function model coefficients can be obtained by a straightforward computation from the cross correlations:

$$\hat{\nu}_k = \frac{\hat{\gamma}_{21}(k)}{\hat{\sigma}_D^2} = \frac{\hat{\sigma}_Z}{\hat{\sigma}_D} \hat{\rho}_{21}(k) , \text{ where } k \geq 0 .$$

where $\hat{\gamma}_{21}$ and $\hat{\rho}_{21}$ denote the empirical cross covariances and cross correlations of D_t and Z_t. However, keep in mind that Z_t and U_t are generally correlated. Thus, the outlined method is not a statistically efficient estimator either. While efficient approaches exist, we will not discuss them in this course and scriptum. Furthermore, for practical application the outlined procedure usually yields reliable results. We conclude this section by showing the results for the permafrost example: the transfer function model coefficients in the example are based on the cross correlation between the $AR(5)$ residuals of the air changes and the ground changes that had been filtered with these $AR(5)$ coefficients.

```
> dd.air  <- resid(fit.air)
```

```
> coefs   <- coef(fit.air)[1:5])
> zz.soil <- filter(diff(soil.na), c(1, -coefs, sides=1)
> as.int  <- ts.intersect(dd.air, zz.soil)
> acf.val <- acf(as.int, na.action=na.pass)
```



Again, in all except for the bottom left panel, the correlation coefficients are mostly zero, respectively only insignificantly or by chance different from that value. This is different in the bottom left panel. Here, we have substantial cross correlation at lags 1 and 2. Also, these values are proportional to the transfer function model coefficients. We can extract these as follows:

```
> multip  <- sd(zz.soil, na.rm=TRUE)/sd(dd.air, na.rm=TRUE)
> multip*acf.val$acf[,2,1]
 [1]  0.054305137  0.165729551  0.250648114  0.008416697
 [5]  0.036091971  0.042582917 -0.014780751  0.065008411
 [9] -0.002900099 -0.001487220 -0.062670672  0.073479065
[13] -0.049352348 -0.060899602 -0.032943583 -0.025975790
```

Thus, the soil temperature in the permafrost boreholes reacts to air temperature changes with a delay of 1-2 days. An analysis of further boreholes has suggested that the delay depends on the type of terrain in which the measurements were made. Fastest response times are found for a very coarse-blocky rock glacier site, whereas slower response times are revealed for blocky scree slopes with smaller grain sizes.